

Classification of Electrocardiograms using Contrastive Predictive Coding

Julian Winter

Masterarbeit

Date of issue: 18. August 2021
Date of submission: 18. Februar 2022
Reviewers: Prof. Dr. Stefan Harmeling
Prof. Dr. Michael Leuschel

Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 18. Februar 2022

Julian Winter

Abstract

The following work deals with classifying multilabel electro-cardiographic data using a "self-supervised" learning method called Contrastive Predictive Coding (short CPC) [1], that learns lower dimensional data representations without requiring class labels. These data representations can then be used in classification tasks. Learning from data without labels is potentially very beneficial in fields where correctly labeled data is scarce and only obtainable slowly by human experts.

The goal of this work is to re-implement CPC, apply and test several changes to the architecture and evaluate its capability of classifying ECG-data. We try to answer the following central questions to shine a light on CPC's strengths and weaknesses:

How does the CPC architecture compare against our baseline models in a fully supervised setting? How capable is CPC's pretraining? How good are the learned representations on their own? Under what circumstances is CPC especially efficient to use? Can the original architecture be improved? Does CPC learn visually verifiable useful latent representations?

We will answer these questions by conducting various experiments that aim to emulate real-life circumstances such as low label availability and limited training times. CPC is compared to fully supervised models, where we make use of architectures like the TCN [2] and a special time-series classification focused residual net [3], but also introduce a wide variety of own baseline models.

For evaluation we look at predictions both quantitatively, through different metrics, and qualitatively, by either inspecting embeddings in lower dimensions of the learned representations or by visualizing the prediction probabilities in a wide variety of plots. Additionally we utilize the gradient of selected trained networks to show spots in the input data which possibly contain ECG-classes.

Our contributions to the field are a more precise explanation of the original publication [1], alterations to the one dimensional architecture suited for timeseries classification, namely non overlapping data windows, normalized latent representations, different encoder-, autoregressive context- and downstream-task networks. We also introduce a "no-context" architecture which predict latents without making use of the standard context recurrent network. In total we trained and tested over 1700¹ networks and will also release their model properties and average scores as table for all those interested.

Last but not least our generated prediction plots and "point of interest"-visualization mainly based on grad-cam [4], show the different models' strengths beyond classification scores and could also assist cardiologists in the future.

¹We count individually trained/tested networks, not necessarily different architectures

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 3 |
| 1.1 | Electrocardiographic Data | 3 |
| 1.2 | Representation Learning/Background | 8 |
| 1.3 | Contrastive Predictive Coding | 9 |
| 1.3.1 | Overview | 9 |
| 1.3.2 | Motivation | 10 |
| 1.3.3 | Math in detail | 11 |
| 1.3.4 | Algorithm | 14 |
| 2 | Implementation | 19 |
| 2.1 | CPC | 19 |
| 2.1.1 | Unsupervised | 19 |
| 2.1.2 | Supervised | 20 |
| 2.2 | CPC with modifications | 20 |
| 2.2.1 | Loss calculation changes | 20 |
| 2.2.1.1 | Latent sampling | 20 |
| 2.2.1.2 | Fewer latent vectors | 21 |
| 2.2.1.3 | Normalized latent vectors | 21 |
| 2.2.2 | Encoder networks | 22 |
| 2.2.2.1 | Data sliding window | 22 |
| 2.2.2.2 | Baseline-like Network | 22 |
| 2.2.3 | Context prediction networks | 23 |
| 2.2.4 | Latent prediction networks | 23 |
| 2.2.5 | Downstream task networks | 23 |
| 2.3 | Baselines | 24 |
| 2.3.1 | Custom baseline architectures | 25 |
| 2.3.2 | Literature architectures | 27 |
| 3 | Training | 29 |
| 3.1 | Data preprocessing | 29 |
| 3.2 | Train settings | 30 |
| 3.2.1 | CPC | 31 |

| | |
|--|-----------|
| 4 Evaluation | 33 |
| 4.1 CPC Evaluation during Pretraining | 33 |
| 4.2 Model Evaluation during Downstream Training | 33 |
| 4.3 Evaluation after training | 34 |
| 4.3.1 Quantitative | 34 |
| 4.3.2 Qualitative | 36 |
| 5 Results | 37 |
| 5.1 Quantitative | 37 |
| 5.1.1 Baselines compared to CPC | 37 |
| 5.1.2 CPC Downstream task with frozen weights | 38 |
| 5.1.3 CPC Downstream task with all weights updated | 40 |
| 5.1.4 Low label availability | 40 |
| 5.1.5 Experiment: Training on Augmented Data | 44 |
| 5.1.6 Additional changes | 48 |
| 5.1.6.1 Normalized Latent Vectors | 49 |
| 5.1.6.2 Encoder like Baseline_v8 | 49 |
| 5.1.6.3 Hidden State Context Network | 49 |
| 5.1.6.4 No Context Network | 50 |
| 5.2 Qualitative | 50 |
| 5.2.1 Finding the best Hyperparameters with Parallel Coordinates . . . | 50 |
| 5.2.2 Latent/Context t-SNE representations | 53 |
| 5.2.3 Transforming Predictions Scores with Model Thresholds | 60 |
| 5.2.4 Hand-picked Samples: Prediction Score Scatterplots | 61 |
| 5.2.5 All Samples: Violinplots | 62 |
| 5.2.6 Explaining predicted classes in input data | 66 |
| 6 Discussion and Future work | 73 |
| A Appendix | 75 |
| B Code | 76 |
| B.1 architectures_baseline_challenge | 76 |
| B.1.1 baseline_FCN.py | 76 |
| B.1.2 baseline_MLP.py | 76 |

| | | |
|--------|----------------------------------|-----|
| B.1.3 | baseline_TCN_block.py | 77 |
| B.1.4 | baseline_TCN_down.py | 77 |
| B.1.5 | baseline_TCN_flatten.py | 78 |
| B.1.6 | baseline_TCN_last.py | 79 |
| B.1.7 | baseline_alex.py | 80 |
| B.1.8 | baseline_alex_v2.py | 80 |
| B.1.9 | baseline_cnn_v0.py | 81 |
| B.1.10 | baseline_cnn_v0_1.py | 82 |
| B.1.11 | baseline_cnn_v0_2.py | 82 |
| B.1.12 | baseline_cnn_v0_3.py | 83 |
| B.1.13 | baseline_cnn_v1.py | 84 |
| B.1.14 | baseline_cnn_v14.py | 85 |
| B.1.15 | baseline_cnn_v15.py | 86 |
| B.1.16 | baseline_cnn_v2.py | 87 |
| B.1.17 | baseline_cnn_v3.py | 88 |
| B.1.18 | baseline_cnn_v4.py | 89 |
| B.1.19 | baseline_cnn_v5.py | 89 |
| B.1.20 | baseline_cnn_v6.py | 90 |
| B.1.21 | baseline_cnn_v7.py | 91 |
| B.1.22 | baseline_cnn_v8.py | 92 |
| B.1.23 | baseline_cnn_v9.py | 93 |
| B.1.24 | baseline_convencoder.py | 94 |
| B.1.25 | baseline_resnet.py | 95 |
| B.1.26 | baseline_rnn.py | 96 |
| B.1.27 | baseline_rnn_simplest_gru.py | 96 |
| B.1.28 | baseline_rnn_simplest_lstm.py | 97 |
| B.2 | architectures_cpc | 97 |
| B.2.1 | cpc_autoregressive_hidden.py | 97 |
| B.2.2 | cpc_autoregressive_v0.py | 97 |
| B.2.3 | cpc_base.py | 98 |
| B.2.4 | cpc_combined.py | 100 |
| B.2.5 | cpc_downstream_cnn.py | 100 |
| B.2.6 | cpc_downstream_latent_average.py | 101 |

| | | |
|---------|--|-----|
| B.2.7 | cpc_downstream_latent_maximum.py | 101 |
| B.2.8 | cpc_downstream_model_multitarget_v1.py | 102 |
| B.2.9 | cpc_downstream_model_multitarget_v2.py | 103 |
| B.2.10 | cpc_downstream_only.py | 104 |
| B.2.11 | cpc_downstream_twolinear.py | 104 |
| B.2.12 | cpc_downstream_twolinear_v2.py | 105 |
| B.2.13 | cpc_encoder_as_strided.py | 106 |
| B.2.14 | cpc_encoder_decoder_v2.py | 106 |
| B.2.15 | cpc_encoder_likev8.py | 107 |
| B.2.16 | cpc_encoder_small.py | 107 |
| B.2.17 | cpc_encoder_v0.py | 108 |
| B.2.18 | cpc_encoder_v1.py | 108 |
| B.2.19 | cpc_encoder_v2.py | 109 |
| B.2.20 | cpc_encoder_v3.py | 109 |
| B.2.21 | cpc_encoder_v4.py | 110 |
| B.2.22 | cpc_encoder_vresnet.py | 110 |
| B.2.23 | cpc_intersect.py | 111 |
| B.2.24 | cpc_intersect_manylatents.py | 112 |
| B.2.25 | cpc_predictor_nocontext.py | 115 |
| B.2.26 | cpc_predictor_stacked.py | 115 |
| B.2.27 | cpc_predictor_v0.py | 116 |
| B.2.28 | cpc_with_decoder.py | 116 |
| B.3 | architectures_various | 117 |
| B.3.1 | explain_network.py | 117 |
| B.3.2 | explain_network2.py | 118 |
| B.4 | deprecated | 120 |
| B.4.1 | architectures_baseline | 120 |
| B.4.1.1 | baseline_cnn_v0.py | 120 |
| B.4.1.2 | baseline_cnn_v0_1.py | 121 |
| B.4.1.3 | baseline_cnn_v0_2.py | 121 |
| B.4.1.4 | baseline_cnn_v0_3.py | 122 |
| B.4.1.5 | baseline_cnn_v1.py | 123 |
| B.4.1.6 | baseline_cnn_v10.py | 124 |

| | | |
|----------|---------------------------------|-----|
| B.4.1.7 | baseline_cnn_v11.py | 125 |
| B.4.1.8 | baseline_cnn_v12.py | 126 |
| B.4.1.9 | baseline_cnn_v13.py | 126 |
| B.4.1.10 | baseline_cnn_v14.py | 127 |
| B.4.1.11 | baseline_cnn_v2.py | 128 |
| B.4.1.12 | baseline_cnn_v3.py | 129 |
| B.4.1.13 | baseline_cnn_v4.py | 130 |
| B.4.1.14 | baseline_cnn_v5.py | 131 |
| B.4.1.15 | baseline_cnn_v6.py | 131 |
| B.4.1.16 | baseline_cnn_v7.py | 132 |
| B.4.1.17 | baseline_cnn_v8.py | 133 |
| B.4.1.18 | baseline_cnn_v9.py | 134 |
| B.4.1.19 | baseline_convencoder.py | 135 |
| B.4.1.20 | baseline_losses.py | 136 |
| B.4.2 | cardio_model_v4.py | 136 |
| B.4.3 | cardio_model_v5.py | 138 |
| B.4.4 | cardio_model_v6.py | 140 |
| B.4.5 | cpc_alpha.py | 141 |
| B.4.6 | cpc_decoder.py | 144 |
| B.4.7 | cpc_utils.py | 144 |
| B.4.8 | data_storage.py | 146 |
| B.4.9 | downstream_model.py | 146 |
| B.4.10 | downstream_model_multitarget.py | 148 |
| B.4.11 | ecg_datasets.py | 149 |
| B.4.12 | main.py | 151 |
| B.4.13 | make_clean_data.py | 158 |
| B.5 | experiments | 158 |
| B.5.1 | create_fewer_labels_data.py | 158 |
| B.5.2 | create_timeseries_plots.py | 161 |
| B.5.3 | main_get_latents.py | 162 |
| B.6 | external | 165 |
| B.6.1 | tcn | 165 |
| B.6.1.1 | TCN | 165 |

| | | |
|-----------|--------------------------------------|-----|
| B.6.1.1.1 | tcn.py | 165 |
| B.6.2 | helper_code.py | 166 |
| B.6.3 | multi_scale_ori.py | 168 |
| B.7 | jupyter_notebooks | 171 |
| B.7.1 | Beat detection.py | 171 |
| B.7.2 | CPC Loss Implementation Test.py | 175 |
| B.7.3 | CPC Loss.py | 185 |
| B.7.4 | Challenge Data Visualization.py | 194 |
| B.7.5 | Export Code.py | 208 |
| B.7.6 | Filter Attributes DataFrame.py | 209 |
| B.7.7 | InfoNCE Loss Psuedo.py | 222 |
| B.7.8 | MIT data.py | 224 |
| B.7.9 | Model Gradient-Prediction Scatter.py | 230 |
| B.7.10 | PTB PCA.py | 243 |
| B.7.11 | PTBxl data.py | 245 |
| B.7.12 | Parallel Coordinates.py | 252 |
| B.7.13 | Physionet Hz Converter.py | 255 |
| B.7.14 | Precision Recall.py | 258 |
| B.7.15 | ROC.py | 264 |
| B.7.16 | Sinus Generator.py | 272 |
| B.7.17 | TSNE.py | 273 |
| B.8 | logs | 278 |
| B.9 | models_evaluated_filtered_csv | 278 |
| B.9.1 | old | 278 |
| B.10 | util | 278 |
| B.10.1 | data | 278 |
| B.10.1.1 | clear_h5_files.py | 278 |
| B.10.1.2 | dataframe_factory.py | 278 |
| B.10.1.3 | dataset_container.py | 279 |
| B.10.1.4 | ecg_data.py | 280 |
| B.10.1.5 | ecg_datasets2.py | 282 |
| B.10.1.6 | ecg_datasets3.py | 292 |
| B.10.1.7 | ptbxl_data.py | 296 |

| | |
|--|------------|
| B.10.2 metrics | 298 |
| B.10.2.1 baseline_losses.py | 298 |
| B.10.2.2 metrics.py | 299 |
| B.10.2.3 training_metrics.py | 302 |
| B.10.3 utility | 303 |
| B.10.3.1 dict_utils.py | 303 |
| B.10.3.2 extract_trained_model_attributes.py | 304 |
| B.10.3.3 full_class_name.py | 309 |
| B.10.3.4 layer_calculations.py | 309 |
| B.10.3.5 print_layer.py | 312 |
| B.10.3.6 sparse_print.py | 312 |
| B.10.3.7 timestamp.py | 312 |
| B.10.4 visualize | 312 |
| B.10.4.1 layer_visualization.py | 312 |
| B.10.4.2 plot_metrics.py | 313 |
| B.10.4.3 timeseries_to_image_converter.py | 316 |
| B.10.5 store_models.py | 320 |
| B.11 main_cpc_benchmark_test.py | 322 |
| B.12 main_cpc_benchmark_train.py | 326 |
| B.13 main_cpc_explain.py | 338 |
| B.14 main_cpc_explain_gradcam.py | 343 |
| B.15 main_produce_plots_for_tested_models.py | 347 |
| References | 357 |
| List of Figures | 361 |
| List of Tables | 363 |

1 Introduction

1.1 Electrocardiographic Data

Electrocardiographic data is obtained by placing electrodes on different parts of the body, mainly around the heart, which measure the electrical changes in voltage over time, produced by the heart muscle [5]. Multiple electrodes are placed to get a better overview over the heart state, since other muscles will add noise under non ideal conditions. A high recording frequency of 257-10000hz and a recording length of 10 seconds up to 30 minutes² or even days (long time ECG used in hospitals) are desirable to not miss important information that could reveal an infarction and other muscular heart diseases.

We obtain sequential data that can be classified like other time-series, however since each individual electrode produces its own signal there is more than one data channel to analyze concurrently, which increases the difficulty to find a generalizing computer system. For example the conventional "12-lead ECG" uses 10 differently placed electrodes resulting in 12 different channels together with the two additional ones added by averaging [6]. Accounting for the high frequency and length this would sum up to $\text{datapoints} = \text{seconds} \cdot \text{frequency} \cdot \text{channels} = 10\text{s} \cdot 1000\text{hz} \cdot 12 = 120000$ data points for short recordings or $120\text{s} \cdot 1000\text{hz} \cdot 12 = 1440000$ data points for longer recordings, per patient (assuming a frequency of 1000hz). The varying lengths and much information make correct prediction challenging. For comparison, the widely used ImageNet [7] counts on average 469×387 pixels[8], which is equal to $469 \cdot 387 \cdot 3 = 544509$ data points. Additionally the recordings can contain more than one "true" label which makes correct classification even more difficult. Like in many medical fields, one big hurdle in training a fully automated system is the inaccessible correctly labeled data. The biggest reason for the data scarcity is that only trained professionals/doctors are able to successfully diagnose patients, while also requiring long periods of time to do so.

Nevertheless we found a few dataset sources that are openly available online:

PTB A dataset provided by the Physikalisch-Technische Bundesanstalt (PTB) containing 549 records with 9 diagnostic class labels. [9]

PTB-XL A dataset provided by the Physikalisch-Technische Bundesanstalt (PTB) containing 21837 records with 52 diagnostic class labels (5 superclasses, split into 52 unique scp codes).[10]

Georgia "Georgia" dataset which is hosted on [Kaggle.com](#) [11]

China A dataset provided during a chinese classification challenge and made openly available.

Nature A 12-lead electrocardiogram database for arrhythmia research covering more than 10,000 patients. [12]

Most of the above datasets and more are also used in a recent ECG classification challenge from 2020 [13] hosted by physionet.org [14]. Since the original datasets have incompatible

²attributes of ECG data reported from 1.1

labels (e.g. the same classes are mapped to different terms) we use the available challenge data because the labels across all datasets were already mapped to unique SNOMED CT codes [15]. The “Nature” data labels, which are not part of the challenge data, had to be dropped since they were incompatible to the other datasets, thus this data will be only used in pretraining for models that require it.

The data has 12 channels and one or more “true” labels per sample. However there is no hint on to where the label is found in the sequential data. See [Figure 1](#) for an example ECG recording.

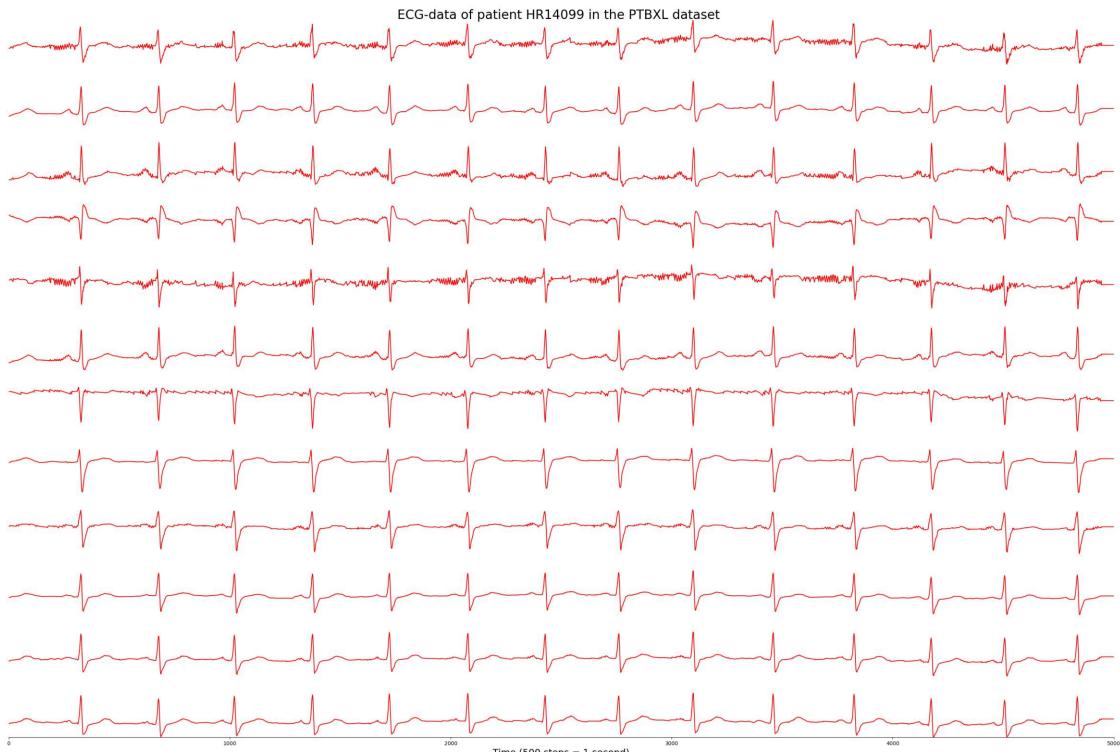


Figure 1: A single patients ECG with 500hz recording frequency. The classes *EKG: T wave abnormal* and *sinus rhythm* can both be found somewhere in the data.

To better understand the data we count class occurrences over all datasets, with their respective SNOMED Code and the index number used in our implementation [Figure 2](#). Readily identifiable is that not every class is present in all datasets and that some classes are very rare in comparison to others. Additionally some classes are closely related or even superclasses of others but are not consistently stated as true even though the sub-class is true (see Bundle Branch Block vs. left/right Bundle Branch Block or Ventricular Hypertrophy vs. left/right Ventricular Hypertrophy). Since some class distinctions might be intentional, we did not apply any changes to the labels, but want to point out the possible classification difficulty increase due to the label inconsistencies.

1.1 Electrocardiographic Data

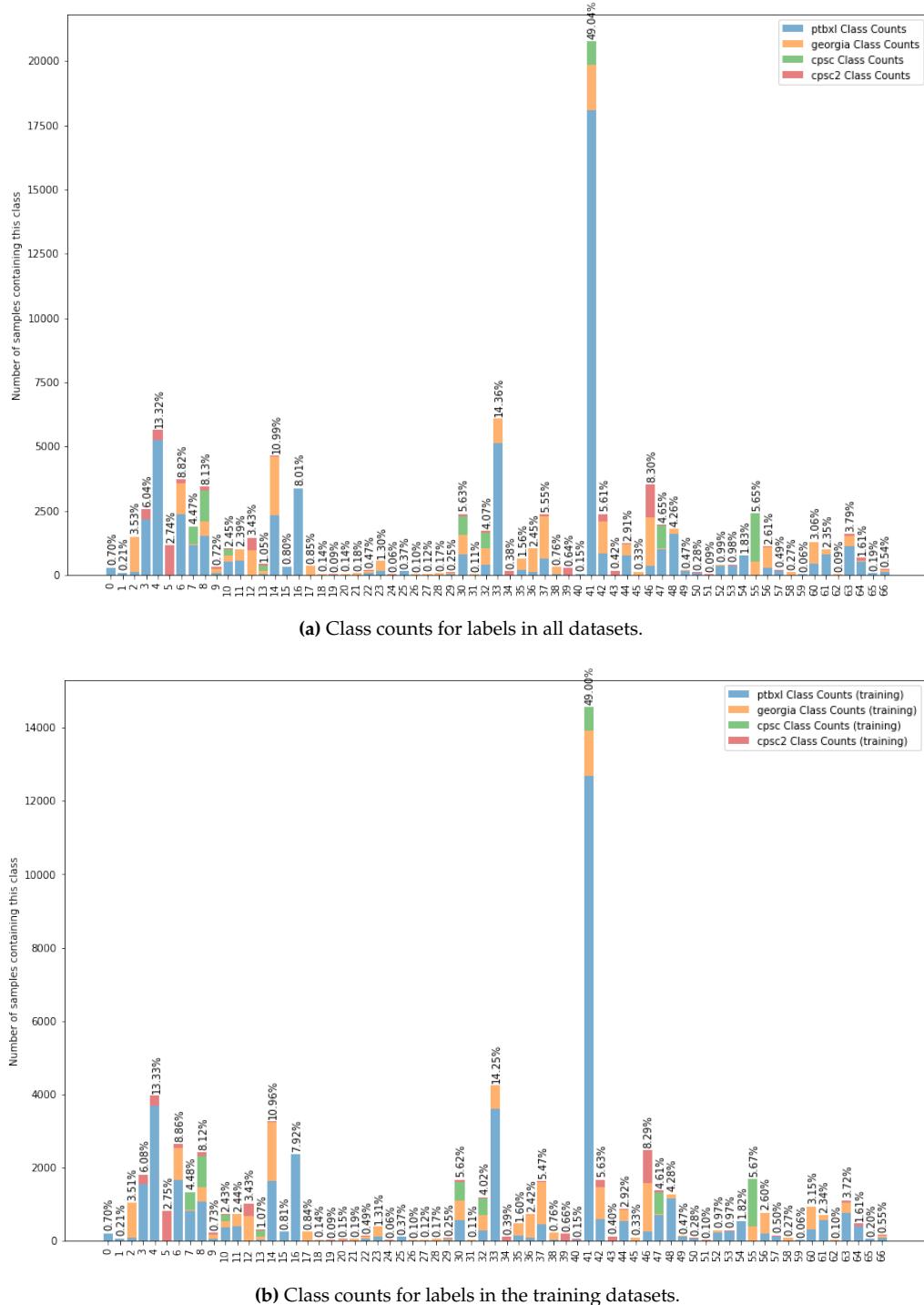
5

Figure 2: All SNOMED Codes with their respective name and count in the datasets.

| Index | Class Snomed Code | Class Term | ptbxl | georgia | cpsc | cpsc2 |
|-------|-------------------|--|-------|---------|------|-------|
| 0 | 10370003 | Rhythm from artificial pacing (298) | 295 | 0 | 0 | 3 |
| 1 | 11157007 | Ventricular bigeminy (89) | 82 | 2 | 0 | 5 |
| 2 | 111975006 | Prolonged QT interval (1493) | 118 | 1372 | 0 | 3 |
| 3 | 164861001 | EKG myocardial ischemia (2556) | 2175 | 0 | 0 | 381 |
| 4 | 164865005 | EKG: myocardial infarction (5640) | 5261 | 7 | 0 | 372 |
| 5 | 164867002 | EKG: old myocardial infarction (1160) | 0 | 0 | 0 | 1160 |
| 6 | 164873001 | EKG:left ventricle hypertrophy (3735) | 2359 | 1226 | 0 | 150 |
| 7 | 164884008 | ECG: ventricular ectopics (1894) | 1154 | 41 | 699 | 0 |
| 8 | 164889003 | ECG: atrial fibrillation (3441) | 1514 | 561 | 1219 | 147 |
| 9 | 164890007 | EKG: atrial flutter (303) | 73 | 185 | 0 | 45 |
| 10 | 164909002 | EKG: left bundle branch block (1038) | 536 | 229 | 235 | 38 |
| 11 | 164917005 | EKG: Q wave abnormal (1010) | 548 | 461 | 0 | 1 |
| 12 | 164930006 | ECG: ST interval abnormal (1453) | 0 | 979 | 0 | 474 |
| 13 | 164931005 | ST elevation (444) | 28 | 133 | 220 | 63 |
| 14 | 164934002 | EKG: T wave abnormal (4651) | 2345 | 2284 | 0 | 22 |
| 15 | 164947007 | Prolonged PR interval (340) | 340 | 0 | 0 | 0 |
| 16 | 164951009 | EKG: QRS complex abnormal (3389) | 3389 | 0 | 0 | 0 |
| 17 | 17338001 | Ventricular premature beats (359) | 0 | 351 | 0 | 8 |
| 18 | 195042002 | Second degree atrioventricular block (58) | 14 | 23 | 0 | 21 |
| 19 | 195080001 | Atrial fibrillation and flutter (39) | 0 | 2 | 0 | 37 |
| 20 | 195126007 | Atrial hypertrophy (61) | 0 | 59 | 0 | 2 |
| 21 | 233917008 | Atrioventricular block (77) | 0 | 74 | 0 | 3 |
| 22 | 251120003 | Incomplete left bundle branch block (201) | 77 | 83 | 0 | 41 |
| 23 | 251146004 | Low QRS voltages (552) | 182 | 370 | 0 | 0 |
| 24 | 251180001 | Ventricular trigeminy (24) | 20 | 1 | 0 | 3 |
| 25 | 251200008 | Indeterminate cardiac axis (156) | 156 | 0 | 0 | 0 |
| 26 | 251266004 | Ventricular pacing pattern (43) | 0 | 43 | 0 | 0 |
| 27 | 251268003 | Atrial pacing pattern (50) | 0 | 50 | 0 | 0 |
| 28 | 253352002 | Left atrial abnormality (70) | 0 | 70 | 0 | 0 |
| 29 | 266249003 | Ventricular hypertrophy (105) | 30 | 70 | 0 | 5 |
| 30 | 270492004 | First degree atrioventricular block (2385) | 797 | 764 | 721 | 103 |
| 31 | 27885002 | Complete atrioventricular block (46) | 16 | 8 | 0 | 22 |
| 32 | 284470004 | Premature atrial contraction (1722) | 398 | 635 | 616 | 73 |
| 33 | 39732003 | Left axis deviation (6080) | 5146 | 934 | 0 | 0 |
| 34 | 413844008 | Chronic myocardial ischemia (160) | 0 | 0 | 0 | 160 |
| 35 | 425419005 | EKG: inferior ischemia (660) | 219 | 441 | 0 | 0 |
| 36 | 425623009 | EKG: lateral ischemia (1039) | 142 | 897 | 0 | 0 |
| 37 | 426177001 | ECG: sinus bradycardia (2351) | 637 | 1670 | 0 | 44 |
| 38 | 426434006 | EKG: anterior ischemia (323) | 44 | 279 | 0 | 0 |
| 39 | 426627000 | ECG: bradycardia (273) | 0 | 6 | 0 | 267 |
| 40 | 426761007 | EKG: supraventricular tachycardia (62) | 27 | 32 | 0 | 3 |
| 41 | 426783006 | ECG: sinus rhythm (20760) | 18090 | 1751 | 916 | 3 |
| 42 | 427084000 | ECG: sinus tachycardia (2374) | 826 | 1247 | 0 | 301 |
| 43 | 427172004 | ECG: premature ventricular contractions (178) | 0 | 0 | 0 | 178 |
| 44 | 427393009 | ECG: sinus arrhythmia (1234) | 772 | 452 | 0 | 10 |
| 45 | 428417006 | Early repolarization (138) | 0 | 138 | 0 | 0 |
| 46 | 428750005 | Nonspecific ST-T abnormality on electrocardiogram (3513) | 381 | 1864 | 0 | 1268 |
| 47 | 429622005 | ST Depression (1967) | 1009 | 36 | 867 | 55 |
| 48 | 445118002 | Left anterior fascicular block on electrocardiogram (1805) | 1626 | 179 | 0 | 0 |
| 49 | 445211001 | Left posterior fascicular block on electrocardiogram (199) | 177 | 22 | 0 | 0 |
| 50 | 446358003 | Right atrial hypertrophy (117) | 99 | 0 | 0 | 18 |
| 51 | 446813000 | Left atrial hypertrophy (40) | 0 | 0 | 0 | 40 |
| 52 | 47665007 | Right axis deviation (421) | 343 | 77 | 0 | 1 |
| 53 | 54329005 | Acute myocardial infarction of anterior wall (414) | 354 | 0 | 0 | 60 |
| 54 | 55930002 | EKG ST segment changes (776) | 770 | 6 | 0 | 0 |
| 55 | 59118001 | Right bundle branch block (2390) | 0 | 534 | 1855 | 1 |
| 56 | 59931005 | Inverted T wave (1105) | 294 | 806 | 0 | 5 |
| 57 | 63593006 | Supraventricular premature beats (208) | 157 | 1 | 0 | 50 |
| 58 | 6374002 | Bundle branch block (115) | 0 | 115 | 0 | 0 |
| 59 | 67198005 | Paroxysmal supraventricular tachycardia (24) | 24 | 0 | 0 | 0 |
| 60 | 67741000119109 | Left atrial enlargement (1295) | 427 | 868 | 0 | 0 |
| 61 | 698252002 | Non-specific intraventricular conduction delay (994) | 789 | 201 | 0 | 4 |
| 62 | 713422000 | EKG: atrial tachycardia (40) | 0 | 27 | 0 | 13 |
| 63 | 713426002 | EKG: Incomplete right bundle branch block (1606) | 1118 | 404 | 0 | 84 |
| 64 | 713427006 | EKG: complete right bundle branch block (681) | 542 | 27 | 0 | 112 |
| 65 | 74390002 | Wolff-Parkinson-White pattern (82) | 80 | 2 | 0 | 0 |
| 66 | 89792004 | Right ventricular hypertrophy (229) | 126 | 84 | 0 | 19 |

Figure 3 shows the class imbalance for the whole dataset and our train split.

Figure 3: Class counts for labels in the datasets as bar diagram. Percentages mean "Class is included in $x\%$ of files"



Since we have multiple possible class labels per patient it might be interesting to see if

1.1 Electrocardiographic Data

7

any of the classes have a correlation to each other. For that we created [Figure 4](#) which shows the Pearson correlation coefficient.

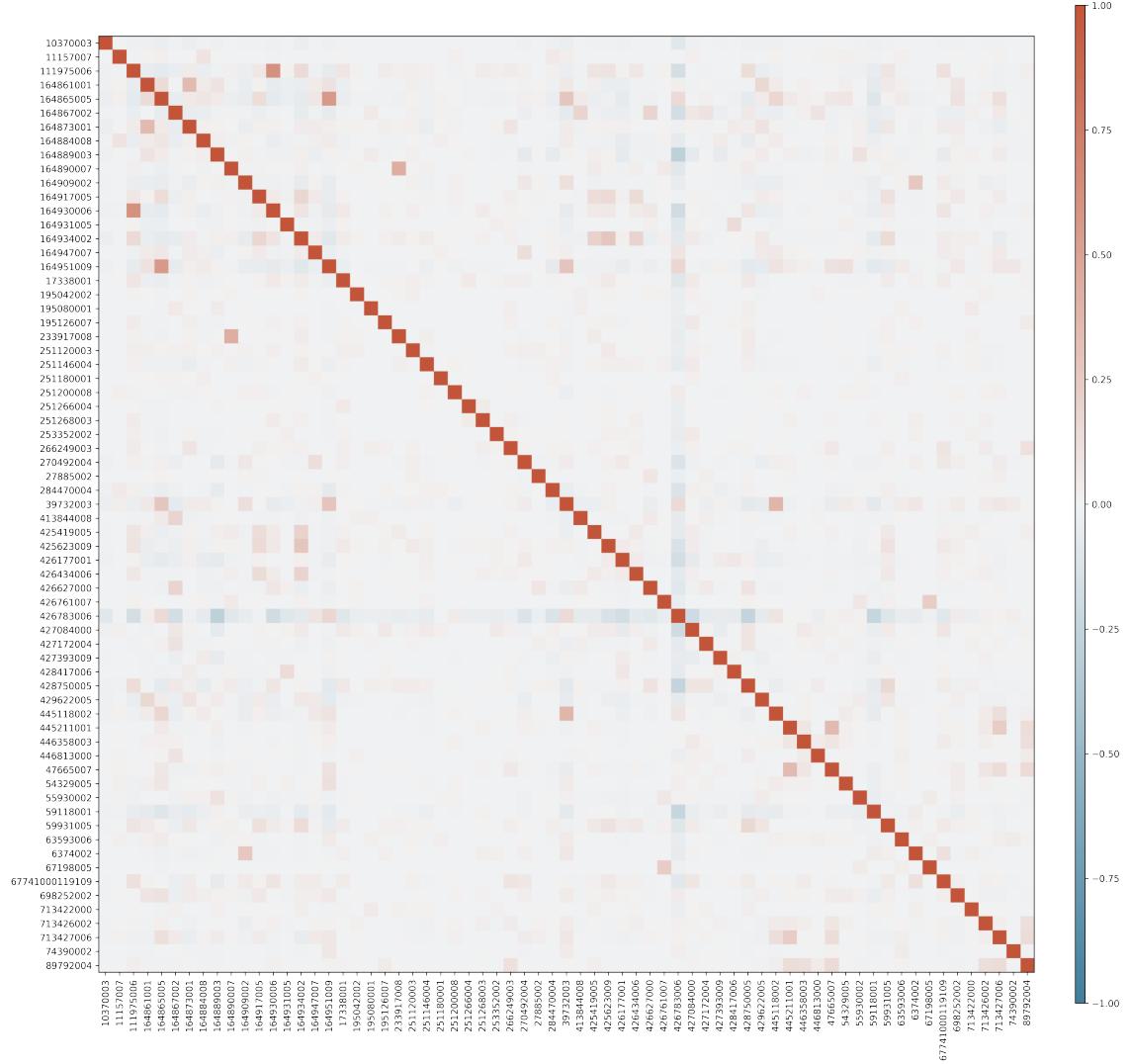


Figure 4: Pearson Class Correlation Coefficient $\rho_{X,Y}$ heatmap: Given class x , what other classes y are likely to appear at the same patient?. $\rho_{X,Y} > 0$ = positive correlation, $\rho_{X,Y} < 0$ = negative correlation, $\rho_{X,Y} \approx 0$ no correlation

Understanding and classifying ECG data (in real time) using an automated computer system could greatly help the likelihood of surviving an (incoming) infarction, since analysis by eye is time consuming and demands a trained medical professional. Monitoring patients at all times in the hospital could also be a great benefit. While doctors will probably still be mandatory to make the final diagnosis and prescribe medication, an automated classification system that also shows potential highly diagnostic locations in the signal could reduce analysis time by a big factor.

To address the problem of few data being publicly available, solutions different to fully supervised trained models could have a big impact on their success.

One of those more unsupervised learning domains is representation learning.

1.2 Representation Learning/Background

In representation/feature learning the algorithm (neural net), tries to learn representations, often in a lower dimension, by not using the real labels directly and instead uses secondary labels extracted from the data itself. After training on these artificial labels, most layers will be used again with their weights to train on the downstream task - the task to predict the desired primary labels. As such representation learning can be split into a pretraining and training phase, which are both needed to produce a working prediction model. The strength of representation learning comes from the fact that unlabeled data is often easier to come by, but which cannot be used by fully supervised methods.

Following Yann LeCun, this is also referred to as self-supervised learning[16], which is partly unsupervised learning since the models do not need the original labels, but there is also an supervisinal aspect, where one needs to define secondary labels, related to the downstream task.

There are many different approaches to representation learning, some train the biggest fraction of the layers using labels that can be extracted with high certainty from the data directly (Jigsaw from image [17], Image Rotation [18], [19]). Others augment the data and the neural net has to calculate the same lower dimensional representation for both augmented images [20].

Another way to learn representation is to predict missing or contextual information, eg. by blacking out known parts of the data and training an algorithm to fill these spots out again. The algorithm will have to find meaningful representations and thus "understand" the data in order predict the missing spots correctly.

It is hypothesized that solving secondary tasks or predicting contextual information "are fruitful partly because the context from which we predict related values are often conditionally dependent on the same shared high-level latent information" [1]. That is to say by solving a secondary task and learning representations useful for that specific task, the model will be easier to train on the real problem because the necessary representations are similar to the ones already learned.

Unfortunately the methods mentioned above focus on image recognition and use knowledge not applicable to general sequential data. Also predicting high dimensional data is a very difficult task and simpler loss functions like Mean Squared Error do not work very well. Furthermore architectures like Recurrent Neural Nets that predict only one "step" into the future will likely resort to exploiting "local smoothness" - learning features that only describe the data locally but fail to infer a more global data description.

That's where methods like CPC, short for Contrastive Predictive Coding from the paper "Representation Learning using Contrastive Predictive Coding"[1] come into play.

1.3 Contrastive Predictive Coding

1.3.1 Overview

The goal of CPC is to learn data representations in a compact (latent) embedding space, which are universally useful in downstream tasks. CPC learns these representations by predicting the future not in the sequential input data, but rather its latent lower dimensional space. Due to the lower dimension, correct predictions of future latents are easier to model. Since there is no ground truth in the unknown embedding space, the architecture is trained with the help of a loss based on Noise-Contrastive Estimation, where the model has to differentiate between latents that stem from the current sample or other samples in the batch.

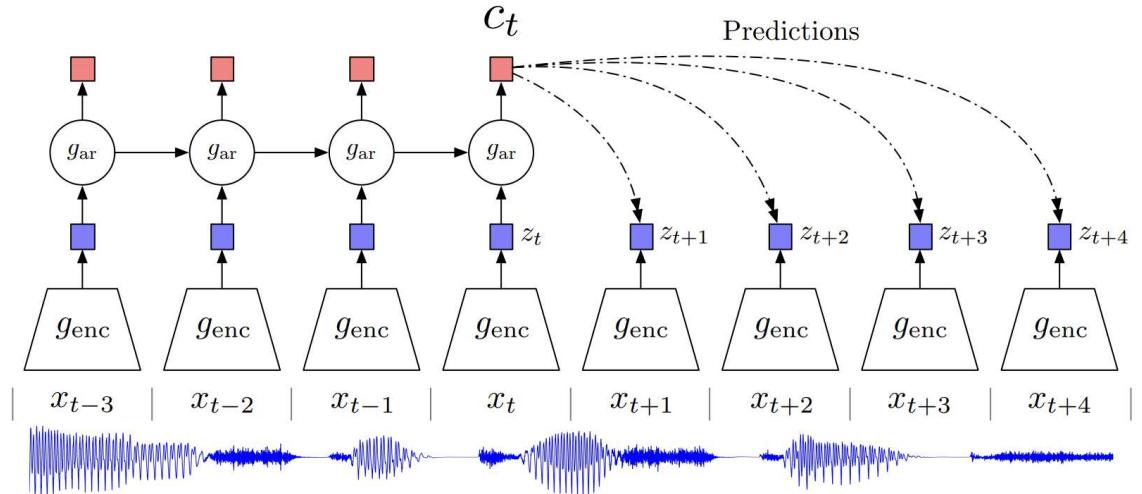


Figure 5: CPC audio architecture how it is visualized in [1]

Both [Figure 5](#) and [Figure 6](#) are CPC architectures, only differing in how the encoder encodes the data into latent representations. They can be used as a close reference for the following description:

First, CPC transforms the sequential data signal x into multiple latent variable vectors $(z_{t-m} \dots z_t, z_{t+1}, \dots, z_{t+k}, \dots, z_{t+n}) = g_{enc}(x)$ in an embedding space (with potentially lower dimension), using any encoder network. In [Figure 5](#) g_{enc} encodes parts of the data separately, in [Figure 6](#) the Encoder calculates all latents "in one go". t denotes the "current" timestep, m and n are steps into the past and future respectively and may vary depending on chosen architecture/hyperparameters and input data. The "past" latents $z_{t-m} \dots z_t$ are fed into any recurrent network/autoregressive model (e.g. g_{ar} in [Figure 5](#)), which produces a context matrix $g_{ar}(z_{t-m}, \dots, z_t) = (c_{t-m}, \dots, c_t)$, where c_t is the context vector for all seen latents z_{t-m}, \dots, z_t . Since an autoregressive model is used, any number of latent vectors can be summarized into a context, which can also be sized differently from the latent vectors. The last context c_t is then multiplied individually with n weight matrices W_1, \dots, W_n to predict n future latent vectors $\hat{z}_{t+1}, \dots, \hat{z}_{t+n}$. Multiple latents are predicted at the same time to counteract the exploitation of local smoothness and force the network to encode globally useful information into the context. In the best

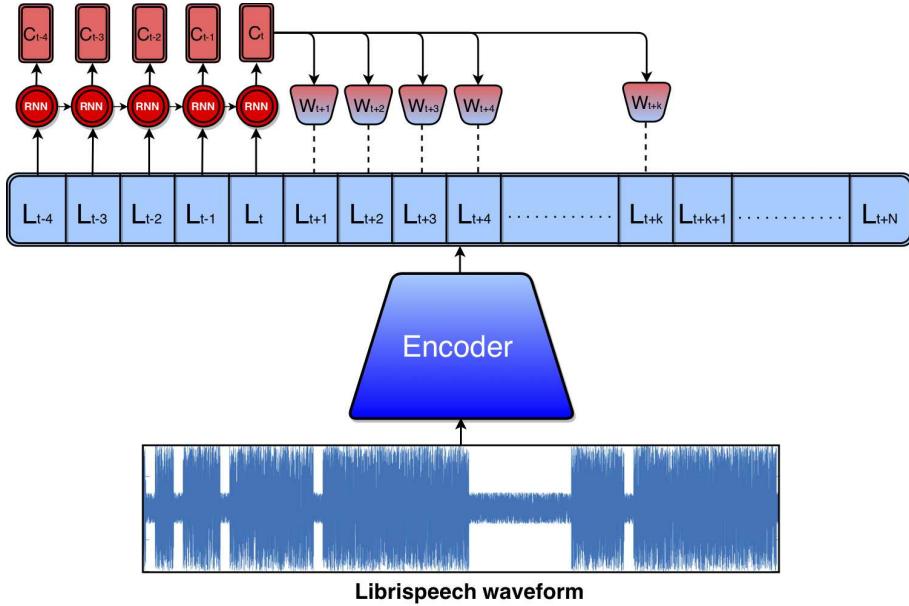


Figure 6: CPC audio architecture how it is visualized in [21]

case scenario the context represents a meaningful description of all past latents and will be a good feature matrix for following downstream tasks. The disparity between the predicted latents \hat{z} and the latents z , produced by the encoder network earlier, can then be used to form the loss and thus train both the encoder and autoregressive model jointly.

To avoid a trivial encoding of e.g. an all-zero-vector for all latents, a loss based on noise contrastive estimation [22] is used, where the network has to differentiate between positive and negative samples. Positive samples are latents that stem from $p(x_{t+k}|c_t)$ — the same datasource that the context was calculated from. Negative samples are latents that were calculated from $p(x_{t+k})$ — the unconditioned data distribution, independent from the current context. Negative samples can e.g. be taken from the data batch or in concrete terms another patient.

1.3.2 Motivation

As in [Subsection 1.2](#) already said it is assumed that unsupervised representation learning approaches "are fruitful partly because the context from which we predict related values are often conditionally dependent on the same shared high-level latent information" [1]. With this motivation we want to learn features (latent representations) that maximize the mutual information between the original input and the encoded representations. Mutual information describes how much two probability density functions have in common and how much knowing one leads to knowledge about the other. "More specifically, it quantifies the "amount of information" [...] obtained about one random variable through observing the other random variable". [23] The equation for mutual information for two probability mass functions is given by [Equation 1](#), where we arrive exactly at the papers definition in the second to last line. Here $x \in X$ are samples from the original signal and

C the contexts derived by $x \in X$:

$$\begin{aligned}
I(X; C) &= \sum_{c \in C} \sum_{x \in X} p_{X,C}(x, c) \log \left(\frac{p_{X,C}(x, c)}{p_X(x)p_C(c)} \right) \\
&= \sum_{c \in C} \sum_{x \in X} p_{X,C}(x, c) \log \left(\frac{p_{X,C}(x|c)p_C(c)}{p_X(x)p_C(c)} \right) \\
&= \sum_{c,x} p(x, c) \log \left(\frac{p(x|c)}{p(x)} \right) \\
&= \mathbb{E} \log \left(\frac{p(x|c)}{p(x)} \right)
\end{aligned} \tag{1}$$

Mutual Information $I(X; C)$ could thus be described as the mean logarithmic difference between the conditional and the unconditional joint probability distributions of X and C . If the conditional probability mass function (PMF) is similar to the unconditional PMF ($\Rightarrow X \perp C$), the difference will be close to 0. If the logarithmic difference is bigger, we know that that X and C are conditionally dependent, which means they have a higher Mutual Information.

As our goal, we want to find weights for our model, that **maximize the mutual information between the input signal X and the calculated context C** .

1.3.3 Math in detail

To reach our goal, we define the probability that given a context c_t and a set of samples X , a specific sample $x_i \in X$ was drawn from the conditional distribution $p(x_{t+k}|c_t)$, rather than from the unconditional distribution $p(x_{t+k})$. This probability is given by [Equation 2](#) with $[d = i]$ being the indicator that sample x_i is the "positive" sample [1, p. 4]:

$$\begin{aligned}
p(d = i|X, c_t) &= \frac{p(x_i|c_t) \prod_{l \neq i} p(x_l)}{\sum_{j=1}^N p(x_j|c_t) \prod_{l \neq j} p(x_l)} \\
&= \frac{p(x_i|c_t) \left(\frac{1}{p(x_i)} \prod_{l=1}^N p(x_l) \right)}{\sum_{j=1}^N p(x_j|c_t) \left(\frac{1}{p(x_j)} \prod_{l=1}^N p(x_l) \right)} \\
&= \frac{\frac{p(x_i|c_t)}{p(x_i)}}{\sum_{j=1}^N \frac{p(x_j|c_t)}{p(x_j)}}
\end{aligned} \tag{2}$$

In [Equation 2](#) the numerator $p(x_i|c_t) \prod_{l \neq i} p(x_l)$ is the probability that x_i was drawn from $p(x_{t+k}|c_t)$, while all other $x_j \in X, j \neq i$ were drawn from $p(x_{t+k})$. This gets normalized by $\sum_{j=1}^N p(x_j|c_t) \prod_{l \neq j} p(x_l)$, the sum of all probabilities where x_j was drawn from $p(x_{t+k}|c_t)$ instead of x_i . Note that $p(x_i|c_t) \prod_{l \neq i} p(x_l)$ is also included in the sum, thus $p(d = i|X, c_t) \in [0, 1]$.

The goal of the loss function will be to maximize this probability over all possible combinations between each sample and the corresponding context. Which means the model

will need to distinguish between positive and negative samples, given a context. Knowing that the mutual information means: given C , how much do we know about X , it becomes clear that a model that maximized the mutual information between the original signal and its context, will have an easier "job" to distinguish between real and wrong sample.

The categorical cross entropy loss is a commonly used loss function for training on class probability scores for N classes: $\sum_{i=1}^N y_i \log(\hat{y}_i)$ — where y_i is the ground truth - and \hat{y}_i is the model prediction.

Combining both the cross entropy loss with [Equation 2](#), we arrive at the following loss function:

$$\mathcal{L}_{\mathcal{N}} = - \mathbb{E}_{x_i \in X} \log \left[\frac{\frac{p(x_i|c_t)}{p(x_i)}}{\sum_{x_j \in X} \frac{p(x_j|c_t)}{p(x_j)}} \right] \quad (3)$$

However, since we cannot evaluate $\frac{p(x_{t+k}|c_t)}{p(x_{t+k})}$ directly, we need a function f s.t. $f_k(x_{t+k}, c_t) \propto \frac{p(x_{t+k}|c_t)}{p(x_{t+k})}$, (f needs to only be proportional to the optimal value and is not restricted to lay in the interval $[0, 1]$). In the paper $f_k(x_{t+k}, c_t) := \exp(z_{t+k}^T W_k c_t)$ is used to form a simple log-bilinear model. $W_k c_t$ is the models prediction for latent z_{t+k} , the dot product between the prediction and the ground truth expresses their similarity, because the dot product between two vectors implies whether they point in the same/opposing/orthogonal direction. This log-bilinear model is also used in the well-known word2vec model in natural language processing [[24](#)].

We thus finally arrive at the InfoNCE loss function from the paper (which can also be described as the logarithmic Softmax over vector similarities):

$$\begin{aligned} \mathcal{L}_{\mathcal{N}} &= -\mathbb{E}_X \log \left[\frac{f_k(x_{t+k}, c_t)}{\sum_{x_j \in X} f_k(x_j, c_t)} \right] \\ &:= -\mathbb{E}_X \log \left[\frac{\exp(z_{t+k}^T W_k c_t)}{\sum_{x_j \in X} \exp(z_j^T W_k c_t)} \right] \end{aligned} \quad (4)$$

The prove from the original paper, that this loss is indeed maximizing the mutual information between c_t and x_{t+k} , is given here with additional steps for easier following ([Equation 5](#)). For $f_k(x_j, c_t)$ it uses the optimal value $\frac{p(x_{t+k}|c_t)}{p(x_{t+k})}$ instead of $\exp(z_{t+k}^T W_k c_t)$:

(5)

Given a set X split into one positive sample x_{t+k} and $N - 1$ negative samples X_{neg} .

$$\mathcal{L}_N^{opt} = -\mathbb{E}_X \log \left[\frac{\frac{p(x_{t+k}|c_t)}{p(x_{t+k})}}{\sum_{x_j \in X} \frac{p(x_j|c_t)}{p(x_j)}} \right] \quad (6)$$

$$= -\mathbb{E}_X \log \left[\frac{\frac{p(x_{t+k}|c_t)}{p(x_{t+k})}}{\frac{p(x_{t+k}|c_t)}{p(x_{t+k})} + \sum_{x_j \in X_{neg}} \frac{p(x_j|c_t)}{p(x_j)}} \right] \quad (7)$$

$$= \mathbb{E}_X \log \left[\frac{\frac{p(x_{t+k}|c_t)}{p(x_{t+k})} + \sum_{x_j \in X_{neg}} \frac{p(x_j|c_t)}{p(x_j)}}{\frac{p(x_{t+k}|c_t)}{p(x_{t+k})}} \right] \quad (8)$$

$$= \mathbb{E}_X \log \left[\frac{p(x_{t+k})}{p(x_{t+k}|c_t)} \cdot \left(\frac{p(x_{t+k}|c_t)}{p(x_{t+k})} + \sum_{x_j \in X_{neg}} \frac{p(x_j|c_t)}{p(x_j)} \right) \right] \quad (9)$$

$$= \mathbb{E}_X \log \left[1 + \frac{p(x_{t+k})}{p(x_{t+k}|c_t)} \sum_{x_j \in X_{neg}} \frac{p(x_j|c_t)}{p(x_j)} \right] \quad (10)$$

$$= \mathbb{E}_X \log \left[1 + \frac{p(x_{t+k})}{p(x_{t+k}|c_t)} (N - 1) \sum_{x_j \in X_{neg}} \frac{1}{N - 1} \frac{p(x_j|c_t)}{p(x_j)} \right] \quad (11)$$

Assuming all $N - 1$ negative samples have approximately the same probability:

$$= \mathbb{E}_X \log \left[1 + \frac{p(x_{t+k})}{p(x_{t+k}|c_t)} (N - 1) \mathbb{E}_{x_j \in X_{neg}} \frac{p(x_j|c_t)}{p(x_j)} \right] \quad (12)$$

$$\approx \mathbb{E}_X \log \left[1 + \frac{p(x_{t+k})}{p(x_{t+k}|c_t)} (N - 1) \right], \text{ Assuming } c_t, x_j \text{ are (nearly) independent} \quad (13)$$

$$= \mathbb{E}_X \log \left[\frac{p(x_{t+k})}{p(x_{t+k}|c_t)} N + 1 - \frac{p(x_{t+k})}{p(x_{t+k}|c_t)} \right] \quad (14)$$

$$\geq \mathbb{E}_X \log \left[\frac{p(x_{t+k})}{p(x_{t+k}|c_t)} N \right], \text{ Assuming } p(x_{t+k}) \leq p(x_{t+k}|c_t) \quad (15)$$

$$= -\mathbb{E}_X \log \left[\frac{p(x_{t+k}|c_t)}{p(x_{t+k})} \right] + \log(N) \quad (16)$$

$$= -I(x_{t+k}, c_t) + \log(N) \quad (17)$$

Therefore $-I(x_{t+k}, c_t) \geq \log(N) - \mathcal{L}_N^{opt}$, which is also true for loss functions greater than \mathcal{L}_N^{opt} (with non optimal $f \propto \frac{p(x_i|c_t)}{p(x_i)}$). "Equation (15) quickly becomes more accurate as N increases. At the same time $\log(N) - \mathcal{L}_N$ also increases, so it's useful to use large values of N " [1].

1.3.4 Algorithm

The original CPC Loss formulation can be seen in [Figure 7a](#).

Figure 7: "Given a set $X = x_1, \dots, x_N$ of N random samples containing one positive sample from $p(x_{t+k}|c_t)$ and $N - 1$ negative samples from [...] $p(x_{t+k})$, we optimize":

$$\begin{aligned} L_N &= -\mathbb{E}_X \log \left[\frac{f_k(x_{t+k}, c_t)}{\sum_{x_j \in X} f_k(x_j, c_t)} \right] \\ &:= -\mathbb{E}_X \log \left[\frac{\exp(z_{t+k}^T W_k c_t)}{\sum_{x_j \in X} \exp(z_j^T W_k c_t)} \right] \end{aligned}$$

(a) InfoNCE Loss from [1]

However we think that this formulation can be misunderstood once it has to be implemented and hence give a mathematical formulation more along the lines of the second paper [\[25\]](#). Note that this version uses every sample $x_i \in X$ once as a positive sample: [algorithm 1](#)

Algorithm 1: InfoNCE Loss calculation

Data: A set of N full data-samples $X = (x_1, \dots, x_N)$.
/ X can e.g. be a set/batch of patients' ECG records with size $N \times \text{channels} \times \text{datalength}$* */

Input: An index $t \in \mathbb{N}$.
/ t marks the "present" timestep and splits data into "past" and "future".* */

Input: The encoder network $g_{enc}(x_i|\theta) \mapsto z_i = (z_{i,1}, \dots, z_{i,t}, z_{i,t+1}, \dots, z_{i,t+k})$
/ z_i is a set of $t+k$ latent vectors, representing consecutive timesteps of the input data in a lower dimension. How many get produced is dependent on the encoder network and data samples $x \in X$* */

Input: The context network $g_{ar}((z_{i,1}, z_{i,2}, \dots, z_{i,t})|\psi) \mapsto c_{i,t}$
/ $c_{i,t}$ is the context vector for a set of t latent vectors, for data-sample x_i* */

Input: A set of n weight matrices $\{W_1 \dots W_n\}$
/ The weight matrices are used to calculate a set of n latent vectors $\hat{z}_i = (\hat{z}_{i,t+1}, \dots, \hat{z}_{i,t+n}) := (W_1 c_{i,t}, W_2 c_{i,t}, \dots, W_n c_{i,t})$* */

Result:

$$Loss_i := -\frac{1}{k} \sum_{k=1}^n \log \left[\frac{\exp(\hat{z}_{i,t+k}^T z_{i,t+k})}{\exp(\hat{z}_{i,t+k}^T z_{i,t+k}) + \sum_{(j,l) \in \{(j,l) | j \neq i \wedge l \leq |z_i|\}} \exp(\hat{z}_{i,t+k}^T z_{j,l})} \right]$$

/ $Loss_i$ is the scalar loss value for a single positive data-sample x_i . This is close to the papers definition.* */

$$\begin{aligned} Loss &:= \frac{1}{N} \sum_{i=1}^N Loss_i = \\ &= -\frac{1}{kN} \sum_{i=1}^N \sum_{k=1}^k \log \left[\frac{\exp(\hat{z}_{i,t+k}^T z_{i,t+k})}{\exp(\hat{z}_{i,t+k}^T z_{i,t+k}) + \sum_{(j,l) \in \{(j,l) | j \neq i \wedge l \leq |z_i|\}} \exp(\hat{z}_{i,t+k}^T z_{j,l})} \right] \\ &\quad \text{/* $Loss$ is the scalar loss value where each data-samples } x_i \in X \text{ is the positive sample once.} \quad /* \end{aligned}$$

Note that the set of negative latent samples $A := \{(j, l) | j \neq i \wedge l \leq |z_i|\}$ can be replaced by e.g. $A_1 \subset \{(j, l) | j \neq i \wedge l \leq |z_i|\} \leftrightarrow A_1$ is a random subset of A , if there are too many negative samples for the hardware to handle, or by e.g. $A_2 := \{(j, l) | j \neq i \wedge l = t+k\} \leftrightarrow A_2$ is only using negative examples that share the same timestep as $z_{i,t+k}$, to provide samples that might be more similar to the positive one.

Furthermore we provide an algorithm in Python pseudo code from our understanding, to further explain the loss function and clear up potential confusion. Listing 1 shows this pseudo code for a naive implementation of the formula we provided in **algorithm 1** (with negative samples that stem from other batches but the same timestep).

```

1 def cpc_loss_naive(X:Tensor, W:List[Tensor], enc:nn.Module, rnn:nn.Module):
2     batch, channels, data_length = X.shape #eg. (64 x 12 x 4500)
3     #Encode data into latents:
4     latent_matrix = enc(X)
5     batch, timesteps, latent_size = latent_matrix.shape #eg. (64 x 27 x 128)
6     #Calculate context from latents:
7     context_matrix = rnn(latent_matrix)
8     batch, timesteps, context_size = context_matrix.shape #eg. (64 x 27 x 256)
9     #Select current timestep (here last possible):
10    n=len(W)
11    current_timestep = timesteps-n-1
12    last_context_vector = context_matrix[:, current_timestep, :]
13    #output shape: (64 x 256)
14    loss = 0.
15    for i in range(batch):
16        loss_i = 0.
17        for k in range(n):
18            #Predict latent for specific timestep k from current context:
19            latent_vector_pred_i = last_context_vector[i] @ W[k]
20            #Select latent vector for batch i at correct timestep:
21            latent_vector_i = latent_matrix[i, current_timestep+k+1, :]
22            #calculate similarity with dot product:
23            sim = latent_vector_pred_i @ latent_vector_i #scalar
24            #calculate exp(sim) for softmax
25            p_xi_ct = exp(sim)
26            nominator = p_xi_ct
27            denominator = 0.
28            for j in range(batch): #replace j with j, 1 for different sampling
29                latent_vector_j = latent_matrix[j, current_timestep+k+1, :]
30                #Calculate dot product to measure similarity:
31                sim = latent_vector_pred_i @ latent_vector_j #scalar
32                #Calculate exp(sim) for softmax
33                p_xj_ct = exp(sim)
34                denominator += p_xj_ct
35            #take log for log softmax
36            loss_i += log(nominator/denominator)
37    loss += -loss_i/n
38    return loss / batch

```

Listing 1: Naive CPC Loss Pseudo Code

```

1 def cpc_loss_fast(X:Tensor, W:List[nn.Module], enc:nn.Module, rnn:nn.Module):
2
3     batch, channels, data_length = X.shape #eg. (64 x 12 x 4500)
4
5     latent_matrix = enc(X)
6     batch, timesteps, latent_size = latent_matrix.shape #eg. (64 x 27 x 128)
7
8     context_matrix = rnn(latent_matrix)
9     batch, timesteps, context_size = context_matrix.shape #eg. (64 x 27 x 256)
10
11    timesteps_out = len(W) #eg. 12
12    current_timestep = timesteps-timesteps_out
13    last_context_vector = context_matrix[:, current_timestep-1, :] #shape: (64 x
14        256)
15    loss = 0.
16    accuracy = 0.
17    for k in range(timesteps_out):
18        #predict latents for timestep k and all items in batch:
19        pred_latent = W[k](last_context_vector) #shape: (64 x 128)
20        #calculate similarity between all encoded latents against pred_latent:
21        sim = latent_matrix @ pred_latent.T #shape eg.: (64 x 27 x 64)
22        #reshape result for easier/efficient indexing:
23        sim_r = sim.reshape(timesteps*batch, batch).T #shape eg.: (64, 1728)
24        #Select the indices where pred_{i,k}==latent_{i,k}
25        labels = torch.arange(batch) + batch*(current_timestep + k)
26        #pytorch: crossentropyloss = LogSoftmax + NLL_Loss
27        loss += crossentropyloss(sim_r, labels)
28        #Count how often the max softmax value is at the correct idx
29        accuracy += (sim_r.max(dim=-1).indices == labels).sum()
30
31    loss = loss / (timesteps_out*batch)
32    accuracy = accuracy / (timesteps_out*batch) # * pred_latent.shape[0]
33    return accuracy, loss, hidden

```

Listing 2: Optimized CPC Loss Pseudo Code. Uses CrossEntropyLoss from Pytorch

2 Implementation

In this section we describe our different models and their implementational details. All code is available at our github repository³.

2.1 CPC

Since CPC is a “self-supervised” architecture, there are different parts of the network that are used according to the current task. The pretraining architecture without labels is described in Subsubsection 2.1.1. The downstream training architecture is described in Subsubsection 2.1.2.

2.1.1 Unsupervised

CPC was implemented following the original paper [1] with focus on the section "3.1 Audio", which uses CPC as a speaker and phone classification method on the librispeech dataset. Since audio data is also sequential and thus comparable to ECG data, the architectural details were a good starting point. We adopted the encoder network with only small changes, which means we use 5 convolutional layers with kernel-sizes [10, 8, 4, 4, 4], strides [5, 4, 2, 2, 2] and ReLU activations in between, directly on the input data. The amount of channels is set to 128 after the initial 12. This is different from the original paper [1] where 512 "hidden units" are used instead. This means that each latent vector has the length $l_{size} = 128$. The latents are calculated “in one go”, which means for this architecture and a data input with dimensions $batch \times 12 \times 4500$, a latent matrix sized $batch \times l_{size} \times t_{total} = batch \times 128 \times 26$ is calculated. The total number of latent timesteps $t_{total} = 26$ is directly given by the architecture and input data length. The transposed encoder output/latent matrix is fed into a GRU model with a hidden state dimension of 256 to form the models context for each of the columns in the latent-matrix, resulting in the context-matrix of shape $batch \times 26 \times 256$. In theory, since a recurrent network is used for its calculation, each context vector in the context-matrix can represent a summary or state of the prior latent vectors. From a selected column t in the context-matrix, $t_{out} = 12$ individual linear/fully-connected layers are used to predict $t_{out} = 12$ latent vectors into the future. The value of t should not be selected to low, as a meaningful context, from which can be predicted correctly, requires multiple latents. In our case, we select the latest possible context vector $t = t_{total} - t_{out} = 26 - 12 = 14$ th timestep, which is also the most interesting context entry, since it encodes all seen latents. Note that you can also make the $t_{out} = 12$ predictions from every possible column $\{t | t_{in} \leq t \leq t_{total} - t_{out}\}$ in the context matrix in one forward pass, to make more use of the input data, however this comes at the cost of training time. t_{in} eg.= 12 is newly introduced additional hyperparameter which defines how many latent vector are at least used to encode the context. A randomly selected timestep from the set $\{t | t_{in} \leq t \leq t_{total} - t_{out}\}$ is another possibility which we used.

Now with the encoded and predicted latent values only, the CPC-loss can be drawn.

³<https://github.com/Lullatsch/ecg-cpc>

Like in the introduction already mentioned the CPC loss uses a modified noise contrastive estimation loss called *InfoNCE*. Here the negative samples are other ECG files drawn from the batch. We differentiate between the latent sets $A = \{z_{j,l} | j \neq i \wedge l \in [1, |z_j|]\}$ (all latents including past and future from other patients) and $A_2 = \{z_{j,l} | j \neq i \wedge l = t + k\}$ (only latents from other patients at the same timesteps) as negative examples. The details on the loss are explained in algorithm [algorithm 1](#), [Listing 1](#), [Listing 2](#) or in [Subsubsection 1.3.1](#).

2.1.2 Supervised

For the downstream task, the InfoNCE-loss function is not used anymore. Nevertheless the whole network that calculates latent-, context- or future latent-vectors will still be used. In the original paper [1], a single linear layer is used on the last context vector in conjunction with a multi-class linear logistic regression classifier to predict the data labels. We applied multiple different architectures to the models output; for a full list see section ??

2.2 CPC with modifications

Apart from the model described above, additional architectures similar to the CPC audio architecture were examined with changes to different parts of the network.

2.2.1 Loss calculation changes

2.2.1.1 Latent sampling As already mentioned (see right sum in denominator of [algorithm 1](#)) there are multiple possibilities as to what negative samples (other latent vectors) are compared to the "current" latent prediction: $N = \text{batch size}$, $t = \text{total timesteps}$, $t_{out} = \text{number of predicted timesteps}$

same uses all latent vectors from the same timestep as the predicted latent vector in the batch. The model has thus to recognize one value under N values for each predicted timestep, for all items in the batch = $N^2 \cdot t_{out}$ total guesses.

future uses all latent vectors from future timesteps in the loss calculation. The model has to recognize the correct latent value under $N \cdot t_{out}$ values for each predicted timestep, for all items in the batch = $N^2 \cdot t_{out}^2$ total guesses.

all uses all latent vectors from all available timesteps in the loss calculation. The model has to recognize the correct latent value under $N \cdot t$ values for each predicted timestep, for all items in the batch = $N^2 \cdot t \cdot t_{out}$ total guesses. In the results this method is the same as crossentropy which just uses a slightly different implementation and is always used in conjunction with "Fewer Latent Vectors" ([Paragraph 2.2.1.2](#)).

2.2.1.2 Fewer latent vectors One handicap we came across during training is that with many latent vectors, which might get generated for long input sequences or an encoder architecture with a lower downscaling factor, the loss calculation takes a long time because the model has to calculate big dot product matrices and the softmax afterwards, which is a costly operation. Furthermore recurrent architectures tend to consume large amounts of memory for long input sequences which even made training impossible altogether in some cases.

As a solution we limited the latents which are to be used in the loss calculation to a fixed number, instead of trying to use all latent vectors. During Downstream training, all data is used again, which allows for a more representative context matrix and is important for correct classification. The network is called `cpc_intersect_manylatents` (because it "deals with" many latents), as opposed to the standard architecture `cpc_intersect`.

2.2.1.3 Normalized latent vectors We test normalizing the latent vectors z and \hat{z} to unit vectors (length is 1), before their multiplication in the loss to receive scalar values in range $[-1, 1]$, instead of an open interval. As a result we can observe the following properties:

1. If $z \cdot \hat{z} = 1 \implies z$ and \hat{z} are parallel and point in the same direction.
2. If $z \cdot \hat{z} = 0 \implies z$ and \hat{z} are orthogonal.
3. If $z \cdot \hat{z} = -1 \implies z$ and \hat{z} are parallel and point in the opposite direction.

We justify this change because higher valued latent vectors that are not calculated from the current context, but still point in the generally same direction, might be falsely classified as the positive sample. Note that during downstream tasks, we tested both to return the un-normalized calculated latents and the unit vectors, where returning the normalized vectors worked better by a margin. The loss function thus becomes:

$$Loss_i := -\frac{1}{k} \sum_{k=1}^n \log \left[\frac{\exp\left(\frac{\hat{z}_{i,t+k}}{\|\hat{z}_{i,t+k}\|} \cdot \frac{z_{i,t+k}}{\|z_{i,t+k}\|}\right)}{\exp\left(\frac{\hat{z}_{i,t+k}}{\|\hat{z}_{i,t+k}\|} \cdot \frac{z_{i,t+k}}{\|z_{i,t+k}\|}\right) + \sum_{(j,l) \in \{(j,l) | j \neq i \wedge l \leq |z_i|\}} \exp\left(\frac{\hat{z}_{i,t+k}}{\|\hat{z}_{i,t+k}\|} \cdot \frac{z_{j,l}}{\|z_{j,l}\|}\right)} \right] \quad (18)$$

To circumvent an erroneous division by 0 and the loss becoming undefined, we clip all vector lengths to be in the interval $[\epsilon, \infty]$, where ϵ is a constant scalar (we chose $1e-6$):

$$\|z\| \cdot = \begin{cases} \|z\|, & \text{if } \|z\| > \epsilon \\ \epsilon, & \text{else} \end{cases}$$

However in practice our model scores in the result section got slightly downgraded by normalizing the latent vectors. We suspect that although having access to the properties of comparing unit vectors, this change also restricted the model, hurting classification accuracy.

2.2.2 Encoder networks

The encoder architecture from the paper produces 26 latent vectors for data with length 4500. There one cannot control how many latent vectors get produced, besides changing the encoder architecture or varying data length. Since each latent vector is calculated from at least 465 datapoints, which can be calculated by transposing the convolutions in the architecture, and each vector represents a window shifted by $5 \cdot 4 \cdot 2 \cdot 2 = 160$ (product of strides, or the downsampling factor given in the original paper), this is equal to an overlap of $\frac{465-160}{465} = \frac{305}{465} \approx \frac{2}{3}$ in the data for neighboring vectors, making the exploitation of local smoothness in the next step prediction likely. Furthermore we would like to point out that the statement from the paper: "The total downsampling factor of the network is 160 so that there is a feature vector for every 10ms of speech", is ambiguous, since it conveys the impression that one latent vector is being created from 10ms of audio. In reality one latent vector is being calculated from 465 data points $\frac{465}{16000\text{hz}} \cdot 1000 \approx 29\text{ms}$ rather than from $\frac{160}{16000\text{hz}} \cdot 1000 = 10\text{ms}$. It would be more correct to say that one latent vector is calculated from $\frac{465}{16000\text{hz}} \cdot 1000 \approx 29\text{ms}$ of audio, with an overlap of $\frac{305}{16000\text{hz}} \cdot 1000 \approx 19\text{ms}$, creating an additional non disjoint latent vector after every $\frac{160}{16000\text{hz}} \cdot 1000 = 10\text{ms}$.

2.2.2.1 Data sliding window We compare the original "intersecting" model to a different encoder network that encodes the data by breaking it into multiple non-overlapping windows and then encodes each separately. This ensures that the autoregressive model has to predict future latent vectors which are not calculated from partly identical data and thus have to learn non-local features that do not exploit "local smoothness". Additionally the differentiation between positive and negative samples should be a lot harder too, improving the latent representations. In detail that means an input array of size $batch \times channels \times length$ is split into a fixed number e.g. $M \times batch \times channels \times \frac{length}{M}$ of non-overlapping windows, which are then encoded into $M \times batch$ latent vector. The model is called "strided" in the results and `cpc_encoder_as_strided` in the code, where it acts as a module wrapper applicable to every encoder network.

One downside is that the window-size $\frac{length}{M}$ needs to be fixed or more specifically for an input with dimensions $batch \times channels \times \frac{length}{M}$ the architecture needs to produce exactly $batch \times latentsize \times 1$ values. This can be easily solved with mean pooling or similar methods but not relying on those techniques may yield more accurate results. Furthermore, using the same encoder as in the original architecture, due to no data overlap, we produce less latent vectors which becomes a problem for small data lengths. Using different encoders that can deal with smaller input sizes than 465 would solve this data restriction.

2.2.2.2 Baseline-like Network To evaluate whether potential downstream task performance differences were stemming from architecture distinctions rather than the special CPC pretraining, we used the good working baseline architecture `baseline_v8` (explained later) as an encoder network, called `cpc_encoder_likev8`. It comes with the property of reducing less values from the input data, making the output larger which can be useful if the data contains many information or if a larger output is needed for following tasks. One difficulty that arises due to the smaller downsampling factor in comparison to the standard cpc encoder, is that more latent vectors are created, which

in turn leads to heavily inflated training durations, mostly because of the latent sampling method. Additionally recurrent architectures sometimes fail to cope with very long input sequences. The issue was solved by combining this encoder network with a different architecture that uses a fixed-size subset of latent vectors, already explained in Paragraph 2.2.1.2, returning in fewer latents during the loss calculation.

2.2.3 Context prediction networks

In the original paper a GRU model is used to summarize all latent vectors prior to the current timestep into a context vector. We assume that, in the paper, the output for each latent vector is meant with "model output" and took the last output as the context. In contrast to that we implemented a network which uses the hidden state as context and predicts the future latent values from there. Since the hidden state is internally used to calculate the autoregressive model output, it should contain a lot of information about past latents, which makes it a good candidate for next-step latent prediction as well. It is called `cpc_autoregressive_hidden` in the code.

2.2.4 Latent prediction networks

Originally CPC uses a fixed number of simple weight matrices/ single linear layers to predict future latents from the current context. Next to this predictor which internally uses 12 linear layers to predict 12 latent vectors, called `cpc_predictor_v0`, we additionally implemented a predictor that completely eliminates the use of an autoregressive context network by directly predicting future latent vectors from past latents. For this we used a Temporal Convolutional Network [2], which is a convolutional many-to-many sequence architecture we also used as one of our baselines (see Baseline Subsubsection 2.3.2). As a result, no pretrained context matrix is available in downstream-training, but we relieve the model from first summarizing latent values and then secondly applying a number of linear layers synchronously.

2.2.5 Downstream task networks

For each ECG sample, CPC calculates latent vectors with the encoder model, and a context vector with the autoregressive model. Although both can be used for classification later on, it is easier to use the context vector because only one is getting calculated no matter the ECG signal length. Nevertheless if we use the latent vectors during downstream training, to account for varying dimensions, latents have either been flattened, summarized with a new autoregressive architecture, or pooled to obtain a single dimension for prediction with linear layers. The output of the networks is then activated with a sigmoid function. We differentiate between multiple network-types:

- Single Linear Layer + Pooling In the original paper, to test the capabilities of CPC, a single linear layer was used on the latent vectors. However since our input size might change we used a pooling operation to condense multiple latents into one

feature vector. Afterwards we apply a linear layer to the output. Because the context vector was used to predict future latents, it should contain enough information about the data for classification. As a result we can use the context vector as a feature vector too, which we do by feeding it to a linear layer. If both latents and context are desired, we add both together to form the output, before a sigmoid activation is used. This model is called `cpc_downstream_only` in the results.

- Single Conv. Layer + RNN Following works that omit linear layers completely, we also test a single convolutional layer that predicts the output classes in the channel dimension. If latents are used in the prediction we summarize latents with an additional GRU with hidden size 256. If both latents and context are used in downstream training the values are concatenated, instead of added. (`cpc_downstream_cnn` in the results.)
- Linear Layer + Pooling In this network we make class prediction for all latents with a single linear layer. To obtain the final result we simply take the maximum value across the data dimension for each of the separate classes. If the context is used a single linear layer is used on the context vector. If both latents and context are used we simply add both outputs together before calculating the sigmoid activation. The model `cpc_downstream_latent_maximum` uses the maximum pooling operation. The idea behind this network is that each latent might include some class that is supposed to be propagated into the final result with the maximum pooling, no matter how often the class appears. The model `cpc_downstream_latent_average` applies an average pooling operation instead, which makes classes that appear in the data at multiple locations more likely to be considered true in the final probability output.
- 2 Linear Layers + RNN If the predicted classes are not linearly separable from context and latents, additional layers have to be used. We examined two stacked linear layers with ReLU activation in between. Again, when latents are used, they are summarized with a GRU network with hidden size 256. If both context and latents are used they are stacked together and then forwarded through the two linear layers. (`cpc_downstream_twolinear` in the code)
- 2 Linear Layers + Max Pool In addition to the 2 linear layers above, we also used an identical network with the difference that latents are maximum pooled instead, and latents and context are added instead of stacked, after they have been feed through the linear layers. (`cpc_downstream_twolinear_v2` in the code)

We decided against using deeper, more powerful models, as we want to focus on the latent representations learnt during pretraining, rather than finding the theoretically best performing model in general.

2.3 Baselines

We implemented different fully supervised models to compare the CPC-models' performance on the ECG data.

2.3.1 Custom baseline architectures

Multiple hand crafted fully supervised convolutional architectures were examined, each with slight modifications to test for the best performance.

The different baseline architectures were build to have different sets of the following parameters:

Number of Convolutions: Two up to six stacked layers of one dimensional convolutional layers have been tested

Use of BatchNorm: We tried if including a BatchNorm[26] layer after each Convolution helps training and found that it slightly sped up convergence but since we trained for a fixed set of epochs this benefit was negligible

Strides: We found that using strides > 1 in early network layers improved accuracy and lead to better performance in most networks.

Pooling: Different Pooling operations have been tried either between all layers, or at the end of the network. Namely Global Average Pooling (AdaptivePooling with output size 1), Maximum and Average Pooling with different kernelsizes.

Dilation: We used dilation in different layers. As argued in the TCN paper [3] dilated convolutions can build big receptive fields in a short number of layers (receptive field is exponential to number of layers) making it ideal for our ECG data.

Kernel size: Different kernel sizes were experimented with ranging from 3 to 12 (or 1 in special cases). We observe that using bigger kernelsizes in early layers helped accuracy.

Final output Layer: We applied a linear layer to the convolutional network output (filter output map), in order to calculate scores for all the diagnostic classes. However multiple approaches have been taken to reshape the network output of shape $(N, C, L) := (\text{batch} \times \text{convchannels} \times \text{datalength})$

1. Flatten the tensor to receive a shape of $(N, C \cdot L)$
2. Take only the last value in the data dimension to receive a shape of $(N, C, 1)$
3. Use a global pooling (mean or max) operation to receive a shape of $(N, C, 1)$
4. Use an additional convolutional layer with L input channels, an output channel of 1 and a kernel size of 1, on the last dimension of the transposed data to receive a shape of $(N, C, 1)$
5. Use a recurrent architecture to summarize the values in the last dimension by outputting a vector at each step and taking the last as context, to receive a shape of $(N, C, 1)$

Although 1 seems like a good idea because no information is lost, the accuracy was low when L was large and the weight matrix for the linear layer gets substantially bigger, resulting in longer training times, higher memory requirements and a harder training process. Option 2 is the least taxing operation, however a lot of information is lost in the

process which also reflected in a lowered accuracy. Option 3 does not introduce new parameters and worked approximately as good as option 2. However the bigger the filter output map, the worse the network performed in general. Option 4 performed the best by a margin, but also introduced new parameters since a new weight matrix for the filter has to be learned. However we opt to still use this option because it is more expressive and can even learn both option 2 or 3, if desired. Furthermore convolutions with a 1x1 kernel (1 in our case because of data dimension) are widely used as a downsampling measure and are established in many models. Option 5 can be compared to the CPC architecture, however when trained supervised end-to-end this method did perform poorly. We suspect that the partly large values for L make it difficult to train recurrent architectures.

While we think that there are many possible approaches to summarizing the last layers output, the best solution might be to choose architectures that obtain small values for L .

Most models follow a general structure:

Multiple one-dimensional Convolution Layers with decreasing kernel sizes and either stride or dilation, get stacked with ReLU after each Layer. Batchnorm is added to some models. The first few layers of the network are supposed to downsample the data using big kernel sizes and strides, reducing memory footprint for the model drastically. After the downsampling layers, additional one-dimensional convolutional layers have been used, this time with a mostly fixed kernel size of 3, low or no strides and no or increasing dilation, which increase the networks receptive field, potentially extracting wide spanning features. The number of filters is increased from 12 to 128 to allow enough features to be learned by the model. The filter number has been kept constant at 128, which is unusual but since we had differing layer numbers we opted for a simple solution. Finally the non-channel dimension has been summarized by a one of the techniques described in [Final output Layer](#). The resulting data of shape (Batch-size, 128, 1) is fed into a fully connected or convolutional layer with sigmoid as activation function, to predict class labels with probability scores $\in [0, 1]$ each. Note that Softmax is not used as the probabilities do not add up to 1 in multi-label classification.

Table 1 summarizes most of the differing model attributes described above. The model names are the same as in the results.

Because the summary table can not fully capture all details of the different models, we will briefly describe our baseline models here, that either do not fit the above general description or get used often:

BL_v0 2 one-dimensional convolutional layers, with kernel-sizes [7, 3], dilations [1, 3] and ReLU activations. The output is summarized by a convolutional with kernel size 1 (option 4).

BL_v0_2 consists of 5 Conv1D Layers with kernel sizes of [7, 5, 5, 3, 3], dilations of [1, 1, 1, 1, 2, 4], strides [4, 3, 3, 1, 1, 1] and ReLU inbetween. The channel number is dropped down to 32 in the first layer with stride 1, and then increased back up to 64. The idea behind this network is to downsample the input in the first few layers and then extract spanning features with dilations without striding.

BL_v2 BL_v2, consists of 5 Conv1D Layers with kernel sizes of 3 and dilations of [1, 2, 4, 8, 16], the channel size is 128 after the initial 12. BatchNorm and ReLU is

applied after each layer. At the end of each besides the last layer a maximum pooling layer with filter size 3 is used. The output is summarized with a global average pool like option 3 describes.

BL_v8 Similar to BL_v2 but smaller, Baseline_v8 consists of 4 Conv1D Layers with kernel sizes of 3 and dilations of [1, 2, 4, 8], the channel size is 128 after the initial 12. BatchNorm and ReLU is applied after each layer. At the end of each besides the last layer a maximum pooling layer with filter size 3 is used. The output is downsampled in the data dimension like option 4 describes.

2.3.2 Literature architectures

In addition to the custom-designed architectures above, we tested already existing or slightly modified architectures that were reported to work on timeseries classification:

MLP The Multi-Layer-Perceptron is a three linear layer network with dropout and ReLU activations from [27] (BL_MLP in results)

FCN The Fully Convolutional Network consists of three convolutional layers with BatchNorm, ReLU and a global pooling operation from [27]. (BL_FCN in results)

Multi Scale 1D ResNet A Residual Network which uses one instead of two dimensional convolutions, tuned for timeseries classification [28]. Multiple subarchitectures are used in parallel to allow for modeling close and long distance coherence in sequential data. Each subarchitecture uses different filter sizes and striding to produce local or global features at the same time. (BL_v14 in results.)

TCN Temporal Convolutional Networks [2] aim to provide an almost fully convolutional sequence-to-sequence alternative to RNNs, which brings perks like low memory requirements, more stable gradients and parallelism (predicting many timesteps at once). Exponentially increasing dilation sizes help in building a big receptive field, theoretically ideal for modeling long distance coherence in sequential data. (BL_TCN in results) Since this model's output has the same length as the models input, we had yet again to find a solution to reduce the output length and tested multiple variants: TCN_down uses a convolutional layer with filter-size 1 to reduce output. TCN_last uses the last value in the output of the data dimension, following one of the examples in the authors github repository⁴

Alexnet The Alexnet is an architecture known for image classification. Here we implemented it with 1D convolutions and two different kernel sizes otherwise following the implementation from pytorch [29]. (BL_alex_v2 in the code)

Again, Table 1 summarizes most model attributes described above.

⁴https://github.com/locuslab/TCN/tree/master/TCN/mnist_pixel.

| Model Name | Convolutional Layer Number | Sum of Strides | Sum of Dilation | Sum of Paddings | Sum of Filters | uses BatchNorm | uses Max Pool | uses Adaptive Average Pooling | uses Linear | uses LSTM | Final Layer |
|---------------------|----------------------------|----------------|-----------------|-----------------|----------------|----------------|---------------|-------------------------------|-------------|-----------|-------------|
| BL_alex_v2 | 5 | 25 | 8 | 7 | 174 | no | yes | yes | yes | no | 1 |
| BL_FCN | 3 | 3 | 3 | 0 | 16 | yes | no | no | no | no | 3 |
| BL_MLP | 0 | 0 | 0 | 0 | 0 | no | no | no | yes | no | 2 |
| BL_mn_simplest_lstm | 0 | 0 | 0 | 0 | 0 | no | no | no | yes | yes | 5 |
| BL_TCN_block | 3 | 4 | 4 | 0 | 7 | no | no | no | yes | no | 1 |
| BL_TCN_down | 15 | 15 | 31 | 56 | 39 | no | no | no | yes | no | 4 |
| BL_TCN_flatten | 15 | 15 | 31 | 56 | 39 | no | no | no | yes | no | 1 |
| BL_TCN_last | 14 | 14 | 30 | 56 | 38 | no | no | no | yes | no | 2 |
| BL_v0 | 3 | 3 | 5 | 0 | 11 | no | no | no | yes | no | 4 |
| BL_v0_1 | 3 | 4 | 5 | 0 | 11 | no | no | no | yes | no | 4 |
| BL_v0_2 | 7 | 14 | 11 | 0 | 27 | no | no | no | yes | no | 4 |
| BL_v0_3 | 7 | 14 | 11 | 0 | 27 | no | no | no | yes | no | 4 |
| BL_v1 | 6 | 17 | 6 | 0 | 36 | yes | no | yes | yes | no | 3 |
| BL_v2 | 5 | 20 | 36 | 0 | 30 | yes | yes | yes | yes | no | 3 |
| BL_v3 | 5 | 5 | 16 | 0 | 13 | yes | no | no | yes | no | 4 |
| BL_v4 | 5 | 5 | 16 | 0 | 13 | no | no | no | yes | no | 4 |
| BL_v5 | 5 | 5 | 16 | 0 | 13 | no | no | no | yes | no | 4 |
| BL_v6 | 7 | 7 | 64 | 0 | 19 | no | no | no | yes | no | 4 |
| BL_v7 | 6 | 7 | 32 | 0 | 20 | no | no | no | yes | no | 4 |
| BL_v8 | 5 | 14 | 19 | 0 | 22 | yes | yes | no | yes | no | 4 |
| BL_v9 | 5 | 14 | 8 | 0 | 22 | yes | yes | no | yes | no | 4 |
| BL_v14 | 29 | 71 | 30 | 22 | 143 | yes | yes | no | yes | no | 4 |
| BL_v15 | 5 | 14 | 96 | 0 | 26 | yes | yes | no | yes | no | 4 |

Table 1: Model attributes summarized. Final Layer number refers to the enumeration Final Output Layer
2.3.1.

3 Training

3.1 Data preprocessing

All data has been resampled to 500hz, if it were not already. The networks are trained on cropped data with length fixed to 4500 (mostly down from 5000), all 12 channels have been used. In rare cases, where the available data was shorter than 4500, we repeatedly padded the rightmost value for each channel. The data was split into training-, validation- and test-set randomly with a fraction of $\frac{7}{10}$, $\frac{2}{10}$, and $\frac{1}{10}$ respectively, while still ensuring that every class follows those fractions approximately. The data distribution is displayed in the introduction in [Figure 3](#).

We count label occurrences and exclude all labels that occur less than 20 times over all datasets, which results in 67 different ECG classes (down from 94). The class/label counts obtained will also be used later as loss function weights in training, where we will differentiate between calculating the loss with and without these weights.

Three different data normalization methods have been tested independently:

min-max The minimum and maximum for each channel was drawn over all datapoints in each channel to rescale the data into the interval $[0, 1]$. This means for data X of shape 12×4500 , 12 different minima and maxima are drawn. The calculation for each datapoint independently is given here, where $x_{i,j} \in X$, the first index i is the channel and the second index j is the entry in the data dimension:

$$\forall i, j : x'_{i,j} = \frac{x_{i,j} - \min_d(x_{i,d})}{\max_d(x_{i,d}) - \min_d(x_{i,d})}$$

This method is equal to the "Min-Max Normalization" found elsewhere.

min-max (augmented) The minimum and maximum at each datapoint is drawn over all channels to rescale each datapoint into the interval $[0, 1]$. For example this means for data X of shape 12×4500 , 4500 different minima and maxima are drawn over all channels:

$$\forall i, j : x'_{i,j} = \frac{x_{i,j} - \min_c(x_{c,j})}{\max_c(x_{c,j}) - \min_c(x_{c,j})}$$

This has the effect that each datapoint $x_{i,j}$ captures the relationship between all channels. We are aware that this changes the input data in a drastical manner and originally we applied this normalization method by mistake. Nevertheless the models were mostly able to train successfully and since all models had to train with the handicap, the results are still interesting to look at.

z-Score Z-Score Normalization shifts the data by subtracting the mean from each point and rescales it afterwards by dividing by the standard deviation.

$$\forall i, j : x'_{i,j} = \frac{x_{i,j} - \text{mean}(x_i)}{\text{std}(x_i)}$$

As a result the normalized data has a mean of 0 and a standard deviation of 1, which is said to improve model training stability. This method resulted in the overall best model performances, especially for the baseline architectures. We used it almost exclusively for all models, besides in our experiment [Subsubsection 5.1.5](#).

Principle Component Analysis has been tested but did not yield good results, feature (heart beat) extraction as in [30] has been applied but random samples suggested big variations in the extracted beat's quality. This is to say that multiple beats got extracted as one, longer beats got cut short etc. Furthermore beat extraction works best on beats that follow the normal pattern, however the most interesting ones for classification might differentiate enough to not be detected anymore and thus introduces a potentially harming bias, where interesting beats get discarded with a higher probability. Also keep in mind that data labels only exist for the whole file where multiple heart beats are present, meaning each beat on its own cannot be correctly labeled with high confidence.

In addition to the beat extraction method from [30] we briefly tested outlier detection based on [31] to find possibly interesting candidates for classification of single beats, but the results were varying widely between different channels and datasets. Furthermore we raise the same concern as above about missing out on beats that are not getting classified as outliers.

3.2 Train settings

Adam was used as optimizer with the parameters set to $lr = 3e - 4, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e - 8, weight_decay = 0$, all values were not changed for the most part. For models trained with z-score normalization, we used a learning rate scheduler with an initial learning rate of $3e - 3$ for pretraining and $3e - 4$ for downstream training, which multiplies the learning rate by a factor of 0.1 after 5 epochs with no validation loss decrease.

We used a batch-size of 128, the loss function is Binary Cross Entropy with or without the class weights multiplied to the loss (obtained in [Subsection 3.1](#)).

To allow a fair comparison between CPC and the baselines, we give access to the data for 120 epochs in total. Our baselines that utilize no pretraining can thus train for unrestricted 120 data iterations — while CPC needs sufficient time to pretrain in order for the loss to be minimized, we chose 100 epochs for the pretraining without labels and 20 epochs with labels for training on the downstream (classification) task. Since trivially 120 epochs with labels should be better or equal than 100 epochs without labels plus 20 epochs with labels, we know that a possible performance edge for CPC is achieved due to its properties and not because it has access to the data for longer. The split 100+20 was obtained by looking at CPC loss curves, where we focused on decreasing the pretraining loss sufficiently. However we want to mention that this specific allocation is not a given and performance might increase with more downstream- and less pretrain epochs. With more GPU time, a training until convergence for all models could possibly be a better alternative. In some experiments we had to downstream train our CPC models for up to 40 epochs which we will denote where applied.

3.2.1 CPC

As already mentioned The CPC architecture from the paper was trained for 100 epochs on the data without labels to find the optimal weights for the encoder network. After the 100 epochs, we trained the network with the added downstream layers for the downstream task, for 20 epochs with labels available.

We differentiate between two downstream training settings:

1. All weights are updated (Gradients for both encoder architecture and classification layers are calculated and updated).
2. Only weight gradients for the classification layer(s) are updated. (All weight gradients in the CPC network remain untouched aka. weights are "frozen").

We make this differentiation because we want to better evaluate the CPC pretraining step, which is very important when weights are "frozen".

4 Evaluation

4.1 CPC Evaluation during Pretraining

As an addition to the CPC loss value, we evaluate how well CPC is able to differentiate between positive and negative latent vectors during training. For that we take the maximum value indices in the logsoftmax matrix and compare those to the index with the correct latent vectors. If the indices match we add one to the count of correctly classified latents (see line 28 in Listing 2). In the end we divide the sum by the number of total predicted latent vectors to receive values $\in [0, 1]$ (see line 31 in Listing 2).

4.2 Model Evaluation during Downstream Training

As an addition to the loss value we calculate metrics during downstream training.

To better evaluate performance on a sparse probability label vector, which means only one to five values of 67 classes (in case of all datasets combined) are set to something other than 0.0, the following accuracy functions have been used during training to evaluate progress. They are based on the widely used Mean Squared Error/Brier score [32], but are split into parts which measure different things:

Given the class label probabilities $Y = (y_1, \dots, y_i, \dots, y_n)$ the prediction probabilities $\hat{Y} = (\hat{y}_1, \dots, \hat{y}_i, \dots, \hat{y}_n)$ with $y_i \in [0, 1]$ and $\hat{y}_i \in [0, 1]$

$$\text{zerofit}(Y, \hat{Y}) = 1 - \frac{\sum_{i, \text{where } y_i=0} (y_i - \hat{y}_i)^2}{|\{\forall i, \text{where } y_i = 0\}|} \quad (19)$$

$$\text{classfit}(Y, \hat{Y}) = 1 - \frac{\sum_{i, \text{where } y_i \neq 0} (y_i - \hat{y}_i)^2}{|\{\forall i, \text{where } y_i \neq 0\}|} \quad (20)$$

$$\text{meanfit}(Y, \hat{Y}) = \frac{\text{zerofit}(Y, \hat{Y}) + \text{classfit}(Y, \hat{Y})}{2} \quad (21)$$

Equation Equation 19 is supposed to capture how well the model reproduces the probability scores for classes labels with probability = 0. $\text{zerofit}(Y, \hat{Y}) \in [0, 1]$, where $\text{zerofit}(Y, \hat{Y}) = 0$ is the worst and $\text{zerofit}(Y, \hat{Y}) = 1$ the best score.

Equation Equation 20 is supposed to capture how well the model reproduces the probability scores for classes labels with probability $\neq 0$. $\text{classfit}(Y, \hat{Y}) \in [0, 1]$, where $\text{classfit}(Y, \hat{Y}) = 0$ is the worst and $\text{classfit}(Y, \hat{Y}) = 1$ the best score.

Equation Equation 21 is the mean between Equation 19 and Equation 20 and captures if both goals have been met. Again it holds that $\text{meanfit}(Y, \hat{Y}) \in [0, 1]$.

Since the loss function is independent of the above metrics, the custom accuracy functions are solely a tool to better monitor network performance during training and do not change the weights in any way. They were used in conjunction with the loss values to determine a suitable fixed number of training epochs across all models — the aforementioned 120 epochs for baselines and 20 epochs for cpc networks.

4.3 Evaluation after training

4.3.1 Quantitative

To evaluate our models we calculate precision, recall, accuracy, ROC-AUC and F1-scores [33] with the prediction probabilities on the test-dataset. Since the models output probability scores for each class and no binary values, which are however needed for most metrics, we need to set thresholds that define whether a class gets classified as True or False for a given datasample:

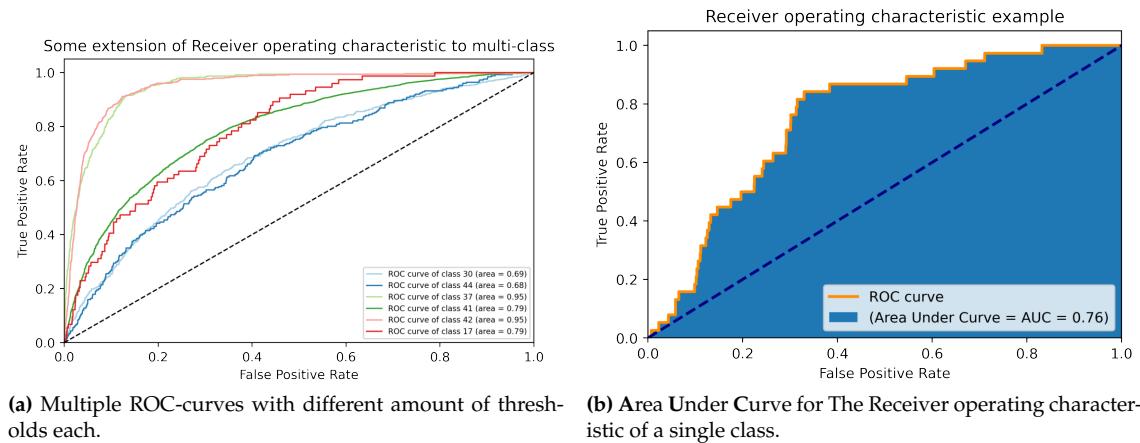
$$\text{class}_i := \begin{cases} \text{False}, & \text{if } p(\text{class}_i) < \text{threshold}_i \\ \text{True}, & \text{if } p(\text{class}_i) \geq \text{threshold}_i \end{cases} \quad (22)$$

With the decision boundary for each class set, we can count the "True Positives" = TP, "False Positives" = FP, "True Negatives" = TN and "False Negatives" = FN, which means that the data has been either classified correctly/falsely as positive/negative sample respectively (see table 4.3.1).

| | | Class is: | |
|-------------------|-------|---------------------|---------------------|
| | | TRUE | FALSE |
| Classified as: | TRUE | True Positive (TP) | False Positive (FP) |
| | FALSE | False Negative (FN) | True Negative (TN) |

We select the "best" thresholds based on the Receiver Operating Characteristic-curve for each individual prediction class. The ROC-curve (see Figure 8b) is the False Positive Rate (FPR) plotted against the True Positive Rate (TPR) for every possible threshold in Equation 22 for class i . The best threshold is then selected by evaluating $\underset{i}{\operatorname{argmax}}(TPR_i - FPR_i)$. This takes place on the validation dataset and each model has their own optimal decision boundaries for each class.

Figure 8: ROC example



(a) Multiple ROC-curves with different amount of thresholds each.

(b) Area Under Curve for The Receiver operating characteristic of a single class.

The TP, FP, TN, FN counts then form the base for the following metrics (see equations 9):

$$\begin{aligned}
precision_i &:= \frac{TP_i}{TP_i + FP_i} \\
recall_i &:= \frac{TP_i}{TP_i + FN_i} \\
accuracy_i &:= \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i} \\
TPR_i &:= \frac{TP_i}{TP_i + FN_i} \\
FPR_i &:= \frac{FP_i}{FP_i + TN_i} \\
F_{1,i} &:= 2 \cdot \frac{precision_i \cdot recall_i}{precision_i + recall_i} \\
ROCAUC_i &:= \text{AUC of ROC-curve}
\end{aligned} \tag{23}$$

Figure 9: Different score metrics to measure performance. All score functions calculate the score for a given class i .

$$\begin{aligned}
\text{macroavg(score)} &:= \frac{1}{N} \sum_i^N \text{score}(TP_i, FP_i, TN_i, FN_i) \\
\text{microavg(score)} &:= \text{score}(TP, FP, TN, FN) \\
&= \text{score}(\sum TP_i, \sum FP_i, \sum TN_i, \sum FN_i)
\end{aligned} \tag{24}$$

For example:

$$\begin{aligned}
\text{macroavg}(TPR) &:= \frac{1}{N} \sum_i^N \frac{TP_i}{TP_i + FN_i} \\
\text{microavg}(TPR) &:= \frac{\sum_i^N TP_i}{\sum_i^N TP_i + \sum_i^N FN_i}
\end{aligned}$$

Figure 10: Micro- and Macro-average calculations for a given score function. Example for TPR at bottom

To obtain comparable results we also calculate *micro-* and *macro-average* scores (see equation 10). The *macro-average* sums the metrics in Figure 9 for every individual class and averages by dividing through the number of classes. That way every class weighs the same in the resulting average, no matter how many samples that class has. The *micro-average* can be calculated by summing the TPs, FPs, TNs and FNs for all classes respectively and then calculate the metrics in Figure 9, or by weighting the calculated class scores by their fraction of the total number of samples. The micro average thus weights the score metrics with the underlying class distribution. It is important to look at both micro and macro average scores to judge a models performance — high micro average scores but low macro average scores suggest that small classes are being overlooked by the model, which is especially disastrous considering fatal illnesses are very rare in most cases. The opposite; a high macro average- but low micro average score suggests that the models accuracy in larger parts of the data is small, which, in our case, would most likely

mean that healthy patients get diagnosed with illnesses on accident.

4.3.2 Qualitative

Apart from giving scores that allow a basic and quick evaluation of all of our models, we show various graphics for selected models to allow a more thorough overview of the models performance. They are explained in their respective section.

Table 2: Baseline Results

| model | micro | macro |
|--------------|--------------|--------------|
| BL_alex_v2 | 0.903 | 0.774 |
| BL_FCN | 0.458 | 0.503 |
| BL_MLP | 0.500 | 0.500 |
| BL_TCN_block | 0.876 | 0.638 |
| BL_TCN_down | 0.924 | 0.806 |
| BL_TCN_last | 0.869 | 0.649 |
| BL_v0 | 0.916 | 0.787 |
| BL_v0_1 | 0.930 | 0.824 |
| BL_v0_2 | 0.867 | 0.664 |
| BL_v0_3 | 0.895 | 0.757 |
| BL_v1 | 0.920 | 0.814 |
| BL_v2 | 0.940 | 0.855 |
| BL_v4 | 0.930 | 0.830 |
| BL_v5 | 0.865 | 0.668 |
| BL_v7 | 0.924 | 0.817 |
| BL_v8 | 0.937 | 0.847 |
| BL_v9 | 0.933 | 0.844 |
| BL_v14 | 0.951 | 0.885 |
| BL_v15 | 0.931 | 0.840 |

Micro and Macro AUC-ROC scores for all baseline models. Baseline Models are trained on the downstream task for 120 epochs.

5 Results

5.1 Quantitative

5.1.1 Baselines compared to CPC

How does the CPC architecture compare against our baseline models in a fully supervised setting?

Table 2 shows the results for our baseline/literature models. The literature architecture BL_v14, which is tuned for timeseries classification by making use of parallel architectures with differently sized receptive fields, performs the best for both micro and macro ROC-AUC scores. This model performed the best compared to all models in general.

The fully connected network BL_FCN and the multilayer perceptron BL_MLP both failed to learn the data beyond random guessing classes.

The best Temporal Convolutional Network is BL_TCN_down which uses a convolutional layer with filtersize 1 to summarize the values in the data dimension.

At this place we also want to mention the BL_v8 model which performed well and is used as encoder network in selected CPC models later on.

We compare the baseline network results to different CPC models (with the standard encoder-, context- and predictor-network), that were trained fully supervised for 120

Table 3: CPC as Baseline Results

| model | Freeze CPC | strided | Downstream Model | micro | macro |
|-------|------------|---------|-------------------------------|--------------|--------------|
| CPC | False | False | cpc_downstream_latent_average | 0.943 | 0.879 |
| CPC | False | False | cpc_downstream_latent_maximum | 0.944 | 0.877 |
| CPC | False | False | cpc_downstream_twolinear_v2 | 0.933 | 0.845 |
| CPC | False | True | cpc_downstream_latent_average | 0.941 | 0.872 |
| CPC | False | True | cpc_downstream_latent_maximum | 0.941 | 0.873 |
| CPC | False | True | cpc_downstream_twolinear_v2 | 0.924 | 0.831 |

CPC Models trained without pretraining for 120 epochs (with labels). All models shown use the standard encoder-, context-network

epochs, to obtain their best possible scores within the 120 epoch limit. These results can be seen in [Table 3](#): The models trained without pretraining have overall high average scores, only falling slightly behind `BL_v14` (Multi-Scale-Residual Network), which proves the CPC architecture itself is clearly capable of classifying the data sufficiently in a supervised setting.

The intersected models have a slim edge over strided models ([Paragraph 2.2.2.1](#)), most likely because they are less restrictive.

Surprisingly the models that utilized the more expressive two layer linear network fell slightly behind in performance, which we can only explain by a higher training difficulty, as the deeper model should be able to at least match the performance of smaller models. Similar occurrences have been already observed in the past, well explained in the Deep Residual Network paper [[34](#)].

5.1.2 CPC Downstream task with frozen weights

How capable is CPC's pretraining? How good are the learned representations on their own?

We want to measure how good the latent vector features are suited in our downstream task, by pretraining the model for 100 epochs without labels and downstream train for only 20 epochs with labels available. After pretraining, we are not updating any weights in the whole CPC network (CPC network is "frozen"). This is to say only the weights in the specific downstream task network (see [Subsubsection 2.2.5](#)) are altered during back-propagation. The results in [Table 4](#) show good results, however it is clear that they fall behind in performance compared to the models who had access to the labeled data for 120 epochs. We observe that the two layered linear network is the best performing downstream model, suggesting that the learned latent vectors are not fully linearly separable for this downstream task. The added complexity from introducing a second layer helped classifying the learned latent representations, in contrast to the networks trained purely on the downstream task, where the added complexity lead to lower scores.

Additionally the strided models show lower macro scores.

In comparison we show the results for the same models with randomly initialized

Table 4: CPC ROC-AUC scores, with no CPC layers updated during downstream training

| model | Freeze CPC | strided | CPC Sampling Mode | Downstream Model | micro | macro |
|-------|------------|---------|-------------------|-------------------------------|--------------|--------------|
| CPC | True | False | all | cpc_downstream_latent_average | 0.899 | 0.792 |
| CPC | True | False | all | cpc_downstream_latent_maximum | 0.910 | 0.797 |
| CPC | True | False | all | cpc_downstream_twolinear_v2 | 0.889 | 0.816 |
| CPC | True | False | crossentropy | cpc_downstream_latent_average | 0.912 | 0.788 |
| CPC | True | False | crossentropy | cpc_downstream_latent_maximum | 0.908 | 0.780 |
| CPC | True | False | crossentropy | cpc_downstream_twolinear_v2 | 0.872 | 0.809 |
| CPC | True | False | same | cpc_downstream_latent_average | 0.903 | 0.772 |
| CPC | True | False | same | cpc_downstream_latent_maximum | 0.901 | 0.802 |
| CPC | True | False | same | cpc_downstream_twolinear_v2 | 0.882 | 0.823 |
| CPC | True | True | all | cpc_downstream_latent_average | 0.902 | 0.766 |
| CPC | True | True | all | cpc_downstream_latent_maximum | 0.904 | 0.779 |
| CPC | True | True | all | cpc_downstream_twolinear_v2 | 0.900 | 0.806 |
| CPC | True | True | crossentropy | cpc_downstream_latent_average | 0.896 | 0.715 |
| CPC | True | True | crossentropy | cpc_downstream_latent_maximum | 0.896 | 0.708 |
| CPC | True | True | crossentropy | cpc_downstream_twolinear_v2 | 0.872 | 0.793 |
| CPC | True | True | same | cpc_downstream_latent_average | 0.905 | 0.780 |
| CPC | True | True | same | cpc_downstream_latent_maximum | 0.903 | 0.794 |
| CPC | True | True | same | cpc_downstream_twolinear_v2 | 0.917 | 0.808 |

ROC-AUC scores for standard CPC models trained with pretraining for 100 and training on the downstream task for 20 epochs (with the CPC weights frozen).

Table 5: CPC ROC-AUC scores, random weight initialization, no CPC layers updated during downstream training

| model | Freeze CPC | strided | Downstream Model | micro | macro |
|-------|------------|---------|-------------------------------|--------------|--------------|
| CPC | True | False | cpc_downstream_latent_average | 0.852 | 0.514 |
| CPC | True | False | cpc_downstream_latent_maximum | 0.861 | 0.580 |
| CPC | True | False | cpc_downstream_twolinear_v2 | 0.864 | 0.582 |
| CPC | True | True | cpc_downstream_latent_average | 0.853 | 0.510 |
| CPC | True | True | cpc_downstream_latent_maximum | 0.852 | 0.575 |
| CPC | True | True | cpc_downstream_twolinear_v2 | 0.861 | 0.577 |

Table 6: ROC-AUC scores for standard CPC models without pretraining (random weights); trained for 20 epochs on the downstream task (with the CPC weights frozen)

weights and no pretraining whatsoever, as to prove the pretraining is actually relevant and the data is not just classifiable with random weights and a single/two layer(s) within 20 epochs of training time: [Table 6](#).

As expected the models fail to satisfactorily classify the ecg-data, which becomes clear when looking at the macro ROC-AUC scores. The high micro scores suggest that one layer is still enough to learn the big parts of the data somehow, where one has to keep in mind that e.g. the biggest class "Sinus Rythm" is included in $\approx 50\%$ of all ecg-files, which means correctly predicting this class has a big positive impact on micro average scores. (Reminder: see [Figure 3](#) for all classes.) Both table [Table 4](#) and [Table 6](#) together suggest that the features learned by CPC in an unsupervised fashion are useful for the supervised downstream task, even if the network is restricted by making weight updates only in a previously untrained new layer, which fits the extracted features to the classes.

Table 7: CPC ROC-AUC scores, with all layers updated in downstream training

| model | Freeze CPC | strided | CPC Sampling Mode | Downstream Model | micro | macro |
|-------|------------|---------|-------------------|-------------------------------|--------------|--------------|
| CPC | False | False | all | cpc_downstream_latent_average | 0.911 | 0.800 |
| CPC | False | False | all | cpc_downstream_latent_maximum | 0.914 | 0.810 |
| CPC | False | False | all | cpc_downstream_twolinear_v2 | 0.900 | 0.818 |
| CPC | False | False | crossentropy | cpc_downstream_latent_average | 0.898 | 0.763 |
| CPC | False | False | crossentropy | cpc_downstream_latent_maximum | 0.902 | 0.785 |
| CPC | False | False | crossentropy | cpc_downstream_twolinear_v2 | 0.887 | 0.803 |
| CPC | False | False | same | cpc_downstream_latent_average | 0.898 | 0.828 |
| CPC | False | False | same | cpc_downstream_latent_maximum | 0.901 | 0.832 |
| CPC | False | False | same | cpc_downstream_twolinear_v2 | 0.896 | 0.851 |
| CPC | False | True | all | cpc_downstream_latent_average | 0.921 | 0.837 |
| CPC | False | True | all | cpc_downstream_latent_maximum | 0.919 | 0.821 |
| CPC | False | True | all | cpc_downstream_twolinear_v2 | 0.931 | 0.845 |
| CPC | False | True | crossentropy | cpc_downstream_latent_average | 0.888 | 0.791 |
| CPC | False | True | crossentropy | cpc_downstream_latent_maximum | 0.892 | 0.792 |
| CPC | False | True | crossentropy | cpc_downstream_twolinear_v2 | 0.881 | 0.824 |
| CPC | False | True | same | cpc_downstream_latent_average | 0.911 | 0.837 |
| CPC | False | True | same | cpc_downstream_latent_maximum | 0.913 | 0.828 |
| CPC | False | True | same | cpc_downstream_twolinear_v2 | 0.905 | 0.847 |

ROC-AUC scores for CPC models trained with pretraining for 100 epochs and training on the downstream task for 20 epochs with the CPC weights also updated.

5.1.3 CPC Downstream task with all weights updated

Can the performance be improved by updating the weights in the whole architecture?

To answer this question, [Table 7](#) shows yet another way of training the CPC architecture: pretrain the representations for 100 epochs and then, when training on the downstream task, do not freeze the weights in the CPC network layers. We assume, compared to the frozen model, this would lead to the better overall prediction accuracy, because the CPC pretraining is utilized in order to train the autoregressive architecture and then all weights are fine-tuned to optimize the classification loss.

As expected we observe that most models receive a slight performance boost compared to the "frozen" model variants and again the two layer downstream network has better scores overall. For the models however, that are trained with fewer latent vectors (`cpc_intersect_manylatents`) and use "crossentropy" non-strided, we observe that the complete opposite is the case: While the micro ROC-AUC scores are still in line with the other models, the macro average scores are worse than the models that did not have their weights updated. We assume the cpc objective lead to a specific latent representation that were in an opposite direction of the downstream loss function. With more training epochs, we assume that the model will converge towards the results of [Table 3](#), assuming the model can get out of the suboptimal local minimum. We cannot completely rule out overfitting, but we think it is unlikely in this case as the micro average scores are also lower than before.

5.1.4 Low label availability

Under what circumstances is CPC especially efficient to use?

To further investigate the strengths of representation learning we reproduce an experiment from [25] that omits labeled data in the training process to simulate data scarcity. Since the representations learned by CPC only rely on unlabeled data, a clear advantage for all CPC models compared to the baselines, which cannot make use of unlabeled data, can be expected.

We start by utilizing the old train-, validation- and test-splits and make changes to the training split by taking a fixed fraction of each class. For this experiment, class label proportions of $\{0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.05, 0.01, 0.005, 0.001\}$ have been desired to achieve. Since the data can have multiple labels per sample, it is difficult or sometimes even impossible to exactly match the desired fractions for all classes, hence why the class distribution will be approximate. Our algorithm uses a greedy approach where all files are put into “class buckets” for each of the files corresponding class. Afterwards a random file is taken from each class bucket ordered by the buckets size, starting with the smallest one. If a file gets randomly drawn, it is removed from all other buckets. This gets repeated until all the desired fractions for each bucket are met. Code can be found in `experiment/create_fewer_labels_data`.

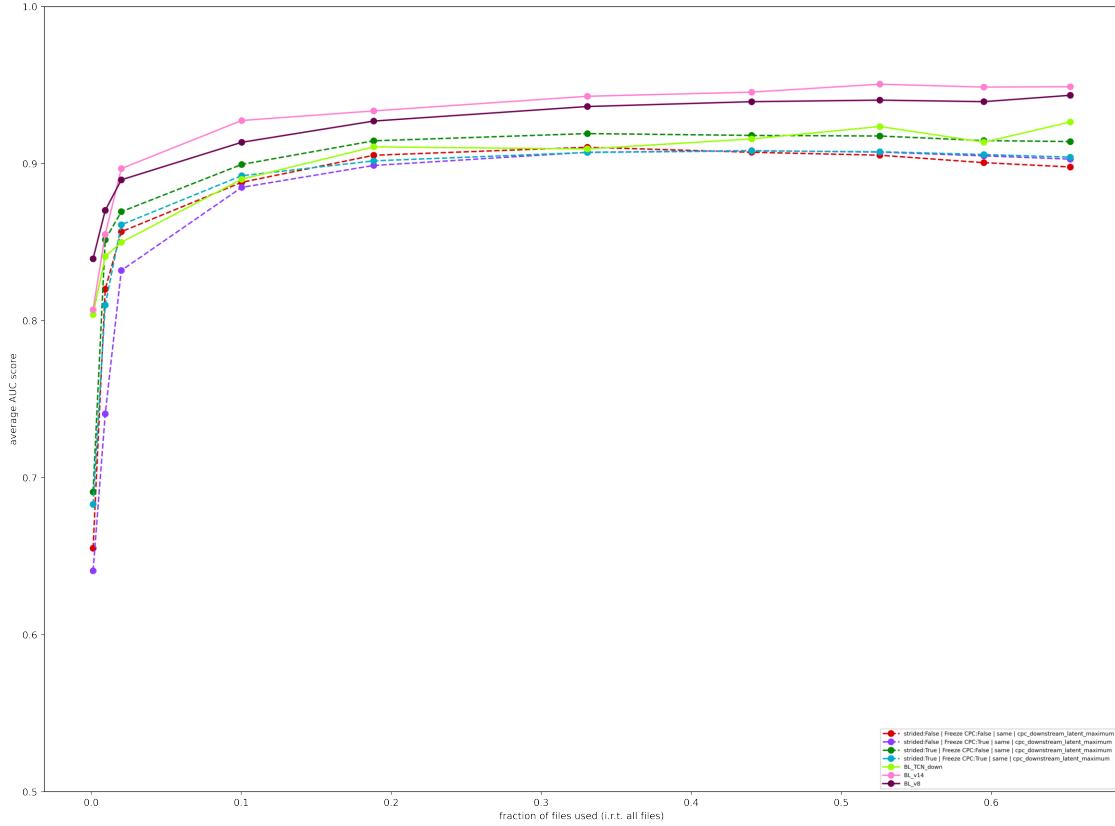
Again, like in the other experiments, we train 120 epochs per baseline model, while CPC trains only for 20 epochs on the labeled data plus 100 epochs on ALL of the original training dataset without the labels.

Finally we measure micro and macro ROC-AUC scores on the same test dataset and plot the results. Instead of the desired proportion we use the actual fraction of total data files used as x-axis, because again, multiple classes are possible per sample, and the actual fraction of data used might differ from the desired proportions. The results for 20 epochs on cpc models and 120 epochs on baseline models can be seen in [Figure 11](#). As one can see more data \equiv better score in general, however the CPC model scores are lower than in baseline models, where few labels are given. The lower CPC ROC-AUC score is contrary to what is to be expected since representation learning makes use of the complete dataset without labels first, hence it should perform considerably better in tasks that provide only few labeled data.

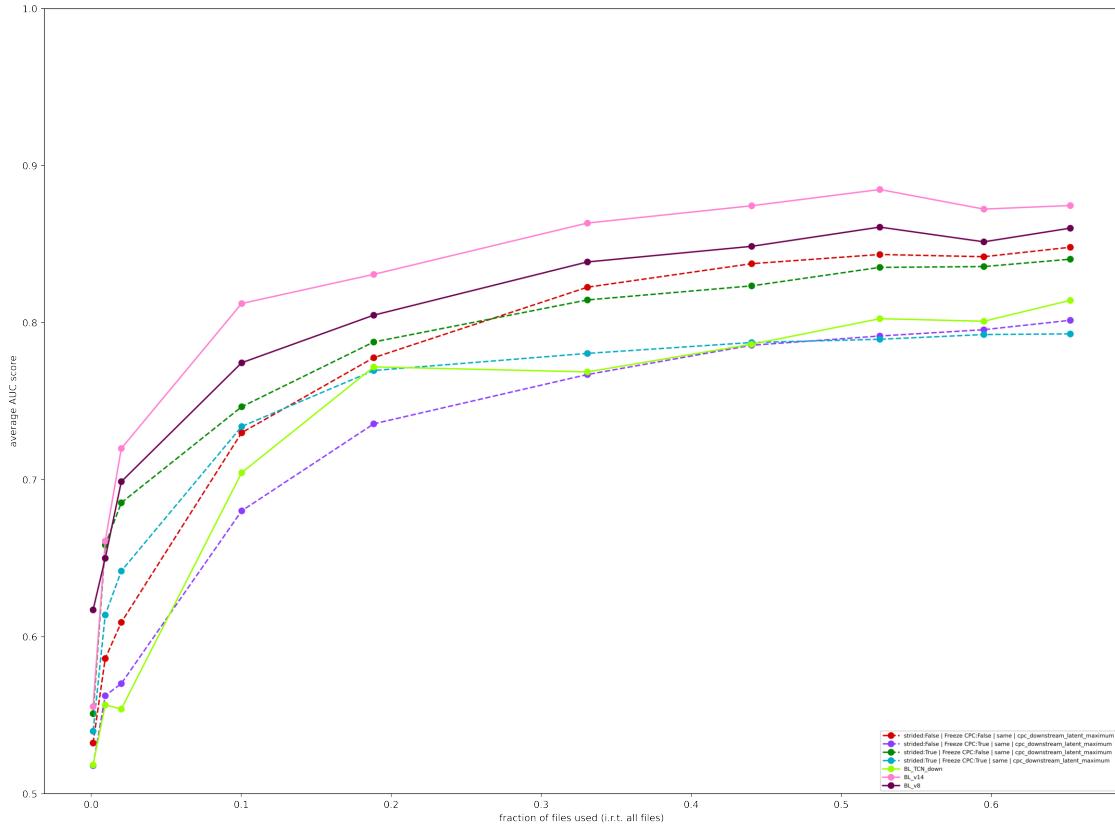
To investigate this counter-intuitive behavior we train selected CPC models for 50 epochs instead of 20. We visualize the scores again in [Figure 12](#) and this time see results closer to what we expected, where CPC models have a slight advantage over baseline models in very low label regimes. Interestingly the performance advantage quickly fades as more labels are given, no matter if trained for 20 or 50 epochs. Our hypothesis is that each class has to be shown to the model a specific number of times at least to successfully fit the model to the classification problem. By using less and less labels, while also limiting the CPC network to training for only 20 epochs, this hypothetical threshold, where the loss is not decreased enough for each individual class, must have been reached. We conclude that CPC is able to slightly outperform non-representation learning networks, where only very few labels are available, but needs sufficient epochs to do so.

Nevertheless we are astounded how well the baselines performed in general and suspect that more tests on different data were few labels are available should be conducted in the future.

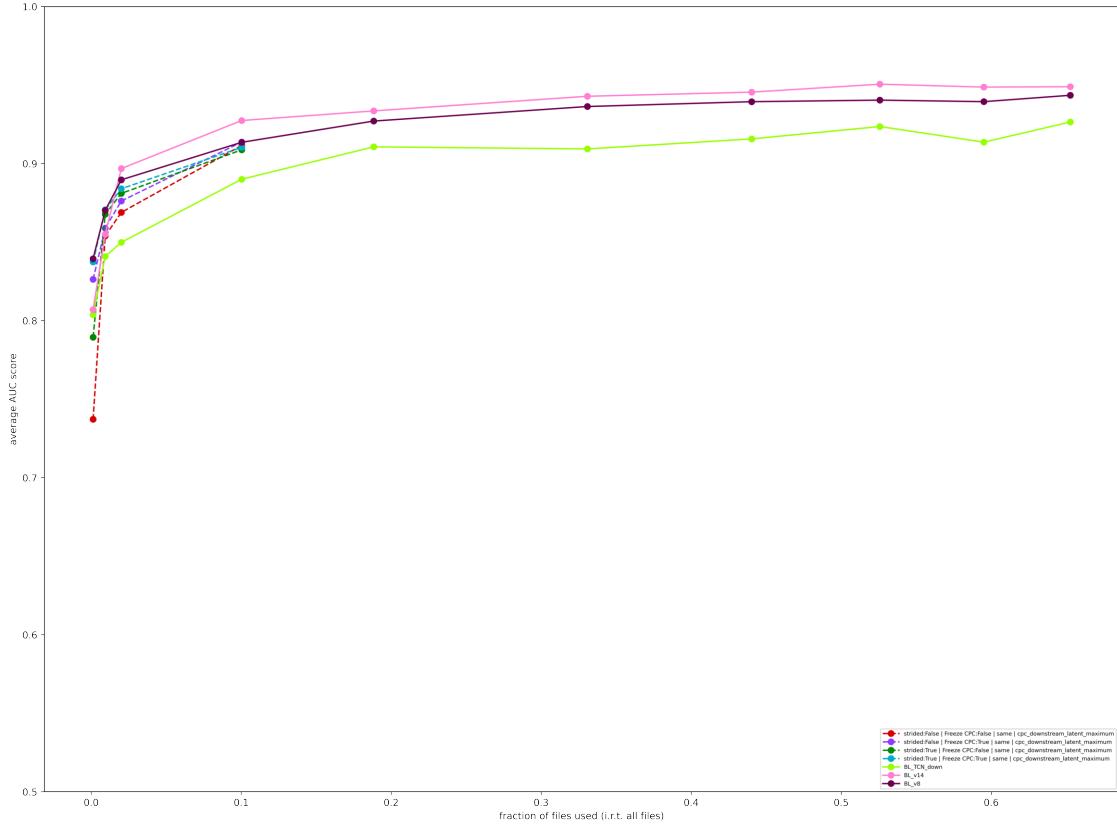
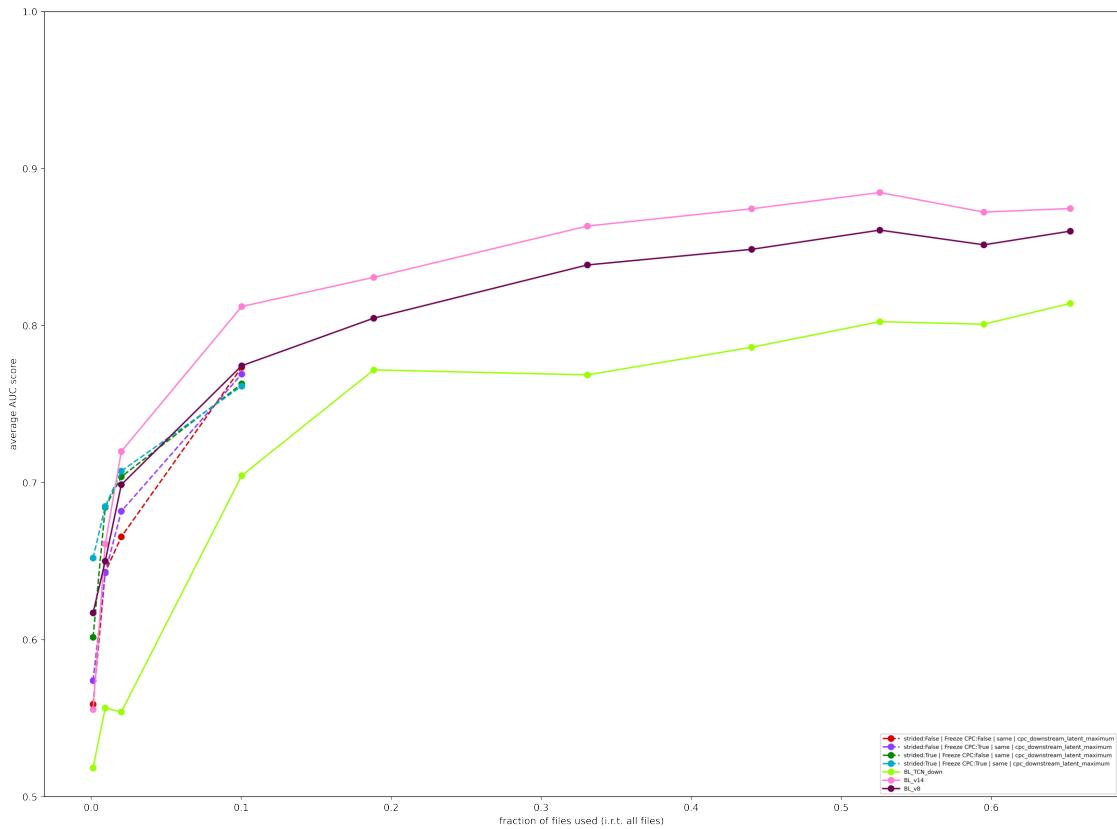
Furthermore this experiment shows one of representation learnings biggest strength:

Figure 11: Test scores for different models trained with a fraction of data labels

(a) Micro average ROC-AUC score. Dashed Line: CPC models, Full Line: Baselines. Baseline-epochs: 120, CPC-epochs: 20



(b) Macro Average ROC-AUC score. Dashed Line: CPC models, Full Line: Baselines. Baseline-epochs: 120, CPC-epochs: 20

Figure 12: Test scores for different models trained with a fraction of data labels (more epochs)**(b)** Macro Average ROC-AUC score. Dashed Line: CPC models, Full Line: Baselines. Baseline-epochs: 120, CPC-epochs: 50**(b)** Macro Average ROC-AUC score. Dashed Line: CPC models, Full Line: Baselines. Baseline-epochs: 120, CPC-epochs: 50

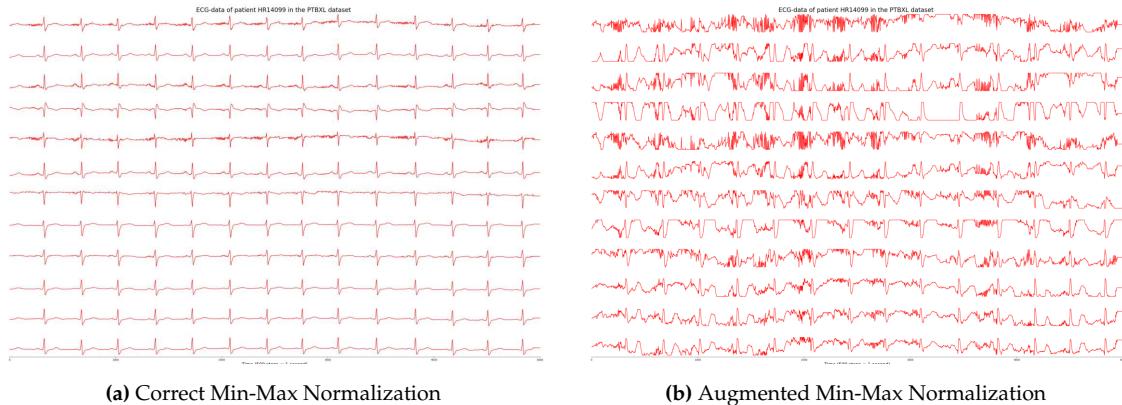
time efficiency. While all the baseline models needed retraining for every single training dataset with 120 epochs each, totaling to more than 24 hours wall clock training time, the CPC model only needed the initial 100 epochs **that is shared amongst all datasets**, plus 20 (or 50) additional training epochs, greatly improving training time by a factor of almost 6 (or >2). Apart from low label availability this of course also the case for different downstream tasks, which is a great opportunity for saving power and time or for neural net enthusiasts that do not have access to capable hardware.

5.1.5 Experiment: Training on Augmented Data

Under what circumstances is CPC especially efficient to use?

We wanted to test whether CPC and the Baseline models are able to correctly predict classes with heavily augmented data, which might be useful for very noisy or even partially broken data. For this we applied the "min-max (augmented)" normalization function on the ecg data and trained the models after. [Figure 13](#) shows the augmented data.

[Figure 13](#): Normalized ECG example from beginning [Figure 1](#)



The following tables show the results for training (and pretraining) multiple baseline and CPC-networks with different training settings on the augmented data. Note that for this section, we used slightly different downstream CPC networks compared to prior results (we refer to [Figure 5](#) for their description).

Table 8a and table 8b show the ROC-AUC scores for all the trained baseline models, while table 9a and table 9b show the AUC scores for all trained cpc models. The left shows the results with class weights utilized in the loss calculation, while the right shows all scores for the same models without the class weights. We make this differentiation, because the macro average scores, which are normally improved by factoring in the class weights during training, were heavily degraded by making use of this method. We suspect the different normalization made the training process more unstable and the added class-weights, which alter the loss directly, contribute to that already unstable training.

Table 8: Baseline ROC-AUC scores

| model | micro | macro | model | micro | macro |
|-----------------------------------|--------------|--------------|-----------------------------------|--------------|--------------|
| BL_v0_1 | 0.579 | 0.485 | BL_v0_1 | 0.835 | 0.773 |
| BL_v0_2 | 0.618 | 0.490 | BL_v0_2 | 0.856 | 0.680 |
| BL_v0_3 | 0.546 | 0.490 | BL_v0_3 | 0.854 | 0.676 |
| BL_v0 | 0.522 | 0.501 | BL_v0 | 0.861 | 0.768 |
| BL_v1 | 0.658 | 0.482 | BL_v1 | <u>0.917</u> | 0.800 |
| BL_v2 | 0.563 | 0.514 | BL_v2 | 0.891 | 0.840 |
| BL_v3 | 0.547 | 0.492 | BL_v3 | 0.852 | 0.813 |
| BL_v4 | 0.404 | 0.504 | BL_v4 | 0.777 | 0.690 |
| BL_v5 | 0.396 | 0.521 | BL_v5 | 0.866 | 0.640 |
| BL_v6 | 0.454 | 0.522 | BL_v6 | 0.667 | 0.436 |
| BL_v7 | 0.870 | 0.730 | BL_v7 | 0.838 | 0.533 |
| BL_v8 | 0.465 | 0.496 | BL_v8 | 0.891 | <u>0.842</u> |
| BL_v9 | 0.439 | 0.511 | BL_v9 | 0.875 | 0.839 |
| BL_v14 | 0.524 | 0.501 | BL_v14 | 0.886 | 0.841 |
| BL_v15 | <u>0.894</u> | <u>0.806</u> | BL_v15 | 0.884 | 0.813 |
| baseline_FCN | 0.484 | 0.505 | baseline_FCN | 0.607 | 0.502 |
| baseline_MLP | 0.500 | 0.500 | baseline_MLP | 0.500 | 0.500 |
| baseline_TCN_block | 0.615 | 0.500 | baseline_TCN_block | 0.628 | 0.500 |
| baseline_TCN_down | 0.874 | 0.747 | baseline_TCN_down | 0.785 | 0.727 |
| baseline_TCN_flatten | 0.841 | 0.547 | baseline_TCN_flatten | 0.855 | 0.584 |
| baseline_TCN_last | 0.846 | 0.567 | baseline_TCN_last | 0.857 | 0.576 |
| baseline_alex_v2 | 0.836 | 0.500 | baseline_alex_v2 | 0.851 | 0.500 |
| baseline_rnn_simplest_lstm | 0.834 | 0.532 | baseline_rnn_simplest_lstm | 0.848 | 0.585 |

(a) Models trained with class weights in the loss function.

(b) Models trained without class weights in the loss function.

Table 9: CPC ROC-AUC scores

| model | micro | macro | model | micro | macro |
|---|-------|-------|---|-------|-------|
| CPC frozen C L m:all cnn | 0.888 | 0.825 | CPC frozen C L m:all cnn | 0.901 | 0.829 |
| CPC frozen C L m:all twolinear | 0.889 | 0.822 | CPC frozen C L m:all twolinear | 0.838 | 0.808 |
| CPC frozen C L m:same cnn | 0.904 | 0.849 | CPC frozen C L m:same cnn | 0.886 | 0.843 |
| CPC frozen C L m:same twolinear | 0.885 | 0.810 | CPC frozen C L m:same twolinear | 0.877 | 0.843 |
| CPC frozen C m:all cnn | 0.890 | 0.813 | CPC frozen C m:all cnn | 0.914 | 0.766 |
| CPC frozen C m:all linear | 0.914 | 0.810 | CPC frozen C m:all linear | 0.921 | 0.812 |
| CPC frozen C m:all twolinear | 0.875 | 0.783 | CPC frozen C m:all twolinear | 0.902 | 0.834 |
| CPC frozen C m:same cnn | 0.918 | 0.822 | CPC frozen C m:same cnn | 0.916 | 0.791 |
| CPC frozen C m:same linear | 0.918 | 0.834 | CPC frozen C m:same linear | 0.911 | 0.795 |
| CPC frozen C m:same twolinear | 0.900 | 0.839 | CPC frozen C m:same twolinear | 0.882 | 0.843 |
| CPC frozen L m:all cnn | 0.883 | 0.832 | CPC frozen L m:all cnn | 0.842 | 0.775 |
| CPC frozen L m:all linear | 0.910 | 0.844 | CPC frozen L m:all linear | 0.769 | 0.540 |
| CPC frozen L m:all twolinear | 0.897 | 0.825 | CPC frozen L m:all twolinear | 0.834 | 0.785 |
| CPC frozen L m:same cnn | 0.914 | 0.769 | CPC frozen L m:same cnn | 0.862 | 0.802 |
| CPC frozen L m:same linear | 0.841 | 0.708 | CPC frozen L m:same linear | 0.781 | 0.595 |
| CPC frozen L m:same twolinear | 0.903 | 0.779 | CPC frozen L m:same twolinear | 0.860 | 0.814 |
| CPC strided frozen C L m:all cnn | 0.909 | 0.780 | CPC strided frozen C L m:all cnn | 0.867 | 0.823 |
| CPC strided frozen C L m:all twolinear | 0.776 | 0.545 | CPC strided frozen C L m:all twolinear | 0.865 | 0.817 |
| CPC strided frozen C L m:same cnn | 0.895 | 0.771 | CPC strided frozen C L m:same cnn | 0.565 | 0.549 |
| CPC strided frozen C L m:same twolinear | 0.751 | 0.591 | CPC strided frozen C L m:same twolinear | 0.870 | 0.814 |
| CPC strided frozen C m:all cnn | 0.921 | 0.792 | CPC strided frozen C m:all cnn | 0.917 | 0.822 |
| CPC strided frozen C m:all linear | 0.921 | 0.812 | CPC strided frozen C m:all linear | 0.921 | 0.792 |
| CPC strided frozen C m:all twolinear | 0.911 | 0.795 | CPC strided frozen C m:all twolinear | 0.914 | 0.842 |
| CPC strided frozen C m:same cnn | 0.903 | 0.754 | CPC strided frozen C m:same cnn | 0.917 | 0.828 |
| CPC strided frozen C m:same linear | 0.892 | 0.841 | CPC strided frozen C m:same linear | 0.909 | 0.779 |
| CPC strided frozen C m:same twolinear | 0.898 | 0.833 | CPC strided frozen C m:same twolinear | 0.897 | 0.838 |
| CPC strided frozen L m:all cnn | 0.898 | 0.836 | CPC strided frozen L m:all cnn | 0.858 | 0.799 |
| CPC strided frozen L m:all linear | 0.884 | 0.806 | CPC strided frozen L m:all linear | 0.902 | 0.775 |
| CPC strided frozen L m:all twolinear | 0.898 | 0.838 | CPC strided frozen L m:all twolinear | 0.822 | 0.795 |
| CPC strided frozen L m:same cnn | 0.876 | 0.824 | CPC strided frozen L m:same cnn | 0.848 | 0.799 |
| CPC strided frozen L m:same linear | 0.910 | 0.852 | CPC strided frozen L m:same linear | 0.895 | 0.781 |
| CPC strided frozen L m:same twolinear | 0.885 | 0.844 | CPC strided frozen L m:same twolinear | 0.828 | 0.794 |

(a) Models trained with class weights in the loss function.

(b) Models trained without class weights in the loss function.

Legend:

m:all All latents are used as negative examples in the cpc loss. **m:same** Latents only from the same timestep are used as negative examples. **linear/cnn/twolinear/latent_maximum** The used downstream architecture. **L** Latent vectors are used for predicting the class labels. **C** Context vector is used for predicting the class labels. **frozen** The weights are not updated in the cpc network layers during downstream task. **strided** Disjoint window data encoder is used instead.

We see that CPC overall performs the best in terms of ROC-AUC score, although labels have not been used as often as in the baseline models. Interesting to see is also that the baseline models cannot cope well with the class weights added to the loss function: Most models cannot confidently surpass even an ROC-AUC area of 0.5, which is the threshold for random guessing. However we admit that the performance drop for the baseline models with the class weights added is suspiciously high and might suggest other factors such as non ideal training settings, that affect weighted models only, are the root of this problem.

Having said this, all CPC models, with only few exceptions, do not see this degradation in performance, meaning they fit all classes in the augmented dataset. As a result we observe better macro average values in some cases with the weights added to the loss function. The overall badly performing baselines and the few CPC models that completely fail to classify the augmented data, suggest that the training for this task is extremely

Table 10: Average CPC ROC-AUC scores, no pretraining

| model | micro | macro | model | micro | macro |
|-------------------------------------|-------|-------|-------------------------------------|-------|-------|
| CPC strided unfrozen C m:all cnn | 0.889 | 0.864 | CPC strided unfrozen C m:all cnn | 0.918 | 0.833 |
| CPC strided unfrozen C m:all linear | 0.808 | 0.482 | CPC strided unfrozen C m:all linear | 0.802 | 0.498 |
| CPC strided unfrozen L m:all cnn | 0.868 | 0.825 | CPC strided unfrozen L m:all cnn | 0.904 | 0.824 |
| CPC strided unfrozen L m:all linear | 0.874 | 0.815 | CPC strided unfrozen L m:all linear | 0.901 | 0.798 |
| CPC unfrozen C m:all cnn | 0.828 | 0.598 | CPC unfrozen C m:all cnn | 0.820 | 0.498 |
| CPC unfrozen C m:all linear | 0.788 | 0.642 | CPC unfrozen C m:all linear | 0.803 | 0.497 |
| CPC unfrozen L m:all cnn | 0.824 | 0.499 | CPC unfrozen L m:all cnn | 0.820 | 0.500 |
| CPC unfrozen L m:all linear | 0.772 | 0.536 | CPC unfrozen L m:all linear | 0.708 | 0.570 |

(a) No class weights were used in loss function.

(b) Class weights were used in loss function.

ROC-AUC scores for CPC models trained without pretraining, trained on the downstream task for 120 epochs. Legend: (like in [Figure 9a](#))

unstable.

To determine whether this difference in performance is caused by the pretraining step or the architecture itself we additionally train a few new CPC models fully supervised without the pretraining step for 120 epochs. Comparing [Figure 10](#) to [Figure 9](#), the non-strided version of the model that used the context, did not work as good as the same model that was pretrained with the CPC-loss, especially the macro average degraded substantially. These results are in line with the baseline results in the stability experiment already observed. We argue that with heavily augmented/noised data, the CPC pretraining step is crucial to correctly learn the latent representations in the CPC architecture.

Interestingly, the “strided” CPC models do not exhibit such a big performance gap compared to the pretrained models and work well for both micro- and macro-average scores. We hypothesize that, even without pretraining, the “strided” architecture is easier to train due to fewer latent vectors being calculated, making it easier for downstream task networks.

[Table 11a](#) and [Table 11b](#) show yet another way of training the CPC architecture: pretrain the representations for 100 epochs and then, when training on the downstream task, do not freeze the weights in the CPC network layers:

Table 11: CPC ROC-AUC scores, with all layers updated in downstream training

| model | micro | macro | model | micro | macro |
|----------------------------------|-------|-------|----------------------------------|-------|-------|
| CPC strided unfrozen C m:all cnn | 0.826 | 0.789 | CPC strided unfrozen C m:all cnn | 0.822 | 0.498 |
| CPC strided unfrozen L m:all cnn | 0.818 | 0.752 | CPC strided unfrozen L m:all cnn | 0.821 | 0.498 |
| CPC unfrozen C m:all cnn | 0.818 | 0.731 | CPC unfrozen C m:all cnn | 0.820 | 0.476 |
| CPC unfrozen L m:all cnn | 0.819 | 0.499 | CPC unfrozen L m:all cnn | 0.821 | 0.520 |

(a) No class-weights are used.

(b) Class-weights are used.

ROC-AUC scores for CPC models trained with pretraining for 100 epochs and training on the downstream task for 20 epochs with the CPC weights also updated. Legend: (like in [Figure 9a](#))

At first one would assume this would lead to the best overall prediction accuracy, be-

cause the CPC pretraining is utilized in order to train the autoregressive architecture and then all weights are fine-tuned to optimize the classification loss. Yet we observe the opposite: while the micro-average ROC-AUC scores are still in line with the other models, the macro-average scores are the overall worst, at least in the training with added class weights. We assume the cpc objective lead to a specific latent representation that were in an opposite direction of the downstream loss function and acted like a bad weight initialization. With more training epochs, we assume that the model will converge towards the results of [Figure 10b](#), assuming the model can get out of the suboptimal local minimum. A different explanation is that the added flexibility allowed the model to more easily overfit on the training data.

Although the CPC models outperformed all baseline architectures ([Figure 8](#) vs. [Figure 9](#)), which suggests that the pretraining step seems to help train the models, we cannot confidently say that pretraining is always helpful for augmented data because the strided CPC networks still perform well in a setting with no pretraining at all (see [Figure 10](#)), while some CPC models could not be successfully trained even with pretraining. Since strided and non-strided models only vary in architecture and not how they are trained, it is possible that difference in the architecture lead to good performance and not the pretraining per se. Nevertheless we suspect that the CPC pretraining leads more easily to overall good representations if the data is difficult to classify, and that too much freedom during the difficult downstream task leads to overfitting or unstable training, which can be followed from [Figure 9](#) vs. [Figure 11](#). We suspect the strided models get around overfitting because of easier model convergence. In summary we argue that the stability experiment results are inconclusive but they hint at that CPC models and their pretraining make finding the latent features easier, in unstable training environments.

5.1.6 Additional changes

Can the original architecture be improved?

Next we take a look at the additional changes we made to the CPC architecture to allow a more flexible use or increase performance. In some cases we relaxed our downstream epoch limit from 20 to 40, because we are not directly compare to baseline networks and in order to more thoroughly test an architectures capabilities. We note that this increases the difficulty of comparing those models to the ones that only trained for 20 epochs above, but if the performance is still lower when trained for 40 epochs, we are more certain that this specific change in the architecture was a step in the wrong direction.

Additionally please note that for certain categories like "Downstream Model", we will only display the models with the highest macro average scores. For example a model that has been trained with `cpc_downstream_latent_average` vs `cpc_downstream_latent_maximum` with otherwise equal settings, will only show the better model with the associated scores. Also all models use `cpc_intersect_manylatents` (see [Paragraph 2.2.1.2](#)) and — besides their one changed module — still the standard encoder-, context- and predictor network `cpc_encoder_v0`, `cpc_autoregressive_v0` and `cpc_predictor_v0`. This restriction is necessary since some models require to be trained with `cpc_intersect_manylatents` in order to function.

Table 12: CPC ROC-AUC scores, 40 Downstream Epochs

| model | Freeze CPC | strided | CPC Sampling Mode | Downstream Epochs | Downstream Model | micro | macro |
|-------|------------|---------|-------------------|-------------------|-------------------------------|--------------|--------------|
| CPC | False | False | crossentropy | 40.000 | cpc_downstream_latent_average | 0.930 | 0.842 |
| CPC | False | False | crossentropy | 40.000 | cpc_downstream_latent_maximum | 0.925 | 0.830 |
| CPC | False | False | crossentropy | 40.000 | cpc_downstream_twolinear_v2 | 0.911 | 0.830 |
| CPC | False | True | crossentropy | 40.000 | cpc_downstream_latent_average | 0.905 | 0.768 |
| CPC | False | True | crossentropy | 40.000 | cpc_downstream_latent_maximum | 0.878 | 0.802 |
| CPC | False | True | crossentropy | 40.000 | cpc_downstream_twolinear_v2 | 0.915 | 0.801 |
| CPC | True | False | crossentropy | 40.000 | cpc_downstream_latent_average | 0.923 | 0.824 |
| CPC | True | False | crossentropy | 40.000 | cpc_downstream_latent_maximum | 0.926 | 0.827 |
| CPC | True | False | crossentropy | 40.000 | cpc_downstream_twolinear_v2 | 0.929 | 0.838 |
| CPC | True | True | crossentropy | 40.000 | cpc_downstream_latent_average | 0.908 | 0.772 |
| CPC | True | True | crossentropy | 40.000 | cpc_downstream_latent_maximum | 0.915 | 0.804 |
| CPC | True | True | crossentropy | 40.000 | cpc_downstream_twolinear_v2 | 0.917 | 0.801 |

ROC-AUC scores for CPC models trained with pretraining for 100 epochs and training on the downstream task for 40 epochs.

Table 13: CPC ROC-AUC scores: normalized latents

| model | Freeze CPC | strided | normalizes latents | CPC Sampling Mode | Downstream Epochs | Downstream Model | micro | macro |
|-------|------------|---------|--------------------|-------------------|-------------------|-------------------------------|--------------|--------------|
| CPC | False | False | True | crossentropy | 20.000 | cpc_downstream_twolinear_v2 | 0.873 | 0.659 |
| CPC | False | False | True | crossentropy | 40.000 | cpc_downstream_latent_average | 0.905 | 0.774 |
| CPC | False | True | True | crossentropy | 20.000 | cpc_downstream_twolinear_v2 | 0.869 | 0.750 |
| CPC | False | True | True | crossentropy | 40.000 | cpc_downstream_latent_maximum | 0.916 | 0.799 |
| CPC | True | False | True | crossentropy | 20.000 | cpc_downstream_twolinear_v2 | 0.845 | 0.690 |
| CPC | True | False | True | crossentropy | 40.000 | cpc_downstream_twolinear_v2 | 0.887 | 0.697 |
| CPC | True | True | True | crossentropy | 20.000 | cpc_downstream_twolinear_v2 | 0.879 | 0.754 |
| CPC | True | True | True | crossentropy | 40.000 | cpc_downstream_twolinear_v2 | 0.902 | 0.753 |

ROC-AUC scores for CPC models trained with pretraining for 100 epochs and training on the downstream task for 20 or 40 epochs (both frozen/updated). Latent vectors are always normalized

Table 12 can be used as a reference for what is possible using all standard modules with 40 epochs (with `cpc_intersect_manylatents`).

5.1.6.1 Normalized Latent Vectors **Table 13** shows the results for models that were trained with normalized latent vectors (see [Paragraph 2.2.1.3](#) for details). We see that even with more epochs the scores are overall lower, for both micro and macro average, indicating this change most likely is not beneficial. We suspect that normalizing the vectors restricted the model too much.

5.1.6.2 Encoder like Baseline_v8 In **Table 14** we see the results for models that were trained with `cpc_encoder_likev8`, which is a clone of the baseline model `BL_v8`. `cpc_encoder_likev8` outputs 147 latent vectors as opposed to 26 like the standard architecture. We report lower scores overall.

Nevertheless we want to mention that generating more latent vectors is a trait which is useful in some tasks, as for example later on in [Subsubsection 5.2.6](#).

5.1.6.3 Hidden State Context Network We look at the results of our networks where `cpc_autoregressive_hidden` is used, which uses the hidden state as a context for the next step latent predictions. The model that was trained for 40 epochs with all weights updated yielded the best results, even surpassing the macro-average score from the best

5 RESULTS

Table 14: CPC ROC-AUC scores: cpc_encoder_likev8

| model | Freeze CPC | strided | CPC Sampling Mode | Encoder | Downstream Epochs | Downstream Model | micro | macro |
|-------|------------|---------|-------------------|--------------------|-------------------|-----------------------------|--------------|--------------|
| CPC | False | False | crossentropy | cpc_encoder_likev8 | 20.00 | cpc_downstream_twolinear_v2 | 0.853 | 0.771 |
| CPC | True | False | crossentropy | cpc_encoder_likev8 | 20.00 | cpc_downstream_twolinear_v2 | 0.873 | 0.787 |

ROC-AUC scores for CPC models trained with pretraining for 100 epochs and training on the downstream task for 20 or 40 epochs (both frozen/updated). `cpc_encoder_likev8` is always used as an encoder

Table 15: CPC ROC-AUC scores: cpc_autoregressive_hidden

| model | Freeze CPC | strided | CPC Sampling Mode | Autoregressive | Downstream Epochs | Downstream Model | micro | macro |
|-------|------------|---------|-------------------|---------------------------|-------------------|-------------------------------|--------------|--------------|
| CPC | False | False | crossentropy | cpc_autoregressive_hidden | 40.00 | cpc_downstream_latent_maximum | 0.927 | 0.851 |
| CPC | True | False | crossentropy | cpc_autoregressive_hidden | 40.00 | cpc_downstream_twolinear_v2 | 0.885 | 0.797 |

ROC-AUC scores for CPC models trained with pretraining for 100 epochs and training on the downstream task for 20 or 40 epochs (both frozen/updated). `cpc_autoregressive_hidden` is always used as context network

model in the comparison [Table 12](#). Interestingly the score got degraded for the model with the weights frozen, making this context network useful only situational.

5.1.6.4 No Context Network For the network that did not use the context for predicting future latents, and instead a TCN trained to predict next-step latents directly, the results are overall a bit lower. Despite not displayed here, we mention that the CPC accuracy during pretraining was higher than for any other model, with exemption of models trained with the ‘same’ sampling method and all standard settings, indicating that the representations were more easily distinguishable for the different entries in the batch. Although the results were slightly less accurate, we note that it is still interesting to see

Table 16: CPC ROC-AUC scores: cpc_predictor_nocontext

| model | Freeze CPC | strided | CPC Sampling Mode | Predictor | Downstream Epochs | Downstream Model | micro | macro |
|-------|------------|---------|------------------------|-------------------------|-------------------|-------------------------------|--------------|--------------|
| CPC | False | False | crossentropy-nocontext | cpc_predictor_nocontext | 20.00 | cpc_downstream_twolinear_v2 | 0.873 | 0.762 |
| CPC | False | False | crossentropy-nocontext | cpc_predictor_nocontext | 40.00 | cpc_downstream_latent_maximum | 0.913 | 0.831 |
| CPC | True | False | crossentropy-nocontext | cpc_predictor_nocontext | 20.00 | cpc_downstream_twolinear_v2 | 0.881 | 0.764 |
| CPC | True | False | crossentropy-nocontext | cpc_predictor_nocontext | 40.00 | cpc_downstream_twolinear_v2 | 0.911 | 0.803 |

ROC-AUC scores for CPC models trained with pretraining for 100 epochs and training on the downstream task for 20 or 40 epochs (both frozen/updated). `cpc_predictor_nocontext` is always used as the latent predictor

that the context network + single linear layers for predicting future latents is not a given and can be exchanged for different architectures.

5.2 Qualitative

5.2.1 Finding the best Hyperparameters with Parallel Coordinates

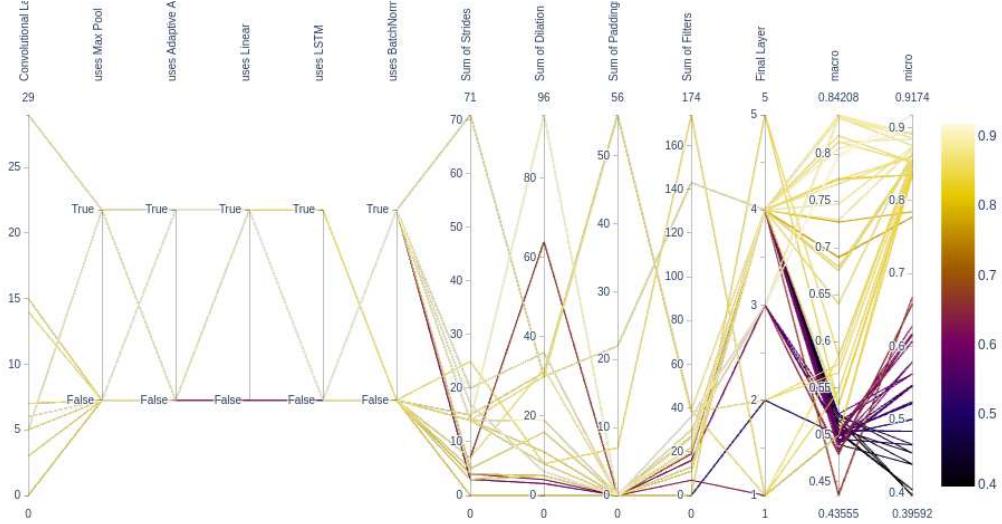
What are the best settings/hyperparameters for CPC?

To better evaluate what model specific baseline hyperparameters lead to a good test performance we make use of a so called “Parallel Coordinates” plot which can visualize

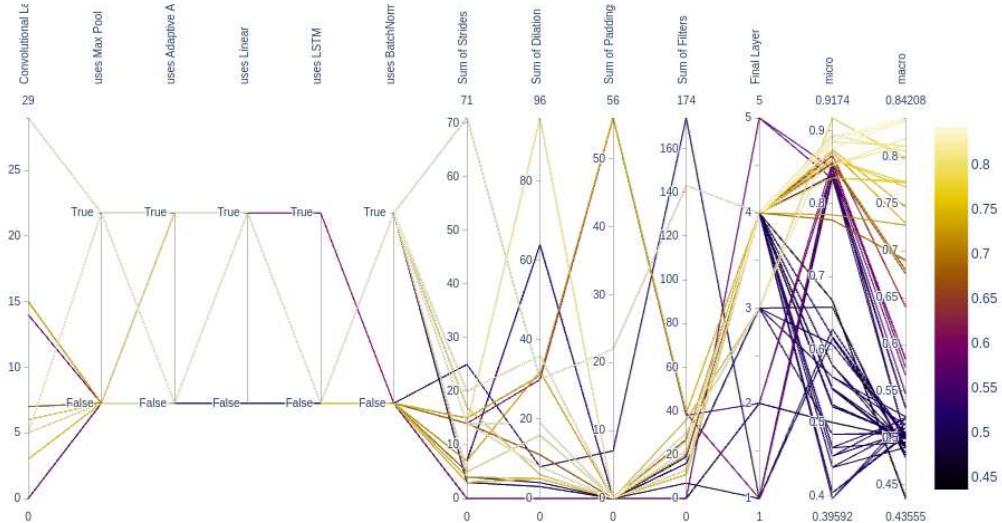
high-dimensional datasets. It consists of one x-axis that uses a categorical label instead of a number at each step with no specific order and multiple differently ranged y-axes, one for each categorical label. Every datasample is drawn as line in a color specific to one of the datasamples attributes. In short, parallel coordinate plots visualize tables by using the column-headers as x-axis and draw each row as a line with a color specific to one selected column.

Since we want to figure out what hyper parameters influence our micro and macro ROC-AUC score in a positive way, both will be used individually as color in the plot, while some of the hyperparameters will be shown on the x-axis. We summarize list-like hyperparameters such as multiple strides or kernel-sizes as a sum in order to visualize them. For strides especially, a product could be used instead to deliver more accurate results, which is due to the fact that strides have a multiplicative impact on input data, while kernel-sizes are closer to additive. The widely different values however made the plot unreadable with big products in our case. The resulting plot can be seen in [Figure 14a](#) and [Figure 14b](#). While it is difficult to judge most hyperparameters with confidence based on this plot, we can see that the final layer choice seems to have a big impact on performance in our baseline models. There, approach 4 — which downsamples the models output with a convolutional layer before feeding it to the final linear layer — seems to achieve the overall best performance.

Figure 14: Parallel Coordinates Plot for all trained baseline networks from [Figure 8a](#).



(a) The color is the **micro** ROC-AUC score



(b) The color is the **macro** ROC-AUC score

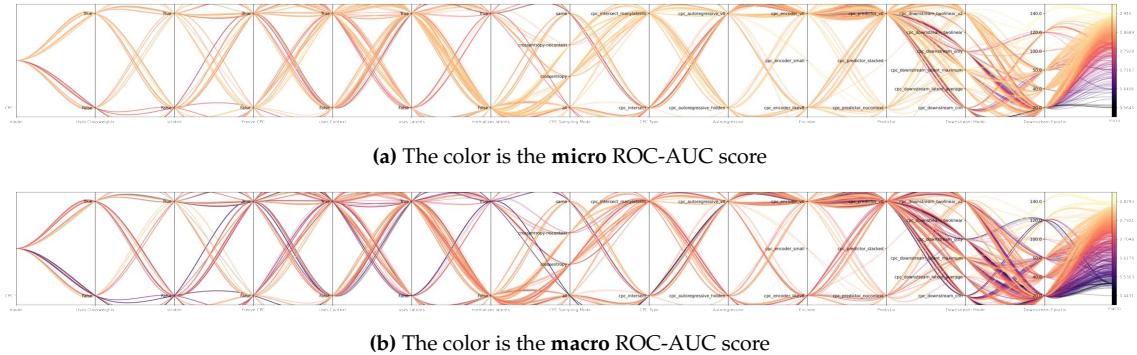
For the trained CPC models we use a different implementation obtained from Github⁵, where the lines are curved to better visualize binary values. [Figure 15b](#) and [Figure 15a](#) show the parallel coordinate plots for all of our trained CPC models. This also includes experiments like "low label availability".

Although more CPC models have been evaluated in the CPC parallel coordinates plots, they are less conclusive regarding model hyperparameters and performance relationship,

⁵https://github.com/jraine/parallel-coordinates-plot-dataframe/blob/master/parallel_plot.py.

because there are both good and bad AUC-scores for most hyperparameters. An exemption of this might be normalizing the latent vectors during training, which results in a worse macro AUC score. Additionally it seems to be beneficial to utilize the context vector, which can be again seen in the macro AUC score.

Figure 15: Parallel Coordinates Plot for all trained baseline networks from [Figure 8a](#).



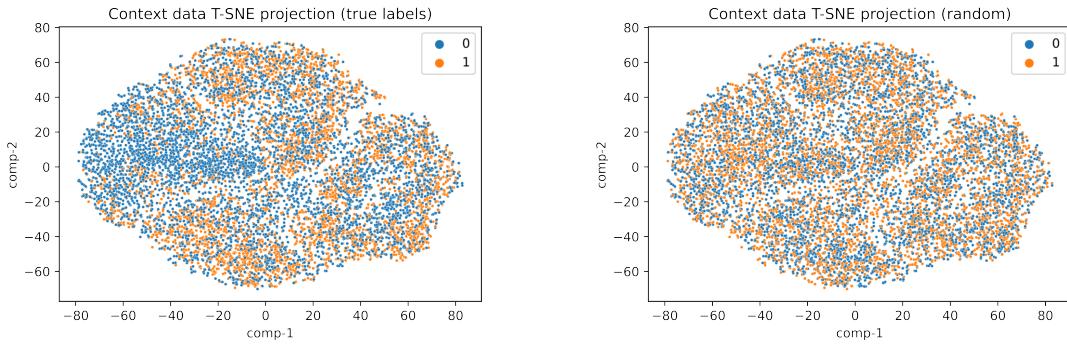
5.2.2 Latent/Context t-SNE representations

Does CPC learn visually verifiable useful latent representations?

In an attempt to visually verify the usefulness of the learned latent data representations we separately perform a dimensionality reduction from the 128 valued latent vectors and the 256 valued context vectors into a two dimensional space. This is done by taking one of the above trained CPC networks (with the settings `use_weights`, `strided`, `unfrozen`, `cpc_downstream_cnn` in this case) and saving all latent and context vectors for the whole dataset. Afterwards we use t-SNE [35] to embed the vectors into two dimensions, the results are visualized in a scatter plot where the embedded dimensions are used as x- and y-axis. Each sample is colored according to its label in the data. Since the data has multiple possible labels per sample, which makes distinct class coloring difficult, we opted to use binary coloring — if class x is present in the data assign 1, otherwise assign 0 to the datapoint. This is done for all classes.

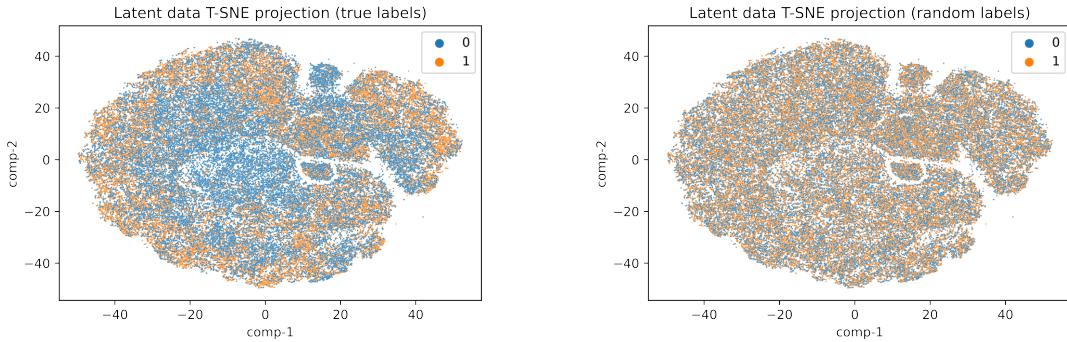
In order to be able to better judge if the colored embeddings have meaning, we added a scatter plot with randomized labels as well. An example for this can be seen in [Figure 16](#) which shows all context vector embeddings, with the labels for class 426783006 (normal sinus rhythm). Although the data points were not fully separated, a clear distinction to the random plot can still be made (see mostly blue big area on the left). The same goes for [Figure 17](#) which shows all latent vector embeddings instead. There the positive labels seem to form a border around the data, the negative labels are mostly centered in the middle.

Figure 16: Two dimensional t-SNE embeddings of all context vectors



t-SNE embedding of all context vectors (one for each patient). Left: Data points are colorized according to their true labels. Right: The colors are randomly assigned.

Figure 17: Two dimensional t-SNE embeddings of all latent vectors



t-SNE embeddings of all latent vectors (nine vectors per patient). Left: Data points are colorized according to their true labels. Right: The colors are randomly assigned.

[Figure 18](#) and [Figure 19](#) show the embeddings with the respective point coloring for 64 of 67 classes, denoted in the upper right of each scatter plot. The plots are ordered by their ROC-AUC-score from top-left to bottom-right, each score visualized as background color. Especially for smaller classes (fewer yellow dots) it is difficult to judge how well the data is clustered, but there are cases where clusters are clearly distinguishable. Take for example classes 427084000 (sinus tachycardia) and 426177001 (sinus bradycardia) in [Figure 18](#), which are on the opposite site in the embedding and also have a high ROC-AUC-score. Interestingly sinus tachycardia means that the heart beat is faster than usual (over 100bpm [36]) and sinus bradycardia that the heart beats slower than usual (under 60bpm [37]), which means the classes are in fact conflicting and very unlikely to occur together as final diagnosis on the same patient. The same classes are also fairly spatially divided with the latent embeddings [Figure 19](#), however not as clearly as with the context embeddings. This might be due to the fact that a patient diagnosed with sinus tachycardia might still have a few slow beats or vice versa with sinus bradycardia. Also since the tested model relies on the context vector for the final prediction, the classes need to

be more distinguishable in the context vector space than the latent vector space, which could explain the better spatial division. Additionally since nine times more latent data points are generated, the two dimensional embedding could be harder to find for t-SNE.

In contrast to spatially divided clusters there also seem to be class clusters which are more or less equivalent. In [Figure 19](#) take for example classes 6374002 (Bundle Branch Block) and 164909002 (Left Bundle Branch Block), which look very similar apart of their sizes, and are also very similar from a medical standpoint.

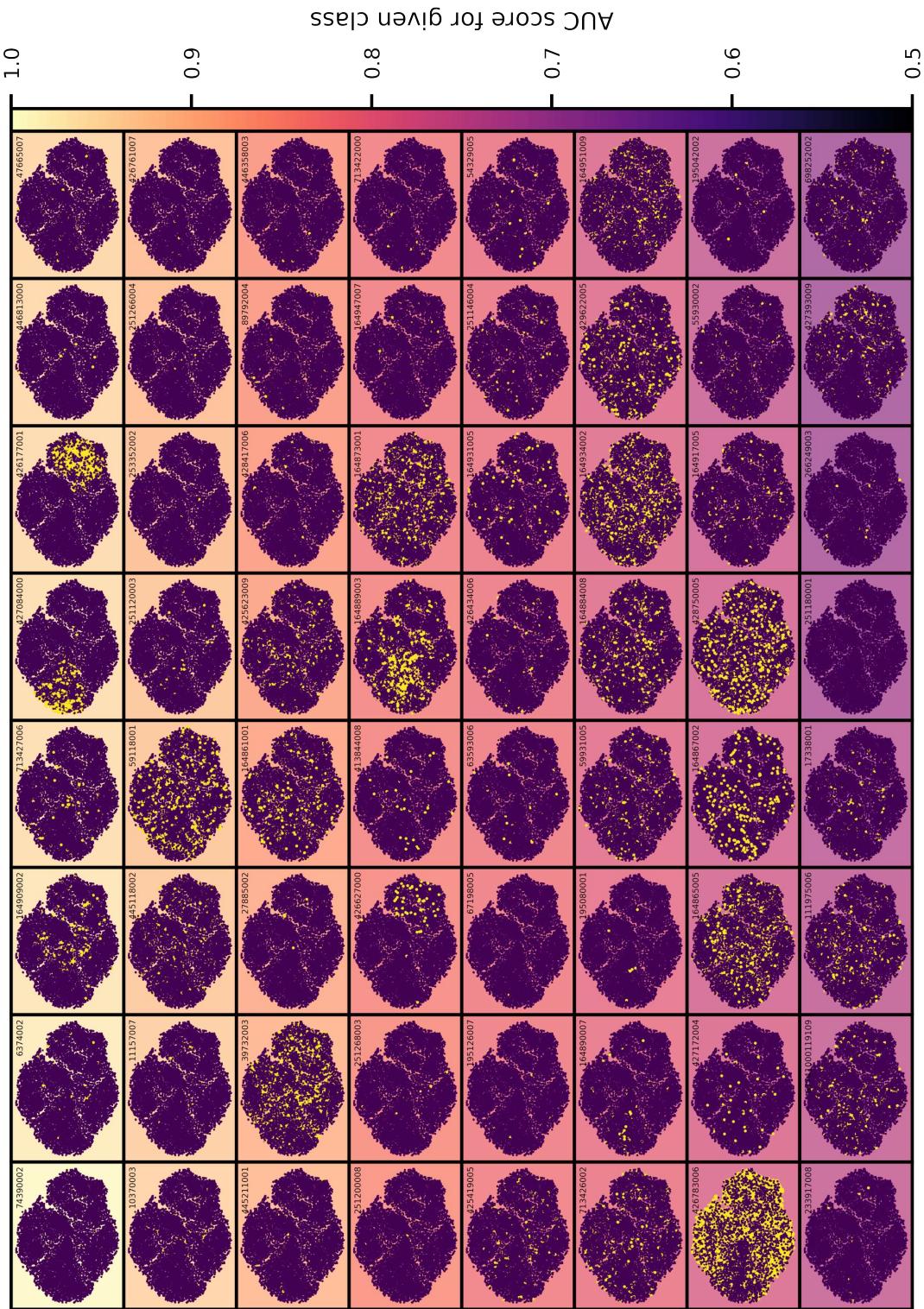


Figure 18: t-SNE embedding of all context vectors (one for each patient) for 64 of 67 classes. Legend: number in top right denotes class name, yellow dot means class name is part of the recording's labels. Background color is the AUC score per class, sorted descending

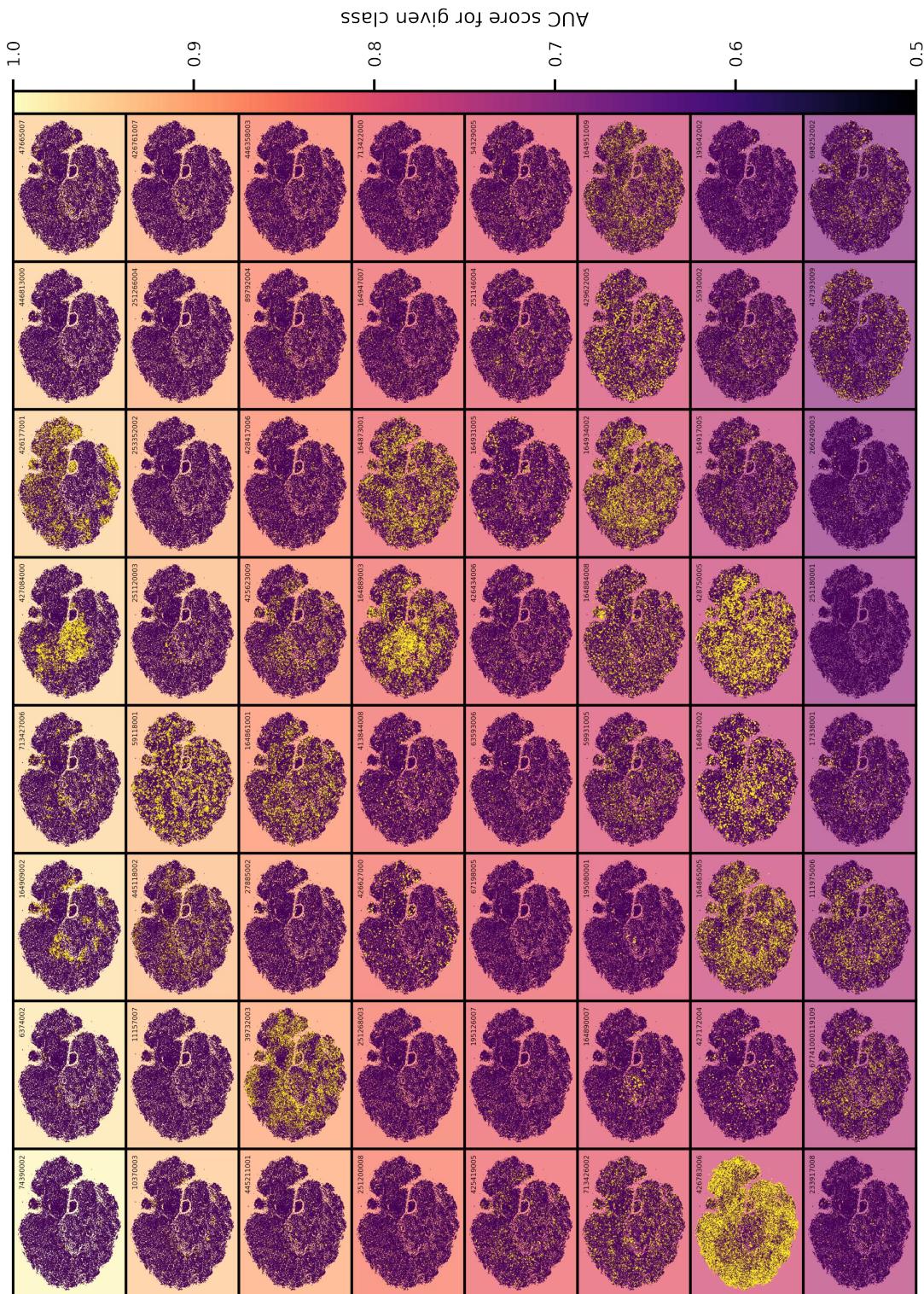


Figure 19: t-SNE embedding of all latent vectors (nine for each patient) for 64 of 67 classes. Legend: number in top right denotes class name, yellow dot means class name is part of the recording's labels. Background color is the AUC score per class, sorted descending



Figure 20: t-SNE embedding of all context vectors (one for each patient) for 64 of 67 classes. Context obtained with frozen model. Legend: number in top right denotes class name, yellow dot means class name is part of the recording’s labels. Background color is the AUC score per class, sorted descending



Figure 21: t-SNE embedding of all latent vectors (26 for each patient) for 64 of 67 classes. Latents obtained with frozen model. Legend: number in top right denotes class name, yellow dot means class name is part of the recording's labels. Background color is the AUC score per class, sorted descending.

We compare prior results of a strided model where all weights are updated, with the t-SNE embeddings of a non-strided network that was trained with frozen weights. As a result the learned latent representations obtained by only the CPC loss are formed completely independent of labeled data. The t-SNE embeddings will show how/if the latents are dividable into clusters and if the highlighted classes are comparably separable.

[Figure 20](#) shows the t-SNE embedding of the context vectors, while [Figure 21](#) shows the embedding for all latent vectors. Comparing the different model's context plots, we notice that the true samples of that specific class (yellow dots) seem to be less grouped and more spread throughout the plot. The distinction we made in the first plot between class sinus tachycardia and sinus bradycardia cannot be made anymore. We suspect that the training with updated weights is important for building a context that is focused on predicting class labels, rather than the next latent steps.

On the other side, for the latent embedding we observe that there seems to be a way finer clustering in general, which are however not restricted to class labels. We argue that the latent representations formed in the model with frozen weights have distinct properties partly independent of their class label in the data, which is not surprising as the CPC loss is minimized without labels. The class labels seem to appear predominantly in few selected clusters which suggests that each class has its own defining properties.

In conclusion the context vectors of the strided model with updated weights seems to be more suited for classification, while the latent representations of the frozen non-strided model appear to be of higher quality and more capable of identifying unique data properties, which are focused on specific classes only sporadically. This suggests that CPC finds general latent representations which encode the data's properties and that during the downstream task those representations get fine tuned for classification. In the case of the strided model those representations were less separable in a lower dimension, where it's not entirely clear if these properties got imperfectly "mixed" into classes, or if the strided model's latent representations have less distinct properties in general.

5.2.3 Transforming Predictions Scores with Model Thresholds

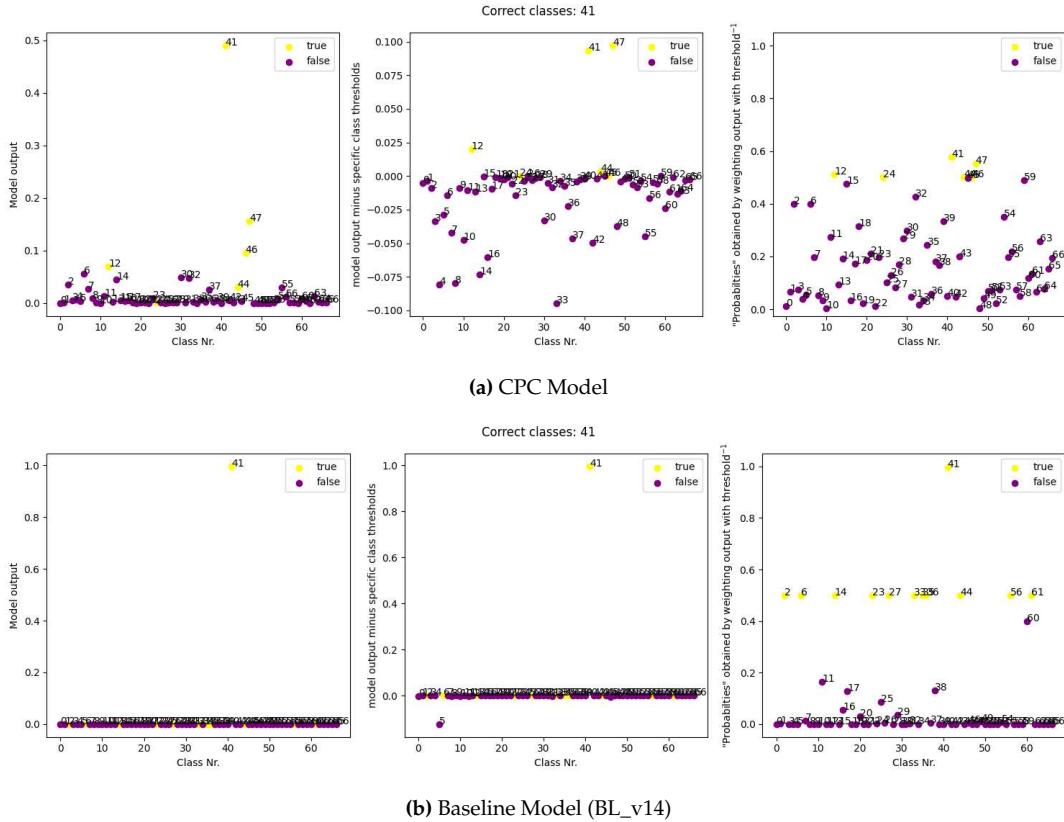
After predictions for the ECG-data have been made it is important to visualize them correctly, to give an idea of what class labels are considered true. Let p be a prediction score vector for n classes $p = (s_1, \dots, s_n)$ with $\forall i \leq n : s_i \in [0, 1]$ and t be a threshold score vector for n classes $t = (t_1, \dots, t_n)$ with $\forall i \leq n : t_i \in [0, 1]$ that decides if class i is considered true ([Equation 22](#)). We define the reweighted probability scores $p_r = (r_1, \dots, r_i, \dots, r_n)$ as:

$$r_i := \begin{cases} (\frac{s_i - t_i}{t_i} + 1)/2, & \text{if } s_i < t_i \\ (\frac{s_i - t_i}{1-t_i} + 1)/2, & \text{if } s_i \geq t_i \end{cases} \quad (25)$$

The reweighting function can be explained as splitting scores into positive and negative predictions where we stretch the space below the threshold $[0, t_i[$ to $[0, 0.5[$ and the space above and equal to threshold t_i : $[t_i, 1] \mapsto [0.5, 1]$. This has the advantage that predictions mean the same across all classes and models: We moved the decision boundary from $\forall i, t_i \mapsto 0.5$. In [Figure 22](#) and [Figure 23](#) we see examples for the process of transforming model outputs to reweighted probabilities.

5.2.4 Hand-picked Samples: Prediction Score Scatterplots

Figure 22: Scatterplots showing from left to right: exact model predictions, model predictions minus specific class thresholds, reweighted prediction probabilities

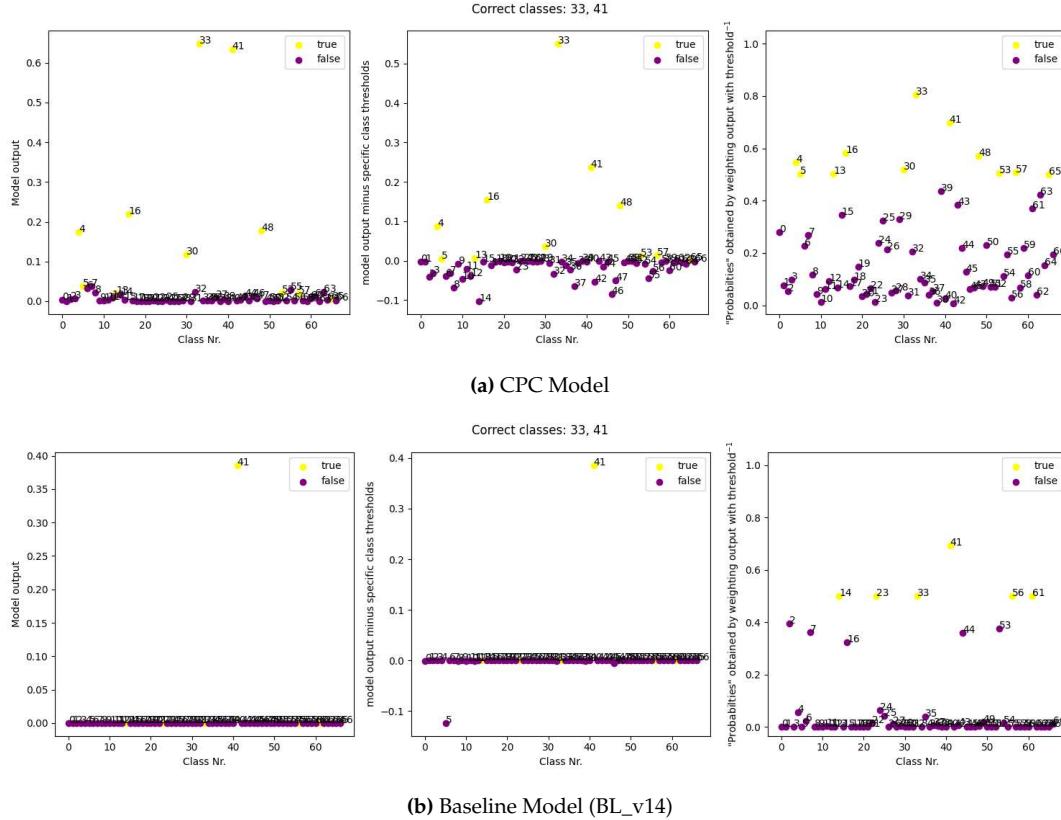


Looking at the leftmost plot in [Figure 22](#), we see why reweighting probabilities is important for readability: the unchanged model outputs for class 41 is considered true by the model but has a lower probability output than for example class 6 which is considered false. We subtract the class specific thresholds and obtain a more readable output. Last but not least we transform the differences into real probabilities and are even able to judge how sure a model is about specific predictions: the closer to 0 or 1 \Rightarrow prediction certainty high.

For the BL_v14 model predictions we observe that only the correct class 41 has a high probability of being true while the other classes falsely classified as true barely go above 0.5. The CPC model predictions are less confident in general, however class 41 has still the highest probability. At this place we also want to point out a flaw of scores that are based on binary predictions: In this specific case CPC would have the higher score, as fewer classes are wrongly classified as true. Nevertheless we make the argument that in this case the predictions of the BL_v14 are higher quality because of the models high certainty prediction of the correct class.

[Figure 23](#) shows yet another prediction example — this time we would argue that the

Figure 23: Scatterplots showing from left to right: exact model predictions, model predictions minus specific class thresholds, reweighted prediction probabilities

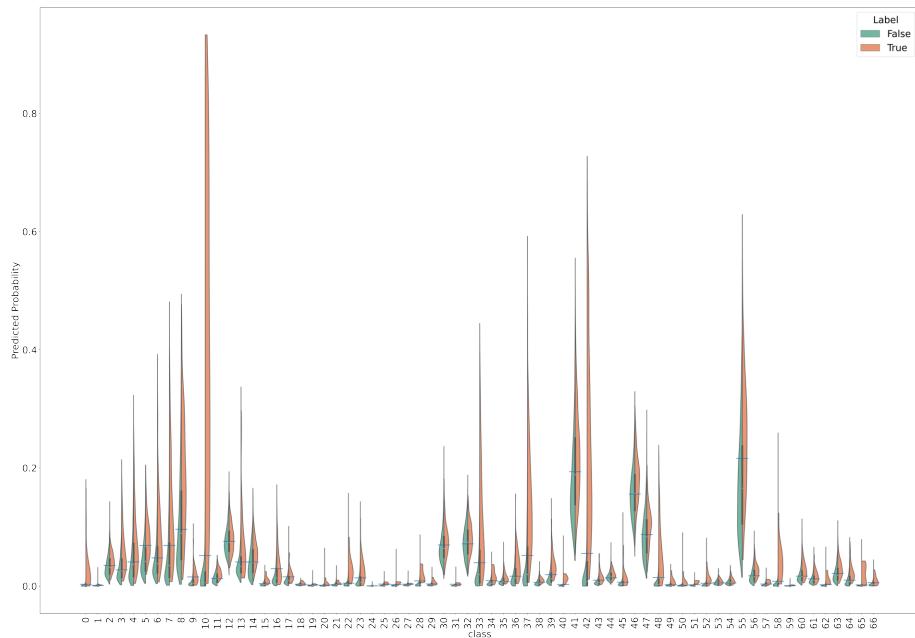


CPC predictions are slightly better, at least concerning classes labeled "true". Although more classes are classified as true erroneously, both 33 and 41 have been predicted with the highest certainty. Compare this to The baseline predictions where 41 is fairly certain, but 33 is just as unlikely as erroneously classified classes.

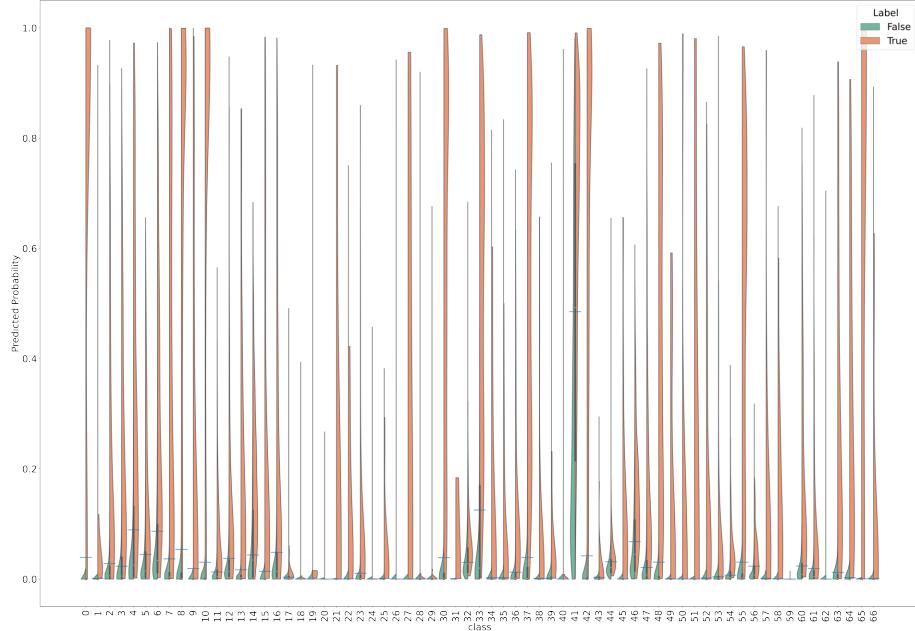
5.2.5 All Samples: Violinplots

Below in [Figure 24](#) we include the raw prediction score data visualized as a violinplot, which show the prediction distributions for all classes. We opted to visualize both True and False prediction distributions with the same width, which removes the information about the group sizes, but increases readability. Noticeable is that even without the reweighting of model predictions, probabilities in the baseline model are more evenly distributed within $[0, 1]$, indicating that the sigmoid was able to more precisely predict the extrema 0 and 1. This is not implicitly better since thresholds get selected to divide the probabilities into true and false anyways and as long as the false and true label predictions are more or less separate from each other, results are equally correct.

Figure 24: Scatterplot showing all exact model predictions, divided into groups depending on their ground truth label.



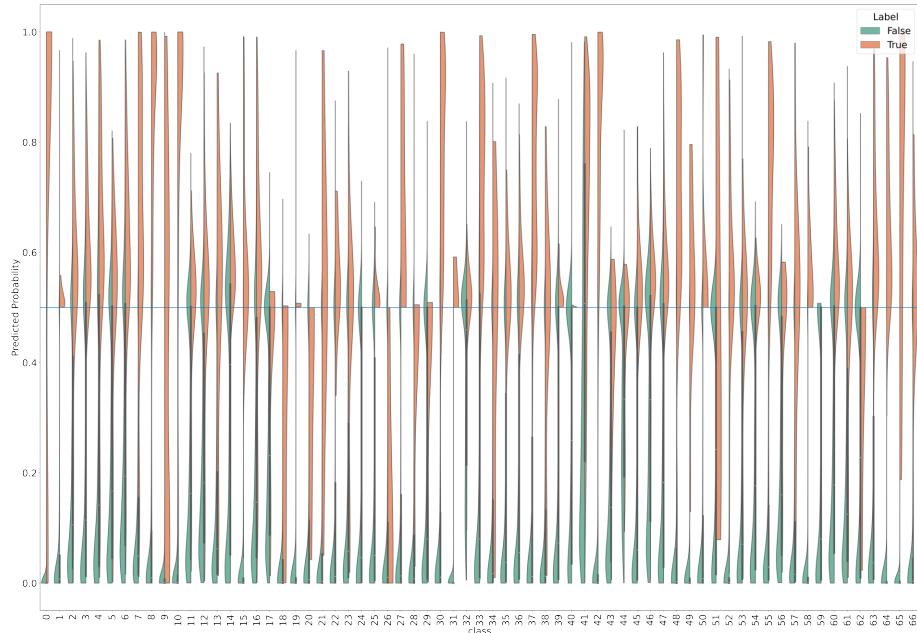
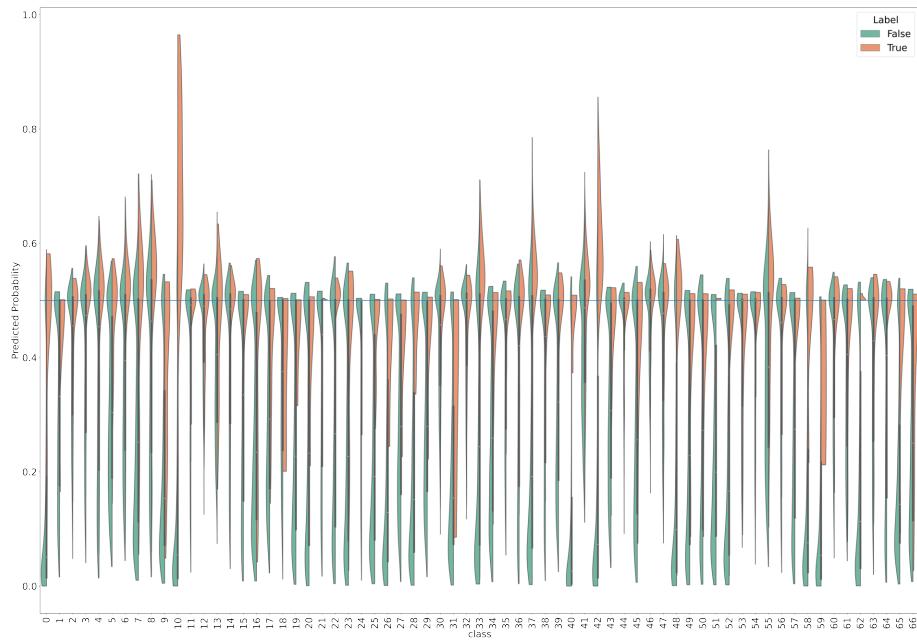
(a) Violinplot for a CPC model; unfrozen during Downstream Training. Horizontal lines show model specific decision thresholds



(b) Violinplot for Baseline (v14) model. Horizontal lines show model specific decision thresholds

Yet again, with the model thresholds added in [Figure ??](#), the groups are more or less visibly split into true and false samples, with some exceptions like class 41 (sinus rythm), where true samples get classified as false or false samples as true fairly often. The comparison between CPC model and baseline model is difficult, but we notice that "False" classes are more oftenly wrongly classified as true for the CPC model. Interesting to see is that both models failed to predict class 36 correctly, where the baseline model even predicted samples as false with high certainty ($p = 0$). On the other side these plots also show strengths of specific models: Class 31 is correctly split into true and false samples for the baseline model but not the CPC model, whereas class 62 is correctly classified by CPC but not the baseline. Looking at the class distribution in [Figure 3](#) we observe that class 62 is a small class only available in select datasets, again hinting at CPC's ability of needing less labeled data.

With the help of these plots one could create a model ensemble which utilizes different networks for each class, creating a more powerful and accurate prediction model.



5.2.6 Explaining predicted classes in input data

As a bonus, having predicted a diagnostic label, it might be of great interest to also highlight where the model assumes the predicted classes are located at in the input data. We try to accomplish this task by performing a feed-forward of the input data through the network and then calculate the loss between the prediction and the target y^c — an all-zero valued vector (~ all diagnostic labels have a probability of 0) besides a specified target class c , which is set to one (class probability 1). As of yet we use the available ground truth labels, which are, if not available, to be replaced with the binary predictions obtained with the model’s thresholds. The gradients are calculated by backpropagation with respect to the input data X . Once the gradient matrices $\frac{\partial y^c}{\partial X}$ has been obtained, we either take their absolute values to visualize areas that have an impact in general or apply a ReLU activation to highlight areas that positively influence the target’s class prediction. The latter idea is based on the Grad-CAM paper [4], the procedure in general is called guided backpropagation [38]. We normalize the matrices individually to $[0, 1]$ and assign colors for each class, which increase in opacity for high values and become transparent below a certain threshold (e.g. < 0.2 , which varies depending on model). We can see what values in the data would have to change in order to decrease the loss: High gradient values \sim high influence on the predicted classes; low gradient values \sim less influence on the predicted classes. To show multiple classes/colors in the same spots we divide the ecg channels into the number of present classes and color each division separately. The final result with multiple patients as example can be seen in [Figure 26](#) for the BL_v8 model and in [Figure 27](#) for a CPC Model.

Interestingly the CPC model’s biggest gradient values are less spread throughout the input which could potentially lead to marking interesting spots with a higher precision. On the other hand locations might not get marked although classes are present there. The markings in the baseline model are almost everywhere, making class judgments more difficult.

Additionally to our approach of calculating $\frac{\partial y^c}{\partial X}$, we tested Grad-CAM [4] which takes the gradient $\frac{\partial y^c}{\partial A^k}$ of the convolutional feature maps A^k in a selected layer and calculates $L_{\text{Grad-CAM}}^c = \text{ReLU}(\sum_k \alpha_k^c A^k)$, where $\alpha_k^c = \frac{1}{ij} \sum_i \sum_j \frac{\partial y^c}{\partial A_{i,j}^k}$ is the mean/ global average pooling of feature map A^k . $L_{\text{Grad-CAM}}^c$ can thus be seen as the sum of all feature maps in a specified layer, weighted according to their importance in predicting a specified class c , which results in a coarse heatmap where the model assumes the class to be located. One big upside compared to our localization approach is that the locations are directly generated from later layers, where most class features are encoded in. The downside however is that the heatmap size is dependent on the individual architecture and not the input data. This leads to few values available for localization in some cases. Additionally, in contrast to our approach, all channel information get lost, meaning eg. ecg-channel v1 and ecg-channel v3 are colored the exact same, although a specific class can only be recognized in one of them.

The coarse heatmap is then rescaled to match the input size, which is 4500 in our case. Since the layers feature map channels get summarized into a single value, no spots depending on input channels can be recovered and all input channels share the same areas of interest. This is to say that even if a certain class is only visible in a specific ecg channel

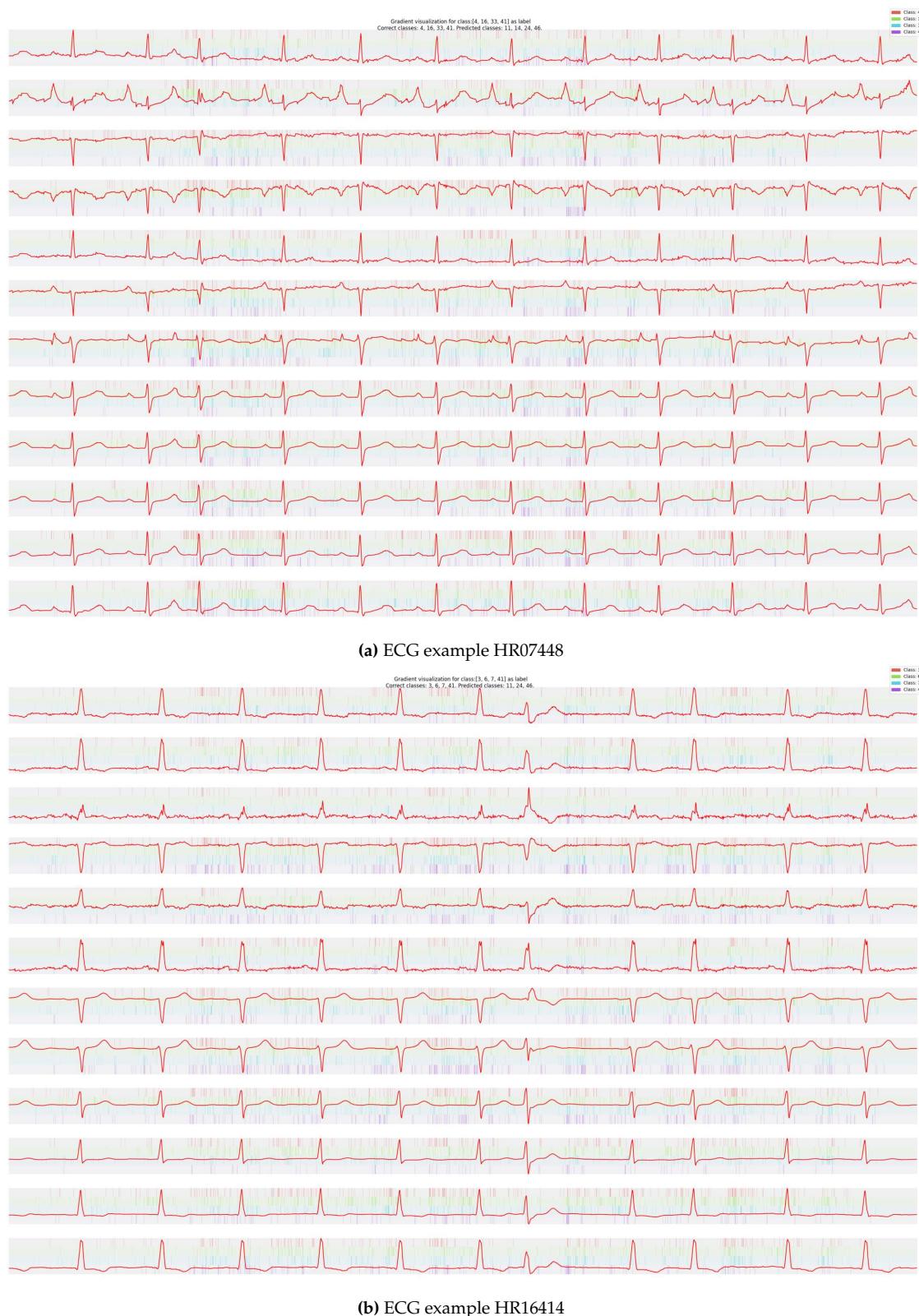


Figure 26: BL_v8 model gradients. Absolute gradient values utilized. Colored vertical bars/areas show "interesting/controversial" spots in the input data.

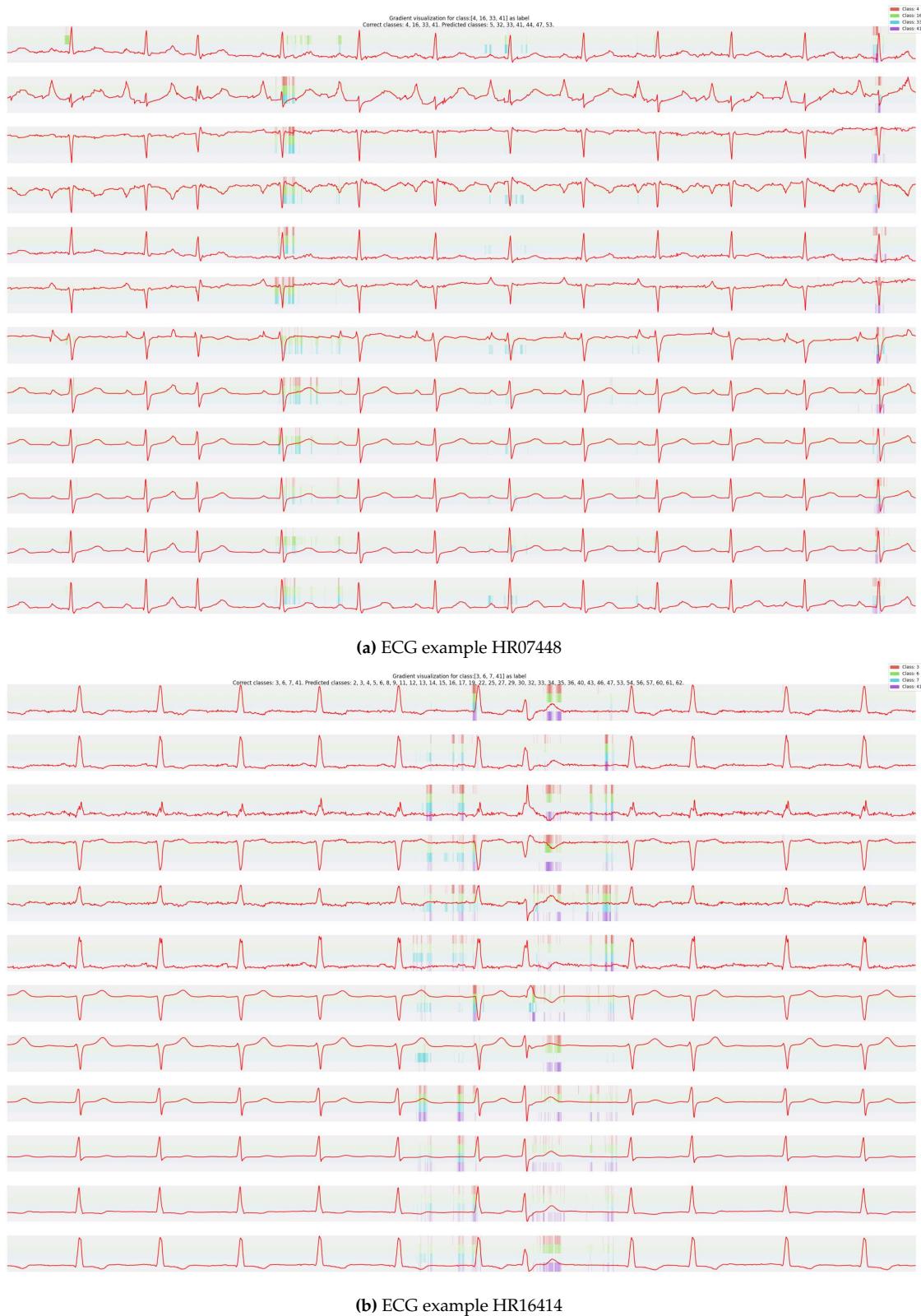


Figure 27: CPC model (with encoder_v0) gradients. Absolute gradient values utilized. Colored vertical bars/areas show "interesting/controversial" spots in the input data.

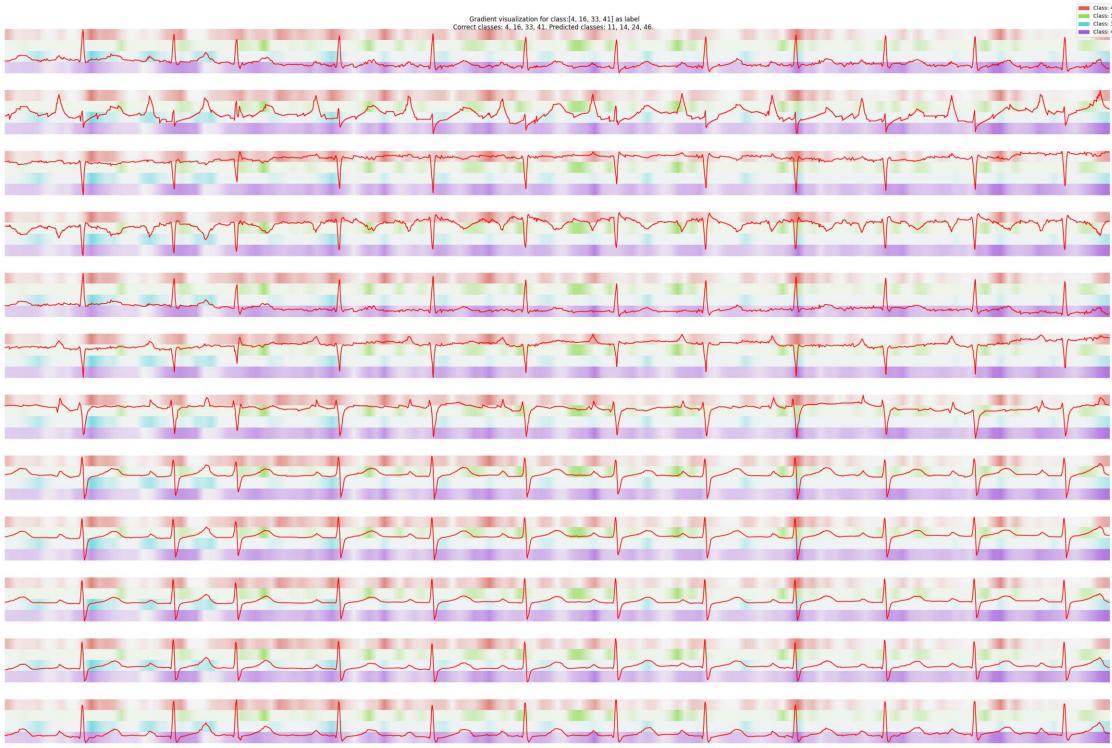


Figure 28: ECG examples, for the BL_v8 Model, Grad-Cam technique used. 147 feature map values stretched to input length of 4500.

all others will be colorized as well, which is a clear downside to our approach. Furthermore the heat map's size is dependent on the downsampling factor in the data dimension of the model architecture: For our `encoder_v0` CPC model only 1×26 values are generated, which allows for a rough localization of classes in the input data only. The BL_v8 model generates 1×147 values which results in much finer localization in the input data and thus more accurate results. The `TCN_down` architecture is probably the most interesting one since it makes use of heavy padding and no strides, making it output 4500 values (same as input) even in the later layers. We expect the best results there.

In [Figure 28](#) we see that Grad-Cam created spots in the data that might be able to successfully explain the classes position in the input data. For example the normal sinus rhythm (no. 41) can be found in regular intervals everywhere in the data as expected, while "Left Axis Deviation" (class 33) can only be found at a single spot. "Mycardial Infarction" (class 4) is assumed at 2-3 spots.

For CPC model [Figure 29a](#) we see that the grad-cam results are not very localized and all over the place, also the different classes do not seem to differ between one another. We compare to a CPC model that is trained with a different encoder to check whether the poor performance is always present: [Figure 29b](#). We observe similar results as with the baseline model.

The TCN baseline model in [Figure 30](#) has a very clear localization of classes, which is probably mostly due to the big output size even in later layers, since the v8 network has a similar macro average score.

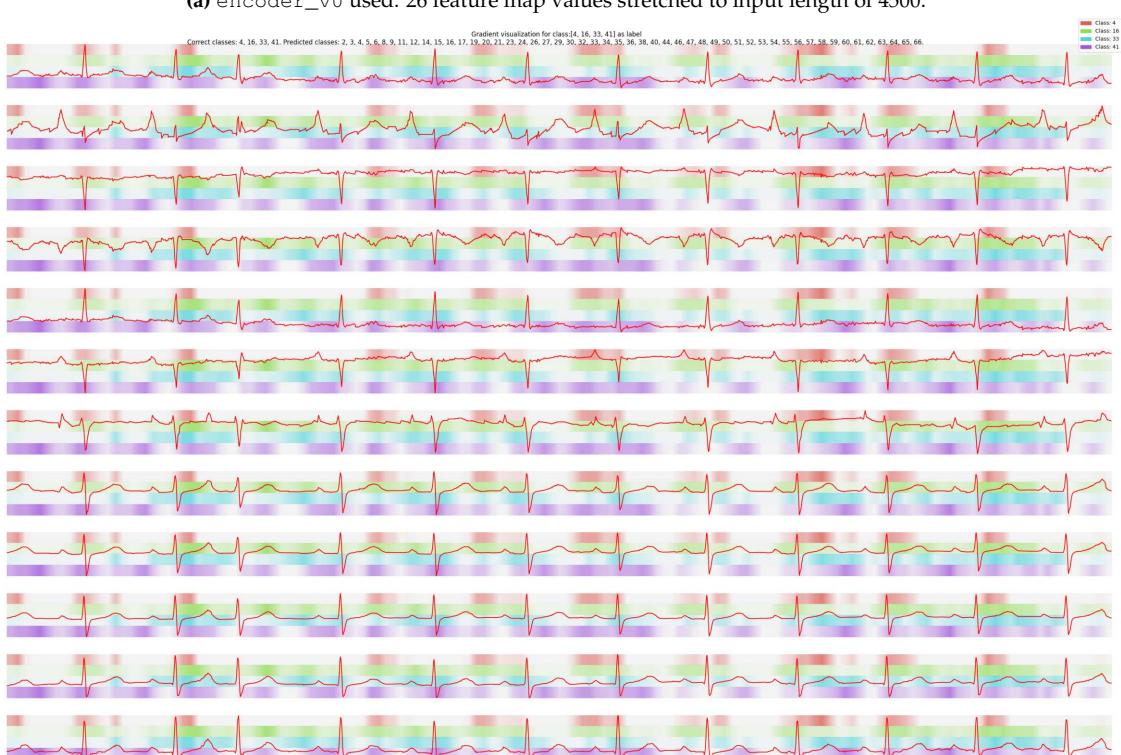
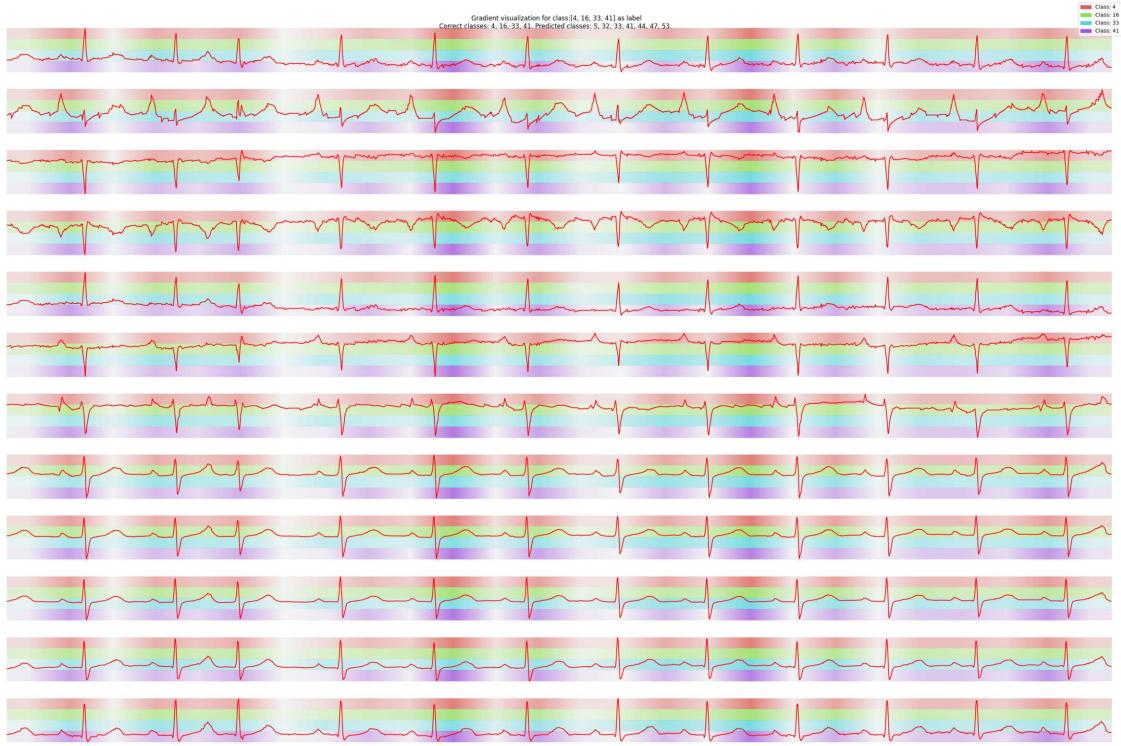


Figure 29: ECG examples, for the CPC Model, Grad-Cam technique used.

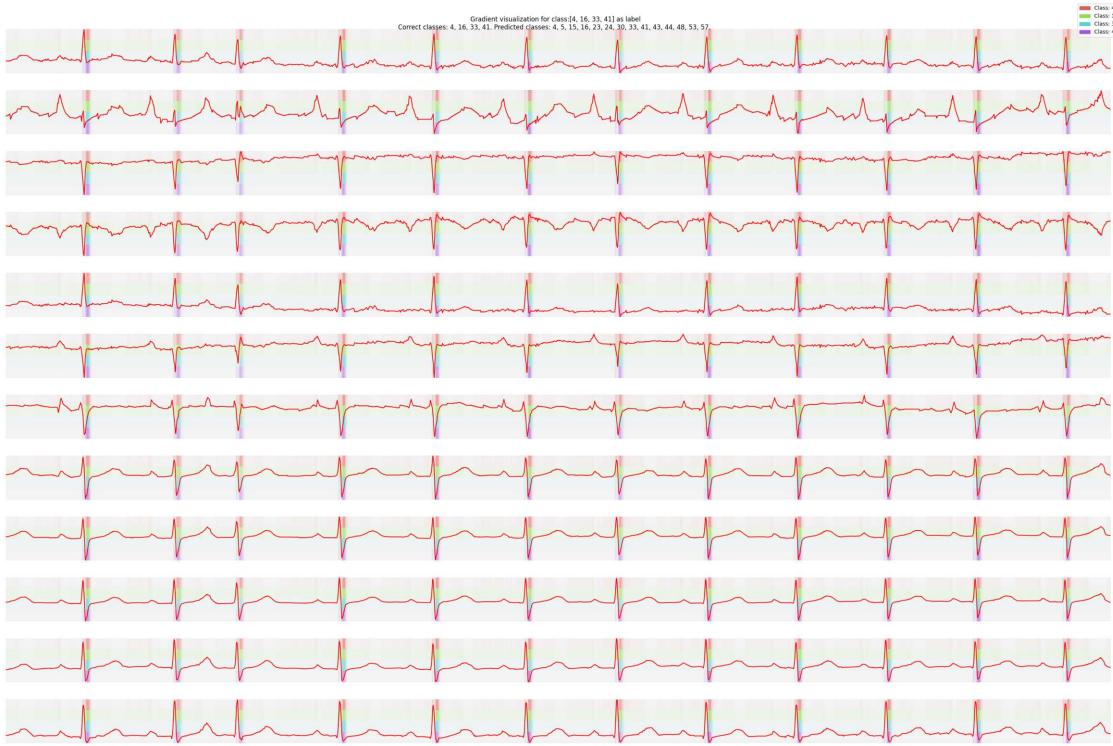


Figure 30: ECG examples, for the TCN (down) architecture, Grad-Cam technique used. 4500 feature map values 'stretched' to input length of 4500.

Last but not least we asked a cardiologist to rate the class markings in the grad-cam images and we were given the following comments:

- In [Figure 28](#) the sinus rhythm (class 41) is off place in some places and generally too spread out. When labeling it you would pick the center of the QRS-points
- [Figure 29a](#) is marking too much, no specific locations are clearly recognizable.
- In [Figure 29b](#) the sinus rhythm is again off beat in some places but otherwise recognizable. The myocardial infarction is diagnosed mostly by looking at the S-T interval, which is also marked here. Left axis deviation is a "turned-around" QRS complex, so the localization seems okay, but should only be marked in selected channels (eg. channel 3).
- [Figure 30](#) perfectly shows the sinus rhythm center at the QRS complex but disregards the P-wave completely. The myocardial infarction is not diagnosed by looking at the QRS complex, making these markings probably incorrect. Left axis deviation is visible in some channels only, but all got marked.

When asked to pick a "best-performing" figure they selected [Figure 29b](#).

Especially the fact that some classes are only visible in certain channels shows that the grad-cam technique is intended for use on images where different data channels encode

color but have no independent role in localizing different classes. Calculating the gradient towards the input does not share this drawback but the gradient is not as suited for localizing class labels. We tried "Guided-Grad-Cam" which in our case struggled to display anything at all, since guided backpropagation and gradient cam are multiplied to receive the final output, the image was almost always mostly without colors due to values close to 0.

6 Discussion and Future work

In a fully supervised setting, our baselines were able to produce slightly better prediction results compared to our Contrastive Predictive Coding models. Once we handicapped CPC by splitting the training into a pretraining and downstream training phase, with the same number of epochs given in total, the advance for the baseline models increased even further. Yet the CPC results were still above average.

Thus, when labels are available in large enough quantities, supervised learning methods, that have seen much research in the past, are probably easier to optimize and are able to produce better results as a consequence.

In environments where only few labels are given, which we tried to emulate in our "Low Label Availability" experiment in [Subsubsection 5.1.4](#), CPC showed better results compared to baseline models, when the least labels were given, after allowing sufficient training epochs. This is an indicator that CPC pretraining can improve classification results in fields where unlabeled data is readily available, while high quality labels are rare and only obtainable with high costs and effort — which is the case in many fields of modern medicine.

Furthermore our augmented/noised data experiment in [Subsubsection 5.1.5](#) suggests that CPC pretraining increases training stability in the downstream task for unbalanced datasets, because both baseline- and the CPC networks that were trained fully supervised, mostly failed to classify the data satisfactory.

An additional strong benefit of CPC is time efficiency. Downstream tasks, where one has to train models with the same input data, but different labels, benefit of the shared weights obtained during pretraining. Often CPC can yield good results with only few downstream epochs, reducing the necessary total training time drastically. This especially showed during training on the different train splits in "Low Label Availability" experiment: while the baselines were trained completely from ground up for every split, needing more than 24 hours wall clock time, the CPC models were trained in a fraction of that time.

CPC allows for many architectural choices, for example the encoder network and many hyperparameters like latent-size, context-size, number of summarized latents, number of predicted latents and the latent sampling method to name just a few. This makes it very flexible and applicable to many data modalities, as it does not make explicit demands on label shapes and does not require pseudo-labels like many other representation learning algorithms mentioned in the introduction, making it a strong contender for universal representation learning. This also shows in the original paper, where CPC was applied to multiple data modalities, such as images, timeseries or even reinforcement learning[[1](#)]. However the great flexibility might also bear the risk of finding only suboptimal network settings for your specific task. This can also be seen in our implementation [Subsubsection 5.1.6](#), where we showed that a context network is not even required to learn useful representations, but also that changing the encoder resulted in worse classification results, albeit giving more flexibility to the user.

Last but not least we applied grad-cam [[4](#)] to selected ECG samples, where a CPC network using our `cpc_encoder_likev8` encoder module, was voted the visually best

performing network by a cardiologist.

In conclusion we infer that Contrastive Predictive Coding is an overall good method for classifying ECG-data, but for the pure classification task our baseline models that were trained purely supervised performed better and should be used instead. CPC networks' strengths can be seen when only few labels are available or the data is noisy, but we had to artificially create these environments for the ECG data to outperform the baseline models. CPC yielded the best results in our colorization task, which are however still far from perfect.

Future work might include the search for additional representation learning methods that also cope well with timeseries/sequential/non-image data. This includes the aforementioned altered CPC architectures, but also unrelated representation learning networks like VQ-VAE or more simple encoder-decoder architectures.

Additionally the findings of [39] show that the original CPC has a high bias and needs a lot of negative samples to correctly approach the mutual information value. They introduce a revised CPC called α -CPC that uses weights on both positive and negative samples. According to their formulation the original CPC then becomes a special case of α -CPC. They also introduced a method called "Multi-label Contrastive Predictive Coding" where multiple positive samples are used to greatly decrease the theoretically needed amount of negative samples. Their conducted experiments showed a superior mutual information estimation and slightly better accuracy overall. We tried to reimplement α -CPC but our networks failed to decrease the loss satisfactory. Correctly reproducing their findings could potentially lead to better results in our experiments.

A Appendix

B Code (folder)

B.1 architectures_baseline_challenge (folder)

B.1.1 architectures_baseline_challenge -> baseline_FCN.py (code)

```

1 import torch
2 from torch import nn
3
4
5 # Time Series Classification from Scratch with Deep
6 # Neural Networks: A Strong Baseline https://arxiv.org/pdf/1611.06455.pdf
7 class BaselineNet(nn.Module):
8
9     def __init__(self, in_channels, out_channels, out_classes=94, verbose=False) -> None:
10         super(BaselineNet, self).__init__()
11         self.verbose = verbose
12         self.features = nn.Sequential(
13             nn.Conv2d(in_channels=in_channels, out_channels=128, kernel_size=8),
14             nn.BatchNorm2d(num_features=128),
15             nn.ReLU(),
16
17             nn.Conv2d(in_channels=128, out_channels=256, kernel_size=5),
18             nn.BatchNorm2d(num_features=256),
19             nn.ReLU(),
20
21             nn.Conv2d(in_channels=256, out_channels=out_classes, kernel_size=3),
22             nn.BatchNorm2d(num_features=out_classes),
23             nn.ReLU(),
24
25         )
26         self.activation = nn.Sigmoid()
27
28     def forward(self, x: torch.Tensor, y=None) -> torch.Tensor:
29         if self.verbose: print(x.shape)
30         x = x.transpose(1, 2)
31         if self.verbose: print(x.shape)
32         x = self.features(x)
33         if self.verbose: print(x.shape)
34         x = torch.mean(x, dim=-1) # Global average pooling
35         if self.verbose: print(x.shape)
36         x = self.activation(x)
37         if self.verbose: print(x.shape)
38
39         return x

```

B.1.2 architectures_baseline_challenge -> baseline_MLP.py (code)

```

1 import torch
2 from torch import nn
3
4
5 # Time Series Classification from Scratch with Deep
6 # Neural Networks: A Strong Baseline https://arxiv.org/pdf/1611.06455.pdf
7 class BaselineNet(nn.Module):
8
9     def __init__(self, in_features=4500, out_classes=94, verbose=False) -> None:
10         super(BaselineNet, self).__init__()
11         self.verbose = verbose
12         self.classifier = nn.Sequential(
13             nn.Dropout(0.1),
14             nn.Linear(in_features=in_features, out_features=500),
15             nn.ReLU(),
16             nn.Dropout(0.2),
17             nn.Linear(in_features=500, out_features=500),
18             nn.ReLU(),
19             nn.Dropout(0.2),
20             nn.Linear(in_features=500, out_features=out_classes),
21             nn.ReLU(),
22             nn.Dropout(0.3)
23         )
24         self.activation = nn.Sigmoid()
25
26     def forward(self, x: torch.Tensor, y=None) -> torch.Tensor:
27         x = x[:, :, 0] # Only take first channel for this network
28         x = self.classifier(x)
29         x = self.activation(x)
30         if self.verbose: print(x.shape)
31

```

B.1.3 architectures_baseline_challenge -> baseline_TCN_block.py (code)

```

1 import torch
2 from torch import nn
3 from torch.nn.utils import weight_norm
4
5
6 # might be broken (bad)
7 class BaselineNet(nn.Module):
8
9     def __init__(self, in_channels, out_channels, out_classes, verbose):
10         super().__init__()
11         self.n_out_classes = out_classes
12         self.verbose = verbose
13         self.tcn_block = nn.Sequential(
14             weight_norm(nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=3, stride=2, padding
15 =0,
16                         dilation=1)),
17             nn.ReLU(),
18             nn.Dropout(0.2),
19
20             weight_norm(nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=3, stride=1, padding
21 =0,
22                         dilation=2)),
23             nn.ReLU(),
24             nn.Dropout(0.2),
25         )
26         self.res_downsample = nn.Conv1d(in_channels=4500, out_channels=2245, kernel_size=1)
27         # self.downsample = nn.Conv1d(in_channels=4745, out_channels=1, kernel_size=1)
28         self.fc = nn.Linear(26940, out_classes)
29         self.activation = nn.Sigmoid()
30         self.criterion = nn.BCELoss()
31
32     def forward(self, x, y=None):
33         if self.verbose: print('input shape', x.shape)
34         batch, window_size, channels = x.shape
35         x_res = x.clone()
36         x = x.transpose(1, 2) # torch.unsqueeze(torch.flatten(x, start_dim=1), 1)#
37         if self.verbose: print('after transpose', x.shape)
38         x = self.tcn_block(x)
39         if self.verbose: print('x shape after tcn', x.shape)
40         x_res = self.res_downsample(x_res)
41         if self.verbose: print('x_res shape after residual downsample', x_res.shape)
42         x = x.transpose(1, 2) + x_res
43         if self.verbose: print('x shape after res add', x.shape)
44         x = torch.flatten(x, start_dim=1)
45         if self.verbose: print('x shape after flatten', x.shape)
46         x = self.fc(
47             x) # Like in https://github.com/locuslab/TCN/blob/master/TCN/mnist_pixel/model.py TODO: why is that okay?
48         if not y is None:
49             # loss = self.criterion(logits, y)
50             loss = torch.sum(torch.square(y - logits)) # Simple own implementation
51             # loss = self.criterion(logits, torch.argmax(y, dim=1))
52             accuracies = []
53             mask = y != 0.0
54             inverse_mask = ~mask
55             zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
56                 inverse_mask) # zero fit goal
57             class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
58             accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
59             accuracies.append(class_fit)
60             accuracies.append(zero_fit)
61
62             accuracies.append(
63                 1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
64             values
65             # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
66             count non zeros in accuracy?
67             accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
68                 self.n_out_classes * batch)) # correct if probability within 0.01
69             # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
70             return accuracies, loss
71         else:
72             return logits

```

B.1.4 architectures_baseline_challenge -> baseline_TCN_down.py (code)

```

1 import torch
2 from torch import nn
3
4 from external.tcn.TCN.tcn import TemporalConvNet
5
6
7 # okay/bad

```

```

8
9 class BaselineNet(nn.Module):
10
11     def __init__(self, in_channels, out_channels, out_classes, verbose):
12         super().__init__()
13         self.n_out_classes = out_classes
14         self.verbose = verbose
15         self.tcn = TemporalConvNet(in_channels, [in_channels * 2 ** 1, in_channels * 2, out_classes], kernel_size=3,
16                                   dropout=0.2)
17         self.downsample = nn.Conv1d(in_channels=4500, out_channels=1, kernel_size=1)
18
19         self.fc = nn.Linear(out_classes, out_classes)
20         # self.activation = nn.LogSoftmax(dim=1)
21         # self.criterion = nn.NLLLoss()
22         self.activation = nn.Sigmoid()
23         self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
24
25     def forward(self, X, y=None):
26         if self.verbose: print('input shape', X.shape)
27         batch, window_size, channels = X.shape
28         x = X.transpose(1, 2)
29         if self.verbose: print(x.shape)
30         x = self.tcn(x)
31         if self.verbose: print('x shape after conv', x.shape)
32         x = x.transpose(1, 2)
33         if self.verbose: print('x shape after transpose', x.shape)
34         x = self.downsample(x)
35         if self.verbose: print('x shape after downsample', x.shape)
36         x = self.fc(x).squeeze(1)
37         if self.verbose: print('x shape after fc', x.shape)
38         logits = self.activation(x)
39         if not y is None:
40             # loss = self.criterion(logits, y)
41             loss = torch.sum(torch.square(y - logits)) # Simple own implementation
42             # loss = self.criterion(logits, torch.argmax(y, dim=1))
43             accuracies = []
44             mask = y != 0.0
45             inverse_mask = ~mask
46             zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
47                 inverse_mask) # zero fit goal
48             class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
49             accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
50             accuracies.append(class_fit)
51             accuracies.append(zero_fit)
52
53             accuracies.append(
54                 1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
55             values
56             # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
57             # count non zeros in accuracy?
58             accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
59                 self.n_out_classes * batch)) # correct if probabiltiy within 0.01
60             # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
61             return accuracies, loss
62         else:
63             return logits

```

B.1.5 architectures_baseline_challenge -> baseline_TCN_flatten.py (code)

```

1 import torch
2 from torch import nn
3
4 from external.tcn.TCN.tcn import TemporalConvNet
5
6
7 class BaselineNet(nn.Module):
8
9     def __init__(self, in_channels, out_channels, out_classes, verbose):
10        super().__init__()
11        self.n_out_classes = out_classes
12        self.verbose = verbose
13        self.tcn = TemporalConvNet(1, [in_channels * 2 ** 0, in_channels * 2 ** 2, in_channels * 2 ** 3], kernel_size
14                                  =3,
15                                  dropout=0.2)
16        # self.downsample = nn.Conv1d(in_channels=9500, out_channels=1, kernel_size=1)
17
18        self.fc = nn.Linear(in_channels * 2 ** 3, out_classes)
19        # self.activation = nn.LogSoftmax(dim=1)
20        # self.criterion = nn.NLLLoss()
21        self.activation = nn.Sigmoid()
22        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
23
24    def forward(self, x, y=None):
25        if self.verbose: print('input shape', x.shape)
26        batch, window_size, channels = x.shape
27        x = torch.unsqueeze(torch.flatten(x, start_dim=1), 1) # .transpose(1, 2)

```

```

27     if self.verbose: print('after flatten + unsqueeze', x.shape)
28     x = self.tcn(x)
29     if self.verbose: print('x shape after tcn', x.shape)
30     x = self.fc(x[:, :, :-1]) # Like in https://github.com/locuslab/TCN/blob/master/TCN/mnist_pixel/model.py TODO: why is
31     that okay?
32     if self.verbose: print('x shape after fc', x.shape)
33     logits = self.activation(x)
34     if not y is None:
35         # loss = self.criterion(logits, y)
36         loss = torch.sum(torch.square(y - logits)) # Simple own implementation
37         # loss = self.criterion(logits, torch.argmax(y, dim=1))
38         accuracies = []
39         mask = y != 0.0
40         inverse_mask = ~mask
41         zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
42             inverse_mask) # zero fit goal
43         class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
44         accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
45         accuracies.append(class_fit)
46         accuracies.append(zero_fit)
47
48         accuracies.append(
49             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
50         values
51         # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
52         # count non zeros in accuracy?
53         accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
54             self.n_out_classes * batch)) # correct if probability within 0.01
55         # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
56         return accuracies, loss
57     else:
58         return logits

```

B.1.6 architectures_baseline_challenge -> baseline_TCN_last.py (code)

```

1 import torch
2 from torch import nn
3
4 from external.tcn.TCN.tcn import TemporalConvNet
5
6
7 class BaselineNet(nn.Module):
8     # Bad
9     def __init__(self, in_channels, out_channels, out_classes, verbose):
10         super().__init__()
11         self.n_out_classes = out_classes
12         self.verbose = verbose
13         self.tcn = TemporalConvNet(in_channels, [in_channels * 2 ** 1, in_channels * 2, out_classes], kernel_size=3,
14                                     dropout=0.2)
15         # self.downsample = nn.Conv1d(in_channels=9500, out_channels=1, kernel_size=1)
16
17         self.fc = nn.Linear(out_classes, out_classes)
18         # self.activation = nn.LogSoftmax(dim=1)
19         # self.criterion = nn.NLLLoss()
20         self.activation = nn.Sigmoid()
21         self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
22
23     def forward(self, X, y=None):
24         if self.verbose: print('input shape', X.shape)
25         batch, window_size, channels = X.shape
26         x = X.transpose(1, 2)
27         if self.verbose: print(x.shape)
28         x = self.tcn(x)
29         if self.verbose: print('x shape after tcn', x.shape)
30         x = self.fc(x[:, :, :-1]) # Like in https://github.com/locuslab/TCN/blob/master/TCN/mnist_pixel/model.py TODO: why is
31         that okay?
32         if self.verbose: print('x shape after fc', x.shape)
33         logits = self.activation(x)
34         if not y is None:
35             # loss = self.criterion(logits, y)
36             loss = torch.sum(torch.square(y - logits)) # Simple own implementation
37             # loss = self.criterion(logits, torch.argmax(y, dim=1))
38             accuracies = []
39             mask = y != 0.0
40             inverse_mask = ~mask
41             zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
42                 inverse_mask) # zero fit goal
43             class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
44             accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
45             accuracies.append(class_fit)
46             accuracies.append(zero_fit)
47
48             accuracies.append(
49                 1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
50             values

```

```

50     # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
51     count non zeros in accuracy?
52     accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
53         self.n_out_classes * batch)) # correct if probability within 0.01
54     # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
55     return accuracies, loss
56 else:
57     return logits

```

B.1.7 architectures_baseline_challenge -> baseline_alex.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes=94, verbose=False) -> None:
8         super(BaselineNet, self).__init__()
9         self.features = nn.Sequential(
10             nn.Conv2d(in_channels, 64, kernel_size=11, stride=4, padding=2),
11             nn.ReLU(inplace=True),
12             nn.MaxPool2d(kernel_size=3, stride=2),
13             nn.Conv2d(64, 192, kernel_size=5, padding=2),
14             nn.ReLU(inplace=True),
15             nn.MaxPool2d(kernel_size=3, stride=2),
16             nn.Conv2d(192, 384, kernel_size=3, padding=1),
17             nn.ReLU(inplace=True),
18             nn.Conv2d(384, 256, kernel_size=3, padding=1),
19             nn.ReLU(inplace=True),
20             nn.Conv2d(256, 256, kernel_size=3, padding=1),
21             nn.ReLU(inplace=True),
22             nn.MaxPool2d(kernel_size=3, stride=2),
23         )
24         self.avgpool = nn.AdaptiveAvgPool2d(6)
25         self.classifier = nn.Sequential(
26             nn.Dropout(),
27             nn.Linear(256 * 6, 4096),
28             nn.ReLU(inplace=True),
29             nn.Dropout(),
30             nn.Linear(4096, 4096),
31             nn.ReLU(inplace=True),
32             nn.Linear(4096, out_classes),
33         )
34         self.activation = nn.Sigmoid()
35
36     def forward(self, x: torch.Tensor, y=None) -> torch.Tensor:
37         x = x.transpose(1, 2)
38         x = self.features(x)
39         x = self.avgpool(x)
40         x = torch.flatten(x, 1)
41         x = self.classifier(x)
42         x = self.activation(x)
43         return x

```

B.1.8 architectures_baseline_challenge -> baseline_alex_v2.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module): # strides and kernel_sizes of alexnet to power of 2
6
7     def __init__(self, in_channels, out_channels, out_classes=94, verbose=False) -> None:
8         super(BaselineNet, self).__init__()
9         self.features = nn.Sequential(
10             nn.Conv2d(in_channels, 64, kernel_size=11 ** 2, stride=3 ** 2, padding=2), # here 3 instead of 4
11             nn.ReLU(inplace=True),
12             nn.MaxPool2d(kernel_size=3 ** 2, stride=2 ** 2),
13             nn.Conv2d(64, 192, kernel_size=5, padding=2),
14             nn.ReLU(inplace=True),
15             nn.MaxPool2d(kernel_size=3 ** 2, stride=2 ** 2),
16             nn.Conv2d(192, 384, kernel_size=3, padding=1),
17             nn.ReLU(inplace=True),
18             nn.Conv2d(384, 256, kernel_size=3 ** 2, padding=1),
19             nn.ReLU(inplace=True),
20             nn.Conv2d(256, 256, kernel_size=3 ** 2, padding=1),
21             nn.ReLU(inplace=True),
22             nn.MaxPool2d(kernel_size=3 ** 2, stride=2 ** 2),
23         )
24         self.avgpool = nn.AdaptiveAvgPool2d(6)
25         self.classifier =

```

```

26         nn.Dropout(),
27         nn.Linear(256 * 6, 4096),
28         nn.ReLU(inplace=True),
29         nn.Dropout(),
30         nn.Linear(4096, 4096),
31         nn.ReLU(inplace=True),
32         nn.Linear(4096, out_classes),
33     )
34     self.activation = nn.Sigmoid()
35
36     def forward(self, x: torch.Tensor, y=None) -> torch.Tensor:
37         x = x.transpose(1, 2)
38         x = self.features(x)
39         x = self.avgpool(x)
40         x = torch.flatten(x, 1)
41         x = self.classifier(x)
42         x = self.activation(x)
43
44     return x

```

B.1.9 architectures_baseline_challenge -> baseline_cnn_v0.py (code)

```

1 import torch
2 from torch import nn
3
4 """
5 2 Conv Layersv0
6 """
7
8
9 class BaselineNet(nn.Module):
10
11     def __init__(self, in_channels, out_channels, out_classes, verbose):
12         super().__init__()
13         self.n_out_classes = out_classes
14         self.verbose = verbose
15         self.convs = nn.Sequential(
16             nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=7, dilation=1),
17             nn.ReLU(),
18             nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=3),
19             nn.ReLU(),
20         )
21         self.downsample = nn.Conv2d(in_channels=4488, out_channels=1, kernel_size=1)
22
23         self.fc = nn.Linear(out_channels, out_classes)
24         # self.activation = nn.LogSoftmax(dim=1)
25         # self.criterion = nn.NLLLoss()
26         self.activation = nn.Sigmoid()
27         self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
28
29     def forward(self, X, y=None):
30         if self.verbose: print('input shape', X.shape)
31         batch, window_size, channels = X.shape
32         x = X.transpose(1, 2)
33         if self.verbose: print(x.shape)
34         x = self.convs(x)
35         if self.verbose: print('x shape after conv', x.shape)
36         x = x.transpose(1, 2)
37         if self.verbose: print('x shape after transpose', x.shape)
38         x = self.downsample(x)
39         if self.verbose: print('x shape after downsample', x.shape)
40         x = self.fc(x).squeeze(1)
41         if self.verbose: print('x shape after fc', x.shape)
42         logits = self.activation(x)
43         if not y is None:
44             # loss = self.criterion(logits, y)
45             loss = torch.sum(torch.square(y - logits)) # Simple own implementation
46             # loss = self.criterion(logits, torch.argmax(y, dim=1))
47             accuracies = []
48             mask = y != 0.0
49             inverse_mask = ~mask
50             zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
51                 inverse_mask) # zero fit goal
52             class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
53             accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
54             accuracies.append(class_fit)
55             accuracies.append(zero_fit)
56
57             accuracies.append(
58                 1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
59             values
60             # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
61             # count non zeros in accuracy?
62             accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
63                 self.n_out_classes * batch)) # correct if probability within 0.01
64             # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
65
66     return accuracies, loss

```

```

64     else:
65         return logits

```

B.1.10 architectures_baseline_challenge -> baseline_cnn_v0_1.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=7, dilation=1, stride=2),
13            nn.ReLU(),
14            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=3, stride=1),
15            nn.ReLU(),
16        )
17        self.downsample = nn.Conv2d(in_channels=2241, out_channels=1, kernel_size=1)
18
19        self.fc = nn.Linear(out_channels, out_classes)
20        # self.activation = nn.LogSoftmax(dim=1)
21        # self.criterion = nn.NLLLoss()
22        self.activation = nn.Sigmoid()
23        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
24
25    def forward(self, X, y=None):
26        if self.verbose: print('input shape', X.shape)
27        batch, window_size, channels = X.shape
28        x = X.transpose(1, 2)
29        if self.verbose: print(x.shape)
30        x = self.convs(x)
31        if self.verbose: print('x shape after conv', x.shape)
32        x = x.transpose(1, 2)
33        if self.verbose: print('x shape after transpose', x.shape)
34        x = self.downsample(x)
35        if self.verbose: print('x shape after downsample', x.shape)
36        x = self.fc(x).squeeze(1)
37        if self.verbose: print('x shape after fc', x.shape)
38        logits = self.activation(x)
39        if not y is None:
40            # loss = self.criterion(logits, y)
41            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
42            # loss = self.criterion(logits, torch.argmax(y, dim=1))
43            accuracies = []
44            mask = y != 0.0
45            inverse_mask = ~mask
46            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
47                inverse_mask) # zero fit goal
48            class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
49            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
50            accuracies.append(class_fit)
51            accuracies.append(zero_fit)
52
53            accuracies.append(
54                1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
55            values
56            # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
57            # count non zeros in accuracy?
58            accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
59                self.n_out_classes * batch)) # correct if probability within 0.01
60            # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
61        else:
62            return logits

```

B.1.11 architectures_baseline_challenge -> baseline_cnn_v0_2.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(

```

```

12     nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=7, dilation=1, stride=4),
13     nn.ReLU(),
14     nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=5, dilation=1, stride=3),
15     nn.ReLU(),
16     nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=5, dilation=1, stride=3),
17     nn.ReLU(),
18     nn.Conv1d(in_channels=in_channels, out_channels=32, kernel_size=3, dilation=1, stride=1),
19     nn.ReLU(),
20     nn.Conv1d(in_channels=32, out_channels=32, kernel_size=3, dilation=2, stride=1),
21     nn.ReLU(),
22     nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3, dilation=4, stride=1),
23     nn.ReLU()
24
25 )
26 self.downsample = nn.Conv1d(in_channels=110, out_channels=1, kernel_size=1)
27
28 self.fc = nn.Linear(64, out_classes)
# self.activation = nn.LogSoftmax(dim=1)
# self.criterion = nn.NLLLoss()
self.activation = nn.Sigmoid()
self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
33
34 def forward(self, X, y=None):
35     if self.verbose: print('input shape', X.shape)
36     batch, window_size, channels = X.shape
37     x = X.transpose(1, 2)
38     if self.verbose: print(x.shape)
39     x = self.convs(x)
40     if self.verbose: print('x shape after conv', x.shape)
41     x = x.transpose(1, 2)
42     if self.verbose: print('x shape after transpose', x.shape)
43     x = self.downsample(x)
44     if self.verbose: print('x shape after downsample', x.shape)
45     x = self.fc(x).squeeze(1)
46     if self.verbose: print('x shape after fc', x.shape)
47     logits = self.activation(x)
48     if not y is None:
49         # loss = self.criterion(logits, y)
50         loss = torch.sum(torch.square(y - logits)) # Simple own implementation
51         # loss = self.criterion(logits, torch.argmax(y, dim=1))
52         accuracies = []
53         mask = y != 0.0
54         inverse_mask = ~mask
55         zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
56             inverse_mask) # zero fit goal
57         class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
58         accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
59         accuracies.append(class_fit)
60         accuracies.append(zero_fit)
61
62         accuracies.append(
63             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
64         values
65         # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
66         count non zeros in accuracy?
67         accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
68             self.n_out_classes * batch)) # correct if probabiltiy within 0.01
69         # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
70         return accuracies, loss
71     else:
72         return logits
73

```

B.1.12 architectures_baseline_challenge -> baseline_cnn_v0_3.py (code)

```

1 import torch
2 from torch import nn
3
4 from util.metrics import training_metrics as acm
5 from deprecated.data_storage import DataStorage
6
7
8 class BaselineNet(nn.Module):
9
10     def __init__(self, in_channels, out_channels, out_classes, verbose):
11         super().__init__()
12         self.n_out_classes = out_classes
13         self.verbose = verbose
14         self.convs = nn.Sequential(
15             nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=7, dilation=1, stride=4),
16             nn.ReLU(),
17             nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=5, dilation=1, stride=3),
18             nn.ReLU(),
19             nn.Conv1d(in_channels=in_channels, out_channels=32, kernel_size=3, dilation=1, stride=1),
20             nn.ReLU(),
21
22

```

```

23     nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3, dilation=2, stride=1),
24     nn.ReLU(),
25     nn.Conv1d(in_channels=64, out_channels=128, kernel_size=3, dilation=4, stride=1),
26     nn.ReLU()
27 )
28 self.downsample = nn.Conv1d(in_channels=110, out_channels=1, kernel_size=1)
29
30 self.fc = nn.Linear(128, out_classes)
31 # self.activation = nn.LogSoftmax(dim=1)
32 # self.criterion = nn.NLLLoss()
33 self.data_storage = DataStorage('../temp')
34 self.activation = nn.Sigmoid()
35 self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
36
37 def forward(self, X, y=None):
38     if self.verbose: print('input shape', X.shape)
39     batch, window_size, channels = X.shape
40     x = X.transpose(1, 2)
41     if self.verbose: print(x.shape)
42     x = self.convs(x)
43     if self.verbose: print('x shape after conv', x.shape)
44     x = x.transpose(1, 2)
45     if self.verbose: print('x shape after transpose', x.shape)
46     x = self.downsample(x)
47     if self.verbose: print('x shape after downsample', x.shape)
48     x = self.fc(x).squeeze(1)
49     if self.verbose: print('x shape after fc', x.shape)
50     logits = self.activation(x)
51     if not y is None:
52         # loss = self.criterion(logits, y)
53         loss = torch.sum(torch.square(y - logits)) # Simple own implementation
54         # loss = self.criterion(logits, torch.argmax(y, dim=1))
55         accuracies = []
56         mask = y != 0.0
57         inverse_mask = ~mask
58         mask_sum = torch.sum(mask)
59         inverse_mask_sum = torch.sum(inverse_mask)
60         zero_fit = 1.0 - torch.sum(
61             torch.square(y[inverse_mask] - logits[inverse_mask])) / inverse_mask_sum # zero fit goal
62         class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / mask_sum # class fit goal
63         accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
64         accuracies.append(class_fit)
65         accuracies.append(zero_fit)
66
67         accuracies.append(
68             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
69         values
70         # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
71         # count non zeros in accuracy?
72         accuracies.append(acm.micro_avg_precision_score(y, logits))
73         accuracies.append(acm.micro_avg_recall_score(y, logits))
74         accuracies.append(acm.f1_score(y, logits))
75         accuracies.append(acm.accuracy(y, logits))
76         # Counts the classes that have been correctly predicted - wrongly predicted with certain prob
77
78         accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
79             self.n_out_classes * batch)) # correct if probability within 0.01
80         # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
81         return accuracies, loss, [acm.tp_score_label(y, logits), acm.fp_score_label(y, logits),
82                                 acm.tn_score_label(y, logits), acm.fn_score_label(y, logits)]
83     else:
84         return logits

```

B.1.13 architectures_baseline_challenge -> baseline_cnn_v1.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose=False):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.convs = nn.Sequential(
11            nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=10, stride=5),
12            nn.BatchNorm1d(in_channels),
13            nn.ReLU(),
14            nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=8, stride=4),
15            nn.BatchNorm1d(in_channels),
16            nn.ReLU(),
17            nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=6, stride=3),
18            nn.BatchNorm1d(in_channels),
19            nn.ReLU(),
20            nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=5, stride=2),

```

```

21         nn.BatchNorm1d(in_channels),
22         nn.ReLU(),
23         nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=4, stride=2),
24         nn.BatchNorm1d(in_channels),
25         nn.ReLU(),
26         nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=3, stride=1),
27         nn.BatchNorm1d(in_channels),
28         nn.ReLU(),
29     )
30     self.pooling = nn.AdaptiveAvgPool1d(1)
31
32     self.fc = nn.Linear(in_channels, out_classes)
33     # self.activation = nn.LogSoftmax(dim=1)
34     # self.criterion = nn.NLLLoss()
35     self.activation = nn.Sigmoid()
36     self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
37
38 def forward(self, X, y=None):
39     # print('input shape', X.shape)
40     batch, window_size, channels = X.shape
41     x = X.transpose(1, 2)
42     # print(x.shape)
43     x = self.convs(x)
44     # print('x shape after convs', x.shape)
45     x = self.pooling(x).squeeze(2)
46     # print('x shape after pooling', x.shape)
47     x = self.fc(x)
48     # print('x shape after fc', x.shape)
49     logits = self.activation(x)
50     if not y is None:
51         # loss = self.criterion(logits, y)
52         loss = torch.sum(torch.square(y - logits)) # Simple own implementation
53         # loss = self.criterion(logits, torch.argmax(y, dim=1))
54         accuracies = []
55         mask = y != 0.0
56         inverse_mask = ~mask
57         zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
58             inverse_mask) # zero fit goal
59         class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
60         accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
61         accuracies.append(class_fit)
62         accuracies.append(zero_fit)
63
64         accuracies.append(
65             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
66         values
67         # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
68         # count non zeros in accuracy?
69         accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
70             self.n_out_classes * batch)) # correct if probability within 0.01
71         # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
72         return accuracies, loss
73     else:
74         return logits

```

B.1.14 architectures_baseline_challenge -> baseline_cnn_v14.py (code)

```

1 import torch
2 from torch import nn
3
4 from external.multi_scale_ori import MSResNet
5
6
7 # Good
8 class BaselineNet(nn.Module):
9
10    def __init__(self, in_channels, out_channels, out_classes, verbose):
11        super().__init__()
12        self.n_out_classes = out_classes
13        self.verbose = verbose
14        self.msresnet = MSResNet(in_channels, num_classes=out_classes)
15        self.activation = nn.Sigmoid()
16
17    def forward(self, x, y=None):
18        if self.verbose: print('input shape', x.shape)
19        batch, window_size, channels = x.shape
20        x = x.transpose(1, 2) # torch.unsqueeze(torch.flatten(x, start_dim=1), 1)#
21        if self.verbose: print('after transpose', x.shape)
22        fc_x, x = self.msresnet(x)
23        # if self.verbose: print('x shape after resnet', x.shape)
24        # x = self.fc(x)
25        # if self.verbose: print('x shape after fc', x.shape)
26        logits = self.activation(fc_x)
27        if not y is None:
28            # loss = self.criterion(logits, y)
29            loss = torch.sum(torch.square(y - logits)) # Simple own implementation

```

```

30     # loss = self.criterion(logits, torch.argmax(y, dim=1))
31     accuracies = []
32     mask = y != 0.0
33     inverse_mask = ~mask
34     zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
35         inverse_mask) # zero fit goal
36     class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
37     accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
38     accuracies.append(class_fit)
39     accuracies.append(zero_fit)
40
41     accuracies.append(
42         1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
43     values
44     # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
45     # count non zeros in accuracy?
46     accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
47         self.n_out_classes * batch)) # correct if probability within 0.01
48     # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
49     return accuracies, loss
50 else:
51     return logits

```

B.1.15 architectures_baseline_challenge -> baseline_cnn_v15.py (code)

```

1 import torch
2 from torch import nn
3
4
5 # okay
6
7 class BaselineNet(nn.Module):
8
9     def __init__(self, in_channels, out_channels, out_classes, verbose):
10        super().__init__()
11        self.n_out_classes = out_classes
12        self.verbose = verbose
13
14        self.convs = nn.Sequential(
15            nn.Conv1d(in_channels=in_channels, out_channels=out_channels, kernel_size=7, dilation=1),
16            nn.BatchNorm1d(out_channels),
17            nn.ReLU(),
18            nn.MaxPool1d(3),
19            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=7),
20            nn.BatchNorm1d(out_channels),
21            nn.ReLU(),
22            nn.MaxPool1d(3),
23            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=7 * 3),
24            nn.BatchNorm1d(out_channels),
25            nn.ReLU(),
26            nn.MaxPool1d(3),
27            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=7 * 3 * 3),
28            nn.BatchNorm1d(out_channels),
29            nn.ReLU(),
30        )
31        self.downsample = nn.Conv1d(in_channels=24, out_channels=1, kernel_size=1)
32
33        self.fc = nn.Linear(out_channels, out_classes)
34        # self.activation = nn.LogSoftmax(dim=1)
35        # self.criterion = nn.NLLLoss()
36        self.activation = nn.Sigmoid()
37        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
38
39    def forward(self, X, y=None):
40        if self.verbose: print('input shape', X.shape)
41        batch, window_size, channels = X.shape
42        x = X.transpose(1, 2)
43        if self.verbose: print(x.shape)
44        x = self.convs(x)
45        if self.verbose: print('x shape after conv', x.shape)
46        x = x.transpose(1, 2)
47        if self.verbose: print('x shape after transpose', x.shape)
48        x = self.downsample(x)
49        if self.verbose: print('x shape after downsample', x.shape)
50        x = self.fc(x).squeeze(1)
51        if self.verbose: print('x shape after fc', x.shape)
52        logits = self.activation(x)
53        if not y is None:
54            # loss = self.criterion(logits, y)
55            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
56            # loss = self.criterion(logits, torch.argmax(y, dim=1))
57            accuracies = []
58            mask = y != 0.0
59            inverse_mask = ~mask
60            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
61                inverse_mask) # zero fit goal

```

```

62     class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
63     accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
64     accuracies.append(class_fit)
65     accuracies.append(zero_fit)
66
67     accuracies.append(
68         1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
69     values
70     # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
71     # count non zeros in accuracy?
72     accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
73         self.n_out_classes * batch)) # correct if probability within 0.01
74     # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
75     return accuracies, loss
76 else:
77     return logits
78

```

B.1.16 architectures_baseline_challenge -> baseline_cnn_v2.py (code)

```

1 import torch
2 from torch import nn
3
4 """
5 CNN Layers
6 """
7
8
9 class BaselineNet(nn.Module):
10     def __init__(self, in_channels, out_channels, out_classes, verbose):
11         super().__init__()
12         self.n_out_classes = out_classes
13         self.verbose = verbose
14         self.convs = nn.Sequential(
15             nn.Conv1d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, dilation=1),
16             nn.BatchNorm1d(out_channels),
17             nn.ReLU(),
18             nn.MaxPool1d(3),
19             nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=2),
20             nn.BatchNorm1d(out_channels),
21             nn.ReLU(),
22             nn.MaxPool1d(3),
23             nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=4),
24             nn.BatchNorm1d(out_channels),
25             nn.ReLU(),
26             nn.MaxPool1d(3),
27             nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=8),
28             nn.BatchNorm1d(out_channels),
29             nn.ReLU(),
30             nn.MaxPool1d(3),
31             nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=16),
32             nn.BatchNorm1d(out_channels),
33             nn.ReLU(),
34             nn.MaxPool1d(3),
35         )
36         self.pooling = nn.AdaptiveAvgPool1d(1) # TODO: FIX and try max
37
38         self.fc = nn.Linear(out_channels, out_classes)
39         # self.activation = nn.LogSoftmax(dim=1)
40         # self.criterion = nn.NLLLoss()
41         self.activation = nn.Sigmoid()
42         self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
43
44     def forward(self, X, y=None):
45         if self.verbose: print('input shape', X.shape)
46         batch, window_size, channels = X.shape
47         X = X.transpose(1, 2)
48         if self.verbose: print(X.shape)
49         X = self.convs(X)
50         if self.verbose: print('x shape after convs', X.shape)
51         X = self.pooling(X).squeeze(2)
52         if self.verbose: print('x shape after pooling', X.shape)
53         X = self.fc(X)
54         if self.verbose: print('x shape after fc', X.shape)
55         logits = self.activation(X)
56         if not y is None:
57             # loss = self.criterion(logits, y)
58             loss = torch.sum(torch.square(y - logits)) # Simple own implementation
59             # loss = self.criterion(logits, torch.argmax(y, dim=1))
60             accuracies = []
61             mask = y != 0.0
62             inverse_mask = ~mask
63             zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
64                 inverse_mask) # zero fit goal
65             class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
66             accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
67

```

```

68     accuracies.append(class_fit)
69     accuracies.append(zero_fit)
70
71     accuracies.append(
72         1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
73     values
74     # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
75     # count non zeros in accuracy?
76     accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
77         self.n_out_classes * batch)) # correct if probability within 0.01
78     # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
79     return accuracies, loss
80 else:
81     return logits

```

B.1.17 architectures_baseline_challenge -> baseline_cnn_v3.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=1),
13            nn.BatchNorm2d(out_channels),
14            nn.ReLU(),
15            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=2),
16            nn.BatchNorm2d(out_channels),
17            nn.ReLU(),
18            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=4),
19            nn.BatchNorm2d(out_channels),
20            nn.ReLU(),
21            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=8),
22            nn.BatchNorm2d(out_channels),
23            nn.ReLU(),
24        )
25        self.downsample = nn.Conv2d(in_channels=4470, out_channels=1, kernel_size=1) # 1x1 Conv
26        self.fc = nn.Linear(out_channels, out_classes)
27        # self.activation = nn.LogSoftmax(dim=1)
28        # self.criterion = nn.NLLLoss()
29        self.activation = nn.Sigmoid()
30        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
31
32    def forward(self, X, y=None):
33        if self.verbose: print('input shape', X.shape)
34        batch, window_size, channels = X.shape
35        x = X.transpose(1, 2)
36        if self.verbose: print(x.shape)
37        x = self.convs(x)
38        if self.verbose: print('x shape after conv', x.shape)
39        x = self.downsample(x.transpose(1, 2))
40        if self.verbose: print('x shape after downsample', x.shape)
41        x = self.fc(x).squeeze(1)
42        if self.verbose: print('x shape after fc', x.shape)
43        logits = self.activation(x)
44        if not y is None:
45            # loss = self.criterion(logits, y)
46            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
47            # loss = self.criterion(logits, torch.argmax(y, dim=1))
48            accuracies = []
49            mask = y != 0.0
50            inverse_mask = ~mask
51            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
52                inverse_mask) # zero fit goal
53            class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
54            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
55            accuracies.append(class_fit)
56            accuracies.append(zero_fit)
57
58            accuracies.append(
59                1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
60            values
61            # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
62            # count non zeros in accuracy?
63            accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
64                self.n_out_classes * batch)) # correct if probability within 0.01
65            # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
66            return accuracies, loss
67        else:
68            return logits

```

B.1.18 architectures_baseline_challenge -> baseline_cnn_v4.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv3d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=1),
13            nn.ReLU(),
14            nn.Conv3d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=2),
15            nn.ReLU(),
16            nn.Conv3d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=4),
17            nn.ReLU(),
18            nn.Conv3d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=8),
19            nn.ReLU(),
20        )
21        self.downsample = nn.Conv3d(in_channels=4470, out_channels=1, kernel_size=1) # 1x1 Conv
22        self.fc = nn.Linear(out_channels, out_classes)
23        # self.activation = nn.LogSoftmax(dim=1)
24        # self.criterion = nn.NLLLoss()
25        self.activation = nn.Sigmoid()
26        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
27
28    def forward(self, X, y=None):
29        if self.verbose: print('input shape', X.shape)
30        batch, window_size, channels = X.shape
31        X = X.transpose(1, 2)
32        if self.verbose: print(X.shape)
33        X = self.convs(X)
34        if self.verbose: print('x shape after conv', X.shape)
35        X = X.transpose(1, 2)
36        if self.verbose: print('x shape after transpose', X.shape)
37        X = self.downsample(X)
38        if self.verbose: print('x shape after downsample', X.shape)
39        X = self.fc(X).squeeze(1)
40        if self.verbose: print('x shape after fc', X.shape)
41        logits = self.activation(X)
42        if not y is None:
43            # loss = self.criterion(logits, y)
44            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
45            # loss = self.criterion(logits, torch.argmax(y, dim=1))
46            accuracies = []
47            mask = y != 0.0
48            inverse_mask = ~mask
49            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
50                inverse_mask) # zero fit goal
51            class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
52            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
53            accuracies.append(class_fit)
54            accuracies.append(zero_fit)
55
56            accuracies.append(
57                1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
58            values
59            # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
60            # count non zeros in accuracy?
61            accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
62                self.n_out_classes * batch)) # correct if probability within 0.01
63            # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
64            return accuracies, loss
65        else:
66            return logits

```

B.1.19 architectures_baseline_challenge -> baseline_cnn_v5.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv3d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, dilation=1),
13            nn.ReLU(),
14            nn.Conv3d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=2),
15            nn.ReLU(),

```

```

16     nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=4),
17     nn.ReLU(),
18     nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=8),
19     nn.ReLU(),
20 )
21 self.downsample = nn.Conv1d(in_channels=out_channels, out_channels=1, kernel_size=1)
22
23 self.fc = nn.Linear(4470, out_classes)
# self.activation = nn.LogSoftmax(dim=1)
# self.criterion = nn.NLLLoss()
self.activation = nn.Sigmoid()
self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
28
29 def forward(self, X, y=None):
30     if self.verbose: print('input shape', X.shape)
31     batch, window_size, channels = X.shape
32     x = X.transpose(1, 2)
33     if self.verbose: print(x.shape)
34     x = self.convs(x)
35     if self.verbose: print('x shape after convs', x.shape)
36     x = self.downsample(x).squeeze(1)
37     if self.verbose: print('x shape after downsample', x.shape)
38     x = self.fc(x)
39     if self.verbose: print('x shape after fc', x.shape)
40     logits = self.activation(x)
41     if not y is None:
42         # loss = self.criterion(logits, y)
43         loss = torch.sum(torch.square(y - logits)) # Simple own implementation
44         # loss = self.criterion(logits, torch.argmax(y, dim=1))
45         accuracies = []
46         mask = y != 0.0
47         inverse_mask = ~mask
48         zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
49             inverse_mask) # zero fit goal
50         class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
51         accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
52         accuracies.append(class_fit)
53         accuracies.append(zero_fit)
54
55         accuracies.append(
56             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
values
57     # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
count non zeros in accuracy?
58     accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
59         self.n_out_classes * batch)) # correct if probability within 0.01
60     # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
61     return accuracies, loss
62 else:
63     return logits

```

B.1.20 architectures_baseline_challenge -> baseline_cnn_v6.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv1d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, dilation=1),
13            nn.ReLU(),
14            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=2),
15            nn.ReLU(),
16            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=4),
17            nn.ReLU(),
18            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=8),
19            nn.ReLU(),
20            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=16),
21            nn.ReLU(),
22            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=32),
23            nn.ReLU(),
24        )
25        self.downsample = nn.Conv1d(in_channels=4374, out_channels=1, kernel_size=1)
26
27        self.fc = nn.Linear(out_channels, out_classes)
# self.activation = nn.LogSoftmax(dim=1)
# self.criterion = nn.NLLLoss()
self.activation = nn.Sigmoid()
self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
32
33 def forward(self, X, y=None):

```

```

34     if self.verbose: print('input shape', X.shape)
35     batch, window_size, channels = X.shape
36     x = X.transpose(1, 2)
37     if self.verbose: print(x.shape)
38     x = self.convs(x)
39     if self.verbose: print('x shape after conv', x.shape)
40     x = x.transpose(1, 2)
41     if self.verbose: print('x shape after transpose', x.shape)
42     x = self.downsample(x)
43     if self.verbose: print('x shape after downsample', x.shape)
44     x = self.fc(x).squeeze(1)
45     if self.verbose: print('x shape after fc', x.shape)
46     logits = self.activation(x)
47     if not y is None:
48         # loss = self.criterion(logits, y)
49         loss = torch.sum(torch.square(y - logits)) # Simple own implementation
50         # loss = self.criterion(logits, torch.argmax(y, dim=1))
51         accuracies = []
52         mask = y != 0.0
53         inverse_mask = ~mask
54         zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
55             inverse_mask) # zero fit goal
56         class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
57         accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
58         accuracies.append(class_fit)
59         accuracies.append(zero_fit)
60
61         accuracies.append(
62             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
63         values
64         # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
65         # count non zeros in accuracy?
66         accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
67             self.n_out_classes * batch)) # correct if probability within 0.01
68         # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
69     return accuracies, loss
70 else:
71     return logits

```

B.1.21 architectures_baseline_challenge -> baseline_cnn_v7.py (code)

```

1 import torch
2 from torch import nn
3
4
5 # BAD (hot garbage)
6
7 class BaselineNet(nn.Module):
8
9     def __init__(self, in_channels, out_channels, out_classes, verbose):
10        super().__init__()
11        self.n_out_classes = out_classes
12        self.verbose = verbose
13        self.convs = nn.Sequential(
14            nn.Conv1d(in_channels=in_channels, out_channels=out_channels, kernel_size=7, dilation=1, stride=2),
15            nn.ReLU(),
16            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=2),
17            nn.ReLU(),
18            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=4),
19            nn.ReLU(),
20            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=8),
21            nn.ReLU(),
22            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=16),
23            nn.ReLU(),
24        )
25        self.downsample = nn.Conv1d(in_channels=2187, out_channels=1, kernel_size=1)
26
27        self.fc = nn.Linear(out_channels, out_classes)
28        # self.activation = nn.LogSoftmax(dim=1)
29        # self.criterion = nn.NLLLoss()
30        self.activation = nn.Sigmoid()
31        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
32
33    def forward(self, X, y=None):
34        if self.verbose: print('input shape', X.shape)
35        batch, window_size, channels = X.shape
36        x = X.transpose(1, 2)
37        if self.verbose: print(x.shape)
38        x = self.convs(x)
39        if self.verbose: print('x shape after conv', x.shape)
40        x = x.transpose(1, 2)
41        if self.verbose: print('x shape after transpose', x.shape)
42        x = self.downsample(x)
43        if self.verbose: print('x shape after downsample', x.shape)
44        x = self.fc(x).squeeze(1)
45        if self.verbose: print('x shape after fc', x.shape)

```

```

46     logits = self.activation(x)
47     if not y is None:
48         # loss = self.criterion(logits, y)
49         loss = torch.sum(torch.square(y - logits)) # Simple own implementation
50         # loss = self.criterion(logits, torch.argmax(y, dim=1))
51         accuracies = []
52         mask = y != 0.0
53         inverse_mask = ~mask
54         zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
55             inverse_mask) # zero fit goal
56         class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
57         accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
58         accuracies.append(class_fit)
59         accuracies.append(zero_fit)
60
61         accuracies.append(
62             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
63         values
64         # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
65         # count non zeros in accuracy?
66         accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
67             self.n_out_classes * batch)) # correct if probability within 0.01
68     # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
69     return accuracies, loss
70 else:
71     return logits

```

B.1.22 architectures_baseline_challenge -> baseline_cnn_v8.py (code)

```

1 import torch
2 from torch import nn
3
4
5 # Good
6
7 class BaselineNet(nn.Module):
8
9     def __init__(self, in_channels, out_channels, out_classes, verbose):
10        super().__init__()
11        self.n_out_classes = out_classes
12        self.verbose = verbose
13        self.convs = nn.Sequential(
14            nn.Conv1d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, dilation=1),
15            nn.BatchNorm1d(out_channels),
16            nn.ReLU(),
17            nn.MaxPool1d(3),
18            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=2),
19            nn.BatchNorm1d(out_channels),
20            nn.ReLU(),
21            nn.MaxPool1d(3),
22            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=4),
23            nn.BatchNorm1d(out_channels),
24            nn.ReLU(),
25            nn.MaxPool1d(3),
26            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=8),
27            nn.BatchNorm1d(out_channels),
28            nn.ReLU(),
29        )
30        self.downsample = nn.Conv1d(in_channels=147, out_channels=1, kernel_size=1)
31
32        self.fc = nn.Linear(out_channels, out_classes)
33        # self.activation = nn.LogSoftmax(dim=1)
34        # self.criterion = nn.NLLLoss()
35        self.activation = nn.Sigmoid()
36        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
37
38    def forward(self, X, y=None):
39        if self.verbose: print('input shape', X.shape)
40        batch, window_size, channels = X.shape
41        x = X.transpose(1, 2)
42        if self.verbose: print(x.shape)
43        x = self.convs(x)
44        if self.verbose: print('x shape after conv', x.shape)
45        x = x.transpose(1, 2)
46        if self.verbose: print('x shape after transpose', x.shape)
47        x = self.downsample(x)
48        if self.verbose: print('x shape after downsample', x.shape)
49        x = self.fc(x).squeeze(1)
50        if self.verbose: print('x shape after fc', x.shape)
51        logits = self.activation(x)
52        if not y is None:
53            # loss = self.criterion(logits, y)
54            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
55            # loss = self.criterion(logits, torch.argmax(y, dim=1))
56            accuracies = []
57            mask = y != 0.0

```

```

58     inverse_mask = ~mask
59     zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
60         inverse_mask) # zero fit goal
61     class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
62     accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
63     accuracies.append(class_fit)
64     accuracies.append(zero_fit)
65
66     accuracies.append(
67         1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
68     values
69     # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
70     # count non zeros in accuracy?
71     accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
72         self.n_out_classes * batch)) # correct if probabiltiy within 0.01
73     # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
74     return accuracies, loss
75 else:
76     return logits

```

B.1.23 architectures_baseline_challenge -> baseline_cnn_v9.py (code)

```

1 import torch
2 from torch import nn
3
4
5 # good
6 class BaselineNet(nn.Module):
7
8     def __init__(self, in_channels, out_channels, out_classes, verbose):
9         super().__init__()
10        self.n_out_classes = out_classes
11        self.verbose = verbose
12        self.convs = nn.Sequential(
13            nn.Conv1d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, dilation=1),
14            nn.BatchNorm1d(out_channels),
15            nn.ReLU(),
16            nn.MaxPool1d(3),
17            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=1),
18            nn.BatchNorm1d(out_channels),
19            nn.ReLU(),
20            nn.MaxPool1d(3),
21            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=1),
22            nn.BatchNorm1d(out_channels),
23            nn.ReLU(),
24            nn.MaxPool1d(3),
25            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=1),
26            nn.BatchNorm1d(out_channels),
27            nn.ReLU(),
28        )
29        self.downsample = nn.Conv1d(in_channels=163, out_channels=1, kernel_size=1)
30
31        self.fc = nn.Linear(out_channels, out_classes)
32        # self.activation = nn.LogSoftmax(dim=1)
33        # self.criterion = nn.NLLLoss()
34        self.activation = nn.Sigmoid()
35        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
36
37    def forward(self, X, y=None):
38        if self.verbose: print('input shape', X.shape)
39        batch, window_size, channels = X.shape
40        x = X.transpose(1, 2)
41        if self.verbose: print(x.shape)
42        x = self.convs(x)
43        if self.verbose: print('x shape after conv', x.shape)
44        x = x.transpose(1, 2)
45        if self.verbose: print('x shape after transpose', x.shape)
46        x = self.downsample(x)
47        if self.verbose: print('x shape after downsample', x.shape)
48        x = self.fc(x).squeeze(1)
49        if self.verbose: print('x shape after fc', x.shape)
50        logits = self.activation(x)
51        if not y is None:
52            # loss = self.criterion(logits, y)
53            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
54            # loss = self.criterion(logits, torch.argmax(y, dim=1))
55            accuracies = []
56            mask = y != 0.0
57            inverse_mask = ~mask
58            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
59                inverse_mask) # zero fit goal
60            class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
61            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
62            accuracies.append(class_fit)
63            accuracies.append(zero_fit)
64

```

```

65         accuracies.append(
66             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
67             values
68             # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
69             count non zeros in accuracy?
70             accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
71                 self.n_out_classes * batch)) # correct if probability within 0.01
72             # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
73             return accuracies, loss
74     else:
75         return logits

```

B.1.24 architectures_baseline_challenge -> baseline_convencoder.py (code)

```

1 import torch
2 import torch.nn.functional as F
3 from torch import nn
4
5
6 class BaselineNet(nn.Module):
7
8     def __init__(self, in_channels, latent_size, out_classes):
9         super().__init__()
10        self.n_out_classes = out_classes
11        self.convs = nn.Sequential(
12            nn.Conv2d(in_channels=in_channels, out_channels=latent_size, kernel_size=5, stride=4),
13            nn.BatchNorm2d(latent_size),
14            nn.ReLU(),
15            nn.Conv2d(in_channels=latent_size, out_channels=latent_size, kernel_size=4, stride=3),
16            nn.BatchNorm2d(latent_size),
17            nn.ReLU(),
18            nn.Conv2d(in_channels=latent_size, out_channels=latent_size, kernel_size=3, stride=2),
19            nn.BatchNorm2d(latent_size),
20            nn.ReLU(),
21            nn.Conv2d(in_channels=latent_size, out_channels=latent_size, kernel_size=3, stride=2),
22            nn.BatchNorm2d(latent_size),
23            nn.ReLU(),
24            nn.Conv2d(in_channels=latent_size, out_channels=latent_size, kernel_size=2, stride=1),
25            nn.BatchNorm2d(latent_size),
26            nn.ReLU(),
27        )
28        self.pooling = nn.AdaptiveAvgPool2d(1)
29        self.t_convs = nn.Sequential(
30            nn.ConvTranspose2d(in_channels=latent_size, out_channels=latent_size, kernel_size=2, stride=1,
31                             output_padding=0),
32            nn.BatchNorm2d(latent_size),
33            nn.ReLU(),
34            nn.ConvTranspose2d(in_channels=latent_size, out_channels=latent_size, kernel_size=3, stride=2,
35                             output_padding=1),
36            nn.BatchNorm2d(latent_size),
37            nn.ReLU(),
38            nn.ConvTranspose2d(in_channels=latent_size, out_channels=latent_size, kernel_size=3, stride=2,
39                             output_padding=1),
40            nn.BatchNorm2d(latent_size),
41            nn.ReLU(),
42            nn.ConvTranspose2d(in_channels=latent_size, out_channels=latent_size, kernel_size=4, stride=3,
43                             output_padding=0),
44            nn.BatchNorm2d(latent_size),
45            nn.ReLU(),
46            nn.ConvTranspose2d(in_channels=latent_size, out_channels=in_channels, kernel_size=5, stride=4,
47                             output_padding=3),
48            nn.BatchNorm2d(in_channels),
49            nn.ReLU(),
50        )
51        self.fc = nn.Linear(latent_size, out_classes)
52        # self.activation = nn.LogSoftmax(dim=1)
53        # self.criterion = nn.NLLLoss()
54        self.activation = nn.Sigmoid()
55        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
56
57    def forward(self, X, y=None):
58        X = X.squeeze(1)
59        # print('input shape', X.shape)
60        batch, window_size, channels = X.shape
61        # X = X.transpose(1, 2)
62        X = X.clone()
63        # print('x original shape', X.shape)
64        X = self.convs(X)
65        # print('x shape after convs', X.shape)
66        unpool_dim = X.shape[-1]
67        X = self.pooling(X)
68        print('x shape after pooling', X.shape)
69
70        if not y is None:
71            X = X.squeeze(-1)
72            X = self.fc(X)

```

```

73     # print('x shape after fc', x.shape)
74     logits = self.activation(x)
75
76     # loss = self.criterion(logits, y)
77     loss = torch.sum(torch.square(y - logits))  # Simple own implementation
78     # loss = self.criterion(logits, torch.argmax(y, dim=1))
79     accuracies = []
80     mask = y != 0.0
81     inverse_mask = ~mask
82     zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
83         inverse_mask)  # zero fit goal
84     class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask)  # class fit goal
85     accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
86     accuracies.append(class_fit)
87     accuracies.append(zero_fit)
88
89     accuracies.append(
90         1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch))  # Distance between all
values
91     # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
count non zeros in accuracy?
92     accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
93         self.n_out_classes * batch))  # correct if probability within 0.01
94     # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
95     return accuracies, loss
96 else:
97     reps = [1] * len(x.shape)
98     reps[-1] = unpool_dim
99     x = x.repeat(*reps)  # unpool avg layer
100    x = F.conv_transpose1d(x)
101    # print('x shape after unpool', x.shape)
102    x = self.t_convs(x)
103    # print('x shape after t_convs', x.shape)
104    loss = torch.sum(torch.square(X - x))  # SE
105    accuracies = []
106    accuracies.append(torch.tensor(0.0).cuda())
107    return accuracies, loss

```

B.1.25 architectures_baseline_challenge -> baseline_resnet.py (code)

```

1 import torch
2 from torch import nn
3
4
5 # Time Series Classification from Scratch with Deep
6 # Neural Networks: A Strong Baseline https://arxiv.org/pdf/1611.06455.pdf
7 class BaselineNet(nn.Module):
8
9     def __init__(self, in_channels, out_channels, out_classes=94, verbose=False) -> None:
10         super(BaselineNet, self).__init__()
11         self.verbose = verbose
12
13         self.conv1_1 = nn.Conv2d(in_channels=in_channels, out_channels=64, kernel_size=8, padding=4)
14         self.bn1_1 = nn.BatchNorm2d(num_features=64)
15         self.conv1_2 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=5, padding=2)
16         self.bn1_2 = nn.BatchNorm2d(num_features=64)
17         self.conv1_3 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1)
18         self.bn1_3 = nn.BatchNorm2d(num_features=64)
19
20         self.conv2_1 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=8, padding=4)
21         self.bn2_1 = nn.BatchNorm2d(num_features=128)
22         self.conv2_2 = nn.Conv2d(in_channels=128, out_channels=128, kernel_size=5, padding=2)
23         self.bn2_2 = nn.BatchNorm2d(num_features=128)
24         self.conv2_3 = nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1)
25         self.bn2_3 = nn.BatchNorm2d(num_features=128)
26
27         self.conv3_1 = nn.Conv2d(in_channels=128, out_channels=out_classes, kernel_size=8, padding=4)
28         self.bn3_1 = nn.BatchNorm2d(num_features=out_classes)
29         self.conv3_2 = nn.Conv2d(in_channels=out_classes, out_channels=out_classes, kernel_size=5, padding=2)
30         self.bn3_2 = nn.BatchNorm2d(num_features=out_classes)
31         self.conv3_3 = nn.Conv2d(in_channels=out_classes, out_channels=out_classes, kernel_size=3, padding=1)
32         self.bn3_3 = nn.BatchNorm2d(num_features=out_classes)
33
34         self.relu = nn.ReLU()
35         self.activation = nn.Sigmoid()
36
37     def forward(self, x: torch.Tensor, y=None) -> torch.Tensor:
38         x = x.transpose(1, 2)
39         print(x.shape)
40         identity = x
41         x = self.conv1_1(x)
42         x = self.bn1_1(x)
43         x = self.relu(x)
44         x = self.conv1_2(x)
45         x = self.bn1_2(x)
46         x = self.relu(x)

```

```

47     x = self.conv1_3(x)
48     x = self.bn1_3(x)
49     x = self.relu(x + identity)
50     print(x.shape)
51     identity = x
52     x = self.conv2_1(x)
53     x = self.bn2_1(x)
54     x = self.relu(x)
55     x = self.conv2_2(x)
56     x = self.bn2_2(x)
57     x = self.relu(x)
58     x = self.conv2_3(x)
59     x = self.bn2_3(x)
60     x = self.relu(x + identity)
61     print(x.shape)
62     identity = x
63     x = self.conv3_1(x)
64     x = self.bn3_1(x)
65     x = self.relu(x)
66     x = self.conv3_2(x)
67     x = self.bn3_2(x)
68     x = self.relu(x)
69     x = self.conv3_3(x)
70     x = self.bn3_3(x)
71     x = self.relu(x + identity)
72     print(x.shape)
73     x = torch.mean(x, dim=-1)
74     print(x.shape)
75     x = self.activation(x)
76     return x

```

B.1.26 architectures_baseline_challenge -> baseline_rnn.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6     def __init__(self, in_channels, out_channels, out_classes=94, verbose=False):
7         super(BaselineNet, self).__init__()
8         self.features = nn.Sequential(
9             nn.Conv2d(in_channels, 64, kernel_size=11, stride=4, padding=2),
10            nn.ReLU(inplace=True),
11            nn.MaxPool2d(kernel_size=3, stride=2),
12            nn.Conv2d(64, 192, kernel_size=5, padding=2),
13            nn.ReLU(inplace=True),
14            nn.MaxPool2d(kernel_size=3, stride=2),
15            nn.Conv2d(192, 384, kernel_size=3, padding=1),
16            nn.ReLU(inplace=True),
17            nn.Conv2d(384, 256, kernel_size=3, padding=1),
18            nn.ReLU()
19        )
20        self.lstm = nn.LSTM(input_size=256, hidden_size=256, num_layers=1)
21        self.classifier = nn.Sequential(
22            nn.Linear(256, out_classes),
23        )
24        self.activation = nn.Sigmoid()
25
26     def forward(self, x: torch.Tensor, y=None) -> torch.Tensor:
27         x = x.transpose(1, 2)
28         x = self.features(x)
29         x = torchmovedim(x, -1, 0)
30         x, c_nh_n = self.lstm(x) # no hidden state so its 0, take last output
31         x = x[-1]
32         x = self.classifier(x)
33         x = self.activation(x)
34         return x

```

B.1.27 architectures_baseline_challenge -> baseline_rnn_simplest_gru.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6     def __init__(self, in_channels, out_channels, out_classes=94, verbose=False):
7         super(BaselineNet, self).__init__()
8         self.gru = nn.GRU(input_size=in_channels, hidden_size=256, num_layers=1, batch_first=True)
9         self.classifier = nn.Sequential(
10            nn.ReLU(),
11            nn.Linear(256, out_classes),
12        )

```

```

13     self.activation = nn.Sigmoid()
14
15 def forward(self, x: torch.Tensor, y=None) -> torch.Tensor:
16     x, c_nh_n = self.gru(x) # no hidden state so its 0, take last output
17     x = x[:, -1] # batch first so seq is at index 1
18     x = self.classifier(x)
19     x = self.activation(x)
20
21     return x

```

B.1.28 architectures_baseline_challenge -> baseline_rnn_simplest_lstm.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6     def __init__(self, in_channels, out_channels, out_classes=94, verbose=False):
7         super(BaselineNet, self).__init__()
8         self.lstm = nn.LSTM(input_size=in_channels, hidden_size=128, num_layers=1, batch_first=True)
9         self.classifier = nn.Sequential(
10             nn.ReLU(),
11             nn.Linear(128, out_classes),
12         )
13     self.activation = nn.Sigmoid()
14
15 def forward(self, x: torch.Tensor, y=None) -> torch.Tensor:
16     x, c_nh_n = self.lstm(x) # no hidden state so its 0, take last output
17     x = x[:, -1] # batch first so seq is at index 1
18     x = self.classifier(x)
19     x = self.activation(x)
20
21     return x

```

B.2 architectures_cpc (folder)**B.2.1 architectures_cpc -> cpc_autoregressive_hidden.py (code)**

```

1 import torch
2 from torch import nn
3
4
5 class AutoRegressor(nn.Module):
6     def __init__(self, n_latents, hidden_size, layers=1):
7         super(AutoRegressor, self).__init__()
8         self.n_latents = n_latents
9         self.hidden_size = hidden_size
10        self.layers = layers
11        self.gru = nn.GRU(input_size=self.n_latents, hidden_size=self.hidden_size, num_layers=self.layers)
12
13    def _weights_init(m):
14        for layer_p in self.gru._all_weights:
15            for p in layer_p:
16                if 'weight' in p:
17                    nn.init.kaiming_normal_(self.gru.__getattr__(p), mode='fan_out', nonlinearity='relu')
18
19    self.apply(_weights_init)
20
21    def init_hidden(self, batch_size, use_gpu=True):
22        if use_gpu:
23            return torch.zeros(self.layers, batch_size, self.hidden_size).cuda()
24        else:
25            return torch.zeros(self.layers, batch_size, self.hidden_size).cpu()
26
27    def forward(self, x, hidden):
28        # Input is (seq, batch, latents) maybe (13, 8, 128)
29        # print('regressor input shape:', x.shape, hidden.shape)
30        x, hidden = self.gru(x, hidden)
31        # print('regressor output shape:', x.shape, hidden.shape)
32        return hidden, x #WARNING! return order has been changed!

```

B.2.2 architectures_cpc -> cpc_autoregressive_v0.py (code)

```

1 import torch
2 from torch import nn
3
4

```

```

5 class AutoRegressor(nn.Module):
6     def __init__(self, n_latents, hidden_size, layers=1):
7         super(AutoRegressor, self).__init__()
8         self.n_latents = n_latents
9         self.hidden_size = hidden_size
10        self.layers = layers
11        self.gru = nn.GRU(input_size=self.n_latents, hidden_size=self.hidden_size, num_layers=self.layers)
12
13    def _weights_init(m):
14        for layer_p in self.gru._all_weights:
15            for p in layer_p:
16                if 'weight' in p:
17                    nn.init.kaiming_normal_(self.gru.__getattr__(p), mode='fan_out', nonlinearity='relu')
18
19    self.apply(_weights_init)
20
21    def init_hidden(self, batch_size, use_gpu=True):
22        if use_gpu:
23            return torch.zeros(self.layers, batch_size, self.hidden_size).cuda()
24        else:
25            return torch.zeros(self.layers, batch_size, self.hidden_size).cpu()
26
27    def forward(self, x, hidden):
28        # Input is (seq, batch, latents) maybe (13, 8, 128)
29        # print('regressor input shape:', x.shape, hidden.shape)
30        x, hidden = self.gru(x, hidden)
31        # print('regressor output shape:', x.shape, hidden.shape)
32        return x, hidden

```

B.2.3 architectures_cpc -> cpc_base.py (code)

```

1 import numpy as np
2 import torch
3 from torch import nn
4 from torch.nn import functional as F
5
6
7 class CPC(nn.Module):
8     def __init__(self, encoder_model, autoregressive_model, predictor_model, latent_size, timesteps_in, timesteps_out,
9                  timesteps_ignore=0, verbose=False):
10        super(CPC, self).__init__()
11        self.timesteps_in = timesteps_in
12        self.timesteps_out = timesteps_out
13        self.latent_size = latent_size
14        self.encoder = encoder_model
15        self.autoregressive = autoregressive_model
16        self.predictor = predictor_model
17        # self.softmax = nn.Softmax(dim=1)
18        self.lsoftmax = nn.LogSoftmax(dim=1)
19        self.cpc_train_mode = True
20        self.timesteps_in = timesteps_in
21        self.timesteps_out = timesteps_out
22
23        self.verbose = verbose
24
25    def forward(self, x_windows, hidden):
26        if self.verbose: print('x_windows has shape:', x_windows.shape) # shape = batch, windows, channels,
27        window_size
28        x_windows = x_windows.transpose(1, 0) # reshaping into windows, batch, channels, windowsize
29        n_windows, n_batches, _, _ = x_windows.shape
30        if self.verbose: print('x_windows has shape:', x_windows.shape)
31        if self.cpc_train_mode and n_windows != self.timesteps_in + self.timesteps_out:
32            print("timesteps in and out not matching total windows")
33        latents = self.encoder(
34            x_windows.reshape(-1, *x_windows.shape[2:])) # reshape windows into batch dimension for only one forward
35        latents = latents.reshape(n_windows, n_batches, *latents.shape[1:]) # reshape shape back
36        if self.verbose: print('latents have shape:', latents.shape) # shape = windows, batch, latent_size
37        context, hidden = self.autoregressive(latents[0:self.timesteps_in, :, :], hidden)
38        context = context[-1, :, :] # We only need the last state. Shape: batch, context_outputszie
39        if not self.cpc_train_mode:
40            return latents, context, hidden
41
42        loss = torch.Tensor([0.0]).cuda()
43        correct = 0 # will become Tensor
44        for k in range(0, self.timesteps_out): # Do this for each timestep
45            latent_k = latents[-self.timesteps_out + k] # batches, latents
46            pred_k = self.predictor(context, k) # Shape (Batches, latents)
47            # pred_k[0].T @ latent_k
48            # loss += torch.log(torch.exp(pred_k[0].T @ latent_k[0]))
49            softmax = self.lsoftmax(torch.mm(latent_k, pred_k.T)) # output: (Batches, Batches)
50            correct += torch.sum(torch.argmax(softmax, dim=0) == torch.arange(n_batches).cuda())
51            loss += torch.sum(torch.diag(softmax))
52
53        loss /= (n_batches * self.timesteps_out) * -1.0
54        accuracy = correct.true_divide(n_batches * self.timesteps_out)
55        return accuracy, loss, hidden

```

```

55     # Calculate the loss
56     # future_latents = []
57     # predicted_latents = []
58     # for k in range(0, self.timesteps_out): # Do this for each timestep
59     #     future_latents.append(self.encoder(x_windows[-self.timesteps_out + k])) # batches, latents
60     #     predicted_latents.append(self.predictor(context, k)) # Shape (Batches, latents)
61     # loss = self.info_NCE_loss_brian(torch.stack(future_latents), torch.stack(predicted_latents))
62     # #Will become Tensor
63     # return torch.Tensor([0.0]).cuda(), loss, hidden
64     def info_NCE_loss_brian(self, target_latents: torch.Tensor, pred_latents: torch.Tensor, emb_scale=0):
65         """
66             Calculates the infoNCE loss for CPC according to 'DATA-EFFICIENT IMAGE RECOGNITION
67             WITH CONTRASTIVE PREDICTIVE CODING' A.2
68             # latents: [B, H, W, D]
69             loss = 0 . 0
70             context = pixelCNN(latents)
71             targets = Conv2D(output_channels=target_dim ,kernel_shape= ( 1 , 1 ) ) (latents)
72
73             batch_dim , rows = targets . shape [ : - 1 ]
74             targets = reshape(targets , [ - 1 , target_dim ] )
75             for i in range(steps_to_ignore , steps_to_predict) :
76                 total_elements = batch_dim * rows
77                 preds_i = Conv2D(output_channels=target_dim , kernel_shape= ( 1 , 1 ) ) (context)
78                 preds_i = preds_i [ : , : - (i+ 1 ) , : , : ] * emb_scale
79                 preds_i = reshape(preds_i , [ - 1 , target_dim ] )
80                 logits = matmul(preds_i , targets , transp_b=True)
81                 b = range(total_elements) / (rows)
82                 col = range(total_elements) % (rows)
83                 labels = b * rows + (i+1) * rows + col
84                 loss += cross_entropy_with_logits(logits , labels)
85             return loss
86         """
87         loss = torch.Tensor([0.0]).cuda()
88         # latents = latents.permute(1, 0, 2)
89         # batch_dim, col_dim, row_dim, _ = latents.shape
90         timesteps_out, batch_dim, _ = target_latents.shape
91         # targets = target_latents.transpose(1, 0)
92         targets = target_latents.view([-1, self.latent_size]) # reshape
93         # print(targets.shape)
94         for i in range(timesteps_out):
95             total_elements = batch_dim # * timesteps_out
96             logits = torch.matmul(pred_latents[i], torch.t(targets[i])) # torch.t(targets)
97             b = torch.arange(total_elements).cuda()
98             col = b % timesteps_out
99             labels = b + (i + 1) * timesteps_out + col
100            # print(labels)
101            temp_loss = torch.sum(- labels * F.log_softmax(logits, -1),
102                                  -1) # From https://gist.github.com/tejaskhot/cf3d087ce4708c422e68b3b747494b9f
103            mean_loss = temp_loss.mean()
104            loss += mean_loss
105        return loss
106
107    def freeze_layers(self):
108        for param in self.parameters():
109            param.requires_grad = False
110
111    def unfreeze_layers(self):
112        for param in self.parameters():
113            param.requires_grad = True
114
115    def set_train_mode(self, mode_bool):
116        self.cpc_train_mode = mode_bool
117
118    def test_modules(self, in_channels, window_size, latent_size, timesteps_in, timesteps_out):
119        self._test_modules(1, in_channels, window_size, latent_size, timesteps_in, timesteps_out)
120        for r in np.random.randint(2, 100, 10):
121            self._test_modules(r, in_channels, window_size, latent_size, timesteps_in, timesteps_out)
122        self._test_modules(128, in_channels, window_size, latent_size, timesteps_in, timesteps_out, verbose=True)
123
124    def _test_modules(self, batch_size, in_channels, window_size, latent_size, timesteps_in, timesteps_out,
125                      verbose=False):
126        enc_in = torch.rand(batch_size, in_channels, window_size)
127        if verbose: print('Encoder input shape:', enc_in.shape)
128        enc_out = self.encoder(enc_in)
129        if verbose: print('Encoder output shape:', enc_out.shape)
130        auto_in = torch.rand(timesteps_in) + enc_in.shape
131        if verbose: print('Autoregressive input shape (stacked encoder):', auto_in.shape)
132        auto_out = self.autoregressive(auto_in)
133        if verbose: print('Autoregressive output shape:', auto_out.shape)
134        pred_in = auto_out[-1, :, :]
135        if verbose: print('Predictor input shape:', pred_in.shape)
136        pred_out = self.predictor(pred_in, timesteps_out)
137        for i in range(1, timesteps_out):
138            pred_out += self.predictor(pred_in, timesteps_out)
139        if verbose: print('Predictor output shape:', pred_out.shape)
140        assert enc_out.shape == pred_out.shape

```

B.2.4 architectures_cpc -> cpc_combined.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class CPCCombined(nn.Module):
6     def __init__(self, cpc_model, downstream_model, freeze_cpc=True):
7         super().__init__()
8         self.cpc_model = cpc_model
9         self.downstream_model = downstream_model
10        self.freeze_cpc = freeze_cpc
11        self.requires_grad_(True)
12        self._unfreeze_cpc()
13        if self.freeze_cpc:
14            self._freeze_cpc()
15
16    def forward(self, X, y=None):
17        if self.cpc_model.cpc_train_mode:
18            self.cpc_model.cpc_train_mode = False
19        if self.freeze_cpc:
20            if not self.is_frozen:
21                self._freeze_cpc()
22            with torch.no_grad():
23                encoded_x, context, hidden = self.cpc_model(X)
24        else:
25            if self.is_frozen:
26                self._unfreeze_cpc()
27            encoded_x, context, hidden = self.cpc_model(X)
28        return self.downstream_model(encoded_x, context, y=y)
29
30    def pretrain(self, X, y=None, hidden=None):
31        if not self.cpc_model.cpc_train_mode:
32            self.cpc_model.cpc_train_mode = True
33        return self.cpc_model(X, y, hidden)
34
35    def _freeze_cpc(self):
36        self.is_frozen = True
37        for m in self.cpc_model.children():
38            for param in m.parameters():
39                param.requires_grad = False
40        for param in self.cpc_model.parameters():
41            param.requires_grad = False
42
43    def _unfreeze_cpc(self):
44        self.is_frozen = False
45        for m in self.cpc_model.children():
46            for param in m.parameters():
47                param.requires_grad = True
48        for param in self.cpc_model.parameters():
49            param.requires_grad = True

```

B.2.5 architectures_cpc -> cpc_downstream_cnn.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class DownstreamLinearNet(nn.Module):
6
7     def __init__(self, latent_size, context_size, out_classes, use_context=True, use_latents=False, verbose=False):
8         super().__init__()
9         self.latent_size = latent_size
10        self.n_out_classes = out_classes
11        self.use_context = use_context
12        self.use_latents = use_latents
13        self.verbose = verbose
14        if self.use_context and self.use_latents:
15            self.classifier = nn.Sequential(
16                nn.ReLU(),
17                nn.Conv1d(in_channels=context_size * 2, out_channels=out_classes, kernel_size=1, stride=1),
18            )
19        else: # if not both are getting used, normal classifier is enough for each (in_channels*1)
20            self.classifier = nn.Sequential(
21                nn.ReLU(),
22                nn.Conv1d(in_channels=context_size, out_channels=out_classes, kernel_size=1, stride=1),
23            )
24        if self.use_latents:
25            self.rnn = nn.GRU(input_size=latent_size, hidden_size=context_size, num_layers=1, batch_first=False)
26
27        self.activation = nn.Sigmoid()
28
29    def forward(self, latents=None, context=None, y=None):
30        if self.verbose and latents: print('latents', latents.shape)
31        if self.verbose and context: print('context', context.shape)
32        if self.use_context and self.use_latents:

```

```

33         x, c = self.rnn(latents)
34         context = torch.cat([context, x[-1, :]], dim=-1)
35     elif self.use_latents:
36         x, c = self.rnn(latents)
37         context = x[-1, :]
38     pred = self.classifier(context.unsqueeze(-1))
39     output = self.activation(pred).squeeze(-1) # do not squeeze on batch?
40     # print('pred shape', pred.shape)
41     if y is None: #
42         return output
43     else: # Training mode return loss instead of prediction
44         raise NotImplementedError

```

B.2.6 architectures_cpc -> cpc_downstream_latent_average.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class DownstreamLinearNet(nn.Module):
6
7     def __init__(self, latent_size, context_size, out_classes, use_context=True, use_latents=False, verbose=False):
8         super().__init__()
9         self.latent_size = latent_size
10        self.n_out_classes = out_classes
11        self.use_context = use_context
12        self.use_latents = use_latents
13        self.verbose = verbose
14        if self.use_latents:
15            self.classifier_l = nn.Sequential(
16                nn.ReLU(),
17                nn.Linear(in_features=latent_size, out_features=out_classes)
18            )
19        if self.use_context: # if not both are getting used, normal classifier is enough for each (in_channels*1)
20            self.classifier_c = nn.Sequential(
21                nn.ReLU(),
22                nn.Linear(in_features=context_size, out_features=out_classes),
23            )
24
25        self.activation = nn.Sigmoid()
26
27    def forward(self, latents=None, context=None, y=None):
28        if self.verbose and latents: print('latents', latents.shape)
29        if self.verbose and context: print('context', context.shape)
30        if self.use_context and self.use_latents:
31            predc = self.classifier_c(context)
32            predl = self.classifier_l(latents)
33            predl = torch.mean(predl, dim=0)
34            pred = (predc+predl)/2
35        elif self.use_latents:
36            pred = self.classifier_l(latents)
37            pred = torch.mean(pred, dim=0)
38        else:
39            pred = self.classifier_c(context)
40        output = self.activation(pred) # do not squeeze on batch?
41        # print('pred shape', pred.shape)
42        if y is None: #
43            return output
44        else: # Training mode return loss instead of prediction
45            raise NotImplementedError
46

```

B.2.7 architectures_cpc -> cpc_downstream_latent_maximum.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class DownstreamLinearNet(nn.Module):
6
7     def __init__(self, latent_size, context_size, out_classes, use_context=True, use_latents=False, verbose=False):
8         super().__init__()
9         self.latent_size = latent_size
10        self.n_out_classes = out_classes
11        self.use_context = use_context
12        self.use_latents = use_latents
13        self.verbose = verbose
14        if self.use_latents:
15            self.classifier_l = nn.Sequential(
16                nn.ReLU(),
17                nn.Linear(in_features=latent_size, out_features=out_classes)
18            )

```

```

19     if self.use_context: # if not both are getting used, normal classifier is enough for each (in_channels*1)
20         self.classifier_c = nn.Sequential(
21             nn.ReLU(),
22             nn.Linear(in_features=context_size, out_features=out_classes),
23         )
24
25     self.activation = nn.Sigmoid()
26
27     def forward(self, latents=None, context=None, y=None):
28         if self.verbose and latents: print('latents', latents.shape)
29         if self.verbose and context: print('context', context.shape)
30         if self.use_context and self.use_latents:
31             predc = self.classifier_c(context)
32             predl = self.classifier_l(latents)
33             predl = torch.max(predl, dim=0).values
34             pred = (predc+predl)/2
35         elif self.use_latents:
36             pred = self.classifier_l(latents)
37             pred = torch.max(pred, dim=0).values
38         else:
39             pred = self.classifier_c(context)
40         output = self.activation(pred) # do not squeeze on batch?
41         # print('pred shape', pred.shape)
42         if y is None: #
43             return output
44         else: # Training mode return loss instead of prediction
45             raise NotImplementedError

```

B.2.8 architectures_cpc -> cpc_downstream_model_multitarget_v1.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class DownstreamLinearNet(nn.Module):
6
7     def __init__(self, cpc_model, latent_size, context_size, out_classes, use_context=True, use_latents=False,
8                  freeze=True, verbose=False):
9         super().__init__()
10        self.latent_size = latent_size
11        self.n_out_classes = out_classes
12        self.cpc_model = cpc_model
13        if freeze:
14            self.cpc_model.freeze_layers()
15        else:
16            self.cpc_model.unfreeze_layers()
17        self.cpc_model.cpc_train_mode = False
18        self.use_context = use_context
19        self.use_latents = use_latents
20        self.verbose = verbose
21        if self.use_context:
22            self.linear_context = nn.Linear(context_size, out_classes)
23        if self.use_latents:
24            self.flatten = nn.Flatten()
25            self.linear_latents = nn.Linear(57 * latent_size, out_classes)
26        self.activation = nn.Sigmoid()
27        # self.loss = nn.KLDivLoss()
28        self.loss = nn.MSELoss()
29
30    def forward(self, X, cpc_hidden=None, y=None):
31        latents, context, cpc_hidden = self.cpc_model(X, hidden=cpc_hidden) # e.g. shape
32        if self.verbose: print('latents', latents.shape, 'context', context.shape)
33        if self.use_context and self.use_latents: # some kind of ensemble?
34            pred = (self.linear_context(context) + self.linear_latents(self.flatten(latents.transpose(0, 1)))) / 2.0
35        elif self.use_context:
36            pred = self.linear_context(context)
37        elif self.use_latents:
38            pred = self.linear_latents(self.flatten(latents.transpose(1, 0)))
39        output = self.activation(pred)
40        # print('pred shape', pred.shape)
41        if not y is None: # Training mode return loss instead of prediction
42            batch, n_classes = y.shape
43            loss = self.loss(pred, y)
44            accuracies = []
45            mask = y != 0.0
46            inverse_mask = ~mask
47            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - output[inverse_mask])) / torch.sum(
48                inverse_mask) # zero fit goal
49            class_fit = 1.0 - torch.sum(torch.square(y[mask] - output[mask])) / torch.sum(mask) # class fit goal
50            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
51            accuracies.append(class_fit)
52            accuracies.append(zero_fit)
53
54            accuracies.append(
55                1.0 - torch.sum(torch.square(y - output)) / (self.n_out_classes * batch)) # Distance between all
values

```

```

56         # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
57         count non zeros in accuracy?
58         accuracies.append(torch.sum(torch.absolute(y - output) <= 0.01) / (
59             self.n_out_classes * batch)) # correct if probability within 0.01
60         # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
61         return accuracies, loss
62
63     def init_hidden(self, batch_size, use_gpu):
64         return self.cpc_model.init_hidden(batch_size, use_gpu)

```

B.2.9 architectures_cpc -> cpc_downstream_model_multitarget_v2.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class DownstreamLinearNet(nn.Module):
6
7     def __init__(self, cpc_model, latent_size, context_size, out_classes, use_context=True, use_latents=False,
8                  freeze=True, verbose=False):
9         super().__init__()
10        self.latent_size = latent_size
11        self.n_out_classes = out_classes
12        self.cpc_model = cpc_model
13        if freeze:
14            self.cpc_model.freeze_layers()
15        else:
16            self.cpc_model.unfreeze_layers()
17        self.cpc_model.cpc_train_mode = False
18        self.use_context = use_context
19        self.use_latents = use_latents
20        self.verbose = verbose
21        if self.use_context:
22            self.linear_context = nn.Sequential(
23                nn.Linear(context_size, context_size),
24                nn.Linear(context_size, out_classes),
25            )
26        if self.use_latents:
27            self.flatten = nn.Flatten()
28            self.linear_latents = nn.Sequential(
29                nn.Linear(57 * latent_size, context_size),
30                nn.Linear(context_size, out_classes),
31            )
32        self.activation = nn.Sigmoid()
33        # self.loss = nn.KLDivLoss()
34        self.loss = nn.MSELoss()
35
36    def forward(self, X, cpc_hidden=None, y=None):
37        latents, context, cpc_hidden = self.cpc_model(X, hidden=cpc_hidden) # e.g shape
38        if self.verbose: print('latents', latents.shape, 'context', context.shape)
39        if self.use_context and self.use_latents: # some kind of ensemble?
40            pred = (self.linear_context(context) + self.linear_latents(self.flatten(latents.transpose(0, 1)))) / 2.0
41        elif self.use_context:
42            pred = self.linear_context(context)
43        elif self.use_latents:
44            pred = self.linear_latents(self.flatten(latents.transpose(1, 0)))
45        output = self.activation(pred)
46        # print('pred shape', pred.shape)
47        if not y is None: # Training mode return loss instead of prediction
48            batch, n_classes = y.shape
49            loss = self.loss(pred, y)
50            accuracies = []
51            mask = y != 0.0
52            inverse_mask = ~mask
53            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - output[inverse_mask])) / torch.sum(
54                inverse_mask) # zero fit goal
55            class_fit = 1.0 - torch.sum(torch.square(y[mask] - output[mask])) / torch.sum(mask) # class fit goal
56            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
57            accuracies.append(class_fit)
58            accuracies.append(zero_fit)
59
60            accuracies.append(
61                1.0 - torch.sum(torch.square(y - output)) / (self.n_out_classes * batch)) # Distance between all
62            values
63            # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
64            count non zeros in accuracy?
65            accuracies.append(torch.sum(torch.absolute(y - output) <= 0.01) / (
66                self.n_out_classes * batch)) # correct if probability within 0.01
67            # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
68            return accuracies, loss
69
70    def init_hidden(self, batch_size, use_gpu):
71        return self.cpc_model.init_hidden(batch_size, use_gpu)

```

B.2.10 architectures_cpc -> cpc_downstream_only.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class DownstreamLinearNet(nn.Module):
6
7     def __init__(self, latent_size, context_size, out_classes, use_context=True, use_latents=False, verbose=False):
8         super().__init__()
9         self.latent_size = latent_size
10        self.n_out_classes = out_classes
11        self.use_context = use_context
12        self.use_latents = use_latents
13        self.verbose = verbose
14        if self.use_context:
15            self.linear_context = nn.Linear(context_size, out_classes)
16        if self.use_latents:
17            self.output_size = 1
18            self.pool = nn.AdaptiveMaxPool1d(output_size=self.output_size)
19            self.linear_latents = nn.Sequential(
20                nn.Flatten(),
21                nn.Linear(self.output_size * latent_size, out_classes),
22            )
23        self.activation = nn.Sigmoid()
24        # self.loss = nn.KLDivLoss()
25        self.loss = nn.MSELoss()
26
27    def forward(self, latents=None, context=None, y=None):
28        if self.verbose and latents: print('latents', latents.shape)
29        if self.verbose and context: print('context', context.shape)
30        if self.use_context and self.use_latents: # some kind of ensemble?
31            pred = (self.linear_context(context) + self.linear_latents(self.flatten(latents.transpose(0, 1)))) / 2.0
32        elif self.use_context:
33            pred = self.linear_context(context)
34        elif self.use_latents:
35            pooled = self.pool(latents.T).permute(1, 0,
36                                              -1) # Reduce outsteps to outputsize and reshape to batch, latents,
37            pred = self.linear_latents(pooled)
38        output = self.activation(pred)
39        # print('pred shape', pred.shape)
40        if y is None: #
41            return output
42        else: # Training mode return loss instead of prediction
43            batch, n_classes = y.shape
44            loss = self.loss(pred, y)
45            accuracies = []
46            mask = y != 0.0
47            inverse_mask = ~mask
48            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - output[inverse_mask])) / torch.sum(
49                inverse_mask) # zero fit goal
50            class_fit = 1.0 - torch.sum(torch.square(y[mask] - output[mask])) / torch.sum(mask) # class fit goal
51            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
52            accuracies.append(class_fit)
53            accuracies.append(zero_fit)
54
55            accuracies.append(
56                1.0 - torch.sum(torch.square(y - output)) / (self.n_out_classes * batch)) # Distance between all
57            values
58            # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
59            # count non zeros in accuracy?
60            accuracies.append(torch.sum(torch.absolute(y - output) <= 0.01) / (
61                self.n_out_classes * batch)) # correct if probability within 0.01
62            # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
63
64        return accuracies, loss

```

B.2.11 architectures_cpc -> cpc_downstream_twolinear.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class DownstreamLinearNet(nn.Module):
6
7     def __init__(self, latent_size, context_size, out_classes, use_context=True, use_latents=False, verbose=False):
8         super().__init__()
9         self.latent_size = latent_size
10        self.n_out_classes = out_classes
11        self.use_context = use_context
12        self.use_latents = use_latents
13        self.verbose = verbose
14        if self.use_context and self.use_latents:
15            self.classifier = nn.Sequential(
16                nn.Linear(context_size * 2, context_size * 2),
17                nn.ReLU(),

```

```

18         nn.Linear(context_size * 2, out_classes)
19     )
20     else: # if not both are getting used, normal classifier is enough for each
21     self.classifier = nn.Sequential(
22         nn.Linear(context_size, context_size),
23         nn.ReLU(),
24         nn.Linear(context_size, out_classes)
25     )
26     if self.use_latents:
27         self.rnn = nn.GRU(input_size=latent_size, hidden_size=context_size, num_layers=1, batch_first=False)
28
29     self.activation = nn.Sigmoid()
30
31 def forward(self, latents=None, context=None, y=None):
32     if self.verbose and latents: print('latents', latents.shape)
33     if self.verbose and context: print('context', context.shape)
34     if self.use_context and self.use_latents:
35         x, c = self.rnn(latents)
36         context = torch.cat([context, x[-1, :]], dim=-1)
37     elif self.use_latents:
38         x, c = self.rnn(latents)
39         context = x[-1, :]
40
41     pred = self.classifier(context)
42     output = self.activation(pred)
43     # print('pred shape', pred.shape)
44     if y is None: #
45         return output
46     else: # Training mode return loss instead of prediction
47         raise NotImplementedError

```

B.2.12 architectures_cpc -> cpc_downstream_twolinear_v2.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class DownstreamLinearNet(nn.Module):
6
7     def __init__(self, latent_size, context_size, out_classes, use_context=True, use_latents=False, verbose=False):
8         super().__init__()
9         self.latent_size = latent_size
10        self.n_out_classes = out_classes
11        self.use_context = use_context
12        self.use_latents = use_latents
13        self.verbose = verbose
14        if self.use_context:
15            self.classifier_c = nn.Sequential(
16                nn.ReLU(),
17                nn.Linear(context_size, context_size*2),
18                nn.ReLU(),
19                nn.Linear(context_size*2, out_classes)
20            )
21        if self.use_latents:
22            self.classifier_l = nn.Sequential(
23                nn.ReLU(),
24                nn.Linear(latent_size, latent_size*2),
25                nn.ReLU(),
26                nn.Linear(latent_size*2, out_classes)
27            )
28
29        self.activation = nn.Sigmoid()
30
31    def forward(self, latents=None, context=None, y=None):
32        if self.verbose and latents: print('latents', latents.shape)
33        if self.verbose and context: print('context', context.shape)
34        if self.use_context and self.use_latents:
35            predc = self.classifier_c(context)
36            predl = self.classifier_l(latents)
37            predl = torch.max(predl, dim=0).values
38            pred = (predc+predl)/2
39        elif self.use_latents:
40            pred = self.classifier_l(latents)
41            pred = torch.max(pred, dim=0).values
42        else:
43            pred = self.classifier_c(context)
44            output = self.activation(pred) # do not squeeze on batch?
45            # print('pred shape', pred.shape)
46        if y is None: #
47            return output
48        else: # Training mode return loss instead of prediction
49            raise NotImplementedError

```

B.2.13 architectures_cpc -> cpc_encoder_as_strided.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class StridedEncoder(nn.Module):
6     def __init__(self, cpc_encoder, window_size):
7         super().__init__()
8         self.window_layer = WindowLayer(window_size=window_size)
9         self.cpc_encoder = cpc_encoder
10        self.requires_grad_(True)
11
12    def forward(self, X):
13        #X2 = nn.Parameter(self.window_layer(X), requires_grad=True)
14        X2 = self.window_layer(X)
15        n_windows, n_batches, channels, window_size = X2.shape
16        latents = self.cpc_encoder(
17            X2.reshape(-1, *X2.shape[2:])) # reshape windows into batch dimension for only one forward
18        latents = latents.reshape(n_windows, n_batches, *latents.shape[1:]).squeeze(
19            -1) # shape is now n_windows, n_batches, n_latents
20        latents = latents.movedim(0, -1) # shape is now n_batches, n_latents, steps
21        return latents
22
23
24 class StridedWindowLayer(nn.Module):
25     def __init__(self, window_size=10, stride=None, stack_dim=0):
26         super().__init__()
27         self.stride = window_size if stride is None else stride
28         self.window_size = window_size
29         self.stack_dim = 0
30
31     def forward(self, X):
32         start_ix = torch.arange(0, X.shape[-1], self.stride)
33         print(start_ix)
34         return torch.narrow(X, dim=-1, start=start_ix, length=self.window_size)
35
36
37 class WindowLayer(nn.Module):
38     def __init__(self, window_size=10, stack_dim=0):
39         super().__init__()
40         self.window_size = window_size
41         self.stack_dim = 0
42
43     def forward(self, X):
44         in_shape = X.shape
45         n_windows = X.shape[-1] // self.window_size # how many windows
46         X1 = X.narrow(-1, 0, self.window_size * n_windows) # slice data
47         X2 = X1.view(*X1.shape[0:-1], n_windows, self.window_size).movedim(-2, self.stack_dim)
48         return X2
49
50
51 if __name__ == '__main__':
52     a = torch.stack([torch.stack([torch.arange(100)] * 8)] * 12, dim=1)
53     o = WindowLayer(10)(a)
54     StridedEncoder(None, 10)(a)

```

B.2.14 architectures_cpc -> cpc_encoder_decoder_v2.py (code)

```

1 from torch import nn
2
3
4 class Decoder(nn.Module):
5     def __init__(self, channels, latent_size):
6         super(Decoder, self).__init__()
7         filters = [8, 6, 3, 3, 3] # , (8,1), (4,1), (4,1), (4,1)] # See https://arxiv.org/pdf/1807.03748.pdf#Audio
8         strides = [4, 2, 1, 1, 1] # , (4,1), (2,1), (2,1), (1,1)]
9         dilations = [1, 1, 1, 2, 4]
10        n_channels = [channels] + [latent_size] * len(filters)
11        # self.batch_norm = nn.BatchNorm1d(n_channels[0]) #not used in paper?
12        self.deconvolutionals = nn.Sequential(
13            *[e for t in [
14                (nn.ConvTranspose1d(in_channels=n_channels[i + 1], out_channels=n_channels[i], kernel_size=filters[i],
15                                dilation=dilations[i], stride=strides[i], padding=0),
16                (nn.BatchNorm1d(n_channels[i])),
17                (nn.ReLU())),
18                ) for i in reversed(range(len(filters)))] for e in t]
19        )
20
21        self.avg_pool = nn.AdaptiveAvgPool1d(1)
22
23        def _weights_init(m):
24            if isinstance(m, nn.Conv1d):
25                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
26
27        self.apply(_weights_init)

```

```

28
29     def forward(self, x):
30         # Input has shape (batches, channels, window_size)
31         # print('decoder input shape:', x.shape)
32         # x = self.batch_norm(x)
33         x = self.deconvolutionals(x.unsqueeze(2))
34         x = x.squeeze(2) # Only squeeze first (NOT BATCH!) dimension
35         # print('Encoder output shape:', x.shape)
36
37     return x

```

B.2.15 architectures_cpc -> cpc_encoder_likev8.py (code)

```

1 from torch import nn
2
3
4 class Encoder(nn.Module):
5     def __init__(self, channels, latent_size):
6         super(Encoder, self).__init__()
7         # self.batch_norm = nn.BatchNorm1d(n_channels[0]) #not used in paper?
8         self.convolutions = nn.Sequential(
9             nn.Conv1d(in_channels=channels, out_channels=latent_size, kernel_size=3, dilation=1),
10            nn.BatchNorm1d(latent_size),
11            nn.ReLU(),
12            nn.MaxPool1d(3),
13            nn.Conv1d(in_channels=latent_size, out_channels=latent_size, kernel_size=3, dilation=2),
14            nn.BatchNorm1d(latent_size),
15            nn.ReLU(),
16            nn.MaxPool1d(3),
17            nn.Conv1d(in_channels=latent_size, out_channels=latent_size, kernel_size=3, dilation=4),
18            nn.BatchNorm1d(latent_size),
19            nn.ReLU(),
20            nn.MaxPool1d(3),
21            nn.Conv1d(in_channels=latent_size, out_channels=latent_size, kernel_size=3, dilation=8),
22            nn.BatchNorm1d(latent_size),
23            nn.ReLU(),
24        )
25
26     def _weights_init(m):
27         if isinstance(m, nn.Conv1d):
28             nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
29
30     self.apply(_weights_init)
31
32     def forward(self, x):
33         # Input has shape (batches, channels, window_size)
34         # print('Encoder input shape:', x.shape)
35         # x = self.batch_norm(x)
36         x = self.convolutions(x)
37         # print('Encoder output shape:', x.shape)
38
39     return x

```

B.2.16 architectures_cpc -> cpc_encoder_small.py (code)

```

1 from torch import nn
2
3
4 class Encoder(nn.Module):
5     def __init__(self, channels, latent_size):
6         super(Encoder, self).__init__()
7         filters = [11, 7, 5, 5, 3] # See https://arxiv.org/pdf/1807.03748.pdf#Audio
8         strides=[3, 2, 2, 1, 1]
9         n_channels = [channels] + [latent_size] * len(filters)
10        # self.batch_norm = nn.BatchNorm1d(n_channels[0]) #not used in paper?
11        self.convolutions = nn.Sequential(
12            *[e for t in [
13                (nn.Conv1d(in_channels=n_channels[i], out_channels=n_channels[i + 1], kernel_size=filters[i],
14                           stride=strides[i], padding=0),
15                (nn.BatchNorm1d(n_channels[i + 1])),
16                (nn.ReLU()))
17            ) for i in range(len(filters))] for e in t]
18        )
19
20     def _weights_init(m):
21         if isinstance(m, nn.Conv1d):
22             nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
23
24     self.apply(_weights_init)
25
26     def forward(self, x):
27         # Input has shape (batches, channels, window_size)

```

```

28     # print('Encoder input shape:', x.shape)
29     # x = self.batch_norm(x)
30     x = self.convolutions(x)
31     # print('Encoder output shape:', x.shape)
32
33     return x

```

B.2.17 architectures_cpc -> cpc_encoder_v0.py (code)

```

1 from torch import nn
2
3
4 class Encoder(nn.Module):
5     def __init__(self, channels, latent_size):
6         super(Encoder, self).__init__()
7         filters = [10, 8, 4, 4, 4] # See https://arxiv.org/pdf/1807.03748.pdf#Audio
8         strides = [5, 4, 2, 2, 2]
9         n_channels = [channels] + [latent_size] * len(filters)
10        # self.batch_norm = nn.BatchNorm1d(n_channels[0]) #not used in paper?
11        self.convolutions = nn.Sequential(
12            *[e for t in [
13                (nn.Conv1d(in_channels=n_channels[i], out_channels=n_channels[i + 1], kernel_size=filters[i],
14                           stride=strides[i], padding=0),
15                (nn.BatchNorm1d(n_channels[i + 1])),
16                (nn.ReLU())),
17                ) for i in range(len(filters))] for e in t]
18        )
19
20    def _weights_init(m):
21        if isinstance(m, nn.Conv1d):
22            nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
23
24    self.apply(_weights_init)
25
26    def forward(self, x):
27        # Input has shape (batches, channels, window_size)
28        # print('Encoder input shape:', x.shape)
29        # x = self.batch_norm(x)
30        x = self.convolutions(x)
31        # print('Encoder output shape:', x.shape)
32
33        return x

```

B.2.18 architectures_cpc -> cpc_encoder_v1.py (code)

```

1 from torch import nn
2
3
4 class Encoder(nn.Module):
5     def __init__(self, channels, latent_size):
6         super(Encoder, self).__init__()
7         filters = [8, 6, 5, 4, 3] # , (8,1), (4,1), (4,1), (4,1)] # See https://arxiv.org/pdf/1807.03748.pdf#Audio
8         strides = [4, 2, 1, 1, 1] # , (4,1), (2,1), (2,1), (1,1)]
9         n_channels = [channels] + [latent_size] * len(filters)
10        # self.batch_norm = nn.BatchNorm1d(n_channels[0]) #not used in paper?
11        self.convolutions = nn.Sequential(
12            *[e for t in [
13                (nn.Conv1d(in_channels=n_channels[i], out_channels=n_channels[i + 1], kernel_size=filters[i],
14                           stride=strides[i], padding=0),
15                nn.BatchNorm1d(n_channels[i + 1]),
16                (nn.ReLU())),
17                ) for i in range(len(filters))] for e in t]
18        )
19
20        self.avg_pool = nn.AdaptiveAvgPool1d(1)
21
22    def _weights_init(m):
23        if isinstance(m, nn.Conv2d):
24            nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
25
26    self.apply(_weights_init)
27
28    def forward(self, x):
29        # Input has shape (batches, channels, window_size)
30        # print('Encoder input shape:', x.shape)
31        # x = self.batch_norm(x)
32        x = self.convolutions(x)
33        # print('out shape', x.shape)
34        x = self.avg_pool(x)
35        # print('out2', x.shape)
36        # Output has shape (batches, latents, 1)
37        # Maybe squeeze??

```

```

38     x = x.squeeze(2) # Only squeeze first (NOT BATCH!) dimension
39     # print('Encoder output shape:', x.shape)
40
41     return x

```

B.2.19 architectures_cpc -> cpc_encoder_v2.py (code)

```

1 from torch import nn
2
3
4 class Encoder(nn.Module):
5     def __init__(self, channels, latent_size):
6         super(Encoder, self).__init__()
7         kernel_sizes = [8, 6, 3, 3, 3]
8         strides = [4, 2, 1, 1, 1]
9         dilations = [1, 1, 1, 3, 9]
10        n_channels = [channels] + [latent_size] * len(kernel_sizes)
11        # self.batch_norm = nn.BatchNorm1d(n_channels[0]) #not used in paper?
12        self.convolutionals = nn.Sequential(
13            *[e for t in [
14                (nn.Conv2d(in_channels=n_channels[i], out_channels=n_channels[i + 1], kernel_size=kernel_sizes[i],
15                           dilation=dilations[i], stride=strides[i], padding=0),
16                           (nn.BatchNorm1d(n_channels[i + 1])),
17                           (nn.ReLU())),
18                           ) for i in range(len(kernel_sizes))] for e in t]
19        )
20
21        # self.avg_pool = nn.AdaptiveAvgPool1d(1)
22        def _weights_init(m):
23            if isinstance(m, nn.Conv2d):
24                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
25
26        self.apply(_weights_init)
27
28    def forward(self, x):
29        # Input has shape (batches, channels, window_size)
30        # print('Encoder input shape:', x.shape)
31        # x = self.batch_norm(x)
32
33        x = self.convolutionals(x)
34        # print('out shape', x.shape)
35        # x = self.avg_pool(x)
36        # print('out2', x.shape)
37        # Output has shape (batches, latents, 1)
38        # Maybe squeeze??
39        x = x.squeeze(2) # Only squeeze first (NOT BATCH!) dimension
40        # print('Encoder output shape:', x.shape)
41
42        return x

```

B.2.20 architectures_cpc -> cpc_encoder_v3.py (code)

```

1 from torch import nn
2
3
4 class Encoder(nn.Module):
5     def __init__(self, channels, latent_size):
6         super(Encoder, self).__init__()
7         kernel_sizes = [7, 3, 3, 3, 3]
8         strides = [2, 1, 1, 1, 1]
9         dilations = [1, 1, 3, 9, 27]
10        n_channels = [channels] + [latent_size] * len(kernel_sizes)
11        # self.batch_norm = nn.BatchNorm1d(n_channels[0]) #not used in paper?
12        self.convolutionals = nn.Sequential(
13            *[e for t in [
14                (nn.Conv2d(in_channels=n_channels[i], out_channels=n_channels[i + 1], kernel_size=kernel_sizes[i],
15                           dilation=dilations[i], stride=strides[i], padding=0),
16                           (nn.ReLU())),
17                           ) for i in range(len(kernel_sizes))] for e in t]
18        )
19
20        # self.avg_pool = nn.AdaptiveAvgPool1d(1)
21        def _weights_init(m):
22            if isinstance(m, nn.Conv2d):
23                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
24
25        self.apply(_weights_init)
26
27    def forward(self, x):
28        # Input has shape (batches, channels, window_size)
29        # print('Encoder input shape:', x.shape)
30        # x = self.batch_norm(x)

```

```

31     x = self.convolutions(x)
32     # print('out shape', x.shape)
33     # x = self.avg_pool(x)
34     # print('out2', x.shape)
35     # Output has shape (batches, latents, 1)
36     # Maybe squeeze??
37     x = x.squeeze(2) # Only squeeze first (NOT BATCH!) dimension
38     # print('Encoder output shape:', x.shape)
39
40
41     return x

```

B.2.21 architectures_cpc -> cpc_encoder_v4.py (code)

```

1 from torch import nn
2
3
4 # Idea: Make a big receptive field using dilations
5 # Do not use strides to not miss any information
6 # Use maxpool instead of strides to select a window where activation is biggest
7
8
9 class Encoder(nn.Module):
10     def __init__(self, channels, latent_size):
11         super(Encoder, self).__init__()
12         kernel_sizes = [5, 3, 3, 3, 3]
13         strides = [1, 1, 1, 1, 1]
14         dilations = [1, 5, 3 * 5, 3 * 3 * 5, 5 * 3 * 3 * 3]
15         max_pool_sizes = [1, 1, 3, 3, 3]
16         n_channels = [channels, 3, 24, 32, 64,
17                         latent_size] # channels in, 3 to downsample into important channels, than upscale to find
18         features_filters
19         # self.batch_norm = nn.BatchNorm1d(n_channels[0]) #not used in paper?
20         self.convolutions = nn.Sequential(
21             *[e for t in [
22                 (nn.Conv1d(in_channels=n_channels[i], out_channels=n_channels[i + 1], kernel_size=kernel_sizes[i],
23                           stride=strides[i], padding=0, dilation=dilations[i]),
24                           # PrintLayer('after conv'),
25                           # nn.MaxPool1d(max_pool_sizes[i]),
26                           # PrintLayer('after pool'),
27                           (nn.ReLU())),
28                 ) for i in range(len(kernel_sizes)))] for e in t]
29         )
30
31         # self.avg_pool = nn.AdaptiveAvgPool1d(1)
32         def _weights_init(m):
33             if isinstance(m, nn.Conv1d):
34                 nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
35
36         self.apply(_weights_init)
37
38     def forward(self, x):
39         # Input has shape (batches, channels, window_size)
40         # print('Encoder input shape:', x.shape)
41         # x = self.batch_norm(x)
42
43         x = self.convolutions(x)
44         # print('out shape', x.shape)
45         # x = self.avg_pool(x)
46         # print('out2', x.shape)
47         # Output has shape (batches, latents, 1)
48         # Maybe squeeze??
49         x = x.squeeze(2) # Only squeeze first (NOT BATCH!) dimension
50         # print('Encoder output shape:', x.shape)
51
52         return x

```

B.2.22 architectures_cpc -> cpc_encoder_vresnet.py (code)

```

1 from torch import nn
2
3 from external.multi_scale_ori import MSResNet
4
5
6 class Encoder(nn.Module):
7     def __init__(self, channels, latent_size):
8         super(Encoder, self).__init__()
9         self.resnet = MSResNet(channels, layers=[1, 1, 1, 1, 1], num_classes=2) # num_classes not really used
10
11     def forward(self, x):
12         # Input has shape (batches, channels, window_size)
13         # print('Encoder input shape:', x.shape)

```

```

14     # x = self.batch_norm(x)
15     _, x = self.resnet(x)
16     # x = x.permute(2, 0, 1).squeeze(0) #Only squeeze first (NOT BATCH!) dimension
17     # print('Encoder output shape:', x.shape)
18
19     return x

```

B.2.23 architectures_cpc -> cpc_intersect.py (code)

```

1 import torch
2 from torch import nn
3 import numpy as np
4 import torch.nn.functional as F
5
6 from util.utility.sparse_print import SparsePrint
7
8
9 class CPC(nn.Module):
10
11     def __init__(self, encoder, autoregressive, predictor, timesteps_in, timesteps_out, latent_size, timesteps_ignore
12                 = 0,
13                 normalize_latents=False, verbose=False, sampling_mode='same'):
14         super().__init__()
15         self.encoder = encoder
16         self.autoregressive = autoregressive
17         self.predictor = predictor
18         self.lsoftmax = nn.LogSoftmax(dim=-1)
19         self.timesteps_in = timesteps_in
20         self.timesteps_out = timesteps_out
21         self.timesteps_ignore = timesteps_ignore
22         self.latent_size = latent_size
23         self.normalize_latents = normalize_latents
24         self.verbose = verbose
25         self.cpc_train_mode = True
26         if sampling_mode in ['all', 'same', 'random', 'future']:
27             self.sampling_mode = sampling_mode
28         else:
29             self.sampling_mode = 'same'
30         self.sprint = SparsePrint()
31
32     def forward(self, X, y=None, hidden=None):
33         if self.verbose: print('input', X.shape)
34         if len(X.shape) == 4: # uses cpc or challenge dataset
35             batch, windows, channels, length = X.shape
36             X = X.transpose(1, 2).reshape((batch, channels, -1))
37         elif len(X.shape) == 3: # uses baseline dataset, shape: (batch, length, channels)
38             X = X.transpose(1, 2)
39             batch, channels, length = X.shape # assume baseline dataset shape
40             encoded_x = self.encoder(X) # encode whole data: shape: (batch, latent_size, length->out_length)
41             encoded_x = encoded_x.permute(2, 0, 1) # shape outlength, batch, latent_size
42             # if self.normalize_latents:
43             #     encoded_x = encoded_x / torch.norm(encoded_x, p=2, dim=-1, keepdim=True)
44             if self.verbose: print('encoder_x', encoded_x.shape)
45             encoded_x_steps, _, _ = encoded_x.shape
46             if hidden is None:
47                 hidden = self.autoregressive.init_hidden(batch_size=batch)
48             context, hidden = self.autoregressive(encoded_x, hidden)
49             if self.verbose: print('context', context.shape)
50
51             if not self.cpc_train_mode:
52                 return encoded_x, context[-1, :, :], hidden
53             # offset = np.random.choice(encoded_x_steps - self.timesteps_out - self.timesteps_ignore, 10, replace=False)
54             # Draw 10 randoms
55
56             loss = torch.tensor([0.0]).cuda()
57             correct = torch.tensor([0.0]).cuda()
58             if encoded_x_steps - self.timesteps_out - self.timesteps_ignore - self.timesteps_in < 1:
59                 print(
60                     'Not enough encoded values for CPC training. Lower timesteps_in/timesteps_out/timesteps_ignore or
61                     change window_size + encoder')
62                 print(
63                     f'Encoded {encoded_x_steps} steps but need at least {(self.timesteps_out + self.timesteps_ignore +
64                     self.timesteps_in)}')
65                 return loss, correct, hidden
66             if self.sampling_mode == 'same':
67                 for k in range(self.timesteps_out):
68                     pred_latent = self.predictor(
69                         context[self.timesteps_in:-(self.timesteps_out + self.timesteps_ignore + 1), :, :, :],
70                         encoded_latent = encoded_x[self.timesteps_in + k + 1:-(self.timesteps_out + self.timesteps_ignore) + k
71
72                         :, :, :] # shape is batch, latent_size
73                     if self.verbose: print(pred_latent.shape, encoded_latent.shape)
74                     if self.normalize_latents:
75                         pred_len = torch.clamp(torch.norm(pred_latent, p=2, dim=-1, keepdim=True), min=1e-10)
76                         enc_len = torch.clamp(torch.norm(encoded_latent, p=2, dim=-1, keepdim=True), min=1e-10)

```

```

73         pred_latent = pred_latent / pred_len
74         encoded_latent = encoded_latent / enc_len
75         for step in range(pred_latent.shape[0]): # TODO: can this be broadcasted?
76             softmax = self.lsoftmax(torch.mm(encoded_latent[step], pred_latent[step].T)) # output: (Batches,
77             Batches)
77             if self.verbose: print('softmax shape', softmax.shape)
78             correct += torch.sum(torch.argmax(softmax, dim=0) == torch.arange(batch).cuda()) # Since
79             loss += torch.sum(torch.diag(softmax))
80
81             loss /= (batch * self.timesteps_out * pred_latent.shape[0]) * -1.0
82             accuracy = correct.true_divide(batch * self.timesteps_out * pred_latent.shape[0])
83             return accuracy, loss, hidden
84
85     if self.sampling_mode == 'all':
86         t = np.random.randint(self.timesteps_in, encoded_x_steps-self.timesteps_out-self.timesteps_ignore)
87         if context.shape[0] > 1: #uses rnn context
88             current_context = context[t-1, :, :]
89         else: #uses hidden or something else
90             current_context = context[0, :, :]
91         #enc_resh = encoded_x.reshape(encoded_x_steps*batch, -1)
92
93         for k in range(self.timesteps_out):
94             pred_latent = self.predictor(current_context, k)
95             if self.verbose: print("pred latent shape", pred_latent.shape)
96
97             sim = encoded_x@pred_latent.T #shape: (encoded_x_steps, batch, batch) # sim[i, j] = scalar prod
98             between li and pred j
99             soft = F.log_softmax(sim.reshape(encoded_x_steps*batch, batch), dim=0) #Where is the highest
100            similarity? (max in each i) == i
101            soft_resh = soft.reshape(encoded_x_steps, batch, batch)
102            # soft2 = F.softmax(sim, dim=0)
103            # self.sprint.print(soft, 1000)
104            loss += torch.diag(soft_resh[t+k+self.timesteps_ignore]).sum() #higher = better
105            correct += (soft.max(dim=0).indices == torch.arange(batch, device=soft.device)+batch*(t+k+self.
106            timesteps_ignore)).sum()
107
108            loss = -loss/(self.timesteps_out*batch)
109            accuracy = correct.true_divide(batch * self.timesteps_out) # * pred_latent.shape[0]
110            return accuracy, loss, hidden
111
112     if self.sampling_mode == 'random':
113         raise NotImplementedError
114
115     if self.sampling_mode == 'future':
116         t = np.random.randint(self.timesteps_in, encoded_x_steps-self.timesteps_out-self.timesteps_ignore)
117         current_context = context[t-1, :, :]
118         #enc_resh = encoded_x.reshape(encoded_x_steps*batch, -1)
119         encoded_future = encoded_x[t:, :, :]
120         for k in range(self.timesteps_out):
121             pred_latent = self.predictor(current_context, k)
122             if self.verbose: print("pred latent shape", pred_latent.shape)
123
123             sim = encoded_future@pred_latent.T #shape: (encoded_x_steps-t, batch, batch) # sim[i, j] = scalar prod
124             between li and pred j
125             soft = F.log_softmax(sim.reshape((encoded_x_steps-t)*batch, batch), dim=0) #Where is the highest
126             similarity? (max in each i) == i
127             soft_resh = soft.reshape((encoded_x_steps-t), batch, batch)
128             # soft2 = F.softmax(sim, dim=0)
129             # self.sprint.print(soft, 1000)
130             loss += torch.diag(soft_resh[k+self.timesteps_ignore]).sum() #higher = better
131             correct += (soft.max(dim=0).indices == torch.arange(batch, device=soft.device)+batch*(k+self.
132             timesteps_ignore)).sum()
133
133     def freeze_layers(self):
134         for param in self.parameters():
135             param.requires_grad = False
136
137     def unfreeze_layers(self):
138         for param in self.parameters():
139             param.requires_grad = True

```

B.2.24 architectures_cpc -> cpc_intersect_manylatents.py (code)

```

1 import torch
2 from torch import nn
3 import numpy as np
4 import torch.nn.functional as F
5
6 from util.utility.sparse_print import SparsePrint
7
8
9 class CPC(nn.Module):

```

```

10
11     def __init__(self, encoder, autoregressive, predictor, timesteps_in, timesteps_out, latent_size, timesteps_ignore
12         =0,
13             normalize_latents=False, verbose=False, sampling_mode='same', alpha=0.01):
14         super().__init__()
15         self.encoder = encoder
16         self.autoregressive = autoregressive
17         self.predictor = predictor
18         self.lsoftmax = nn.LogSoftmax(dim=-1)
19         self.timesteps_in = timesteps_in
20         self.timesteps_out = timesteps_out
21         self.timesteps_ignore = timesteps_ignore
22         self.latent_size = latent_size
23         self.normalize_latents = normalize_latents
24         self.verbose = verbose
25         self.cpc_train_mode = True
26         self.sampling_mode = sampling_mode
27         self.sparse = SparsePrint()
28         self.alpha = alpha
29         self.crossentropyloss = nn.CrossEntropyLoss(reduction='sum')
30
31     def forward(self, X, y=None, hidden=None):
32         if self.verbose: print('input', X.shape)
33         if len(X.shape) == 4: # uses cpc or challenge dataset
34             batch, windows, channels, length = X.shape
35             X = X.transpose(1, 2).reshape((batch, channels, -1))
36         elif len(X.shape) == 3: # uses baseline dataset, shape: (batch, length, channels)
37             X = X.transpose(1, 2)
38         batch, channels, length = X.shape # assume baseline dataset shape
39         encoded_x = self.encoder(X) # encode whole data: shape: (batch, latent_size, length->out_length)
40         encoded_x = encoded_x.permute(2, 0, 1) # shape outlength, batch, latent_size
41         if self.verbose: print('encoder_x', encoded_x.shape)
42         encoded_x_steps, _, _ = encoded_x.shape
43         if self.normalize_latents:
44             enc_len = torch.clamp(torch.norm(encoded_x, p=2, dim=-1, keepdim=True), min=1e-10)
45             encoded_x = encoded_x / enc_len
46
47         if hidden is None:
48             hidden = self.autoregressive.init_hidden(batch_size=batch)
49         if not self.cpc_train_mode:
50             context, hidden = self.autoregressive(encoded_x, hidden)
51             if self.verbose: print('context', context.shape)
52             return encoded_x, context[-1, :, :], hidden
53         else:
54             t = np.random.randint(self.timesteps_in, encoded_x_steps-self.timesteps_out-self.timesteps_ignore)
55             context, hidden = self.autoregressive(encoded_x[t-self.timesteps_in:t], hidden) #Calculate less contextual
56             context
57             encoded_x = encoded_x[t-self.timesteps_in:t+self.timesteps_ignore+self.timesteps_out]
58             encoded_x_steps, _, _ = encoded_x.shape
59             # offset = np.random.choice(encoded_x_steps - self.timesteps_out - self.timesteps_ignore, 10, replace=False)
60             # Draw 10 randoms
61
62             loss = torch.tensor([0.0]).cuda()
63             correct = torch.tensor([0.0]).cuda()
64             if self.sampling_mode == 'same':
65                 current_context = context[-1, :, :]
66                 for k in range(self.timesteps_out):
67                     pred_latent = self.predictor(current_context, k)
68                     encoded_latent = encoded_x[self.timesteps_in + self.timesteps_ignore + k, :, :] # shape is batch,
69                     latent_size
70                     if self.verbose: print(pred_latent.shape, encoded_latent.shape)
71                     softmax = self.lsoftmax(torch.mm(encoded_latent, pred_latent.T)) # output: (Batches, Batches)
72                     if self.verbose: print('softmax shape', softmax.shape)
73                     correct += torch.sum(torch.argmax(softmax, dim=0) == torch.arange(batch).cuda()) # Since
74                     loss += torch.sum(torch.diag(softmax))
75
76             loss /= (batch * self.timesteps_out) * -1.0
77             accuracy = correct.true_divide(batch * self.timesteps_out)
78             return accuracy, loss, hidden
79
80         if self.sampling_mode == 'multisame':
81             for k in range(self.timesteps_out):
82                 pred_latent = self.predictor(context, k) #use whole context instead of current context
83                 encoded_latent = encoded_x[k+1:self.timesteps_in + self.timesteps_ignore + k + 1, :, :] # shape is
84                 batch, latent_size
85                 for step in range(pred_latent.shape[0]):
86                     if self.verbose: print(pred_latent.shape, encoded_latent.shape)
87                     softmax = self.lsoftmax(torch.mm(encoded_latent[step], pred_latent[step].T)) # output: (Batches,
88                     Batches)
89                     if self.verbose: print('softmax shape', softmax.shape)
90                     correct += torch.sum(torch.argmax(softmax, dim=0) == torch.arange(batch).cuda()) # Since
91                     loss += torch.sum(torch.diag(softmax))
92
93             loss /= (batch * self.timesteps_out * pred_latent.shape[0]) * -1.0
94             accuracy = correct.true_divide(batch * self.timesteps_out * pred_latent.shape[0])
95             return accuracy, loss, hidden
96
97         if self.sampling_mode == 'all':
98             current_context = context[-1, :, :]
99

```

```

94     for k in range(self.timesteps_out):
95         pred_latent = self.predictor(current_context, k)
96         if self.verbose: print("pred latent shape", pred_latent.shape)
97
98         sim = encoded_x@pred_latent.T # sim[i, j] = scalar prod between li and pred j
99         soft = F.log_softmax(sim.reshape(encoded_x_steps*batch, -1), dim=0) #Where is the highest similarity?
100        (max in each i) == i
101        soft_resh = soft.reshape(encoded_x_steps, batch, -1)
102        # soft2 = F.softmax(sim, dim=0)
103        # self.sprint.print(soft, 10000)
104        loss += torch.diag(soft_resh[self.timesteps_in+self.timesteps_ignore+k]).sum() #higher = better
105        correct += (soft.max(dim=0).indices == torch.arange(batch, device=soft.device)+batch*(self.
106        timesteps_in+k)).sum()
107
108        loss = -loss/(self.timesteps_out*batch)
109        accuracy = correct.true_divide(batch * self.timesteps_out) # * pred_latent.shape[0]
110        return accuracy, loss, hidden
111
112    if self.sampling_mode == 'random':
113        raise NotImplementedError
114
115    if self.sampling_mode == 'future':
116        current_context = context[-1, :, :]
117        encoded_x = encoded_x[-self.timesteps_out:, :, :] #only use future latents
118        encoded_x_steps, _, _ = encoded_x.shape
119
120        for k in range(self.timesteps_out):
121            pred_latent = self.predictor(current_context, k)
122            if self.verbose: print("pred latent shape", pred_latent.shape)
123
124            sim = encoded_x@pred_latent.T # sim[i, j] = scalar prod between li and pred j
125            soft = F.log_softmax(sim.reshape(encoded_x_steps*batch, -1), dim=0) #Where is the highest similarity?
126            (max in each i) == i
127            soft_resh = soft.reshape(encoded_x_steps, batch, -1)
128            # soft2 = F.softmax(sim, dim=0)
129            # self.sprint.print(soft, 10000)
130            loss += torch.diag(soft_resh[k]).sum() #higher = better
131            correct += (soft.max(dim=0).indices == torch.arange(batch, device=soft.device)+batch*(k)).sum()
132
133            loss = -loss/(self.timesteps_out*batch)
134            accuracy = correct.true_divide(batch * self.timesteps_out) # * pred_latent.shape[0]
135            return accuracy, loss, hidden
136
137    if self.sampling_mode == 'alpha': #same as all but with alpha
138        multiplier = torch.eye(batch, device=encoded_x.device) * (np.log(self.alpha)-np.log((batch-self.alpha)/(batch-1)))
139        current_context = context[-1, :, :]
140
141        for k in range(self.timesteps_out):
142            pred_latent = self.predictor(current_context, k)
143            if self.verbose: print("pred latent shape", pred_latent.shape)
144            sim = encoded_x@pred_latent.T # sim[i, j] = scalar prod between li and pred j
145            sim = sim + np.log((batch-self.alpha)/(batch-1))
146            sim[self.timesteps_in+self.timesteps_ignore+k] = sim[self.timesteps_in+self.timesteps_ignore+k] +
147            multiplier
148
149            soft = F.log_softmax(sim.reshape(encoded_x_steps*batch, -1), dim=0) #(batch/self.alpha) *Where is the
150            highest similarity? (max in each i) == i
151            soft_resh = soft.reshape(encoded_x_steps, batch, -1)
152            # soft2 = F.softmax(sim, dim=0)
153            # self.sprint.print(soft, 10000)
154            loss += torch.diag(soft_resh[self.timesteps_in+self.timesteps_ignore+k]).sum() #higher = better
155            correct += (soft.max(dim=0).indices == torch.arange(batch, device=soft.device)+batch*(self.
156            timesteps_in+k)).sum()
157
158            loss = -loss/(self.timesteps_out*batch)
159            accuracy = correct.true_divide(batch * self.timesteps_out) # * pred_latent.shape[0]
160            return accuracy, loss, hidden
161
162    if self.sampling_mode == 'softmax':
163        current_context = context[-1, :, :]
164
165        for k in range(self.timesteps_out):
166            pred_latent = self.predictor(current_context, k)
167            if self.verbose: print("pred latent shape", pred_latent.shape)
168
169            sim = encoded_x@pred_latent.T # sim[i, j] = scalar prod between li and pred j
170            soft = 1-F.softmax(sim.reshape(encoded_x_steps*batch, -1), dim=0) #Where is the highest similarity? (max in each i) == i
171            soft_resh = soft.reshape(encoded_x_steps, batch, -1)
172            # soft2 = F.softmax(sim, dim=0)
173            # self.sprint.print(soft, 10000)
174            loss += torch.diag(soft_resh[self.timesteps_in+self.timesteps_ignore+k]).sum() #higher = better
175            correct += (soft.min(dim=0).indices == torch.arange(batch, device=soft.device)+batch*(self.
176            timesteps_in+k)).sum()
177
178            loss = loss/(self.timesteps_out*batch)
179            accuracy = correct.true_divide(batch * self.timesteps_out) # * pred_latent.shape[0]
180            return accuracy, loss, hidden

```

```

175     if self.sampling_mode == 'crossentropy': #same as all, implemented with crossentropyloss instead of logsoftmax
176         current_context = context[-1, :, :]
177
178         for k in range(self.timesteps_out):
179             pred_latent = self.predictor(current_context, k)
180             if self.verbose: print("pred latent shape", pred_latent.shape)
181
182             sim = encoded_x@pred_latent.T # sim[i, j] = scalar prod between li and pred j
183             simr = sim.reshape(encoded_x_steps*batch, batch).T #Where is the highest similarity? (max in each i)
184
185             == i
186             # soft2 = F.softmax(sim, dim=0)
187             # self.print(soft, 10000)
188             labels = torch.arange(batch, device=simr.device)+batch*(self.timesteps_in+self.timesteps_ignore+k)
189             loss += self.crossentropyloss(simr, labels) #higher = better
190             correct += (simr.max(dim=-1).indices == labels).sum()
191
192             loss = loss/(self.timesteps_out*batch)
193             accuracy = correct.true_divide(batch * self.timesteps_out) # * pred_latent.shape[0]
194             return accuracy, loss, hidden
195
196         if self.sampling_mode == 'crossentropy-nocontext': #same as all, implemented with crossentropyloss instead of
197             logsoftmax
198             pred_full = self.predictor(encoded_x[:self.timesteps_in], timestep=None)
199             for k in range(self.timesteps_out):
200                 pred_latent = pred_full[k]
201                 if self.verbose: print("pred latent shape", pred_latent.shape)
202
203                 sim = encoded_x@pred_latent.T # sim[i, j] = scalar prod between li and pred j
204                 simr = sim.reshape(encoded_x_steps*batch, batch).T #Where is the highest similarity? (max in each i)
205
206                 == i
207                 # soft2 = F.softmax(sim, dim=0)
208                 # self.print(soft, 10000)
209                 labels = torch.arange(batch, device=simr.device)+batch*(self.timesteps_in+self.timesteps_ignore+k)
210                 loss += self.crossentropyloss(simr, labels) #higher = better
211                 correct += (simr.max(dim=-1).indices == labels).sum()
212
213                 loss = loss/(self.timesteps_out*batch)
214                 accuracy = correct.true_divide(batch * self.timesteps_out) # * pred_latent.shape[0]
215                 return accuracy, loss, hidden
216
217         print("Unknown sampling mode")
218
219
220     def freeze_layers(self):
221         for param in self.parameters():
222             param.requires_grad = False
223
224     def unfreeze_layers(self):
225         for param in self.parameters():
226             param.requires_grad = True

```

B.2.25 architectures_cpc -> cpc_predictor_nocontext.py (code)

```

1 import torch
2 from torch import nn
3
4 from external.tcn.TCN.tcn import TemporalConvNet
5
6
7 class Predictor(nn.Module):
8     def __init__(self, context_size, latent_size, timesteps: int):
9         super().__init__()
10        self.tcn = TemporalConvNet(latent_size, [context_size, context_size, context_size, latent_size])
11
12    def forward(self, x, timestep: int):
13        # input shape = (timesteps, batch, latent_size)
14        # print('Predictor input shape:', x.shape)
15        if timestep is None: # output shape = (timesteps, batch, latent_size)
16            return self.tcn(x.permute(1, 2, 0)).permute(2, 0, 1) #out (timesteps, batch, latent_size)

```

B.2.26 architectures_cpc -> cpc_predictor_stacked.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class Predictor(nn.Module):
6     def __init__(self, context_size, latent_size, timesteps: int):
7         super().__init__()
8         self.linears = nn.ModuleList(

```

```

9     [nn.Sequential(
10         nn.Linear(context_size, context_size),
11         nn.ReLU(),
12         nn.BatchNorm1d(context_size),
13         nn.Linear(context_size, latent_size),
14         nn.ReLU(),
15         nn.BatchNorm1d(latent_size),
16     )] * timesteps
17 )
18
19     def _weights_init(m):
20         if isinstance(m, nn.Linear):
21             nn.init.kaiming_normal_(m.weight, mode='fan_in', nonlinearity='relu')
22
23     self.apply(_weights_init)
24
25     def forward(self, x, timestep: int):
26         # print('Predictor input shape:', x.shape)
27
28         if timestep is None: # output shape = (timesteps, batch, latent_size)
29             preds = []
30             for i, l in enumerate(self.linears):
31                 preds.append(self.linears[i](x))
32             return torch.stack(preds, dim=0)
33
34         else: # output shape = (batch, latent_size)
35             return self.linears[timestep](x)

```

B.2.27 architectures_cpc -> cpc_predictor_v0.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class Predictor(nn.Module):
6     def __init__(self, context_size, latent_size, timesteps: int):
7         super().__init__()
8         self.linears = nn.ModuleList(
9             [nn.Linear(context_size, latent_size)] * timesteps
10        )
11
12     def _weights_init(m):
13         if isinstance(m, nn.Linear):
14             nn.init.kaiming_normal_(m.weight, mode='fan_in', nonlinearity='relu')
15
16     self.apply(_weights_init)
17
18     def forward(self, x, timestep: int):
19         # print('Predictor input shape:', x.shape)
20
21         if timestep is None: # output shape = (timesteps, batch, latent_size)
22             preds = []
23             for i, l in enumerate(self.linears):
24                 preds.append(self.linears[i](x))
25             return torch.stack(preds, dim=0)
26
27         else: # output shape = (batch, latent_size)
28             return self.linears[timestep](x)

```

B.2.28 architectures_cpc -> cpc_with_decoder.py (code)

```

1 import torch
2 from torch import nn
3 from torch.optim import Adam
4 from torch.utils.data import DataLoader
5
6 from util.data import ecg_datasets2
7 from architectures_cpc import cpc_encoder_v2, cpc_encoder_decoder_v2
8 from external.tcn.TCN import TemporalConvNet
9
10
11 class SimpleCPC(nn.Module):
12
13     def __init__(self, encoder, decoder, autoregressive, timesteps_in, timesteps_out, latent_size):
14         super().__init__()
15         self.encoder = encoder
16         self.decoder = decoder
17         self.autoregressive = autoregressive
18         self.timesteps_in = timesteps_in
19         self.timesteps_out = timesteps_out
20         self.latent_size = latent_size
21

```

```

22     def forward(self, X):
23         batch, n_windows, channels, length = X.shape
24         assert n_windows >= self.timesteps_in + self.timesteps_out
25         x = X.reshape((batch * n_windows, channels, length)) # squash into batch dimension
26         encoded_x = self.encoder(x)
27         encoded_x = encoded_x.reshape((batch, n_windows, -1)) # reshape into original
28         # if self.autoregressive.uses_hidden_state():
29         #     if hidden is None:
30         #         hidden = self.autoregressive.init_hidden(batch, use_gpu=False)
31         #         predicted, hidden = self.autoregressive(encoded_x[:self.timesteps_in], hidden)
32         #     else:
33         encoded_x = encoded_x.transpose(1, 2) # new shape is (batch, latents, steps)
34         predicted = self.autoregressive(encoded_x[:, :, :self.timesteps_in])
35         predicted_timesteps_out = predicted[:, :, -self.timesteps_out:]).transpose(1, 2).reshape(
36             (batch * self.timesteps_out, -1))
37         # encoded_x_timesteps_out = encoded_x[:, :, -self.timesteps_out:]).transpose(1, 2).reshape((batch*self.
38         timesteps_out, -1))
39         decoded_pred = self.decoder(predicted_timesteps_out).reshape((batch, self.timesteps_out, channels, length))
40         if self.training:
41             loss = torch.sum(torch.square(decoded_pred - X[:, -self.timesteps_out:])) / (batch * self.timesteps_out)
42             return loss
43         else:
44             return decoded_pred
45
46 if __name__ == '__main__':
47     timesteps_in = 6
48     timesteps_out = 6
49     latents = 64
50     enc = cpc_encoder_v2.Encoder(12, latents)
51     auto = TemporalConvNet(latents, [latents] * 3, kernel_size=3)
52     decoder = cpc_encoder_decoder_v2.Decoder(12, latents)
53     scpc = SimpleCPC(enc, decoder, auto, timesteps_in, timesteps_out, latents)
54     train = ecg_datasets2.ECGChallengeDatasetBatching('/media/julian/data/data/ECG/ptbxl_challenge',
55                                                     window_size=140, n_windows=timesteps_in + timesteps_out)
56     val = ecg_datasets2.ECGChallengeDatasetBatching('/media/julian/data/data/ECG/ptbxl_challenge',
57                                                    window_size=140, n_windows=timesteps_in + timesteps_out)
58     dataloader = DataLoader(train, batch_size=128, drop_last=True, num_workers=1)
59     valloader = DataLoader(val, batch_size=128, drop_last=True, num_workers=1)
60     optimizer = Adam(scpc.parameters())
61     scpc.train()
62     for i in range(2000):
63         avg_train_loss, avg_test_loss = 0, 0
64         for batch_idx, data in enumerate(dataloader):
65             optimizer.zero_grad()
66             loss = scpc(data.float())
67             loss.backward()
68             optimizer.step()
69             avg_train_loss = (avg_train_loss * batch_idx + loss.item()) / (batch_idx + 1)
70             print("loss {}".format(loss.item()))
71         print('EPOCH {} finished. Average train loss {}'.format(i + 1, avg_train_loss))
72         # for batch_idx, data in enumerate(dataloader):
73         #     loss = scpc(data.float())
74         #     avg_test_loss = (avg_test_loss*batch_idx+loss.item())/(batch_idx+1)
75         # print('EPOCH {} finished. Average validation loss {}'.format(i+1, avg_test_loss))
76     scpc.train(False)

```

B.3 architectures_various (folder)

B.3.1 architectures_various -> explain_network.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class ExplainLabel(nn.Module):
6
7     def __init__(self, baseline_model, normal_label_index=-1):
8         super().__init__()
9         self.model = baseline_model
10        self.normal_label_index = normal_label_index
11
12    def forward(self, X: torch.tensor, explain_class_index=-1, y=None):
13        X.requires_grad = True
14        if y is None:
15            output = self.model(X)
16            if explain_class_index > -1:
17                fake_truth = torch.zeros_like(output)
18                if self.normal_label_index >= 0:
19                    fake_truth[self.normal_label_index] = 1
20                else:
21                    fake_truth = output.clone()
22                    fake_truth[:, explain_class_index] = 0.0

```

```

23     loss = torch.sum(torch.square(fake_truth - output)) # Same as just sum of squared output in this case
24     loss.backward()
25     grad = torch.abs(X.grad)
26     X.grad = None
27     return output, grad
28 else:
29     accuracies, loss = self.model(X, y)
30     loss.backward()
31     grad = torch.abs(X.grad) # in Heat map distance is important not specific direction
32     X.grad = None
33     return accuracies, loss, grad

```

B.3.2 architectures_various -> explain_network2.py (code)

```

1 import torch
2 import cv2
3 from torch import nn, autograd
4 from torch.optim import Adam
5 from util.metrics import training_metrics, baseline_losses as bl
6 import functools
7 import numpy as np
8
9
10 class ExplainLabel(nn.Module):
11     def __init__(self, model, class_weights=None, cuda=True):
12         super().__init__()
13         self.model = model
14         self.class_weights = class_weights
15
16     def forward(self, X1: torch.tensor, y=None):
17         if y is None:
18             return self.model(X1)
19             # output = self.model(X)
20             # if explain_class_index > -1:
21             #     fake_truth = torch.zeros_like(output)
22             #     if self.normal_label_index>0:
23             #         fake_truth[self.normal_label_index] = 1
24             #     else:
25             #         fake_truth = output.clone()
26             #         fake_truth[:, explain_class_index] = 0.0
27             # loss = torch.sum(torch.square(fake_truth-output)) #Same as just sum of squared output in this case
28             # loss.backward()
29             # grad = torch.abs(X.grad)
30             # X.grad = None
31             # return output, grad
32     else:
33         grads = []
34         X = torch.tensor(X1, requires_grad=True, device=X1.device) # X1.clone().detach().requires_grad_(True)
35         X.retain_grad()
36         pred = self.model(X, y=None) # makes model return prediction instead of loss
37         if len(pred.shape) == 1: # hack for squeezed batch dimension
38             pred = pred.unsqueeze(0)
39         loss = bl.binary_cross_entropy(pred=pred, y=y,
40                                       weight=self.class_weights) # bl.multi_loss_function([bl.
41         binary_cross_entropy, bl.MSE_loss])(pred=pred, y=labels)
42         loss.backward()
43         # grad = autograd.grad(loss, X, create_graph=True, retain_graph=True, allow_unused=True)#
44         # print(grad)
45         # print('#####')
46         grad = X.grad
47         # grad = torch.abs(grad) #in Heat map distance is important not specific direction
48         X.grad = None
49         return pred, grad
50
51 class ExplainLabelLayer(nn.Module):
52     def __init__(self, model, class_weights=None, cuda=True, layer=None, guided=False):
53         super().__init__()
54         self.model = model
55         if guided:
56             convert_relu_to_custom_relu(self.model)
57             self.class_weights = class_weights
58         if type(layer) == str:
59             self.layer = functools.reduce(lambda o, n: getattr(o, n), [self.model] + layer.split('.'))
60         else:
61             self.layer = layer
62
63
64         self.guided = guided
65
66         self.gradients = None
67         self.data_gradient = None
68         self.activations = None
69         self._register_hooks()
70
71

```

```

72
73     def _register_hooks(self):
74         self.layer.register_forward_hook(self.save_activation)
75         self.layer.register_full_backward_hook(self.save_gradient)
76         # if not self.input_layer is None:
77         #     self.input_layer.register_full_backward_hook(self.save_data_gradient)
78
79     def save_activation(self, module, input, output):
80         self.activations = output.cpu().detach()
81
82     def save_gradient(self, module, grad_input, grad_output):
83         self.gradients = grad_output[0].cpu().detach() #last layer comes first
84         #
85     # def save_data_gradient(self, module, grad_input, grad_output):
86     #     self.data_gradient = grad_output[0].cpu().detach()
87
88     def forward(self, X1: torch.tensor, y=None):
89         self.gradients = None
90         self.activations = None
91         if self.guided:
92             X1.requires_grad = True
93             X1.grad = None
94         self.data_gradient = None
95         if y is None:
96             return self.model(X1)
97         else:
98
99             pred = self.model(X1, y=None) # makes model return prediction instead of loss
100            if len(pred.shape) == 1: # hack for squeezed batch dimension
101                pred = pred.unsqueeze(0)
102            loss = bl.binary_cross_entropy(pred=pred, y=y,
103                                         weight=self.class_weights) # bl.multi_loss_function([bl.
104                                         binary_cross_entropy, bl.MSE_loss])(pred=pred, y=labels)
105            loss.backward()
106            if self.guided:
107                self.data_gradient = X1.grad.cpu().detach()
108            # grad = torch.abs(grad) #in Heat map distance is important not specific direction
109            return pred
110
111     def get_gradcam_weights(self):
112         return torch.mean(self.gradients, dim=-1, keepdim=True) #over data dimension
113
114     def get_gradcam(self, target_size=[1, 4500], scale=False):
115         w = self.get_gradcam_weights()
116         B, C, _ = w.shape
117         x = torch.sum(w * self.activations, dim=1) #Sum channel dim
118         print('cam shape', x.shape)
119         x[x<0]=0
120         if scale:
121             x = x - x.min()
122             x = x / (1e-12 + x.max())
123         if target_size is None:
124             return x
125         else:
126             return torch.Tensor(scale_cam_image(x, target_size=target_size))
127
128     def get_gradcam_and_guided(self, target_size=[1, 4500]):
129         print('data shape', self.data_gradient.shape)
130         #self.data_gradient[self.data_gradient<0]=0.
131         self.data_gradient = torch.abs(self.data_gradient)
132         grad_img = torch.Tensor(scale_cam_image(self.data_gradient, target_size))
133         print(grad_img.shape)
134         B, L, C = grad_img.shape
135         target_size = [C, L]
136         cam = self.get_gradcam(target_size)
137         print(cam.shape)
138         return grad_img * cam
139
140 class F_GuidedRelu(torch.autograd.Function):
141     @staticmethod
142     def forward(ctx, input):
143         ctx.save_for_backward(input)
144         return input.clamp(min=0)
145
146     @staticmethod
147     def backward(ctx, grad_output):
148         input, = ctx.saved_tensors
149         grad_input = (grad_output>0).float() * (input>0).float() * grad_output
150         return grad_input
151
152
153 class GuidedRelu(nn.Module):
154     def __init__(self):
155         super(GuidedRelu, self).__init__()
156         self.relu = F_GuidedRelu.apply
157     def forward(self, x):
158         return self.relu(x)
159

```

```

160 def scale_cam_image(cam, target_size=None): #@https://github.com/jacobgil/pytorch-grab-cam/blob/770
161     f29027598a8bf3ef660e04d14c44770b4d03c/pytorch_grad_cam/base_cam.py#L136
162     result = []
163     for img in cam:
164         img = img - torch.min(img)
165         img = img / (1e-12 + torch.max(img))
166         if target_size is not None:
167             img = cv2.resize(img.numpy(), target_size)
168             result.append(img)
169             print('img; ', img.shape)
170     result = np.stack(result)
171     print('RES; ', result.shape)
172     return result
173
174 def convert_relu_to_custom_relu(model):
175     for child_name, child in model.named_children():
176         if isinstance(child, nn.ReLU):
177             setattr(model, child_name, GuidedRelu())
178         else:
179             convert_relu_to_custom_relu(child)

```

B.4 deprecated (folder)

B.4.1 deprecated -> architectures_baseline (folder)

B.4.1.1 deprecated -> architectures_baseline -> baseline_cnn_v0.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=7, dilation=1),
13            nn.ReLU(),
14            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=3),
15            nn.ReLU(),
16        )
17        self.downsample = nn.Conv2d(in_channels=9488, out_channels=1, kernel_size=1)
18
19        self.fc = nn.Linear(out_channels, out_classes)
20        # self.activation = nn.LogSoftmax(dim=1)
21        # self.criterion = nn.NLLLoss()
22        self.activation = nn.Sigmoid()
23        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
24
25    def forward(self, X, y=None):
26        if self.verbose: print('input shape', X.shape)
27        batch, window_size, channels = X.shape
28        X = X.transpose(1, 2)
29        if self.verbose: print(X.shape)
30        X = self.convs(X)
31        if self.verbose: print('x shape after conv', X.shape)
32        X = X.transpose(1, 2)
33        if self.verbose: print('x shape after transpose', X.shape)
34        X = self.downsample(X)
35        if self.verbose: print('x shape after downsample', X.shape)
36        X = self.fc(X).squeeze(1)
37        if self.verbose: print('x shape after fc', X.shape)
38        logits = self.activation(X)
39        if not y is None:
40            # loss = self.criterion(logits, y)
41            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
42            # loss = self.criterion(logits, torch.argmax(y, dim=1))
43            accuracies = []
44            mask = y != 0.0
45            inverse_mask = ~mask
46            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
47                inverse_mask) # zero fit goal
48            class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
49            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
50            accuracies.append(class_fit)
51            accuracies.append(zero_fit)
52

```

```

53         accuracies.append(
54             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
55             values
56             # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
57             count non zeros in accuracy?
58             accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
59                 self.n_out_classes * batch)) # correct if probability within 0.01
60             # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
61             return accuracies, loss
62     else:
63         return logits

```

B.4.1.2 deprecated -> architectures_baseline -> baseline_cnn_v0_1.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=7, dilation=1, stride=2),
13            nn.ReLU(),
14            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=3, stride=1),
15            nn.ReLU(),
16        )
17        self.downsample = nn.Conv2d(in_channels=4741, out_channels=1, kernel_size=1)
18
19        self.fc = nn.Linear(out_channels, out_classes)
20        # self.activation = nn.LogSoftmax(dim=1)
21        # self.criterion = nn.NLLLoss()
22        self.activation = nn.Sigmoid()
23        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
24
25    def forward(self, X, y=None):
26        if self.verbose: print('input shape', X.shape)
27        batch, window_size, channels = X.shape
28        x = X.transpose(1, 2)
29        if self.verbose: print(x.shape)
30        x = self.convs(x)
31        if self.verbose: print('x shape after conv', x.shape)
32        x = x.transpose(1, 2)
33        if self.verbose: print('x shape after transpose', x.shape)
34        x = self.downsample(x)
35        if self.verbose: print('x shape after downsample', x.shape)
36        x = self.fc(x).squeeze(1)
37        if self.verbose: print('x shape after fc', x.shape)
38        logits = self.activation(x)
39        if not y is None:
40            # loss = self.criterion(logits, y)
41            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
42            # loss = self.criterion(logits, torch.argmax(y, dim=1))
43            accuracies = []
44            mask = y != 0.0
45            inverse_mask = ~mask
46            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
47                inverse_mask) # zero fit goal
48            class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
49            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
50            accuracies.append(class_fit)
51            accuracies.append(zero_fit)
52
53            accuracies.append(
54                1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
55            values
56            # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
57            count non zeros in accuracy?
58            accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
59                self.n_out_classes * batch)) # correct if probability within 0.01
60            # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
61            return accuracies, loss
62    else:
63        return logits

```

B.4.1.3 deprecated -> architectures_baseline -> baseline_cnn_v0_2.py (code)

```

1 import torch
2 from torch import nn

```

```

3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv2d(in_channels=in_channels, out_channels=in_channels, kernel_size=7, dilation=1, stride=4),
13            nn.ReLU(),
14            nn.Conv2d(in_channels=in_channels, out_channels=in_channels, kernel_size=5, dilation=1, stride=3),
15            nn.ReLU(),
16            nn.Conv2d(in_channels=in_channels, out_channels=in_channels, kernel_size=5, dilation=1, stride=3),
17            nn.ReLU(),
18            nn.Conv2d(in_channels=in_channels, out_channels=32, kernel_size=3, dilation=1, stride=1),
19            nn.ReLU(),
20            nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, dilation=2, stride=1),
21            nn.ReLU(),
22            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, dilation=4, stride=1),
23            nn.ReLU()
24        )
25        self.downsample = nn.Conv2d(in_channels=248, out_channels=1, kernel_size=1)
26
27        self.fc = nn.Linear(64, out_classes)
28        # self.activation = nn.LogSoftmax(dim=1)
29        # self.criterion = nn.NLLLoss()
30        self.activation = nn.Sigmoid()
31        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
32
33    def forward(self, X, y=None):
34        if self.verbose: print('input shape', X.shape)
35        batch, window_size, channels = X.shape
36        x = X.transpose(1, 2)
37        if self.verbose: print(x.shape)
38        x = self.convs(x)
39        if self.verbose: print('x shape after conv', x.shape)
40        x = x.transpose(1, 2)
41        if self.verbose: print('x shape after transpose', x.shape)
42        x = self.downsample(x)
43        if self.verbose: print('x shape after downsample', x.shape)
44        x = self.fc(x).squeeze(1)
45        if self.verbose: print('x shape after fc', x.shape)
46        logits = self.activation(x)
47        if not y is None:
48            # loss = self.criterion(logits, y)
49            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
50            # loss = self.criterion(logits, torch.argmax(y, dim=1))
51            accuracies = []
52            mask = y != 0.0
53            inverse_mask = ~mask
54            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
55                inverse_mask) # zero fit goal
56            class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
57            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
58            accuracies.append(class_fit)
59            accuracies.append(zero_fit)
60
61            accuracies.append(
62                1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
63            values
64            # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
65            count non zeros in accuracy?
66            accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) /
67                (self.n_out_classes * batch)) # correct if probability within 0.01
68            # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
69        else:
70            return logits

```

B.4.1.4 deprecated -> architectures_baseline -> baseline_cnn_v0_3.py (code)

```

1 import torch
2 from torch import nn
3
4 from util.metrics import training_metrics as acm
5 from deprecated.data_storage import DataStorage
6
7
8 class BaselineNet(nn.Module):
9
10    def __init__(self, in_channels, out_channels, out_classes, verbose):
11        super().__init__()
12        self.n_out_classes = out_classes
13        self.verbose = verbose

```

```

14     self.convs = nn.Sequential(
15         nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=7, dilation=1, stride=4),
16         nn.ReLU(),
17         nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=5, dilation=1, stride=3),
18         nn.ReLU(),
19         nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=5, dilation=1, stride=3),
20         nn.ReLU(),
21         nn.Conv1d(in_channels=in_channels, out_channels=32, kernel_size=3, dilation=1, stride=1),
22         nn.ReLU(),
23         nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3, dilation=2, stride=1),
24         nn.ReLU(),
25         nn.Conv1d(in_channels=64, out_channels=128, kernel_size=3, dilation=4, stride=1),
26         nn.ReLU()
27     )
28     self.downsample = nn.Conv1d(in_channels=248, out_channels=1, kernel_size=1)
29
30     self.fc = nn.Linear(128, out_classes)
31     # self.activation = nn.LogSoftmax(dim=1)
32     # self.criterion = nn.NLLLoss()
33     self.data_storage = DataStorage('../temp')
34     self.activation = nn.Sigmoid()
35     self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
36
37
38 def forward(self, X, y=None):
39     if self.verbose: print('input shape', X.shape)
40     batch, window_size, channels = X.shape
41     x = X.transpose(1, 2)
42     if self.verbose: print(x.shape)
43     x = self.convs(x)
44     if self.verbose: print('x shape after conv', x.shape)
45     x = x.transpose(1, 2)
46     if self.verbose: print('x shape after transpose', x.shape)
47     x = self.downsample(x)
48     if self.verbose: print('x shape after downsample', x.shape)
49     x = self.fc(x).squeeze(1)
50     if self.verbose: print('x shape after fc', x.shape)
51     logits = self.activation(x)
52     if not y is None:
53         # loss = self.criterion(logits, y)
54         loss = torch.sum(torch.square(y - logits)) # Simple own implementation
55         # loss = self.criterion(logits, torch.argmax(y, dim=1))
56         accuracies = []
57         mask = y != 0.0
58         inverse_mask = ~mask
59         mask_sum = torch.sum(mask)
60         inverse_mask_sum = torch.sum(inverse_mask)
61         zero_fit = 1.0 - torch.sum(
62             torch.square(y[inverse_mask] - logits[inverse_mask])) / inverse_mask_sum # zero fit goal
63         class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / mask_sum # class fit goal
64         accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
65         accuracies.append(class_fit)
66         accuracies.append(zero_fit)
67
68         accuracies.append(
69             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
70         values
71         # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
72         # count non zeros in accuracy?
73         accuracies.append(acm.micro_avg_precision_score(y, logits))
74         accuracies.append(acm.micro_avg_recall_score(y, logits))
75         accuracies.append(acm.f1_score(y, logits))
76         accuracies.append(acm.accuracy(y, logits))
77         # Counts the classes that have been correctly predicted - wrongly predicted with certain prob
78         accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
79             self.n_out_classes * batch)) # correct if probabilty within 0.01
80         # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
81         return accuracies, loss, [acm.tp_score_label(y, logits), acm.fp_score_label(y, logits),
82                                 acm.tn_score_label(y, logits), acm.fn_score_label(y, logits)]
83     else:
84         return logits

```

B.4.1.5 deprecated -> architectures_baseline -> baseline_cnn_v1.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose=False):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.convs = nn.Sequential(
11            nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=10, stride=5),

```

```

12     nn.BatchNorm1d(in_channels),
13     nn.ReLU(),
14     nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=8, stride=4),
15     nn.BatchNorm1d(in_channels),
16     nn.ReLU(),
17     nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=6, stride=3),
18     nn.BatchNorm1d(in_channels),
19     nn.ReLU(),
20     nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=5, stride=2),
21     nn.BatchNorm1d(in_channels),
22     nn.ReLU(),
23     nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=4, stride=2),
24     nn.BatchNorm1d(in_channels),
25     nn.ReLU(),
26     nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=3, stride=1),
27     nn.BatchNorm1d(in_channels),
28     nn.ReLU(),
29 )
30 self.pooling = nn.AdaptiveAvgPool1d(1) # TODO: FIX and try max
31
32 self.fc = nn.Linear(in_channels, out_classes)
# self.activation = nn.LogSoftmax(dim=1)
# self.criterion = nn.NLLLoss()
33 self.activation = nn.Sigmoid()
34 self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
35
36 def forward(self, X, y=None):
37     # print('input shape', X.shape)
38     batch, window_size, channels = X.shape
39     x = X.transpose(1, 2)
40     # print(x.shape)
41     x = self.convs(x)
42     # print('x shape after convs', x.shape)
43     x = self.pooling(x).squeeze(2)
44     # print('x shape after pooling', x.shape)
45     x = self.fc(x)
46     # print('x shape after fc', x.shape)
47     logits = self.activation(x)
48
49     if not y is None:
50         # loss = self.criterion(logits, y)
51         loss = torch.sum(torch.square(y - logits)) # Simple own implementation
52         # loss = self.criterion(logits, torch.argmax(y, dim=1))
53         accuracies = []
54         accuracies.append(0.0)
55         mask = y != 0.0
56         inverse_mask = ~mask
57         zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
58             inverse_mask) # zero fit goal
59         class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
60         accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
61         accuracies.append(class_fit)
62         accuracies.append(zero_fit)
63
64         accuracies.append(
65             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
66         values
67         # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
68         # count non zeros in accuracy?
69         accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
70             self.n_out_classes * batch)) # correct if probability within 0.01
71         # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
72         return accuracies, loss
73     else:
74         return logits

```

B.4.1.6 deprecated -> architectures_baseline -> baseline_cnn_v10.py (code)

```

1 import torch
2 from torch import nn
3
4 from external.tcn.TCN.tcn import TemporalConvNet
5
6
7 class BaselineNet(nn.Module):
8
9     def __init__(self, in_channels, out_channels, out_classes, verbose):
10        super().__init__()
11        self.n_out_classes = out_classes
12        self.verbose = verbose
13        self.tcn = TemporalConvNet(in_channels, [in_channels * 2 ** 1, in_channels * 2, out_classes], kernel_size=3,
14                                  dropout=0.2)
15        self.downsample = nn.Conv1d(in_channels=4500, out_channels=1, kernel_size=1)
16
17        self.fc = nn.Linear(out_classes, out_classes)
18        # self.activation = nn.LogSoftmax(dim=1)
19        # self.criterion = nn.NLLLoss()
20        self.activation = nn.Sigmoid()

```

```

21     self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
22
23 def forward(self, X, y=None):
24     if self.verbose: print('input shape', X.shape)
25     batch, window_size, channels = X.shape
26     x = X.transpose(1, 2)
27     if self.verbose: print(x.shape)
28     x = self.tcn(x)
29     if self.verbose: print('x shape after conv', x.shape)
30     x = x.transpose(1, 2)
31     if self.verbose: print('x shape after transpose', x.shape)
32     x = self.downsample(x)
33     if self.verbose: print('x shape after downsample', x.shape)
34     x = self.fc(x).squeeze(1)
35     if self.verbose: print('x shape after fc', x.shape)
36     logits = self.activation(x)
37     if not y is None:
38         # loss = self.criterion(logits, y)
39         loss = torch.sum(torch.square(y - logits)) # Simple own implementation
40         # loss = self.criterion(logits, torch.argmax(y, dim=1))
41         accuracies = []
42         mask = y != 0.0
43         inverse_mask = ~mask
44         zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
45             inverse_mask) # zero fit goal
46         class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
47         accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
48         accuracies.append(class_fit)
49         accuracies.append(zero_fit)
50
51         accuracies.append(
52             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
53         values
54         # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
55         # count non zeros in accuracy?
56         accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
57             self.n_out_classes * batch)) # correct if probability within 0.01
58         # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
59         return accuracies, loss
60     else:
61         return logits

```

B.4.1.7 deprecated -> architectures_baseline -> baseline_cnn_v11.py (code)

```

1 import torch
2 from torch import nn
3
4 from external.tcn.TCN.tcn import TemporalConvNet
5
6
7 class BaselineNet(nn.Module):
8
9     def __init__(self, in_channels, out_channels, out_classes, verbose):
10        super().__init__()
11        self.n_out_classes = out_classes
12        self.verbose = verbose
13        self.tcn = TemporalConvNet(in_channels, [in_channels * 2 ** 1, in_channels * 2, out_classes], kernel_size=3,
14                                  dropout=0.2)
15        # self.downsample = nn.Conv1d(in_channels=9500, out_channels=1, kernel_size=1)
16
17        self.fc = nn.Linear(out_classes, out_classes)
18        # self.activation = nn.LogSoftmax(dim=1)
19        # self.criterion = nn.NLLLoss()
20        self.activation = nn.Sigmoid()
21        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
22
23     def forward(self, X, y=None):
24        if self.verbose: print('input shape', X.shape)
25        batch, window_size, channels = X.shape
26        x = X.transpose(1, 2)
27        if self.verbose: print(x.shape)
28        x = self.tcn(x)
29        if self.verbose: print('x shape after tcn', x.shape)
30        x = self.fc(x[:, :, -1]) # Like in https://github.com/locuslab/TCN/blob/master/TCN/mnist_pixel/model.py TODO: why is
31        # that okay?
32        if self.verbose: print('x shape after fc', x.shape)
33        logits = self.activation(x)
34        if not y is None:
35            # loss = self.criterion(logits, y)
36            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
37            # loss = self.criterion(logits, torch.argmax(y, dim=1))
38            accuracies = []
39            mask = y != 0.0
40            inverse_mask = ~mask
41            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(

```

```

42     inverse_mask) # zero fit goal
43     class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
44     accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
45     accuracies.append(class_fit)
46     accuracies.append(zero_fit)
47
48     accuracies.append(
49         1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
50     values
51     # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
52     # count non zeros in accuracy?
53     accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
54         self.n_out_classes * batch)) # correct if probability within 0.01
55     # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
56     return accuracies, loss
else:
    return logits

```

B.4.1.8 deprecated -> architectures_baseline -> baseline_cnn_v12.py (code)

```

1 import torch
2 from torch import nn
3
4 from external.tcn.TCN.tcn import TemporalConvNet
5
6
7 class BaselineNet(nn.Module):
8
9     def __init__(self, in_channels, out_channels, out_classes, verbose):
10        super().__init__()
11        self.n_out_classes = out_classes
12        self.verbose = verbose
13        self.tcn = TemporalConvNet(1, [in_channels * 2 ** 0, in_channels * 2 ** 2, in_channels * 2 ** 3], kernel_size
14 =3,
15             dropout=0.2)
16        # self.downsample = nn.Conv2d(in_channels=9500, out_channels=1, kernel_size=1)
17
18        self.fc = nn.Linear(in_channels * 2 ** 3, out_classes)
19        # self.activation = nn.LogSoftmax(dim=1)
20        # self.criterion = nn.NLLLoss()
21        self.activation = nn.Sigmoid()
22        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
23
24    def forward(self, x, y=None):
25        if self.verbose: print('input shape', x.shape)
26        batch, window_size, channels = x.shape
27        x = torch.unsqueeze(torch.flatten(x, start_dim=1), 1) # .transpose(1, 2)
28        if self.verbose: print('after flatten + unsqueeze', x.shape)
29        x = self.tcn(x)
30        if self.verbose: print('x shape after tcn', x.shape)
31        x = self.fc(x[:, :, -1]) # Like in https://github.com/locuslab/TCN/blob/master/TCN/mnist_pixel/model.py TODO: why is
32        that okay?
33        if self.verbose: print('x shape after fc', x.shape)
34        logits = self.activation(x)
35        if not y is None:
36            # loss = self.criterion(logits, y)
37            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
38            # loss = self.criterion(logits, torch.argmax(y, dim=1))
39            accuracies = []
40            mask = y != 0.0
41            inverse_mask = ~mask
42            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
43                inverse_mask) # zero fit goal
44            class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
45            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
46            accuracies.append(class_fit)
47            accuracies.append(zero_fit)
48
49            accuracies.append(
50                1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
51            values
52            # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
53            # count non zeros in accuracy?
54            accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
55                self.n_out_classes * batch)) # correct if probability within 0.01
56            # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
57            return accuracies, loss
else:
    return logits

```

B.4.1.9 deprecated -> architectures_baseline -> baseline_cnn_v13.py (code)

```

1 import torch
2 from torch import nn
3 from torch.nn.utils import weight_norm
4
5
6 class BaselineNet(nn.Module):
7
8     def __init__(self, in_channels, out_channels, out_classes, verbose):
9         super().__init__()
10        self.n_out_classes = out_classes
11        self.verbose = verbose
12        self.tcn_block = nn.Sequential(
13            weight_norm(nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=3, stride=2, padding
14=0,
15                           dilation=1)),
16            nn.ReLU(),
17            nn.Dropout(0.2),
18
19            weight_norm(nn.Conv1d(in_channels=in_channels, out_channels=in_channels, kernel_size=3, stride=1, padding
20=0,
21                           dilation=2)),
22            nn.ReLU(),
23            nn.Dropout(0.2),
24        )
25        self.res_downsample = nn.Conv1d(in_channels=9500, out_channels=4745, kernel_size=1)
26        # self.downsample = nn.Conv1d(in_channels=4745, out_channels=1, kernel_size=1)
27        self.fc = nn.Linear(56940, out_classes)
28        self.activation = nn.Sigmoid()
29        self.criterion = nn.BCELoss()
30
31    def forward(self, x, y=None):
32        if self.verbose: print('input shape', x.shape)
33        batch, window_size, channels = x.shape
34        x_res = x.clone()
35        x = x.transpose(1, 2) # torch.unsqueeze(torch.flatten(x, start_dim=1), 1)#
36        if self.verbose: print('after transpose', x.shape)
37        x = self.tcn_block(x)
38        if self.verbose: print('x shape after tcn', x.shape)
39        x_res = self.res_downsample(x_res)
40        if self.verbose: print('x_res shape after residual downsample', x_res.shape)
41        x = x.transpose(1, 2) + x_res
42        if self.verbose: print('x shape after res add', x.shape)
43        x = torch.flatten(x, start_dim=1)
44        if self.verbose: print('x shape after flatten', x.shape)
45        x = self.fc(
46            x) # Like in https://github.com/locuslab/TCN/blob/master/TCN/mnist_pixel/model.py TODO: why is that okay?
47        if self.verbose: print('x shape after fc', x.shape)
48        logits = self.activation(x)
49        if not y is None:
50            # loss = self.criterion(logits, y)
51            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
52            # loss = self.criterion(logits, torch.argmax(y, dim=1))
53            accuracies = []
54            mask = y != 0.0
55            inverse_mask = ~mask
56            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
57                inverse_mask) # zero fit goal
58            class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
59            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
60            accuracies.append(class_fit)
61            accuracies.append(zero_fit)
62            accuracies.append(
63                1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
64            values
65            # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
66            count non zeros in accuracy?
67            accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
68                self.n_out_classes * batch)) # correct if probability within 0.01
69            # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
70            return accuracies, loss
71        else:
72            return logits

```

B.4.1.10 deprecated -> architectures_baseline -> baseline_cnn_v14.py (code)

```

1 import torch
2 from torch import nn
3
4 from external.multi_scale_ori import MSResNet
5
6
7 class BaselineNet(nn.Module):
8
9     def __init__(self, in_channels, out_channels, out_classes, verbose):
10        super().__init__()

```

```

11     self.n_out_classes = out_classes
12     self.verbose = verbose
13     self.msresnet = MSResNet(in_channels, num_classes=out_classes)
14     self.activation = nn.Sigmoid()
15
16     def forward(self, x, y=None):
17         if self.verbose: print('input shape', x.shape)
18         batch, window_size, channels = x.shape
19         x = x.transpose(1, 2) # torch.unsqueeze(torch.flatten(x, start_dim=1), 1)#
20         if self.verbose: print('after transpose', x.shape)
21         fc_x, x = self.msresnet(x)
22         # if self.verbose: print('x shape after resnet', x.shape)
23         # x = self.fc(x)
24         # if self.verbose: print('x shape after fc', x.shape)
25         logits = self.activation(fc_x)
26         if not y is None:
27             # loss = self.criterion(logits, y)
28             loss = torch.sum(torch.square(y - logits)) # Simple own implementation
29             # loss = self.criterion(logits, torch.argmax(y, dim=1))
30             accuracies = []
31             mask = y != 0.0
32             inverse_mask = ~mask
33             zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
34                 inverse_mask) # zero fit goal
35             class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
36             accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
37             accuracies.append(class_fit)
38             accuracies.append(zero_fit)
39
40             accuracies.append(
41                 1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
42             values
43             # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
44             # count non zeros in accuracy?
45             accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
46                 self.n_out_classes * batch)) # correct if probability within 0.01
47             # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
48             return accuracies, loss
49         else:
50             return logits

```

B.4.1.11 deprecated -> architectures_baseline -> baseline_cnn_v2.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv1d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, dilation=1),
13            nn.BatchNorm1d(out_channels),
14            nn.ReLU(),
15            nn.MaxPool1d(3),
16            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=2),
17            nn.BatchNorm1d(out_channels),
18            nn.ReLU(),
19            nn.MaxPool1d(3),
20            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=4),
21            nn.BatchNorm1d(out_channels),
22            nn.ReLU(),
23            nn.MaxPool1d(3),
24            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=8),
25            nn.BatchNorm1d(out_channels),
26            nn.ReLU(),
27            nn.MaxPool1d(3),
28            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=16),
29            nn.BatchNorm1d(out_channels),
30            nn.ReLU(),
31            nn.MaxPool1d(3),
32        )
33        self.pooling = nn.AdaptiveAvgPool1d(1) # TODO: FIX and try max
34
35        self.fc = nn.Linear(out_channels, out_classes)
36        # self.activation = nn.LogSoftmax(dim=1)
37        # self.criterion = nn.NLLLoss()
38        self.activation = nn.Sigmoid()
39        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
40
41    def forward(self, X, y=None):
42        if self.verbose: print('input shape', X.shape)
43        batch, window_size, channels = X.shape

```

```

44     x = X.transpose(1, 2)
45     if self.verbose: print(x.shape)
46     x = self.convs(x)
47     if self.verbose: print('x shape after convs', x.shape)
48     x = self.pooling(x).squeeze(2)
49     if self.verbose: print('x shape after pooling', x.shape)
50     x = self.fc(x)
51     if self.verbose: print('x shape after fc', x.shape)
52     logits = self.activation(x)
53     if not y is None:
54         # loss = self.criterion(logits, y)
55         loss = torch.sum(torch.square(y - logits)) # Simple own implementation
56         # loss = self.criterion(logits, torch.argmax(y, dim=1))
57         accuracies = []
58         mask = y != 0.0
59         inverse_mask = ~mask
60         zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
61             inverse_mask) # zero fit goal
62         class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
63         accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
64         accuracies.append(class_fit)
65         accuracies.append(zero_fit)
66
67         accuracies.append(
68             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
69         values
70         # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
71         # count non zeros in accuracy?
72         accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
73             self.n_out_classes * batch)) # correct if probability within 0.01
74         # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
75     return accuracies, loss
76 else:
77     return logits

```

B.4.1.12 deprecated -> architectures_baseline -> baseline_cnn_v3.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv3d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=1),
13            nn.BatchNorm3d(out_channels),
14            nn.ReLU(),
15            nn.Conv3d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=2),
16            nn.BatchNorm3d(out_channels),
17            nn.ReLU(),
18            nn.Conv3d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=4),
19            nn.BatchNorm3d(out_channels),
20            nn.ReLU(),
21            nn.Conv3d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=8),
22            nn.BatchNorm3d(out_channels),
23            nn.ReLU(),
24        )
25        self.downsample = nn.Conv1d(in_channels=9470, out_channels=1, kernel_size=1) # 1x1 Conv
26        self.fc = nn.Linear(out_channels, out_classes)
27        # self.activation = nn.LogSoftmax(dim=1)
28        # self.criterion = nn.NLLLoss()
29        self.activation = nn.Sigmoid()
30        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
31
32    def forward(self, X, y=None):
33        if self.verbose: print('input shape', X.shape)
34        batch, window_size, channels = X.shape
35        X = X.transpose(1, 2)
36        if self.verbose: print(x.shape)
37        x = self.convs(x)
38        if self.verbose: print('x shape after conv', x.shape)
39        x = self.downsample(x.transpose(1, 2))
40        if self.verbose: print('x shape after downsample', x.shape)
41        x = self.fc(x).squeeze(1)
42        if self.verbose: print('x shape after fc', x.shape)
43        logits = self.activation(x)
44        if not y is None:
45            # loss = self.criterion(logits, y)
46            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
47            # loss = self.criterion(logits, torch.argmax(y, dim=1))
48            accuracies = []
49            mask = y != 0.0

```

```

50     inverse_mask = ~mask
51     zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
52         inverse_mask) # zero fit goal
53     class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
54     accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
55     accuracies.append(class_fit)
56     accuracies.append(zero_fit)
57
58     accuracies.append(
59         1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
60     values
61     # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
62     # count non zeros in accuracy?
63     accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
64         self.n_out_classes * batch)) # correct if probability within 0.01
65     # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
66     return accuracies, loss
67 else:
68     return logits

```

B.4.1.13 deprecated -> architectures_baseline -> baseline_cnn_v4.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=1),
13            nn.ReLU(),
14            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=2),
15            nn.ReLU(),
16            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=4),
17            nn.ReLU(),
18            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, stride=1, dilation=8),
19            nn.ReLU(),
20        )
21        self.downsample = nn.Conv2d(in_channels=9470, out_channels=1, kernel_size=1) # 1x1 Conv
22        self.fc = nn.Linear(out_channels, out_classes)
23        # self.activation = nn.LogSoftmax(dim=1)
24        # self.criterion = nn.NLLLoss()
25        self.activation = nn.Sigmoid()
26        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
27
28    def forward(self, X, y=None):
29        if self.verbose: print('input shape', X.shape)
30        batch, window_size, channels = X.shape
31        x = X.transpose(1, 2)
32        if self.verbose: print(x.shape)
33        x = self.convs(x)
34        if self.verbose: print('x shape after conv', x.shape)
35        x = x.transpose(1, 2)
36        if self.verbose: print('x shape after transpose', x.shape)
37        x = self.downsample(x)
38        if self.verbose: print('x shape after downsample', x.shape)
39        x = self.fc(x).squeeze(1)
40        if self.verbose: print('x shape after fc', x.shape)
41        logits = self.activation(x)
42        if not y is None:
43            # loss = self.criterion(logits, y)
44            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
45            # loss = self.criterion(logits, torch.argmax(y, dim=1))
46            accuracies = []
47            mask = y != 0.0
48            inverse_mask = ~mask
49            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
50                inverse_mask) # zero fit goal
51            class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
52            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
53            accuracies.append(class_fit)
54            accuracies.append(zero_fit)
55
56            accuracies.append(
57                1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
58            values
59            # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
60            # count non zeros in accuracy?
61            accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
62                self.n_out_classes * batch)) # correct if probability within 0.01
63            # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
64            return accuracies, loss

```

```

63     else:
64         return logits

```

B.4.1.14 deprecated -> architectures_baseline -> baseline_cnn_v5.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv1d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, dilation=1),
13            nn.ReLU(),
14            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=2),
15            nn.ReLU(),
16            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=4),
17            nn.ReLU(),
18            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=8),
19            nn.ReLU(),
20        )
21        self.downsample = nn.Conv1d(in_channels=out_channels, out_channels=1, kernel_size=1)
22
23        self.fc = nn.Linear(9470, out_classes)
24        # self.activation = nn.LogSoftmax(dim=1)
25        # self.criterion = nn.NLLLoss()
26        self.activation = nn.Sigmoid()
27        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
28
29    def forward(self, X, y=None):
30        if self.verbose: print('input shape', X.shape)
31        batch, window_size, channels = X.shape
32        x = X.transpose(1, 2)
33        if self.verbose: print(x.shape)
34        x = self.convs(x)
35        if self.verbose: print('x shape after convs', x.shape)
36        x = self.downsample(x).squeeze(1)
37        if self.verbose: print('x shape after downsample', x.shape)
38        x = self.fc(x)
39        if self.verbose: print('x shape after fc', x.shape)
40        logits = self.activation(x)
41        if not y is None:
42            # loss = self.criterion(logits, y)
43            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
44            # loss = self.criterion(logits, torch.argmax(y, dim=1))
45            accuracies = []
46            mask = y != 0.0
47            inverse_mask = ~mask
48            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
49                inverse_mask) # zero fit goal
50            class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
51            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
52            accuracies.append(class_fit)
53            accuracies.append(zero_fit)
54
55            accuracies.append(
56                1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
values
57            # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
count non zeros in accuracy?
58            accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
59                self.n_out_classes * batch)) # correct if probability within 0.01
60            # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
61            return accuracies, loss
62        else:
63            return logits

```

B.4.1.15 deprecated -> architectures_baseline -> baseline_cnn_v6.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes

```

```

10    self.verbose = verbose
11    self.convs = nn.Sequential(
12        nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, dilation=1),
13        nn.ReLU(),
14        nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=2),
15        nn.ReLU(),
16        nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=4),
17        nn.ReLU(),
18        nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=8),
19        nn.ReLU(),
20        nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=16),
21        nn.ReLU(),
22        nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=32),
23        nn.ReLU(),
24    )
25    self.downsample = nn.Conv2d(in_channels=9374, out_channels=1, kernel_size=1)
26
27    self.fc = nn.Linear(out_channels, out_classes)
28    # self.activation = nn.LogSoftmax(dim=1)
29    # self.criterion = nn.NLLLoss()
30    self.activation = nn.Sigmoid()
31    self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
32
33    def forward(self, X, y=None):
34        if self.verbose: print('input shape', X.shape)
35        batch, window_size, channels = X.shape
36        x = X.transpose(1, 2)
37        if self.verbose: print(x.shape)
38        x = self.convs(x)
39        if self.verbose: print('x shape after conv', x.shape)
40        x = x.transpose(1, 2)
41        if self.verbose: print('x shape after transpose', x.shape)
42        x = self.downsample(x)
43        if self.verbose: print('x shape after downsample', x.shape)
44        x = self.fc(x).squeeze(1)
45        if self.verbose: print('x shape after fc', x.shape)
46        logits = self.activation(x)
47        if not y is None:
48            # loss = self.criterion(logits, y)
49            loss = torch.sum(torch.square(y - logits)) # Simple own implementation
50            # loss = self.criterion(logits, torch.argmax(y, dim=1))
51            accuracies = []
52            mask = y != 0.0
53            inverse_mask = ~mask
54            zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
55                inverse_mask) # zero fit goal
56            class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
57            accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
58            accuracies.append(class_fit)
59            accuracies.append(zero_fit)
60
61            accuracies.append(
62                1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
63            values
64            # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
65            # count non zeros in accuracy?
66            accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
67                self.n_out_classes * batch)) # correct if probabilt within 0.01
68            # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
69        return accuracies, loss
70    else:
71        return logits

```

B.4.1.16 deprecated -> architectures_baseline -> baseline_cnn_v7.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=7, dilation=1, stride=2),
13            nn.ReLU(),
14            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=2),
15            nn.ReLU(),
16            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=4),
17            nn.ReLU(),
18            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=8),
19            nn.ReLU(),
20            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=16),
21            nn.ReLU(),

```

```

22     )
23     self.downsample = nn.Conv1d(in_channels=4687, out_channels=1, kernel_size=1)
24
25     self.fc = nn.Linear(out_channels, out_classes)
26     # self.activation = nn.LogSoftmax(dim=1)
27     # self.criterion = nn.NLLLoss()
28     self.activation = nn.Sigmoid()
29     self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
30
31 def forward(self, X, y=None):
32     if self.verbose: print('input shape', X.shape)
33     batch, window_size, channels = X.shape
34     x = X.transpose(1, 2)
35     if self.verbose: print(x.shape)
36     x = self.convs(x)
37     if self.verbose: print('x shape after conv', x.shape)
38     x = x.transpose(1, 2)
39     if self.verbose: print('x shape after transpose', x.shape)
40     x = self.downsample(x)
41     if self.verbose: print('x shape after downsample', x.shape)
42     x = self.fc(x).squeeze(1)
43     if self.verbose: print('x shape after fc', x.shape)
44     logits = self.activation(x)
45     if not y is None:
46         # loss = self.criterion(logits, y)
47         loss = torch.sum(torch.square(y - logits)) # Simple own implementation
48         # loss = self.criterion(logits, torch.argmax(y, dim=1))
49         accuracies = []
50         mask = y != 0.0
51         inverse_mask = ~mask
52         zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
53             inverse_mask) # zero fit goal
54         class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
55         accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
56         accuracies.append(class_fit)
57         accuracies.append(zero_fit)
58
59         accuracies.append(
60             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
61         values
62         # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
63         # count non zeros in accuracy?
64         accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
65             self.n_out_classes * batch)) # correct if probability within 0.01
66         # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
67         return accuracies, loss
68     else:
69         return logits

```

B.4.1.17 deprecated -> architectures_baseline -> baseline_cnn_v8.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv1d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, dilation=1),
13            nn.BatchNorm1d(out_channels),
14            nn.ReLU(),
15            nn.MaxPool1d(3),
16            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=2),
17            nn.BatchNorm1d(out_channels),
18            nn.ReLU(),
19            nn.MaxPool1d(3),
20            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=4),
21            nn.BatchNorm1d(out_channels),
22            nn.ReLU(),
23            nn.MaxPool1d(3),
24            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=8),
25            nn.BatchNorm1d(out_channels),
26            nn.ReLU(),
27        )
28        self.downsample = nn.Conv1d(in_channels=332, out_channels=1, kernel_size=1)
29
30        self.fc = nn.Linear(out_channels, out_classes)
31        # self.activation = nn.LogSoftmax(dim=1)
32        # self.criterion = nn.NLLLoss()
33        self.activation = nn.Sigmoid()
34        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
35

```

```

36     def forward(self, x, y=None):
37         if self.verbose: print('input shape', x.shape)
38         batch, window_size, channels = x.shape
39         x = x.transpose(1, 2)
40         if self.verbose: print(x.shape)
41         x = self.convs(x)
42         if self.verbose: print('x shape after conv', x.shape)
43         x = x.transpose(1, 2)
44         if self.verbose: print('x shape after transpose', x.shape)
45         x = self.downsample(x)
46         if self.verbose: print('x shape after downsample', x.shape)
47         x = self.fc(x).squeeze(1)
48         if self.verbose: print('x shape after fc', x.shape)
49         logits = self.activation(x)
50         if not y is None:
51             # loss = self.criterion(logits, y)
52             loss = torch.sum(torch.square(y - logits)) # Simple own implementation
53             # loss = self.criterion(logits, torch.argmax(y, dim=1))
54             accuracies = []
55             mask = y != 0.0
56             inverse_mask = ~mask
57             zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
58                 inverse_mask) # zero fit goal
59             class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
60             accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
61             accuracies.append(class_fit)
62             accuracies.append(zero_fit)
63
64             accuracies.append(
65                 1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
66             values
67             # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
68             # count non zeros in accuracy?
69             accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
70                 self.n_out_classes * batch)) # correct if probability within 0.01
71             # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
72             return accuracies, loss
73         else:
74             return logits

```

B.4.1.18 deprecated -> architectures_baseline -> baseline_cnn_v9.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class BaselineNet(nn.Module):
6
7     def __init__(self, in_channels, out_channels, out_classes, verbose):
8         super().__init__()
9         self.n_out_classes = out_classes
10        self.verbose = verbose
11        self.convs = nn.Sequential(
12            nn.Conv1d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, dilation=1),
13            nn.BatchNorm1d(out_channels),
14            nn.ReLU(),
15            nn.MaxPool1d(3),
16            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=1),
17            nn.BatchNorm1d(out_channels),
18            nn.ReLU(),
19            nn.MaxPool1d(3),
20            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=1),
21            nn.BatchNorm1d(out_channels),
22            nn.ReLU(),
23            nn.MaxPool1d(3),
24            nn.Conv1d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, dilation=1),
25            nn.BatchNorm1d(out_channels),
26            nn.ReLU(),
27        )
28        self.downsample = nn.Conv1d(in_channels=348, out_channels=1, kernel_size=1)
29
30        self.fc = nn.Linear(out_channels, out_classes)
31        # self.activation = nn.LogSoftmax(dim=1)
32        # self.criterion = nn.NLLLoss()
33        self.activation = nn.Sigmoid()
34        self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
35
36    def forward(self, x, y=None):
37        if self.verbose: print('input shape', x.shape)
38        batch, window_size, channels = x.shape
39        x = x.transpose(1, 2)
40        if self.verbose: print(x.shape)
41        x = self.convs(x)
42        if self.verbose: print('x shape after conv', x.shape)
43        x = x.transpose(1, 2)
44        if self.verbose: print('x shape after transpose', x.shape)

```

```

45     x = self.downsample(x)
46     if self.verbose: print('x shape after downsample', x.shape)
47     x = self.fc(x).squeeze(1)
48     if self.verbose: print('x shape after fc', x.shape)
49     logits = self.activation(x)
50     if not y is None:
51         # loss = self.criterion(logits, y)
52         loss = torch.sum(torch.square(y - logits)) # Simple own implementation
53         # loss = self.criterion(logits, torch.argmax(y, dim=1))
54         accuracies = []
55         mask = y != 0.0
56         inverse_mask = ~mask
57         zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
58             inverse_mask) # zero fit goal
59         class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
60         accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
61         accuracies.append(class_fit)
62         accuracies.append(zero_fit)
63
64         accuracies.append(
65             1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
66         values
67         # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
68         # count non zeros in accuracy?
69         accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) / (
70             self.n_out_classes * batch)) # correct if probability within 0.01
71         # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
72         return accuracies, loss
73     else:
74         return logits

```

B.4.1.19 deprecated -> architectures_baseline -> baseline_convencoder.py (code)

```

1 import torch
2 import torch.nn.functional as F
3 from torch import nn
4
5
6 class BaselineNet(nn.Module):
7
8     def __init__(self, in_channels, latent_size, out_classes):
9         super().__init__()
10        self.n_out_classes = out_classes
11        self.convs = nn.Sequential(
12            nn.Conv1d(in_channels=in_channels, out_channels=latent_size, kernel_size=5, stride=4),
13            nn.BatchNorm1d(latent_size),
14            nn.ReLU(),
15            nn.Conv1d(in_channels=latent_size, out_channels=latent_size, kernel_size=4, stride=3),
16            nn.BatchNorm1d(latent_size),
17            nn.ReLU(),
18            nn.Conv1d(in_channels=latent_size, out_channels=latent_size, kernel_size=3, stride=2),
19            nn.BatchNorm1d(latent_size),
20            nn.ReLU(),
21            nn.Conv1d(in_channels=latent_size, out_channels=latent_size, kernel_size=3, stride=2),
22            nn.BatchNorm1d(latent_size),
23            nn.ReLU(),
24            nn.Conv1d(in_channels=latent_size, out_channels=latent_size, kernel_size=2, stride=1),
25            nn.BatchNorm1d(latent_size),
26            nn.ReLU(),
27        )
28        self.pooling = nn.AdaptiveAvgPool1d(1)
29        self.t_convs = nn.Sequential(
30            nn.ConvTranspose1d(in_channels=latent_size, out_channels=latent_size, kernel_size=2, stride=1,
31                              output_padding=0),
32            nn.BatchNorm1d(latent_size),
33            nn.ReLU(),
34            nn.ConvTranspose1d(in_channels=latent_size, out_channels=latent_size, kernel_size=3, stride=2,
35                              output_padding=1),
36            nn.BatchNorm1d(latent_size),
37            nn.ReLU(),
38            nn.ConvTranspose1d(in_channels=latent_size, out_channels=latent_size, kernel_size=3, stride=2,
39                              output_padding=1),
40            nn.BatchNorm1d(latent_size),
41            nn.ReLU(),
42            nn.ConvTranspose1d(in_channels=latent_size, out_channels=latent_size, kernel_size=4, stride=3,
43                              output_padding=0),
44            nn.BatchNorm1d(latent_size),
45            nn.ReLU(),
46            nn.ConvTranspose1d(in_channels=latent_size, out_channels=in_channels, kernel_size=5, stride=4,
47                              output_padding=3),
48            nn.BatchNorm1d(in_channels),
49            nn.ReLU(),
50        )
51        self.fc = nn.Linear(latent_size, out_classes)
52        # self.activation = nn.LogSoftmax(dim=1)
53        # self.criterion = nn.NLLLoss()

```

```

54     self.activation = nn.Sigmoid()
55     self.criterion = nn.BCELoss() # nn.MultiLabelSoftMarginLoss()
56
57     def forward(self, X, y=None):
58         X = X.squeeze(1)
59         # print('input shape', X.shape)
60         batch, window_size, channels = X.shape
61         # X = X.transpose(1, 2)
62         X = X.clone()
63         # print('x original shape', x.shape)
64         x = self.convs(x)
65         # print('x shape after convs', x.shape)
66         unpool_dim = x.shape[-1]
67         x = self.pooling(x)
68         print('x shape after pooling', x.shape)
69
70         if not y is None:
71             x = x.squeeze(-1)
72             x = self.fc(x)
73             # print('x shape after fc', x.shape)
74             logits = self.activation(x)
75
76             # loss = self.criterion(logits, y)
77             loss = torch.sum(torch.square(y - logits)) # Simple own implementation
78             # loss = self.criterion(logits, torch.argmax(y, dim=1))
79             accuracies = []
80             mask = y != 0.0
81             inverse_mask = ~mask
82             zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - logits[inverse_mask])) / torch.sum(
83                 inverse_mask) # zero fit goal
84             class_fit = 1.0 - torch.sum(torch.square(y[mask] - logits[mask])) / torch.sum(mask) # class fit goal
85             accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
86             accuracies.append(class_fit)
87             accuracies.append(zero_fit)
88
89             accuracies.append(
90                 1.0 - torch.sum(torch.square(y - logits)) / (self.n_out_classes * batch)) # Distance between all
91             values
92             # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
93             count non zeros in accuracy?
94             accuracies.append(torch.sum(torch.absolute(y - logits) <= 0.01) /
95                 self.n_out_classes * batch) # correct if probability within 0.01
96             # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
97             return accuracies, loss
98         else:
99             reps = [1] * len(x.shape)
100            reps[-1] = unpool_dim
101            x = x.repeat(reps) # unpool avg layer
102            x = F.conv_transpose1d(x)
103            # print('x shape after unpool', x.shape)
104            x = self.t_convs(x)
105            # print('x shape after t_convs', x.shape)
106            loss = torch.sum(torch.square(X - x)) # SE
107            accuracies = []
108            accuracies.append(torch.tensor(0.0).cuda())
109            return accuracies, loss

```

B.4.1.20 deprecated -> architectures_baseline -> baseline_losses.py (code)

```

1 from torch.nn import functional as F
2 import torch
3
4
5 def MSE_loss(pred, y):
6     return F.mse_loss(pred, y)
7
8
9 def multi_loss(pred, y, loss_fns):
10    return sum([lfn(pred, y) for lfn in loss_fns]) / len(loss_fns)

```

B.4.2 deprecated -> cardio_model_v4.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class CPC(nn.Module):
6     def __init__(self, encoder_model, autoregressive_model, predictor_model, timesteps_in, timesteps_out,
7                  timesteps_ignore=0):
8         super(CPC, self).__init__()
9         self.timesteps_in = timesteps_in
10        self.timesteps_out = timesteps_out

```

```

11     self.encoder = encoder_model
12     self.autoregressive = autoregressive_model
13     self.predictor = predictor_model
14     self.softmax = nn.Softmax(dim=0)
15     self.lsoftmax = torch.nn.LogSoftmax(dim=0)
16
17     def forward(self, x_windows, n_timesteps_in, n_timesteps_out, hidden, verbose=False):
18         if verbose: print('x_windows has shape:', x_windows.shape) # shape = batch, windows, channels, window_size
19         x_windows = x_windows.permute(1, 0, 2, 3) # reshaping into windows, batch, channels, windowsize
20         n_windows, n_batches, _, _ = x_windows.shape
21         if n_windows != n_timesteps_in + n_timesteps_out:
22             print("timesteps in and out not matching total windows")
23             if verbose: print('x_windows has shape:', x_windows.shape)
24             latent_list = []
25             for x in x_windows[0:n_timesteps_in]:
26                 if verbose: print(x.shape)
27                 latent_list.append(self.encoder(x))
28             latents = torch.stack(latent_list)
29             if verbose: print('latents have shape:', latents.shape) # shape = timesteps,
30             context, hidden = self.autoregressive(latents, hidden)
31             context = context[-1, :, :] # We only need the latest state. Shape: batch, context_outputsize
32             # Calculate the loss
33             loss = 0.0
34             correct = 0
35             for k in range(0, n_timesteps_out): # Do this for each timestep
36                 latent_k = self.encoder(x_windows[-n_timesteps_out + k])
37                 pred_k = self.predictor(context, k) # Shape (Batches, latents)
38                 inner_f = torch.mm(latent_k, pred_k.transpose(0, 1))
39                 correct += torch.sum(torch.eq(torch.argmax(self.softmax(inner_f)), torch.arange(0,
40                                         n_batches).cuda())))
41             many_correct? Also I don't understand this completely
42             log_softmax_k = self.lsoftmax(torch.diag(inner_f))
43             loss += torch.sum(log_softmax_k)
44             loss /= (
45                 n_timesteps_out * n_batches * -1.0) # Unterschiedlich gewichten auch moeglich. Oder mehr als nur
46             n_batches-1 negative samples
47             # loss /= (n_timesteps_out * 1.0) #Man könnte auch jeden Timestep unterschiedlich gewichten. Nahe = wichtiger
48             # entfernt
49             accuracy = correct / (n_batches * n_timesteps_out)
50             return accuracy, loss, hidden
51
52
53     class Encoder(nn.Module):
54         def __init__(self, channels, latent_size):
55             super(Encoder, self).__init__()
56             filters = [10, 8, 4, 4, 4] # See https://arxiv.org/pdf/1807.03748.pdf#Audio
57             strides = [5, 4, 2, 2, 2]
58             n_channels = [channels] + [latent_size] * len(filters)
59             self.convolutions = nn.Sequential(
60                 *[e for t in [
61                     (nn.Conv1d(in_channels=n_channels[i], out_channels=n_channels[i + 1], kernel_size=filters[i],
62                               stride=strides[i], padding=0),
63                               nn.ReLU())
64                 ) for i in range(len(filters))] for e in t]
65             )
66
67             self.avg_pool = nn.AdaptiveAvgPool1d(1)
68
69             def _weights_init(m):
70                 if isinstance(m, nn.Conv1d):
71                     nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
72
73             self.apply(_weights_init)
74
75         def forward(self, x):
76             # Input has shape (batches, channels, window_size)
77             # print('Encoder input shape:', x.shape)
78             x = self.convolutions(x)
79             x = self.avg_pool(x)
80             # Output has shape (batches, latents, 1)
81             # Maybe squeeze?
82             x = x.squeeze(2)
83             # print('Encoder output shape:', x.shape)
84
85             return x
86
87
88     class AutoRegressor(nn.Module):
89         def __init__(self, n_latents, hidden_size):
90             super(AutoRegressor, self).__init__()
91             self.n_latents = n_latents
92             self.hidden_size = hidden_size
93             self.gru = nn.GRU(input_size=self.n_latents, hidden_size=self.hidden_size, num_layers=1)
94
95             def _weights_init(m):
96                 for layer_p in self.gru._all_weights:
97                     for p in layer_p:

```

```

98         if 'weight' in p:
99             nn.init.kaiming_normal_(self.gru.__getattr__(p), mode='fan_out', nonlinearity='relu')
100
101     self.apply(_weights_init)
102
103 def init_hidden(self, batch_size, use_gpu=True):
104     if use_gpu:
105         return torch.zeros(1, batch_size, self.hidden_size).double().cuda()
106     else:
107         return torch.zeros(1, batch_size, self.hidden_size)
108
109 def forward(self, x, hidden):
110     # Input is (seq, batch, latents) maybe (1, ..., ...)
111     # print('regressor input shape:', x.shape, hidden.shape)
112     x, hidden = self.gru(x, hidden)
113     # print('regressor output shape:', x.shape, hidden.shape)
114     return x, hidden
115
116
117 class Predictor(nn.Module):
118     def __init__(self, encoding_size, code_size, timesteps):
119         super().__init__()
120         self.code_size = code_size
121         self.linears = nn.ModuleList(
122             [nn.Linear(encoding_size, code_size)] * timesteps
123         )
124
125     def _weights_init(m):
126         if isinstance(m, nn.Linear):
127             nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
128
129     self.apply(_weights_init)
130
131     def forward(self, x, timestep):
132         # print('Predictor input shape:', x.shape)
133         prediction = self.linears[timestep](x)
134         # print('Predictor output shape:', prediction.shape)
135         return prediction

```

B.4.3 deprecated -> cardio_model_v5.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class CPC(nn.Module):
6     def __init__(self, encoder_model, autoregressive_model, predictor_model, timesteps_in, timesteps_out,
7                  timesteps_ignore=0):
8         super(CPC, self).__init__()
9         self.timesteps_in = timesteps_in
10        self.timesteps_out = timesteps_out
11        self.encoder = encoder_model
12        self.autoregressive = autoregressive_model
13        self.predictor = predictor_model
14        # self.softmax = nn.Softmax(dim=1)
15        self.lsoftmax = torch.nn.LogSoftmax(dim=1)
16
17    def forward(self, x_windows, n_timesteps_in, n_timesteps_out, hidden, verbose=False):
18        if verbose: print('x_windows has shape:', x_windows.shape) # shape = batch, windows, channels, window_size
19        x_windows = x_windows.permute(1, 0, 2, 3) # reshaping into windows, batch, channels, windowsize
20        n_windows, n_batches, _, _ = x_windows.shape
21        if n_windows != n_timesteps_in + n_timesteps_out:
22            print("timesteps_in and out not matching total windows")
23        if verbose: print('x_windows has shape:', x_windows.shape)
24        latent_list = []
25        for x in x_windows[0:n_timesteps_in]:
26            if verbose: print(x.shape)
27            latent_list.append(self.encoder(x))
28        latents = torch.stack(latent_list)
29        if verbose: print('latents have shape:', latents.shape) # shape = timesteps,
30        context, hidden = self.autoregressive(latents, hidden)
31        context = context[-1, :, :] # We only need the latest state. Shape: batch, context_outputszie
32        # Calculate the loss
33        loss = 0.0
34        correct = 0
35        for k in range(0, n_timesteps_out): # Do this for each timestep
36            latent_k = self.encoder(x_windows[-n_timesteps_out + k]) # batches, latents
37            pred_k = self.predictor(context, k) # Shape (Batches, latents)
38            softmax = self.lsoftmax(torch.mm(latent_k, pred_k.T)) # output: (Batches,)
39            correct += torch.sum(torch.argmax(softmax) == torch.arange(n_batches).cuda())
40            loss += torch.sum(torch.diag(softmax))
41        loss /= n_batches * -1.0
42        # loss /= (n_timesteps_out * n_batches * -1.0) #Unterschiedlich gewichten auch moeglich. Oder mehr als nur
43        n_batches-1 negative samples
44        # loss /= (n_timesteps_out * 1.0) #Man konnte auch jeden Timestep unterschiedlich gewichten. Nahe = wichtiger
45        als entfernt

```

```

44     accuracy = correct / (n_batches * n_timesteps_out)
45     return accuracy, loss, hidden
46
47     def freeze_layers(self):
48         for param in self.parameters():
49             param.requires_grad = False
50
51     def unfreeze_layers(self):
52         for param in self.parameters():
53             param.requires_grad = True
54
55     def init_hidden(self, batch_size, use_gpu=True):
56         return self.autoregressive.init_hidden(batch_size, use_gpu)
57
58
59 class Encoder(nn.Module):
60     def __init__(self, channels, latent_size):
61         super(Encoder, self).__init__()
62         filters = [10, 8, 4, 4, 4] # See https://arxiv.org/pdf/1807.03748.pdf#Audio
63         strides = [5, 4, 2, 2, 2]
64         n_channels = [channels] + [latent_size] * len(filters)
65         self.convolutions = nn.Sequential(
66             *[e for t in [
67                 (nn.Conv1d(in_channels=n_channels[i], out_channels=n_channels[i + 1], kernel_size=filters[i],
68                           stride=strides[i], padding=0),
69                           nn.ReLU())
70                         ) for i in range(len(filters))] for e in t]
71         )
72
73         self.avg_pool = nn.AdaptiveAvgPool1d(1)
74
75     def _weights_init(m):
76         if isinstance(m, nn.Conv1d):
77             nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
78
79     self.apply(_weights_init)
80
81     def forward(self, x):
82         # Input has shape (batches, channels, window_size)
83         # print('Encoder input shape:', x.shape)
84         x = self.convolutions(x)
85         x = self.avg_pool(x)
86         # Output has shape (batches, latents, 1)
87         # Maybe squeeze??
88         x = x.permute(2, 0, 1).squeeze()
89         # print('Encoder output shape:', x.shape)
90
91         return x
92
93
94 class AutoRegressor(nn.Module):
95     def __init__(self, n_latents, hidden_size):
96         super(AutoRegressor, self).__init__()
97         self.n_latents = n_latents
98         self.hidden_size = hidden_size
99         self.gru = nn.GRU(input_size=self.n_latents, hidden_size=self.hidden_size, num_layers=1)
100
101    def _weights_init(m):
102        for layer_p in self.gru._all_weights:
103            for p in layer_p:
104                if 'weight' in p:
105                    nn.init.kaiming_normal_(self.gru.__getattr__(p), mode='fan_out', nonlinearity='relu')
106
107    self.apply(_weights_init)
108
109    def init_hidden(self, batch_size, use_gpu=True):
110        if use_gpu:
111            return torch.zeros(1, batch_size, self.hidden_size).double().cuda()
112        else:
113            return torch.zeros(1, batch_size, self.hidden_size)
114
115    def forward(self, x, hidden):
116        # Input is (seq, batch, latents) maybe (1, ..., ...)
117        # print('regressor input shape:', x.shape, hidden.shape)
118        x, hidden = self.gru(x, hidden)
119        # print('regressor output shape:', x.shape, hidden.shape)
120        return x, hidden
121
122
123 class Predictor(nn.Module):
124     def __init__(self, encoding_size, code_size, timesteps):
125         super().__init__()
126         self.code_size = code_size
127         self.linears = nn.ModuleList(
128             [nn.Linear(encoding_size, code_size)] * timesteps
129         )
130
131     def _weights_init(m):
132         if isinstance(m, nn.Linear):
133             nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')

```

```

134     self.apply(_weights_init)
135
136     def forward(self, x, timestep):
137         # print('Predictor input shape:', x.shape)
138         prediction = self.linears[timestep](x)
139         # print('Predictor output shape:', prediction.shape)
140
141         return prediction

```

B.4.4 deprecated -> cardio_model_v6.py (code)

```

1 import torch
2 from torch import nn
3
4
5 class CPC(nn.Module):
6     def __init__(self, encoder_model, autoregressive_model, predictor_model, timesteps_in, timesteps_out,
7                  timesteps_ignore=0):
8         super(CPC, self).__init__()
9         self.timesteps_in = timesteps_in
10        self.timesteps_out = timesteps_out
11        self.encoder = encoder_model
12        self.autoregressive = autoregressive_model
13        self.predictor = predictor_model
14        # self.softmax = nn.Softmax(dim=1)
15        self.lsoftmax = nn.LogSoftmax(dim=1)
16        self.cpc_train_mode = True
17
18    def forward(self, x_windows, n_timesteps_in: int, n_timesteps_out: int, hidden, verbose=False):
19        # TODO: make plot of data
20        if verbose: print('x_windows has shape:', x_windows.shape) # shape = batch, windows, channels, window_size
21        x_windows = x_windows.permute(1, 0, 2, 3) # reshaping into windows, batch, channels, windowsize
22        n_windows, n_batches, _, _ = x_windows.shape
23        if self.cpc_train_mode and n_windows != n_timesteps_in + n_timesteps_out:
24            print("timesteps in and out not matching total windows")
25        if verbose: print('x_windows has shape:', x_windows.shape)
26        latent_list = []
27        for x in x_windows[0:n_timesteps_in]:
28            if verbose: print(x.shape)
29            latent_list.append(self.encoder(x))
30        latents = torch.stack(latent_list)
31        if verbose: print('latents have shape:', latents.shape) # shape = timesteps,
32        context, hidden = self.autoregressive(latents, hidden)
33        context = context[-1, :, :] # We only need the latest state. Shape: batch, context_outputszie
34        # TODO: decoder of latents
35        # TODO: plot latents
36        if not self.cpc_train_mode:
37            return latents, context, hidden # CPC Mode
38        # Calculate the loss
39        loss = 0.0 # Will become Tensor
40        correct = 0 # will become Tensor
41        for k in range(0, n_timesteps_out): # Do this for each timestep
42            latent_k = self.encoder(x_windows[-n_timesteps_out + k]) # batches, latents
43            pred_k = self.predictor(context, k) # Shape (Batches, latents)
44            softmax = self.lsoftmax(torch.mm(latent_k, pred_k.T)) # output: (Batches, Batches)
45            correct += torch.sum(torch.eq(torch.argmax(softmax, dim=0), torch.arange(n_batches).cuda()))
46            loss += torch.sum(torch.diag(softmax))
47        loss /= n_batches * -1.0
48        # loss /= (n_timesteps_out * n_batches * -1.0) #Unterschiedlich gewichten auch moeglich. Oder mehr als nur
49        n_batches-1 negative samples
50        # loss /= (n_timesteps_out * 1.0) #Man könnte auch jeden Timestep unterschiedlich gewichten. Nahe = wichtiger
51        als entfernt
52        accuracy = correct.true_divide(n_batches * n_timesteps_out)
53        return accuracy, loss, hidden
54
55    def freeze_layers(self):
56        for param in self.parameters():
57            param.requires_grad = False
58
59    def unfreeze_layers(self):
60        for param in self.parameters():
61            param.requires_grad = True
62
63    def set_train_mode(self, mode_bool):
64        self.cpc_train_mode = mode_bool
65
66    def init_hidden(self, batch_size, use_gpu=True):
67        return self.autoregressive.init_hidden(batch_size, use_gpu)
68
69 class Encoder(nn.Module):
70     def __init__(self, channels, latent_size):
71         super(Encoder, self).__init__()
72         filters = [10, 8, 4, 4, 4] # See https://arxiv.org/pdf/1807.03748.pdf#Audio
73         strides = [5, 4, 2, 2, 2]
74         n_channels = [channels] + [latent_size] * len(filters)

```

```

74     # self.batch_norm = nn.BatchNorm1d(n_channels[0]) #not used in paper?
75     self.convolutions = nn.Sequential(
76         *[e for t in [
77             (nn.Conv1d(in_channels=n_channels[i], out_channels=n_channels[i + 1], kernel_size=filters[i],
78                         stride=strides[i], padding=0),
79                         # nn.BatchNorm1d(n_channels[i + 1]),
80                         (nn.ReLU())),
81                     ) for i in range(len(filters))] for e in t]
82     )
83
84     self.avg_pool = nn.AdaptiveAvgPool1d(1)
85
86     def _weights_init(m):
87         if isinstance(m, nn.Conv1d):
88             nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
89
90     self.apply(_weights_init)
91
92     def forward(self, x):
93         # Input has shape (batches, channels, window_size)
94         # print('Encoder input shape:', x.shape)
95         # x = self.batch_norm(x)
96         x = self.convolutions(x)
97         x = self.avg_pool(x)
98         # Output has shape (batches, latents, 1)
99         # Maybe squeeze??
100        x = x.permute(2, 0, 1).squeeze(0) # Only squeeze first (NOT BATCH!) dimension
101        # print('Encoder output shape:', x.shape)
102
103        return x
104
105
106    class AutoRegressor(nn.Module):
107        def __init__(self, n_latents, hidden_size):
108            super(AutoRegressor, self).__init__()
109            self.n_latents = n_latents
110            self.hidden_size = hidden_size
111            self.gru = nn.GRU(input_size=self.n_latents, hidden_size=self.hidden_size, num_layers=1)
112
113            def _weights_init(m):
114                for layer_p in self.gru._all_weights:
115                    for p in layer_p:
116                        if 'weight' in p:
117                            nn.init.kaiming_normal_(self.gru.__getattr__(p), mode='fan_out', nonlinearity='relu')
118
119            self.apply(_weights_init)
120
121        def init_hidden(self, batch_size, use_gpu=True):
122            if use_gpu:
123                return torch.zeros(1, batch_size, self.hidden_size).cuda()
124            else:
125                return torch.zeros(1, batch_size, self.hidden_size)
126
127        def forward(self, x, hidden):
128            # Input is (seq, batch, latents) maybe (13, 8, 128)
129            # print('regressor input shape:', x.shape, hidden.shape)
130            x, hidden = self.gru(x, hidden)
131            # print('regressor output shape:', x.shape, hidden.shape)
132            return x, hidden
133
134
135    class Predictor(nn.Module):
136        def __init__(self, encoding_size, code_size, timesteps):
137            super().__init__()
138            self.code_size = code_size
139            self.linears = nn.ModuleList(
140                [nn.Linear(encoding_size, code_size)] * timesteps
141            )
142
143            def _weights_init(m):
144                if isinstance(m, nn.Linear):
145                    nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
146
147            self.apply(_weights_init)
148
149        def forward(self, x, timestep):
150            # print('Predictor input shape:', x.shape)
151            prediction = self.linears[timestep](x)
152            # print('Predictor output shape:', prediction.shape)
153            return prediction

```

B.4.5 deprecated -> cpc_alpha.py (code)

```

1 import torch
2 from torch import nn
3 import numpy as np

```

```

4 import torch.nn.functional as F
5
6 from util.utility.sparse_print import SparsePrint
7
8
9 class CPC(nn.Module):
10
11     def __init__(self, encoder, autoregressive, predictor, timesteps_in, timesteps_out, latent_size, timesteps_ignore
12 =0,
13         normalize_latents=False, verbose=False, sampling_mode='same'):
14         super().__init__()
15         self.encoder = encoder
16         self.autoregressive = autoregressive
17         self.predictor = predictor
18         self.lsoftmax = nn.LogSoftmax(dim=-1)
19         self.timesteps_in = timesteps_in
20         self.timesteps_out = timesteps_out
21         self.timesteps_ignore = timesteps_ignore
22         self.latent_size = latent_size
23         self.normalize_latents = normalize_latents
24         self.verbose = verbose
25         self.cpc_train_mode = True
26         if sampling_mode in ['all', 'same', 'random', 'future']:
27             self.sampling_mode = sampling_mode
28         else:
29             self.sampling_mode = 'same'
30         self.sprint = SparsePrint()
31
32     def forward(self, X, y=None, hidden=None):
33         if self.verbose: print('input', X.shape)
34         if len(X.shape) == 4: # uses cpc or challenge dataset
35             batch, windows, channels, length = X.shape
36             X = X.transpose(1, 2).reshape((batch, channels, -1))
37         elif len(X.shape) == 3: # uses basline dataset, shape: (batch, length, channels)
38             X = X.transpose(1, 2)
39         batch, channels, length = X.shape # assume baseline dataset shape
40         encoded_x = self.encoder(X) # encode whole data: shape: (batch, latent_size, length->out_length)
41         encoded_x = encoded_x.permute(2, 0, 1) # shape outlength, batch, latent_size
42         # if self.normalize_latents:
43         #     encoded_x = encoded_x / torch.norm(encoded_x, p=2, dim=-1, keepdim=True)
44         if self.verbose: print('encoder_x', encoded_x.shape)
45         encoded_x_steps, _, _ = encoded_x.shape
46         if hidden is None:
47             hidden = self.autoregressive.init_hidden(batch_size=batch)
48         context, hidden = self.autoregressive(encoded_x, hidden)
49         if self.verbose: print('context', context.shape)
50
51         if not self.cpc_train_mode:
52             return encoded_x, context[-1, :, :], hidden
53         # offset = np.random.choice(encoded_x_steps - self.timesteps_out - self.timesteps_ignore, 10, replace=False)
54         # Draw 10 randoms
55
56         loss = torch.tensor([0.0]).cuda()
57         correct = torch.tensor([0.0]).cuda()
58         if encoded_x_steps - self.timesteps_out - self.timesteps_ignore - self.timesteps_in < 1:
59             print(
60                 'Not enough encoded values for CPC training. Lower timesteps_in/timesteps_out/timesteps_ignore or
61                 change window_size + encoder')
62             print(
63                 f'Encoded {encoded_x_steps} steps but need at least {(self.timesteps_out + self.timesteps_ignore +
64                 self.timesteps_in)}')
65             return loss, correct, hidden
66         if self.sampling_mode == 'same':
67             for k in range(self.timesteps_out):
68                 pred_latent = self.predictor(
69                     context[self.timesteps_in:-self.timesteps_out + self.timesteps_ignore + 1], :, :, k)
70                 encoded_latent = encoded_x[self.timesteps_in + k + 1:-self.timesteps_out + self.timesteps_ignore] + k
71
72                 # shape is batch, latent_size
73                 if self.verbose: print(pred_latent.shape, encoded_latent.shape)
74                 if self.normalize_latents:
75                     pred_len = torch.clamp(torch.norm(pred_latent, p=2, dim=-1, keepdim=True), min=1e-10)
76                     enc_len = torch.clamp(torch.norm(encoded_latent, p=2, dim=-1, keepdim=True), min=1e-10)
77                     pred_latent = pred_latent / pred_len
78                     encoded_latent = encoded_latent / enc_len
79                 for step in range(pred_latent.shape[0]): # TODO: can this be broadcasted?
80                     softmax = self.lsoftmax(torch.mm(encoded_latent[step], pred_latent[step].T)) # output: (Batches,
81                     Batches)
82                     if self.verbose: print('softmax shape', softmax.shape)
83                     correct += torch.sum(torch.argmax(softmax, dim=0) == torch.arange(batch).cuda()) # Since
84                     loss += torch.sum(torch.diag(softmax))
85
86             loss /= (batch * self.timesteps_out * pred_latent.shape[0]) * -1.0
87             accuracy = correct.true_divide(batch * self.timesteps_out * pred_latent.shape[0])
88             return accuracy, loss, hidden
89
90         # if self.sampling_mode == 'all':
91         #     encoded_latent = encoded_x.transpose(0, 1) # output shape: Batch, latents, latent_size
92         #     # TODO: maybe also use all timesteps instead of random

```

```

88     # t = np.random.randint(low=self.timesteps_in, high=encoded_x_steps-self.timesteps_out-self.
89     timesteps_ignore) #Select current timestep at random
90     # t = encoded_x_steps - self.timesteps_out - self.timesteps_ignore
91     # if self.normalize_latents:
92     #     enc_len = torch.clamp(torch.norm(encoded_latent, p=2, dim=-1, keepdim=True), min=1e-6)
93     #     encoded_latent = encoded_latent / enc_len
94     # for k in range(self.timesteps_ignore, self.timesteps_ignore + self.timesteps_out):
95     #     pred_latent = self.predictor(context[t - 1, :, :], k).squeeze(0)
96     #     # shape is batch, n_latents, latent_size
97     #     if self.normalize_latents:
98     #         pred_len = torch.clamp(torch.norm(pred_latent, p=2, dim=-1, keepdim=True), min=1e-6)
99     #         pred_latent = pred_latent / pred_len
100    #     prod = torch.matmul(encoded_latent, pred_latent.T).reshape(batch, encoded_x_steps*batch)
101    #     prod_sum = torch.sum(prod, dim=-1, keepdim=True)
102    #     softmax = torch.log(torch.div(prod, prod_sum))
103    #
104    #     torch.matmul(encoded_latent, pred_latent.T).shape == B x latents x B
105    softmax = self.lsoftmax(torch.matmul(encoded_latent, pred_latent.T).reshape(batch,
106                                         encoded_x_steps * batch)
107    #
108    # reshape for argmax to B, latents*B
109    argmaxs = torch.argmax(softmax, dim=1)
110    correct += torch.sum(
111        argmaxs == torch.arange(batch).cuda() + t * batch) # Correct index will be at offset t
112    loss += torch.sum(
113        torch.diag(softmax.reshape(batch, encoded_x_steps, batch)[:, t + self.timesteps_ignore + k, :]))
114    #
115    loss /= (batch * self.timesteps_out) * -1.0 # * pred_latent.shape[0]
116    accuracy = correct.true_divide(batch * self.timesteps_out) # * pred_latent.shape[0]
117    #print(loss)
118    # return accuracy, loss, hidden
119
120 if self.sampling_mode == 'all':
121     t = np.random.randint(self.timesteps_in, encoded_x_steps-self.timesteps_out-self.timesteps_ignore)
122     current_context = context[t-1, :, :]
123     print(encoded_x_steps)
124     enc_resh = encoded_x.reshape(encoded_x_steps*batch, -1)
125
126     for k in range(self.timesteps_out):
127         pred_latent = self.predictor(current_context, k)
128         if self.verbose: print("pred latent shape", pred_latent.shape)
129
130         sim = enc_resh@pred_latent.T # sim[i, j] = scalar prod between li and pred j
131         soft = F.log_softmax(sim, dim=0) #Where is the highest similarity? (max in each i) == i
132         soft_resh = soft.reshape(encoded_x_steps, batch, -1)
133         # soft2 = F.softmax(sim, dim=0)
134         # self.print.print(soft, 10000)
135         loss += torch.diag(soft_resh[t+k]).sum() #higher = better
136         correct += (soft.max(dim=0).indices == torch.arange(batch, device=soft.device)+batch*(t+k)).sum()
137
138     loss = -loss/(self.timesteps_out*batch)
139     accuracy = correct.true_divide(batch * self.timesteps_out) # * pred_latent.shape[0]
140     return accuracy, loss, hidden
141
142 if self.sampling_mode == 'random':
143     raise NotImplementedError
144
145 if self.sampling_mode == 'future':
146     # TODO: maybe also use all timesteps instead of random
147     t = np.random.randint(low=self.timesteps_in,
148                           high=encoded_x_steps - self.timesteps_out - self.timesteps_ignore) # Select current
149     timestep at random
150
151     encoded_latent = encoded_x[t:].transpose(0, 1) # output shape: Batch, latents*, latent_size
152     encoded_x_steps = encoded_latent.shape[1]
153     if self.normalize_latents:
154         encoded_latent = batch * encoded_latent / torch.norm(encoded_latent, p=2, dim=-1, keepdim=True)
155     for k in range(self.timesteps_ignore, self.timesteps_out):
156         pred_latent = self.predictor(context[t, :, :], k).squeeze(0)
157         # shape is batch, n_latents, latent_size
158         if self.normalize_latents:
159             pred_latent = batch * pred_latent / torch.norm(pred_latent, p=2, dim=-1, keepdim=True)
160             # torch.matmul(encoded_latent, pred_latent.T).shape == B x latents x B
161             softmax = self.lsoftmax(torch.matmul(encoded_latent, pred_latent.T).reshape(batch,
162                                         encoded_x_steps * batch))
163
164             # reshape for argmax to B, latents*B
165             argmaxs = torch.argmax(softmax, dim=1)
166             correct += torch.sum(
167                 argmaxs == torch.arange(batch).cuda()) # TODO: Correct index will be at offset t
168             loss += torch.sum(torch.diag(softmax.reshape(batch, encoded_x_steps, batch)[:, k, :])) # +t is gone
169
170             loss /= (batch * self.timesteps_out) * -1.0 # * pred_latent.shape[0]
171             accuracy = correct.true_divide(batch * self.timesteps_out) # * pred_latent.shape[0]
172
173     return accuracy, loss, hidden
174
175 def freeze_layers(self):
176     for param in self.parameters():
177         param.requires_grad = False
178
179 def unfreeze_layers(self):

```

```
174     for param in self.parameters():
175         param.requires_grad = True
```

B.4.6 deprecated -> cpc_decoder.py (code)

```
1 from PIL import Image
2 import matplotlib.pyplot as plt
3 import torch
4 from PIL import Image
5 from torch import nn
6
7
8 class CPCDecoder(nn.Module):
9
10    def __init__(self, cpc_model_trained):
11        super().__init__()
12        self.cpc = cpc_model_trained
13        self.cpc.set_train_mode(False) # Makes the model not calculate loss/accuracy and not predict future latents
14        self.cpc.freeze_layers()
15        self.cpc.cuda()
16
17    def forward(self, X, cpc_latents, cpc_contexts, cpc_hidden, y=None, finished=False):
18        pred = None
19        if not finished:
20            cpc_latent, cpc_context, cpc_hidden = self.cpc(X, self.timesteps_in, -1, cpc_hidden)
21            if self.show_latents:
22                if self.counter % 10 == 0 and self.counter > 1000:
23                    copy = X.clone().detach().cpu().numpy()
24                    for i in range(copy.shape[0]):
25                        fig = plt.figure()
26                        for c in range(copy.shape[2]):
27                            plt.plot(copy[i, :, c, :].flatten())
28                            plt.savefig("images/%d-%d-batch%d-signal.png" % (self.image_counter, self.counter, i))
29                        plt.close(fig)
30                    copy = cpc_latent.clone().detach().cpu().numpy()
31                    for i in range(copy.shape[1]):
32                        im = Image.fromarray(copy[:, i, :].T, 'L')
33                        im.save("images/%d-%d-batch%d-latents.png" % (self.image_counter, self.counter, i))
34                    self.image_counter += 1
35            return cpc_latent, cpc_context, cpc_hidden
36        self.image_counter = 0
37        self.counter += 1
38
39        if self.use_context and self.use_latents:
40            conc = torch.cat([cpc_latents, cpc_contexts], dim=0) # TODO: multi input nets... or concat?
41            pred = self.linear(conc)
42
43        elif self.use_context: # use last context for now
44            stacked = torch.cat(cpc_latents, dim=0)
45            context, hidden = self.cpc.autoregressive(stacked, None)
46            pred = self.linear(context[-1, :, :])
47
48        else: # use only latents, summarize and oredict with linear
49            # whole_context, hidden = self.gru(cpc_latents, hidden)
50            # pred = self.gru_linear(whole_context[-1, :, :])
51            stacked = torch.cat(cpc_latents, dim=0) # outshape = (timesteps*m, batch, latent_size)
52            # print('stckd down', stacked.shape)
53            summarized_X, hidden = self.gru(stacked)
54            # print(summarized_X.shape)
55            pred = self.linear(summarized_X[-1, :, :])
56
57        pred = self.log_softmax(pred)
58        # print('pred shape', pred.shape)
59        if not y is None: # Training mode return loss instead of prediction
59            # print('y shape', y.shape)
60            y = torch.argmax(y, 1)
61            loss = self.criterion(pred, y)
62            # print(torch.argmax(pred, dim=0), y)
63            accuracy = torch.sum(torch.eq(torch.argmax(pred, dim=1), y)) / pred.shape[0]
64
65            return accuracy, loss, cpc_hidden, (torch.argmax(pred, dim=1), y)
66        else:
67            return pred, cpc_hidden
68
69    def init_hidden(self, batch_size, use_gpu):
70        return self.cpc.init_hidden(batch_size, use_gpu)
```

B.4.7 deprecated -> cpc_utils.py (code)

```
1 import numpy as np
2 import torch
3 import torch.nn as nn
```

```

4 import torch.nn.functional as F
5
6
7 # This class is not used right now
8
9
10 def info_NCE_loss(latents: torch.Tensor, context: torch.Tensor, future_latents: torch.Tensor, predictions: torch.
11     Tensor,
12     target_dim=64, emb_scale=0):
13     loss = 0.0
14     latents = latents.permute(1, 0, 2)
15     timesteps_in, batch_dim, n_latents = latents.shape
16     timesteps_out, batch_dim, n_latents = predictions.shape
17     total_elements = batch_dim * n_latents
18     # print('Loss calculation.')
19     # print('latents', latents.shape)
20     # print('context', context.shape)
21     # print('predictions', predictions.shape)
22     # print('future_latents:', future_latents.shape)
23     for i in range(timesteps_out):
24         preds_i = predictions[i]
25         logits = torch.mm(future_latents[i], torch.transpose(preds_i, 0, 1))
26         labels = torch.arange(0, batch_dim).cuda()
27         temp_loss = torch.sum(- labels * F.log_softmax(logits, -1),
28                               -1) # From https://gist.github.com/tejaskhot/cf3d087ce4708c422e68b3b747494b9f
29         mean_loss = temp_loss.mean()
30         loss += mean_loss
31     return loss
32
33
34 def info_NCE_loss_brian(latents: torch.Tensor, context, target_dim=64, emb_scale=0, steps_to_ignore=0,
35     steps_to_predict=3):
36     """
37     Calculates the infoNCE loss for CPC according to 'DATA-EFFICIENT IMAGE RECOGNITION
38     WITH CONTRASTIVE PREDICTIVE CODING' A.2
39     :param latents: latent variables with shape [B, H, W, D]
40     :param target_dim:
41     :param emb_scale:
42     :param steps_to_ignore:
43     :param steps_to_predict:
44     :return: The calculated loss
45     """
46     loss = 0.0
47     # latents = latents.permute(1, 0, 2)
48     targets = nn.Conv1d(in_channels=latents.shape[1], out_channels=target_dim, kernel_size=1).cuda()(latents)
49     batch_dim, col_dim, row_dim, _ = targets.shape
50     targets = targets.view([-1, target_dim]) # reshape
51     for i in range(steps_to_ignore, steps_to_predict):
52         col_dim_i = col_dim - i - 1
53         total_elements = batch_dim * col_dim_i * row_dim
54         preds_i = nn.Conv1d(out_channels=target_dim, kernel_size=1)(context)
55         preds_i = preds_i[:, :, -(i + 1), :, :] * emb_scale
56         preds_i = preds_i.view([-1, target_dim])
57         logits = torch.matmul(preds_i, torch.t(targets))
58         b = np.arange(total_elements) / (col_dim_i * row_dim)
59         col = np.arange(total_elements) % (col_dim_i * row_dim)
60         labels = b * col_dim * row_dim + (i + 1) * row_dim + col
61         temp_loss = torch.sum(- labels * F.log_softmax(logits, -1),
62                               -1) # From https://gist.github.com/tejaskhot/cf3d087ce4708c422e68b3b747494b9f
63         mean_loss = temp_loss.mean()
64         loss += mean_loss
65     return loss
66
67
68 def pixelCNN(latents):
69     from keras.layers import Conv2D, ReLU
70     from tensorflow_core.python import Pad
71     import tensorflow as tf
72     # latents: [B, H, W, D]
73     cres = latents
74     cres_dim = cres.shape[-1]
75     for _ in range(5):
76         c = tf.nn.conv2d(filters=256, kernel_size=(1, 1))(cres)
77         c = tf.nn.relu(c)
78         c = tf.nn.conv2d(filters=256, kernel_size=(1, 3))(c)
79         c = Pad(c, [[0, 0], [1, 0], [0, 0], [0, 0]])
80         c = Conv2D(filters=256, kernel_size=(2, 1),
81                    type='VALID')(c)
82         c = ReLU(c)
83         c = Conv2D(filters=cres_dim, kernel_size=(1, 1))(c)
84         cres = cres + c
85         cres = ReLU(cres)
86     return cres
87
88
89 def CPC(latents, target_dim=64, emb_scale=0.1, steps_to_ignore=2, steps_to_predict=3):
90     from tensorflow_core import reshape, matmul
91
92     from keras.layers import Conv2D

```

```

93     # latents: [B, H, W, D]
94     loss = 0.0
95     context = pixelCNN(latents)
96     targets = Conv2D(output_channels=target_dim,
97                       kernel_shape=(1, 1))(latents)
98     batch_dim, col_dim, rows = targets.shape[: - 1]
99     targets = reshape(targets, [- 1, target_dim])
100    for i in range(steps_to_ignore, steps_to_predict):
101        col_dim_i = col_dim - i - 1
102        total_elements = batch_dim * col_dim_i * rows
103        preds_i = Conv2D(output_channels=target_dim,
104                           kernel_shape=(1, 1))(context)
105        preds_i = preds_i[:, : - (i + 1), :, :] * emb_scale
106        preds_i = reshape(preds_i, [- 1, target_dim])
107        logits = matmul(preds_i, targets, transp_b=True)
108        b = range(total_elements) / (col_dim_i * rows)
109        col = range(total_elements) % (col_dim_i * rows)
110        labels = b * col_dim * rows + (i + 1) * rows + col
111        print(labels.shape)
112    return loss
113
114 # if __name__ == '__main__':
115 #     B, H, W, D = (12, 4, 5, 100)
116 #     tf.compat.v1.enable_eager_execution()
117 #     latents = tensorflow.random.normal((B, H, W, D))
118 #     CPC(latents)

```

B.4.8 deprecated -> data_storage.py (code)

```

1 import os
2
3 import numpy as np
4 import torch
5
6
7 class DataStorage():
8     def __init__(self, save_path):
9         self.data_dict = {}
10        self.save_path = save_path
11
12    def append(self, key: str, data: object):
13        if key in self.data_dict:
14            v = self.data_dict[key]
15            if isinstance(data, torch.Tensor):
16                self.data_dict[key] = torch.cat([v, data], dim=0)
17            else:
18                self.data_dict[key] = data
19
20    def add(self, key: str, data: object):
21        if key in self.data_dict:
22            v = self.data_dict[key]
23            if isinstance(data, torch.Tensor):
24                self.data_dict[key] += data
25            else:
26                self.data_dict[key] += data
27        else:
28            self.data_dict[key] = data
29
30    def save_to_file(self):
31        for k, v in self.data_dict.items():
32            p = os.path.join(self.save_path, k) + '.pt'
33            if isinstance(v, torch.Tensor):
34                torch.save(v, p)
35            elif isinstance(v, np.ndarray):
36                np.save(p, v)
37
38
39    def load_np_array(p):
40        return np.load(p)
41
42
43    def load_torch_tensor(p):
44        return torch.load(p)

```

B.4.9 deprecated -> downstream_model.py (code)

```

1 from PIL import Image
2 import matplotlib.pyplot as plt
3 import torch
4 from PIL import Image
5 from torch import nn
6

```

```

7
8 class DownstreamLinearNet(nn.Module):
9
10    def __init__(self, cpc_model_trained, timesteps_in, context_size, latent_size, out_classes, use_context=False,
11                 use_latents=True):
12        super().__init__()
13        self.cpc = cpc_model_trained
14        self.cpc.set_train_mode(False) # Makes the model not calculate loss/accuracy and not predict future latents
15        # self.cpc.freeze_layers()
16        self.cpc.cuda()
17        self.use_context = use_context
18        self.use_latents = use_latents
19        self.hidden_size = 512
20        if self.use_context and self.use_latents:
21            self.linear = nn.Linear(in_features=latent_size + context_size,
22                                   out_features=out_classes) # TODO: this is broken
23        elif self.use_context:
24            self.linear = nn.Linear(in_features=context_size, out_features=out_classes)
25        else: # If neither or just latents has been selected take latents
26            # self.linear = nn.Linear(in_features=latent_size*timesteps_in, out_features=out_classes)
27
28            self.gru = nn.GRU(input_size=latent_size, hidden_size=self.hidden_size, num_layers=1)
29            self.linear = nn.Linear(in_features=self.hidden_size, out_features=out_classes)
30            self.log_softmax = nn.LogSoftmax(dim=1)
31            self.criterion = nn.NLLLoss()
32            self.show_latents = False
33            self.counter = 0
34            self.image_counter = 0
35            self.timesteps_in = timesteps_in
36
37    def forward(self, X, cpc_latents, cpc_contexts, cpc_hidden, y=None, finished=False):
38        pred = None
39        if not finished:
40            cpc_latent, cpc_context, cpc_hidden = self.cpc(X, self.timesteps_in, -1, cpc_hidden)
41            if self.show_latents:
42                if self.counter % 10 == 0 and self.counter > 1000:
43                    copy = X.clone().detach().cpu().numpy()
44                    for i in range(copy.shape[0]):
45                        fig = plt.figure()
46                        for c in range(copy.shape[2]):
47                            plt.plot(copy[i, :, c, :].flatten())
48                        plt.savefig("images/%d-%d-signal.png" % (self.image_counter, self.counter, i))
49                        plt.close(fig)
50                    copy = cpc_latent.clone().detach().cpu().numpy()
51                    for i in range(copy.shape[1]):
52                        im = Image.fromarray(copy[:, i, :].T, 'L')
53                        im.save("images/%d-%d-latents.png" % (self.image_counter, self.counter, i))
54                    self.image_counter += 1
55            return cpc_latent, cpc_context, cpc_hidden
56            self.image_counter = 0
57            self.counter += 1
58
59        if self.use_context and self.use_latents:
60            conc = torch.cat([cpc_latents, cpc_contexts], dim=0) # TODO: multi input nets... or concat?
61            pred = self.linear(conc)
62
63        elif self.use_context: # use last context for now
64            stacked = torch.cat(cpc_latents, dim=0)
65            context, hidden = self.cpc.autoregressive(stacked, None)
66            pred = self.linear(context[-1, :, :])
67
68        else: # use only latents, summarize and predict with linear
69            # whole_context, hidden = self.gru(cpc_latents, hidden)
70            # pred = self.gru_linear(whole_context[-1, :, :])
71            stacked = torch.cat(cpc_latents, dim=0) # outshape = (timesteps*m, batch, latent_size)
72            # print('stacked down', stacked.shape)
73            summarized_X, hidden = self.gru(stacked)
74            # print(summarized_X.shape)
75            pred = self.linear(summarized_X[-1, :, :])
76
77        pred = self.log_softmax(pred)
78        # print('pred shape', pred.shape)
79        if not y is None: # Training mode return loss instead of prediction
80            # print('y shape', y.shape)
81            y = torch.argmax(y, 1)
82            loss = self.criterion(pred, y)
83            # print(torch.argmax(pred, dim=0), y)
84            accuracy = torch.sum(torch.eq(torch.argmax(pred, dim=1), y)) / pred.shape[0]
85
86        return accuracy, loss, cpc_hidden, (torch.argmax(pred, dim=1), y)
87    else:
88        return pred, cpc_hidden
89
90    def init_hidden(self, batch_size, use_gpu):
91        return self.cpc.init_hidden(batch_size, use_gpu)

```

B.4.10 deprecated -> downstream_model_multitarget.py (code)

```

1  from PIL import Image
2  import matplotlib.pyplot as plt
3  import torch
4  from PIL import Image
5  from torch import nn
6
7
8  class DownstreamLinearNet(nn.Module):
9
10     def __init__(self, cpc_model_trained, timesteps_in, context_size, latent_size, out_classes, use_context=False,
11                  use_latents=True):
12         super().__init__()
13         self.cpc = cpc_model_trained
14         self.cpc.set_train_mode(False) # Makes the model not calculate loss/accuracy and not predict future latents
15         # self.cpc.freeze_layers()
16         self.cpc.cuda()
17         self.use_context = use_context
18         self.use_latents = use_latents
19         self.hidden_size = 512
20         if self.use_context and self.use_latents:
21             self.linear = nn.Linear(in_features=latent_size + context_size,
22                                    out_features=out_classes) # TODO: this is broken
23         elif self.use_context:
24             self.linear = nn.Linear(in_features=context_size, out_features=out_classes)
25         else: # If neither or just latents has been selected take latents
26             # self.linear = nn.Linear(in_features=latent_size*timesteps_in, out_features=out_classes)
27
28             self.gru = nn.GRU(input_size=latent_size, hidden_size=self.hidden_size, num_layers=1)
29             self.linear = nn.Linear(in_features=self.hidden_size, out_features=out_classes)
30             self.activation = nn.Sigmoid()
31             self.criterion = nn.BCELoss()
32             self.show_latents = False
33             self.counter = 0
34             self.image_counter = 0
35             self.timesteps_in = timesteps_in
36
37     def forward(self, X, cpc_latents, cpc_contexts, cpc_hidden, y=None, finished=False):
38         pred = None
39         if not finished:
40             cpc_latent, cpc_context, cpc_hidden = self.cpc(X, self.timesteps_in, -1, cpc_hidden)
41             if self.show_latents:
42                 if self.counter % 10 == 0 and self.counter > 1000:
43                     copy = X.clone().detach().cpu().numpy()
44                     for i in range(copy.shape[0]):
45                         fig = plt.figure()
46                         for c in range(copy.shape[2]):
47                             plt.plot(copy[i, :, c, :].flatten())
48                         plt.savefig("images/%d-%d-batch%d_signal.png" % (self.image_counter, self.counter, i))
49                         plt.close(fig)
50                     copy = cpc_latent.clone().detach().cpu().numpy()
51                     for i in range(copy.shape[1]):
52                         im = Image.fromarray(copy[:, i, :].T, 'L')
53                         im.save("images/%d-%d-batch%d_latents.png" % (self.image_counter, self.counter, i))
54                     self.image_counter += 1
55             return cpc_latent, cpc_context, cpc_hidden
56         self.image_counter = 0
57         self.counter += 1
58
59         if self.use_context and self.use_latents:
60             conc = torch.cat([cpc_latents, cpc_contexts], dim=0) # TODO: multi input nets... or concat?
61             pred = self.linear(conc)
62
63         elif self.use_context: # use last context for now
64             stacked = torch.cat(cpc_latents, dim=0)
65             context, hidden = self.cpc.autoregressive(stacked, None)
66             pred = self.linear(context[-1, :, :])
67
68         else: # use only latents, summarize and oredict with linear
69             # whole_context, hidden = self.gru(cpc_latents, hidden)
70             # pred = self.gru_linear(whole_context[-1, :, :])
71             stacked = torch.cat(cpc_latents, dim=0) # outshape = (timesteps*m, batch, latent_size)
72             # print('stacked down', stacked.shape)
73             summarized_X, hidden = self.gru(stacked)
74             # print(summarized_X.shape)
75             pred = self.linear(summarized_X[-1, :, :])
76
77         pred = self.activation(pred)
78         # print('pred shape', pred.shape)
79         if not y is None: # Training mode return loss instead of prediction
80             batch, n_classes = y.shape
81             # loss = self.criterion(pred, y)
82             loss = torch.sum(torch.square(y - pred))
83             accuracies = []
84             mask = y != 0.0
85             inverse_mask = ~mask
86             zero_fit = 1.0 - torch.sum(torch.square(y[inverse_mask] - pred[inverse_mask])) / torch.sum(
87                 inverse_mask) # zero fit goal
88             class_fit = 1.0 - torch.sum(torch.square(y[mask] - pred[mask])) / torch.sum(mask) # class fit goal

```

```

89         accuracies.append(0.5 * class_fit + 0.5 * zero_fit)
90         accuracies.append(class_fit)
91         accuracies.append(zero_fit)
92
93         accuracies.append(
94             1.0 - torch.sum(torch.square(y - pred)) / (n_classes * batch)) # Distance between all values
95         # accuracy = 1.0 - torch.sum(torch.square(y - logits)) / torch.sum((y != 0.0) | (logits != 0.0)) #only
96         count non zeros in accuracy?
97         accuracies.append(torch.sum(torch.absolute(y - pred) <= 0.01) / (
98             n_classes * batch)) # correct if probability within 0.01
99         # accuracy = torch.sum(torch.eq(torch.argmax(logits, dim=1), torch.argmax(y, dim=1))) / batch
100        return accuracies, loss, cpc_hidden, (torch.argmax(pred, dim=1), y)
101    else:
102        return pred, cpc_hidden
103
104    def init_hidden(self, batch_size, use_gpu):
105        return self.cpc.init_hidden(batch_size, use_gpu)

```

B.4.11 deprecated -> `ecg_datasets.py` (code)

```

1 import os
2 from random import shuffle
3
4 import h5py
5 import numpy as np
6 import torch
7
8
9 class ECGDataset(torch.utils.data.IterableDataset):
10     def __init__(self, BASE_DIR, window_size, n_windows, files=None):
11         super(ECGDataset).__init__()
12         self.BASE_DIR = BASE_DIR
13         self.n_windows = n_windows
14         self.window_size = window_size
15         if files:
16             self.files = files
17         else:
18             self.files = self.search_files()
19             self.print_file_attributes()
20
21     def __iter__(self):
22         # multiple workers?
23         file_index = 0
24         while file_index < len(self.files):
25             current_file = self.files[file_index]
26             with h5py.File(current_file, 'r') as f:
27                 index = 0
28                 offset = np.random.randint(0, self.window_size) # Random offset
29                 data = f.get('data') # Make sure this exists in your dataset
30                 if not data is None:
31                     while (index + 1) * self.window_size * self.n_windows + offset <= len(data):
32                         yield np.swapaxes(data[index * self.window_size * self.n_windows + offset: (index +
33                                         1) * self.window_size * self.n_windows + offset,
34                                         0:2].reshape((self.n_windows, self.window_size, -1)), 1, 2)
35                         index += self.n_windows
36             file_index += 1
37
38     def __len__(self):
39         return 1 # Whatever
40
41     def search_files(self):
42         record_files = []
43         file_endings = ('.h5')
44         for root, dirs, files in os.walk(self.BASE_DIR):
45             for file in files:
46                 if file.endswith(file_endings):
47                     record_files.append(os.path.join(root, file))
48         print(len(record_files), 'record files found in ', self.BASE_DIR)
49         return record_files
50
51     def print_file_attributes(self):
52         with h5py.File(self.files[0], 'r') as f: # only look at first file
53             print('Keys contained in dataset:', f.keys())
54             for k in f.keys():
55                 data = f.get(k)
56                 print('Data with key [%s] has shape:' % k, data.shape)
57
58
59 class ECGLabelDataset(torch.utils.data.IterableDataset):
60     def __init__(self, BASE_DIR, window_size, n_windows, files=None):
61         super(ECGLabelDataset).__init__()
62         self.BASE_DIR = BASE_DIR
63         if files:
64             self.files = files
65         else:

```

```

66         self.files = self.search_files()
67         self.window_size = window_size
68         self.n_windows = n_windows
69         self.print_file_attributes()
70
71     def __iter__(self):
72         # multiple workers?
73         file_index = 0
74         while file_index < len(self.files):
75             self.current_file = self.files[file_index]
76             with h5py.File(self.current_file, 'r') as f:
77                 index = 0
78                 offset = np.random.randint(self.window_size) # Use a random offset
79                 data = f.get('data')
80                 labels = f.get('labels') # Uses data and labels group in .h5py file. make sure those exist
81                 while (index + 1) * self.window_size * self.n_windows <= len(data):
82                     # tmp_data = data[index*self.window_size:self.n_windows:(index+1)*self.window_size*self.n_windows]
83                     yield np.swapaxes(data[index * self.window_size * self.n_windows + offset: (
84                                         index + 1)
84                                         * self.window_size * self.n_windows + offset].reshape(
85                                         (self.n_windows, self.window_size, -1)), 1, 2), \
86                                         np.swapaxes(labels[index * self.window_size * self.n_windows + offset: (
87                                         index +
87                                         1) * self.window_size * self.n_windows + offset].reshape(
88                                         (self.n_windows, self.window_size, -1)), 1,
89                                         2) # labels[self.n_windows:(index+1)*self.window_size*self.n_windows] #TODO:
90             Try returning as list
91                 index += 1
92                 file_index += 1
93
94     def search_files(self):
95         record_files = []
96         file_endings = ('.h5')
97         for root, dirs, files in os.walk(self.BASE_DIR):
98             for file in files:
99                 if file.endswith(file_endings):
100                     record_files.append(os.path.join(root, file))
101     print(len(record_files), 'record files found in ', self.BASE_DIR)
102     return record_files
103
104     def print_file_attributes(self):
105         print('Printing attributes for dataset at dir:', self.BASE_DIR)
106         with h5py.File(self.files[0], 'r') as f: # only look at first file
107             print('looking at file', f)
108             print('Keys contained in dataset:', f.keys())
109             for k in f.keys():
110                 data = f.get(k)
111                 print('Data with key [%s] has shape:' % k, data.shape)
112
113             labels = f.get('labels') # Uses data and labels group in .h5py file. make sure those exist
114             print(np.min(labels), np.max(labels))
115
116     class ECGDatasetMultiple(torch.utils.data.IterableDataset):
117         def __init__(self, BASE_DIRECTORIES, window_size, n_windows, files=None, selected_channels=None):
118             super(ECGDatasetMultiple).__init__()
119             self.BASE_DIRECTORIES = BASE_DIRECTORIES
120             self.n_windows = n_windows
121             self.window_size = window_size
122             if files:
123                 self.files = files
124             else:
125                 self.files = self.search_files()
126                 self.print_file_attributes()
127
128         def __iter__(self):
129             # multiple workers?
130             file_index = 0
131             while file_index < len(self.files):
132                 self.current_file = self.files[file_index]
133                 with h5py.File(self.current_file, 'r') as f:
134                     index = 0
135                     offset = np.random.randint(0, self.window_size) # Random offset
136                     data = f.get('data') # Make sure this exists in your dataset
137                     while (index + 1) * self.window_size * self.n_windows + offset <= len(data):
138                         yield np.swapaxes(data[index * self.window_size * self.n_windows + offset: (
139                                         index + 1) *
140                                         self.window_size * self.n_windows + offset,
141                                         0:2].reshape(
142                                         (self.n_windows, self.window_size, -1)), 1,
143                                         2) # TODO: MAKE chosen channel index selectable for each dataset
144                     index += self.n_windows
145                     file_index += 1
146
147         def __len__(self):
148             return 1 # Whatever
149
150         def search_files(self):
151             record_files = []
152             file_endings = ('.h5')
```

```

152     for base_dir in self.BASE_DIRECTORIES:
153         for root, dirs, files in os.walk(base_dir):
154             for file in files:
155                 if file.endswith(file_endings):
156                     record_files.append(os.path.join(root, file))
157     print(len(record_files), 'record files found in ', self.BASE_DIRECTORIES)
158     return record_files
159
160 def print_file_attributes(self):
161     with h5py.File(self.files[0], 'r') as f: # only look at first file
162         print('Keys contained in dataset:', f.keys())
163         for k in f.keys():
164             data = f.get(k)
165             print('Data with key [%s] has shape:' % k, data.shape)
166
167
168 class ECGDatasetBaseline(torch.utils.data.IterableDataset):
169
170     def __init__(self, BASE_DIR, window_size, files=None):
171         super(ECGDataset).__init__()
172         self.BASE_DIR = BASE_DIR
173         self.window_size = window_size
174         if files:
175             self.files = files
176         else:
177             self.files = self.search_files()
178         self.print_file_attributes()
179
180     def __iter__(self):
181         file_index = 0
182         shuffle(self.files)
183         while file_index < len(self.files):
184             self.current_file = self.files[file_index]
185             with h5py.File(self.current_file, 'r') as f:
186                 index = 0
187                 data = f.get('data')
188                 labels = f.get('label')
189                 offset = np.random.randint(len(data) - self.window_size) # Random offset
190                 if not data is None and not labels is None:
191                     yield data[offset:self.window_size + offset, :], labels[:, :]
192             file_index += 1
193
194     def __len__(self):
195         return 1 # Whatever
196
197     def search_files(self):
198         record_files = []
199         file_endings = ('.h5')
200         for root, dirs, files in os.walk(self.BASE_DIR):
201             for file in files:
202                 if file.endswith(file_endings):
203                     record_files.append(os.path.join(root, file))
204         print(len(record_files), 'record files found in ', self.BASE_DIR)
205         return record_files
206
207     def print_file_attributes(self):
208         with h5py.File(self.files[0], 'r') as f: # only look at first file
209             print('Keys contained in dataset:', f.keys())
210             for k in f.keys():
211                 data = f.get(k)
212                 print('Data with key [%s] has shape:' % k, data.shape)

```

B.4.12 deprecated -> main.py (code)

```

1 import argparse
2 import datetime
3 import os
4 from pathlib import Path
5
6 import numpy as np
7 import torch
8 from torch import optim
9 from torch.utils.data import DataLoader, ChainDataset
10
11 from architectures_various import explain_network
12 from util.data import ecg_datasets2
13 import baseline_convencoder
14 import baseline_cnn_v0_3
15 # from cardio_model_small import CPC, Predictor, AutoRegressor, Encoder
16 # from architectures_cpc.cpc_encoder_vresnet import CPC, Predictor, AutoRegressor, Encoder
17 from architectures_cpc import cpc_encoder_v0, cpc_autoregressive_v0, cpc_predictor_v0, cpc_intersect, \
18     cpc_downstream_model_multitarget_v2
19 from deprecated.downstream_model_multitarget import DownstreamLinearNet
20 from deprecated.optimizer import ScheduledOptim
21 from training import cpc_train, cpc_validation, down_validation, baseline_train, baseline_validation, \
22     decoder_validation, decoder_train

```

```

23 from util.utility.full_class_name import fullname
24 from util.data.ptbxl_data import PTBXLData
25 from util.store_models import save_model_state, load_model_state
26 from util.visualize.timeseries_to_image_converter import timeseries_to_image
27
28
29 def main(args):
30     np.random.seed(args.seed)
31
32     # TODO: Put these params in the arguments as well
33     # params
34
35     timesteps_in = args.timesteps_in
36     timesteps_out = args.timesteps_in
37     number_of_latents = args.latent_size
38     channels = args.channels
39     train_batch_size = args.batch_size
40     validation_batch_size = args.batch_size
41     window_length = args.window_length # Total size = window_length*n_windows
42     hidden_size = args.hidden_size
43     n_windows = timesteps_in + timesteps_out
44     starting_epoch = 0
45
46     epochs = args.epochs
47
48     out_path = args.out_path
49     Path(out_path).mkdir(parents=True, exist_ok=True)
50     with open(os.path.join(out_path, 'params.txt'), 'w') as cfg:
51         cfg.write('\n'.join([str(pa) for pa in
52                             [timesteps_in, timesteps_out, number_of_latents, channels, train_batch_size,
53                              validation_batch_size, window_length, n_windows, args]]))
53
54
55     # train_dataset_peters = ecg_datasets2.ECGDataset(
56     #     '/media/julian/Volume/data/ECG/st-petersburg-arrythmia-annotations/resampled/train', window_size=
57     #     window_length,
58     #     n_windows=n_windows, preload_windows=40)
59     # train_dataset_ptb = ecg_datasets2.ECGDatasetBatching('/media/julian/data/data/ECG/ptb-diagnostic-ecg-database
60     -1.0.0/generated/normalized/train', window_size=window_length, n_windows=n_windows, preload_windows=40,
61     batch_size=train_batch_size)
62     # train_dataset_ptbxl = ecg_datasets2.ECGDatasetBatching('/media/julian/data/data/ECG/ptb-xl-a-large-publicly-
63     # available-electrocardiography-dataset-1.0.1/generated/1000/normalized-labels/train', window_size=window_length,
64     n_windows=n_windows, preload_windows=40)
65     # train_challenge_ptbxl = ecg_datasets2.ECGChallengeDatasetBatching('/media/julian/data/data/ECG/ptbxl_challenge',
66     # window_size=236,
67     # n_windows=timesteps_in + timesteps_out)
68     # val_dataset_mit = ecg_datasets2.ECGDataset('/media/julian/Volume/data/ECG/mit-bih-arrhythmia-database-1.0.0/
69     generated/resampled/val', window_size=window_length, n_windows=n_windows)
70     # val_dataset_peters = ecg_datasets2.ECGDataset('/media/julian/Volume/data/ECG/st-petersburg-arrythmia-annotations
71     /resampled/val', window_size=window_length, n_windows=n_windows, preload_windows=40)
72     # val_dataset_ptb = ecg_datasets2.ECGDatasetBatching('/media/julian/data/data/ECG/ptb-diagnostic-ecg-database
73     -1.0.0/generated/normalized/val', window_size=window_length, n_windows=n_windows, preload_windows=40,
74     batch_size=validation_batch_size)
75     # val_dataset_ptbxl = ecg_datasets2.ECGDataset('/media/julian/data/data/ECG/ptb-xl-a-large-publicly-available-
76     # electrocardiography-dataset-1.0.1/generated/1000/normalized-labels/val', window_size=236, n_windows=n_windows,
77     preload_windows=40)
78     # val_challenge_ptbxl = ecg_datasets2.ECGChallengeDatasetBatching('/media/julian/data/data/ECG/china_challenge',
79     # window_size=236,
80     # n_windows=timesteps_in + timesteps_out)
81     # train_baseline_ptbxl = ecg_datasets2.ECGDatasetBaselineMulti(
82     #     '/media/julian/data/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/generated
83     /1000/normalized-labels/train',
84     #     '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
85     generated/1000/normalized-labels/train',
86     window_size=9500)
87
88     val_baseline_ptbxl = ecg_datasets2.ECGDatasetBaselineMulti(
89     '/media/julian/data/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/generated
90     /1000/normalized-labels/val',
91     # '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
92     generated/1000/normalized-labels/val',
93     window_size=9500)
94
95     # enc = cpc_encoder_v2.Encoder(channels=channels, latent_size=args.latent_size) # TODO: automatically fit
96     # this to data

```

```

93     #
94     # enc.cuda()
95     # print(enc)
96     #
97     # auto = cpc_autoregressive_v0.AutoRegressor(n_latents=args.latent_size, hidden_size=hidden_size)
98     # auto.cuda()
99     # print(auto)
100    #
101    # predictor = cpc_predictor_v0.Predictor(hidden_size, args.latent_size, timesteps_out)
102    # predictor.cuda()
103    # print(predictor)
104    #
105    # model = cpc_base.CPC(enc, auto, predictor, args.latent_size, timesteps_in=timesteps_in, timesteps_out=
timesteps_out)
106    model = cpc_intersect.CPC(
107        encoder=cpc_encoder_v0.Encoder(channels=channels, latent_size=args.latent_size),
108        autoregressive=cpc_autoregressive_v0.AutoRegressor(n_latents=args.latent_size,
109                                                              hidden_size=args.hidden_size),
110        predictor=cpc_predictor_v0.Predictor(hidden_size, args.latent_size, timesteps_out),
111        latent_size=args.latent_size, timesteps_in=timesteps_in, timesteps_out=timesteps_out, verbose=False)
112
113    trainset_total = ChainDataset([train_baseline_ptbxl])
114    dataloader = DataLoader(trainset_total, batch_size=args.batch_size, drop_last=True, num_workers=1)
115
116    valset_total = ChainDataset([val_baseline_ptbxl])
117    valloader = DataLoader(valset_total, batch_size=args.batch_size, drop_last=True, num_workers=1)
118    model.cuda()
119    optimizer = ScheduledOptim(
120        optim.Adam(
121            filter(lambda p: p.requires_grad, model.parameters()),
122            betas=(0.9, 0.98), eps=1e-09, weight_decay=1e-4, amsgrad=True),
123        args.warmup_steps)
124
125    if args.saved_model:
126        model, optimizer, starting_epoch = load_model_state(args.saved_model, model, optimizer)
127    else:
128        torch.save(model, os.path.join(out_path, "cpc_model_full.pt"))
129    model.cuda()
130
131    # model_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
132
133    ## Start training
134    best_acc = 0
135    best_loss = np.inf
136    best_epoch = -1
137    train_losses = []
138    val_losses = []
139    train_accuracies = []
140    val_accuracies = []
141    for epoch in range(starting_epoch, epochs + starting_epoch):
142
143        train_acc, train_loss = cpc_train(model, dataloader, timesteps_in, timesteps_out, optimizer, epoch,
144                                         train_batch_size)
145        val_acc, val_loss = cpc_validation(model, valloader, timesteps_in, timesteps_out, validation_batch_size)
146        val_losses.append(val_loss.item())
147        train_losses.append(train_loss.item())
148        train_accuracies.append(train_acc.item())
149        val_accuracies.append(val_acc.item())
150
151        # Save
152        if val_loss < best_loss: # TODO: maybe use accuracy (not sure if accuracy is a good measurement)
153            best_loss = val_loss
154            best_acc = max(val_acc, best_acc)
155            best_epoch = epoch
156            print("saving model")
157            torch.save(model.state_dict(), os.path.join(out_path, "cpc_model_validation.pt"))
158
159        if epoch - best_epoch >= 5:
160            # update learning rate
161            pass
162
163        if epoch % 10 == 0:
164            save_model_state(out_path, epoch, args.train_mode, model, optimizer,
165                             [train_accuracies, val_accuracies],
166                             [train_losses, val_losses])
167
168    save_model_state(out_path, epochs, args.train_mode, model, optimizer,
169                     [train_accuracies, val_accuracies], [train_losses, val_losses])
170
171    # https://pytorch.org/tutorials/beginner/saving_loading_models.html
172    if args.train_mode == 'downstream':
173
174        # train_dataset_ptb = ecg_datasets2.ECGDatasetBatching('/media/julian/Volume/data/ECG/ptb-diagnostic-ecg-
database-1.0.0/generated/normalized/train', window_size=window_length, n_windows=n_windows, channels=slice(0,12)
, preload_windows=40, batch_size=train_batch_size, use_labels=True)
175        # train_dataset_ptbxl = ecg_datasets2.ECGDatasetBatching('/media/julian/Volume/data/ECG/ptb-xl-a-large-
publicly-available-electrocardiography-dataset-1.0.1/generated/1000/normalized-labels/train', window_size=
window_length, n_windows=n_windows, channels=slice(0,12), preload_windows=40, batch_size=train_batch_size,
use_labels=True)
176        # trainset_total = ecg_datasets2.ECGMultipleDatasets([train_dataset_ptbxl])
177        # trainloader = DataLoader(trainset_total, batch_size=train_batch_size, drop_last=True, num_workers=1,
collate_fn=ecg_datasets2.collate_fn)
178        #

```

```

175     # #val_dataset_ptb = ecg_datasets2.ECGDatasetBatching('/media/julian/Volume/data/ECG/ptb-diagnostic-ecg-
176     database-1.0.0/generated/normalized/val',window_size=window_length, n_windows=n_windows, channels=slice(0, 12),
177     preload_windows=40, batch_size=validation_batch_size, use_labels=True)
178     # val_dataset_ptbx1 = ecg_datasets2.ECGDatasetBatching('/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-
179     available-electrocardiography-dataset-1.0.1/generated/1000/normalized-labels/val', window_size=window_length,
180     n_windows=n_windows, channels=slice(0,12), preload_windows=40, batch_size=train_batch_size, use_labels=True)
181     # # valset_total = ecg_datasets2.ECGDatasetMultiple([val_dataset_ptb, val_dataset_ptbx1]) #val_dataset_mit,
182     # valset_total = ecg_datasets2.ECGMultipleDatasets([val_dataset_ptb, val_dataset_ptbx1])
183     # valloader = DataLoader(valset_total, batch_size=validation_batch_size, drop_last=True, num_workers=1,
184     # collate_fn=ecg_datasets2.collate_fn)
185     train_baseline_ptbx1 = ecg_datasets2.ECGDatasetBaselineMulti(
186         '/media/julian/data/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/generated
187         /1000/normalized-labels/train',
188         # '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
189         generated/1000/normalized-labels/train',
190         window_size=9500)
191
192     val_baseline_ptbx1 = ecg_datasets2.ECGDatasetBaselineMulti(
193         '/media/julian/data/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/generated
194         /1000/normalized-labels/val',
195         # '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
196         generated/1000/normalized-labels/val',
197         window_size=9500)
198
199     model = cpc_intersect.CPC(
200         encoder=cpc_encoder_v0.Encoder(channels=channels, latent_size=args.latent_size),
201         autoregressive=cpc_autoregressive_v0.AutoRegressor(n_latents=args.latent_size,
202             hidden_size=args.hidden_size),
203         predictor=cpc_predictor_v0.Predictor(hidden_size, args.latent_size, timesteps_out),
204         latent_size=args.latent_size, timesteps_in=timesteps_in, timesteps_out=timesteps_out, verbose=False)
205
206     trainset_total = ChainDataset([train_baseline_ptbx1])
207     trainloader = DataLoader(trainset_total, batch_size=args.batch_size, drop_last=True, num_workers=1)
208
209     valset_total = ChainDataset([val_baseline_ptbx1])
210     valloader = DataLoader(valset_total, batch_size=args.batch_size, drop_last=True, num_workers=1)
211
212     if args.saved_model:
213         model, _, starting_epoch = load_model_state(args.saved_model, model, optimizer=None)
214     else:
215         torch.save(model, os.path.join(out_path, "cpc_model_full.pt"))
216         model.cuda()
217         f_m = args.forward_mode
218         downstream_model = cpc_downstream_model_multitarget_v2.DownstreamLinearNet(cpc_model=model,
219             latent_size=number_of_latents,
220             context_size=hidden_size,
221             out_classes=args.forward_classes,
222             use_context=f_m == "context" or f_m
223             use_latents=f_m == "latents" or f_m
224             == "all",
225             == "all",
226             freeze=False, verbose=False)
227         torch.save(downstream_model, os.path.join(out_path, "downstream_model_full.pt"))
228         downstream_model.cuda()
229         print()
230         optimizer = ScheduledOptim(
231             optim.Adam(
232                 filter(lambda p: p.requires_grad, downstream_model.parameters()),
233                 betas=(0.9, 0.98), eps=1e-09, weight_decay=1e-4, amsgrad=True),
234             args.warmup_steps)
235
236         best_acc = 0
237         best_loss = np.inf
238         best_epoch = -1
239         train_losses = []
240         val_losses = []
241         train_accuracies = []
242         val_accuracies = []
243
244         for epoch in range(1, epochs + 1):
245
246             train_acc, train_loss = baseline_train(downstream_model, trainloader, optimizer, epoch, None)
247             val_acc, val_loss = baseline_validation(downstream_model, valloader, None, epoch, None)
248             val_losses.append(val_loss.item())
249             train_losses.append(train_loss.item())
250             train_accuracies.append(train_acc.item())
251             val_accuracies.append(val_acc.item())
252
253             # Save
254             if val_loss < best_loss: # TODO: maybe use accuracy (not sure if accuracy is a good measurement)
255                 best_loss = val_loss
256                 best_acc = max(val_acc, best_acc)
257                 best_epoch = epoch
258                 print("saving model")
259                 torch.save(downstream_model.state_dict(), os.path.join(out_path, "cpc_model_downstream_validation.pt"))
260
261         if epoch - best_epoch >= 5:
262             # update learning rate
263             best_epoch = epoch
264             if epoch % 5 == 0:
265                 save_model_state(out_path, epoch, args.train_mode, model, optimizer,

```

```

253                     [train_accuracies, val_accuracies],
254                     [train_losses, val_losses])
255     save_model_state(out_path, epochs, args.train_mode, model, optimizer,
256                      [train_accuracies], [train_losses, val_losses])
257
258 if args.train_mode == 'test':
259     test_dataset_ptb = ecg_datasets2.ETCGDatasetBatching(
260         '/media/julian/Volume/data/ECG/ptb-diagnostic-ecg-database-1.0.0/generated/normalized/test',
261         window_size=window_length, n_windows=n_windows, channels=slice(0, 12), preload_windows=40,
262         batch_size=validation_batch_size, use_labels=True)
263
264     testset_total = ecg_datasets2.ETCGMultipleDatasets([test_dataset_ptb])
265     testloader = DataLoader(testset_total, batch_size=validation_batch_size, drop_last=True, num_workers=1,
266                            collate_fn=ecg_datasets2.collate_fn)
267
268     model.load_state_dict(torch.load(args.saved_model)) # Load the trained cpc model
269     model.eval()
270     model.freeze_layers()
271     model.cuda()
272     f_m = args.forward_mode
273     downstream_model = DownstreamLinearNet(cpc_model_trained=model, timesteps_in=timesteps_in,
274                                             context_size=hidden_size, latent_size=number_of_latents,
275                                             out_classes=args.forward_classes,
276                                             use_context=f_m == "context" or f_m == "all",
277                                             use_latents=f_m == "latents" or f_m == "all")
278     downstream_model.cuda()
279     test_acc, test_loss = down_validation(downstream_model, testloader, timesteps_in, timesteps_out, None)
280
281 if args.train_mode == 'baseline':
282     train_dataset_ptbxl = ecg_datasets2.ETCGDatasetBaselineMulti(
283         '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/generated
284         /1000/normalized-labels/train',
285         # '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
286         generated/1000/normalized-labels/train',
287         window_size=9500)
288
289     trainloader = DataLoader(train_dataset_ptbxl, batch_size=train_batch_size, drop_last=True, num_workers=1,
290                            collate_fn=ecg_datasets2.collate_fn)
291
292     val_dataset_ptbxl = ecg_datasets2.ETCGDatasetBaselineMulti(
293         '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/generated
294         /1000/normalized-labels/val',
295         # '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
296         generated/1000/normalized-labels/val',
297         window_size=9500)
298     valloader = DataLoader(val_dataset_ptbxl, batch_size=validation_batch_size, drop_last=True, num_workers=1,
299                           collate_fn=ecg_datasets2.collate_fn)
300
301     model = baseline_cnn_v0_3.BaselineNet(in_channels=args.channels, out_channels=args.latent_size,
302                                            out_classes=args.forward_classes, verbose=False)
303
304     print(model)
305     model.cuda()
306     with open(os.path.join(args.out_path, 'model_arch.txt'), 'w') as f:
307         print(fullname(model), file=f)
308         print(model, file=f)
309
310     optimizer = ScheduledOptim(
311         optim.Adam(
312             filter(lambda p: p.requires_grad, model.parameters()),
313             betas=(0.9, 0.98), eps=1e-09, weight_decay=1e-4, amsgrad=True),
314         args.warmup_steps)
315
316     start_epoch = 0
317     if args.saved_model:
318         model, optimizer, start_epoch = load_model_state(args.saved_model, model, optimizer)
319     else:
320         torch.save(model, os.path.join(out_path, 'full_model.pt'))
321
322     best_acc = 0
323     best_loss = np.inf
324     best_epoch = -1
325     train_losses = []
326     val_losses = []
327     train_accuracies = []
328
329     val_accuracies = []
330
331     for epoch in range(start_epoch + 1, epochs + start_epoch + 1):
332
333         train_acc, train_loss = baseline_train(model, trainloader, optimizer, epoch, args)
334         val_acc, val_loss = baseline_validation(model, valloader, optimizer, epoch, args)
335         val_losses.append(val_loss.item())
336         train_losses.append(train_loss.item())
337         train_accuracies.append(train_acc.item())
338         val_accuracies.append(val_acc.item())
339
340         # Save
341         if val_acc < best_acc: # TODO: maybe use accuracy (not sure if accuracy is a good measurement)
342             best_acc = max(val_acc, best_acc)
343             best_epoch = epoch

```

```

339         if epoch - best_epoch >= 5:
340             pass
341             # optimizer.update_learning_rate()
342             # best_epoch = epoch
343             if epoch % 10 == 0:
344                 save_model_state(out_path, epoch, args.train_mode, model, optimizer, [train_accuracies, val_accuracies
345                 ],
346                 [train_losses, val_losses])
347             save_model_state(out_path, epochs, args.train_mode, model, optimizer,
348                 [train_accuracies, val_accuracies], [train_losses, val_losses])
349
350     if args.train_mode == 'explain':
351
352         test_dataset_ptbxl = ecg_datasets2.ECGDatasetBaselineMulti(
353             '/media/julian/data/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/generated
354             /1000/normalized-labels/test',
355             # '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
356             generated/1000/normalized-labels/val',
357             window_size=9500)
358         totalset = ChainDataset([test_dataset_ptbxl]) # [train_dataset_ptbxl, val_dataset_ptbxl])
359         dataloader = DataLoader(totalset, batch_size=validation_batch_size, drop_last=True, num_workers=1,
360             collate_fn=ecg_datasets2.collate_fn)
361
362         bmodel, optimizer, epoch = load_model_state(
363             os.path.join(os.path.split(args.saved_model)[0], 'downstream_model_full.pt'))
364         optimizer = ScheduledOptim(
365             optim.Adam(bmodel.parameters(),
366             betas=(0.9, 0.98), eps=1e-09, weight_decay=1e-4, amsgrad=True),
367             args.warmup_steps)
368
369         bmodel, optimizer, epoch = load_model_state(args.saved_model, bmodel, optimizer)
370
371         model = explain_network.ExplainLabel(bmodel)
372         model.eval()
373         model.cuda()
374
375         ecg = PTBXLData(
376             base_directory='/media/julian/data/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset
377             -1.0.1/generated/1000/')
378         ecg.init_multilabel_encoder()
379         print(ecg.code_list)
380
381         print(model)
382         # vw = VideoWriter('images/ptbxl.mpeg', fps=2.0)
383         model.train()
384
385         for batch_idx, data_and_labels in enumerate(dataloader):
386             data, labels = data_and_labels
387             data = data.float().cuda()
388             optimizer.zero_grad()
389             output, grad = model(data, y=None)
390             output = output.detach().cpu()
391             top3 = np.argsort(output)[:, -3:]
392             # TODO: Iterate over output != 0 and calc grad respectively
393             pred_prob = []
394             for i, t3 in enumerate(top3):
395                 pred_prob.append(list(zip([ecg.code_list[t] for t in t3], output[i, t3])))
396             not0 = [[i for i, e in enumerate(a) if e != 0] for a in labels]
397
398             ground_truth = []
399             for i, t3 in enumerate(not0):
400                 ground_truth.append(list(zip([ecg.code_list[t] for t in t3], labels[i, t3])))
401             img = timeseries_to_image(grad.cpu(), grad.cpu(), downsample_factor=5, convert_to_rgb=False,
402             pred_classes=pred_prob, ground_truth=ground_truth,
403             filename='images/ptbxl/gradient/ptbxl_timeseries_' + str(batch_idx), show=False)
404             # write_video('test.mkv', img, 1.0) broken
405             # vw.tensor_to_video_continuous(img)
406             # vw.close()
407
408         if args.train_mode == 'decoder':
409             start_epoch = 1
410             # train_dataset_ptbxl = ecg_datasets2.ECGDatasetBaselineMulti('/media/julian/Volume/data/ECG/ptb-xl-a-large-
411             # publicly-available-electrocardiography-dataset-1.0.1/generated/1000/normalized-labels/train',#/media/julian/
412             # Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/generated/1000/normalized-
413             # labels/train',
414             # window_size=9500)
415             train_dataset_ptbxl = ecg_datasets2.ECGDatasetBatching(
416                 '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
417                 generated/1000/normalized-labels/train',
418                 window_size=args.window_length, n_windows=1, preload_windows=40)
419
420             trainloader = DataLoader(train_dataset_ptbxl, batch_size=train_batch_size, drop_last=False,
421                 collate_fn=ecg_datasets2.collate_fn)
422
423             # val_dataset_ptbxl = ecg_datasets2.ECGDatasetBaselineMulti('/media/julian/Volume/data/ECG/ptb-xl-a-large-
424             # publicly-available-electrocardiography-dataset-1.0.1/generated/1000/normalized-labels/val', #'media/julian/
425             # Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/generated/1000/normalized-
426             # labels/val',
427             # window_size=9500)
428             val_dataset_ptbxl = ecg_datasets2.ECGDatasetBatching(

```

```

417     '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
418     generated/1000/normalized-labels/val',
419     window_size=args.window_length, n_windows=1, preload_windows=40)
420
420     valloader = DataLoader(val_dataset_ptbxl, batch_size=validation_batch_size, drop_last=False,
421                           collate_fn=ecg_datasets2.collate_fn)
422
423     optimizer = ScheduledOptim(
424         optim.Adam(
425             filter(lambda p: p.requires_grad, model.parameters()),
426             betas=(0.9, 0.98), eps=1e-09, weight_decay=1e-4, amsgrad=True),
427             args.warmup_steps)
428
429     model = baseline_convencoder.BaselineNet(args.channels, args.latent_size, args.forward_classes)
430     if args.saved_model:
431         model, optimizer, start_epoch = load_model_state(args.saved_model, model, optimizer)
432
433     model.cuda()
434
435     print(model)
436
437     best_acc = 0
438     best_loss = np.inf
439     best_epoch = -1
440     train_losses = []
441     val_losses = []
442     train_accuracies = []
443
444     val_accuracies = []
445
446     for epoch in range(start_epoch, epochs + 1):
447
448         train_acc, train_loss = decoder_train(model, trainloader, optimizer, epoch)
449         val_acc, val_loss = decoder_validation(model, valloader, optimizer, epoch)
450         val_losses.append(val_loss.item())
451         train_losses.append(train_loss.item())
452         train_accuracies.append(train_acc.item())
453         val_accuracies.append(val_acc.item())
454
455         # Save
456         if val_loss < best_loss:
457             best_loss = val_loss
458             best_acc = max(val_acc, best_acc)
459             best_epoch = epoch
460
461         if epoch - best_epoch >= 5:
462             # update learning rate
463             optimizer.increase_delta()
464             best_epoch = epoch
465
466         if epoch % 10 == 0:
467             save_model_state(out_path, epoch, args.train_mode, model, optimizer, [train_accuracies, val_accuracies],
468                             [train_losses, val_losses])
468
469
470     if __name__ == "__main__":
471         import sys
472
473         print(sys.argv)
474
475     parser = argparse.ArgumentParser(description='Contrastive Predictive Coding')
476     parser.add_argument('--train_mode', type=str, choices=['cpc', 'downstream', 'baseline', 'decoder', 'explain'],
477                         help='Select mode. Possible: cpc, downstream, baseline, decoder')
478     # datapath
479     # Other params
480     parser.add_argument('--saved_model', type=str,
481                         help='Model path to load weights from. Has to be given for downstream mode.')
482
483     parser.add_argument('--epochs', type=int, help='The number of Epochs to train', default=100)
484
485     parser.add_argument('--seed', type=int, help='The seed used', default=None)
486
487     parser.add_argument('--forward_mode', help="The forward mode to be used.", default='context',
488                         type=str) # , choices=['context', latents, all']
489
490     parser.add_argument('--out_path', help="The output directory for losses and models",
491                         default='models/' + str(datetime.datetime.now().strftime("%d_%m_%y-%H")), type=str)
492
493     parser.add_argument('--forward_classes', type=int, default=41,
494                         help="The number of possible output classes (only relevant for downstream)")
495
496     parser.add_argument('--warmup_steps', type=int, default=0, help="The number of warmup steps")
497
498     parser.add_argument('--batch_size', type=int, default=24, help="The batch size")
499
500     parser.add_argument('--latent_size', type=int, default=512,
501                         help="The size of the latent encoding for one window")
502
503     parser.add_argument('--timesteps_in', type=int, default=6,
504                         help="The number of windows being used to form a context for prediction")

```

```

505 parser.add_argument('--timesteps_out', type=int, default=4,
506                     help="The number of windows being predicted from the context (cpc task exclusive)")
507
508 parser.add_argument('--channels', type=int, default=12,
509                     help="The number of channels the data will have") # TODO: auto detect
510
511 parser.add_argument('--window_length', type=int, default=512,
512                     help="The number of datapoints per channel per window")
513
514 parser.add_argument('--hidden_size', type=int, default=128,
515                     help="The size of the cell state/context used for predicting future latents or solving
516             downstream tasks")
517
518 parser.add_argument('--grad_clip', type=int, default=0.0,
519                     help="The number where to clip gradients at (useful if they become nan")
520
521 args = parser.parse_args()
522 main(args)

```

B.4.13 deprecated -> make_clean_data.py (code)

```

1 # from util.ecg_data import ECGData
2 import os
3 from random import shuffle
4
5 from util.data.ptbxl_data import PTBXLData
6
7
8 def ptb_clean():
9     ecg = ECGData(base_directory='/media/julian/Volume/data/ECG/ptb-diagnostic-ecg-database-1.0.0/')
10    out_path = '/media/julian/Volume/data/ECG/ptb-diagnostic-ecg-database-1.0.0/generated/normalized-pca/'
11    train_val_test_ratio = 0.7, 0.2, 0.1
12    assert sum(train_val_test_ratio) <= 1.0
13    h5_files = ecg.search_files()
14    shuffle(h5_files)
15    n = len(h5_files)
16    train_len, val_len = int(n * train_val_test_ratio[0]), int(n * train_val_test_ratio[1])
17    terms = [] # empty for now? Requires multi target loss
18    ecg.init_label_encoder(h5_files, terms, ecg.default_key_function_dict)
19    ecg.convert_dat_to_h5(os.path.join(out_path, 'train'),
20                          h5_files[0:train_len], normalize_data=True, use_header=True, terms=terms,
21                          key_function_dict=ecg.default_key_function_dict, pca_components=2, channels=slice(0, 12))
22    ecg.convert_dat_to_h5(os.path.join(out_path, 'val'),
23                          h5_files[train_len:train_len + val_len], normalize_data=True, use_header=True, terms=terms,
24                          key_function_dict=ecg.default_key_function_dict, pca_components=2, channels=slice(0, 12))
25    ecg.convert_dat_to_h5(os.path.join(out_path, 'test'),
26                          h5_files[train_len + val_len:], normalize_data=True, use_header=True, terms=terms,
27                          key_function_dict=ecg.default_key_function_dict, pca_components=2, channels=slice(0, 12))
28    print(ecg.label_mappings)
29
30
31 def ptbxl_clean():
32     ecg = PTBXLData(
33         base_directory='/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset
34         -1.0.1/generated/1000/')
35     out_path = '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
36         _generated/1000/normalized-labels/'
37     record_files_relative = ecg.search_files()
38     ecg.init_label_encoder()
39     ecg.init_multilabel_encoder()
40     train, val, test = ecg.train_test_split(record_files_relative)
41     print("final split: train %d; validation %d; test %d" % (len(train), len(val), len(test)))
42     ecg.convert_dat_to_h5(os.path.join(out_path, 'train'),
43                           train, normalize_data=True, use_labels=True, pca_components=0, channels=slice(0, 12))
44     ecg.convert_dat_to_h5(os.path.join(out_path, 'val'),
45                           val, normalize_data=True, use_labels=True, pca_components=0, channels=slice(0, 12))
46     ecg.convert_dat_to_h5(os.path.join(out_path, 'test'),
47                           test, normalize_data=True, use_labels=True, pca_components=0, channels=slice(0, 12))
48
49 if __name__ == '__main__':
50     # ptb_clean()
51     ptbxl_clean()

```

B.5 experiments (folder)

B.5.1 experiments -> create_fewer_labels_data.py (code)

```

1 import os
2 import numpy as np
3 from util.data import ecg_datasets2
4
5
6 def create_few_labels_splits(train_fraction, val_fraction, test_fraction):
7     crop_size = 4500
8     georgia_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/georgia_challenge/',
9         window_size=crop_size, pad_to_size=crop_size,
10        return_labels=True,
11        normalize_fn=ecg_datasets2.normalize_minmax_scaling,
12        verbose=False)
13     cpsc_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/cpsc2018_challenge/',
14         window_size=crop_size, pad_to_size=crop_size,
15        return_labels=True,
16        normalize_fn=ecg_datasets2.normalize_minmax_scaling,
17        verbose=False)
18     cpsc2_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/china_challenge',
19         window_size=crop_size,
20        pad_to_size=crop_size, return_labels=True,
21        normalize_fn=ecg_datasets2.normalize_minmax_scaling,
22        verbose=False)
23     ptbxl_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/ptb xl_challenge',
24         window_size=crop_size,
25        pad_to_size=crop_size, return_labels=True,
26        normalize_fn=ecg_datasets2.normalize_minmax_scaling)
27     nature = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/nature_database',
28         window_size=crop_size,
29        pad_to_size=crop_size, return_labels=True,
30        normalize_fn=ecg_datasets2.normalize_minmax_scaling,
31        verbose=False)
32
33     ecg_datasets2.filter_update_classes_by_count(
34         [georgia_challenge, cpsc_challenge, ptbxl_challenge, cpsc2_challenge, nature], min_count=20)
35     print("Warning! Redoing splits!")
36     filename = f'train-test-splits-fewer-labels{train_fraction * 100}-{val_fraction * 100}-{test_fraction * 100}.txt'
37     ptbxl_challenge.random_train_split_with_class_count(train_fraction, val_fraction, test_fraction,
38             filename_overwrite=filename)
39     cpsc_challenge.random_train_split_with_class_count(train_fraction, val_fraction, test_fraction,
40             filename_overwrite=filename)
41     cpsc2_challenge.random_train_split_with_class_count(train_fraction, val_fraction, test_fraction,
42             filename_overwrite=filename)
43     georgia_challenge.random_train_split_with_class_count(train_fraction, val_fraction, test_fraction,
44             filename_overwrite=filename)
45
46     ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.generate_datasets_from_split_file(ttsfile=filename)
47     georgia_train, georgia_val, t2 = georgia_challenge.generate_datasets_from_split_file(ttsfile=filename)
48     cpsc_train, cpsc_val, t3 = cpsc_challenge.generate_datasets_from_split_file(ttsfile=filename)
49     cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.generate_datasets_from_split_file(ttsfile=filename)
50
51     ecg_datasets2.filter_update_classes_by_count(
52         [nature, ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train,
53         cpsc2_val, t4], 1)
54     print('Classes after last update', len(ptbxl_train.classes), ptbxl_train.classes)
55     print(ecg_datasets2.count_merged_classes(
56         [nature, ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train,
57         cpsc2_val, t4]))
58
59
60 def create_few_labels_splits_like_old(train_fraction):
61     crop_size = 4500
62     georgia_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/georgia_challenge/',
63         window_size=crop_size, pad_to_size=crop_size,
64        return_labels=True,
65        normalize_fn=ecg_datasets2.normalize_minmax_scaling)
66     cpsc_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/cpsc2018_challenge/',
67         window_size=crop_size, pad_to_size=crop_size,
68        return_labels=True,
69        normalize_fn=ecg_datasets2.normalize_minmax_scaling)
70     cpsc2_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/china_challenge',
71         window_size=crop_size,
72        pad_to_size=crop_size, return_labels=True,
73        normalize_fn=ecg_datasets2.normalize_minmax_scaling)
74     ptbxl_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/ptb xl_challenge',
75         window_size=crop_size,
76        pad_to_size=crop_size, return_labels=True,
77        normalize_fn=ecg_datasets2.normalize_minmax_scaling)
78     nature = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/nature_database',
79         window_size=crop_size,
80        pad_to_size=crop_size, return_labels=True,
81        normalize_fn=ecg_datasets2.normalize_minmax_scaling)
82
83     OLD_TRAIN_FRACTION = 0.7
84     if train_fraction > OLD_TRAIN_FRACTION:
85         print(f"Can't take {train_fraction} of data as train dataset, when previous was {OLD_TRAIN_FRACTION}.")
86         return
87     else:
88         new_train_fraction = train_fraction / OLD_TRAIN_FRACTION # Between 0 and 1
89         print(f"Taking {new_train_fraction} of old train split")
90     ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.generate_datasets_from_split_file()

```

```

91     georgia_train, georgia_val, t2 = georgia_challenge.generate_datasets_from_split_file()
92     cpsc_train, cpsc_val, t3 = cpsc_challenge.generate_datasets_from_split_file()
93     cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.generate_datasets_from_split_file()
94
95     # ecg_datasets2.filter_update_classes_by_count([georgia_challenge, cpsc_challenge, ptbxl_challenge,
96     # cpsc2_challenge, nature], min_count=20)
97     print("Warning! Redoing splits!")
98     filename = f'train-test-splits-fewer-labels(str(train_fraction)).txt'
99     print(f'saving as {filename}')
100    ptbxl_new_train_split, _, _ = ptbxl_train.random_train_split_with_class_count(new_train_fraction, 0.0, 0.0,
101                                         save=False)
102    georgia_new_train_split, _, _ = georgia_train.random_train_split_with_class_count(new_train_fraction, 0.0, 0.0,
103                                         save=False)
104    cpsc_new_train_split, _, _ = cpsc_train.random_train_split_with_class_count(new_train_fraction, 0.0, 0.0,
105                                         save=False)
106    cpsc2_new_train_split, _, _ = cpsc2_train.random_train_split_with_class_count(new_train_fraction, 0.0, 0.0,
107                                         save=False)
108
109    ecg_datasets2.save_train_test_split(os.path.join(ptbxl_train.BASE_DIR, filename), ptbxl_new_train_split,
110                                         ptbxl_val.files, t1.files)
111    ecg_datasets2.save_train_test_split(os.path.join(georgia_train.BASE_DIR, filename), georgia_new_train_split,
112                                         georgia_val.files, t2.files)
113    ecg_datasets2.save_train_test_split(os.path.join(cpsc_train.BASE_DIR, filename), cpsc_new_train_split,
114                                         cpsc_val.files, t3.files)
115    ecg_datasets2.save_train_test_split(os.path.join(cpsc2_train.BASE_DIR, filename), cpsc2_new_train_split,
116                                         cpsc2_val.files, t4.files)
117
118    ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.generate_datasets_from_split_file(ttsfile=filename)
119    georgia_train, georgia_val, t2 = georgia_challenge.generate_datasets_from_split_file(ttsfile=filename)
120    cpsc_train, cpsc_val, t3 = cpsc_challenge.generate_datasets_from_split_file(ttsfile=filename)
121    cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.generate_datasets_from_split_file(ttsfile=filename)
122
123    ecg_datasets2.filter_update_classes_by_count(
124        [nature, ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train,
125         cpsc2_val, t4], 1)
126    print('Classes after last update', len(ptbxl_train.classes), ptbxl_train.classes)
127    print(ecg_datasets2.count_merged_classes(
128        [nature, ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train,
129         cpsc2_val, t4]))
130
131 def create_few_labels_splits_minimum_cut(min_cut=20):
132     crop_size = 4500
133     georgia_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/georgia_challenge/',
134                                                               window_size=crop_size, pad_to_size=crop_size,
135                                                               return_labels=True,
136                                                               normalize_fn=ecg_datasets2.normalize_minmax_scaling)
137     cpsc_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/cps2018_challenge/',
138                                                               window_size=crop_size, pad_to_size=crop_size,
139                                                               return_labels=True,
140                                                               normalize_fn=ecg_datasets2.normalize_minmax_scaling)
141     cpsc2_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/china_challenge',
142                                                               window_size=crop_size,
143                                                               pad_to_size=crop_size, return_labels=True,
144                                                               normalize_fn=ecg_datasets2.normalize_minmax_scaling)
145     ptbxl_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/ptbxl_challenge',
146                                                               window_size=crop_size,
147                                                               pad_to_size=crop_size, return_labels=True,
148                                                               normalize_fn=ecg_datasets2.normalize_minmax_scaling)
149     nature = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/nature_database',
150                                                       window_size=crop_size,
151                                                       pad_to_size=crop_size, return_labels=True,
152                                                       normalize_fn=ecg_datasets2.normalize_minmax_scaling)
153     ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.generate_datasets_from_split_file()
154     georgia_train, georgia_val, t2 = georgia_challenge.generate_datasets_from_split_file()
155     cpsc_train, cpsc_val, t3 = cpsc_challenge.generate_datasets_from_split_file()
156     cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.generate_datasets_from_split_file()
157
158     ecg_datasets2.filter_update_classes_by_count(
159         [ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train, cpsc2_val,
160          t4], 1)
161     print('Classes after last update', len(ptbxl_train.classes), ptbxl_train.classes)
162
163     class_counts, counted_classes = ecg_datasets2.count_merged_classes(
164         [ptbxl_train, georgia_train, cpsc_train, cpsc2_train])
165     extracted_counts = class_counts
166     extracted_counts[np.where(class_counts > min_cut)[0]] = min_cut
167     print(extracted_counts)
168     class_counts_dset = np.empty((4, len(class_counts))).astype(int)
169     for i, dset in enumerate([ptbxl_train, georgia_train, cpsc_train, cpsc2_train]):
170         c_count, counted_c = ecg_datasets2.count_merged_classes([dset])
171         class_counts_dset[i] = np.array(c_count)
172     dsets = [ptbxl_train, georgia_train, cpsc_train, cpsc2_train]
173     buckets = np.zeros((4, len(class_counts))).astype(int)
174     sort_idx = np.argsort(class_counts)
175     for ix in sort_idx:
176         # find this label in files.
177         dset_order = np.argsort(class_counts_dset[:, ix])
178         have = 0
179         for i, dset_i in enumerate(dset_order):

```

```

180     n_per_dataset = int(np.ceil((extracted_counts[ix] - have) / (len(dsets) - i)))
181     cc_in_dset = class_counts_dset[dset_i, ix]
182     take_n_files = min(cc_in_dset, n_per_dataset)
183     buckets[dset_i, ix] = take_n_files
184     have += take_n_files
185     print('Found buckets', buckets)
186     filename = f'train-test-splits_min_cut{min_cut}.txt'
187
188     ptbxl_new_train_split, _, _ = ptbxl_train.random_train_split_with_class_count_mins(buckets[0],
189                                         np.zeros_like(buckets[0]),
190                                         np.zeros_like(buckets[0]),
191                                         save=False)
192     georgia_new_train_split, _, _ = georgia_train.random_train_split_with_class_count_mins(buckets[1],
193                                         np.zeros_like(buckets[1]),
194                                         np.zeros_like(buckets[1]),
195                                         save=False)
196     cpsc_new_train_split, _, _ = cpsc_train.random_train_split_with_class_count_mins(buckets[2],
197                                         np.zeros_like(buckets[2]),
198                                         np.zeros_like(buckets[2]),
199                                         save=False)
200     cpsc2_new_train_split, _, _ = cpsc2_train.random_train_split_with_class_count_mins(buckets[3],
201                                         np.zeros_like(buckets[3]),
202                                         np.zeros_like(buckets[3]),
203                                         save=False)
204
205     ecg_datasets2.save_train_test_split(os.path.join(ptbxl_train.BASE_DIR, filename), ptbxl_new_train_split,
206                                         ptbxl_val.files, t1.files)
207     ecg_datasets2.save_train_test_split(os.path.join(georgia_train.BASE_DIR, filename), georgia_new_train_split,
208                                         georgia_val.files, t2.files)
209     ecg_datasets2.save_train_test_split(os.path.join(cpsc_train.BASE_DIR, filename), cpsc_new_train_split,
210                                         cpsc_val.files, t3.files)
211     ecg_datasets2.save_train_test_split(os.path.join(cpsc2_train.BASE_DIR, filename), cpsc2_new_train_split,
212                                         cpsc2_val.files, t4.files)
213
214
215 if __name__ == '__main__': # DO NOT CALL UNLESS YOU WANT TO OVERWRITE
216     create_few_labels_splits_like_old(train_fraction=0.001)
217     create_few_labels_splits_like_old(train_fraction=0.005)
218     create_few_labels_splits_like_old(train_fraction=0.01)
219     create_few_labels_splits_like_old(train_fraction=0.05)
220     # create_few_labels_splits_like_old(train_fraction=0.1)
221     # create_few_labels_splits_like_old(train_fraction=0.2)
222     # create_few_labels_splits_like_old(train_fraction=0.3)
223     # create_few_labels_splits_like_old(train_fraction=0.4)
224     # create_few_labels_splits_like_old(train_fraction=0.5)
225     # create_few_labels_splits_like_old(train_fraction=0.6)
226     # create_few_labels_splits_minimum_cut(min_cut=200)
227     # create_few_labels_splits_minimum_cut(min_cut=150)
228     # create_few_labels_splits_minimum_cut(min_cut=100)
229     # create_few_labels_splits_minimum_cut(min_cut=50)
230     # create_few_labels_splits_minimum_cut(min_cut=25)
231     # create_few_labels_splits_minimum_cut(min_cut=3)

```

B.5.2 experiments -> create_timeseries_plots.py (code)

```

1 import json
2 import os
3 import numpy as np
4 from torch.utils.data import DataLoader, ChainDataset
5
6 from util.data import ecg_datasets2
7 from util.visualize.timeseries_to_image_converter import timeseries_to_image
8
9
10 def create_timeseries_image():
11     crop_size = 4500
12     georgia_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/georgia_challenge/',
13                                         window_size=crop_size, pad_to_size=crop_size,
14                                         return_labels=True, return_filename=True,
15                                         normalize_fn=ecg_datasets2.normalize_minmax_scaling)
16     cpsc_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/cpsc2018_challenge/',
17                                         window_size=crop_size, pad_to_size=crop_size,
18                                         return_labels=True, return_filename=True,
19                                         normalize_fn=ecg_datasets2.normalize_minmax_scaling)
20     cpsc2_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/china_challenge',
21                                         window_size=crop_size,
22                                         pad_to_size=crop_size, return_labels=True,
23                                         return_filename=True,
24                                         normalize_fn=ecg_datasets2.normalize_minmax_scaling)
25     ptbxl_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/ptbxl_challenge',
26                                         window_size=crop_size,
27                                         pad_to_size=crop_size, return_labels=True,
28                                         return_filename=True,
29                                         normalize_fn=ecg_datasets2.normalize_minmax_scaling)
30     ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.generate_datasets_from_split_file()
31     georgia_train, georgia_val, t2 = georgia_challenge.generate_datasets_from_split_file()

```

```

32 cpsc_train, cpsc_val, t3 = cpsc_challenge.generate_datasets_from_split_file()
33 cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.generate_datasets_from_split_file()
34 ecg_datasets2.filter_update_classes_by_count(
35     [ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train, cpsc2_val,
36      t4], 1)
37 counts_all, counted_classes_all = ecg_datasets2.count_merged_classes(
38     [ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train, cpsc2_val,
39      t4])
40 print(counted_classes_all)
41 with open('/experiments/snomed_data.json', 'r') as f:
42     snomed_data = json.load(f)
43     inverted_named_counted_classes = {v: snomed_data[k]['pt']['term'] for k, v in counted_classes_all.items()}
44     dl = DataLoader(ChainDataset([t1, t2, t3, t4]), batch_size=1)
45     for data, labels, filenames in dl:
46         data = data.float().cpu()
47         labels = labels.cpu().numpy()
48         print(np.nonzero(labels[0])[0])
49         fname = os.path.basename(filenames[0])
50         ground_truth = [inverted_named_counted_classes[i] for i in np.nonzero(labels[0])[0]]
51         timeseries_to_image(data, grad=None, pred_classes=None, ground_truth=[ground_truth], save=True, show=True,
52                             downsample_factor=1,
53                             filename=f'/home/julian/Downloads/Github/contrastive-predictive-coding/images/
54                             data_visualization/{fname}')
55
56 if __name__ == '__main__': # DO NOT CALL UNLESS YOU WANT TO OVERWRITE
57     create_timeseries_image()

```

B.5.3 experiments -> main_get_latents.py (code)

```

1 import argparse
2 import datetime
3 import os
4 import pickle
5 import time
6 from pathlib import Path
7
8 import numpy as np
9 import torch
10 from torch.utils.data import DataLoader, ChainDataset
11
12 from util.data import ecg_datasets2
13 from util.store_models import load_model_checkpoint, load_model_architecture, extract_model_files_from_dir
14
15
16 def main(args):
17     np.random.seed(args.seed)
18     # os.environ['CUDA_LAUNCH_BLOCKING'] = '1'
19     torch.cuda.set_device(args.gpu_device)
20     print(f'Device set to : {torch.cuda.current_device()}. Selected was {args.gpu_device}')
21     torch.cuda.manual_seed(args.seed)
22     print(f'Seed set to : {args.seed}.')
23     print(f'Model outputpath: {args.out_path}')
24     Path(args.out_path).mkdir(parents=True, exist_ok=True)
25     with open(os.path.join(args.out_path, 'params.txt'), 'w') as cfg:
26         cfg.write(str(args))
27     # georgia = ecg_datasets2.ECGChallengeDatasetBaseline('/home/juwin106/data/georgia/WFDB', window_size=4500,
28     # pad_to_size=4500, use_labels=True)
29     # cpsc_train = ecg_datasets2.ECGChallengeDatasetBaseline('/home/juwin106/data/cpsc_train', window_size=4500,
30     # pad_to_size=4500, use_labels=True)
31     # cpsc = ecg_datasets2.ECGChallengeDatasetBaseline('/home/juwin106/data/cpsc', window_size=4500, pad_to_size=4500,
32     # use_labels=True)
33     # ptbxl = ecg_datasets2.ECGChallengeDatasetBaseline('/home/juwin106/data/ptbxl/WFDB', window_size=4500,
34     # pad_to_size=4500, use_labels=True)
35
36     georgia_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/georgia_challenge/',
37         window_size=args.crop_size,
38         pad_to_size=args.crop_size,
39         return_labels=True, return_filename=True,
40         normalize_fn=ecg_datasets2.normalize_minmax_scaling)
41     cpsc_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/cps2018_challenge/',
42         window_size=args.crop_size, pad_to_size=args.crop_size,
43         return_labels=True, return_filename=True,
44         normalize_fn=ecg_datasets2.normalize_minmax_scaling)
45     cpsc2_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/china_challenge',
46         window_size=args.crop_size, pad_to_size=args.crop_size
47         ,
48         return_labels=True, return_filename=True,
49         normalize_fn=ecg_datasets2.normalize_minmax_scaling)
50     ptbxl_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/ptbxl_challenge',
51         window_size=args.crop_size, pad_to_size=args.crop_size
52         ,
53         return_labels=True, return_filename=True,
54         normalize_fn=ecg_datasets2.normalize_minmax_scaling)
55
56     a1, b1, ptbxl_test = ptbxl_challenge.generate_datasets_from_split_file()

```

```

51     a2, b2, georgia_test = georgia_challenge.generate_datasets_from_split_file()
52     a3, b3, cpdc_test = cpdc_challenge.generate_datasets_from_split_file()
53     a4, b4, cpdc2_test = cpdc2_challenge.generate_datasets_from_split_file()
54
55     classes = ecg_datasets2.filter_update_classes_by_count(
56         [a1, b1, ptbxl_test, a2, b2, georgia_test, a3, b3, cpdc_test, a4, b4, cpdc2_test],
57         1) # Set classes if specified in split files (filter out classes with no occurrence)
58     print(classes)
59     print(classes == {'10370003': 0, '11157007': 1, '111975006': 2, '164861001': 3, '164865005': 4, '164867002': 5,
60         '164873001': 6, '164884008': 7, '164889003': 8, '164890007': 9, '164909002': 10, '164917005': 11,
61             '164930006': 12, '164931005': 13, '164934002': 14, '164947007': 15, '164951009': 16,
62             '17338001': 17, '195042002': 18, '195080001': 19, '195126007': 20, '233917008': 21,
63             '251120003': 22, '251146004': 23, '251180001': 24, '251200008': 25, '251266004': 26,
64             '251268003': 27, '253352002': 28, '266249003': 29, '270492004': 30, '27885002': 31,
65             '284470004': 32, '39732003': 33, '413844008': 34, '425419005': 35, '425623009': 36,
66             '426177001': 37, '426434006': 38, '426627000': 39, '426761007': 40, '426783006': 41,
67             '427084000': 42, '427172004': 43, '427393009': 44, '428417006': 45, '428750005': 46,
68             '429622005': 47, '445211001': 49, '446358003': 50, '446813000': 51,
69             '47665007': 52, '54329005': 53, '55930002': 54, '59118001': 55, '59931005': 56, '63593006': 57,
70             '6374002': 58, '67198005': 59, '67741000119109': 60, '698252002': 61, '713422000': 62,
71             '713426002': 63, '713427006': 64, '74390002': 65, '89792004': 66})
72
73     train_dataset_challenge = ChainDataset([a1, a2, a3, a4])
74     val_dataset_challenge = ChainDataset([b1, b2, b3, b4])
75     test_dataset_challenge = ChainDataset([ptbxl_test, georgia_test, cpdc_test, cpdc2_test])
76     # all_dataset_challenge = ChainDataset(ptbxl_challenge, georgia_challenge, cpdc_challenge, cpdc2_challenge)
77     model_folders = [
78         #'home/julian/Downloads/Github/contrastive-predictive-coding/models_symbolic_links/train/class_weights/14_05_21-15-train|(8x)cpc',
79         #'home/julian/Downloads/Github/contrastive-predictive-coding/models_symbolic_links/train/class_weights/19_05_21-16-train|cpc',
80         #'home/julian/Downloads/Github/contrastive-predictive-coding/models_symbolic_links/train/class_weights/19_05_21-17-train|cpc',
81         #'home/julian/Downloads/Github/contrastive-predictive-coding/models_symbolic_links/train/class_weights/20_05_21-18-train|(2x)bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v14+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4+bl_cnn_v5+bl_cnn_v6+bl_cnn_v8+bl_cnn_v9',
82         #'home/julian/Downloads/Github/contrastive-predictive-coding/models_symbolic_links/train/class_weights/25_05_21-13-train|bl_FCN' #used class weights
83         #'models_symbolic_links/train/correct-age/class_weights/'
84         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/28_01_22-16-23-train|(12x)cpc/architectures_cpc.cpc_combined.CPCCombined3|train-test-splits|use_weights|strided|frozen|C|L|m:all|cpc_downstream_latent_maximum'
85         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/28_01_22-14-24-train|(12x)cpc/architectures_cpc.cpc_combined.CPCCombined2|train-test-splits|use_weights|unfrozen|C|L|m:all|cpc_downstream_latent_maximum'
86         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/28_01_22-16-23-train|(12x)cpc/architectures_cpc.cpc_combined.CPCCombined2|train-test-splits|use_weights|frozen|C|L|m:all|cpc_downstream_latent_maximum'
87         #'***home/julian/Downloads/Github/contrastive-predictive-coding/models/14_08_21-15_36-train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined1|train-test-splits-fewer-labels60|use_weights|strided|frozen|C|m:all|cpc_downstream_cnn''.split(
88             '\n')
89
90     ]
91     # infer class from model-arch file
92     model_dicts = []
93     for train_folder in model_folders:
94         model_files = extract_model_files_from_dir(train_folder)
95         for mfile in model_files:
96             fm_fs, cp_fs, root = mfile
97             fm_f = fm_fs[0]
98             cp_f = sorted(cp_fs)[-1]
99             model = load_model_architecture(fm_f)
100            if model is None:
101                continue
102            model, _, epoch = load_model_checkpoint(cp_f, model, optimizer=None, device_id=f'cuda:{args.gpu_device}')
103            print(
104                f'Found architecturefile {os.path.basename(fm_f)}, checkpointfile {os.path.basename(cp_f)} in folder {root}. Appending model for testing.')
105            model_dicts.append({'model': model, 'model_folder': root})
106    if len(model_dicts) == 0:
107        print(f"Could not find any models in {model_folders}.")
108    loaders = [
109        DataLoader(test_dataset_challenge, batch_size=args.batch_size, drop_last=False, num_workers=1,
110            collate_fn=ecg_datasets2.collate_fn),
111        DataLoader(val_dataset_challenge, batch_size=args.batch_size, drop_last=False, num_workers=1,
112            collate_fn=ecg_datasets2.collate_fn),
113        # DataLoader(train_dataset_challenge, batch_size=args.batch_size, drop_last=False, num_workers=1,
114        #     collate_fn=ecg_datasets2.collate_fn), #Train usually not required
115    ]
116
117    for model_i, model_dict in enumerate(model_dicts):
118        combined_model = model_dict['model']
119        model = combined_model.cpc_model
120        model.cpc_train_mode = False
121        model_name = os.path.split(model_dict['model_folder'])[1]
122        train_folder = os.path.split(os.path.split(model_dict['model_folder'])[0])[1]
123        output_path = os.path.join(args.out_path, train_folder, model_name)

```

```

125     print("Evaluating {}. Output will be saved to dir: {}".format(model_name, output_path))
126     # Create dirs and model info
127     Path(output_path).mkdir(parents=True, exist_ok=True)
128     model.cuda()
129     latent_list = []
130     for epoch in range(1, 2):
131         starttime = time.time()
132         for loader_i, loader in enumerate(loaders):
133             for dataset_tuple in loader:
134                 with torch.no_grad():
135                     data, labels, filenames = dataset_tuple
136                     data = data.float().cuda()
137                     labels = labels.float().cuda()
138                     encoded_x, context, hidden = model(data,
139                                                 y=None) # makes model return prediction instead of loss
140
141                     encoded_x = encoded_x.detach().cpu()
142                     context = context.detach().cpu()
143                     labels = labels.cpu()
144                     labels_numpy = parse_tensor_to_numpy_or_scalar(labels)
145                     encoded_x_numpy = parse_tensor_to_numpy_or_scalar(encoded_x)
146                     context_numpy = parse_tensor_to_numpy_or_scalar(context)
147                     latent_list.append(
148                         {'labels': np.nonzero(labels_numpy), 'latents': encoded_x_numpy, 'context': context_numpy})
149
150             del data, labels, encoded_x, encoded_x_numpy, context, context_numpy
151             torch.cuda.empty_cache()
152             elapsed_time = str(datetime.timedelta(seconds=time.time() - starttime))
153             print("Done. Elapsed time: {}".format(elapsed_time))
154             if args.dry_run:
155                 break
156         with open(os.path.join(output_path, 'latent_list.pickle'), 'wb') as f:
157             pickle.dump(latent_list, f)
158
159     del model # delete and free
160     torch.cuda.empty_cache()
161
162 def save_dict_to_csv_file(filepath, data, column_names=None):
163     with open(filepath) as f:
164         if not column_names is None:
165             f.write(','.join(column_names) + '\n')
166         for dc in data:
167             f.write(','.join(dc) + '\n')
168
169
170 def parse_tensor_to_numpy_or_scalar(input_tensor):
171     if type(input_tensor) == torch.Tensor:
172         arr = input_tensor.detach().cpu().numpy() if input_tensor.is_cuda else input_tensor.numpy()
173         if arr.size == 1:
174             return arr.item()
175         return arr
176     return input_tensor
177
178
179 if __name__ == "__main__":
180     import sys
181
182     print(sys.argv)
183
184     parser = argparse.ArgumentParser(description='Contrastive Predictive Coding')
185     # datapath
186     # Other params
187     parser.add_argument('--saved_model', type=str,
188                         help='Model path to load weights from. Has to be given for downstream mode.')
189
190     parser.add_argument('--pretrain_epochs', type=int, help='The number of Epochs to pretrain', default=100)
191
192     parser.add_argument('--downstream_epochs', type=int, help='The number of Epochs to downtrain', default=100)
193
194     parser.add_argument('--seed', type=int, help='The seed used', default=0)
195
196     parser.add_argument('--forward_mode', help="The forward mode to be used.", default='context',
197                         type=str) # , choices=['context', latents, all']
198
199     parser.add_argument('--out_path', help="The output directory for losses and models",
200                         default='data_output/' + str(datetime.datetime.now().strftime("%d_%m_%y-%H")), type=str)
201
202     parser.add_argument('--forward_classes', type=int, default=67,
203                         help="The number of possible output classes (only relevant for downstream)")
204
205     parser.add_argument('--warmup_steps', type=int, default=0, help="The number of warmup steps")
206
207     parser.add_argument('--batch_size', type=int, default=1, help="The batch size")
208
209     parser.add_argument('--latent_size', type=int, default=128,
210                         help="The size of the latent encoding for one window")
211
212     parser.add_argument('--timesteps_in', type=int, default=6,
213                         help="The number of windows being used to form a context for prediction")

```

```

214
215     parser.add_argument('--timesteps_out', type=int, default=6,
216                         help="The number of windows being predicted from the context (cpo task exclusive)")
217
218     parser.add_argument('--channels', type=int, default=12,
219                         help="The number of channels the data will have") # TODO: auto detect
220
221     parser.add_argument('--window_size', type=int, default=512,
222                         help="The number of datapoints per channel per window")
223
224     parser.add_argument('--crop_size', type=int, default=4500,
225                         help="The size of the data that it is cropped to. If data is smaller than this number, data
226     gets padded with zeros")
227
228     parser.add_argument('--hidden_size', type=int, default=512,
229                         help="The size of the cell state/context used for predicting future latents or solving
230     downstream tasks")
231
232     parser.add_argument('--dry_run', dest='dry_run', action='store_true',
233                         help="Only run minimal samples to test all models functionality")
234     parser.set_defaults(dry_run=False)
235
236     parser.add_argument('--use_class_weights', dest='use_class_weights', action='store_true',
237                         help="Use class weights determined by datasets class count")
238     parser.set_defaults(use_class_weights=False)
239
240     parser.add_argument('--redo_splits', dest='redo_splits', action='store_true',
241                         help="Redo splits. Warning! File will be overwritten!")
242     parser.set_defaults(redo_splits=False)
243
244     parser.add_argument("--gpu_device", type=int, default=0)
245
246     parser.add_argument("--comment", type=str, default=None)
247
248     args = parser.parse_args()
249     main(args)

```

B.6 external (folder)

B.6.1 external -> tcn (folder)

B.6.1.1 external -> tcn -> TCN (folder)

B.6.1.1.1 external -> tcn -> TCN -> tcn.py (code)

```

1 import torch.nn as nn
2 from torch.nn.utils import weight_norm
3
4
5 class Chomp1d(nn.Module):
6     def __init__(self, chomp_size):
7         super(Chomp1d, self).__init__()
8         self.chomp_size = chomp_size
9
10    def forward(self, x):
11        return x[:, :, :-self.chomp_size].contiguous()
12
13
14 class TemporalBlock(nn.Module):
15    def __init__(self, n_inputs, n_outputs, kernel_size, stride, dilation, padding, dropout=0.2):
16        super(TemporalBlock, self).__init__()
17        self.conv1 = weight_norm(nn.Conv1d(n_inputs, n_outputs, kernel_size,
18                                         stride=stride, padding=padding, dilation=dilation))
19        self.chomp1 = Chomp1d(padding)
20        self.relu1 = nn.ReLU()
21        self.dropout1 = nn.Dropout(dropout)
22
23        self.conv2 = weight_norm(nn.Conv1d(n_outputs, n_outputs, kernel_size,
24                                         stride=stride, padding=padding, dilation=dilation))
25        self.chomp2 = Chomp1d(padding)
26        self.relu2 = nn.ReLU()
27        self.dropout2 = nn.Dropout(dropout)
28
29        self.net = nn.Sequential(self.conv1, self.chomp1, self.relu1, self.dropout1,
30                               self.conv2, self.chomp2, self.relu2, self.dropout2)
31        self.downsample = nn.Conv1d(n_inputs, n_outputs, 1) if n_inputs != n_outputs else None
32        self.relu = nn.ReLU()

```

```

33     self.init_weights()
34
35     def init_weights(self):
36         self.conv1.weight.data.normal_(0, 0.01)
37         self.conv2.weight.data.normal_(0, 0.01)
38         if self.downsample is not None:
39             self.downsample.weight.data.normal_(0, 0.01)
40
41     def forward(self, x):
42         out = self.net(x)
43         res = x if self.downsample is None else self.downsample(x)
44         return self.relu(out + res)
45
46
47 class TemporalConvNet(nn.Module):
48     def __init__(self, num_inputs, num_channels, kernel_size=2, dropout=0.2):
49         super(TemporalConvNet, self).__init__()
50         layers = []
51         num_levels = len(num_channels)
52         for i in range(num_levels):
53             dilation_size = 2 ** i
54             in_channels = num_inputs if i == 0 else num_channels[i - 1]
55             out_channels = num_channels[i]
56             layers += [TemporalBlock(in_channels, out_channels, kernel_size, stride=1, dilation=dilation_size,
57                                     padding=(kernel_size - 1) * dilation_size, dropout=dropout)]
58
59         self.network = nn.Sequential(*layers)
60
61     def forward(self, x):
62         return self.network(x)

```

B.6.2 external -> helper_code.py (code)

```

1 #!/usr/bin/env python
2
3 # These are helper variables and functions that you can use with your code.
4 # Do not edit this script.
5
6 import numpy as np
7 import os
8 from scipy.io import loadmat
9
10 # Define 12, 6, and 2 lead ECG sets.
11 twelve_leads = ('I', 'II', 'III', 'aVR', 'aVL', 'aVF', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6')
12 six_leads = ('I', 'II', 'III', 'aVR', 'aVL', 'aVF')
13 two_leads = ('II', 'V5')
14
15
16 # Check if a variable is an integer or represents an integer.
17 def is_integer(x):
18     try:
19         if int(x) == float(x):
20             return True
21         else:
22             return False
23     except (ValueError, TypeError):
24         return False
25
26
27 # Find header and recording files.
28 def find_challenge_files(data_directory):
29     header_files = list()
30     recording_files = list()
31     for f in os.listdir(data_directory):
32         root, extension = os.path.splitext(f)
33         if not root.startswith('.') and extension == '.hea':
34             header_file = os.path.join(data_directory, root + '.hea')
35             recording_file = os.path.join(data_directory, root + '.mat')
36             if os.path.isfile(header_file) and os.path.isfile(recording_file):
37                 header_files.append(header_file)
38                 recording_files.append(recording_file)
39     return header_files, recording_files
40
41
42 # Load header file as a string.
43 def load_header(header_file):
44     with open(header_file, 'r') as f:
45         header = f.read()
46     return header
47
48
49 # Load recording file as an array.
50 def load_recording(recording_file, header=None, leads=None, key='val'):
51     x = loadmat(recording_file)[key]
52     recording = np.asarray(x, dtype=np.float32)
53     return recording

```

```

54
55
56 # Get leads from header.
57 def get_leads(header):
58     leads = []
59     for i, l in enumerate(header.split('\n')):
60         entries = l.split(' ')
61         if i == 0:
62             num_leads = int(entries[1])
63         elif i <= num_leads:
64             leads.append(entries[-1])
65         else:
66             break
67     return leads
68
69
70 # Get age from header.
71 def get_age(header):
72     age = None
73     for l in header.split('\n'):
74         if l.startswith('#Age'):
75             try:
76                 age = float(l.split(': ')[1].strip())
77             except:
78                 age = float('nan')
79     return age
80
81
82 # Get sex from header.
83 def get_sex(header):
84     sex = None
85     for l in header.split('\n'):
86         if l.startswith('#Sex'):
87             try:
88                 sex = l.split(': ')[1].strip()
89             except:
90                 pass
91     return sex
92
93
94 # Get frequency from header.
95 def get_frequency(header):
96     frequency = None
97     for i, l in enumerate(header.split('\n')):
98         if i == 0:
99             try:
100                 frequency = float(l.split(' ')[2])
101             except:
102                 pass
103         else:
104             break
105     return frequency
106
107
108 # Get amplitudes from header.
109 def get_amplitudes(header, leads):
110     amplitudes = np.zeros(len(leads), dtype=np.float32)
111     for i, l in enumerate(header.split('\n')):
112         entries = l.split(' ')
113         if i == 0:
114             num_leads = int(entries[1])
115         elif i <= num_leads:
116             current_lead = entries[-1]
117             if current_lead in leads:
118                 j = leads.index(current_lead)
119                 try:
120                     amplitudes[j] = float(entries[2].split('/')[0])
121                 except:
122                     pass
123             else:
124                 break
125     return amplitudes
126
127
128 # Get baselines from header.
129 def get_baselines(header, leads):
130     baselines = np.zeros(len(leads), dtype=np.float32)
131     for i, l in enumerate(header.split('\n')):
132         entries = l.split(' ')
133         if i == 0:
134             num_leads = int(entries[1])
135         elif i <= num_leads:
136             current_lead = entries[-1]
137             if current_lead in leads:
138                 j = leads.index(current_lead)
139                 try:
140                     baselines[j] = float(entries[4].split('/')[0])
141                 except:
142                     pass
143             else:

```

```

144         break
145     return baselines
146
147
148 # Get labels from header.
149 def get_labels(header):
150     labels = list()
151     for l in header.split('\n'):
152         if l.startswith('#Dx'):
153             entries = l.split(': ')[1].split(',')
154             for entry in entries:
155                 labels.append(entry.strip())
156
157
158
159 # Save outputs from model.
160 def save_outputs(output_file, classes, labels, probabilities):
161     # Extract the recording identifier from the filename.
162     head, tail = os.path.split(output_file)
163     root, extension = os.path.splitext(tail)
164     recording_identifier = root
165
166     # Format the model outputs.
167     recording_string = '#{}'.format(recording_identifier)
168     class_string = '/'.join(str(c) for c in classes)
169     label_string = '/'.join(str(l) for l in labels)
170     probabilities_string = '/'.join(str(p) for p in probabilities)
171     output_string = recording_string + '\n' + class_string + '\n' + label_string + '\n' + probabilities_string + '\n'
172
173     # Save the model outputs.
174     with open(output_file, 'w') as f:
175         f.write(output_string)
176
177
178 def get_classes(files_without_ext):
179     classes = set()
180     for filename in files_without_ext:
181         with open(filename + '.hea', 'r') as f:
182             for l in f:
183                 if l.startswith('#Dx'):
184                     tmp = l.split(':')[1].split(',')
185                     for c in tmp:
186                         classes.add(c.strip())
187
188     return dict(zip(sorted(classes), range(len(classes))))
189
190 def encode_header_labels(header, classes, onerror_class='426783006'):
191     labels_act = np.zeros(len(classes))
192     for l in header.split('\n'):
193         if l.startswith('#Dx'):
194             tmp = l.split(':')[1].split(',')
195             for c in tmp:
196                 cl = c.strip()
197                 if cl in classes:
198                     class_index = classes[cl]
199                 else:
200                     if onerror_class is None:
201                         return None
202                     class_index = classes[onerror_class]
203                     labels_act[class_index] = 1
204
205     return labels_act
206
207 def save_challenge_predictions(output_directory, filenames, classes, scores, labels):
208     for i in range(0, len(filenames)):
209         filename = filenames[i]
210         sc = scores[i]
211         ls = labels[i]
212         recording = os.path.splitext(filename)[0]
213         new_file = filename.replace('.mat', '') + '.csv'
214
215         output_file = os.path.join(output_directory, new_file)
216
217         # Include the filename as the recording number
218         recording_string = '#{}'.format(recording)
219         class_string = '/'.join(classes)
220         label_string = '/'.join(str(i) for i in ls)
221         score_string = '/'.join(str(i) for i in sc)
222         with open(output_file, 'w') as f:
223             f.write(recording_string + '\n' + class_string + '\n' + label_string + '\n' + score_string + '\n')

```

B.6.3 external -> multi_scale_ori.py (code)

```

1 # author: https://github.com/geekfeiw/Multi-Scale-1D-ResNet
2 import torch
3 # author: https://github.com/geekfeiw/Multi-Scale-1D-ResNet

```

```

4 import torch
5 import torch.nn as nn
6
7
8 def conv3x3(in_planes, out_planes, stride=1):
9     """3x3 convolution with padding"""
10    return nn.Conv1d(in_planes, out_planes, kernel_size=3, stride=stride,
11                   padding=1, bias=False)
12
13
14 def conv5x5(in_planes, out_planes, stride=1):
15    return nn.Conv1d(in_planes, out_planes, kernel_size=5, stride=stride,
16                   padding=1, bias=False)
17
18
19 def conv7x7(in_planes, out_planes, stride=1):
20    return nn.Conv1d(in_planes, out_planes, kernel_size=7, stride=stride,
21                   padding=1, bias=False)
22
23
24 class BasicBlock3x3(nn.Module):
25     expansion = 1
26
27     def __init__(self, inplanes3, planes, stride=1, downsample=None):
28         super(BasicBlock3x3, self).__init__()
29         self.conv1 = conv3x3(inplanes3, planes, stride)
30         self.bn1 = nn.BatchNorm1d(planes)
31         self.relu = nn.ReLU(inplace=True)
32         self.conv2 = conv3x3(planes, planes)
33         self.bn2 = nn.BatchNorm1d(planes)
34         self.downsample = downsample
35         self.stride = stride
36
37     def forward(self, x):
38         residual = x
39
40         out = self.conv1(x)
41         out = self.bn1(out)
42         out = self.relu(out)
43
44         out = self.conv2(out)
45         out = self.bn2(out)
46
47         if self.downsample is not None:
48             residual = self.downsample(x)
49
50         out += residual
51         out = self.relu(out)
52
53         return out
54
55
56 class BasicBlock5x5(nn.Module):
57     expansion = 1
58
59     def __init__(self, inplanes5, planes, stride=1, downsample=None):
60         super(BasicBlock5x5, self).__init__()
61         self.conv1 = conv5x5(inplanes5, planes, stride)
62         self.bn1 = nn.BatchNorm1d(planes)
63         self.relu = nn.ReLU(inplace=True)
64         self.conv2 = conv5x5(planes, planes)
65         self.bn2 = nn.BatchNorm1d(planes)
66         self.downsample = downsample
67         self.stride = stride
68
69     def forward(self, x):
70         residual = x
71
72         out = self.conv1(x)
73         out = self.bn1(out)
74         out = self.relu(out)
75
76         out = self.conv2(out)
77         out = self.bn2(out)
78
79         if self.downsample is not None:
80             residual = self.downsample(x)
81
82         d = residual.shape[2] - out.shape[2]
83         out1 = residual[:, :, 0:-d] + out
84         out1 = self.relu(out1)
85         # out += residual
86
87         return out1
88
89
90 class BasicBlock7x7(nn.Module):
91     expansion = 1
92
93     def __init__(self, inplanes7, planes, stride=1, downsample=None):

```

```

94     super(BasicBlock7x7, self).__init__()
95     self.conv1 = conv7x7(inplanes7, planes, stride)
96     self.bn1 = nn.BatchNorm1d(planes)
97     self.relu = nn.ReLU(inplace=True)
98     self.conv2 = conv7x7(planes, planes)
99     self.bn2 = nn.BatchNorm1d(planes)
100    self.downsample = downsample
101    self.stride = stride
102
103   def forward(self, x):
104       residual = x
105
106       out = self.conv1(x)
107       out = self.bn1(out)
108       out = self.relu(out)
109
110       out = self.conv2(out)
111       out = self.bn2(out)
112
113       if self.downsample is not None:
114           residual = self.downsample(x)
115
116       d = residual.shape[2] - out.shape[2]
117       out1 = residual[:, :, 0:-d] + out
118       out1 = self.relu(out1)
119       # out += residual
120
121       return out1
122
123
124 class MSResNet(nn.Module):
125     def __init__(self, input_channel, layers=[1, 1, 1, 1], num_classes=10):
126         self.inplanes3 = 64
127         self.inplanes5 = 64
128         self.inplanes7 = 64
129
130         super(MSResNet, self).__init__()
131
132         self.conv1 = nn.Conv1d(input_channel, 64, kernel_size=7, stride=2, padding=3,
133                             bias=False)
134         self.bn1 = nn.BatchNorm1d(64)
135         self.relu = nn.ReLU(inplace=True)
136         self.maxpool = nn.MaxPool1d(kernel_size=3, stride=2, padding=1)
137
138         self.layer3x3_1 = self._make_layer3(BasicBlock3x3, 64, layers[0], stride=2)
139         self.layer3x3_2 = self._make_layer3(BasicBlock3x3, 128, layers[1], stride=2)
140         self.layer3x3_3 = self._make_layer3(BasicBlock3x3, 256, layers[2], stride=2)
141         # self.layer3x3_4 = self._make_layer3(BasicBlock3x3, 512, layers[3], stride=2)
142
143         # maxpooling kernel size: 16, 11, 6
144         self.maxpool3 = nn.AvgPool1d(kernel_size=16, stride=1, padding=0)
145
146         self.layer5x5_1 = self._make_layer5(BasicBlock5x5, 64, layers[0], stride=2)
147         self.layer5x5_2 = self._make_layer5(BasicBlock5x5, 128, layers[1], stride=2)
148         self.layer5x5_3 = self._make_layer5(BasicBlock5x5, 256, layers[2], stride=2)
149         # self.layer5x5_4 = self._make_layer5(BasicBlock5x5, 512, layers[3], stride=2)
150         self.maxpool5 = nn.AvgPool1d(kernel_size=11, stride=1, padding=0)
151
152         self.layer7x7_1 = self._make_layer7(BasicBlock7x7, 64, layers[0], stride=2)
153         self.layer7x7_2 = self._make_layer7(BasicBlock7x7, 128, layers[1], stride=2)
154         self.layer7x7_3 = self._make_layer7(BasicBlock7x7, 256, layers[2], stride=2)
155         # self.layer7x7_4 = self._make_layer7(BasicBlock7x7, 512, layers[3], stride=2)
156         self.maxpool7 = nn.AvgPool1d(kernel_size=6, stride=1, padding=0)
157
158         self.downsample = nn.Conv1d(128, 1, 1) # added myself
159         # self.drop = nn.Dropout(p=0.2)
160         self.fc = nn.Linear(256 * 3, num_classes)
161
162         # todo: modify the initialization
163         # for m in self.modules():
164             # if isinstance(m, nn.Conv1d):
165                 # n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
166                 # m.weight.data.normal_(0, math.sqrt(2. / n))
167             # elif isinstance(m, nn.BatchNorm1d):
168                 # m.weight.data.fill_(1)
169                 # m.bias.data.zero_()
170
171     def _make_layer3(self, block, planes, blocks, stride=2):
172         downsample = None
173         if stride != 1 or self.inplanes3 != planes * block.expansion:
174             downsample = nn.Sequential(
175                 nn.Conv1d(self.inplanes3, planes * block.expansion,
176                         kernel_size=1, stride=stride, bias=False),
177                 nn.BatchNorm1d(planes * block.expansion),
178             )
179
180         layers = []
181         layers.append(block(self.inplanes3, planes, stride, downsample))
182         self.inplanes3 = planes * block.expansion
183         for i in range(1, blocks):

```

```

184         layers.append(block(self.inplanes3, planes))
185
186     return nn.Sequential(*layers)
187
188 def _make_layer5(self, block, planes, blocks, stride=2):
189     downsample = None
190     if stride != 1 or self.inplanes5 != planes * block.expansion:
191         downsample = nn.Sequential(
192             nn.Conv1d(self.inplanes5, planes * block.expansion,
193                     kernel_size=1, stride=stride, bias=False),
194             nn.BatchNorm1d(planes * block.expansion),
195         )
196
197     layers = []
198     layers.append(block(self.inplanes5, planes, stride, downsample))
199     self.inplanes5 = planes * block.expansion
200     for i in range(1, blocks):
201         layers.append(block(self.inplanes5, planes))
202
203     return nn.Sequential(*layers)
204
205 def _make_layer7(self, block, planes, blocks, stride=2):
206     downsample = None
207     if stride != 1 or self.inplanes7 != planes * block.expansion:
208         downsample = nn.Sequential(
209             nn.Conv1d(self.inplanes7, planes * block.expansion,
210                     kernel_size=1, stride=stride, bias=False),
211             nn.BatchNorm1d(planes * block.expansion),
212         )
213
214     layers = []
215     layers.append(block(self.inplanes7, planes, stride, downsample))
216     self.inplanes7 = planes * block.expansion
217     for i in range(1, blocks):
218         layers.append(block(self.inplanes7, planes))
219
220     return nn.Sequential(*layers)
221
222 def forward(self, x0):
223     x0 = self.conv1(x0)
224     x0 = self.bn1(x0)
225     x0 = self.relu(x0)
226     x0 = self.maxpool(x0)
227
228     x = self.layer3x3_1(x0)
229     x = self.layer3x3_2(x)
230     x = self.layer3x3_3(x)
231     # x = self.layer3x3_4(x)
232     x = self.maxpool3(x)
233
234     y = self.layer5x5_1(x0)
235     y = self.layer5x5_2(y)
236     y = self.layer5x5_3(y)
237     # y = self.layer5x5_4(y)
238     y = self.maxpool5(y)
239
240     z = self.layer7x7_1(x0)
241     z = self.layer7x7_2(z)
242     z = self.layer7x7_3(z)
243     # z = self.layer7x7_4(z)
244     z = self.maxpool7(z)
245
246     out = torch.cat([x, y, z], dim=1)
247
248     out = self.downsample(out.transpose(1, 2)) # added myself
249     # out = self.drop(out)
250     # print(out.shape)
251     # print(out.shape)
252     out1 = self.fc(out.squeeze(1))
253     # print(out1.shape)
254     return out1, out

```

B.7 jupyter_notebooks (folder)

B.7.1 jupyter_notebooks -> Beat detection.py (code)

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[11]:
5
6
7 import numpy as np

```

```

8 import matplotlib.pyplot as plt
9 import h5py as h5
10 import os
11 import scipy.io
12
13
14 # In[2]:
15
16
17 from sys import platform
18 print(platform)
19 if platform == "linux" or platform == "linux2": # linux
20     volume = '/media/julian/Volume/'
21 elif platform == "win32": # Windows.
22     volume = 'G:/'
23
24
25 # ### PTBXL
26
27 # In[3]:
28
29
30 p = os.path.join(volume, 'data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/generated/1000/
    normalized-labels/train/00006_hr.h5')
31 full_data = None
32 with h5.File(p, 'r') as f:
33     full_data = np.copy(f['data'])
34 data=full_data
35 print(data.shape)
36
37
38 # In[4]:
39
40
41 full_data = None
42 with h5.File('/media/julian/Volume/data/ECG/ptb-diagnostic-ecg-database-1.0.0/generated/normalized/test/patient007-
    s00781re.h5', 'r') as f:
43     full_data = np.copy(f['data'])
44 data=full_data
45 print(data.shape)
46
47
48 # ### Challenge
49
50 # In[20]:
51
52
53 data = np.array(scipy.io.loadmat('/media/julian/data/data/ECG/ptbxl_challenge/HR21380.mat')['val']).T
54 full_data = data
55 data.shape
56
57
58 # In[69]:
59
60
61 from matplotlib import pyplot as plt
62 from mpl_toolkits.mplot3d import Axes3D
63 import random
64 data = full_data - np.mean(data, axis=0)
65
66 fig = plt.figure(figsize=(20, 20))
67 ax = Axes3D(fig, auto_add_to_figure=False)
68
69 ax.scatter(np.arange(len(data)), data[:, 0], data[:, 1], s = 1)
70 fig.add_axes(ax)
71 plt.show()
72
73
74 # In[23]:
75
76
77 plt.figure(figsize=(20, 20))
78 plt.scatter(data[:, 0], data[:, 1], s=1)
79 plt.show()
80
81
82 # In[25]:
83
84
85 data = (data-np.mean(data, axis=0))/(np.max(data, axis=0) - np.mean(data, axis=0))
86 dists = np.sqrt(np.square(data[:,0])+np.square(data[:,1]))
87 print(dists)
88 medi = np.median(dists, axis=0)
89 print(medi)
90 mean = np.mean(dists, axis=0)
91 tresh = 0.5
92 x1 = data[:, 0]
93 x2 = data[:, 1]
94 plt.figure(figsize=(20, 20))
95 plt.scatter(x1[dists>tresh], x2[dists>tresh], s=1)

```

```

96 plt.scatter(x1[dists<=tresh], x2[dists<=tresh], s=1)
97 plt.show()
98
99
100 # In[6]:
101
102
103 np.min(data, axis=0)
104
105
106 # ## Calculating distance matrix between points
107
108 # In[27]:
109
110
111 from scipy.spatial import distance as d
112 #data = full_data[0:10000, :]
113 distv = d.pdist(data)
114 distm = d.squareform(distv)
115 print(data.shape)
116
117
118 # In[28]:
119
120
121 plt.figure(figsize=(20,20))
122 plt.imshow(distm)
123 plt.show()
124
125
126 # ## Summing columns of distance matrix
127 # * Sum data in each column up
128 # * Calculate mean and variance
129 # * Divide data into bins where points are more than {0, 1, 2, ...} variances away from mean.
130
131 # In[29]:
132
133
134 sums = np.sum(distm, axis=1)/len(distm)
135 sums_inv = np.exp(-sums)
136 m = np.mean(sums)
137 s = np.std(sums)
138 print('mean', m, 'standard deviation', s)
139 devs = []
140 for i in range(1, 100):
141     devs += [np.argwhere((np.abs(sums-m) >= (i-1)*s) & (np.abs(sums-m)<i*s))]
142     if len(devs[-1]) == 0:
143         devs.pop()
144         break
145 print('elements in bins:', *map(len, devs))
146
147
148 # ## Plot data (mean over all channels) with colors depending on bin
149
150 # In[30]:
151
152
153 plt.figure(figsize=(30, 10))
154 for i,d in enumerate(devs):
155     plt.scatter(np.arange(len(sums))[d], np.mean(data, axis=1)[d], label=str(i+1)+' std away from mean')
156 #plt.plot(sums_inv, label='inverted sums (exp(-sums))')
157 plt.legend()
158 plt.show()
159
160
161 # In[34]:
162
163
164 from scipy.spatial import distance as d
165 #data = full_data[0:10000, :]
166 distv = d.pdist(data)
167 distm = d.squareform(distv)
168 print(data.shape)
169
170
171 # In[35]:
172
173
174 epsilon = np.var(distm)
175 neighbours = []
176 for i in range(distm.shape[1]):
177     column = distm[:, i]
178     neighbours += [np.argwhere(column<=epsilon)]
179 neighbour_counts = np.array(list(map(len,neighbours)))
180
181
182 # In[38]:
183
184
185 plt.figure(figsize=(30, 10))

```

```

186 plt.scatter(np.arange(len(data)), np.mean(data, axis=1), c=neighbour_counts/max(neighbour_counts), label='datapoints
187     with yellow=most neighbours, dark blue=least neighbours')
188 #plt.plot(neighbour_counts/max(neighbour_counts), label='neighbour counts (scaled down)')
189 plt.legend()
190 plt.show()
191
192 # In[39]:
193
194
195 plt.figure(figsize=(30, 10))
196 plt.plot(np.sum(distm<=s, axis=1))
197
198
199 # In[40]:
200
201
202 def _calc_and_plot(full_data, interval):
203     data = full_data[interval]
204     sums = np.sum(distm, axis=1)/len(distm)
205     sums_inv = np.exp(-sums)
206     m = np.mean(sums)
207     s = np.std(sums)
208     print('mean', m, 'standard deviation', s)
209     devs = []
210     for i in range(1, 100):
211         devs += [np.argwhere((np.abs(sums-m) >= (i-1)*s) & (np.abs(sums-m)<i*s))]
212         if len(devs[-1]) == 0:
213             devs.pop()
214             break
215     print('elements in bins:', *map(len, devs))
216     plt.figure(figsize=(30, 10))
217     plt.title("Outlier detection" + str(interval))
218     for i,d in enumerate(devs):
219         plt.scatter(np.arange(len(sums))[d], data[d, 1], label=str(i+1)+' std away from mean')
220     plt.plot(sums_inv, label='inverted sums')
221     plt.legend()
222     plt.show()
223
224
225 # In[41]:
226
227
228 def calc_and_plot(full_data):
229     base_size = 10000
230     n_windows = (len(full_data)-3)/base_size
231     window_size = int(base_size*(n_windows-int(n_windows)))
232     for i in range(int(n_windows)):
233         _calc_and_plot(full_data, slice(i*window_size, (i+1)*window_size))
234
235
236 # In[48]:
237
238
239 _calc_and_plot(full_data, slice(0, len(full_data)))
240
241
242 # In[49]:
243
244
245 def outlier_detection_nearest_epsilon(sorted_cols, k):
246     l = []
247     for col in sorted_cols:
248         nei = col[0:k]
249         kappa = nei[-1]
250         min_dist = nei[0]
251         gamma = np.mean(nei)
252         var = np.var(nei)
253         l.append([kappa, min_dist, gamma, var])
254     return map(np.array, zip(*l))
255
256
257 # In[53]:
258
259
260 def plot_metrics(data, kappas, min_dists, gammas, var, save=None):
261     x = np.arange(len(data))
262     fig, axs = plt.subplots(data.shape[1]+1, 1, figsize=(len(data)/4/100, data.shape[1]*300/4/100), dpi=100, sharex=True)
263     for i in range(len(axs)-1):
264         axs[i].scatter(x, data[:, i], c=kappas, s=0.5)
265         axs[-1].plot(kappas, label='k')
266         axs[-1].plot(min_dists, label='min')
267         axs[-1].plot(gammas, label='g')
268         axs[-1].plot(var, label='var')
269     if save:
270         plt.savefig(save, dpi=100)
271     else:
272         plt.legend()
273         plt.show()

```

```

274     plt.close()
275
276
277 # In[62]:
278
279 kappas, min_dists, gammas, var = map(np.array, outlier_detection_nearest_epsilon(distm, 10000))
280
281
282 # In[63]:
283
284
285 plot_metrics(data, kappas, min_dists, gammas, var, save=None)
286
287
288 # In[ ]:
289
290
291 sorted_cols = np.sort(distm, axis=1)
292 for i in range(1, len(distm)//2, 2):
293     kappas, min_dists, gammas, var = outlier_detection_nearest_epsilon(sorted_cols, k=i)
294     plot_metrics(data, kappas, min_dists, gammas, var, save=None)
295     print('Completed {} of {}'.format(i, len(distm)//2), flush=True)
296
297
298 # In[ ]:
299

```

B.7.2 jupyter_notebooks -> CPC Loss Implementation Test.py (code)

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[6]:
5
6
7 import colorcet as cc
8 import matplotlib.pyplot as plt
9 import seaborn as sns
10 from sklearn.datasets import make_blobs
11
12
13 blobs, labels = make_blobs(n_samples=1000, centers=67, center_box=(-100, 100))
14 palette = sns.color_palette(cc.glasbey, n_colors=67)
15
16 sns.scatterplot(x=blobs[:,0], y=blobs[:, 1], hue=labels, data=blobs, palette=palette)
17 plt.legend(ncol=5, bbox_to_anchor=(1, 1))
18 plt.show()
19
20
21 # In[1]:
22
23
24 import torch
25 from torch import nn
26 import numpy as np
27
28
29 class CPC(nn.Module):
30
31     def __init__(self, encoder, autoregressive, predictor, timesteps_in, timesteps_out, latent_size, timesteps_ignore
32                  =0,
33                  normalize_latents=False, verbose=False, sampling_mode='same'):
34         super().__init__()
35         self.encoder = encoder
36         self.autoregressive = autoregressive
37         self.predictor = predictor
38         self.lsoftmax = nn.LogSoftmax(dim=-1)
39         self.timesteps_in = timesteps_in
40         self.timesteps_out = timesteps_out
41         self.timesteps_ignore = timesteps_ignore
42         self.latent_size = latent_size
43         self.normalize_latents = normalize_latents
44         self.verbose = verbose
45         self.cpc_train_mode = True
46         if sampling_mode in ['all', 'same', 'random', 'future']:
47             self.sampling_mode = sampling_mode
48         else:
49             self.sampling_mode = 'same'
50
51     def forward(self, X, y=None, hidden=None):
52         if self.verbose: print('input', X.shape)
53         if len(X.shape) == 4: # uses cpc or challenge dataset
54             batch, windows, channels, length = X.shape
55             X = X.transpose(1, 2).reshape((batch, channels, -1))
56         elif len(X.shape) == 3: # uses baseline dataset, shape: (batch, length, channels)
57             X = X.transpose(1, 2)

```

```

57     if self.verbose: print("input reshaped to:", X.shape)
58     batch, channels, length = X.shape # assume baseline dataset shape
59     encoded_x = self.encoder(X) # encode whole data: shape: (batch, latent_size, length->out_length)
60     encoded_x = encoded_x.permute(2, 0, 1) # shape outlength, batch, latent_size
61     # if self.normalize_latents:
62     #     encoded_x = encoded_x / torch.norm(encoded_x, p=2, dim=-1, keepdim=True)
63     if self.verbose: print('encoder_x', encoded_x.shape)
64     encoded_x_steps, _, _ = encoded_x.shape
65     if hidden is None:
66         hidden = self.autoregressive.init_hidden(batch_size=batch)
67     context, hidden = self.autoregressive(encoded_x, hidden)
68     if self.verbose: print('context', context.shape)
69
70     if not self.cpc_train_mode:
71         return encoded_x, context[-1, :, :], hidden
72     # offset = np.random.choice(encoded_x_steps - self.timesteps_out - self.timesteps_ignore, 10, replace=False)
73
74     # Draw 10 randoms
75     loss = torch.tensor([0.0])
76     correct = torch.tensor([0.0])
77     if encoded_x_steps - self.timesteps_out - self.timesteps_ignore - self.timesteps_in < 1:
78         print(
79             'Not enough encoded values for CPC training. Lower timesteps_in/timesteps_out/timesteps_ignore or
change window_size + encoder')
80         print(
81             f'Encoded {encoded_x_steps} steps but need at least {(self.timesteps_out + self.timesteps_ignore +
self.timesteps_in)!}')
82         return loss, correct, hidden
83
84
85     if self.sampling_mode == 'all':
86         encoded_latent = encoded_x.transpose(0, 1) # output shape: Batch, latents, latent_size
87         if self.verbose: print('encoded after T', encoded_latent.shape)
88         t = encoded_x_steps - self.timesteps_out - self.timesteps_ignore
89
90         for k in range(self.timesteps_ignore, self.timesteps_ignore + self.timesteps_out):
91             pred_latent = self.predictor(context[t - 1, :, :], k).squeeze(0)
92             print(k, encoded_latent.shape, pred_latent.shape)
93             # shape is batch, n_latents, latent_size
94             softmax = self.lsoftmax(torch.matmul(encoded_latent, pred_latent.T)).reshape(batch,
95                                         encoded_x_steps * batch))
96
97         # reshape for argmax to B, latents*B
98         argmaxs = torch.argmax(softmax, dim=1)
99         correct += torch.sum(
100             argmaxs == torch.arange(batch) + t * batch) # Correct index will be at offset t
101         loss += torch.sum(
102             torch.diag(softmax.reshape(batch, encoded_x_steps, batch)[:, t + self.timesteps_ignore + k, :]))
103
104         loss /= (batch * self.timesteps_out) * -1.0 # * pred_latent.shape[0]
105         accuracy = correct.true_divide(batch * self.timesteps_out) # * pred_latent.shape[0]
106         #print(loss)
107         return accuracy, loss, hidden
108
109     def freeze_layers(self):
110         for param in self.parameters():
111             param.requires_grad = False
112
113     def unfreeze_layers(self):
114         for param in self.parameters():
115             param.requires_grad = True
116
117     import torch
118     from torch import nn
119
120
121     class AutoRegressor(nn.Module):
122         def __init__(self, n_latents, hidden_size, layers=1):
123             super(AutoRegressor, self).__init__()
124             self.n_latents = n_latents
125             self.hidden_size = hidden_size
126             self.layers = layers
127             self.gru = nn.GRU(input_size=self.n_latents, hidden_size=self.hidden_size, num_layers=self.layers)
128
129             def _weights_init(m):
130                 for layer_p in self.gru._all_weights:
131                     for p in layer_p:
132                         if 'weight' in p:
133                             nn.init.kaiming_normal_(self.gru.__getattr__(p), mode='fan_out', nonlinearity='relu')
134
135             self.apply(_weights_init)
136
137         def init_hidden(self, batch_size, use_gpu=False):
138             if use_gpu:
139                 return torch.zeros(self.layers, batch_size, self.hidden_size).cuda()
140             else:
141                 return torch.zeros(self.layers, batch_size, self.hidden_size).cpu()
142

```

```

143     def forward(self, x, hidden):
144         # Input is (seq, batch, latents) maybe (13, 8, 128)
145         # print('regressor input shape:', x.shape, hidden.shape)
146         x, hidden = self.gru(x, hidden)
147         # print('regressor output shape:', x.shape, hidden.shape)
148         return x, hidden
149
150 class Predictor(nn.Module):
151     def __init__(self, context_size, latent_size, timesteps: int):
152         super().__init__()
153         self.linears = nn.ModuleList(
154             [nn.Linear(context_size, latent_size)] * timesteps
155         )
156
157     def _weights_init(m):
158         if isinstance(m, nn.Linear):
159             nn.init.kaiming_normal_(m.weight, mode='fan_in', nonlinearity='relu')
160
161     self.apply(_weights_init)
162
163     def forward(self, x, timestep: int):
164         # print('Predictor input shape:', x.shape)
165
166         if timestep is None: # output shape = (timesteps, batch, latent_size)
167             preds = []
168             for i, l in enumerate(self.linears):
169                 preds.append(self.linears[i](x))
170             return torch.stack(preds, dim=0)
171
172         else: # output shape = (batch, latent_size)
173             return self.linears[timestep](x)
174
175
176 # In[3]:
177
178
179 e = torch.randn(64, 128, 26)
180 enc = lambda x: e
181 auto = AutoRegressor(128, 256, 1)
182 predi = lambda x,k: e.permute(2, 0, 1)[12+k]
183
184
185 # In[4]:
186
187
188 cpc = CPC(enc, auto, predi, 12, 12, 128, 0, sampling_mode='all', verbose=True)
189
190
191 # In[5]:
192
193
194 X = torch.randn(64, 4500, 12)
195 cpc(X)
196
197
198 # In[6]:
199
200
201 torch.ones(3, 2, 8)
202
203
204 # In[4]:
205
206
207 import torch
208 from torch import nn
209 import numpy as np
210
211 class AutoRegressor(nn.Module):
212     def __init__(self, n_latents, hidden_size, layers=1):
213         super(AutoRegressor, self).__init__()
214         self.n_latents = n_latents
215         self.hidden_size = hidden_size
216         self.layers = layers
217         self.gru = nn.GRU(input_size=self.n_latents, hidden_size=self.hidden_size, num_layers=self.layers)
218
219     def _weights_init(m):
220         for layer_p in self.gru._all_weights:
221             for p in layer_p:
222                 if 'weight' in p:
223                     nn.init.kaiming_normal_(self.gru.__getattr__(p), mode='fan_out', nonlinearity='relu')
224
225     self.apply(_weights_init)
226
227     def init_hidden(self, batch_size, use_gpu=False):
228         if use_gpu:
229             return torch.zeros(self.layers, batch_size, self.hidden_size).cuda()
230         else:
231             return torch.zeros(self.layers, batch_size, self.hidden_size).cpu()
232

```

```

233     def forward(self, x, hidden):
234         # Input is (seq, batch, latents) maybe (13, 8, 128)
235         # print('regressor input shape:', x.shape, hidden.shape)
236         x, hidden = self.gru(x, hidden)
237         # print('regressor output shape:', x.shape, hidden.shape)
238         return x, hidden
239
240 class Predictor(nn.Module):
241     def __init__(self, context_size, latent_size, timesteps: int):
242         super().__init__()
243         self.linears = nn.ModuleList(
244             [nn.Linear(context_size, latent_size)] * timesteps
245         )
246
247     def _weights_init(m):
248         if isinstance(m, nn.Linear):
249             nn.init.kaiming_normal_(m.weight, mode='fan_in', nonlinearity='relu')
250
251     self.apply(_weights_init)
252
253     def forward(self, x, timestep: int):
254         # print('Predictor input shape:', x.shape)
255
256         if timestep is None: # output shape = (timesteps, batch, latent_size)
257             preds = []
258             for i, l in enumerate(self.linears):
259                 preds.append(self.linears[i](x))
260             return torch.stack(preds, dim=0)
261
262         else: # output shape = (batch, latent_size)
263             return self.linears[timestep](x)
264
265
266
267 class Encoder(nn.Module):
268     def __init__(self, channels, latent_size):
269         super(Encoder, self).__init__()
270         filters = [10, 8, 4, 4, 4] # See https://arxiv.org/pdf/1807.03748.pdf#Audio
271         strides = [5, 4, 2, 2, 2]
272         n_channels = [channels] + [latent_size] * len(filters)
273         # self.batch_norm = nn.BatchNorm1d(n_channels[0]) #not used in paper?
274         self.convolutions = nn.Sequential(
275             *[e for t in [
276                 (nn.Conv1d(in_channels=n_channels[i], out_channels=n_channels[i + 1], kernel_size=filters[i],
277                             stride=strides[i], padding=0),
278                             # nn.BatchNorm1d(n_channels[i + 1]),
279                             (nn.ReLU())),
280                             ) for i in range(len(filters))] for e in t]
281         )
282
283     def _weights_init(m):
284         if isinstance(m, nn.Conv1d):
285             nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
286
287     self.apply(_weights_init)
288
289     def forward(self, x):
290         # Input has shape (batches, channels, window_size)
291         # print('Encoder input shape:', x.shape)
292         # x = self.batch_norm(x)
293         x = self.convolutions(x)
294         # print('Encoder output shape:', x.shape)
295
296         return x
297
298 class CPC(nn.Module):
299
300     def __init__(self, encoder, autoregressive, predictor, timesteps_in, timesteps_out, latent_size, timesteps_ignore
301 =0,
302             normalize_latents=False, verbose=False, sampling_mode='same'):
303         super().__init__()
304         self.encoder = encoder
305         self.autoregressive = autoregressive
306         self.predictor = predictor
307         self.lsoftmax = nn.LogSoftmax(dim=0)
308         self.timesteps_in = timesteps_in
309         self.timesteps_out = timesteps_out
310         self.timesteps_ignore = timesteps_ignore
311         self.latent_size = latent_size
312         self.normalize_latents = normalize_latents
313         self.verbose = verbose
314         self.cpc_train_mode = True
315         if sampling_mode in ['all', 'same', 'random', 'future']:
316             self.sampling_mode = sampling_mode
317         else:
318             self.sampling_mode = 'same'
319
320     def forward(self, x, y=None, hidden=None):
321         if self.verbose: print('input received', x.shape)
322         if len(x.shape) == 4: # uses cpc or challenge dataset

```

```

322         batch, windows, channels, length = X.shape
323         X = X.transpose(1, 2).reshape((batch, channels, -1))
324     elif len(X.shape) == 3: # uses baseline dataset, shape: (batch, length, channels)
325         X = X.transpose(1, 2)
326     if self.verbose: print("input reshaped to:", X.shape)
327     batch, channels, length = X.shape # assume baseline dataset shape
328     encoded_x = self.encoder(X) # encode whole data: shape: (batch, latent_size, length->out_length)
329     _, _, encoded_x_steps = encoded_x.shape
330     if self.verbose: print('encoded x', encoded_x.shape)
331     encoded_x = encoded_x.permute(2, 0, 1) #reshape to timesteps, batch, latent_size
332     if self.verbose: print('encoded x (transposed)', encoded_x.shape)
333     if hidden is None:
334         hidden = self.autoregressive.init_hidden(batch_size=batch)
335     context, hidden = self.autoregressive(encoded_x, hidden)
336     if self.verbose: print('context', context.shape)
337
338
339     if not self.cpc_train_mode:
340         return encoded_x, context[-1, :, :], hidden
341     # offset = np.random.choice(encoded_x_steps - self.timesteps_out - self.timesteps_ignore, 10, replace=False)
# Draw 10 randoms
342
343     loss = torch.tensor([0.0])
344     correct = torch.tensor([0.0])
345     if encoded_x_steps - self.timesteps_out - self.timesteps_ignore - self.timesteps_in < 1:
346         print(
347             'Not enough encoded values for CPC training. Lower timesteps_in/timesteps_out/timesteps_ignore or
change window_size + encoder')
348         print(
349             f'Encoded {encoded_x_steps} steps but need at least {(self.timesteps_out + self.timesteps_ignore +
self.timesteps_in)})')
350     return loss, correct, hidden
351
352
353     if self.sampling_mode == 'all':
354         t = encoded_x_steps - self.timesteps_out - self.timesteps_ignore
355         current_context = context[t-1, :, :]
356         enc_resh = encoded_x.reshape(encoded_x_steps*batch, -1)
357
358         for k in range(self.timesteps_out):
359             pred_latent = self.predictor(current_context, k)
360             if self.verbose: print("pred latent shape", pred_latent.shape)
361
362             sim = enc_resh@pred_latent.T # sim[i, j] = scalar prod between li and pred j
363             soft = self.lsoftmax(sim) #Where is the highest similarity? (max in each i) == i
364             soft_resh = soft.reshape(encoded_x_steps, batch, -1)
365             print(soft_resh)
366             loss += torch.diag(soft_resh[t+k]).sum() #higher = better
367             correct += (soft.max(dim=0).indices == torch.arange(batch, device=soft.device)+batch*(t+k)).sum()
368
369             loss = -loss/(self.timesteps_out*batch)
370             accuracy = correct.true_divide(batch * self.timesteps_out) # * pred_latent.shape[0]
371     return accuracy, loss, hidden
372
373
374
375     def freeze_layers(self):
376         for param in self.parameters():
377             param.requires_grad = False
378
379     def unfreeze_layers(self):
380         for param in self.parameters():
381             param.requires_grad = True
382
383
384 # In[ ]:
385
386
387
388
389
390 # In[5]:
391
392
393 enc = Encoder(12, 128)
394 auto = AutoRegressor(128, 256, 1)
395 predi = Predictor(256, 128, 10)
396 cpc = CPC(enc, auto, predi, 11, 10, 128, 0, sampling_mode='all', verbose=True)
397
398
399 # In[6]:
400
401
402 cpc(X)
403
404
405 # In[7]:
406
407
408 a = torch.randn(9, 4, 128)

```

```

409 b = torch.randn(3, 128)
410
411 # In[8]:
412
413
414 l = a
415 p = b
416 #l = l.transpose(0, 1)
417 p = l[4]
418 l = l.reshape(9*4, 128)
419 m = l@p.T
420 m.shape
421
422
423 # In[9]:
424
425
426 s = nn.Softmax(dim=0)(l@p.T)
427 s
428
429
430 # In[10]:
431
432
433 s.max(dim=0).indices
434
435
436 # In[11]:
437
438
439 s[3]
440
441
442 # In[12]:
443
444
445 s = s.reshape(9, 4, -1)
446 s
447
448
449 # In[13]:
450
451
452 s[4]
453
454
455 # In[28]:
456
457
458 torch.diag(s[4]).sum()
459
460
461 # In[31]:
462
463
464
465 enc = Encoder(12, 128)
466 auto = AutoRegressor(128, 256, 1)
467 predi = Predictor(256, 128, 10)
468 cpc = CPC(enc, auto, predi, 11, 10, 128, 0, sampling_mode='all', verbose=True)
469
470
471 # In[32]:
472
473
474 from scipy.io import arff
475
476
477 # In[59]:
478
479
480 batch = 8
481 datar, meta = arff.loadarff('/home/julian/Downloads/Multi-Download/EEG Eye State.arff')
482
483
484 # In[64]:
485
486
487 data = np.array([d[0:-1] for d in datar.tolist()])
488 labels = np.array([d[-1] for d in datar.tolist()])
489
490
491 # In[66]:
492
493
494 data.shape
495
496
497 # In[18]:
498

```

```

499
500 import os
501 p='/home/julian/Downloads/Github/contrastive-predictive-coding/models/20_12_21-13-40-test | (32x)cpc'
502
503
504 # In[17]:
505
506
507 for root, dirs, files in os.walk(p):
508     for d in dirs:
509         if d.startswith('architectures_cpc.cpc_combined.CPCCombined'):
510             n = int(d.split('/')[-1][0].replace('architectures_cpc.cpc_combined.CPCCombined', '')) 
511             if n >= 0 and n <= 7:
512                 rep = 'architectures_cpc.cpc_combined.CPCCombined'+str(n+8) + '/' + d.split('/', 1)[1]
513                 os.rename(os.path.join(root,d), os.path.join(root, rep))
514     for d in files:
515         if d.startswith('model-architectures_cpc.cpc_combined.CPCCombined'):
516             n = int(d.split('/')[-1][0].replace('model-architectures_cpc.cpc_combined.CPCCombined', '')) 
517             if n >= 0 and n <= 7:
518                 rep = 'model-architectures_cpc.cpc_combined.CPCCombined'+str(n+8) + '/' + d.split('/', 1)[1]
519                 os.rename(os.path.join(root,d), os.path.join(root, rep))
520
521
522
523
524 # In[20]:
525
526
527 for root, dirs, files in os.walk(p):
528     for d in dirs:
529         if d.startswith('architectures_cpc.cpc_combined.CPCCombined'):
530             n = int(d.split('/')[-1][0].replace('architectures_cpc.cpc_combined.CPCCombined', '')) 
531             if n >= 8:
532                 rep = 'architectures_cpc.cpc_combined.CPCCombined'+str(n-8) + '/' + d.split('/', 1)[1]
533                 os.rename(os.path.join(root,d), os.path.join(root, rep))
534     for d in files:
535         if d.startswith('model-architectures_cpc.cpc_combined.CPCCombined'):
536             n = int(d.split('/')[-1][0].replace('model-architectures_cpc.cpc_combined.CPCCombined', '')) 
537             if n >= 8:
538                 rep = 'model-architectures_cpc.cpc_combined.CPCCombined'+str(n-8) + '/' + d.split('/', 1)[1]
539                 os.rename(os.path.join(root,d), os.path.join(root, rep))
540
541
542 # In[1]:
543
544
545 import torch
546 import numpy as np
547 from torch import nn
548 import torch.nn.functional as F
549
550
551 # # All latents
552
553 # In[138]:
554
555
556 t_out = 2
557 t_in = 3
558 t_ignore = 1
559 batch = 4
560 l_size = 128
561 x_steps = 16
562 k = 1
563 t = np.random.randint(t_in, x_steps-t_out-t_ignore)
564 enc = torch.abs(torch.randn(16, batch, l_size))*-1
565 # for i in range(batch):
566 #     enc[i, -t_out, :] = i
567 pred = torch.zeros(batch, l_size)
568 for i in range(batch):
569     pred[i, :] = enc[t+k+t_ignore, i, :]
570 t
571
572
573 # In[139]:
574
575
576 pred
577
578
579 # In[140]:
580
581
582 enc
583
584
585 # In[141]:
586
587
588 prod = enc@pred.T

```

```

589 print(prod.shape)
590 prod
591
592
593 # In[142]:
594
595
596 s = prod.reshape((x_steps)*batch, batch)
597 s
598
599
600 # In[143]:
601
602
603 so = F.softmax(s, dim=0)
604 so
605
606
607 # In[144]:
608
609
610 sor = so.reshape(x_steps, batch, batch)
611 sor
612
613
614 # In[146]:
615
616
617 sor[t+k+t_ignore]
618
619
620 # In[147]:
621
622
623 torch.diag(sor[t+k+t_ignore])
624
625
626 # In[148]:
627
628
629 so.max(dim=0).indices == torch.arange(batch)+(t+k+t_ignore)*batch
630
631
632 # # Future latents
633
634 # In[157]:
635
636
637 t_out = 2
638 t_in = 3
639 t_ignore = 1
640 batch = 4
641 l_size = 128
642 x_steps = 16
643 k = 1
644 t = np.random.randint(t_in, x_steps-t_out-t_ignore)
645 enc = torch.abs(torch.randn(16, batch, l_size))*-1
646 # for i in range(batch):
647 #     enc[i, -t_out, :] = i
648 pred = torch.zeros(batch, l_size)
649 for i in range(batch):
650     pred[i, :] = enc[t+k+t_ignore, i, :]
651 t
652
653
654 # In[158]:
655
656
657 prod = enc[t:, :, :] @ pred.T
658 prod
659
660
661 # In[159]:
662
663
664 prodr = prod.reshape((x_steps-t)*batch, batch)
665 prodr
666
667
668 # In[160]:
669
670
671 soft = F.softmax(prodr, dim=0)
672 soft
673
674
675 # In[161]:
676
677
678 soft_resh = soft.reshape((x_steps-t), batch, batch)

```

```
679 soft_resh
680
681
682 # In[162]:
683
684
685 soft_resh[k+t_ignore]
686
687
688 # # Alpha
689 #
690
691 # In[11]:
692
693
694 t_out = 2
695 t_in = 3
696 t_ignore = 1
697 batch = 4
698 l_size = 128
699 x_steps = 16
700 k = 1
701 t = np.random.randint(t_in, x_steps-t_out-t_ignore)
702 enc = torch.abs(torch.randn(16, batch, l_size))*-1
703 # for i in range(batch):
704 #     enc[i, -t_out, :] = i
705 pred = torch.zeros(batch, l_size)
706 for i in range(batch):
707     pred[i, :] = enc[t+k+t_ignore, i, :]
708 alpha = 0.05
709 m = batch
710 t
711
712
713 # In[26]:
714
715
716 prod = enc @ pred.T
717 prod = prod + np.log((m-alpha)/(m-1))
718 prod[t+k+t_ignore] += torch.eye(prod[t+k+t_ignore].shape[0])*(np.log(alpha)-np.log((m-alpha)/(m-1)))
719 prod
720
721
722 # In[29]:
723
724
725 prodr = prod.reshape((x_steps)*batch, batch)
726 prodr
727
728
729 # In[30]:
730
731
732 soft = F.softmax(prodr, dim=0)
733 soft
734
735
736 # In[31]:
737
738
739 soft_resh = soft.reshape(x_steps, batch, batch)
740 soft_resh
741
742
743 # In[33]:
744
745
746 (m/alpha)*soft_resh
747
748
749 # In[39]:
750
751
752 loss = torch.Tensor([1.], require_grad=True).cuda()
753
754
755 # In[36]:
756
757
758 get_ipython().run_line_magic('timeit', 'loss.item()')
759
760
761 # In[37]:
762
763
764 get_ipython().run_line_magic('timeit', 'loss.cpu()')
765
766
767 # In[66]:
768
```

```

769 t_out = 2
770 t_in = 3
772 t_ignore = 1
773 batch = 4
774 l_size = 128
775 x_steps = 16
776 k = 1
777 t = np.random.randint(t_in, x_steps-t_out-t_ignore)
778 enc = torch.abs(torch.randn(16, batch, l_size))*-1
779 # for i in range(batch):
780 #     enc[i, -t_out, :] = i
781 pred = torch.zeros(batch, l_size)
782 for i in range(batch):
783     pred[i, :] = enc[t+k+t_ignore, i, :]
784 t
785
786
787 # In[67]:
788
789
790 enc
791
792
793 # In[68]:
794
795
796 prod = enc @ pred.T
797 prod
798
799
800 # In[73]:
801
802
803 cross_entropy_with_logits = torch.nn.CrossEntropyLoss(reduction='none')
804
805
806 # In[77]:
807
808
809 prod = enc @ pred.T
810 logits = prod.reshape(x_steps*batch, batch).T
811
812
813 # In[78]:
814
815
816 cross_entropy_with_logits(logits , torch.arange(batch)+batch*(t+k+t_ignore))
817
818
819 # In[79]:
820
821
822 torch.arange(batch)+batch*(t+k+t_ignore)
823
824
825 # In[76]:
826
827
828 [logits[i, (torch.arange(batch)+batch*(t+k+t_ignore))[i]] for i in range(batch)]
829
830
831 # In[41]:
832
833
834 def log_softmax(x, dim=0):
835
836     d = x - x.max(dim=dim, keepdim=True).values
837     return d - torch.log(torch.exp(d).sum(dim=dim))
838
839
840 # In[17]:
841
842
843 import re
844 N = []
845 for t in table.split('\n'):
846     name, mic, mac = t.split("&")
847     name = name.replace("\textbf{", "").replace("}", "").strip()
848     mic = float(mic.strip())
849     mac = float(mac.replace('\\\\', '').strip())
850     N.append([name, mic, mac])
851
852
853 # In[26]:
854
855
856 best_macro = max(N, key=lambda t: t[2]+t[1]/10000)
857 best_macro
858

```

```

859
860 # In[27]:
861
862
863 best_micro = max(N, key=lambda t: t[1]+t[2]/10000)
864 best_micro
865
866
867 # In[29]:
868
869
870 import pandas as pd
871
872
873 # In[30]:
874
875
876 pd.DataFrame(sorted(N, key=lambda t: -(t[2]+t[1]/10000)))
877
878
879 # In[31]:
880
881
882 pd.DataFrame(sorted(N, key=lambda t: -(t[1]+t[2]/10000)))
883
884
885 # In[ ]:
```

B.7.3 jupyter_notebooks -> CPC Loss.py (code)

```

1#!/usr/bin/env python
2# coding: utf-8
3
4# In[1]:
5
6
7import numpy as np
8import math
9import matplotlib.pyplot as plt
10import pickle
11import torch
12import os
13
14
15# In[6]:
16
17
18def load_many_pickles(paths, multi=False):
19    lists = []
20    for path in paths:
21        loss_path = os.path.join(path, 'losses.pkl')
22        acc_path = os.path.join(path, 'accuracies.pkl')
23        lists.append(_load_pickles(loss_path, acc_path, multi))
24    return [np.concatenate(a, axis=0) for a in zip(*lists)]
25
26def load_pickles(path, multi=False):
27    loss_path = os.path.join(path, 'losses.pkl')
28    acc_path = os.path.join(path, 'accuracies.pkl')
29    return _load_pickles(loss_path, acc_path, multi)
30
31def _load_pickles(loss_path, acc_path, multi):
32    with open(loss_path, 'rb') as f:
33        loaded = pickle.load(f)
34    with open(acc_path, 'rb') as f:
35        loaded2 = pickle.load(f)
36    if multi: #potential different handling for multiple losses accuracies
37        train_losses = loaded[0]
38        val_losses = loaded[1]
39        train_acc = loaded2[0]
40        val_acc = loaded2[1]
41    else:
42        train_losses = [l for l in loaded[0]]
43        val_losses = [l for l in loaded[1]]
44        train_acc = [l for l in loaded2[0]]
45        val_acc = [l for l in loaded2[1]]
46    return train_losses, val_losses, train_acc, val_acc
47
48def _load_pickles_new(path):
49    pl = []
50    for pick_path in glob.glob(os.path.join(path, '*/*.pickle')):
51        with open(pick_path, 'rb') as f:
52            loaded = pickle.load(f)
53            parsed = defaultdict(list)
54            for ep, vd in loaded.items():
55                for name, value in vd.items():
56                    if name != 'elapsed_time':
```

```

57             parsed[name].append(np.mean(value))
58     pl.append(parsed)
59     return pl
60
61
62 def plot(train_losses, val_losses, train_acc, val_acc, title=None):
63     plt.title(title)
64     plt.plot(train_losses, label='train loss')
65     plt.plot(val_losses, label='val loss')
66     plt.annotate('train min:%.4f'%np.min(train_losses),
67                 xy=(np.argmin(train_losses), np.min(train_losses)), xycoords='data',
68                 xytext=(-90, 70), textcoords='offset points', fontsize=8,
69                 arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
70     plt.annotate('val min:%.4f'%np.min(val_losses),
71                 xy=(np.argmin(val_losses), np.min(val_losses)), xycoords='data',
72                 xytext=(-90, 40), textcoords='offset points', fontsize=8,
73                 arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
74     plt.xlabel("epoch")
75     plt.ylabel('loss')
76     plt.legend()
77     plt.show()
78     plt.title(title)
79     plt.plot(train_acc, label='train acc')
80     plt.plot(val_acc, label='val acc')
81     plt.hlines(np.max(val_acc), 0, len(train_acc), linestyle="dashed", color='black', linewidth=0.5)
82     plt.annotate('train max:%.4f'%np.max(train_acc),
83                 xy=(np.argmax(train_acc), np.max(train_acc)), xycoords='data',
84                 xytext=(-90, -30), textcoords='offset points', fontsize=8,
85                 arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
86     plt.annotate('val max:%.4f'%np.max(val_acc),
87                 xy=(np.argmax(val_acc), np.max(val_acc)), xycoords='data',
88                 xytext=(-90, -60), textcoords='offset points', fontsize=8,
89                 arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
90     plt.xlabel("epoch")
91     plt.ylabel('accuracy')
92     plt.legend()
93     plt.show()
94
95 def plot_new(metric_dict, title=None):
96     train_losses = metric_dict['trainloss']
97     val_losses = metric_dict['valloss']
98     train_acc = metric_dict['trainacc']
99     val_acc = metric_dict['valacc']
100    plt.title(title)
101    plt.plot(train_losses, label='train loss')
102    plt.plot(val_losses, label='val loss')
103    plt.annotate('train min:%.4f'%np.min(train_losses),
104                 xy=(np.argmin(train_losses), np.min(train_losses)), xycoords='data',
105                 xytext=(-90, 70), textcoords='offset points', fontsize=8,
106                 arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
107    plt.annotate('val min:%.4f'%np.min(val_losses),
108                 xy=(np.argmin(val_losses), np.min(val_losses)), xycoords='data',
109                 xytext=(-90, 40), textcoords='offset points', fontsize=8,
110                 arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
111    plt.xlabel("epoch")
112    plt.ylabel('loss')
113    plt.legend()
114    plt.show()
115    plt.title(title)
116    plt.plot(train_acc, label='train acc')
117    plt.plot(val_acc, label='val acc')
118    plt.hlines(np.max(val_acc), 0, len(train_acc), linestyle="dashed", color='black', linewidth=0.5)
119    plt.annotate('train max:%.4f'%np.max(train_acc),
120                 xy=(np.argmax(train_acc), np.max(train_acc)), xycoords='data',
121                 xytext=(-90, -30), textcoords='offset points', fontsize=8,
122                 arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
123    plt.annotate('val max:%.4f'%np.max(val_acc),
124                 xy=(np.argmax(val_acc), np.max(val_acc)), xycoords='data',
125                 xytext=(-90, -60), textcoords='offset points', fontsize=8,
126                 arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
127    plt.xlabel("epoch")
128    plt.ylabel('accuracy')
129    plt.legend()
130    plt.show()
131
132 def plot_all_models_in_folder(path):
133     for pick in _load_pickles_new(path):
134         plot_new(pick, title='')
135     #     trl, tra, val, vaa = pick.values()
136     #     plot(trl, tra, val, vaa, title='')
137
138
139
140
141 # In[19]:
142
143
144 t = {}
145 for i in range(1000000):
146     t['/home/julian/Downloads/Github/contrastive-predictive-coding/models/13_12_21-16-train|cpc'+str(i)]=i*2

```

```

147 def test():
148     if '/home/julian/Downloads/Github/contrastive-predictive-coding/models/13_12_21-16-train|cpc99999' in t:
149         return t['/home/julian/Downloads/Github/contrastive-predictive-coding/models/13_12_21-16-train|cpc99999']
150     else:
151         return False
152
153
154 # In[20]:
155
156
157 get_ipython().run_line_magic('timeit', 'test()')
158
159
160 # In[7]:
161
162
163 import glob
164 from collections import defaultdict
165
166
167 # In[11]:
168
169
170 path = '/home/julian/Downloads/Github/contrastive-predictive-coding/models/13_12_21-11-train|(4x)cpc'
171 _load_pickles_new(path)[0].keys()
172
173
174 # In[12]:
175
176
177 plot_all_models_in_folder(path)
178
179
180 # ## Losses for multi target
181
182 # In[3]:
183
184
185 #Downstream on cpc(baseline dataset) batch_size 16->64 windowslength 236
186 """
187 latents used as prediction method (flatten + double linear no RELU inbetween) (relu made it smoother but accuracy was
    worse)
188 layers frozen
189 """
190 train_losses, val_losses, train_acc, val_acc = load_many_pickles(['/home/julian/Downloads/Github/contrastive-
    predictive-coding/models/06_12_21-17-train|(2x)cpc'])
191 plot(train_losses, val_losses, train_acc, val_acc)
192
193
194 # In[4]:
195
196
197 #Downstream on cpc(baseline dataset) batch_size 16->64 windowslength 236
198 """
199 latents used as prediction method (flatten + double linear no RELU inbetween) (relu made it smoother but accuracy was
    worse)
200 layers frozen
201 """
202 train_losses, val_losses, train_acc, val_acc = load_many_pickles(['/home/julian/Downloads/Github/contrastive-
    predictive-coding/models/01_02_21-18'])
203 plot(train_losses, val_losses, train_acc, val_acc)
204
205
206 # In[43]:
207
208
209 #Downstream on cpc(baseline dataset) batch_size 16->64 windowslength 236
210 """
211 Dont run again
212 latents used as prediction method (flatten + double linear no RELU inbetween) (relu made it smoother but accuracy was
    worse)
213 layers frozen
214 """
215 train_losses, val_losses, train_acc, val_acc = load_many_pickles(['/home/julian/Downloads/Github/contrastive-
    predictive-coding/models/01_02_21-18'])
216 plot(train_losses, val_losses, train_acc, val_acc)
217
218
219 # In[39]:
220
221
222 #Downstream on cpc (baseline dataset) batch_size 16->64 windowslength 236
223 """
224 layers unfrozen
225 """
226 train_losses, val_losses, train_acc, val_acc = load_many_pickles(['/home/julian/Downloads/Github/contrastive-
    predictive-coding/models/01_02_21-17'])
227 plot(train_losses, val_losses, train_acc, val_acc)
228
229

```

```

230 # In[37]:
231
232 #Downstream on cpc(baseline dataset) batch_size 16->64 windowslength 236
233 """
234 latents used as prediction method (flatten + linear)
235 layers frozen
236 """
237 train_losses, val_losses, train_acc, val_acc = load_many_pickles(['/home/julian/Downloads/Github/contrastive-
238 predictive-coding/models/01_02_21-16'])
239 plot(train_losses, val_losses, train_acc, val_acc)
240
241
242 # In[29]:
243
244
245 #batch_size 16->64 windowslength 236
246 """
247 CPC intersect sadly smaller batch size
248 Properly working? Cant be compared to old cpc directly (measure downstream performance instead)
249 init hidden on each batch of data
250 """
251 train_losses, val_losses, train_acc, val_acc = load_many_pickles(['/home/julian/Downloads/Github/contrastive-
252 predictive-coding/models/31_01_21-20', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/01
253 _02_21-11'])
252 plot(train_losses, val_losses, train_acc, val_acc)
253
254
255 # In[3]:
256
257
258 #batch_size 16 windowslength 236
259 """
260 CPC intersect sadly smaller batch size
261 Properly working? Cant be compared to old cpc directly (measure downstream performance instead)
262 init hidden on each batch of data
263 """
264 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
265 coding/models/31_01_21-20')
266 plot(train_losses, val_losses, train_acc, val_acc)
267
268 # In[7]:
269
270
271 #batch_size 128 windowslength 236
272 """
273 CPC intersect sadly small batch size
274 New encoder
275 init hidden on each batch of data
276 """
277 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
278 coding/models/29_01_21-17')
279 plot(train_losses, val_losses, train_acc, val_acc)
280
281 # In[6]:
282
283
284 #Baseline v14 on Ptbxl 52 classes no pca, batch_size 128 windowslength 236
285 """
286 Trying CPC again with cleaned code
287 New encoder, challenge dataset ptbxl train, val challenge china
288 init hidden on each batch of data
289 """
290 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
291 coding/models/29_01_21-13')
292 plot(train_losses, val_losses, train_acc, val_acc)
293
294 # In[3]:
295
296
297 #on Ptbxl 52 classes no pca, batch_size 128 windowslength 236
298 """
299 Trying CPC again with cleaned code
300 New encoder, challenge dataset ptbxl train, val challenge china
301 init hidden on each batch of data
302 """
303 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
304 coding/models/28_01_21-20')
305 plot(train_losses, val_losses, train_acc, val_acc)
306
307 # In[3]:
308
309
310 #Baseline v2encoder on Ptbxl 52 classes no pca, batch_size 128 windowslength 236
311 """
312 Trying CPC again with cleaned code

```

```
313 New encoder, challenge dataset ptbxl train, val challenge china
314 """
315 """
316 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
317   coding/models/28_01_21-19')
318 plot(train_losses, val_losses, train_acc, val_acc)
319
320 # In[6]:
321
322 """
323 #Baseline v14 on Ptbxl 52 classes no pca, batch_size 128 new accuracy function, simple loss, windowslength 9500
324 """
325 Trying CPC again with cleaned code and obscure loss function
326 """
327 """
328 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
329   coding/models/26_01_21-15')
330 plot(train_losses, val_losses, train_acc, val_acc)
331
332 # In[4]:
333
334 """
335 #Baseline v14 on Ptbxl 52 classes no pca, batch_size 128 new accuracy function, simple loss, windowslength 9500
336 """
337 Trying CPC again with cleaned code and obscure loss function
338 """
339 """
340 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
341   coding/models/25_01_21-19')
342 plot(train_losses, val_losses, train_acc, val_acc)
343
344 # In[4]:
345
346 """
347 #Baseline v14 on Ptbxl 52 classes no pca, batch_size 128 new accuracy function, simple loss, windowslength 9500
348 """
349 Testing Multiscale ResNet (Conv1d implementation from github) + conv1x1 to summarize 182 values at end
350
351 """
352 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
353   coding/models/19_01_21-14')
354 plot(train_losses, val_losses, train_acc, val_acc)
355
356 # In[8]:
357
358 """
359 #Baseline v0_3 on Ptbxl 52 classes no pca, batch_size 128 new accuracy function, simple loss, windowslength 9500
360 """
361 6 Cnn1d, 3 downsample, other make features
362 learn rate fixed
363
364 """
365 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
366   coding/models/19_01_21-18')
367 plot(train_losses, val_losses, train_acc, val_acc)
368
369 # In[11]:
370
371 """
372 #Baseline v0_3 on Ptbxl 52 classes no pca, batch_size 128 new accuracy function, simple loss, windowslength 9500
373 """
374 6 Cnn1d, 3 downsample, other make features
375 learn rate smaller over time (21 *0.5 here)
376
377 """
378 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
379   coding/models/18_01_21-14')
380 plot(train_losses, val_losses, train_acc, val_acc)
381
382 # In[3]:
383
384 """
385 #Baseline v13 on Ptbxl 52 classes no pca, batch_size 128 new accuracy function, simple loss, windowslength 9500
386 """
387 TCN-like Block: 2 Conv1D, first stride 2, second dilation 2
388 weight_norm as in TCN
389 residual connection (needs downsample in data dimension to 4745)
390
391 No Downsampling, Linear on on 52x9500[-1]? Weird
392 Linear 52->52
393 """
394 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
395   coding/models/14_01_21-16')
396 plot(train_losses, val_losses, train_acc, val_acc)
```

```

396
397 # In[11]:
398
399
400 #Baseline v12 on Ptbxl 52 classes no pca, batch_size 8 new accuracy function, simple loss, windowslength 9500
401 """
402 BATCH_SIZE ONLY 8! (Model too big) So 10 times more updates than Baseline v11
403 Input is also flattened like in Mnist example (hm)
404 so channel input is 1 instead of 12
405 TCN with 3 Blocks (12, 24, 52 channels), dropout 0.2, kernel_size 3
406 No Downsampling, Linear on on 52x9500[-1]? Weird
407 Linear 52->52
408 """
409
410 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
411 coding/models/14_01_21-16')
412 plot(train_losses, val_losses, train_acc, val_acc)
413
414 # In[10]:
415
416
417 #Baseline v11 on Ptbxl 52 classes no pca, batch_size 80 new accuracy function, simple loss, windowslength 9500
418 """
419 TCN with 3 Blocks (12, 24, 52 channels), dropout 0.2, kernel_size 3
420 No Downsampling, Linear on on 52x9500[-1]? Weird
421 Very close to official tcn mnist example
422 Linear 52->52
423 """
424 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
425 coding/models/14_01_21-15')
426 plot(train_losses, val_losses, train_acc, val_acc)
427
428 # In[8]:
429
430
431 #Baseline v10 on Ptbxl 52 classes no pca, batch_size 80 new accuracy function, simple loss, windowslength 9500
432 """
433 TCN with 3 Blocks (12, 24, 52 channels), dropout 0.2, kernel_size 3
434 Downsampling with Conv1x1 to summarize all 9500 values
435
436 Linear 52->52
437 """
438 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
439 coding/models/14_01_21-14')
440 plot(train_losses, val_losses, train_acc, val_acc)
441
442 # In[6]:
443
444
445 #Baseline v0 on Ptbxl 52 classes no pca, batch_size 128 new accuracy function, simple loss, windowslength 9500
446 """
447 2 Layers of Convs with filter size 7 and 3 and dilation of 1, 3
448 Summarizes all 9488 values with Transpose + Con1x1
449 Full connected has only out_channels as input
450 """
451 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
452 coding/models/14_01_21-11')
453 plot(train_losses, val_losses, train_acc, val_acc)
454
455 # In[4]:
456
457
458 #Baseline v9 on Ptbxl 52 classes no pca, batch_size 24 new accuracy function, simple loss, windowslength 9500
459 """
460 Similiar to v2, v8
461 4 Layers of Convs with filter size 3 and NO DILATION
462 Max Poolwith size 3 after every Layer (besides last) to downsample
463 Summarizes all 348 values with Transpose + Con1x1
464 Full connected has only out_channels as input
465 """
466 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
467 coding/models/13_01_21-19')
468 plot(train_losses, val_losses, train_acc, val_acc)
469
470 # In[23]:
471
472
473 #Baseline v8 on Ptbxl 52 classes no pca, batch_size 24 new accuracy function, simple loss, windowslength 9500
474 """
475 Similiar to v2
476 4 Layers of Convs with filter size 3 and dilation of 2^n to get a big receptive field
477 Max Poolwith size 3 after every Layer (besides last) to downsample
478 Summarizes all 334 values with Transpose + Con1x1
479 Full connected has only out_channels as input
480 """

```

```

481 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/13_01_21-18')
482 plot(train_losses, val_losses, train_acc, val_acc)
483
484
485 # In[15]:
486
487
488 #Baseline v7 (no Batchnorm) on Ptbxl 52 classes no pca, batch_size 128 new accuracy function, simple loss,
    windowslength 9500
489 """Has one less layer than v6 but a stride of 2 + kernel size of 7 in the first layer
490 (to downsample as in resnet).
491 Summarizes all values using transpose + conv1x1
492 """
493 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/13_01_21-16')
494 plot(train_losses, val_losses, train_acc, val_acc)
495
496
497 # In[3]:
498
499
500 #Baseline v6 (no Batchnorm) on Ptbxl 52 classes no pca, batch_size 128 new accuracy function, simple loss,
    windowslength 9500
501 """6 Layers of Convs with filter size 3 and dilation of 2^n to get a really big receptive field
502 Summarizes all values using transpose + conv1x1
503 """
504 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/13_01_21-15')
505 plot(train_losses, val_losses, train_acc, val_acc)
506
507
508 # In[10]:
509
510
511 #Baseline v5 (no Batchnorm) on Ptbxl 52 classes no pca, batch_size 128 new accuracy function, simple loss,
    windowslength 9500
512 """4 Layers of Convs with filter size 3 and dilation of 2^n to get a big receptive field
513 Summarizes all channels using conv1x1
514 Fully connected has to over 9400 inputs!
515 """
516 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/12_01_21-19')
517 plot(train_losses, val_losses, train_acc, val_acc)
518
519
520 # In[11]:
521
522
523 #DO NOT RERUN
524 """4 Layers of Convs with filter size 3 and dilation of 2^n to get a big receptive field
525 left over maxpool not removed so incorrect
526 Summarizes all channels using conv1x1
527 Fully connected has to over 9400 inputs!
528 """
529 #Baseline v5 (no Batchnorm/ false maxpool) on Ptbxl 52 classes no pca, batch_size 128 new accuracy function, simple
    loss, windowslength 9500
530 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/12_01_21-19')
531 plot(train_losses, val_losses, train_acc, val_acc)
532
533
534 # In[9]:
535
536
537 #Baseline v4 (no Batchnorm v3) on Ptbxl 52 classes no pca, batch_size 128 new accuracy function, simple loss,
    windowslength 9500
538 """
539 4 Layers of Convs with filter size 3 and dilation of 2^n to get a big receptive field
540 Summarizes all values using transpose + conv1x1
541 Full connected has only out_channels as input
542 """
543 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/12_01_21-18')
544 plot(train_losses, val_losses, train_acc, val_acc)
545
546
547 # In[4]:
548
549
550 #Baseline v3 on Ptbxl 52 classes no pca, batch_size 128 new accuracy function, simple loss, windowslength 9500
551 """
552 4 Layers of Convs with filter size 3 and dilation of 2^n to get a big receptive field
553 Batchnorm after every Layer before ReLU
554 Summarizes all values using transpose + conv1x1
555 Full connected has only out_channels as input
556 """
557 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/12_01_21-17')
558 plot(train_losses, val_losses, train_acc, val_acc)

```

```

559
560
561 # In[3]:
562
563
564 #Baseline v2 on Ptbxl 52 classes no pca, batch_size 24 new accuracy function, simple loss, windowslength 9500
565 """
566 5 Layers of Convs with filter size 3 and dilation of 2^n to get a big receptive field
567 Max Poolwith size 3 after every Layer to downsample
568 Pools all values at the end using Adaptive Average Pooling
569 Full connected has only out_channels as input
570 """
571 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
572   coding/models/12_01_21-15')
573 plot(train_losses, val_losses, train_acc, val_acc)
574
575 # In[6]:
576
577
578 #Baseline Autoencoder on Ptbxl 52 classes no pca, batch_size 24 new accuracy function, simple loss, windowslength 512,
579   latents 256
580 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
581   coding/models/06_01_21-16')
582 plot(train_losses, val_losses, train_acc, val_acc)
583
584 # In[4]:
585
586
587 #Baseline Autoencoder on Ptbxl 52 classes no pca, batch_size 24 new accuracy function, simple loss, windowslength 9500
588 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
589   coding/models/06_01_21-13')
590 plot(train_losses, val_losses, train_acc, val_acc)
591
592 # In[3]:
593
594 #Autoencoder on Ptbxl 52 classes no pca, batch_size 24 new accuracy function, simple loss
595 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
596   coding/models/04_01_21-18')
597 plot(train_losses, val_losses, train_acc, val_acc)
598
599 # In[11]:
600
601
602 #Baseline on Ptbxl 52 classes no pca, batch_size 24 new accuracy function, simple loss
603 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
604   coding/models/03_01_21-19')
605 plot(train_losses, val_losses, train_acc, val_acc)
606
607 # In[9]:
608
609
610 #Downstream on Ptbxl 52 classes no pca, batch_size 24 new accuracy function, simple loss
611 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
612   coding/models/03_01_21-18')
613 plot(train_losses, val_losses, train_acc, val_acc)
614
615 # In[4]:
616
617
618 #Downstream on Ptbxl 52 classes no pca, batch_size 24 new accuracy function, BCELoss
619 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
620   coding/models/03_01_21-17')
621 plot(train_losses, val_losses, train_acc, val_acc)
622
623 # In[26]:
624
625
626 #BASELINE on Ptbxl 52 classes, 6 Layers of Convs, no pca, batch_size 24 new accuracy function,
627   MultiLabelSoftMarginLoss?
628 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
629   coding/models/18_12_20-17')
630 plot(train_losses, val_losses, train_acc, val_acc)
631
632 # In[17]:
633
634
635 #BASELINE on Ptbxl 52 classes, 6 Layers of Convs, no pca, batch_size 24
636 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
637   coding/models/18_12_20-15')
638 plot(train_losses[5:], val_losses[5:], train_acc[5:], val_acc[5:])
639

```

```
638
639 # In[6]:
640
641
642 #BASELINE on Ptbxl 52 classes, 6 Layers of Convs, no pca, batch_size 24
643 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
644   coding/models/18_12_20-14')
645 plot(train_losses, val_losses, train_acc, val_acc)
646
647 # ## Losses for single target
648
649 # In[41]:
650
651
652 #BASELINE on Ptb, 6 Layers of Convs, no pca, batch_size 24 #WRONG!!! train=valset
653 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
654   coding/models/15_12_20-14')
655 plot(train_losses, val_losses, train_acc, val_acc)
656
657 # In[33]:
658
659
660 #BASELINE on Ptbxl (correct split), 6 Layers of Convs, no pca, batch_size 24 #WRONG!!! train=valset
661 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
662   coding/models/14_12_20-19')
663 plot(train_losses, val_losses, train_acc, val_acc)
664
665 # In[13]:
666
667
668 #Downstream on PTBxl (correct split), ResNet, less windows 6-6, bigger context 512, no pca (12 channels), layers not?
669   frozen, batch_size 12
670 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
671   coding/models/10_12_20-22')
672 plot(train_losses, val_losses, train_acc, val_acc)
673
674 # In[11]:
675
676
677 #Downstream on PTBxl (correct split), ResNet, less windows 6-6, bigger context 512, no pca (12 channels), layers
678   frozen, batch_size 12
679 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
680   coding/models/10_12_20-21')
681 plot(train_losses, val_losses, train_acc, val_acc)
682
683 # In[9]:
684
685
686 #Downstream on PTB, ResNet, less windows 6-6, bigger context 512, no pca (12 channels), batch_size 1, layers NOT
687   frozen
688 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
689   coding/models/10_12_20-15')
690 plot(train_losses, val_losses, train_acc, val_acc)
691
692 # In[5]:
693
694
695 #CPC on PTB, PTBxl (correct split), ResNet, less windows 6-6, bigger context 512, no pca (12 channels)
696 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
697   coding/models/10_12_20-12')
698 plot(train_losses, val_losses, train_acc, val_acc)
699
700 #CPC on PTB, PTBxl (correct split), ResNet, less windows 6-6, bigger context 512, pca 2 channels
701 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
702   coding/models/09_12_20-15')
703 plot(train_losses, val_losses, train_acc, val_acc)
704
705 # In[5]:
706
707
708 #CPC on PTB, PTBxl (correct split), ResNet, less windows 6-6, pca 2 channels
709 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
710   coding/models/09_12_20-14')
711 plot(train_losses, val_losses, train_acc, val_acc)
712
713 # In[4]:
```

```

716 #CPC on PTB, PTBx1 (correct split), ResNet, many windows 12-12, pca 2 channels
717 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/09_12_20-13')
718 plot(train_losses, val_losses, train_acc, val_acc)
719
720
721 # In[6]:
722
723
724 #CPC on PTB, PTBx1 (correct split), small model (slightly bigger), pca 2 channels
725 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/08_12_20-19')
726 plot(train_losses, val_losses, train_acc, val_acc)
727
728
729 # In[4]:
730
731
732 #CPC on PTB, PTBx1 (correct split), small model, pca 2 channels
733 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/08_12_20-16')
734 plot(train_losses, val_losses, train_acc, val_acc)
735
736
737 # In[13]:
738
739
740 #CPC on PTB, PTBx1 (correct split), small model, no pca
741 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/07_12_20-18')
742 plot(train_losses, val_losses, train_acc, val_acc)
743
744
745 # In[9]:
746
747
748 #CPC on sinus data, small model, no pca
749 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/07_12_20-17')
750 plot(train_losses, val_losses, train_acc, val_acc) #Sinus data too similar for contrastive loss
751
752
753 # In[11]:
754
755
756 #CPC with PCA and 2 components (PTB + PTBXL), also small model
757 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/30_11_20-16')
758 plot(train_losses, val_losses, train_acc, val_acc)
759
760
761 # In[8]:
762
763
764 #CPC with PCA (PTB + PTBXL) and 2 components
765 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/30_11_20-14')
766 plot(train_losses, val_losses, train_acc, val_acc)
767
768
769 # In[4]:
770
771
772 #CPC with PCA (only PTB data) and 2 components
773 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/30_11_20-13')
774 plot(train_losses, val_losses, train_acc, val_acc)
775
776
777 # In[27]:
778
779
780 #CPC Got rid of batchnorm
781 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/27_11_20-14')
782 plot(train_losses, val_losses, train_acc, val_acc)
783
784
785 # In[21]:
786
787
788 train_losses, val_losses, train_acc, val_acc = load_pickles('/home/julian/Downloads/Github/contrastive-predictive-
    coding/models/27_11_20-10')
789 plot(train_losses, val_losses, train_acc, val_acc)

```

B.7.4 jupyter_notebooks -> Challenge Data Visualization.py (code)

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[2]:
5
6
7 import torch
8 import os
9 import numpy as np
10 import matplotlib.pyplot as plt
11 from random import shuffle
12 from torch.utils.data import IterableDataset
13 from torch.utils.data import ChainDataset
14 import seaborn as sns
15 from scipy import stats
16 import pandas as pd
17 from torch.utils.data._utils.collate import np_str_obj_array_pattern, default_collate_err_msg_format
18
19 import sys
20 sys.path.insert(0, 'Downloads/Github/contrastive-predictive-coding/util/visualize')
21 #import timeseries_to_image_converter
22
23
24 # In[3]:
25
26
27 class ECGChallengeDatasetBaseline(torch.utils.data.IterableDataset):
28     def __init__(self, BASE_DIR, window_size, pad_to_size=None, files=None, channels=None, return_labels=False,
29                  return_filename=False, classes=None, normalize_fn=None, verbose=False):
30         super(torch.utils.data.IterableDataset).__init__()
31         self.BASE_DIR = BASE_DIR
32         self.window_size = window_size
33         self.pad_to_size = pad_to_size or window_size
34         self.files = files or self.search_files()
35         self.classes = classes or get_classes(self.files)
36         if verbose:
37             self.print_file_attributes()
38         self.channels = channels
39         self.total_length = 1 #Trying a weird approach (calculated in __iter__)
40         self.return_labels = return_labels
41         self.return_filename = return_filename
42         self.normalize_fn = normalize_fn if not normalize_fn is None else lambda x: x
43
44     def generate_datasets_from_split_file(self, ttsfile='train-test-splits.txt'):
45         splits = load_train_test_split(os.path.join(self.BASE_DIR, ttsfile))
46         return tuple(ECGChallengeDatasetBaseline(self.BASE_DIR, self.window_size, self.pad_to_size, files=s,
47                                                 channels=self.channels, return_labels=self.return_labels,
48                                                 return_filename=self.return_filename, classes=self.classes,
49                                                 normalize_fn=self.normalize_fn)
49         for s in splits)
50
51     def generate_datasets_from_split_list(self, trainf, valf, testf):
52         splits = [trainf, valf, testf]
53         return tuple(ECGChallengeDatasetBaseline(self.BASE_DIR, self.window_size, self.pad_to_size, files=s,
54                                                 channels=self.channels, return_labels=self.return_labels,
55                                                 return_filename=self.return_filename, classes=self.classes,
56                                                 normalize_fn=self.normalize_fn)
56         for s in splits)
57
58     def __iter__(self):
59         file_index = 0
60         shuffle(self.files)
61         while file_index < len(self.files):
62             current_file = self.files[file_index]
63             data = self.normalize_fn(self._read_recording_file(current_file))
64             if self.channels:
65                 data = data[:, self.channels]
66             if self.return_labels:
67                 labels = self._read_header_labels(current_file)
68                 if len(data) - self.window_size > 0:
69                     offset = np.random.randint(len(data) - self.window_size) # Random offset
70                     data = data[offset:self.window_size+offset]
71                 else:
72                     offset = 0
73                     data = np.pad(data, ((max(0, self.pad_to_size-min(self.window_size, len(data))), 0), (0,0)))
74                 if not any([self.return_filename, self.return_labels]):
75                     yield data
76                 else:
77                     yield [data] + ([labels] if self.return_labels else [None]) + ([current_file] if self.return_filename
78                     else []])
79             file_index += 1
80
81     def _read_recording_file(self, path_without_ext):
82         fp = path_without_ext + '.mat'
83         return load_recording(fp, key='val').transpose()
84
85     def _read_header_file(self, path_without_ext):
86         fp = path_without_ext + '.hea'
87         return load_header(fp)
88

```

```

89     def _read_header_labels(self, path_without_ext, onerror_class='426783006'):
90         header = self._read_header_file(path_without_ext)
91         return encode_header_labels(header, self.classes, onerror_class)
92
93     def __len__(self):
94         return self.total_length
95
96     def search_files(self):
97         headers, records = find_challenge_files(self.BASE_DIR)
98         print(len(records), 'record files found in ', self.BASE_DIR)
99         return list(map(lambda x: os.path.splitext(x)[0], records)) #remove extension
100
101    def random_train_split(self, train_fraction=0.7, val_fraction=0.2, test_fraction=0.1, save=True,
102                           save_path_overwrite=None, filename_overwrite=None):
103        assert train_fraction+val_fraction+test_fraction <= 1
104        N = len(self.files)
105        shuffle(self.files)
106        train_slice = slice(0, int(train_fraction*N))
107        val_slice = slice(train_slice.stop, train_slice.stop+int(val_fraction*N))
108        test_slice = slice(val_slice.stop, val_slice.stop+int(test_fraction*N))
109        if save:
110            p = save_path_overwrite or self.BASE_DIR
111            fname = filename_overwrite or 'train-test-splits.txt'
112            save_train_test_split(os.path.join(p, fname), self.files[train_slice], self.files[val_slice], self.files[test_slice])
113        return self.files[train_slice], self.files[val_slice], self.files[test_slice]
114
115    def random_train_split_with_class_count(self, train_fraction=0.7, val_fraction=0.2, test_fraction=0.1, save=True,
116                                           save_path_overwrite=None, filename_overwrite=None):
117        assert train_fraction+val_fraction+test_fraction <= 1
118        class_buckets = [[] for x in range(len(self.classes))] #Creates classes buckets
119        for i, f in enumerate(self.files):
120            label = self._read_header_labels(f)
121            label_idx = np.argwhere(label).flatten()
122            for l in label_idx: #can return more than one (multi-label)
123                class_buckets[l].append(f) #Put this file into class l bucket
124        train_files, val_files, test_files = [], [], []
125        sorted_idx = list(sorted(range(len(class_buckets)), key=lambda x: len(class_buckets[x]))) #make index sorted
126        by class count low->high
127        print(sorted_idx)
128        while len(sorted_idx) > 0:
129            idx = sorted_idx[0]
130            shuffle(class_buckets[idx])
131            b = class_buckets[idx]
132            c_N = len(b)
133            train_slice = slice(0, int(train_fraction * c_N))
134            val_slice = slice(train_slice.stop, train_slice.stop + int(val_fraction * c_N))
135            test_slice = slice(val_slice.stop, val_slice.stop + int(test_fraction * c_N))
136            train_files += b[train_slice]
137            val_files += b[val_slice]
138            test_files += b[test_slice]
139            used_set = set(b[train_slice]+b[val_slice]+b[test_slice])
140            for j in sorted_idx[1:]:
141                class_buckets[j] = [x for x in class_buckets[j] if x not in used_set] #REMOVE THIS FILE FROM ALL OTHER
BUCKETS
142        sorted_idx = list(sorted(sorted_idx[1:], key=lambda x: len(class_buckets[x]))) #sort again (removal may
change order)
143
144        train_files = list(set(train_files))
145        val_files = list(set(val_files))
146        test_files = list(set(test_files))
147        if save:
148            p = save_path_overwrite or self.BASE_DIR
149            fname = filename_overwrite or 'train-test-splits.txt'
150            save_train_test_split(os.path.join(p, fname), train_files, val_files, test_files)
151        return train_files, val_files, test_files
152
153    def count_classes(self):
154        counts = np.zeros(len(self.classes), dtype=int)
155        for i, f in enumerate(self.files):
156            labels = self._read_header_labels(f).astype(float)
157            counts += labels != 0.0 #Count where label isnt 0
158
159    def train_split_with_function(self, file_mapping_function, save_path=None):
160        splits = [[], [], []]
161        for f in self.files:
162            i = file_mapping_function(f)
163            splits[i].append(f)
164        if not save_path is None:
165            save_train_test_split(os.path.join(save_path, 'train-test-splits.txt'), splits[0], splits[1], splits[2])
166        return splits[0], splits[1], splits[2]
167
168    def print_file_attributes(self):
169        f = self.files[0]
170        print('Information for file', f)
171        data = self._read_recording_file(f)
172        print('Data has shape:', data.shape)

```

```

173     header = self._read_header_file(f)
174     print('Header is:', header, end='#####\n')
175     print('Classes found in data folder:', self.classes)
176     labels = self._read_header_labels(f)
177     print('Labels have shape', labels.shape)
178
179     def merge_and_update_classes(self, datasets):
180         all_classes = set()
181         for d in datasets:
182             all_classes = all_classes | set(d.classes.keys())
183         all_classes = dict(zip(sorted(all_classes), range(len(all_classes))))
184         for d in datasets:
185             d.classes = all_classes
186         print('Labels for datasets set to:', all_classes)
187
188     def remove_unknown_label_files(self):
189         for f in self.files[:]:
190             if self._read_header_labels(f, onerror_class=None) is None:
191                 print('removed', f)
192                 self.files.remove(f)
193
194
195     def filter_update_classes_by_count(datasets, min_count, add_unknown=False):
196         counts, all_classes = count_merged_classes(datasets)
197         filtered_classes = set()
198         for k, v in all_classes.items():
199             if counts[v] >= min_count:
200                 filtered_classes.add(k)
201         if add_unknown:
202             filtered_classes.add('-1')
203         filtered_classes = dict(zip(sorted(filtered_classes), range(len(filtered_classes))))
204         for d in datasets:
205             d.classes = filtered_classes
206             d.remove_unknown_label_files()
207         return filtered_classes
208
209     def count_merged_classes(datasets):
210         all_classes = set()
211         for d in datasets:
212             all_classes = all_classes | set(d.classes.keys())
213         all_classes = sorted(all_classes)
214         all_classes = dict(zip(all_classes, range(len(all_classes))))
215         counts = np.zeros(len(all_classes), dtype=int)
216         for d in datasets:
217             temp_classes = d.classes.copy() #set back later
218             d.classes = all_classes
219             counts += d.count_classes()
220             d.classes = temp_classes #set back
221         return counts, all_classes
222
223     def load_train_test_split(tts_file_path:str):
224         splits = [[],[],[]]
225         with open(tts_file_path, 'r') as f:
226             line_count = 0
227             for line in f:
228                 if not line.strip().startswith('#') or line == '\n': #Comment line or empty line
229                     splits[line_count] = [f.strip() for f in line.split(',')]
230                     line_count += 1
231                 if line_count >= 3:
232                     break
233         return splits[0], splits[1], splits[2]
234
235     def save_train_test_split(tts_file:str, trainf=[], valf=[], testf=[]):
236         if os.path.isfile(tts_file): #make a backup just in case
237             sf = os.path.split(tts_file)
238             print(os.path.join(sf[0], timestamp.string_timestamp_minutes())+sf[1])
239             os.rename(tts_file, os.path.join(sf[0], timestamp.string_timestamp_minutes())+sf[1])
240
241         with open(tts_file, 'w') as f:
242             f.write('#train files\n')
243             f.write(",".join(trainf)+'\n')
244             f.write('#val files\n')
245             f.write(",".join(valf) + '\n')
246             f.write('#test files\n')
247             f.write(",".join(testf))
248
249
250     def collate_fn(batch): #https://github.com/pytorch/pytorch/blob/master/torch/utils/data/_utils/collate.py
251         r"""Puts each data field into a tensor with outer dimension batch size"""
252
253         elem = batch[0]
254         elem_type = type(elem)
255         if isinstance(elem, torch.Tensor):
256             out = None
257             if torch.utils.data.get_worker_info() is not None:
258                 # If we're in a background process, concatenate directly into a
259                 # shared memory tensor to avoid an extra copy
260                 numel = sum([x.numel() for x in batch])
261                 storage = elem.storage().__new_shared(numel)
262                 out = elem.new(storage)

```

```

263     return torch.stack(batch, 0, out=out)
264 elif elem_type.__module__ == 'numpy' and elem_type.__name__ != 'str__':
265     string_:
266         if elem_type.__name__ == 'ndarray' or elem_type.__name__ == 'memmap':
267             # array of string classes and object
268             if np_str_obj_array_pattern.search(elem.dtype.str) is not None:
269                 raise TypeError(default_collate_err_msg_format.format(elem.dtype))
270
271         return collate_fn([torch.as_tensor(b) for b in batch])
272     elif elem.shape == (): # scalars
273         return torch.as_tensor(batch)
274     elif isinstance(elem, float):
275         return torch.tensor(batch)
276     elif issubclass(type(elem), int):
277         return torch.tensor(batch)
278     elif issubclass(type(elem), str):
279         return batch
280     elif issubclass(type(elem), dict):
281         return {key: collate_fn([d[key] for d in batch]) for key in elem}
282     elif isinstance(elem, tuple) and hasattr(elem, '_fields'): # namedtuple
283         return elem_type(*collate_fn(samples) for samples in zip(*batch)))
284     elif issubclass(type(elem), list):
285         # check to make sure the elements in batch have consistent size
286         transposed = zip(*batch)
287         return [collate_fn(samples) for samples in transposed]
288
289 def normalize_feature_scaling(data, low:int=0, high:int=1):
290     mini = np.min(data, axis=1)[:, np.newaxis]
291     maxi = np.max(data, axis=1)[:, np.newaxis]
292     dif = np.where(maxi-mini==0, 1, maxi-mini)
293     return (data-mini)/(dif)*high-low
294
295 #!/usr/bin/env python
296
297 # These are helper variables and functions that you can use with your code.
298 # Do not edit this script.
299
300 import numpy as np
301 import os
302 from scipy.io import loadmat
303
304 # Define 12, 6, and 2 lead ECG sets.
305 twelve_leads = ('I', 'II', 'III', 'aVR', 'aVL', 'aVF', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6')
306 six_leads = ('I', 'II', 'III', 'aVR', 'aVL', 'aVF')
307 two_leads = ('II', 'V5')
308
309 # Check if a variable is an integer or represents an integer.
310 def is_integer(x):
311     try:
312         if int(x)==float(x):
313             return True
314         else:
315             return False
316     except (ValueError, TypeError):
317         return False
318
319 # Find header and recording files.
320 def find_challenge_files(data_directory):
321     header_files = list()
322     recording_files = list()
323     for f in os.listdir(data_directory):
324         root, extension = os.path.splitext(f)
325         if not root.startswith('.') and extension=='.hea':
326             header_file = os.path.join(data_directory, root + '.hea')
327             recording_file = os.path.join(data_directory, root + '.mat')
328             if os.path.isfile(header_file) and os.path.isfile(recording_file):
329                 header_files.append(header_file)
330                 recording_files.append(recording_file)
331
332 # Load header file as a string.
333 def load_header(header_file):
334     with open(header_file, 'r') as f:
335         header = f.read()
336     return header
337
338 # Load recording file as an array.
339 def load_recording(recording_file, header=None, leads=None, key='val'):
340     x = loadmat(recording_file)[key]
341     recording = np.asarray(x, dtype=np.float32)
342     return recording
343
344 # Get leads from header.
345 def get_leads(header):
346     leads = list()
347     for i, l in enumerate(header.split('\n')):
348         entries = l.split(' ')
349         if i==0:
350             num_leads = int(entries[1])
351             elif i<=num_leads:

```

```

352         leads.append(entries[-1])
353     else:
354         break
355     return leads
356
357 # Get age from header.
358 def get_age(header):
359     age = None
360     for l in header.split('\n'):
361         if l.startswith('#Age'):
362             try:
363                 age = float(l.split(': ')[1].strip())
364             except:
365                 age = float('nan')
366     return age
367
368 # Get sex from header.
369 def get_sex(header):
370     sex = None
371     for l in header.split('\n'):
372         if l.startswith('#Sex'):
373             try:
374                 sex = l.split(': ')[1].strip()
375             except:
376                 pass
377     return sex
378
379 # Get frequency from header.
380 def get_frequency(header):
381     frequency = None
382     for i, l in enumerate(header.split('\n')):
383         if i==0:
384             try:
385                 frequency = float(l.split(' ')[2])
386             except:
387                 pass
388         else:
389             break
390     return frequency
391
392 # Get amplitudes from header.
393 def get_amplitudes(header, leads):
394     amplitudes = np.zeros(len(leads), dtype=np.float32)
395     for i, l in enumerate(header.split('\n')):
396         entries = l.split(' ')
397         if i==0:
398             num_leads = int(entries[1])
399         elif i<=num_leads:
400             current_lead = entries[-1]
401             if current_lead in leads:
402                 j = leads.index(current_lead)
403                 try:
404                     amplitudes[j] = float(entries[2].split('//')[0])
405                 except:
406                     pass
407             else:
408                 break
409     return amplitudes
410
411 # Get baselines from header.
412 def get_baselines(header, leads):
413     baselines = np.zeros(len(leads), dtype=np.float32)
414     for i, l in enumerate(header.split('\n')):
415         entries = l.split(' ')
416         if i==0:
417             num_leads = int(entries[1])
418         elif i<=num_leads:
419             current_lead = entries[-1]
420             if current_lead in leads:
421                 j = leads.index(current_lead)
422                 try:
423                     baselines[j] = float(entries[4].split('//')[0])
424                 except:
425                     pass
426             else:
427                 break
428     return baselines
429
430 # Get labels from header.
431 def get_labels(header):
432     labels = list()
433     for l in header.split('\n'):
434         if l.startswith('#Dx'):
435             entries = l.split(': ')[1].split(',')
436             for entry in entries:
437                 labels.append(entry.strip())
438     return labels
439
440 # Save outputs from model.
441 def save_outputs(output_file, classes, labels, probabilities):

```

```

442     # Extract the recording identifier from the filename.
443     head, tail = os.path.split(output_file)
444     root, extension = os.path.splitext(tail)
445     recording_identifier = root
446
447     # Format the model outputs.
448     recording_string = '#{0}'.format(recording_identifier)
449     class_string = ','.join(str(c) for c in classes)
450     label_string = ','.join(str(l) for l in labels)
451     probabilities_string = ','.join(str(p) for p in probabilities)
452     output_string = recording_string + '\n' + class_string + '\n' + label_string + '\n' + probabilities_string + '\n'
453
454     # Save the model outputs.
455     with open(output_file, 'w') as f:
456         f.write(output_string)
457
458
459 def get_classes(files_without_ext):
460     classes = set()
461     for filename in files_without_ext:
462         with open(filename+'.hea', 'r') as f:
463             for l in f:
464                 if l.startswith('#Dx'):
465                     tmp = l.split(':')[1].split(',')
466                     for c in tmp:
467                         classes.add(c.strip())
468     return dict(zip(sorted(classes), range(len(classes))))
469
470 def encode_header_labels(header, classes, onerror_class='426783006'):
471     labels_act = np.zeros(len(classes))
472     for l in header.split('\n'):
473         if l.startswith('#x'):
474             tmp = l.split(':')[1].split(',')
475             for c in tmp:
476                 cl = c.strip()
477                 if cl in classes:
478                     class_index = classes[cl]
479                 else:
480                     if onerror_class is None:
481                         return None
482                     class_index = classes[onerror_class]
483                     labels_act[class_index] = 1
484     return labels_act
485
486 def save_challenge_predictions(output_directory, filenames, classes, scores, labels):
487     for i in range(0, len(filenames)):
488         filename = filenames[i]
489         sc = scores[i]
490         ls = labels[i]
491         recording = os.path.splitext(filename)[0]
492         new_file = filename.replace('.mat','') + '.csv'
493
494         output_file = os.path.join(output_directory, new_file)
495
496         # Include the filename as the recording number
497         recording_string = '#{0}'.format(recording)
498         class_string = ','.join(classes)
499         label_string = ','.join(str(i) for i in ls)
500         score_string = ','.join(str(i) for i in sc)
501         with open(output_file, 'w') as f:
502             f.write(recording_string + '\n' + class_string + '\n' + label_string + '\n' + score_string + '\n')
503
504
505 # In[5]:
506
507
508 crop_size = 4500
509
510
511 georgia_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/georgia_challenge/',
512                                                 window_size=crop_size, pad_to_size=crop_size, return_labels=True,
513                                                 normalize_fn=normalize_feature_scaling)
514 cpsc_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/cpsc2018_challenge/',
515                                                 window_size=crop_size, pad_to_size=crop_size, return_labels=
516                                                 True,
517                                                 normalize_fn=normalize_feature_scaling)
518 cpsc2_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/china_challenge', window_size=crop_size,
519                                                 pad_to_size=crop_size, return_labels=True,
520                                                 normalize_fn=normalize_feature_scaling)
521 ptbxl_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/ptbxl_challenge', window_size=crop_size,
522                                                 pad_to_size=crop_size, return_labels=True,
523                                                 normalize_fn=normalize_feature_scaling)
524
525
526 ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.generate_datasets_from_split_file()
527 georgia_train, georgia_val, t2 = georgia_challenge.generate_datasets_from_split_file()
528 cpsc_train, cpsc_val, t3 = cpsc_challenge.generate_datasets_from_split_file()
529 cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.generate_datasets_from_split_file()
530

```

```

531 filter_update_classes_by_count([ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3,
532     cpsc2_train, cpsc2_val, t4], 1)
533 print('Classes after last update', len(ptbxl_train.classes), ptbxl_train.classes)
533
534 counts_all, counted_classes_all = count_merged_classes([ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2,
535     cpsc_train, cpsc_val, t3, cpsc2_train, cpsc2_val, t4])
535 PATIENTS_TRAIN_TOTAL = sum(map(lambda x: len(x.files), [ptbxl_train, georgia_train, cpsc_train, cpsc2_train]))
536 PATIENTS_VAL_TOTAL = sum(map(lambda x: len(x.files), [ptbxl_val, georgia_val, cpsc_val, cpsc2_val]))
537 PATIENTS_TEST_TOTAL = sum(map(lambda x: len(x.files), [t1, t2, t3, t4]))
538 PATIENTS_TOTAL = PATIENTS_TRAIN_TOTAL + PATIENTS_VAL_TOTAL + PATIENTS_TEST_TOTAL
539 print(f"Train: {PATIENTS_TRAIN_TOTAL}, Val: {PATIENTS_VAL_TOTAL}, Test: {PATIENTS_TEST_TOTAL}, Total: {PATIENTS_TOTAL}")
539
540 train_set = ChainDataset([ptbxl_train, georgia_train, cpsc_train, cpsc2_train])
541 val_set = ChainDataset([ptbxl_val, georgia_val, cpsc_val, cpsc2_val])
542 test_set = ChainDataset([t1, t2, t3, t4])
543
544
545 # In[24]:
546
547
548 counts = {}
549 counted_classes = {}
550 counts['ptbxl'], counted_classes['ptbxl'] = count_merged_classes([ptbxl_train, ptbxl_val, t1])
551 counts['georgia'], counted_classes['georgia'] = count_merged_classes([georgia_train, georgia_val, t2,])
552 counts['cpsc'], counted_classes['cpsc'] = count_merged_classes([cpsc_train, cpsc_val, t3])
553 counts['cpsc2'], counted_classes['cpsc2'] = count_merged_classes([cpsc2_train, cpsc2_val, t4])
554
555
556 # In[65]:
557
558
559 save_path = '/home/julian/Documents/projekt-master'
560
561
562 # ## Class descriptions
563
564 # In[6]:
565
566
567 import pandas as pd
568 import urllib
569 import time
570 import json
571
572
573 # In[8]:
574
575
576 snomed_data = {}
577 DOWNLOAD AGAIN = False
578 if not DOWNLOAD AGAIN:
579     try:
580         with open('/home/julian/Downloads/Github/contrastive-predictive-coding/experiments/snomed_data.json', 'r') as f:
581             snomed_data = json.load(f)
582     except FileNotFoundError:
583         print("File not Found, Download again")
584         DOWNLOAD AGAIN = True
585 if DOWNLOAD AGAIN:
586     for c_n in counted_classes_all.keys():
587         with urllib.request.urlopen(f"https://browser.ihtsdotools.org/snowstorm/snomed-ct/browser/MAIN/2021-01-31/concepts/{c_n}") as url:
588             snomed_data[c_n] = json.loads(url.read().decode())
589             time.sleep(0.5)
590     with open('Downloads/Github/contrastive-predictive-coding/snomed_data.json', 'w') as f:
591         json.dump(snomed_data, f)
592
593
594
595 # In[9]:
596
597
598 code_names = {}
599 for c_n in counted_classes_all.keys():
600     code_names[c_n] = {}
601     code_names[c_n]['Term'] = snomed_data[c_n]['pt']['term']
602     code_names[c_n]['Count'] = counts_all[counted_classes_all[c_n]]
603
604 code_df = pd.DataFrame.from_dict(code_names, orient='index')
605 #code_df.to_latex(os.path.join(save_path, 'tables/SnomedCodes.tex'))
606 display(code_df)
607
608
609 # ## Count visualization
610
611 # In[69]:
612
613
614 use_names = False
615 use_percentage = True

```

```

616
617
618 # ### all samples
619
620 # In[70]:
621
622
623 plt.figure(figsize=(15,10))
624 plt.tight_layout()
625 dsets = ['ptbxl', 'georgia', 'cpsc', 'cpsc2']
626 counts_combined = np.stack(list(counts.values()))
627 for i in range(0, len(dsets)):
628     dset = dsets[i]
629     cumm = [sum([counts[dset][c] for dset in dsets[0:i]]) for c in counted_classes[dset].values()]
630     ax = plt.bar(counted_classes[dset].keys(), [counts[dset][c] for c in counted_classes[dset].values()], bottom=cumm,
631                  label=f'{dset} Class Counts', alpha=0.6)
632 if use_percentage:
633     for i,p in enumerate(ax):
634         height = counts_combined.sum(axis=0)[i]
635         plt.text(x=p.get_x() + p.get_width() / 1.5, y=height+counts_combined.max()/100,
636                   s="{:2f}%".format(height/PATIENTS_TOTAL*100),
637                   ha="center", rotation=90, label="% in dataset (patient sum)")
638 if use_names:
639     plt.xticks(range(len(counted_classes_all.keys())), [code_names[c]['Term'] for c in counted_classes_all.keys()],
640                rotation='vertical')
641 else:
642     plt.xticks(range(len(counted_classes_all.keys())), list(counted_classes_all.keys()), rotation='vertical')
643 plt.ylabel("Number of samples containing this class")
644 plt.legend()
645 plt.savefig(os.path.join(save_path, 'bilder/ClassCountsPerDatasetAll.png'), bbox_inches = "tight")
646 plt.show()
647
648 # ## train set
649 #
650 # In[71]:
651
652
653 train_counts = {}
654 train_counted_classes = {}
655 train_counts['ptbxl'], train_counted_classes['ptbxl'] = count_merged_classes([ptbxl_train])
656 train_counts['georgia'], train_counted_classes['georgia'] = count_merged_classes([georgia_train])
657 train_counts['cpsc'], train_counted_classes['cpsc'] = count_merged_classes([cpsc_train])
658 train_counts['cpsc2'], train_counted_classes['cpsc2'] = count_merged_classes([cpsc2_train])
659
660
661 # In[72]:
662
663
664 plt.figure(figsize=(15,10))
665 plt.tight_layout()
666 dsets = ['ptbxl', 'georgia', 'cpsc', 'cpsc2']
667 counts_combined = np.stack(list(train_counts.values()))
668 for i in range(0, len(dsets)):
669     dset = dsets[i]
670     cumm = [sum([train_counts[dset][c] for dset in dsets[0:i]]) for c in train_counted_classes[dset].values()]
671     plt.bar(train_counted_classes[dset].keys(), [train_counts[dset][c] for c in train_counted_classes[dset].values()],
672             bottom=cumm, label=f'{dset} Class Counts (training)', alpha=0.6)
673 if use_percentage:
674     for i,p in enumerate(ax):
675         height = counts_combined.sum(axis=0)[i]
676         plt.text(x=p.get_x() + p.get_width() / 1.5, y=height+counts_combined.max()/100,
677                   s="{:2f}%".format(height/PATIENTS_TRAIN_TOTAL*100),
678                   ha="center", rotation=90, label="% in dataset (patient sum)")
679 if use_names:
680     plt.xticks(range(len(train_counted_classes_all.keys())), [code_names[c]['Term'] for c in train_counted_classes_all.keys()],
681                rotation='vertical')
682 else:
683     plt.xticks(range(len(counted_classes_all.keys())), list(counted_classes_all.keys()), rotation='vertical')
684 plt.ylabel("Number of samples containing this class")
685 plt.legend()
686 plt.savefig(os.path.join(save_path, 'bilder/ClassCountsPerDatasetTrain.png'), bbox_inches = "tight")
687 plt.show()
688
689 # ## Count Visualization by Dataset
690
691 # In[18]:
692
693 def dataframe_to_image(df, savepath):
694     with pd.option_context('display.max_rows', 300, 'display.max_columns', 7, 'display.expand_frame_repr', False):
695         html = df.render()
696         options = {
697             'zoom': 1,
698             'quality': 100
699         }
700     imgkit.from_string(html, savepath, options=options)
701

```

```

702 def dataframe_to_latex(df, savepath):
703     with open(savepath, 'w') as f:
704         f.write(df.to_latex(convert_css=True))
705
706
707 # In[19]:
708
709
710 import seaborn as sns
711 import imgkit
712 cm = sns.light_palette("seagreen", as_cmap=True)
713
714 counts_all
715 counted_classes_all
716 fractions = {}
717 for i, dset in enumerate(['ptbxl', 'georgia', 'cpsc', 'cpsc2']):
718     fractions[dset] = {}
719     for c, cidx in counted_classes[dset].items():
720         cidx_in_all = counted_classes_all[c]
721         fractions[dset][f"{{code_names[c]['Term']}} ({counts_all[cidx_in_all]})"] = counts[dset][cidx]/counts_all[
722             cidx_in_all]
723 df = pd.DataFrame.from_dict(fractions)
724 df.insert(loc=0, column='Class Snomed Code', value=list(counted_classes[dset].keys()))
725 df.insert(loc=0, column='Nr.', value=range(len(df)))
726 df.index.name = 'Class Term'
727 df = df.reset_index().set_index(['Nr.', 'Class Snomed Code', 'Class Term'])
728 # df = df.set_index("Class Snomed Code", append=True)
729 # df = df.set_index("Nr.", append=True)
730 s = df.style.background_gradient(cmap=cm, axis=1)
731 dataframe_to_latex(s, os.path.join(save_path, 'tables/count_visualization.tex'))
732
733
734
735 # ## Class label visualization
736
737 # In[68]:
738
739
740 from torch.utils.data import DataLoader, ChainDataset
741 downstream_train_dataset = ChainDataset([ptbxl_train, georgia_train, cpsc_train, cpsc2_train])
742 dl = DataLoader(downstream_train_dataset, batch_size=1, collate_fn=collate_fn)
743 occ = np.zeros((len(counted_classes_all.keys()), len(counted_classes_all.keys())))
744 patient_count = 0
745 for _, labels in dl:
746     labels = labels[0]
747     patient_count += 1
748     for l_ix in np.nonzero(labels):
749         occ[l_ix, :] += labels.numpy()
750         occ[:, l_ix] += labels.numpy()
751
752
753 # In[77]:
754
755
756 from matplotlib.colors import LogNorm
757 plt.figure(figsize=(20, 20), dpi=300)
758 #plt.title('Classes occurring together')
759 plt.tight_layout()
760 plt.imshow(occ.astype(int), norm=LogNorm())
761 if use_names:
762     plt.xticks(range(len(counted_classes_all.keys())), [code_names[c]['Term'] for c in counted_classes_all.keys()],
763                rotation='vertical')
764     plt.yticks(range(len(counted_classes_all.keys())), [code_names[c]['Term'] for c in counted_classes_all.keys()])
765 else:
766     plt.xticks(range(len(counted_classes_all.keys())), list(counted_classes_all.keys()), rotation='vertical')
767     plt.yticks(range(len(counted_classes_all.keys())), list(counted_classes_all.keys()))
768 plt.colorbar(fraction=0.046, pad=0.04, aspect=100)
769 plt.savefig(os.path.join(save_path, 'bilder/ClassOccurrences.png'), bbox_inches='tight')
770 plt.show()
771
772
773 # ## Count files for few labels
774 # In[3]:
775
776
777 crop_size = 4500
778 georgia_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/georgia_challenge/',
779                                                 window_size=crop_size, pad_to_size=crop_size, return_labels=True,
780                                                 normalize_fn=normalize_feature_scaling)
781 cpdc_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/cps2018_challenge/',
782                                                 window_size=crop_size, pad_to_size=crop_size, return_labels=
783                                                 True,
784                                                 normalize_fn=normalize_feature_scaling)
785 cpdc2_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/china_challenge', window_size=crop_size,
786                                                 pad_to_size=crop_size, return_labels=True,
787                                                 normalize_fn=normalize_feature_scaling)
788 ptbxl_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/ptbxl_challenge', window_size=crop_size,
789                                                 pad_to_size=crop_size, return_labels=True),

```

```

789     normalize_fn=normalize_feature_scaling)
790 nature = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/nature_database', window_size=crop_size,
791                                         pad_to_size=crop_size, return_labels=True,
792                                         normalize_fn=normalize_feature_scaling)
793 ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.generate_datasets_from_split_file()
794 georgia_train, georgia_val, t2 = georgia_challenge.generate_datasets_from_split_file()
795 cpsc_train, cpsc_val, t3 = cpsc_challenge.generate_datasets_from_split_file()
796 cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.generate_datasets_from_split_file()
797 total_train_file_sum = sum(map(len, [ptbxl_train.files, georgia_train.files, cpsc_train.files, cpsc2_train.files]))
798 total_file_sum = sum(map(len, map(lambda x: x.files, [ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2,
799                                         cpsc_train, cpsc_val, t3, cpsc2_train, cpsc2_val, t4])))
800
801 # In[4]:
802
803
804 fractions = {}
805 few_labels_splits_filenames = ['train-test-splits-fewer-labels0.001.txt',
806     'train-test-splits-fewer-labels0.005.txt',
807     'train-test-splits-fewer-labels0.05.txt',
808     'train-test-splits-fewer-labels0.01.txt',
809     'train-test-splits-fewer-labels0.10.txt',
810     'train-test-splits-fewer-labels14.txt',
811     'train-test-splits-fewer-labels20.txt',
812     'train-test-splits-fewer-labels30.txt',
813     'train-test-splits-fewer-labels40.txt',
814     'train-test-splits-fewer-labels50.txt',
815     'train-test-splits-fewer-labels60.txt',
816     'train-test-splits.txt',
817     'train-test-splits_min_cut10.txt',
818     'train-test-splits_min_cut25.txt',
819     'train-test-splits_min_cut50.txt',
820     'train-test-splits_min_cut100.txt',
821     'train-test-splits_min_cut150.txt',
822     'train-test-splits_min_cut200.txt']
823 for few_labels_splits_filename in few_labels_splits_filenames:
824     ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
825     georgia_train, georgia_val, t2 = georgia_challenge.generate_datasets_from_split_file(ttsfile=
826         few_labels_splits_filename)
827     cpsc_train, cpsc_val, t3 = cpsc_challenge.generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
828     cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
829     file_sum = sum(map(len, [ptbxl_train.files, georgia_train.files, cpsc_train.files, cpsc2_train.files]))
830     fractions[few_labels_splits_filename] = file_sum/total_file_sum
831
832
833 # In[5]:
834
835
836 fractions
837
838
839 # ## Fewer Labels
840
841 # In[67]:
842
843
844 few_labels_splits_filename = 'train-test-splits-fewer-labels40.txt'
845
846 crop_size = 4500
847 georgia_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/georgia_challenge',
848                                                 window_size=crop_size, pad_to_size=crop_size, return_labels=True,
849                                                 normalize_fn=normalize_feature_scaling)
850 cpsc_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/cpsc2018_challenge',
851                                                 window_size=crop_size, pad_to_size=crop_size, return_labels=
852         True,
853                                                 normalize_fn=normalize_feature_scaling)
854 cpsc2_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/china_challenge', window_size=crop_size,
855                                                 pad_to_size=crop_size, return_labels=True,
856                                                 normalize_fn=normalize_feature_scaling)
857 ptbxl_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/ptbxl_challenge', window_size=crop_size,
858                                                 pad_to_size=crop_size, return_labels=True,
859                                                 normalize_fn=normalize_feature_scaling)
860 nature = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/nature_database', window_size=crop_size,
861                                                 pad_to_size=crop_size, return_labels=True,
862                                                 normalize_fn=normalize_feature_scaling)
863 ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
864 georgia_train, georgia_val, t2 = georgia_challenge.generate_datasets_from_split_file(ttsfile=
865     few_labels_splits_filename)
866 cpsc_train, cpsc_val, t3 = cpsc_challenge.generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
867 cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
868
869 # In[30]:
870
871
872 filter_update_classes_by_count([ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3,
873     cpsc2_train, cpsc2_val, t4], 1)
874 print('Classes after last update', len(ptbxl_train.classes), ptbxl_train.classes)

```

```

874 counts_all, counted_classes_all = count_merged_classes([ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2,
875     cpsc_train, cpsc_val, t3, cpsc2_train, cpsc2_val, t4])
876 counts = {}
877 counted_classes = {}
878 counts['ptbxl'], counted_classes['ptbxl'] = count_merged_classes([ptbxl_train])
879 counts['georgia'], counted_classes['georgia'] = count_merged_classes([georgia_train])
880 counts['cpsc'], counted_classes['cpsc'] = count_merged_classes([cpsc_train])
881 counts['cpsc2'], counted_classes['cpsc2'] = count_merged_classes([cpsc2_train])
882
883
884 # In[31]:
885
886
887 plt.figure(figsize=(15,10))
888 plt.title("Train data distribution")
889 dsets = ['ptbxl', 'georgia', 'cpsc', 'cpsc2']
890 for i, dset in enumerate(dsets):
891     cumm = [sum([counts[dset][c] for dset in dsets[0:i]]) for c in counted_classes[dset].values()]
892     plt.bar(counted_classes[dset].keys(), [counts[dset][c] for c in counted_classes[dset].values()], bottom=cumm,
893             label=f'{dset} Class Counts', alpha=0.6)
894 if use_names:
895     plt.xticks(range(len(counted_classes_all.keys())), [code_names[c]['Term'] for c in counted_classes_all.keys()],
896                rotation='vertical')
897 else:
898     plt.xticks(range(len(counted_classes_all.keys())), list(counted_classes_all.keys()), rotation='vertical')
899 plt.legend()
900 plt.savefig(os.path.join(save_path, 'bilder/ClassCountsPerDatasetFew.png'))
901 plt.show()
902
903 # ## Fewer Labels (min_cut)
904 # In[75]:
905
906
907 few_labels_splits_filename = '05_07_21-15-17train-test-splits_min_cut100.txt'
908
909 crop_size = 4500
910 georgia_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/georgia_challenge/',
911                                                 window_size=crop_size, pad_to_size=crop_size, return_labels=True,
912                                                 normalize_fn=normalize_feature_scaling)
913 cpsc_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/cpsc2018_challenge/',
914                                                 window_size=crop_size, pad_to_size=crop_size, return_labels=
915                                                 True,
916                                                 normalize_fn=normalize_feature_scaling)
917 cpsc2_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/china_challenge', window_size=crop_size,
918                                                 pad_to_size=crop_size, return_labels=True,
919                                                 normalize_fn=normalize_feature_scaling)
920 ptbxl_challenge = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/ptbxl_challenge', window_size=crop_size,
921                                                 pad_to_size=crop_size, return_labels=True,
922                                                 normalize_fn=normalize_feature_scaling)
923 nature = ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/nature_database', window_size=crop_size,
924                                                 pad_to_size=crop_size, return_labels=True,
925                                                 normalize_fn=normalize_feature_scaling)
926
927 ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
928 georgia_train, georgia_val, t2 = georgia_challenge.generate_datasets_from_split_file(ttsfile=
929     few_labels_splits_filename)
930 cpsc_train, cpsc_val, t3 = cpsc_challenge.generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
931 cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.generate_datasets_from_split_file(ttsfile=few_labels_splits_filename)
932
933 # In[76]:
934
935 filter_update_classes_by_count([ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3,
936     cpsc2_train, cpsc2_val, t4], 1)
937 print('Classes after last update', len(ptbxl_train.classes), ptbxl_train.classes)
938 counts_all, counted_classes_all = count_merged_classes([ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2,
939     cpsc_train, cpsc_val, t3, cpsc2_train, cpsc2_val, t4])
940
941 counts = {}
942 counted_classes = {}
943 counts['ptbxl'], counted_classes['ptbxl'] = count_merged_classes([ptbxl_train])
944 counts['georgia'], counted_classes['georgia'] = count_merged_classes([georgia_train])
945 counts['cpsc'], counted_classes['cpsc'] = count_merged_classes([cpsc_train])
946 counts['cpsc2'], counted_classes['cpsc2'] = count_merged_classes([cpsc2_train])
947
948 # In[63]:
949
950 plt.figure(figsize=(15,10))
951 plt.title("Distribution of training data")
952 for i, dset in enumerate(['ptbxl', 'georgia', 'cpsc', 'cpsc2']):
953     plt.bar(counted_classes[dset].keys(), [counts[dset][c] for c in counted_classes[dset].values()], label=f'{dset}
954         Class Counts', alpha=0.6)
955 if use_names:

```

```

955     plt.xticks(range(len(counted_classes_all.keys())), [code_names[c]['Term'] for c in counted_classes_all.keys()], rotation='vertical')
956 else:
957     plt.xticks(range(len(counted_classes_all.keys())), list(counted_classes_all.keys()), rotation='vertical')
958 plt.legend()
959 plt.savefig(os.path.join(save_path, 'bilder/ClassCountsPerDatasetFew.png'))
960 plt.show()
961
962
963 # In[77]:
964
965
966 patient_counts = sum(map(lambda x: len(x.files), [ptbxl_train, georgia_train, cpsc_train, cpsc2_train]))
967
968 patient_counts/PATIENTS_TRAIN_TOTAL
970
971
972 # In[48]:
973
974
975 DESIRED_SIZE = class_total*0.1
976 print(DESIRED_SIZE)
977 sort_idx = np.argsort(class_counts)
978 sorted_class_counts = class_counts[sort_idx]
979 total = 0
980 last_size = 0
981 i = 0
982 extracted_counts = np.zeros(len(sorted_class_counts)).astype(int)
983 if DESIRED_SIZE/sorted_class_counts[0] > len(sorted_class_counts):
984     while total + (len(class_counts)-i)*last_size < DESIRED_SIZE:
985         last_size = sorted_class_counts[i]
986         total += last_size
987         extracted_counts[sort_idx[i]] = last_size
988         i += 1
989 rest_needed = int((DESIRED_SIZE-sum(extracted_counts))/(len(class_counts)-i))
990 extracted_counts[np.where(extracted_counts==0)[0]] = rest_needed
991 print(extracted_counts)
992
993
994 # In[ ]:
995
996
997 from collections import defaultdict
998 class_counts_dset = np.empty((4, len(class_counts))).astype(int)
999 counted_classes_dset = np.empty((4, len(class_counts)))
1000 for i, dset in enumerate([ptbxl_train, georgia_train, cpsc_train, cpsc2_train]):
1001     c_count, counted_c = count_merged_classes([dset])
1002     class_counts_dset[i] = np.array(c_count)
1003 class_counts_dset
1004 dsets = [ptbxl_train, georgia_train, cpsc_train, cpsc2_train]
1005 buckets = np.zeros((4, len(class_counts))).astype(int)
1006 sort_idx = np.argsort(class_counts)
1007
1008 for ix in sort_idx:
1009     #find this label in files.
1010     dset_order = np.argsort(class_counts_dset[:, ix])
1011     n_per_dataset = 0
1012     have = 0
1013     for i, dset_i in enumerate(dset_order):
1014         n_per_dataset = int(np.ceil((extracted_counts[ix] - have)/(len(dsets)-i)))
1015         cc_in_dset = class_counts_dset[dset_i, ix]
1016         take_n_files = min(cc_in_dset, n_per_dataset)
1017         buckets[dset_i, ix] = take_n_files
1018         have += take_n_files
1019
1020 sum(buckets), buckets
1021
1022
1023 # In[50]:
1024
1025
1026 np.sum(buckets, axis=0) - extracted_counts
1027
1028
1029 # In[130]:
1030
1031
1032 np.sum(class_counts_dset, axis=0)
1033
1034
1035 # In[ ]:
1036
1037
1038 def min_splits(DESIRED_SIZE):
1039     print(DESIRED_SIZE)
1040     sort_idx = np.argsort(class_counts)
1041     sorted_class_counts = class_counts[sort_idx]
1042     total = 0
1043     last_size = 0

```

```

1044     i = 0
1045     extracted_counts = []
1046     if DESIRED_SIZE/sorted_class_counts[0] > len(sorted_class_counts):
1047         while total + (len(class_counts)-i)*last_size < DESIRED_SIZE:
1048             last_size = sorted_class_counts[i]
1049             total += last_size
1050             extracted_counts.append(last_size)
1051             i += 1
1052     rest_needed = int((DESIRED_SIZE-sum(extracted_counts))/(len(class_counts)-len(extracted_counts)))
1053     extracted_counts = extracted_counts + [rest_needed] * (len(class_counts) - len(extracted_counts))
1054     extracted_counts
1055     assert train_fraction+val_fraction+test_fraction <= 1
1056     class_buckets = [[] for x in range(len(self.classes))] #Creates classes buckets
1057     for i, f in enumerate(self.files):
1058         label = self._read_header_labels(f)
1059         label_idx = np.argwhere(label).flatten()
1060         for l in label_idx: #can return more than one (multi-label)
1061             class_buckets[l].append(f) #Put this file into class l bucket
1062     train_files, val_files, test_files = [], [], []
1063     sorted_idx = list(sorted(range(len(class_buckets)), key=lambda x: len(class_buckets[x]))) #make index sorted by
1064     class count low->high
1065     print(sorted_idx)
1066     while len(sorted_idx) > 0:
1067         idx = sorted_idx[0]
1068         shuffle(class_buckets[idx])
1069         b = class_buckets[idx]
1070         c_N = len(b)
1071         train_slice = slice(0, int(train_fraction * c_N))
1072         val_slice = slice(train_slice.stop, train_slice.stop + int(val_fraction * c_N))
1073         test_slice = slice(val_slice.stop, val_slice.stop + int(test_fraction * c_N))
1074         train_files += b[train_slice]
1075         val_files += b[val_slice]
1076         test_files += b[test_slice]
1077         used_set = set(b[train_slice]+b[val_slice]+b[test_slice])
1078         for j in sorted_idx[1:]:
1079             class_buckets[j] = [x for x in class_buckets[j] if x not in used_set] #REMOVE THIS FILE FROM ALL OTHER
1080             BUCKETS
1081             sorted_idx = list(sorted(sorted_idx[1:], key=lambda x: len(class_buckets[x]))) #sort again (removal may change
1082             order)
1083     train_files = list(set(train_files))
1084     val_files = list(set(val_files))
1085     test_files = list(set(test_files))
1086     if save:
1087         p = save_path_overwrite or self.BASE_DIR
1088         fname = filename_overwrite or 'train-test-splits.txt'
1089         save_train_test_split(os.path.join(p, fname), train_files, val_files, test_files)
1090     return train_files, val_files, test_files
1091
1092
1093
1094 # In[45]:
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
from torch.utils.data import DataLoader, ChainDataset
whole_dataset = ChainDataset([ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3,
    cpsc2_train, cpsc2_val, t4])
dl = DataLoader(whole_dataset, batch_size=1, collate_fn=collate_fn)
occ = np.zeros(len(counted_classes_all.keys()), len(counted_classes_all.keys())))
patient_count = 0
all_labels = []
for _, labels in dl:
    labels = labels[0]
    patient_count += 1
    all_labels.append(labels)
all_labels = np.concatenate(all_labels).reshape((-1, 67))

# In[53]:
pearson = np.zeros((67, 67))
for cix1 in range(67):
    for cix2 in range(67):
        pearson[cix1, cix2] = stats.pearsonr(all_labels[:, cix1], all_labels[:, cix2])[0]
        #pearson[cix1, cix2] = np.cov(all_labels[:, cix1], all_labels[:, cix2])[0, 1]/(np.std(all_labels[:, cix1])*np.
        std(all_labels[:, cix2]))

# In[54]:
pd.DataFrame(pearson, columns=ptbxl_train.classes, index=ptbxl_train.classes)

# In[78]:
import matplotlib as mpl
plt.figure(figsize=(20, 20), dpi=300)
#plt.title('Classes occuring together')

```

```

1129 plt.tight_layout()
1130 plt.imshow(pearson, cmap=cmap, norm=mpl.colors.Normalize(vmin=-1, vmax=1))
1131 if use_names:
1132     plt.xticks(range(len(counted_classes_all.keys())), [code_names[c]['Term'] for c in counted_classes_all.keys()],
1133                 rotation='vertical')
1133     plt.yticks(range(len(counted_classes_all.keys())), [code_names[c]['Term'] for c in counted_classes_all.keys()])
1134 else:
1135     plt.xticks(range(len(counted_classes_all.keys())), list(counted_classes_all.keys()), rotation='vertical')
1136     plt.yticks(range(len(counted_classes_all.keys())), list(counted_classes_all.keys()))
1137 plt.colorbar(fraction=0.046, pad=0.04, aspect=100)
1138 plt.savefig(os.path.join(save_path, 'bilder/ClassOccurrencesCorrelation.png'), bbox_inches='tight')
1139 plt.show()

```

B.7.5 jupyter_notebooks -> Export Code.py (code)

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 import os
8
9
10 # In[2]:
11
12
13 def walkp(b, replace='', skip=[]):
14     ps = []
15     for root, folders, files in os.walk(b):
16         rest = root.replace(replace, "").lstrip(os.path.sep)
17         if rest.startswith('.'):
18             continue
19         ps = [rest]
20         for fo in folders:
21             if not fo in skip and not fo.startswith('.'):
22                 ps.append(walkp(os.path.join(root, fo), b, skip=skip))
23         fis = []
24         for fi in files:
25             if not fi in skip and not fi.startswith('.'):
26                 fis.append(fi)
27             ps.append(fis)
28         break
29     return ps
30
31 def walkd(b, replace='', skip=[]):
32     ps = {}
33     for root, folders, files in os.walk(b):
34         rest = root.replace(replace, "").lstrip(os.path.sep)
35         if rest.startswith('.'):
36             continue
37         ps[rest] = []
38         for fo in folders:
39             if not fo in skip and not fo.startswith('.'):
40                 ps[rest].append(walkd(os.path.join(root, fo), b, skip=skip))
41         fis = []
42         for fi in files:
43             if not fi in skip and not fi.startswith('.'):
44                 fis.append(fi)
45             ps[rest].append(fis)
46         break
47     return ps
48
49
50 # In[3]:
51
52
53 base = '/home/julian/Downloads/Github/contrastive-predictive-coding/'
54 l = walkd(base, base, skip=['models', 'models_symbolic_links', '__pycache__', 'models_evaluated_filtered_csv', 'logs',
55                           'images', 'external'])
56
57 # In[4]:
58
59
60 #print(str(l).replace(',', '\n').replace('\"', '').replace("_", '\\_'))
61
62
63 # In[5]:
64
65
66 def print_latex_code(base, replace='', folder_filter_fns=[], file_filter_fns=[], subsection_level=0):
67     for root, folders, files in os.walk(base):
68         if not (folders == [] and files == []):
69             long_name = ' -> '.join(base.replace(replace, '').split(os.path.sep))
70             print(make_section(subsection_level, os.path.basename(base), long_name)+` (folder)`)

```

```

71         #print('\t'*subsection_level)
72         for fo in sorted(folders):
73             if not any([fn(fo) for fn in folder_filter_fns]):
74                 print_latex_code(os.path.join(root, fo), replace, folder_filter_fns, file_filter_fns, subsection_level
+1)
75             for fi in sorted(files):
76                 if not any([fn(fi) for fn in file_filter_fns]):
77                     long_name = ' -> '.join(os.path.join(base, fi).replace(replace, '') .split(os.path.sep))
78                     print(make_section(subsection_level+1, fi, long_name) +" "+ (code))
79                     #print("\t"+(subsection_level+1)+"code")
80                     if fi.endswith('.pdf'):
81                         print('\t'+(subsection_level+1)+
82                               f'\\includepdf[pages=-]{{{os.path.join(base,fi)}}}')
83                     else:
84                         print('\t'+(subsection_level+1)+
85                               f'\\listinputlisting[language=Python]{{{os.path.join(base,fi)}}}')
86                 break
87
88     def make_section(subsection_level, short_name, long_name):
89         sec = '\t'*subsection_level+'\n\\noindent\\'
90         if subsection_level < 3:
91             sec += 'sub'+subsection_level+'section'
92         elif subsection_level == 3:
93             sec += 'paragraph'
94         elif subsection_level == 4:
95             sec += ' subparagraph'
96         elif subsection_level > 4:
97             return ''
98         short_name = short_name.replace('_', '\\_')
99         long_name = long_name.replace('_', '\\_')
100        return sec+f'{{short_name}}{{long_name}})'
101
102
103 # In[6]:
104
105
106 folder_filter_fns = [
107     lambda x: x.startswith('.'),
108     lambda x: 'pycache' in x,
109     lambda x: x == 'models',
110     lambda x: x == 'models_symbolic_links',
111     lambda x: x == 'data_output',
112     lambda x: x == 'images',
113     lambda x: 'TCN' in x and os.path.split(x)[-2]=='TCN'
114 ]
115 file_filter_fns = [
116     lambda x: x.startswith('.'),
117     lambda x: not x.endswith('.py'), # or x.endswith('.pdf')),
118     lambda x: x == '__init__.py'
119 ]
120
121 print_latex_code(base, base, folder_filter_fns, file_filter_fns, 0)
122
123
124 # In[20]:
125
126
127 import glob
128 for f in glob.glob('*.ipynb'):
129     get_ipython().system('jupyter nbconvert --to script "$f"')
130
131
132
133 # In[19]:
134
135
136 glob.glob('*.ipynb')
137
138
139 # In[ ]:
```

B.7.6 jupyter_notebooks -> Filter Attributes DataFrame.py (code)

```

1#!/usr/bin/env python
2# coding: utf-8
3
4# In[1]:
5
6
7import pandas as pd
8import numpy as np
9import datetime
10import re
11import os
12from functools import reduce
13pd.set_option('display.max_columns', None)
```

```

14 pd.set_option('display.max_colwidth', None)
15
16
17 # In[2]:
18
19
20 convert = lambda text: int(text) if text.isdigit() else text.lower()
21 alphanum_key = lambda key: [[convert(c) for c in re.split('([0-9]+)', k)] for k in key]
22 def filter_df_with_subset(dataframe, isin_subset={}, isnnotin_subset={}):
23     if isin_subset or isnnotin_subset:
24         bit_masks = [dataframe[k].isin(v) for k,v in isin_subset.items()]
25         bit_masks += [-dataframe[k].isin(v) for k,v in isnnotin_subset.items()]
26         combined_mask = reduce(lambda x, y: x & y, bit_masks)
27         return dataframe[combined_mask].dropna(axis=1, how='all')
28     else:
29         return dataframe
30
31
32 # In[3]:
33
34
35 all_df = pd.read_csv("/home/julian/Desktop/All_attributes_with_scores.csv", index_col=0, parse_dates=['train timestamp'])
36 all_df = all_df.dropna(subset=['train timestamp', 'Uses Classweights'])
37 all_df = all_df[['train timestamp', 'Model Path'] + [c for c in all_df.columns if not c in ['train timestamp', 'Model Path', 'micro', 'macro']] + ['macro', 'micro']]
38 all_df[["Train Normalization Function", "Test Normalization Function"]] = all_df[["Train Normalization Function", "Test Normalization Function"]].apply(lambda s:s.str.replace("", ""))
39 all_df[['model']] = all_df[['model']].apply(lambda s:s.str.replace(":d*", "", regex=True))
40 all_df = all_df.drop(columns=['train timestamp'])
41 #all_df['CPC Sampling Mode'] = all_df['CPC Sampling Mode'].apply(lambda x: 'all' if x == 'crossentropy' else x)
42
43
44 # In[4]:
45
46
47 OUTPUT_PATH = "/home/julian/Downloads/Github/contrastive-predictive-coding/models_evaluated_filtered_csv/"
48
49
50 # # All std_scaled models
51
52 # In[16]:
53
54
55 subset = {
56     'Train Normalization Function': ['normalize_std_scaling'],
57     'Test Normalization Function': ['normalize_std_scaling']
58 }
59 df = filter_df_with_subset(all_df, isin_subset=subset)
60
61
62 # # All normal new BL Models
63
64 # In[111]:
65
66
67 subset = {
68     'Train Normalization Function': ['normalize_std_scaling'],
69     'Test Normalization Function': ['normalize_std_scaling'],
70     'uses Max Pool': [False, True],
71     'Train splitsfile': ['train-test-splits'],
72 }
73 bl_df = filter_df_with_subset(all_df, subset)
74 len(bl_df)
75
76
77 # In[112]:
78
79
80 d = {}
81 for c in bl_df.columns:
82     if c != 'macro' and c != 'train timestamp' and c != 'Model Path':
83         d[c] = [v for v in bl_df[c].unique() if not pd.isna(v)]
84
85 print('subset options:', d)
86
87
88 # In[113]:
89
90
91 g = bl_df.groupby(by=[cn for cn in bl_df.columns if not cn in ['Model Path', 'micro', 'macro']], axis=0, dropna=False)
92 result = g.max().reset_index(drop=False).sort_values(by=['model'], key=alphanum_key).sort_index(level='model', key=
93     alphanum_key)
94 result = result[['model', 'micro', 'macro']]
95 result.to_csv(os.path.join(OUTPUT_PATH, "all_BL_std.csv"), index=False)
96 result
97
98
99 # # All CPC models

```

```

100 # In[114]:
101
102
103
104 subset = {
105     'normalizes_latents': [True, False],
106     'Train splitsfile':['train-test-splits'],
107 }
108 all_cpc_df = filter_df_with_subset(all_df, subset).sort_values(by='macro', ascending=False)
109 d = {}
110 for c in all_cpc_df.columns:
111     if c != 'micro' and c != 'macro' and c != 'train timestamp' and c != 'Model Path':
112         d[c] = [v for v in all_cpc_df[c].unique() if not pd.isna(v)]
113
114 print('subset options:', d)
115
116
117 # In[115]:
118
119
120 all_cpc_df
121
122
123 # # CPC no changes, no pretrain 120
124
125 # In[116]:
126
127
128 subset = {
129     'Train Normalization Function':['normalize_std_scaling'],
130     'Test Normalization Function':['normalize_std_scaling'],
131     'Train splitsfile':['train-test-splits'],
132     'Uses Classweights': [True],
133     'normalizes latents': [False],
134     'Predictor':['cpc_predictor_v0'],
135     'Encoder': ['cpc_encoder_v0'],
136     'Autoregressive': ['cpc_autoregressive_v0'],
137     'Downstream Epochs': [120], #140=120
138 }
139 not_subset = {
140 }
141 selection = ['model', 'Freeze CPC', 'strided', 'Downstream Model', 'micro', 'macro']
142
143 cpc_df = filter_df_with_subset(all_df, subset, not_subset)
144 print(len(cpc_df))
145 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'CPC Type', 'CPC Sampling Mode',
146     'micro', 'macro']], axis=0, dropna=False)
147 result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=selection).
148     reset_index(drop=True)
149 result = result[selection]
150 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_nopretrain_nochanges_120_std.csv"), index=False)
151 result
152
153 # # ALL standard CPC Models (like in the paper) frozen
154 #
155 # In[131]:
156
157
158 subset = {
159     'Train Normalization Function':['normalize_std_scaling'],
160     'Test Normalization Function':['normalize_std_scaling'],
161     'Uses Classweights': [True],
162     'Train splitsfile':['train-test-splits'],
163     'Freeze CPC':[True],
164     'normalizes latents': [False],
165     'Predictor':['cpc_predictor_v0'],
166     'Encoder': ['cpc_encoder_v0'],
167     'Autoregressive': ['cpc_autoregressive_v0'],
168     'Downstream Model': ['cpc_downstream_latent_maximum', 'cpc_downstream_latent_average', 'cpc_downstream_twolinear_v2'],
169     #'CPC Type':['cpc_intersect'],
170     'Downstream Epochs': [20],
171     "Pretrain Epochs": [100]
172 }
173 selection = ['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Downstream Model', 'micro', 'macro']
174 cpc_df = filter_df_with_subset(all_df, subset)
175 print(len(cpc_df))
176 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro', 'uses Context', 'uses
177     Latents']], axis=0, dropna=False)
178 result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=selection).
179     reset_index(drop=True)
180 result = result[selection]
181 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_nochanges_frozen_std.csv"), index=False)
182 display(result)
183
184 # # ALL standard CPC Models (like in the paper) frozen NO PRETRAIN
185

```

```

185 # In[132]:
186
187 subset = {
188     'Train Normalization Function': ['normalize_std_scaling'],
189     'Test Normalization Function': ['normalize_std_scaling'],
190     'Uses Classweights': [True],
191     'Train splitsfile': ['train-test-splits'],
192     'Freeze CPC': [True],
193     'Uses Classweights': [True],
194     'normalizes latents': [False],
195     'Predictor': ['cpc_predictor_v0'],
196     'Encoder': ['cpc_encoder_v0'],
197     'Autoregressive': ['cpc_autoregressive_v0'],
198     'Downstream Model': ['cpc_downstream_latent_maximum', 'cpc_downstream_latent_average', 'cpc_downstream_twolinear_v2'],
199     #'CPC Type': ['cpc_intersect'],
200     'Downstream Epochs': [20],
201     "Pretrain Epochs": [0]
202 }
203
204 selection = ['model', 'Freeze CPC', 'strided', 'Downstream Model', 'micro', 'macro']
205 cpc_df = filter_df_with_subset(all_df, subset)
206 print(len(cpc_df))
207 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro', 'uses Context', 'uses Latents']], axis=0, dropna=False)
208 result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=selection).
    reset_index(drop=True)
209 result = result[selection]
210 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_nochanges_frozen_nopretrain_std.csv"), index=False)
211 display(result)
212
213
214 # # ALL standard CPC Models (like in the paper) unfrozen
215
216 # In[119]:
217
218
219 subset = {
220     'Train Normalization Function': ['normalize_std_scaling'],
221     'Test Normalization Function': ['normalize_std_scaling'],
222     'normalizes latents': [False],
223     'Uses Classweights': [True, False],
224     'Train splitsfile': ['train-test-splits'],
225     'Freeze CPC': [False],
226     'Uses Classweights': [True],
227     'normalizes latents': [False],
228     'Predictor': ['cpc_predictor_v0'],
229     'Encoder': ['cpc_encoder_v0'],
230     'Autoregressive': ['cpc_autoregressive_v0'],
231     'Downstream Model': ['cpc_downstream_latent_maximum', 'cpc_downstream_latent_average', 'cpc_downstream_twolinear_v2'],
232     'Downstream Epochs': [20],
233     "Pretrain Epochs": [100]
234 }
235 selection = ['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Downstream Model', 'micro', 'macro']
236 cpc_df = filter_df_with_subset(all_df, subset)
237 print(len(cpc_df))
238 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro', 'uses Context', 'uses Latents']], axis=0, dropna=False)
239 result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=selection).
    reset_index(drop=True)
240 result = result[selection]
241 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_nochanges_unfrozen_std.csv"), index=False)
242 display(result)
243
244
245 # # ALL standard CPC Models (like in the paper) unfrozen (40 epochs)
246
247 # In[120]:
248
249
250 subset = {
251     'Train Normalization Function': ['normalize_std_scaling'],
252     'Test Normalization Function': ['normalize_std_scaling'],
253     'normalizes latents': [False],
254     'Uses Classweights': [True, False],
255     'Train splitsfile': ['train-test-splits'],
256     'Freeze CPC': [False],
257     'Uses Classweights': [True],
258     'normalizes latents': [False],
259     'Predictor': ['cpc_predictor_v0'],
260     'Encoder': ['cpc_encoder_v0'],
261     'Autoregressive': ['cpc_autoregressive_v0'],
262     'Downstream Model': ['cpc_downstream_latent_maximum', 'cpc_downstream_latent_average', 'cpc_downstream_twolinear_v2'],
263     'Downstream Epochs': [40],
264     "Pretrain Epochs": [100]
265 }
266 selection = ['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Downstream Model', 'micro', 'macro']
267 cpc_df = filter_df_with_subset(all_df, subset)

```

```

268 print(len(cpc_df))
269 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro']], axis=0, dropna=False)
270 result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=selection).
    reset_index(drop=True)
271 result = result[selection]
272 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_nochanges_unfrozen_40_std.csv"), index=False)
273 display(result)
274
275
276 # # Additional changes
277
278 # In[130]:
279
280
281 subset = {
282     'Train Normalization Function': ['normalize_std_scaling'],
283     'Test Normalization Function': ['normalize_std_scaling'],
284     'normalizes latents': [False],
285     'Uses Classweights': [True],
286     'Train splitsfile': ['train-test-splits'],
287     'Predictor': ['cpc_predictor_v0'],
288     'Encoder': ['cpc_encoder_v0'],
289     'Autoregressive': ['cpc_autoregressive_v0'],
290     'Downstream Model': ['cpc_downstream_latent_maximum', 'cpc_downstream_latent_average', 'cpc_downstream_twolinear_v2'],
291     'Downstream Epochs': [40],
292     "Pretrain Epochs": [100]
293 }
294 selection = ['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Downstream Model', 'micro', 'macro']
295 cpc_df = filter_df_with_subset(all_df, subset)
296 print(len(cpc_df))
297 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro', 'uses Context', 'uses
    latents']], axis=0, dropna=False)
298 result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=selection).
    reset_index(drop=True)
299 result = result[selection]
300 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_nochanges_40epochs_std.csv"), index=False)
301 display(result)
302
303
304 # # Normalized latents standard(relaxed Downstream to 40 epochs)
305
306 # In[122]:
307
308
309 subset = {
310     'Train Normalization Function': ['normalize_std_scaling'],
311     'Test Normalization Function': ['normalize_std_scaling'],
312     'Train splitsfile': ['train-test-splits'],
313     'normalizes latents': [True],
314     'Predictor': ['cpc_predictor_v0'],
315     'Encoder': ['cpc_encoder_v0'],
316     'Autoregressive': ['cpc_autoregressive_v0'],
317     'Downstream Epochs': [20, 40],
318     "Pretrain Epochs": [100]
319 }
320 notsubset = {
321     'Downstream Epochs': [100, 120]
322 }
323 selection = ['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Downstream Epochs', 'Downstream Model', 'micro',
    'macro']
324 cpc_df = filter_df_with_subset(all_df, subset, notsubset)
325 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro', 'Encoder', 'Encoder',
    'Autoregressive', 'Predictor', 'Downstream Model']], axis=0, dropna=False)
326 result = result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=
    selection).reset_index(drop=True)
327 result = result[selection]
328 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_somuchanges_normalized_both.csv"), index=False)
329 display(result)
330
331
332 # # Normalized latents all(relaxed Downstream to 40 epochs)
333
334 # In[123]:
335
336
337 subset = {
338     'Train Normalization Function': ['normalize_std_scaling'],
339     'Test Normalization Function': ['normalize_std_scaling'],
340     'Train splitsfile': ['train-test-splits'],
341     'normalizes latents': [True],
342     'Downstream Epochs': [20, 40],
343     "Pretrain Epochs": [100]
344 }
345 notsubset = {
346     'Downstream Epochs': [100, 120],
347     "Downstream Model": ['cpc_downstream_only']
348 }

```

```

349 selection = ['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Encoder', 'Autoregressive', 'Predictor', 'Downstream Model', 'micro', 'macro']
350 cpc_df = filter_df_with_subset(all_df, subset, notsubset)
351 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro', 'Encoder', 'Autoregressive', 'Predictor', 'Downstream Epochs', "Downstream Model"]], axis=0, dropna=False)
352 result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=selection).
    reset_index(drop=True)
353 result = result[selection]
354 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_somechanges_normalized_all.csv"), index=False)
355 display(result)
356
357
358 # # Autoregressive hidden (standard)
359 # ignores 'uses Context', 'uses Latents', 'Downstream Model' in categoric groups
360
361 # In[124]:
362
363
364 subset = {
365     'Train Normalization Function': ['normalize_std_scaling'],
366     'Test Normalization Function': ['normalize_std_scaling'],
367     'Train splitsfile': ['train-test-splits'],
368     'strided': [True, False],
369     'normalizes latents': [False],
370     'Downstream Epochs': [20, 40],
371     'Predictor': ['cpc_predictor_v0'],
372     'Encoder': ['cpc_encoder_v0'],
373     'Pretrain Epochs': [100],
374     'Autoregressive': ['cpc_autoregressive_hidden'],
375     #'Freeze CPC': [True]
376 }
377 notsubset = {
378     'Downstream Epochs': [100, 120],
379     'CPC Sampling Mode': ['crossentropy-nocontext'],
380     #'CPC Type': ['cpc_intersect_manylatents']
381 }
382 selection = ['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Autoregressive', 'Downstream Epochs', 'Downstream Model', 'micro', 'macro']
383 cpc_df = filter_df_with_subset(all_df, subset, notsubset)
384 print(len(cpc_df))
385 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro', 'uses Context', 'uses Latents', 'Downstream Model']], axis=0, dropna=False)
386 result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=selection).
    reset_index(drop=True)
387 result = result[selection]
388 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_somechanges_hidden.csv"), index=False)
389 display(result)
390
391
392 # # Encoder likev8 (standard)
393 # ignores 'uses Context', 'uses Latents', 'Downstream Model' in categoric groups
394
395 # In[125]:
396
397
398 subset = {
399     'Train Normalization Function': ['normalize_std_scaling'],
400     'Test Normalization Function': ['normalize_std_scaling'],
401     'Train splitsfile': ['train-test-splits'],
402     'normalizes latents': [False],
403     'Predictor': ['cpc_predictor_v0'],
404     'Encoder': ['cpc_encoder_v0'],
405     'Autoregressive': ['cpc_autoregressive_v0'],
406     'Downstream Epochs': [20, 40],
407     'Pretrain Epochs': [100],
408     'Encoder': ['cpc_encoder_likev8'],
409 }
410 notsubset = {
411     'Downstream Epochs': [100, 120],
412     'CPC Type': ['crossentropy-nocontext'],
413     'CPC Sampling Mode': ['same', 'multisame', 'all']
414     #'CPC Type': ['cpc_intersect_manylatents']
415 }
416 selection = ['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Encoder', 'Downstream Epochs', 'Downstream Model', 'micro', 'macro']
417 cpc_df = filter_df_with_subset(all_df, subset, notsubset)
418 len(cpc_df)
419 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro', 'uses Context', 'uses Latents', 'Downstream Model']], axis=0, dropna=False)
420 result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=selection).
    reset_index(drop=True)
421 result = result[selection]
422 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_somechanges_likev8.csv"), index=False)
423 display(result)
424
425
426 # # Encoder likev8 (unfrozen)
427 # ignores 'uses Context', 'uses Latents', 'Downstream Model' in categoric groups
428
429 # In[99]:

```

```

430
431
432 subset = {
433     'Train Normalization Function': ['normalize_std_scaling'],
434     'Test Normalization Function': ['normalize_std_scaling'],
435     'Train splitsfile': ['train-test-splits'],
436     'normalizes latents': [False],
437     'Downstream Epochs': [20, 40, 50],
438     'Pretrain Epochs': [100],
439     'Encoder': ['cpc_encoder_likev8'],
440     'Freeze CPC': [False]
441 }
442 notsubset = {
443     'Downstream Epochs': [100, 120],
444     #'CPC Type': ['cpc_intersect_manylatents']
445 }
446 selection = ['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Encoder', 'Autoregressive', 'Predictor', 'Downstream Model', 'micro', 'macro']
447 cpc_df = filter_df_with_subset(all_df, subset, notsubset)
448 len(cpc_df)
449 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro', 'uses Context', 'uses Latents']], axis=0, dropna=False)
450 result = cpc_df[g['macro']].transform(max) == cpc_df['macro'].reset_index(drop=False).sort_values(by=selection).reset_index(drop=True)
451 result = result[selection]
452 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_new_settings_unfreeze.csv"), index=False)
453 display(result)
454
455
456 # # No Context models (standard)
457
458 # In[126]:
459
460
461 subset = {
462     'Train Normalization Function': ['normalize_std_scaling'],
463     'Test Normalization Function': ['normalize_std_scaling'],
464     'Train splitsfile': ['train-test-splits'],
465     'normalizes latents': [False],
466     'Encoder': ['cpc_encoder_v0'],
467     #'Autoregressive': ['cpc_autoregressive_v0'],
468     'Downstream Epochs': [20, 40],
469     'Pretrain Epochs': [100],
470     'CPC Sampling Mode': ['crossentropy-nocontext'],
471 }
472 notsubset = {
473     'Downstream Epochs': [100, 120],
474     #'CPC Type': ['cpc_intersect_manylatents']
475 }
476 selection = ['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Predictor', 'Downstream Epochs', 'Downstream Model', 'micro', 'macro']
477 cpc_df = filter_df_with_subset(all_df, subset, notsubset)
478 len(cpc_df)
479 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro', 'uses Context', 'uses Latents', 'Downstream Model']], axis=0, dropna=False)
480 result = cpc_df[g['macro']].transform(max) == cpc_df['macro'].reset_index(drop=False).sort_values(by=selection).reset_index(drop=True)
481 result = result[selection]
482 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_somuchanges_nocontext.csv"), index=False)
483 display(result)
484
485
486 # # All settings, reduced to freeze and strided
487
488 # In[127]:
489
490
491 subset = {
492     'Train Normalization Function': ['normalize_std_scaling'],
493     'Test Normalization Function': ['normalize_std_scaling'],
494     'Train splitsfile': ['train-test-splits'],
495     'Downstream Epochs': [20, 40],
496     'Pretrain Epochs': [100],
497 }
498 notsubset = {
499     'Downstream Epochs': [100, 120, 140],
500     #'CPC Type': ['cpc_intersect_manylatents']
501 }
502 selection = ['model', 'Freeze CPC', 'strided', 'Downstream Epochs', 'CPC Sampling Mode', 'Encoder', 'Autoregressive', 'Predictor', 'Downstream Model', 'micro', 'macro']
503 cpc_df = filter_df_with_subset(all_df, subset, notsubset)
504 len(cpc_df)
505 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'CPC Sampling Mode', 'Downstream Epochs', 'Freeze CPC', 'micro', 'macro', 'Encoder', 'Autoregressive', 'Predictor', 'uses Context', 'uses Latents', 'Downstream Model']], axis=0, dropna=False)
506 result = cpc_df[g['macro']].transform(max) == cpc_df['macro'].reset_index(drop=False).sort_values(by=selection).reset_index(drop=True)
507 result = result[selection]
508 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_new_settings_unfreeze.csv"), index=False)
509 display(result)

```

```

510
511
512 # # ALL settings no pretrain 120
513
514 # In[128]:
515
516
517 subset = {
518     'Train Normalization Function': ['normalize_std_scaling'],
519     'Test Normalization Function': ['normalize_std_scaling'],
520     'Train splitsfile': ['train-test-splits'],
521     'normalizes latents': [False, True],
522     'Freeze CPC': [False],
523     'Downstream Epochs': [120],
524     'Uses Classweights': [True],
525 }
526 not_subset = {
527 }
528 selection = ['model', 'strided', 'Encoder', 'Autoregressive', 'Predictor', 'Downstream Model', 'micro', 'macro']
529
530 cpc_df = filter_df_with_subset(all_df, subset, not_subset)
531 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'CPC Type', 'CPC Sampling Mode', '',
532     'micro', 'macro']], axis=0, dropna=False)
533 result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=selection).
534     reset_index(drop=True).sort_index(level='model', key=alphanum_key)
535 result = result[selection]
536 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_new_settings_nopretrain_120.csv"), index=False)
537 display(result)
538
539 # # END for now
540
541
542 # In[16]:
543
544 subset = {
545     'Train Normalization Function': ['normalize_std_scaling'],
546     'Test Normalization Function': ['normalize_std_scaling'],
547     'Train splitsfile': ['train-test-splits'],
548     'normalizes latents': [False],
549     'Downstream Epochs': [120],
550     #'Pretrain Epochs': [100],
551     'CPC Sampling Mode': ['crossentropy-nocontext'],
552     'Freeze CPC': [False]
553 }
554 notsubset = {
555     #'Downstream Epochs': [100, 120],
556     #'CPC Type': ['cpc_intersect_manylatents']
557 }
558 selection = ['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Encoder', 'Autoregressive', 'Predictor', '',
559     'Downstream Model', 'micro', 'macro']
560 cpc_df = filter_df_with_subset(all_df, subset, notsubset)
561 len(cpc_df)
562 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro', 'uses Context', 'uses
563     Latents']], axis=0, dropna=False)
564 result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=selection).
565     reset_index(drop=True)
566 result = result[selection]
567 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_new_settings_unfreeze.csv"), index=False)
568 display(result)
569
570 # # CPC new Settings
571
572 # In[25]:
573
574 subset = {
575     'Train Normalization Function': ['normalize_std_scaling'],
576     'Test Normalization Function': ['normalize_std_scaling'],
577     'Train splitsfile': ['train-test-splits'],
578     'normalizes latents': [False],
579     'Downstream Epochs': [20, 40, 50],
580     'Pretrain Epochs': [100],
581     'CPC Type': ['cpc_intersect_manylatents'],
582 }
583 notsubset = {
584     'Downstream Epochs': [100, 120]
585 }
586 selection = ['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Encoder', 'Autoregressive', 'Predictor', '',
587     'Downstream Model', 'micro', 'macro']
588 cpc_df = filter_df_with_subset(all_df, subset, notsubset)
589 print(len(cpc_df))
590 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'Downstream Model', 'Downstream Epochs',
591     'strided', 'micro', 'macro']], axis=0, dropna=False)
592 result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=selection).
593     reset_index(drop=True)
594 result = result[selection]
595 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_new_settings_context.csv"), index=False)
596

```

```

592 display(result)
593
594
595 # # CPC new Settings nocontext
596
597 # In[33]:
598
599
600 subset = {
601     'Train Normalization Function': ['normalize_std_scaling'],
602     'Test Normalization Function': ['normalize_std_scaling'],
603     'Train splitsfile': ['train-test-splits'],
604     'normalizes latents': [False],
605     'Downstream Epochs': [20, 40, 50],
606     'Pretrain Epochs': [100],
607     'CPC Type': ['cpc_intersect_manylatents'],
608     'CPC Sampling Mode': ['crossentropy-nocontext'],
609     'Downstream Model': ['cpc_downstream_twolinear_v2', 'cpc_downstream_latent_maximum', 'cpc_downstream_latent_average'],
610 }
611
612 notsubset = {
613     'Downstream Epochs': [100, 120],
614     #'CPC Type': ['cpc_intersect_manylatents']
615 }
616 cpc_df = filter_df_with_subset(all_df, subset, notsubset)
617 print(len(cpc_df))
618 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'Downstream Model', 'Downstream Epochs',
619     'strided', 'micro', 'macro']], axis=0, dropna=False)
620 result = cpc_df[g['macro']].transform(max) == cpc_df['macro'].reset_index(drop=False).sort_values(by=selection).
621     reset_index(drop=True)
622 result.to_csv(os.path.join(OUTPUT_PATH, "cpc_new_settings_context.csv"), index=False)
623 display(result)
624
625
626 # ## Low label
627
628 # ## Baseline
629
630 # In[214]:
631
632
633 subset = {
634     'Train Normalization Function': ['normalize_std_scaling'],
635     'Test Normalization Function': ['normalize_std_scaling'],
636     'Uses Classweights': [True],
637     'uses Max Pool': [True, False],
638 }
639 notsubset = {
640     'Train splitsfile': ['train-test-splits'],
641     #'Downstream Epochs': [50],
642 }
643 selection = all_df.columns #['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Downstream Model', 'micro',
644     'macro']
645 ll_df = filter_df_with_subset(all_df, subset, notsubset)
646
647 g = ll_df.groupby(by=['model', 'Train splitsfile'], axis=0, dropna=False)
648 temp = ll_df[g['macro']].transform(max) == ll_df['macro'].groupby(by=[c for c in ll_df.columns if not c in ['Model
649     Path', 'micro', 'macro']], axis=0, dropna=False, as_index=False)
650 bl_group = temp.groupby(by=['model'], as_index=True)
651
652
653 # ## CPC 20
654
655 # In[229]:
656
657
658 subset = {
659     'Train Normalization Function': ['normalize_std_scaling'],
660     'Test Normalization Function': ['normalize_std_scaling'],
661     'Uses Classweights': [True],
662     'strided': [True, False],
663     'Downstream Epochs': [20],
664 }
665 notsubset = {
666     'Train splitsfile': ['train-test-splits'],
667     #'Downstream Epochs': [50],
668 }
669 selection = all_df.columns #['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Downstream Model', 'micro',
670     'macro']
671 ll_df = filter_df_with_subset(all_df, subset, notsubset)
672 # print(len(cpc_df))
673 # g = ll_df.groupby(by=[cn for cn in ll_df.columns if not cn in ['Model Path', 'micro', 'macro', 'uses Context', 'uses
674     'Latents']], axis=0, dropna=False)
675 # result = ll_df[g['macro']].transform(max) == ll_df['macro'].reset_index(drop=False).sort_values(by=selection).
676     #reset_index(drop=True)
677 # #
678 # result.to_csv(os.path.join(OUTPUT_PATH, "cpc_nochanges_unfrozen_std.csv"), index=False)
679 # display(result)
680 g = ll_df.groupby(by=[c for c in ll_df.columns if not c in ['Model Path', 'micro', 'macro']], axis=0, dropna=False)

```

```

674 temp = ll_df[g['macro'].transform(max) == ll_df['macro']]#.groupby(by=[c for c in ll_df.columns if not c in ['Model
675     Path', 'micro', 'macro']], axis=0, dropna=False, as_index=False)
676 cpc_group = temp.groupby(by=[c for c in ll_df.columns if not c in ['Model Path', 'micro', 'macro', "Train splitsfile"
677     ]], axis=0, dropna=False, as_index=True)
678 # In[238]:
679
680 plot_lowlabel_availability([cpc_group, bl_group], title='', save_to="/home/julian/Downloads/Multi-Download/lowlabell",
681     data_col='macro', filename='lowlabell-macro-20.png')
682
683
684 # In[239]:
685
686 plot_lowlabel_availability([cpc_group, bl_group], title='', save_to="/home/julian/Downloads/Multi-Download/lowlabell",
687     data_col='micro', filename='lowlabell-micro-20.png')
688
689
690 # In[240]:
691
692 subset = {
693     'Train Normalization Function': ['normalize_std_scaling'],
694     'Test Normalization Function': ['normalize_std_scaling'],
695     'Uses Classweights': [True],
696     'strided': [True, False],
697     'Downstream Epochs': [50],
698 }
699
700 notsubset = {
701     'Train splitsfile': ['train-test-splits'],
702     #'Downstream Epochs': [50],
703 }
704 selection = all_df.columns #[['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Downstream Model', 'micro',
705     'macro']]
706 ll_df = filter_df_with_subset(all_df, subset, notsubset)
707 # print(len(cpc_df))
708 # g = ll_df.groupby(by=[cn for cn in ll_df.columns if not cn in ['Model Path', 'micro', 'macro', 'uses Context', 'uses
709     Latents']], axis=0, dropna=False)
710 # result = ll_df[g['macro'].transform(max) == ll_df['macro']].reset_index(drop=False).sort_values(by=selection).
711     reset_index(drop=True)
712 # #
713 # result.to_csv(os.path.join(OUTPUT_PATH, "cpc_nochanges_unfrozen_std.csv"), index=False)
714 # display(result)
715 g = ll_df.groupby(by=[c for c in ll_df.columns if not c in ['Model Path', 'micro', 'macro']], axis=0, dropna=False)
716 temp = ll_df[g['macro'].transform(max) == ll_df['macro']]#.groupby(by=[c for c in ll_df.columns if not c in ['Model
717     Path', 'micro', 'macro']], axis=0, dropna=False, as_index=False)
718 cpc_group2 = temp.groupby(by=[c for c in ll_df.columns if not c in ['Model Path', 'micro', 'macro', "Train splitsfile"
719     ]], axis=0, dropna=False, as_index=True)
720 # In[241]:
721
722
723 plot_lowlabel_availability([cpc_group2, bl_group], title='', save_to="/home/julian/Downloads/Multi-Download/lowlabell",
724     data_col='macro', filename='lowlabell-macro-50.png')
725
726
727 # In[242]:
728
729 plot_lowlabel_availability([cpc_group2, bl_group], title='', save_to="/home/julian/Downloads/Multi-Download/lowlabell",
730     data_col='micro', filename='lowlabell-micro-50.png')
731
732
733 # In[237]:
734
735 import os
736 from itertools import cycle
737 from itertools import chain
738
739
740
741
742 import pandas as pd
743 import numpy as np
744 import plotly.graph_objects as go
745 import matplotlib.pyplot as plt
746 import seaborn as sns
747 import torch
748 from matplotlib.font_manager import FontProperties
749 from numpy import interp
750 from sklearn.metrics import auc, ConfusionMatrixDisplay
751 def plot_lowlabel_availability(df_groups_list, title, save_to, filename, data_col='micro', save_legend_seperate=False):
752     fractions = {'train-test-splits-fewer-labels0.001': 0.0012756005952802778,

```

```

753     'train-test-splits-fewer-labels0.005': 0.009378026598634634,
754     'train-test-splits-fewer-labels0.05': 0.10032362459546926,
755     'train-test-splits-fewer-labels0.01': 0.02010252049228734,
756     'train-test-splits-fewer-labels10': 0.18834006566980843,
757     'train-test-splits-fewer-labels14': 0.18829282120331656,
758     'train-test-splits-fewer-labels20': 0.3308057543760187,
759     'train-test-splits-fewer-labels30': 0.44031842770415514,
760     'train-test-splits-fewer-labels40': 0.5257600453546878,
761     'train-test-splits-fewer-labels50': 0.5950912999314956,
762     'train-test-splits-fewer-labels60': 0.6526823045850755,
763     'train-test-splits': 0.7017220608036284,
764     'train-test-splits_min_cut10': 0.015330829376609265,
765     'train-test-splits_min_cut25': 0.037252261828833295,
766     'train-test-splits_min_cut50': 0.06798478728178962,
767     'train-test-splits_min_cut100': 0.116174143103489,
768     'train-test-splits_min_cut150': 0.15470200552760258,
769     'train-test-splits_min_cut200': 0.18864715470200552
770 ordered_splits = [k for k, v in sorted(fractions.items(), key=lambda item: item[1])]
771
772 def get_fraction_x_for_splitsname(name):
773     return fractions[name]
774
775 fontP = FontProperties()
776 fontP.set_size('xx-small')
777 fig, ax = plt.subplots(figsize=(20, 15), dpi=600)
778 #seaborn.set_palette("hls", sum(map(len, df_groups_list)))
779 palette = sns.color_palette(cc.glasbe, n_colors=sum(map(len, df_groups_list)))
780 plt.title(title)
781 num = 0
782 for df_groups in df_groups_list:
783     for name, group in df_groups:
784         cols = df_groups.keys()
785         g = group.reset_index()
786         g.insert(loc=0, column='splitfraction', value=[get_fraction_x_for_splitsname(sp) for sp in g['Train
    splitsfile']])
787         g = g.sort_values(by='splitfraction')
788         if 'cpc' in g['model'].values[0].lower():
789             name = ' | '.join([f'{a[0]}:{a[1]}' if not type(a[1]) == str else a[1] for a in zip(cols, name) if a[0]
    in ['strided', 'Freeze CPC', 'CPC Sampling Mode', 'Downstream Model']])
790
791             ax.plot(g['splitfraction'], g[data_col], '--o', label=name, color=palette[num])
792         else:
793             ax.plot(g['splitfraction'], g[data_col], '-o', label=name, color=palette[num])
794         num += 1
795 plt.ylabel('average AUC score')
796 plt.ylim(0.5, 1.)
797
798 plt.xlabel('fraction of files used (i.r.t. all files)')
799 handles, labels = ax.get_legend_handles_labels()
800 if save_legend_seperate:
801     legend = plt.legend(handles, labels, loc=3, framealpha=1, frameon=False)
802     export_legend(legend, save_to, 'legend-' + filename)
803 else:
804     ax.legend(handles, labels, loc='lower right', prop=fontP, handlelength=3) # bbox_to_anchor=(1.05, 1)
805 if save_to:
806     fig.savefig(os.path.join(save_to, filename), bbox_inches='tight', dpi=fig.dpi)
807 plt.show()
808
809 # ##### unique cpc column values
810
811 # ### 10 best CPC
812
813 # In[44]:
814
815
816
817 N = 20
818 topN_df = all_cpc_df.sort_values(by='macro', ascending=False).head(N)
819 topN_df
820
821
822 # In[54]:
823
824
825 topN_df['CPC Type'].value_counts(),
826
827
828 # In[66]:
829
830
831 d = {c:topN_df[c].value_counts() for c in topN_df.columns if not c in ['model', 'Model Path', 'Downstream Epochs',
    'Pretrain Epochs', 'micro', 'macro']}
832
833
834 # In[78]:
835
836
837 import matplotlib.pyplot as plt
838 fig, axs = plt.subplots(1, len(d), figsize=(40, 5))
839 for i, kv in enumerate(d.items()):

```

```

840     k,v = kv
841     ax = axs[i]
842     ax.set_title(k)
843     v.plot.bar(ax=ax)
844
845
846 # ### Without pretrain
847
848 # In[193]:
849
850
851 #20 epochs frozen
852 subset = {
853     'model': ['CPC'],
854     'Downstream Epochs': [120]
855 }
856 filter_df_with_subset(df, subset).sort_values(by='macro', ascending=False)
857
858
859 # ### All frozen CPC
860
861 # In[157]:
862
863
864 #20 epochs frozen
865 subset = {
866     'Train Normalization Function': ['normalize_std_scaling'],
867     'Test Normalization Function': ['normalize_std_scaling'],
868     'Train splitsfile': ['train-test-splits'],
869     'Freeze CPC': [True],
870     'Uses Classweights': [True],
871     'Predictor': ['cpc_predictor_v0'],
872     'normalizes latents': [False],
873     'Encoder': ['cpc_encoder_v0'],
874     'Autoregressive': ['cpc_autoregressive_v0'],
875     'Downstream Epochs': [20]
876 }
877 filter_df_with_subset(df, subset).sort_values(by='macro', ascending=False)
878
879
880 # In[158]:
881
882
883 #20 epochs unfrozen
884 subset = {
885     'Train Normalization Function': ['normalize_std_scaling'],
886     'Test Normalization Function': ['normalize_std_scaling'],
887     'Train splitsfile': ['train-test-splits'],
888     'Freeze CPC': [False],
889     'Uses Classweights': [True],
890     'Predictor': ['cpc_predictor_v0'],
891     'normalizes latents': [False],
892     'Encoder': ['cpc_encoder_v0'],
893     'Autoregressive': ['cpc_autoregressive_v0'],
894     'Downstream Epochs': [20]
895 }
896 filter_df_with_subset(df, subset)
897
898
899 # In[195]:
900
901
902 #More epochs (40)
903 subset = {
904     'Train Normalization Function': ['normalize_std_scaling'],
905     'Test Normalization Function': ['normalize_std_scaling'],
906     'Train splitsfile': ['train-test-splits'],
907     'Freeze CPC': [True, False],
908     'Uses Classweights': [True],
909     'Predictor': ['cpc_predictor_v0'],
910     'normalizes latents': [False],
911     'Encoder': ['cpc_encoder_v0'],
912     'Autoregressive': ['cpc_autoregressive_v0'],
913     'Downstream Epochs': [40]
914 }
915 filter_df_with_subset(df, subset).sort_values(by='macro', ascending=False)
916
917
918 # In[197]:
919
920
921 #20 epochs unfrozen
922 subset = {
923     'normalizes latents': [True],
924 }
925 filter_df_with_subset(df, subset).sort_values(by='macro', ascending=False)
926
927
928 # In[9]:
929

```

```

930 filter_df_with_subset(df, {}, {}).sort_values(by='macro', ascending=False).drop_duplicates(subset=['model', 'Uses
931 Classweights',
932     'Train Normalization Function', 'Train splitsfile', 'Crop Size',
933     'Test Normalization Function', 'strided', 'Freeze CPC', 'uses Context',
934     'uses Latents', 'normalizes latents', 'CPC Sampling Mode', 'CPC Type',
935     'Autoregressive', 'Encoder', 'Predictor', 'Downstream Model',
936     'Pretrain Epochs', 'Downstream Epochs', 'Latent Size', 'Context Size',
937     'timesteps in', 'timesteps out', 'Convolutional Layer Number',
938     'uses Max Pool', 'uses Adaptive Average Pooling', 'uses Linear',
939     'uses LSTM', 'uses BatchNorm', 'Sum of Strides', 'Sum of Dilation',
940     'Sum of Paddings', 'Sum of Filters', 'Final Layer'])
941
942
943 # In[36]:
944
945
946 #Best CPC networks in general:
947 subset = {
948     'normalizes latents': [True, False],
949     'Uses Classweights': [True, False]
950 }
951 cpc_df = filter_df_with_subset(df, subset)
952 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro']], axis=0, dropna=
953     False)
954 g.max()
955
956 # In[ ]:
957
958
959 #Best BL networks in general:
960 subset = {
961     'normalizes latents': [True, False],
962     'Uses Classweights': [True, False]
963 }
964 cpc_df = filter_df_with_subset(df, subset)
965 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro']], axis=0, dropna=
966     False)
967 g.max()
968
969 # In[10]:
970
971 df.plot.scatter(x='model', y='macro')
972
973
974
975 # In[12]:
976
977
978 all_df[all_df["train timestamp"]>datetime.datetime.fromtimestamp(1643720270)]
979
980
981 # In[5]:
982
983
984 subset = {
985     'Train Normalization Function':['normalize_std_scaling'],
986     'Test Normalization Function':['normalize_std_scaling'],
987     'Uses Classweights': [True],
988     'Train splitsfile':['train-test-splits'],
989     'Freeze CPC':[True],
990     'Uses Classweights': [True],
991     'normalizes latents': [False],
992     'Predictor':['cpc_predictor_v0'],
993     'Encoder': ['cpc_encoder_v0'],
994     'Autoregressive': ['cpc_autoregressive_v0'],
995     'Downstream Model': ['cpc_downstream_latent_maximum', 'cpc_downstream_latent_average', 'cpc_downstream_twolinear_v2'],
996     #'CPC Type':['cpc_intersect'],
997     'Downstream Epochs': [20],
998     "Pretrain Epochs": [100]
999 }
1000 selection = ['model', 'Model Path', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Downstream Model', 'micro', 'macro'
1001 ]
1002 cpc_df = filter_df_with_subset(all_df, subset)
1003 print(len(cpc_df))
1004 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro', 'uses Context', 'uses
1005     Latents']], axis=0, dropna=False)
1006 result = g.max().reset_index(drop=False).sort_index(level='model', key=alphanum_key)
1007 result = result[selection]
1008 display(result)
1009
1010
1011
1012 subset = {
1013     'Train Normalization Function':['normalize_std_scaling'],
1014 }
```

```

1014     'Test Normalization Function':['normalize_std_scaling'],
1015     'Freeze CPC':[True, False]
1016   }
1017   notsubset = {
1018     'Train splitsfile':['train-test-splits'],
1019     #'CPC Type':['cpc_intersect_manylatents']
1020   }
1021 }
1022 selection = ['Model Path', 'model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Encoder', 'Autoregressive', 'Predictor', 'Downstream Epochs', 'Downstream Model', 'micro', 'macro']
1023 cpc_df = filter_df_with_subset(all_df, subset, notsubset)
1024 len(cpc_df)
1025 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro', 'uses Context', 'uses Latents', 'Encoder', 'Autoregressive', 'Predictor', 'Downstream Epochs', 'Downstream Model']], axis=0, dropna=False)
1026 result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=selection).
    reset_index(drop=True)
1027 result = result[selection]
1028 display(result)
1029
1030
1031 # In[80]:
1032
1033
1034 subset = {
1035   'Train Normalization Function':['normalize_std_scaling'],
1036   'Test Normalization Function':['normalize_std_scaling'],
1037   'Uses Classweights': [True],
1038   'Train splitsfile':['train-test-splits'],
1039   'Freeze CPC':[True, False],
1040   'strided' : [True, False],
1041   'normalizes latents': [False],
1042   'Predictor':['cpc_predictor_v0'],
1043   'Encoder': ['cpc_encoder_v0'],
1044   'Autoregressive': ['cpc_autoregressive_v0'],
1045   'Downstream Model': ['cpc_downstream_latent_maximum', 'cpc_downstream_latent_average', 'cpc_downstream_twolinear_v2'],
1046   #'CPC Type':['cpc_intersect'],
1047   'Downstream Epochs': [40],
1048   "Pretrain Epochs": [100]
1049 }
1050 selection = ['model', 'Freeze CPC', 'strided', 'CPC Sampling Mode', 'Downstream Epochs', 'Downstream Model', 'micro',
    'macro']
1051 cpc_df = filter_df_with_subset(all_df, subset)
1052 print(len(cpc_df))
1053 g = cpc_df.groupby(by=[cn for cn in cpc_df.columns if not cn in ['Model Path', 'micro', 'macro', 'uses Context', 'uses Latents']], axis=0, dropna=False)
1054 result = cpc_df[g['macro'].transform(max) == cpc_df['macro']].reset_index(drop=False).sort_values(by=selection).
    reset_index(drop=True)
1055 result = result[selection]
1056 # result.to_csv(os.path.join(OUTPUT_PATH, "cpc_nochanges_frozen_std.csv"), index=False)
1057 display(result)
1058
1059
1060 # In[ ]:
```

B.7.7 jupyter_notebooks -> InfoNCE Loss Psuedo.py (code)

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[30]:
5
6
7 import torch
8 import numpy as np
9 import random
10 from torch import Tensor
11 from torch import nn
12 from typing import List
13 from torch import exp
14 from torch import log
15 from torch.nn import LogSoftmax
16 from torch import diag
17 from torch import arange
18
19
20 # In[2]:
21
22
23 t = torch.empty((30,20))
24 t.shape
25
26
27 # In[7]:
28
```

```

29 def cpc_loss_naive(X:Tensor, W:List[Tensor], encoder:nn.Module, autoregressive:nn.Module):
30     batch, channels, data_length = X.shape #eg. (64 x 12 x 4500)
31
32     latent_matrix = encoder(X)
33     batch, timesteps, latent_size = latent_matrix.shape #eg. (64 x 27 x 128)
34
35     context_matrix = autoregressive(latent_matrix)
36     batch, timesteps, context_size = context_matrix.shape #eg. (64 x 27 x 256)
37
38     k = len(W) #eg. 12
39     current_timestep = timesteps-k-1
40     last_context_vector = context_matrix[:, current_timestep, :] #shape: (64 x 256)
41     loss = 0.
42     for i in range(batch):
43         loss_i = 0
44         for step in range(k):
45             latent_vector_pred_i = last_context_vector[i] @ W[step] #shape: (256) @ (256 x 128) = (128)
46             latent_vector_i = latent_matrix[i, current_timestep+step+1, :] #shape: (128)
47             p_xi_ct = exp(latent_vector_pred_i @ latent_vector_i) #scalar
48             denominator = 0.
49             for j in range(batch):
50                 latent_vector_j = latent_matrix[j, current_timestep+step+1, :] #shape: (128)
51                 p_xj_ct = exp(latent_vector_pred_i @ latent_vector_j) #scalar
52                 denominator += p_xj_ct
53                 loss_i += log(p_xi_ct/denominator)
54             loss += loss_i/-k
55     return loss / batch
56
57
58 # In[105]:
59
60
61 def cpc_loss_logsoftmax(X:Tensor, W:List[nn.Module], encoder:nn.Module, autoregressive:nn.Module):
62     batch, channels, data_length = X.shape #eg. (64 x 12 x 4500)
63
64     latent_matrix = encoder(X)
65     batch, timesteps, latent_size = latent_matrix.shape #eg. (64 x 27 x 128)
66
67     context_matrix = autoregressive(latent_matrix)
68     batch, timesteps, context_size = context_matrix.shape #eg. (64 x 27 x 256)
69
70     k = len(W) #eg. 12
71     current_timestep = timesteps-k-1
72     last_context_vector = context_matrix[:, current_timestep, :] #shape: (64 x 256)
73     loss = 0.
74     accuracy = 0.
75     for t in range(k):
76         #W[t].shape eg. 256 x 128
77         latent_vector = latent_matrix[:, current_timestep+t+1, :] #shape: (64 x 128)
78         latent_vector_pred = W[t](last_context_vector) #shape: (64 x 128)
79
80         scalar_sim = latent_vector_pred @ latent_vector.T #shape (64 x 64)
81         softmax = LogSoftmax(dim=1)(scalar_sim) #shape (64 x 64)
82         p_xi_c_t = diag(softmax) #shape (64)
83         loss += p_xi_c_t.sum() #scalar
84         accuracy += (softmax.argmax(dim=1) == arange(batch)).sum() #scalar
85     loss /= -k*batch
86     accuracy /= k*batch
87     return loss, accuracy
88
89
90
91 # In[ ]:
92
93
94 def cpc_loss_crossentropy_nll(X:Tensor, W:List[nn.Module], encoder:nn.Module, autoregressive:nn.Module):
95     batch, channels, data_length = X.shape #eg. (64 x 12 x 4500)
96
97     latent_matrix = encoder(X)
98     batch, timesteps, latent_size = latent_matrix.shape #eg. (64 x 27 x 128)
99
100    context_matrix = autoregressive(latent_matrix)
101    batch, timesteps, context_size = context_matrix.shape #eg. (64 x 27 x 256)
102
103    timesteps_out = len(W) #eg. 12
104    current_timestep = timesteps-timesteps_out
105    last_context_vector = context_matrix[:, current_timestep-1, :] #shape: (64 x 256)
106    loss = 0.
107    accuracy = 0.
108    for k in range(timesteps_out):
109        pred_latent = W[k](last_context_vector) #shape: (64 x 128)
110        similarity = latent_matrix @ pred_latent.T #shape eg.: (64 x 27 x 64)
111        #ax0 is encoded latent (batch), ax1 is encoded latent (timestep), ax2 is pred latent (batch, timestep k)
112        similarity_r = similarity.reshape(timesteps*batch, batch).T #shape eg.: (64, 1728)
113        labels = torch.arange(batch) + batch*(current_timestep + k) #Select the idx where pred_i==latent_(i,k)
114        loss += crossentropyloss(similarity_r, labels) #pytorch: LogSoftmax + NLL_Loss
115        accuracy += (similarity_r.max(dim=-1).indices == labels).sum()
116
117
118

```

```

119 loss = loss / (timesteps_out*batch)
120 accuracy = accuracy / (timesteps_out*batch) # * pred_latent.shape[0]
121 return accuracy, loss, hidden
122
123
124 # In[97]:
125
126
127 encoder = lambda x: torch.randn((64, 27, 128))
128 auto = lambda x: torch.randn((64, 27, 256))
129 W = [torch.ones((256, 128))] * 12
130
131
132 # In[104]:
133
134
135 X = torch.empty((64, 12, 4500))
136 loss = info_NCE(X, W, encoder, auto)
137 loss
138
139
140 # In[75]:
141
142 latent()
143
144
145
146 # In[28]:
147
148
149 soft = torch.nn.Softmax(dim=0)
150 s = soft(encoder(1)[:, 0] @ encoder(1)[:, 0].T)
151 s[:, 0].sum()
152
153
154 # In[107]:
155
156
157 l = np.arange(12).repeat(4).reshape(12, 4) #shape 12x4
158 lp = np.ones((12, 4))
159
160
161 # In[109]:
162
163
164 l @ lp.T
165
166
167 # In[110]:
168
169
170 lp @ l.T
171
172
173 # In[69]:
174
175
176 s = torch.nn.Softmax(dim=1)(torch.Tensor(lp @ l.T))
177 s
178
179
180 # In[70]:
181
182
183 s[:, 0] #same latent other, prediction
184
185
186 # In[71]:
187
188
189 s[0, :] #same prediction, other latent
190
191
192 # In[ ]:

```

B.7.8 jupyter_notebooks -> MIT data.py (code)

```

1#!/usr/bin/env python
2# coding: utf-8
3
4# In[11]:
5
6
7import wfdb
8import numpy as np
9import os

```

```

10 import pandas as pd
11 from random import randint
12 import math
13 import h5py
14 import matplotlib.pyplot as plt
15 from scipy.signal import find_peaks
16 from collections import Counter
17
18
19 # In[12]:
20
21
22 def print_object_attributes(obj): #https://stackoverflow.com/questions/192109/is-there-a-built-in-function-to-print-
23     #all-the-current-properties-and-values-of-a
24     for attr in dir(obj):
25         print("obj.%s = %r" % (attr, getattr(obj, attr)))
26
27 # ### Read MIT format .dat ecg data files and .hea headers
28
29 # In[13]:
30
31
32 BASE_DIR = '/media/julian/Volume/data/ECG/mit-bih-arrhythmia-database-1.0.0/' #Arrhythmia
33 #BASE_DIR = '/media/julian/Volume/data/ECG/ptb-diagnostic-ecg-database-1.0.0/'
34 READ_ANNOTATIONS = True
35 def get_file_list(BASE_DIR):
36     record_files = []
37     #file_endings = ['.dat', '.hea', '.xyz']
38     with open(os.path.join(BASE_DIR, 'RECORDS')) as recs:
39         record_files = recs.read().splitlines()
40     recs.close()
41     return record_files
42 record_files = get_file_list(BASE_DIR)
43 print(len(record_files))
44 print(record_files)
45
46
47 # ### Extract signal from *.dat files & Read annotations & Read comments
48
49 # In[14]:
50
51
52 def read_comment(record_path):
53     record = wfdb.rdrecord(record_path)
54     return record.comments
55
56
57 # In[15]:
58
59
60 def read_signal(record_path, physical=True):
61     #print(record_path)
62     record = wfdb.rdrecord(record_path, physical=physical)
63     #print_object_attributes(record)
64     if physical:
65         data = record.p_signal
66     else:
67         data = record.d_signal
68     return data
69
70
71 # In[16]:
72
73
74 def read_annotation(record_path, physical=True, return_label_elements=['symbol', 'label_store', 'description']):
75     try:
76         annotation = wfdb.rdnann(record_path, 'atr', return_label_elements=return_label_elements)
77         #print(record_path)
78         #print('sample:', annotation.sample, 'symbol', annotation.symbol, 'contained labels', annotation.label_store, annotation.description)
79         return (annotation.sample, annotation.symbol, annotation.label_store, annotation.description)
80     except ValueError as ve:
81         print(record_path, ' annotation read failed:', ve)
82     return None
83
84 def read_annotation_object(record_path, physical=True, return_label_elements=['symbol', 'label_store', 'description']):
85     try:
86         annotation = wfdb.rdnann(record_path, 'atr', return_label_elements=return_label_elements)
87         return annotation
88     except ValueError as ve:
89         print(record_path, ' annotation read failed:', ve)
90     return None
91
92
93 # In[17]:
94
95
96 def read_record_infos(record_path):
97     record = wfdb.rdrecord(record_path)

```

```

98     return {
99         'record_name' : record.record_name,
100        'file_name' : record.file_name,
101        'record_comments' : record.comments,
102        'number_of_signals' : record.n_sig,
103        'is_physical_signal' : not (record.p_signal is None),
104        'is_digital_signal' : not (record.d_signal is None),
105        'signal_sampling_frequency' : record.fs,
106        'signal_length' : record.sig_len,
107        'signal_channel_names' : record.sig_name,
108        'signal_channel_units' : record.units
109    }
110
111
112 # In[63]:
113
114
115 def plot_record(record_path, index_from, index_to, channels=None):
116     data = read_signal(record_path)
117     annotation = read_annotation(record_path)
118     if annotation:
119         ann_index = 0
120         while annotation[0][ann_index] < index_from:
121             ann_index+=1
122         ann_index_to = ann_index - 1
123         while annotation[0][ann_index_to] < index_to:
124             ann_index_to+=1
125         for i in range(ann_index, ann_index_to):
126             plt.annotate(annotation[1][i], (annotation[0][i], data[annotation[0][i], 0]))
127     if channels is None or channels is []:
128         plt.plot(np.arange(index_from, index_to), data[index_from:index_to, :])
129     else:
130         plt.plot(np.arange(index_from, index_to), data[index_from:index_to, channels])
131     plt.show()
132
133
134 # In[ ]:
135
136
137 def save_to_h5(record_path):
138     if not os.path.exists(storage_path):
139         os.makedirs(storage_path)
140     for f in dat_file_paths:
141         data = read_signal(os.path.join(BASE_DIR, f))
142         print(f)
143         target = os.path.join(storage_path, f.replace('/', '-') +'.h5')
144         with h5py.File(target, 'w') as wf:
145             wf['windows'] = partitioned
146             wf.flush()
147             if verbose: print(target, 'file created and written. %d windows saved.' % (len(partitioned)))
148
149
150 # In[68]:
151
152
153 plot_record(os.path.join(BASE_DIR, record_files[0]), 3000, 20000, channels=None)
154
155
156 # In[18]:
157
158
159 def partition_data(data, window_size=360*10, overlap=1.0, store_in_array=True, align_right=True, normalize_windows=False, mirror_data=False, verbose=True): #maybe allow non float overlap too
160     samples, channels = data.shape
161     if samples < window_size:
162         print('too few samples (%d) to support window size of %d' % (samples, window_size))
163         return None
164     if verbose: print('Input data has shape:', data.shape)
165     shift = window_size*overlap
166     offset = int(samples % shift)
167     if align_right:
168         used_data = data[offset:]
169     else:
170         used_data = data[:-offset]
171     samples, _ = used_data.shape
172     if mirror_data:
173         used_data = np.flip(used_data)
174         #used_data *= -1
175     if verbose: print('The window of size %d will be shifted by %f. The total data used is %d' % (window_size, shift, samples))
176     partitioned = np.empty((int(samples/shift)-1, window_size, channels))
177     if verbose: print('The partitioned data now has shape:', partitioned.shape)
178     for i in range(len(partitioned)):
179         index = int(i*shift)
180         partitioned[i, :, :] = used_data[index:index+window_size, :]
181     if normalize_windows:
182         partitioned = (partitioned-np.amin(partitioned, axis=1)[:, None])/(np.amax(partitioned, axis=1)-np.amin(partitioned, axis=1))[:, None]
183
184     return partitioned, offset

```

```

185
186 # In[19]:
187
188
189 def extract_ecg_beat(record_path, annotation_data=None, channel_index=0, threshold=0.9,
190     window_sampling_duration_in_seconds=10, fixed_window_datapoints=1440, verbose=True): #https://arxiv.org/pdf
191     /1805.00794.pdf methodology
192     infos = read_record_infos(record_path)
193     signal_data = read_signal(record_path)
194     if annotation_data:
195         timestamps, _, label, _ = annotation_data
196     beat_data = []
197     # if fixed_window_datapoints is None:
198     #     fixed_window_datapoints = infos['signal_sampling_frequency']*60/28 #https://de.wikipedia.org/wiki/
199     #     Herzfrequenz_lowest_bpm = 28
200     partitioned_data, offset = partition_data(signal_data, infos['signal_sampling_frequency']*
201     window_sampling_duration_in_seconds, verbose=verbose, mirror_data=False, normalize_windows=True) #1+2
202     total_peaks = total_failed = 0
203     label_index = 0
204     for frame_index in range(len(partitioned_data)):
205         frame = partitioned_data[frame_index]
206         channel = frame[:, channel_index] #channel_index is the 'main' channel
207         peaks, _ = find_peaks(channel, distance=int(infos['signal_sampling_frequency']/10)) #3) #Distance parameter
208         selected_arbitrary_to_remove_peaks_right_next_to_eachother
209         threshold_peaks = peaks[channel[peaks]>threshold] #4)
210         if len(threshold_peaks) > 1:
211             median_RR_interval = np.median((np.roll(threshold_peaks, -1) - threshold_peaks)[:-1]) #5
212             beat_length = int(1.2 * median_RR_interval) #6
213
214         if verbose:
215             plt.plot(channel)
216             plt.plot(threshold_peaks, channel[threshold_peaks], "x")
217             plt.plot(np.zeros_like(channel), "--", color="gray")
218             plt.show()
219             print(median_RR_interval)
220         fails = 0
221         for peak_index:
222             peak_index_in_signal = offset+frame_index*len(frame)+peak_index
223             if beat_length <= fixed_window_datapoints and peak_index_in_signal+beat_length<len(signal_data):
224                 peak_data = signal_data[peak_index_in_signal:peak_index_in_signal+beat_length, :]
225                 peak_data = (peak_data-np.amin(peak_data, axis=0))/(np.amax(peak_data, axis=0)-np.amin(peak_data,
226                 axis=0)) #Normalize again! Not clear in paper!
227                 peak_data = np.pad(peak_data, pad_width=((0, fixed_window_datapoints-beat_length), (0, 0)))
228             if annotation_data:
229                 while timestamps[label_index+1] <= peak_index_in_signal:
230                     label_index += 1
231
232                 if timestamps[label_index] <= peak_index_in_signal and peak_index_in_signal < timestamps[
233                     label_index+1]:
234                     if verbose:
235                         print(timestamps[label_index], '<=', peak_index_in_signal, '<', timestamps[label_index
236                     +1])
237                         print(peak_index_in_signal, ' to ', label[label_index], 'with index:', label_index)
238                         beat_data.append([peak_data, label[label_index]])
239
240                 else:
241                     beat_data.append([peak_data, 0])
242             if verbose:
243                 plt.plot(peak_data)
244                 plt.show()
245             else:
246                 fails += 1
247             if verbose:
248                 print("beat_length > fixed_window_datapoints:", 'median beat length:', median_RR_interval)
249             if verbose:
250                 print(fails, 'of', len(threshold_peaks), 'failed')
251             total_peaks += len(threshold_peaks)
252             total_failed += fails
253             print(total_failed, 'of', total_peaks, 'detected failed')
254             if annotation_data:
255                 print("originally had", len(label), 'labels')
256             return beat_data
257
258 # ### Save all extracted beats in a file
259
260 # In[10]:
261
262
263 import h5py as h5
264 def write_h5_file(file_data, out_filepath):
265     if not os.path.exists(os.path.dirname(out_filepath)):
266         os.makedirs(os.path.dirname(out_filepath))
267     print("Writing file:", out_filepath)
268     all_beats = []
269     all_labels = []
270     with h5.File(out_filepath, 'w') as hdf_file:
271         for f, _, b, _ in file_data:
272             all_beats.append(np.stack([beat[0] for beat in b]))

```

```

267         all_labels.append(np.stack([beat[1] for beat in b]))
268         all_beats = np.concatenate(all_beats)
269         all_labels = np.concatenate(all_labels)
270         hdf_file.create_dataset('data', data=all_beats)
271         hdf_file.create_dataset('label', data=all_labels)
272         hdf_file.flush()
273         hdf_file.close()
274
275
276
277 # ##### Save all signals and attributes in file_data (also note how many had functioning annotations)
278
279 # In[ ]:
280
281
282 patients_in_file = 48 #Choose a number so that it fits into your ram. Higher ~ Bigger files
283 continue_index = 0
284 file_data = None
285
286 while continue_index < len(record_files):
287     del file_data
288     file_data = []
289     success = 0
290     for f in record_files[continue_index:continue_index+patients_in_file]:
291         p = os.path.join(BASE_DIR, f)
292         #print(read_record_infos(p))
293         d = read_signal(p, physical=True)
294         a = None
295         if READ_ANNOTATIONS:
296             a = read_annotation(p, return_label_elements=['label_store'])
297             c_before = Counter(a[2]) #index 2 = label_store
298             print('label vorher:', c_before)
299             #print(a)
300             beats = extract_ecg_beat(p, annotation_data=a, threshold=0.8, verbose=False, channel_index=0,
301             fixed_window_datapoints=2800)
302             if not a is None:
303                 c_after = Counter([beat[1] for beat in beats])
304                 print('label automatisch extracted:', c_after)
305                 success += 1
306                 file_data.append((f, d, beats, a)) #Annotation/Labels available
307             else:
308                 file_data.append((f, d, beats, None)) #only beat-candidates available
309
310     out_filepath = os.path.join(BASE_DIR, 'generated/extracted-beats-patient-'+str(continue_index)+ '-' + str(
311         continue_index+patients_in_file) + '.h5')
312     write_h5_file(file_data, out_filepath)
313     continue_index += patients_in_file
314
315 if READ_ANNOTATIONS:
316     print('%d of %d annotation reads succeeded.' % (success, len(record_files)))
317
318 # # Plotting
319 #
320
321 # In[ ]:
322
323
324 fig, axs = plt.subplots(nrows=6, ncols=6, constrained_layout=True, figsize=(23,23))
325 for ax in axs.flat:
326     random_beat_index = randint(0, len(all_labels))
327     ax.plot(all_beats[random_beat_index, :, 0])
328     ax.set_title(all_labels[random_beat_index], fontsize=14)
329
330
331 # In[ ]:
332
333
334 wfdb.show_ann_labels()
335
336
337 # In[21]:
338
339
340 from hdfviewer.widgets.HDFViewer import HDFViewer
341
342 hdf5 = h5py.File('/media/julian/Volume/data/ECG/mit-bih-arrhythmia-database-1.0.0/generated/extracted-beats-patient-
343 -0-48.h5','r')
344 display(HDFViewer(hdf5))
345
346 # In[35]:
347
348
349
350 for f, d, a in file_data:
351     if a:
352         print(f, d.shape, max(a[0]))
353         plt.plot(d[546885-1000:546885+1000])

```

```

354     break
355
356
357 # In[ ]:
358
359
360 patient0 = file_data[0]
361 data = patient0[1]
362 annotation = patient0[2]
363 timestamp, label_symbol, label_int, label_description = annotation
364
365
366 # In[ ]:
367
368
369 plt.scatter(timestamp, label_symbol)
370 plt.show()
371
372
373 # In[ ]:
374
375
376 plt.gca().invert_yaxis()
377 from_t, to_t = 360, 4000
378 plt.plot(np.arange(from_t, to_t, 1), data[from_t:to_t])
379 plt.show()
380
381
382 # # Resample
383
384 # In[2]:
385
386
387 from wfdb.processing import resample_multichan
388 import glob
389 import sys
390 import traceback
391 def resample_frequency(record_signal_array, annotation_object, hz_frq_in:int, hz_frq_out:int):
392     return resample_multichan(record_signal_array, annotation_object, hz_frq_in, hz_frq_out, resamp_ann_chan=0)
393
394 def resample_frequency_all(record_files, hz_frq_out, output_filepath, physical=True, overwrite=False):
395     print('Resampling of files has started:', record_files)
396     if not os.path.exists(output_filepath):
397         os.makedirs(output_filepath)
398     for file in record_files:
399         print("Resampling:", file)
400         record_path = os.path.join(BASE_DIR, file)
401         if overwrite or not glob.glob(output_filepath+file+'resampled.*'):
402             file_infos = read_record_infos(record_path)
403             print("read record infos")
404             hz_frq_in = file_infos['signal_sampling_frequency']
405             record_signal = read_signal(record_path, physical=physical)
406             print("read record signal", record_signal.shape)
407             annotation_object = read_annotation_object(record_path, physical=physical, return_label_elements=['symbol'])
408             print("read record annotation")
409             try:
410                 resampled_signal, resampled_ann = resample_frequency(record_signal, annotation_object, hz_frq_in,
411                 hz_frq_out) #Throws random assertionerrors at times
412                 print("computed resampled signal", resampled_signal.shape)
413                 wfdb.wrann(file+"resampled", 'atr', sample=resampled_ann.sample, label_store=annotation_object.
414                 label_store, write_dir=output_filepath)
415                 a_temp = wfdb.Annotation(file+"resampled", 'atr', resampled_ann.sample, symbol=annotation_object.
416                 symbol, fs=hz_frq_out, label_store=annotation_object.label_store)
417                 a_temp.wrann(write_fs=True, write_dir=output_filepath)
418                 if physical:
419                     wfdb.wrsamp(file+"resampled", fs=hz_frq_out, p_signal=resampled_signal, sig_name=file_infos['
420                     signal_channel_names'], units=file_infos['signal_channel_units'], comments=file_infos['record_comments'],
421                     write_dir=output_filepath)
422                 else:
423                     wfdb.wrsamp(file+"resampled", fs=hz_frq_out, d_signal=resampled_signal, sig_name=file_infos['
424                     signal_channel_names'], units=file_infos['signal_channel_units'], comments=file_infos['record_comments'],
425                     write_dir=output_filepath)
426                     print('finished')
427                     except AssertionError: #https://stackoverflow.com/questions/11587223/how-to-handle-assertionerror-in-
428                     python-and-find-out-which-line-or-statement-it-
429                         ___, tb = sys.exc_info()
430                         traceback.print_tb(tb) # Fixed format
431                         tb_info = traceback.extract_tb(tb)
432                         filename, line, func, text = tb_info[-1]
433
434                         print('An error occurred on line {} in statement {}'.format(line, text))
435
436                         else:
437                             print("skipping", file)
438
439 # In[ ]:
440
441
442

```

```

435 record_files = get_file_list(BASE_DIR)
436 resample_frequency_all(record_files, hz_frq_out=1000, output_filepath=os.path.join(BASE_DIR,'generated/resampled'))
437
438
439 # ## Read resampled data
440
441 # In[21]:
442
443
444 BASE_DIR = '/media/julian/Volume/data/ECG/mit-bih-arrhythmia-database-1.0.0/generated/resampled/' #Arrhythmia
445 #BASE_DIR = '/media/julian/Volume/data/ECG/pvt-diagnostic-ecg-database-1.0.0/'
446 READ_ANNOTATIONS = True
447 def get_file_list(BASE_DIR):
448     record_files = []
449     file_endings = ['.dat', '.hea', '.xyz']
450     with open(os.path.join(BASE_DIR, 'RECORDS')) as recs:
451         record_files = recs.read().splitlines()
452         recs.close()
453     return record_files
454 record_files = get_file_list(BASE_DIR)
455 print(len(record_files))
456 print(record_files)
457
458
459 # In[26]:
460
461 read_record_infos(os.path.join(BASE_DIR,record_files[0]))
462
463
464 # In[ ]:
465
466
467 plot_record(os.path.join(BASE_DIR,record_files[0]), 3000, 15000)

```

B.7.9 jupyter_notebooks -> Model Gradient-Prediction Scatter.py (code)

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[2]:
5
6
7 import pickle
8 import numpy as np
9 import torch
10 import matplotlib.pyplot as plt
11 import glob
12 import seaborn as sns
13 from torch import tensor
14
15
16 # In[2]:
17
18
19 import seaborn as sns
20 cmap_green = sns.light_palette("seagreen", as_cmap=True)
21 cmap_green
22
23
24 # In[3]:
25
26
27 cmap_red = sns.light_palette('salmon', as_cmap=True)
28 cmap_red
29
30
31 # In[4]:
32
33
34 cmap_both = sns.diverging_palette(10, 150, s=75, l=50, center='light', as_cmap=True)
35 cmap_both
36
37
38 # ## Normal
39
40 # In[33]:
41
42
43 def timeseries_to_image_with_gradient(data: torch.Tensor, labels:torch.Tensor, gradient: torch.Tensor, pred:torch.Tensor=None, model_thresholds=None, title=None, filenames :str=None, show=False, save=True):
44     batches, width, height = data.shape
45     for batch in range(batches):
46         cmap_green = sns.light_palette("seagreen", as_cmap=True)
47         cmap_both = sns.diverging_palette(10, 150, s=75, l=50, center='light', as_cmap=True)
48         fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))

```

```

49     plt.xlim((0, width))
50     plt.ylim((0, 1))
51     gradient = gradient[batch].abs()
52     grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
53     title = title or ""
54     if not labels is None:
55         title += "Correct classes: " + ", ".join(map(str, np.nonzero(labels[batch].numpy())[0])) + ". "
56     if not (pred is None or model_thresholds is None):
57         binary_pred = pred[batch] >= model_thresholds
58         title += "Predicted classes: " + ", ".join(map(str, np.nonzero(binary_pred.numpy())[0])) + ". "
59     fig.suptitle(title, y=0.9)
60
61     for i, ax in enumerate(axes):
62         ax.set_xlim((0, width))
63         ax.set_ylim((-0.1, 1.1))
64         ax.axis('off')
65         d = data[batch, :, i]
66         g = grad_norm[:, i:i+1].T
67         ax.plot(range(width), d, color='red')
68         ax.autoscale(False)
69         ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_green)
70     if save:
71         plt.savefig(filenames[batch], dpi=fig.dpi)
72     if show:
73         plt.show()
74     plt.close()
75
76
77 # In[7]:
78
79
80 thresh_dict = {0: 0.0052883076, 1: 0.004021993, 2: 0.045075733, 3: 0.039263155, 4: 0.087553956, 5: 0.03217059, 6:
80     0.07084644, 7: 0.06982235, 8: 0.08900607, 9: 0.009527704, 10: 0.04808709, 11: 0.023643928, 12: 0.04898496, 13:
80     0.014053739, 14: 0.11855617, 15: 0.0057171667, 16: 0.06450285, 17: 0.014764133, 18: 0.0027846538, 19:
80     0.0014224607, 20: 0.0042695478, 21: 0.0020088167, 22: 0.0056370394, 23: 0.023299428, 24: 0.0043616697, 25:
80     0.0043225554, 26: 0.0006829281, 27: 0.003797319, 28: 0.0023218843, 29: 0.0025232348, 30: 0.08151142, 31:
80     0.0057842294, 32: 0.055861708, 33: 0.09779097, 34: 0.0037202945, 35: 0.013829965, 36: 0.025294341, 37:
80     0.072333194, 38: 0.006157023, 39: 0.0070163156, 40: 0.0024529276, 41: 0.396414, 42: 0.054770038, 43:
80     0.0033453542, 44: 0.027208127, 45: 0.0036930004, 46: 0.095755436, 47: 0.05874352, 48: 0.037359316, 49:
80     0.0045002145, 50: 0.0026231722, 51: 0.0012562033, 52: 0.0064664735, 53: 0.0096522, 54: 0.011148318, 55:
80     0.07452798, 56: 0.028948307, 57: 0.005614491, 58: 0.006150292, 59: 0.0003258857, 60: 0.03152733, 61:
80     0.015745273, 62: 0.0013439627, 63: 0.02767622, 64: 0.012850131, 65: 0.0044486015, 66: 0.003912842}
81 thresholds = torch.Tensor(list(thresh_dict.values()))
82
83
84 # In[3]:
85
86
87 with open('/home/julian/Downloads/Github/contrastive-predictive-coding/temp-grad.pickle', 'rb') as f:
88     saved = pickle.load(f)
89 data, labels, gradient, pred = saved
90 filename = 'gradient-normal(correct:41).png'
91 timeseries_to_image_with_gradient(data, labels, gradient, pred, model_thresholds=thresholds, title='Gradient for
91     normal label\n', filenames=[filename], save=False, show=True)
92
93
94 # ## Inverse
95
96 # In[29]:
97
98
99 with open('/home/julian/Downloads/Github/contrastive-predictive-coding/temp-grad-inverse41.pickle', 'rb') as f:
100     saved = pickle.load(f)
101 data, labels, gradient, pred = saved
102 _, width, height = data.shape
103 cmap_green = sns.light_palette("seagreen", as_cmap=True)
104 fig, axes = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
105 plt.xlim((0, width))
106 plt.ylim((0, 1))
107 gradient = gradient.abs()
108 grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
109 for i, ax in enumerate(axes):
110     ax.set_xlim((0, width))
111     ax.set_ylim((-0.1, 1.1))
112     ax.axis('off')
113     d = data[0, :, i]
114     g = grad_norm[0, :, i:i+1].T
115     ax.plot(range(width), d, color='red')
116     ax.autoscale(False)
117     ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_green)
118
119
120 # In[14]:
121
122
123 with open('/home/julian/Downloads/Github/contrastive-predictive-coding/temp-grad14.pickle', 'rb') as f:
124     saved = pickle.load(f)
125 data, labels, gradient, pred = saved
126 _, width, height = data.shape
127 cmap_green = sns.light_palette("seagreen", as_cmap=True)

```

```

128 fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
129 plt.xlim((0, width))
130 plt.ylim((0, 1))
131 gradient = gradient.abs()
132 grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
133 for i, ax in enumerate(axs):
134     ax.set_xlim((0, width))
135     ax.set_ylim((-0.1, 1.1))
136     ax.axis('off')
137     d = data[0, :, i]
138     g = grad_norm[0, :, i:i+1].T
139     ax.plot(range(width), d, color='red')
140     ax.autoscale(False)
141     ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_green)
142
143
144 # In[15]:
145
146
147 with open('/home/julian/Downloads/Github/contrastive-predictive-coding/temp-grad-inverse14.pickle', 'rb') as f:
148     saved = pickle.load(f)
149 data, labels, gradient, pred = saved
150 _, width, height = data.shape
151 cmap_green = sns.light_palette("seagreen", as_cmap=True)
152 fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
153 plt.xlim((0, width))
154 plt.ylim((0, 1))
155 gradient = gradient.abs()
156 grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
157 for i, ax in enumerate(axs):
158     ax.set_xlim((0, width))
159     ax.set_ylim((-0.1, 1.1))
160     ax.axis('off')
161     d = data[0, :, i]
162     g = grad_norm[0, :, i:i+1].T
163     ax.plot(range(width), d, color='red')
164     ax.autoscale(False)
165     ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_green)
166
167
168 # In[92]:
169
170
171 with open('/home/julian/Downloads/Github/contrastive-predictive-coding/temp-grad41.pickle', 'rb') as f:
172     saved = pickle.load(f)
173 data, labels, gradient, pred = saved
174 _, width, height = data.shape
175 cmap_green = sns.light_palette("seagreen", as_cmap=True)
176 fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
177 plt.xlim((0, width))
178 plt.ylim((0, 1))
179 gradient = np.clip(gradient, a_min=0, a_max=None)
180 grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
181 #fig.tight_layout()
182 for i, ax in enumerate(axs):
183     ax.set_xlim((0, width))
184     ax.set_ylim((-0.1, 1.1))
185     ax.axis('off')
186     d = data[0, :, i]
187     g = grad_norm[0, :, i:i+1].T
188     ax.plot(range(width), d, color='red')
189     ax.autoscale(False)
190     ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_green)
191
192
193 # In[120]:
194
195
196 for figi in range(67):
197     with open('/home/julian/Downloads/Github/contrastive-predictive-coding/temp-grad{figi}.pickle', 'rb') as f:
198         saved = pickle.load(f)
199 data, labels, gradient, pred = saved
200 _, width, height = data.shape
201 cmap_green = sns.light_palette("seagreen", as_cmap=True)
202 fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
203 plt.xlim((0, width))
204 plt.ylim((0, 1))
205 gradient = gradient.abs()
206 grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
207 for i, ax in enumerate(axs):
208     ax.set_xlim((0, width))
209     ax.set_ylim((-0.1, 1.1))
210     ax.axis('off')
211     d = data[0, :, i]
212     g = grad_norm[0, :, i:i+1].T
213     ax.plot(range(width), d, color='red')
214     ax.autoscale(False)
215     ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_green)
216 plt.savefig(f'gradient_visual-{figi}.png', dpi=fig.dpi)
217 plt.close()

```

```

218
219
220 # In[119]:
221
222
223 for figi in range(67):
224     with open(f'/home/julian/Downloads/Github/contrastive-predictive-coding/temp-grad{figi}.pickle', 'rb') as f:
225         saved = pickle.load(f)
226     data, labels, gradient, pred = saved
227     _, width, height = data.shape
228     cmap_green = sns.light_palette("seagreen", as_cmap=True)
229     fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
230     plt.xlim((0, width))
231     plt.ylim((0, 1))
232     gradient = gradient
233     grad_norm = (gradient.gradient.min())/(gradient.max()-gradient.min())
234     for i, ax in enumerate(axs):
235         ax.set_xlim((0, width))
236         ax.set_ylim((-0.1, 1.1))
237         ax.axis('off')
238         d = data[0, :, i]
239         g = grad_norm[0, :, i:i+1].T
240         ax.plot(range(width), d, color='red')
241         ax.autoscale(False)
242         ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_both)
243     plt.savefig(f'gradient_visual-noabs-{figi}.png', dpi=fig.dpi)
244     plt.close()
245
246
247 # ## Denoising data with gradient
248
249 # In[27]:
250
251
252 with open(f'/home/julian/Downloads/Github/contrastive-predictive-coding/temp-grad.pickle', 'rb') as f:
253     saved = pickle.load(f)
254 data, labels, gradient, pred = saved
255 _, width, height = data.shape
256 cmap_green = sns.light_palette("seagreen", as_cmap=True)
257 fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
258 gradient = gradient
259 grad_norm = (gradient.gradient.min())/(gradient.max()-gradient.min())
260 for i, ax in enumerate(axs):
261     ax.axis('off')
262     d = data[0, :, i]
263     g = grad_norm[0, :, i]
264     da = d+g
265     ax.set_xlim((0, width))
266     ax.set_ylim((da.min()-0.1, da.max()+0.1))
267     ax.plot(range(width), da, color='red')
268
269     ax.imshow(grad_norm[0, :, i:i+1].T, extent=[0, width, da.min()-0.1, da.max()+0.1], aspect='auto', cmap=cmap_both)
270 plt.show()
271
272
273 # In[57]:
274
275
276 with open(f'/home/julian/Downloads/Github/contrastive-predictive-coding/temp-grad.pickle', 'rb') as f:
277     saved = pickle.load(f)
278 data, labels, gradient, pred = saved
279 _, width, height = data.shape
280 cmap_green = sns.light_palette("seagreen", as_cmap=True)
281 fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
282 grad_abs = gradient.abs()
283 grad_norm = (grad_abs-grad_abs.min())/(grad_abs.max()-grad_abs.min())*torch.sign(gradient)+0.5
284 fig.suptitle("Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].numpy())))
285 for i, ax in enumerate(axs):
286     ax.axis('off')
287     ax.set_xlim((0, width))
288     ax.set_ylim((-0.1, 1.1))
289     d = data[0, :, i]
290     g = grad_norm[0, :, i]
291     ax.plot(range(width), d, color='red')
292
293     ax.imshow(grad_norm[0, :, i:i+1].T, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_both)
294 plt.show()
295
296
297 # In[109]:
298
299
300 str([s for s in np.nonzero(labels)[:, 1].numpy()])
301
302
303 # In[117]:
304
305
306 "Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].numpy()))
307

```

```

308
309 # In[35]:
310
311
312
313
314
315 # In[11]:
316
317
318 with open('/home/julian/Downloads/Github/contrastive-predictive-coding/temp-grad41.pickle', 'rb') as f:
319     saved = pickle.load(f)
320 data, labels, gradient, pred = saved
321 _, width, height = data.shape
322 cmap_green = sns.light_palette("seagreen", as_cmap=True)
323 fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
324 plt.xlim((0, width))
325 plt.ylim((0, 1))
326 gradient = gradient.abs()
327 grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
328 fig.suptitle("Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].numpy()))), y=0.9
329 for i, ax in enumerate(axs):
330     ax.set_xlim((0, width))
331     ax.set_ylim((-0.1, 1.1))
332     ax.axis('off')
333     d = data[0, :, i]
334     g = grad_norm[0, :, i:i+1].T
335     ax.plot(range(width), d, color='red')
336     ax.autoscale(False)
337     ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_green)
338 plt.savefig('gradient-vis.png', dpi=fig.dpi)
339
340
341 # In[62]:
342
343
344 with open('/home/julian/Downloads/Github/contrastive-predictive-coding/temp-grad41.pickle', 'rb') as f:
345     saved = pickle.load(f)
346 data, labels, gradient, pred = saved
347 _, width, height = data.shape
348 cmap_green = sns.light_palette("seagreen", as_cmap=True)
349 fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
350 plt.xlim((0, width))
351 plt.ylim((0, 1))
352 sign = torch.sign(gradient)
353 #gradient = gradient.abs()
354 grad_norm = ((gradient-gradient.min())/(gradient.max()-gradient.min()))
355 fig.suptitle("Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].numpy()))), y=0.9
356 for i, ax in enumerate(axs):
357     ax.set_xlim((0, width))
358     ax.set_ylim((-0.1, 1.1))
359     ax.axis('off')
360     d = data[0, :, i]
361     g = grad_norm[0, :, i:i+1].T
362     ax.plot(range(width), d, color='red')
363     ax.autoscale(False)
364     ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_both)
365 plt.savefig('gradient-vis.png', dpi=fig.dpi)
366
367
368 # # Prediction Visualization
369
370 # In[4]:
371
372
373 with open('/home/julian/Downloads/Github/contrastive-predictive-coding/temp-grad.pickle', 'rb') as f:
374     saved = pickle.load(f)
375 data, labels, gradient, pred = saved
376
377
378 # In[56]:
379
380
381 threshs = {0: 0.0052883076, 1: 0.004021993, 2: 0.045075733, 3: 0.039263155, 4: 0.087553956, 5: 0.03217059, 6: 0.07084644, 7: 0.06982235, 8: 0.008900607, 9: 0.009527704, 10: 0.04808709, 11: 0.023643928, 12: 0.04898496, 13: 0.014053739, 14: 0.11855617, 15: 0.0057171667, 16: 0.06450285, 17: 0.014764133, 18: 0.0027846538, 19: 0.0014224607, 20: 0.0042695478, 21: 0.0020088167, 22: 0.0056370394, 23: 0.023299428, 24: 0.00043616697, 25: 0.0043225554, 26: 0.0006829281, 27: 0.003797319, 28: 0.0023218843, 29: 0.0025232348, 30: 0.08151142, 31: 0.0057842294, 32: 0.055861708, 33: 0.09779097, 34: 0.0037202945, 35: 0.013829965, 36: 0.025294341, 37: 0.072333194, 38: 0.006157023, 39: 0.0070163156, 40: 0.0024529276, 41: 0.396414, 42: 0.054770038, 43: 0.0033453542, 44: 0.027208127, 45: 0.0036930004, 46: 0.095755436, 47: 0.05874352, 48: 0.037359316, 49: 0.0045002145, 50: 0.0026231722, 51: 0.0012562033, 52: 0.0064664735, 53: 0.0096522, 54: 0.011148318, 55: 0.07452798, 56: 0.028948307, 57: 0.005614491, 58: 0.006150292, 59: 0.0003258857, 60: 0.03152733, 61: 0.015745273, 62: 0.0013439627, 63: 0.02767622, 64: 0.012850131, 65: 0.0044486015, 66: 0.003912842}
382 binary_pred = pred[0]>torch.Tensor(list(threshs.values()))
383
384
385 # In[57]:
386
387

```

```

388 threshold_tensor = torch.Tensor(list(threshs.values()))
389 diff = (pred[0]-threshold_tensor).numpy()
390 diff
391 x = np.arange(len(threshold_tensor))
392
393
394 # In[58]:
395
396
397 plt.title("Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].numpy())))
398 plt.scatter(x[binary_pred], pred[0][binary_pred], c='yellow', label='true')
399 plt.scatter(x[~binary_pred], pred[0][~binary_pred], c='purple', label='false')
400
401 plt.xlabel('Class Nr.')
402 plt.ylabel('Model output')
403 for class_i in x:
404     plt.annotate(str(class_i), (class_i, pred[0][class_i]))
405 plt.legend()
406 plt.savefig('scatterplot-prediction.png')
407 plt.show()
408
409
410 # In[59]:
411
412
413 plt.title("Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].numpy())))
414 plt.scatter(x[binary_pred], diff[binary_pred], c='yellow', label='true')
415 plt.scatter(x[~binary_pred], diff[~binary_pred], c='purple', label='false')
416
417 plt.xlabel('Class Nr.')
418 plt.ylabel('model output minus specific class thresholds')
419 for class_i in x:
420     plt.annotate(str(class_i), (class_i, diff[class_i]))
421 plt.legend(title='class i')
422 plt.savefig('scatterplot-difference.png')
423 plt.show()
424
425
426 # In[64]:
427
428
429 normed_pos = diff/(1-threshold_tensor)
430 normed_neg = diff/threshold_tensor
431 probs = np.where(diff>=0, normed_pos, normed_neg)
432
433
434 # In[65]:
435
436
437 plt.title("Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].numpy())))
438 plt.scatter(x[binary_pred], probs[binary_pred], c='yellow', label='true')
439 plt.scatter(x[~binary_pred], probs[~binary_pred], c='purple', label='false')
440
441 plt.xlabel('Class Nr.')
442 plt.ylabel(r'"Probabilities" obtained by weighting output with threshold$^{(-1)}$')
443 for class_i in x:
444     plt.annotate(str(class_i), (class_i, probs[class_i]))
445 plt.savefig('scatterplot-probabilities.png')
446 plt.show()
447
448
449 # In[220]:
450
451
452 softp = torch.nn.Softmax(dim=0)(torch.Tensor(diff[binary_pred]))
453 softn = torch.nn.Softmax(dim=0)(torch.Tensor(diff[~binary_pred]))
454
455
456 # In[221]:
457
458
459 plt.scatter(x[binary_pred], softp, c='yellow')
460 plt.scatter(x[~binary_pred], -softn, c='purple')
461
462
463 # In[8]:
464
465
466 def plot_prediction_scatterplots(labels, pred, model_thresholds, filename=None):
467     binary_pred = pred[0] >= model_thresholds
468     diff = (pred[0] - model_thresholds).numpy()
469     x = np.arange(len(model_thresholds))
470     fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
471     title = "Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].numpy()))
472     title += "\n Predicted classes (binary): " + ", ".join(map(str, np.nonzero(binary_pred.numpy())[0]))
473     fig.suptitle(title)
474     ##First plot
475     ax1.set_aspect='equal'
476     ax1.scatter(x[binary_pred], pred[0][binary_pred], c='yellow', label='true')
477     ax1.scatter(x[~binary_pred], pred[0][~binary_pred], c='purple', label='false')

```

```

478 ax1.set_xlabel('Class Nr.')
479
480 ax1.set_ylabel('Model output')
481 for class_i in x:
482     ax1.annotate(str(class_i), (class_i, pred[0][class_i]))
483 ax1.legend()
484 ##Second Plot
485 ax2.set_aspect='equal'
486 ax2.scatter(x[binary_pred], diff[binary_pred], c='yellow', label='true')
487 ax2.scatter(x[~binary_pred], diff[~binary_pred], c='purple', label='false')
488 ax2.set_xlabel('Class Nr.')
489 ax2.set_ylabel('model output minus specific class thresholds')
490 for class_i in x:
491     ax2.annotate(str(class_i), (class_i, diff[class_i]))
492 ax2.legend()
493 ##Third Plot
494 normed_pos = diff/(1-model_thresholds)
495 normed_neg = -pred/model_thresholds
496 probs = np.where(diff>=0, normed_pos, normed_neg)
497 ax3.scatter(x[binary_pred], probs[binary_pred], c='yellow', label='true')
498 ax3.scatter(x[~binary_pred], probs[~binary_pred], c='purple', label='false')
499
500 ax3.set_aspect='equal'
501 ax3.set_xlabel('Class Nr.')
502 ax3.set_ylabel(r'"Probabilities" obtained by weighting output with threshold$^{(-1)}$')
503 for class_i in x:
504     ax3.annotate(str(class_i), (class_i, probs[class_i]))
505 ax3.legend()
506 filename = filename or ""
507 fig.tight_layout()
508 plt.savefig(filename + 'scatterplot-predictions(combined).png', dpi=fig.dpi)
509 plt.show()
510
511
512 # In[9]:
513
514
515 plot_prediction_scatterplots(labels, pred, thresholds, filename=None)
516
517
518 # # Average prediction probs
519
520 # In[5]:
521
522
523 import glob
524 import os
525 import pandas as pd
526 def read_label_csv_from_model_folder(model_folder, data_loader_index=0):
527     label_path = glob.glob(os.path.join(model_folder, f"labels-dataloader-{(data_loader_index)}.csv"))
528     if len(label_path) == 0:
529         raise FileNotFoundError
530     df1 = pd.read_csv(label_path[0])
531     labels = df1.values[:, 1:].astype(float)
532     return labels, df1.columns[1:].values
533
534 def read_binary_label_csv_from_model_folder(model_folder, data_loader_index=0):
535     l, c = read_label_csv_from_model_folder(model_folder, data_loader_index)
536     l = (l > 0).astype(int)
537     return l, c
538
539 def read_output_csv_from_model_folder(model_folder, data_loader_index=0):
540     pred_path = glob.glob(os.path.join(model_folder, f"model-*-dataloader-{(data_loader_index)}-output.csv"))
541     if len(pred_path) == 0:
542         raise FileNotFoundError
543     dfp = pd.read_csv(pred_path[0])
544     return dfp.values[:, 1:].astype(float), dfp.columns[1:].values #1 is file
545
546
547 # In[6]:
548
549
550 def plot_errorbar_minmax(meanx, minx, maxx):
551     plt.figure(figsize=(20, 15), dpi=200)
552     x = np.arange(0, len(meanx))
553     (dots, caps, _) = plt.errorbar(x, meanx, yerr=[meanx-minx, maxx-meanx], fmt='k.', capsize=5)
554     for cap in caps:
555         cap.set_color('red')
556         cap.set_markeredgewidth(2)
557     plt.show()
558
559 def plot_errorbar_std(meanxi, stdxi, idx_order, group_names=None, title='', save_path=None):
560     if not (type(meanxi) == list and type(stdxi) == list):
561         meanxi, stdxi = [meanxi], [stdxi]
562     fig, ax = plt.subplots(dpi=200, figsize=[6.4*2, 4.8*2])
563     ax.set_xlabel('Class')
564     ax.set_ylabel('Probability')
565     if not save_path:
566         ax.set_title(title)
567     first = True

```

```

568     for i in range(len(meanxi)):
569         line_label = "Standard Deviation"
570         if group_names:
571             line_label += f" ({group_names[i]})"
572         meanx, stdx = meanxi[i], stdxi[i]
573         if idx_order is None:
574             idx_order = np.arange(len(meanx))
575         if type(meanx) == list:
576             meanx = [m[idx_order] for m in meanx]
577         else:
578             meanx = meanx[idx_order]
579         if type(stdx) == list:
580             stdx = [s[idx_order] for s in stdx]
581         else:
582             stdx = stdx[idx_order]
583         x = np.arange(len(meanx))+0.2*i
584         (dots, caps, error) = ax.errorbar(x, meanx, yerr=stdx, fmt='.', capsizes=5, label=line_label)
585         dots.set_label(f"Mean Prediction Probability ({group_names[i]} if type(group_names)==list else group_names))")
586         for cap in caps:
587             cap.set_markeredgewidth(2)
588         #dots.set_markersize(10)
589         ax.set_ylim(top=1.)
590         if not all(idx_order==np.arange(len(idx_order))):
591             ax.set_xticks(np.arange(len(idx_order)))
592             ax.set_xticklabels(idx_order, rotation='vertical')
593         ax.legend(loc='upper right')
594         ax.grid(which='major', color="#CCCCCC", linestyle='--')
595         ax.grid(which='minor', color="#CCCCCC", linestyle=':')
596         if save_path:
597             fig.savefig(save_path, dpi=fig.dpi)
598     def calculate_plot_with_thresholds(labels, preds, thresholds, model_name, title='Prediction Probabilities for each
599                                         class with Mean and Standard Deviation', save_path=None, sort_fn=None, std_two_directional=False):
600         diff = (preds - thresholds)
601         transformed_probs = (np.where(diff<0, diff/thresholds, diff/(1-thresholds))+1.)/2.
602         calculate_plot(labels, transformed_probs, model_name, title, save_path, sort_fn, std_two_directional)
603     def calculate_plot(labels, preds, model_name, title='Prediction Probabilities for each class with Mean and Standard
604                                         Deviation', save_path=None, sort_fn=None, std_two_directional=False):
605         title = model_name + "\n" + title
606         sel0 = np.where(labels==0, preds, np.nan)
607         mi0 = np.nanmin(sel0, axis=0)
608         ma0 = np.nanmax(sel0, axis=0)
609         mean0 = np.nanmean(sel0, axis=0)
610         std0 = np.nanstd(sel0, axis=0)
611         sell = np.where(labels!=0, preds, np.nan)
612         mi1 = np.nanmin(sell, axis=0)
613         ma1 = np.nanmax(sell, axis=0)
614         mean1 = np.nanmean(sell, axis=0)
615         std1 = np.nanstd(sell, axis=0)
616         if std_two_directional:
617             diff0 = sel0-mean0
618             std0 = np.sqrt(np.nanmean(np.where(diff0>=0, diff0, np.nan)**2, axis=0))#np.nanstd(np.where(diff0>=0, sel, np.
619             nan), axis=0)
620             stdn0 = np.sqrt(np.nanmean(np.where(diff0<=0, diff0, np.nan)**2, axis=0))#np.nanstd(np.where(diff0<=0, sel, np.
621             .nan), axis=0)
622             diff1 = sell-mean1
623             stdp1 = np.sqrt(np.nanmean(np.where(diff1>=0, diff1, np.nan)**2, axis=0))#np.nanstd(np.where(diff1>=0, sel, np.
624             .nan), axis=0)
625             std1 = np.sqrt(np.nanmean(np.where(diff1<=0, diff1, np.nan)**2, axis=0))#np.nanstd(np.where(diff1<=0, sel, np.
626             .nan), axis=0)
627         idx_order = np.arange(labels.shape[1])
628         if not sort_fn is None:
629             idx_order = sort_fn(sel0, mi0, ma0, mean0, std0, sell, mi1, ma1, mean1, std1)
630             print(idx_order)
631         if std_two_directional:
632             plot_errorbar_std([mean0, mean1], [[stdn0, std0], [std1, stdp1]], group_names=['Ground Truth = False', 'Ground
633             Truth = True'], title=title, save_path=save_path, idx_order=idx_order)
634         else:
635             plot_errorbar_std([mean0, mean1], [std0, std1], group_names=['Ground Truth = False', 'Ground Truth = True'],
636             title=title, save_path=save_path, idx_order=idx_order)
637     def sort_by_std_gap(sel0, mi0, ma0, mean0, std0, sell, mi1, ma1, mean1, std1):
638         return np.argsort((mean0+std0) - (mean1-std1))
639     def sort_by_mean_gap(sel0, mi0, ma0, mean0, std0, sell, mi1, ma1, mean1, std1):
640         return np.argsort(mean0 - mean1)
641     def sort_by_group_count(sel0, mi0, ma0, mean0, std0, sell, mi1, ma1, mean1, std1):
642         return np.argsort(np.count_nonzero(~np.isnan(sel0)), axis=0)+np.count_nonzero(~np.isnan(sell), axis=0)
643     # ## (no model threshold reweighting)
644     # In[9]:
645     model_test_folder = '/home/julian/Downloads/Github/contrastive-predictive-coding/models/01_12_21-12-48-test|(12x)cpc
646     /01_12_21-10-train|(12x)cpc/architectures_cpc.cpc_combined.CPCCombined9|use_weights|strided|frozen|L|m:same|

```

```

cpc_downstream_cnn'
649 #' /home/julian/Downloads/Github/contrastive-predictive-coding/models/26_06_21-15-test|(2x)bl_MLP+bl_FCN+bl_TCN_block+
  bl_TCN_down+bl_TCN_flatten+bl_TCN_last+bl_alex_v2+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+
  bl_cnn_v14+bl_cnn_v15+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4+bl_cnn_v5+bl_cnn_v6+bl_cnn_v7+bl_cnn_v8+bl_cnn_v9/21_05_21-11-
  train|bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v14+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4+bl_cnn_v5
  +bl_cnn_v6+bl_cnn_v8+bl_cnn_v9/architectures_baseline_challenge.baseline_cnn_v14.BaselineNet12|dte:120'
650 test_labels, classes = read_binary_label_csv_from_model_folder(model_test_folder, 0)
651 test_pred, _ = read_output_csv_from_model_folder(model_test_folder, 0)
652 val_labels, classes = read_binary_label_csv_from_model_folder(model_test_folder, 1)
653 val_pred, _ = read_output_csv_from_model_folder(model_test_folder, 1)
654 save_path='/home/julian/Documents/projekt-master/bilder/errorplot-prediction-cpc.png'
655 calculate_plot(test_labels, test_pred, model_name='CPC', save_path=None)
656
657
658 # In[8]:
659
660
661 # model_test_folder = '/home/julian/Downloads/Github/contrastive-predictive-coding/models/16_08_21-10-16-test|(40x)cpc
  /14_08_21-15_36-train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined0|train-test-splits-fewer-labels60|
  use_weights|frozen|C|m:all|cpc_downstream_cnn'
662 model_test_folder = '://home/julian/Downloads/Github/contrastive-predictive-coding/models/30_11_21-20-25-test|
  bl_cnn_v14/30_11_21-18-train|bl_cnn_v14/architectures_baseline_challenge.baseline_cnn_v14.BaselineNet0|
  use_weights|ConvLyrs:29|MaxPool|Linear|BatchNorm|stride_sum:71|dilation_sum:30|padding_sum:22|krnls_sum:143'
663 test_labels, classes = read_binary_label_csv_from_model_folder(model_test_folder, 0)
664 test_pred, _ = read_output_csv_from_model_folder(model_test_folder, 0)
665 val_labels, classes = read_binary_label_csv_from_model_folder(model_test_folder, 1)
666 val_pred, _ = read_output_csv_from_model_folder(model_test_folder, 1)
667 save_path='/home/julian/Documents/projekt-master/bilder/errorplot-prediction-blv15.png'
668 calculate_plot(test_labels, test_pred, model_name='BL_v15', save_path=None)
669
670
671 # ## with threshold reweighting
672
673 # ### CPC
674
675 # In[10]:
676
677
678 model_test_folder = '/home/julian/Downloads/Github/contrastive-predictive-coding/models/16_08_21-10-16-test|(40x)cpc
  /14_08_21-15_36-train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined0|train-test-splits-fewer-labels60|
  use_weights|frozen|C|m:all|cpc_downstream_cnn'
679 #model_test_folder = '/home/julian/Downloads/Github/contrastive-predictive-coding/models/26_06_21-15-test|(2x)bl_MLP+
  bl_FCN+bl_TCN_block+bl_TCN_down+bl_TCN_flatten+bl_TCN_last+bl_alex_v2+bl_cnn_v0_1+bl_cnn_v0_2+
  bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v14+bl_cnn_v15+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4+bl_cnn_v5+bl_cnn_v6+bl_cnn_v7+
  bl_cnn_v8+bl_cnn_v9/21_05_21-11-train|bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v14+bl_cnn_v2+
  bl_cnn_v3+bl_cnn_v4+bl_cnn_v5+bl_cnn_v6+bl_cnn_v8+bl_cnn_v9/architectures_baseline_challenge.baseline_cnn_v14.
  BaselineNet12|dte:120'
680 test_labels, classes = read_binary_label_csv_from_model_folder(model_test_folder, 0)
681 test_pred, _ = read_output_csv_from_model_folder(model_test_folder, 0)
682 val_labels, classes = read_binary_label_csv_from_model_folder(model_test_folder, 1)
683 val_pred, _ = read_output_csv_from_model_folder(model_test_folder, 1)
684 #for now use ready file:
685 #threshold_file = '/home/julian/Downloads/Github/contrastive-predictive-coding/images/explain/BaselineNet12/thresholds.
  txt'
686 threshold_file = '/home/julian/Downloads/Github/contrastive-predictive-coding/images/explain/CPCCombined0/thresholds.
  txt'
687 with open(threshold_file, 'r') as f:
688     data = f.read()
689 thresholds = np.array(list(map(lambda x: x.item(), map(eval, data.split(','))))) #'slightly' dangerous
690
691
692 # In[10]:
693
694
695 pos = np.where(test_labels==0, test_pred, np.nan)
696 neg = np.where(test_labels!=0, test_pred, np.nan)
697 x1 = np.random.normal(scale=0.1, size=pos.shape)+np.arange(67)[np.newaxis, :]
698 plt.figure(figsize=[6.4*3, 4.8*3], dpi=400)
699 plt.scatter(x1, pos, s=1)
700 x2 = np.random.normal(scale=0.1, size=neg.shape)+np.arange(67)[np.newaxis, :]
701 plt.scatter(x2, neg, s=1)
702 plt.xlabel('Class')
703 plt.ylabel("Probability")
704 #plt.savefig('/home/julian/Documents/projekt-master/bilder/errorplot-scattered-cpc.png', dpi=400)
705
706 plt.show()
707
708
709 # In[12]:
710
711
712 diff = (test_pred - thresholds)
713 transformed_probs = (np.where(diff<0, diff/thresholds, diff/(1-thresholds))+1.)/2.
714 pos = np.where(test_labels==0, transformed_probs, np.nan)
715 neg = np.where(test_labels!=0, transformed_probs, np.nan)
716 x1 = np.random.normal(scale=0.1, size=pos.shape)+np.arange(67)[np.newaxis, :]
717 plt.figure(figsize=[6.4*3, 4.8*3], dpi=400)
718 plt.scatter(x1, pos, s=0.1)
719 x2 = np.random.normal(scale=0.1, size=neg.shape)+np.arange(67)[np.newaxis, :]
720 plt.scatter(x2, neg, s=0.1)

```

```

721 plt.xlabel('Class')
722 plt.ylabel("Probability")
723 plt.savefig('/home/julian/Documents/projekt-master/bilder/errorplot-scattered-thresholds-cpc.png', dpi=400)
724 plt.show()
725
726
727 # In[13]:
728
729
730 calculate_plot_with_thresholds(test_labels, test_pred, thresholds, model_name='CPC',
731                               save_path='/home/julian/Documents/projekt-master/bilder/errorplot-prediction-threshold-
732                               cpc-twodirectional.png',
733                               sort_fn=None, std_two_directional=True)
734
735 # In[14]:
736
737
738 calculate_plot_with_thresholds(test_labels, test_pred, thresholds, model_name='CPC',
739                               save_path='/home/julian/Documents/projekt-master/bilder/errorplot-prediction-threshold-
740                               cpc.png',
741                               sort_fn=None, std_two_directional=False)
742
743 # In[15]:
744
745
746 calculate_plot_with_thresholds(test_labels, test_pred, thresholds, model_name='CPC',
747                               save_path='/home/julian/Documents/projekt-master/bilder/errorplot-prediction-threshold-
748                               cpc-stdgap.png',
749                               sort_fn=sort_by_std_gap, std_two_directional=False)
750
751 # In[16]:
752
753
754 calculate_plot_with_thresholds(test_labels, test_pred, thresholds, model_name='CPC',
755                               save_path='/home/julian/Documents/projekt-master/bilder/errorplot-prediction-threshold-
756                               cpc-meangap.png',
757                               sort_fn=sort_by_mean_gap, std_two_directional=False)
758
759 # ### BL
760
761 # In[17]:
762
763
764 # model_test_folder = '/home/julian/Downloads/Github/contrastive-predictive-coding/models/16_08_21-10-16-test|(40x)cpc
765 # /14_08_21-15_36-train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined0|train-test-splits-fewer-labels60|
766 # use_weights|frozen|C|m:all|cpc_downstream_cnn'
767 model_test_folder = '/home/julian/Downloads/Github/contrastive-predictive-coding/models/11_08_21-15_58-test|(10x)
768 bl_TCN_down+(10x)bl_cnn_v1+(10x)bl_cnn_v14+(10x)bl_cnn_v15+(10x)bl_cnn_v8/10_08_21-14_42-train|bl_TCN_down+
769 bl_cnn_v1+bl_cnn_v14+bl_cnn_v15+bl_cnn_v8/architectures_baseline_challenge.baseline_cnn_v15.BaselineNet4|train-
770 test-splits-fewer-labels60|use_weights|ConvLyrs:5|MaxPool|Linear|BatchNorm|stride_sum:14|dilation_sum:96|
771 padding_sum:0|krnls_sum:26'
772 test_labels, classes = read_binary_label_csv_from_model_folder(model_test_folder, 0)
773 test_pred, _ = read_output_csv_from_model_folder(model_test_folder, 0)
774 val_labels, classes = read_binary_label_csv_from_model_folder(model_test_folder, 1)
775 val_pred, _ = read_output_csv_from_model_folder(model_test_folder, 1)
776 #for now use ready file:
777
778 # threshold_file = '/home/julian/Downloads/Github/contrastive-predictive-coding/images/explain/CPCCombined0/thresholds
779 # .txt'
780 threshold_file = '/home/julian/Downloads/Github/contrastive-predictive-coding/images/explain/BaselineNet4/thresholds.
781 txt'
782 with open(threshold_file, 'r') as f:
783     data = f.read()
784 thresholds = np.array(list(map(lambda x: x.item(), map(eval, data.split(','))))) #'slightly' dangerous
785
786 # In[18]:
787
788
789 pos = np.where(test_labels==0, test_pred, np.nan)
790 neg = np.where(test_labels!=0, test_pred, np.nan)
791 x1 = np.random.normal(scale=0.1, size=pos.shape)+np.arange(67)[np.newaxis, :]
792 plt.figure(figsize=[6.4*3, 4.8*3], dpi=400)
793 plt.scatter(x1, pos, s=0.1)
794 x2 = np.random.normal(scale=0.1, size=neg.shape)+np.arange(67)[np.newaxis, :]
795 plt.scatter(x2, neg, s=0.1)
796 plt.title("BL_v15")
797 plt.xlabel('Class')
798 plt.ylabel("Probability")
799 plt.savefig('/home/julian/Documents/projekt-master/bilder/errorplot-scattered-bl.png', dpi=400)
800 plt.show()
801
802 # In[19]:
803
804

```

```

799 diff = (test_pred - thresholds)
800 transformed_probs = (np.where(diff<0, diff/thresholds, diff/(1-thresholds))+1.)/2.
801 pos = np.where(test_labels==0, transformed_probs, np.nan)
802 neg = np.where(test_labels!=0, transformed_probs, np.nan)
803 x1 = np.random.normal(scale=0.1, size=pos.shape)+np.arange(67)[np.newaxis, :]
804 plt.figure(figsize=[6.4*3, 4.8*3], dpi=400)
805 plt.scatter(x1, pos, s=0.1)
806 x2 = np.random.normal(scale=0.1, size=neg.shape)+np.arange(67)[np.newaxis, :]
807 plt.scatter(x2, neg, s=0.1)
808 plt.title('BL_v15')
809 plt.xlabel('Class')
810 plt.ylabel("Probability")
811 plt.savefig('/home/julian/Documents/projekt-master/bilder/errorplot-scattered-thresholds-bl.png', dpi=400)
812 plt.show()
813
814
815 # In[20]:
816
817
818 calculate_plot_with_thresholds(test_labels, test_pred, thresholds, model_name='BL_v15',
819                                save_path='/home/julian/Documents/projekt-master/bilder/errorplot-prediction-threshold-
820                                blv15-twodirectional.png',
821                                sort_fn=None, std_two_directional=True)#
822
823
824
825
826 calculate_plot_with_thresholds(test_labels, test_pred, thresholds, model_name='BL_v15',
827                                save_path='/home/julian/Documents/projekt-master/bilder/errorplot-prediction-threshold-
828                                blv15.png',
829                                sort_fn=None, std_two_directional=False)
830
831
832
833
834 calculate_plot_with_thresholds(test_labels, test_pred, thresholds, model_name='BL_v15',
835                                save_path='/home/julian/Documents/projekt-master/bilder/errorplot-prediction-threshold-
836                                blv15-stdgap.png',
837                                sort_fn=sort_by_std_gap, std_two_directional=False)
838
839
840
841
842 calculate_plot_with_thresholds(test_labels, test_pred, thresholds, model_name='BL_v15',
843                                save_path='/home/julian/Documents/projekt-master/bilder/errorplot-prediction-threshold-
844                                blv15-meangap.png',
845                                sort_fn=sort_by_mean_gap, std_two_directional=False)
846
847
848
849
850 import glob
851 import os
852 import pandas as pd
853 import numpy as np
854 import matplotlib.pyplot as plt
855 import seaborn as sns
856
857 def load_model_test_files(test_folder):
858     val_label, classes = read_output_csv_from_model_folder(test_folder, 1)
859     test_label, _ = read_output_csv_from_model_folder(test_folder, 0)
860     val_pred, _ = read_label_csv_from_model_folder(test_folder, 1)
861     test_pred, _ = read_label_csv_from_model_folder(test_folder, 0)
862     return val_label.set_index("Unnamed: 0"), test_label.set_index("Unnamed: 0"), val_pred.set_index("Unnamed: 0"),
863             test_pred.set_index("Unnamed: 0"), classes
864
865 def read_output_csv_from_model_folder(model_folder, data_loader_index=0):
866     pred_path = glob.glob(os.path.join(model_folder, f"model-*-dataloader-{data_loader_index}-output.csv"))
867     if len(pred_path) == 0:
868         raise FileNotFoundError
869     dfp = pd.read_csv(pred_path[0])
870     return dfp, dfp.columns[1:].values # 1 is file
871
872 def read_label_csv_from_model_folder(model_folder, data_loader_index=0):
873     label_path = glob.glob(os.path.join(model_folder, f"labels-dataloader-{data_loader_index}.csv"))
874     if len(label_path) == 0:
875         raise FileNotFoundError
876     dfl = pd.read_csv(label_path[0])
877     labels = dfl
878     return labels, dfl.columns[1:].values
879
880
881
882
883 # import glob
884 # import os
885 # import pandas as pd
886 # def read_label_csv_from_model_folder(model_folder, data_loader_index=0):
887 #     label_path = glob.glob(os.path.join(model_folder, f"labels-dataloader-{data_loader_index}.csv"))

```

```

884 #     if len(label_path) == 0:
885 #         raise FileNotFoundError
886 #     dfl = pd.read_csv(label_path[0])
887 #     labels = dfl.values[:, 1:].astype(float)
888 #     return labels, dfl.columns[1:].values
889
890 # def read_binary_label_csv_from_model_folder(model_folder, data_loader_index=0):
891 #     l, c = read_label_csv_from_model_folder(model_folder, data_loader_index)
892 #     l = (l > 0).astype(int)
893 #     return l, c
894
895 # def read_output_csv_from_model_folder(model_folder, data_loader_index=0):
896 #     pred_path = glob.glob(os.path.join(model_folder, f"model--dataloader-{data_loader_index}-output.csv"))
897 #     if len(pred_path) == 0:
898 #         raise FileNotFoundError
899 #     dfp = pd.read_csv(pred_path[0])
900 #     return dfp.values[:, 1:].astype(float), dfp.columns[1:].values #l is file
901
902
903 # def read_binary_label_csv_from_model_folder(model_folder, data_loader_index=0):
904 #     l, c = read_label_csv_from_model_folder(model_folder, data_loader_index)
905 #     l = (l > 0).astype(int)
906 #     return l, c
907
908
909 # # CPC
910
911 # In[60]:
912
913
914 model_test_folder = '/home/julian/Downloads/Github/contrastive-predictive-coding/models/07_02_22-15-03-test|(36x)cpc
915 #/07_02_22-13-29-train|(30x)cpc/architectures_cpc.cpc_combined.CPCCombined14|train-test-splits|use_weights|
916 #unfrozen|C|L|m:crossentropy|cpc_downstream_twolinear_v2'
917 test_labels, classes = read_label_csv_from_model_folder(model_test_folder, 0)
918 test_pred, _ = read_output_csv_from_model_folder(model_test_folder, 0)
919 val_labels, classes = read_label_csv_from_model_folder(model_test_folder, 1)
920 val_pred, _ = read_output_csv_from_model_folder(model_test_folder, 1)
921 threshold_file = os.path.join(model_test_folder, 'thresholds.txt')
922 with open(threshold_file, 'r') as f:
923     data = f.read()
924 thresholds = np.array(list(map(float, data.split(',')))) #'slightly' dangerous
925
926 # In[61]:
927
928 mask = test_pred.values[:, 1:]>=thresholds
929 diff = (test_pred.values[:, 1:] - thresholds)
930 transformed_probs = (np.where(diff<0, diff/thresholds, diff/(1-thresholds))+1.)/2.
931 #test_pred_transformed = test_pred.copy()
932 test_pred_transformed = pd.DataFrame(test_pred.values[:, 1:], columns=test_pred.columns[1:], dtype=float)#
933 transformed_probs
934 #test_pred_transformed.index = test_pred["Unnamed: 0"]
935 X = pd.melt(test_pred_transformed, var_name='class', value_name='Predicted Probability')
936 X['Label1'] = pd.melt(test_labels.set_index("Unnamed: 0").astype(bool), var_name='class', value_name='Predicted
937     Probability')['Predicted Probability']
938 #codes = X['class'].astype('category').cat.codes
939 #X['class'] = codes
940
941 # In[62]:
942
943 plt.rc("font", size=24)
944 fig = plt.figure(figsize=(40,40/np.sqrt(2)), dpi=400)
945
946 ax = sns.violinplot(x='class', y='Predicted Probability', hue='Label', data=X, palette="Set2",
947                      scale="width", split=True, cut=0.)
948 ax.hlines(y=thresholds, xmin=ax.get_xticks()-0.5, xmax=ax.get_xticks()+0.5)
949 plt.xticks(rotation=90)
950 plt.savefig('/home/julian/Downloads/Multi-Download/master/cpc-probdistribution-unfrozen-standard-width.png', dpi=fig.
951 dpi)
952 plt.show()
953
954 # In[63]:
955
956
957 mask = test_pred.values[:, 1:]>=thresholds
958 diff = (test_pred.values[:, 1:] - thresholds)
959 transformed_probs = (np.where(diff<0, diff/thresholds, diff/(1-thresholds))+1.)/2.
960 #test_pred_transformed = test_pred.copy()
961 test_pred_transformed = pd.DataFrame(transformed_probs, columns=test_pred.columns[1:], dtype=float)#
962 transformed_probs
963 X = pd.melt(test_pred_transformed, var_name='class', value_name='Predicted Probability')
964 X['Label1'] = pd.melt(test_labels.set_index("Unnamed: 0").astype(bool), var_name='class', value_name='Predicted
965     Probability')['Predicted Probability']
966 #codes = X['class'].astype('category').cat.codes
967 #X['class'] = codes
968

```

```

968 # In[64]:
969
970
971
972 plt.rc("font", size=24)
973 fig = plt.figure(figsize=(40,40/np.sqrt(2)), dpi=400)
974
975 ax = sns.violinplot(x='class', y='Predicted Probability', hue='Label', data=X, palette="Set2",
976                      scale="width", split=True, cut=0.)
977 ax.axhline(y=0.5)
978 plt.xticks(rotation=90)
979 plt.savefig('/home/julian/Downloads/Multi-Download/master/cpc-probdistribution-unfrozen-reweighted-width.png', dpi=fig.dpi)
980 plt.show()
981
982
983 # # BL
984
985 # In[65]:
986
987
988 model_test_folder = '/home/julian/Downloads/Github/contrastive-predictive-coding/models/15_12_21-15-57-test|bl_FCN+
989     bl_MLP+bl_TCN_block+bl_TCN_down+bl_TCN_last+bl_alex_v2+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+
990     bl_cnn_v14+bl_cnn_v15+bl_cnn_v2+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+bl_cnn_v8+bl_cnn_v9/14_12_21-10-train|bl_MLP+
991     bl_alex_v2+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v2+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+
992     bl_cnn_v8+bl_cnn_v9/architectures_baseline_challenge.baseline_cnn_v14.BaselineNet3|use_weights|ConvLyrs:29|
993     MaxPool|Linear|BatchNorm|stride_sum:71|dilation_sum:30|padding_sum:22|krnls_sum:143'
994 test_labels, classes = read_label_csv_from_model_folder(model_test_folder, 0)
995 test_pred, _ = read_output_csv_from_model_folder(model_test_folder, 0)
996 val_labels, classes = read_label_csv_from_model_folder(model_test_folder, 1)
997 val_pred, _ = read_output_csv_from_model_folder(model_test_folder, 1)
998 threshold_file = os.path.join(model_test_folder, 'thresholds.txt')
999 with open(threshold_file, 'r') as f:
1000     data = f.read()
1001 thresholds = np.array(list(map(float, data.split(','))))
1002
1003 # In[66]:
1004
1005 mask = test_pred.values[:, 1:]>thresholds
1006 diff = (test_pred.values[:, 1:] - thresholds)
1007 transformed_probs = (np.where(diff<0, diff/thresholds, diff/(1-thresholds))+1.)/2.
1008 #test_pred_transformed = test_pred.copy()
1009 test_pred_transformed = pd.DataFrame(test_pred.values[:, 1:], columns=test_pred.columns[1:], dtype=float)#
1010     transformed_probs
1011 #test_pred_transformed.index = test_pred["Unnamed: 0"]
1012 X = pd.melt(test_pred_transformed, var_name='class', value_name='Predicted Probability')
1013 X['Label'] = pd.melt(test_labels.set_index("Unnamed: 0").astype(bool), var_name='class', value_name='Predicted
1014     Probability')['Predicted Probability']
1015 #codes = X['class'].astype('category').cat.codes
1016 #X['class'] = codes
1017
1018 # In[67]:
1019
1020 plt.rc("font", size=24)
1021 fig = plt.figure(figsize=(40,40/np.sqrt(2)), dpi=400)
1022
1023 ax = sns.violinplot(x='class', y='Predicted Probability', hue='Label', data=X, palette="Set2",
1024                      scale="width", split=True, cut=0.)
1025 ax.hlines(y=thresholds, xmin=ax.get_xticks()-0.5, xmax=ax.get_xticks()+0.5)
1026 plt.xticks(rotation=90)
1027 plt.savefig('/home/julian/Downloads/Multi-Download/master/bl-probdistribution-standard-width.png', dpi=fig.dpi)
1028 plt.show()
1029
1030 # In[68]:
1031
1032 mask = test_pred.values[:, 1:]>thresholds
1033 diff = (test_pred.values[:, 1:] - thresholds)
1034 transformed_probs = (np.where(diff<0, diff/thresholds, diff/(1-thresholds)) + 1.) / 2.
1035 #test_pred_transformed = test_pred.copy()
1036 test_pred_transformed = pd.DataFrame(transformed_probs, columns=test_pred.columns[1:], dtype=float)#
1037 #test_pred_transformed.index = test_pred["Unnamed: 0"]
1038 X = pd.melt(test_pred_transformed, var_name='class', value_name='Predicted Probability')
1039 X['Label'] = pd.melt(test_labels.set_index("Unnamed: 0").astype(bool), var_name='class', value_name='Predicted
1040     Probability')['Predicted Probability']
1041 #codes = X['class'].astype('category').cat.codes
1042 #X['class'] = codes
1043 # In[69]:
1044
1045 plt.rc("font", size=24)
1046 fig = plt.figure(figsize=(40,40/np.sqrt(2)), dpi=400)
1047

```

```

1049 ax = sns.violinplot(x='class', y='Predicted Probability', hue='Label', data=X, palette="Set2",
1050                     scale="width", split=True, cut=0.)
1051 ax.axhline(y=0.5)
1052 plt.xticks(rotation=90)
1053 plt.savefig('/home/julian/Downloads/Multi-Download/master/bl-probdistribution-reweighted-width.png', dpi=fig.dpi)
1054 plt.show()
1055
1056
1057 # In[ ]:
```

B.7.10 jupyter_notebooks -> PTB PCA.py (code)

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[4]:
5
6
7 import wfdb
8 import numpy as np
9 import os
10 import pandas as pd
11 import math
12 import h5py
13 import matplotlib.pyplot as plt
14
15
16 # In[5]:
17
18
19 def print_object_attributes(obj): #https://stackoverflow.com/questions/192109/is-there-a-built-in-function-to-print-
20     #all-the-current-properties-and-values-of-a
21     for attr in dir(obj):
22         print("obj.%s = %r" % (attr, getattr(obj, attr)))
23
24
25 # ### Read MIT format .dat ecg data files and .hea headers
26
27 # In[8]:
28
29 #Download first at: https://physionet.org/content/ptbdb/1.0.0/
30 #BASE_DIR = '/media/julian/Volume/data/ECG/mit-bih-arrhythmia-database-1.0.0/' #Arrhythmia
31 BASE_DIR = '/media/julian/Volume/data/ECG/ptb-diagnostic-ecg-database-1.0.0/'
32 def get_file_list(BASE_DIR, relative=True):
33     record_files = []
34     file_endings = ['.dat', '.hea', '.xyz']
35     with open(os.path.join(BASE_DIR, 'RECORDS')) as recs:
36         record_files = recs.read().splitlines()
37     if not relative:
38         record_files = [os.path.join(BASE_DIR, f) for f in record_files]
39     return record_files
40 record_files = get_file_list(BASE_DIR)
41 print(len(record_files))
42 print(record_files[:10])
43
44
45 # ### Extract signal from *.dat files & Read annotations & Read comments
46
47 # In[9]:
48
49
50 def plot_cov(co, title=None):
51     if title:
52         plt.title(title)
53     plt.imshow(co)
54     plt.colorbar()
55     plt.show()
56
57
58 # In[10]:
59
60
61 def read_comment_map_PTB(record_path):
62     #print(record_path)
63     record = wfdb.rdrecord(record_path)
64     comment_map = {}
65     for c in record.comments:
66         e = c.split(':')
67         comment_map[e[0]] = e[1].strip()
68     return comment_map
69
70
71 # In[11]:
72
73
```

```

74 def filter_comment(comment, key):
75     c = comment
76     if key == 'Reason for admission':
77         if 'Cardiomyopathy' in c or 'Heart failure' in c:
78             return 'Cardiomyopathy'
79         elif 'n/a' in k or 'Palpitation' in k:
80             return 'Miscellaneous'
81         elif 'angina' in k:
82             new_comments['Angina'] = comments[k]
83         else:
84             new_comments[k] = comments[k]
85
86
87 # In[12]:
88
89
90 def read_comment(record_path):
91     record = wfdb.rdrecord(record_path)
92     return record.comments
93
94
95 # In[13]:
96
97
98 def read_header(record_path):
99     record = wfdb.rdheader(record_path, rd_segments=True)
100    return record.comments
101
102
103 # In[14]:
104
105
106 def read_signal(record_path, physical=True):
107     #print(record_path)
108     record = wfdb.rdrecord(record_path, physical=physical)
109     #print_object_attributes(record)
110     if physical:
111         data = record.p_signal
112     else:
113         data = record.d_signal
114     return data
115
116
117 # In[15]:
118
119
120 def read_annotation(record_path, physical=True):
121     try:
122         annotation = wfdb.rdnann(record_path, 'hea', return_label_elements=['symbol', 'label_store', 'description'])
123         #print(record_path)
124         #print('sample:', annotation.sample, 'symbol', annotation.symbol, 'contained labels', annotation.description)
125         return (annotation.sample, annotation.symbol, annotation.label_store, annotation.description)
126     except ValueError as ve:
127         print(record_path, 'annotation read failed:', ve)
128         return None
129
130
131 # #### Save all signals and attributes in file_data (also note how many had functioning annotations)
132
133 # In[16]:
134
135
136 file_data = []
137 success = 0
138 for f in record_files[:]:
139     p = os.path.join(BASE_DIR, f)
140     d = read_signal(p, physical=False)
141     file_data.append((f, d))
142
143
144 # In[18]:
145
146
147 import numpy as np
148 from numpy import linalg as LA
149 from sklearn.preprocessing import normalize
150 for f in file_data:
151     data = normalize(f[1], norm='l2').T
152     co = np.cov(data)
153     w, v = LA.eig(co)
154     w = np.expand_dims(w, 0)
155     plot_cov(co, f[0])
156     plot_cov(w, f[0])
157
158
159 # In[ ]:
```

B.7.11 jupyter_notebooks -> PTBx1 data.py (code)

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 import wfdb
8 import numpy as np
9 import os
10 import pandas as pd
11 import math
12 import h5py
13 import matplotlib.pyplot as plt
14 import re
15
16
17 # In[2]:
18
19
20 def print_object_attributes(obj): #https://stackoverflow.com/questions/192109/is-there-a-built-in-function-to-print-
21     #all-the-current-properties-and-values-of-a
22     for attr in dir(obj):
23         print("obj.%s = %r" % (attr, getattr(obj, attr)))
24
25 # ### Read MIT format .dat ecg data files and .hea headers
26
27 # In[3]:
28
29
30 #Download first at: https://physionet.org/content/ptbdb/1.0.0/
31 #BASE_DIR = '/media/julian/Volume/data/ECG/mit-bih-arrhythmia-database-1.0.0/' #Arrhythmia
32 BASE_DIR = '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/'
33 def get_file_list(BASE_DIR, relative=True, filter_function=None):
34     record_files = []
35     file_endings = ['.dat', '.hea', '.xyz']
36     with open(os.path.join(BASE_DIR, 'RECORDS')) as recs:
37         record_files = recs.read().splitlines()
38     if filter_function:
39         record_files = list(filter(filter_function, record_files))
40     if not relative:
41         record_files = [os.path.join(BASE_DIR, f) for f in record_files]
42     return record_files
43
44
45 record_files = get_file_list(BASE_DIR)
46 print(len(record_files), 'files found')
47
48
49 # ### Extract signal from *.dat files & Read annotations & Read comments
50
51 # In[4]:
52
53
54 def read_comment_map_PTB(record_path):
55     #print(record_path)
56     record = wfdb.rdrecord(record_path)
57     comment_map = {}
58     for c in record.comments:
59         e = c.split(':')
60         comment_map[e[0]] = e[1].strip()
61     return comment_map
62
63
64 # In[5]:
65
66
67 def filter_comment(comment, key):
68     c = comment
69     if key == 'Reason for admission':
70         if 'Cardiomyopathy' in c or 'Heart failure' in c:
71             return 'Cardiomyopathy'
72         elif '/a' in c or 'Palpitation' in c:
73             return 'Miscellaneous'
74         elif 'angina' in c:
75             new_comments['Angina'] = comments[k]
76         else:
77             new_comments[k] = comments[k]
78
79
80 # In[6]:
81
82
83 def read_comment(record_path):
84     record = wfdb.rdrecord(record_path)
85     return record.comments
86
87

```

```

88 # In[7]:
89
90
91 def read_header(record_path):
92     record = wfdb.rdheader(record_path, rd_segments=True)
93     return record.comments
94
95
96 # In[8]:
97
98
99 def read_signal(record_path, physical=True):
100     #print(record_path)
101     record = wfdb.rdrecord(record_path, physical=physical)
102     #print_object_attributes(record)
103     if physical:
104         data = record.p_signal
105     else:
106         data = record.d_signal
107     return data
108
109
110 # In[9]:
111
112
113 def read_annotation(record_path, physical=True):
114     try:
115         annotation = wfdb.rdann(record_path, 'hea', return_label_elements=['symbol', 'label_store', 'description'])
116         #print(record_path)
117         #print('sample:', annotation.sample, 'symbol', annotation.symbol, 'contained labels', annotation.label_store, annotation.description)
118         return (annotation.sample, annotation.symbol, annotation.label_store, annotation.description)
119     except ValueError as ve:
120         print(record_path, 'annotation read failed:', ve)
121     return None
122
123
124 # #### Save all signals and attributes in file_data (also note how many had functioning annotations)
125
126 # In[10]:
127
128
129 def read_ptbxl_database():
130     csvfile = os.path.join(BASE_DIR, 'ptbxl_database.csv')
131     dataframe = pd.read_csv(csvfile)
132     return dataframe
133
134 def read_ptbxl_scp_statements():
135     csvfile = os.path.join(BASE_DIR, 'scp_statements.csv')
136     dataframe = pd.read_csv(csvfile)
137     return dataframe
138
139 def train_test_split(record_files_relative):
140     df = read_ptbxl_csv()
141     train, val, test = [], [], []
142     for rf in record_files_relative:
143         temp = rf.replace('resampled', '') #resampled file contains 'resampled' but csv not
144         row = df.loc[(df['filename_hr'].str.contains(temp)) | (df['filename_lr'].str.contains(temp))]
145         if len(row) < 1:
146             print('no row found containing file', rf)
147             fold = row['strat_fold'].values[0]
148             if fold <= 8: #https://physionet.org/content/ptb-xl/1.0.1/ #Cross-validation Folds
149                 train.append(rf)
150             elif fold == 9:
151                 val.append(rf)
152             elif fold == 10:
153                 test.append(rf)
154             else:
155                 print('found unknown strat fold number', fold)
156             print("final split: train %d; validation %d; test %d" % (len(train), len(val), len(test)))
157     return train, val, test
158
159 def read_label(ptbxl_database_dataframe, spc_codes_dataframe, rf, likelihood_threshold=0.0):
160     df = ptbxl_database_dataframe
161     spc_df = spc_codes_dataframe
162     temp = rf.replace('resampled', '') #resampled file contains 'resampled' but csv not
163     row = df.loc[(df['filename_hr'].str.contains(temp)) | (df['filename_lr'].str.contains(temp))]
164     if len(row) < 1:
165         print(filename, 'not found in dataframe')
166     return
167 code = row['scp_codes'].values[0]
168 labels = [(re.sub(r'\W+', '', c.split(':')[0]), c.split(':')[1].replace(')', '').strip()) for c in code.split(',')]
169 diagnostic_classes = []
170 for l, p in labels:
171     if float(p) > likelihood_threshold:
172         scp_row = spc_df.loc[(spc_df.iloc[:, 0] == 1) | (spc_df['diagnostic_subclass'] == 1)]
173         if len(scp_row) < 1:
174             print(l, 'not found in scp_statements')
175             break
176         diagnostic_classes.append((scp_row['diagnostic_class'].values[0], p))

```

```

177     return sorted(diagnostic_classes, key=lambda x: x[1], reverse=True)
178 record_files500 = get_file_list(BASE_DIR, filter_function=lambda x: 'records500' in x)
179
180 db_df = read_ptbxl_database()
181 scp_df = read_ptbxl_scp_statements()
182 print(scp_df.iloc[:, 0])
183 read_label(db_df, scp_df, 'records500/00000/00001_hr')
184
185
186
187 # In[75]:
188
189
190 import ast
191 from collections import defaultdict
192 dbdf = read_ptbxl_database()
193 all_labels = defaultdict(set)
194
195 #Collect all possible labels
196 codes = dbdf['scp_codes']
197 for row in codes:
198     lbl_dict = ast.literal_eval(row)
199     for k,v in lbl_dict.items():
200         all_labels[k].update({v})
201
202 #Filter out labels that have only 0 probability
203 all_labels = {k: v for k, v in all_labels.items() if max(v) > 0.0}
204
205 #Build a dict with filename as key and scp code as values
206 filenames = dbdf[['filename_hr', 'scp_codes']]
207 file_codes = defaultdict(dict)
208 for i, (f, c) in filenames.iterrows():
209     lbl_dict = ast.literal_eval(c)
210     for k, v in lbl_dict.items():
211         if k in all_labels and v > 0.0: #First check not necessary
212             file_codes[f][k] = v/100.0
213
214 code_indices = dict(zip(all_labels.keys(), range(len(all_labels.keys()))))
215 file_codes_onehot = dict()
216 for k, v in file_codes.items():
217     hot_prob = np.zeros(len(code_indices))
218     for ck, cv in v.items():
219         hot_prob[code_indices[ck]] = cv
220     file_codes_onehot[k] = hot_prob
221 print(len(filenames), len(file_codes_onehot))
222
223
224 # In[ ]:
225
226
227 [a[0] for a in sorted(code_indices.items(), key=lambda x: x[1])]
228
229
230 # In[11]:
231
232
233 import glob
234 p = '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/generated/1000'
235 for f in glob.glob(os.path.join(p, '*_hr.*')):
236     fn = os.path.basename(f)
237     fp = os.path.dirname(f)
238     os.rename(f, os.path.join(fp, os.path.splitext(fn)[0] + 'resampled' + os.path.splitext(fn)[1]))
239
240
241 # ## Playing around with PTB Comments
242
243 # In[20]:
244
245
246 comment_data = []
247 success = 0
248 record_files = get_file_list(BASE_DIR, filter_function=lambda x: 'records500' in x)
249 for f in record_files:
250     p = os.path.join(BASE_DIR, f)
251     d = read_comment_map_PTB(p)
252     comment_data.append(d)
253 print(comment_data[0])
254
255
256 # In[ ]:
257
258
259 comment_data
260
261
262 # In[34]:
263
264
265 for k in comment_data[0].keys():
266     plot([c[k] for c in comment_data], k)

```

```

267
268
269 # In[53]:
270
271
272 from collections import Counter
273 k = "Reason for admission"
274 class_counts = Counter([c[k] for c in comment_data])
275 print(class_counts)
276
277
278 # In[33]:
279
280
281 def plot(data, title): #https://stackoverflow.com/questions/6352740/matplotlib-label-each-bin
282     import matplotlib.pyplot as plt
283     import numpy as np
284     from matplotlib.ticker import FormatStrFormatter
285
286     fig, ax = plt.subplots()
287     counts, bins, patches = ax.hist(data, facecolor='blue', edgecolor='gray')
288     ax.set_title(title)
289     # Set the ticks to be at the edges of the bins.
290     ax.set_xticks(bins)
291     # Set the xaxis's tick labels to be formatted with 1 decimal place...
292     for tick in ax.get_xticklabels():
293         tick.set_rotation(45)
294     # Label the raw counts and the percentages below the x-axis...
295     bin_centers = 0.5 * np.diff(bins) + bins[:-1]
296     for count, x in zip(counts, bin_centers):
297         # Label the raw counts
298         ax.annotate(str(count), xy=(x, 0), xycoords='data', 'axes fraction'),
299             xytext=(0, -18), textcoords='offset points', va='top', ha='center')
300
301         # Label the percentages
302         percent = '%0.0f%%' % (100 * float(count) / counts.sum())
303         ax.annotate(percent, xy=(x, 0), xycoords='data', 'axes fraction'),
304             xytext=(0, -32), textcoords='offset points', va='top', ha='center')
305
306
307     # Give ourselves some more room at the bottom of the plot
308     plt.subplots_adjust(bottom=0.15)
309     plt.show()
310
311
312 # #### Plotting (does not seem to work for annotations)
313
314 # In[1]:
315
316
317 wfdb.plot_wfdb(record=file_data[0], plot_sym=True, time_units='samples', title='Test', figsize=(10,4), ecg_grids='all
318 ')
319
320
321 # ##### Method for partitioning data into windows with a specific overlap
322 # Warning: overlaps other than 0.5 have not been tested. Might also behave unexpected when 'shift' is not an Integer
323
324
325 # In[6]:
326
327 def partition_data(data, window_size=3000, overlap=0.5, store_in_array=True, align_right=True, verbose=True): #maybe
328     allow_non_float_overlap = True
329     samples, channels = data.shape
330     if samples < window_size:
331         print('too few samples (%d) to support window size of %d' % (samples, window_size))
332         return None
333     if verbose: print('Input data has shape:', data.shape)
334     shift = window_size*overlap
335     offset = int(samples % shift)
336     if align_right:
337         used_data = data[offset:]
338     else:
339         used_data = data[:-offset]
340     samples, _ = used_data.shape
341     if verbose: print('The window of size %d will be shifted by %f. The total data used is %d' % (window_size, shift,
342         samples))
343     partitioned = np.empty((int(samples/shift)-1, window_size, channels))
344     if verbose: print('The partitioned data now has shape:', partitioned.shape)
345     for i in range(len(partitioned)):
346         index = int(i*shift)
347         partitioned[i, :, :] = used_data[index:index+window_size, :]
348
349 # In[ ]:
350
351 partition_data(d) #nur zum testen
352
353

```

```

354 # In[8]:
355
356
357 #testing
358 print(partition_data(np.repeat(np.arange(10)[np.newaxis, :].T, 5, axis=1), window_size=4))
359 print(partition_data(np.repeat(np.arange(11)[np.newaxis, :].T, 5, axis=1), window_size=4))
360 print(partition_data(np.repeat(np.arange(9)[np.newaxis, :].T, 5, axis=1), window_size=4))
361 print(partition_data(np.repeat(np.arange(10)[np.newaxis, :].T, 5, axis=1), window_size=10))
362
363
364 # ##### Generate a (hopefully correct) context task for any given partitioned data
365 # *N* : The number of samples generated for each current window. If this number is too high (No more windows to
#       randomly sample from) numpy will throw an error.
366 #
367 # *observations* : The number of windows you have to predict the future.
368 #
369 # *predictions* : The number of windows that will be predicted.
370 #
371 # Total amount of data output will be (*Number of windows* - *observations* - *predictions*) * *N*
372
373 # In[7]:
374
375
376 def generate_context_task(partitioned_data, N, observations=5, predictions=3, verbose=True, shuffle_all=False): #maybe
    use_shuffle?
377     #generate N-1 negative samples and 1 positive sample:
378     windows, samples, channels = partitioned_data.shape
379     positive_samples_x = []
380     positive_samples_y = []
381     negative_samples_x = []
382     negative_samples_y = []
383     for i in range(0, windows - observations - predictions):
384         i_cur = i+observations
385         positive_samples_x += [partitioned_data[i:i_cur, :, :]]
386         positive_samples_y += [partitioned_data[i_cur:i_cur+predictions]]
387         possible_choices = list(range(i))+list(range(i_cur+predictions, windows))get_file_list(BASE_DIR)
388         for _ in range(N-1):
389             choices = np.random.choice(possible_choices, predictions, replace=False) #advanced use p param for
                different probabilities
390             negative_samples_x += [partitioned_data[i:i_cur, :, :]]
391             negative_samples_y += [partitioned_data[choices, :, :]]
392
393     return positive_samples_x, positive_samples_y, negative_samples_x, negative_samples_y
394
395
396 # In[10]:
397
398
399 #Testing
400 px, py, nx, ny = generate_context_task(partition_data(d), 10)
401
402
403 # In[121]:
404
405
406 (len(px), len(py)), (len(nx), len(ny))
407
408
409 # In[12]:
410
411
412 px[0].shape, py[0].shape
413
414
415 # In[18]:
416
417
418 def convert_dat_to_h5(storage_path, dat_file_paths, window_size=3000, overlap=0.5, align_right=True, verbose=True):
419     if not os.path.exists(storage_path):
420         os.makedirs(storage_path)
421     for f in dat_file_paths:
422         data = read_signal(os.path.join(BASE_DIR, f))
423         print(f)
424         partitioned = partition_data(data, window_size, overlap, True, align_right, verbose)
425         target = os.path.join(storage_path, f.replace('/', '-') +'.h5')
426         with h5py.File(target, 'w') as wf:
427             wf['windows'] = partitioned
428             wf.flush()
429             if verbose: print(target, 'file created and written. %d windows saved.' % (len(partitioned)))
430
431
432 # In[ ]:
433
434
435 record_files = get_file_list(BASE_DIR)
436 convert_dat_to_h5('test', record_files)
437
438
439 # In[17]:
440

```

```

441 BASE_DIR = '/media/julian/Volume/data/ECG/ptb-diagnostic-ecg-database-1.0.0/'
442 record_files = get_file_list(BASE_DIR, relative=False)
443
444 file_data = []
445 success = 0
446 for f in record_files:
447     header_record = read_header(f)
448     print(header_record.comments)
449     break
450
451
452 # In[44]:
453
454
455 import datetime
456 import time
457 str(datetime.datetime.now().strftime("%d_%m_%y-%H"))
458
459
460
461 # In[81]:
462
463
464 def parse_comment_dict(wfdb_comment):
465     comment_map = {}
466     for c in wfdb_comment:
467         e = c.lower().split(':')
468         comment_map[e[0]] = e[1].strip()
469     return comment_map
470 label_mappings = {}
471 def onehot_comment(wfdb_comment, terms: list, key_function_dict: dict = None):
472     onehots = []
473     if len(terms) > 0:
474         terms_encoded = np.zeros(len(terms), dtype=bool)
475         for i, term in enumerate(terms):
476             for c in wfdb_comment:
477                 if term in c:
478                     terms_encoded[i] = True
479             onehots.append(terms_encoded)
480     comment_dict = parse_comment_dict(wfdb_comment)
481     for key, (func, n) in key_function_dict.items():
482         if not key in label_mappings:
483             label_mappings[key] = {}
484         value = func(comment_dict[key])
485         if not value in label_mappings[key]:
486             label_mappings[key][value] = len(label_mappings[key])
487             encoded_key = np.zeros(n, dtype=bool)
488             encoded_key[label_mappings[key][value]] = value
489             onehots.append(encoded_key)
490     return np.concatenate(onehots)
491
492 def filter_comment(key, comment_string):
493     c = comment_string
494     if key == 'reason for admission':
495         if 'cardiomyopathy' in c or 'heart failure' in c:
496             return 'cardiomyopathy'
497         elif 'n/a' in c or 'palpitation' in c:
498             return 'miscellaneous'
499         elif 'angina' in c:
500             return 'angina'
501     return comment_string
502
503
504 # In[82]:
505
506
507 from collections import defaultdict
508 default_key_function_dict = defaultdict(lambda y: (lambda x: True, 2))
509 default_key_function_dict['age'] = lambda x: int(x)>50 if x.isnumeric() else False, 2 #age
510 default_key_function_dict['smoker'] = lambda x: x == 'yes', 2
511 default_key_function_dict['reason for admission'] = lambda x: filter_comment('reason for admission', x), 10
512
513
514 # In[ ]:
515
516
517 BASE_DIR = '/media/julian/Volume/data/ECG/ptb-diagnostic-ecg-database-1.0.0/'
518 record_files = get_file_list(BASE_DIR, relative=False)
519
520 for f in record_files:
521     wfdb_comment = read_header(f)
522     print(onehot_comment(wfdb_comment, [], default_key_function_dict))
523
524
525 # In[85]:
526
527
528 label_mappings
529
530

```

```
531 # In[20]:  
532  
533  
534 import numpy as np  
535 def generate_sin_data(n, hz_low, hz_high):  
536     m_phase=0  
537  
538     signal = np.empty(n)  
539  
540     f = hz_low  
541     fs= 44100  
542  
543     phaseInc = 2*np.pi*f/fs  
544  
545  
546     for i in range(int(n/2)):  
547         signal[i] = np.sin(m_phase)  
548         m_phase = m_phase + phaseInc  
549  
550     m_phase = m_phase % 2*np.pi  
551  
552  
553     f=hz_high  
554     phaseInc = 2*np.pi*f/fs  
555  
556  
557     for i in range(int(n/2), n):  
558         signal[i] = np.sin(m_phase)  
559         m_phase = m_phase + phaseInc  
560  
561     m_phase = m_phase % 2*np.pi  
562     return signal  
563  
564  
565 # In[23]:  
566  
567  
568 import matplotlib.pyplot as plt  
569  
570 plt.plot(generate_sin_data(1005, 500, 1000))  
571 plt.show()  
572  
573  
574 # In[19]:  
575  
576  
577 record_files = get_file_list(BASE_DIR, filter_function=lambda x: 'records500' in x)  
578 print(len(record_files), 'files found')  
579  
580  
581 # In[6]:  
582  
583  
584 import torch as t  
585  
586  
587 # In[25]:  
588  
589  
590 tar = t.zeros((4,8))  
591 tar[0,1] = 1.  
592 tar  
593 y = tar  
594  
595  
596 # In[26]:  
597  
598  
599 pred = torch.rand_like(tar)  
600 pred[:, 1:5] = 0.  
601 pred  
602 logits = pred  
603  
604  
605 # In[24]:  
606  
607  
608 t.sum((pred != 0.0) | (tar != 0.0))  
609  
610  
611 # In[37]:  
612  
613  
614 t.sum(t.abs(y-pred) <= 0.000005) / (4*8)  
615  
616  
617 # In[4]:  
618  
619  
620 import pandas as pd
```

```

621 import numpy as np
622 import wfdb
623 import ast
624
625 def load_raw_data(df, sampling_rate, path):
626     if sampling_rate == 100:
627         data = [wfdb.rdsamp(path+f) for f in df.filename_lr]
628     else:
629         data = [wfdb.rdsamp(path+f) for f in df.filename_hr]
630     data = np.array([signal for signal, meta in data])
631     return data
632
633 path = '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/'
634 sampling_rate=500
635
636 # load and convert annotation data
637 Y = pd.read_csv(path+'ptbxl_database.csv', index_col='ecg_id')
638 Y.scp_codes = Y.scp_codes.apply(lambda x: ast.literal_eval(x))
639
640 # Load raw signal data
641 #X = load_raw_data(Y, sampling_rate, path)
642
643 # Load scp_statements.csv for diagnostic aggregation
644 agg_df = pd.read_csv(path+'scp_statements.csv', index_col=0)
645 agg_df = agg_df[agg_df.diagnostic == 1]
646
647 def aggregate_diagnostic(y_dic):
648     tmp = []
649     for key in y_dic.keys():
650         if key in agg_df.index:
651             tmp.append(agg_df.loc[key].diagnostic_class)
652     return list(set(tmp))
653
654 # Apply diagnostic superclass
655 Y['diagnostic_superclass'] = Y.scp_codes.apply(aggregate_diagnostic)
656
657 # Split data into train and test
658 test_fold = 10
659 # Train
660 #X_train = X[np.where(Y.strat_fold != test_fold)]
661 y_train = Y[(Y.strat_fold != test_fold)].diagnostic_superclass
662 # Test
663 #X_test = X[np.where(Y.strat_fold == test_fold)]
664 y_test = Y[Y.strat_fold == test_fold].diagnostic_superclass
665
666
667 # In[5]:
668
669
670 y_train

```

B.7.12 jupyter_notebooks -> Parallel Coordinates.py (code)

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 import plotly.graph_objects as go
8 import seaborn as sns
9 import matplotlib
10 import numpy as np
11 import pandas as pd
12 import matplotlib.pyplot as plt
13 from matplotlib import ticker
14 from pandas.plotting import parallel_coordinates
15 import plotly.express as px
16 from scipy.interpolate import make_interp_spline
17
18
19 # In[2]:
20
21
22 def parallel_plot(df: pd.DataFrame, color_column, exclude_columns:list=[], put_last_columns:list=[],
23     exclude_color_column=True, save_to=None):
24     cols = df.columns.tolist()
25     #cols = list(set(cols)-set(put_last_columns)-set(exclude_columns))+put_last_columns
26     cols = [c for c in cols if not (c in exclude_columns or c in put_last_columns)] + put_last_columns
27     df = df[cols]
28     dimensions = [_make_plotly_dict(column_name, data) for column_name, data in df.iteritems() if not (
29         exclude_color_column and column_name == color_column)]
30     fig = go.Figure(data=
31         go.Parcoords(
32             line = dict(color = df[color_column],
33                         colorscale = 'Jet',

```

```

32             #autocolorscale=True,
33             showscale = True,
34             cmin = df[color_column].min(),
35             cmax = df[color_column].max()),
36             dimensions = dimensions
37         )
38     )
39 fig.update_traces(labelangle=-90, selector=dict(type='parcoords'))
40 fig.update_traces(line_cmid=0.75, selector=dict(type='parcoords'))
41 fig.update_layout(width=1920)
42 fig.update_layout(height=1080)
43 fig.show();
44
45 def _make_plotly_dict(column_name, data):
46     d = dict()
47     t = data.dtype
48     if t == bool:
49         d['range'] = [-0.5,1.5]
50         d['tickvals'] = [True, False]
51         d['ticktext'] = ['True', 'False']
52         d['values'] = data
53     elif t == str or t == object:
54         da = data.astype('category').cat
55         d['tickvals'] = da.codes
56         d['ticktext'] = da.categories
57         d['values'] = da.codes
58     elif t == int:
59         print(column_name, 'is int')
60         d['range'] = [data.min(), data.max()]
61         d['tickformat'] = 'd'
62         d['values'] = data
63     else:
64         d['range'] = [data.min(), data.max()]
65         d['values'] = data
66     d['label'] = column_name
67     return d
68
69
70 # In[14]:
71
72
73
74 # https://github.com/jraine/parallel-coordinates-plot-dataframe/blob/master/parallel_plot.py
75 def parallel_plot(df, cols, rank_attr, cmap='magma', spread=None, curved=False, add_random=0.0, curvedextend=0.1, save
    =None, show=True):
76     """Produce a parallel coordinates plot from pandas dataframe with line colour with respect to a column.
77     Required Arguments:
78         df: dataframe
79         cols: columns to use for axes
80         rank_attr: attribute to use for ranking
81     Options:
82         cmap: Colour palette to use for ranking of lines
83         spread: Spread to use to separate lines at categorical values
84         curved: Spline interpolation along lines
85         curvedextend: Fraction extension in y axis, adjust to contain curvature
86     Returns:
87         x coordinates for axes, y coordinates of all lines"""
88     colmap = sns.color_palette(cmap, as_cmap=True)
89     cols = cols + [rank_attr]
90
91     fig, axes = plt.subplots(1, len(cols) - 1, sharey=False, figsize=(3 * len(cols) + 3, 5), dpi=800)
92     valmat = np.ndarray(shape=(len(cols), len(df)))
93     x = np.arange(0, len(cols), 1)
94     ax_info = {}
95     for i, col in enumerate(cols):
96         vals = df[col]
97         if (vals.dtype == float) and (len(np.unique(vals)) > 20):
98             minval = np.min(vals)
99             maxval = np.max(vals)
100            rangeval = maxval - minval
101            vals = np.true_divide(vals - minval, maxval - minval)
102            nticks = 5
103            tick_labels = [round(minval + i * (rangeval / nticks), 4) for i in range(nticks + 1)]
104            ticks = [0 + i * (1.0 / nticks) for i in range(nticks + 1)]
105            valmat[i] = vals
106            ax_info[col] = [tick_labels, ticks]
107        else:
108            vals = vals.astype('category')
109            cats = vals.cat.categories
110            c_vals = vals.cat.codes
111            minval = 0
112            maxval = len(cats) - 1
113            if maxval == 0:
114                c_vals = 0.5
115            else:
116                c_vals = np.true_divide(c_vals - minval, maxval - minval)
117            tick_labels = cats
118            ticks = np.unique(c_vals)
119            ax_info[col] = [tick_labels, ticks]
120        if spread is not None:

```

```

121         offset = np.arange(-1, 1, 2. / (len(c_vals))) * 2e-2
122         np.random.shuffle(offset)
123         c_vals = c_vals + offset
124         valmat[:,] = c_vals
125
126     extendfrac = curvedextend if curved else 0.05
127     for i, ax in enumerate(axes):
128         for idx in range(valmat.shape[-1]):
129             if curved:
130                 x_new = np.linspace(0, len(x), len(x) * 20)
131                 if add_random > 0:
132                     r = np.random.uniform(low=-add_random, high=add_random, size=len(valmat[:, idx]))
133                     a_BSpline = make_interp_spline(x, valmat[:, idx]+r, k=3, bc_type='clamped')
134                 else:
135                     a_BSpline = make_interp_spline(x, valmat[:, idx], k=3, bc_type='clamped')
136                 y_new = a_BSpline(x_new)
137                 ax.plot(x_new, y_new, color=colmap(valmat[-1, idx]), alpha=0.3)
138             else:
139                 ax.plot(x, valmat[:, idx], color=colmap(valmat[-1, idx]), alpha=0.3)
140             ax.set_ylim(0 - extendfrac, 1 + extendfrac)
141             ax.set_xlim(i, i + 1)
142
143     for dim, (ax, col) in enumerate(zip(axes, cols)):
144         ax.xaxis.set_major_locator(ticker.FixedLocator([dim]))
145         ax.yaxis.set_major_locator(ticker.FixedLocator(ax_info[col][1]))
146         ax.set_yticks(np.linspace(0., 1.0, len(ax_info[col][0])))
147         ax.set_yticklabels(ax_info[col][0])
148     #
149     #         try:
150     #             print(ax_info[col])
151     #             ax.set_yticklabels(ax_info[col][0])
152     #         except Exception as e:
153     #             print(e)
154     ax.set_xticklabels([cols[dim]])
155
156     plt.subplots_adjust(wspace=0)
157     norm = matplotlib.colors.Normalize(0, 1) # *axes[-1].get_ylim())
158     sm = plt.cm.ScalarMappable(cmap=colmap, norm=norm)
159     cbar = plt.colorbar(sm, pad=0, ticks=ax_info[rank_attr][1], extend='both', extendrect=True, extendfrac=extendfrac)
160     if curved:
161         cbar.ax.set_ylim(0 - curvedextend, 1 + curvedextend)
162         cbar.ax.set_yticklabels(ax_info[rank_attr][0])
163         cbar.ax.set_xlabel(rank_attr)
164     if save:
165         fig.savefig(save, bbox_inches='tight', dpi=fig.dpi)
166     if show:
167         plt.show()
168
169     return x, valmat
170
171 # In[ ]:
172
173
174 df = pd.read_csv("/home/julian/Desktop/attributesmodelsnosort (curated).csv")
175 #df = df.set_index("Model Name")
176 import re
177
178 convert = lambda text: int(text) if text.isdigit() else text.lower()
179 alphanum_key = lambda key: [[convert(c) for c in re.split('([0-9]+)', k)] for k in key]
180 df = df.sort_values("Model Name", key=alphanum_key)
181 df['random'] = (np.random.randn(len(df))+10)*10
182
183
184 # In[ ]:
185
186
187 df
188
189
190 # In[13]:
191
192
193 parallel_plot(df.set_index("Model Name"), 'random', put_last_columns=['Final Layer', 'random'], exclude_color_column=False)
194
195
196 # In[6]:
197
198
199 all_df = pd.read_csv('/home/julian/Desktop/All_attributes_with_scores.csv', index_col=0, parse_dates=['train timestamp'])
200 all_df = all_df.dropna(subset=['train timestamp', 'Uses Classweights'])
201 all_df = all_df[['train timestamp', 'Model Path'] + [c for c in all_df.columns if not c in ['train timestamp', 'Model Path', 'micro', 'macro']] + ['macro', 'micro']]
202 all_df[["Train Normalization Function", "Test Normalization Function"]] = all_df[["Train Normalization Function", "Test Normalization Function"]].apply(lambda s:s.str.replace(":", ""))
203 all_df[['model']] = all_df[['model']].apply(lambda s:s.str.replace(":d*", "", regex=True))
204 all_df = all_df.drop(columns=['train timestamp'])
205 cpc_df = all_df[all_df['normalizes latents'].isin([False, True])].dropna(axis=1, how='all')
206 #print(cpc_df)

```

```

207 bl_df = all_df[all_df['uses Max Pool'].isin([False, True])].dropna(axis=1, how='all')
208
209
210 # In[7]:
211
212
213 cols = [c for c in cpc_df.columns if not c in ['Model Path', 'timesteps in', 'timesteps out', "Train Normalization
214     Function", "Test Normalization Function", 'Train splitsfile', 'Pretrain Epochs', 'Context Size', 'Latent Size',
215     'macro', 'micro', 'Crop Size']]
216 parallel_plot(cpc_df, 'macro', save_to='/home/julian/Downloads/Multi-Download/lowlable/parcoords-cpc.png',
217     exclude_columns=['Model Path', 'timesteps in', 'timesteps out', "Train Normalization Function", "Test
218     Normalization Function", 'Train splitsfile', 'Pretrain Epochs', 'micro'], )
219 parallel_plot(cpc_df, cols, rank_attr='macro', cmap='magma', spread=None, curved=True, curvedextend=0.1, save='/home/
220     julian/Downloads/Multi-Download/lowlable-paarcards-cpc-macro.png')
221
222 # In[8]:
223
224
225 cols = [c for c in cpc_df.columns if not c in ['Model Path', 'timesteps in', 'timesteps out', "Train Normalization
226     Function", "Test Normalization Function", 'Train splitsfile', 'Pretrain Epochs', 'Context Size', 'Latent Size',
227     'macro', 'micro', 'Crop Size']]
228 parallel_plot(cpc_df, 'macro', save_to='/home/julian/Downloads/Multi-Download/lowlable/parcoords-cpc.png',
229     exclude_columns=['Model Path', 'timesteps in', 'timesteps out', "Train Normalization Function", "Test
230     Normalization Function", 'Train splitsfile', 'Pretrain Epochs', 'micro'], )
231 parallel_plot(cpc_df, cols, rank_attr='micro', cmap='magma', spread=None, curved=True, curvedextend=0.1, save='/home/
232     julian/Downloads/Multi-Download/lowlable-paarcards-cpc-micro.png')
233
234
235 # # BL
236
237
238 # In[12]:
239
240
241 cols = [c for c in bl_df.columns if not c in ['Model Path', "Train Normalization Function", "Test Normalization
242     Function", 'Train splitsfile', 'Pretrain Epochs', 'macro', 'micro', 'Crop Size', 'Downstream Epochs']]
243 parallel_plot(cpc_df, 'macro', save_to='/home/julian/Downloads/Multi-Download/lowlable/parcoords-cpc.png',
244     exclude_columns=['Model Path', 'timesteps in', 'timesteps out', "Train Normalization Function", "Test
245     Normalization Function", 'Train splitsfile', 'Pretrain Epochs', 'micro'], )
246 parallel_plot(bl_df, cols, rank_attr='macro', cmap='magma', spread=None, curved=True, curvedextend=0.1, save='/home/
247     julian/Downloads/Multi-Download/lowlable-paarcards-bl-macro.png')
248
249
250 # In[16]:
251
252
253 cols = [c for c in bl_df.columns if not c in ['Model Path', "Train Normalization Function", "Test Normalization
254     Function", 'Train splitsfile', 'Pretrain Epochs', 'macro', 'micro', 'Crop Size']]
255 parallel_plot(cpc_df, 'macro', save_to='/home/julian/Downloads/Multi-Download/lowlable/parcoords-cpc.png',
256     exclude_columns=['Model Path', 'timesteps in', 'timesteps out', "Train Normalization Function", "Test
257     Normalization Function", 'Train splitsfile', 'Pretrain Epochs', 'micro'], )
258 parallel_plot(bl_df, cols, rank_attr='micro', cmap='magma', spread=None, curved=True, add_random=0.0, curvedextend
259     =0.1, save='/home/julian/Downloads/Multi-Download/lowlable-paarcards-bl-micro.png')
260
261
262 # In[ ]:
263
264
265 df = pd.read_csv("/home/julian/Desktop/ALLMODELSATTRIBUTESTrue.csv")
266 #df = df.set_index("Model Name")
267 import re
268
269 convert = lambda text: int(text) if text.isdigit() else text.lower()
270 alphanum_key = lambda key: [(convert(c) for c in re.split('([0-9]+)', k)) for k in key]
271 df = df.sort_values("model", key=alphanum_key)
272
273
274 # In[25]:
275
276
277 parallel_plot(df.set_index("model"), 'macro', put_last_columns=['Final Layer', 'micro', 'macro'], exclude_columns=['
278     use_class_weights'], exclude_color_column=False)
279
280
281 # In[27]:
282
283
284 parallel_plot(df.set_index("model"), 'micro', put_last_columns=['Final Layer', 'macro', 'micro'], exclude_columns=['
285     use_class_weights'], exclude_color_column=False)
286
287
288 # In[ ]:
```

B.7.13 jupyter_notebooks -> Physionet Hz Converter.py (code)

```
1#!/usr/bin/env python
```

```

2 # coding: utf-8
3
4 # In[1]:
5
6
7 import wfdb
8 import numpy as np
9 import os
10 import pandas as pd
11 from random import randint
12 import math
13 import h5py
14 import matplotlib.pyplot as plt
15 from scipy.signal import find_peaks
16 from collections import Counter
17 from wfdb.processing import resample_multichan
18 import glob
19 import sys
20 import traceback
21 from functools import reduce
22
23
24 # In[6]:
25
26
27 def get_file_list(BASE_DIR):
28     record_files = []
29     file_endings = ['.dat', '.hea', '.xyz']
30     with open(os.path.join(BASE_DIR, 'RECORDS')) as recs:
31         record_files = recs.readlines()
32     recs.close()
33     return record_files
34
35 def get_file_path_list(base_dir, file_endings=['.dat'], remove_extension=True):
36     record_file_paths = []
37     for end in file_endings:
38         end_set = []
39         for path, subdirs, files in os.walk(base_dir):
40             for name in files:
41                 if name.endswith(end):
42                     if remove_extension:
43                         end_set.append(os.path.splitext(os.path.join(path, name))[0])
44                     else:
45                         end_set.append(os.path.join(path, name))
46             record_file_paths.append(set(end_set))
47     if remove_extension:
48         return list(reduce(lambda a, b: a & b, record_file_paths))
49     else:
50         return list(reduce(lambda a, b: a | b, record_file_paths))
51
52 def read_record_infos(record_path):
53     record = wfdb.rdrecord(record_path)
54     return {
55         'record_name' : record.record_name,
56         'file_name' : record.file_name,
57         'record_comments' : record.comments,
58         'number_of_signals' : record.n_sig,
59         'is_physical_signal' : not (record.p_signal is None),
60         'is_digital_signal' : not (record.d_signal is None),
61         'signal_sampling_frequency' : record.fs,
62         'signal_length' : record.sig_len,
63         'signal_channel_names' : record.sig_name,
64         'signal_channel_units' : record.units
65     }
66
67 def read_annotation(record_path, physical=True, return_label_elements=['symbol', 'label_store', 'description']):
68     try:
69         annotation = wfdb.rdnann(record_path, 'atr', return_label_elements=return_label_elements)
70         #print(record_path)
71         #print('sample:', annotation.sample, 'symbol', annotation.symbol, 'contained labels', annotation.label_store, annotation.description)
72         return (annotation.sample, annotation.symbol, annotation.label_store, annotation.description)
73     except ValueError as ve:
74         print(record_path, 'annotation read failed:', ve)
75     return None
76
77 def read_annotation_object(record_path, physical=True, return_label_elements=['symbol', 'label_store', 'description']):
78     try:
79         annotation = wfdb.rdnann(record_path, 'atr', return_label_elements=return_label_elements)
80         return annotation
81     except ValueError as ve:
82         print(record_path, 'annotation read failed:', ve)
83     return None
84
85 def read_signal(record_path, physical=True):
86     #print(record_path)
87     record = wfdb.rdrecord(record_path, physical=physical)
88     #print_object_attributes(record)
89     if physical:
90         data = record.p_signal

```

```

91     else:
92         data = record.d_signal
93     return data
94
95 def resample_frequency(record_signal_array, annotation_object, hz_frq_in:int, hz_frq_out:int):
96     return resample_multichan(record_signal_array, annotation_object, hz_frq_in, hz_frq_out, resamp_ann_chan=0)
97
98 def resample_frequency_all(record_file_paths, hz_frq_out, output_filepath, physical=True, resample_annotation=True,
99                           overwrite=False):
100    print('Resampling of files has started:', record_files)
101    if not os.path.exists(output_filepath):
102        os.makedirs(output_filepath)
103    for record_path in record_file_paths:
104        file = os.path.basename(record_path)
105        print("Resampling:", file)
106        if overwrite or not glob(os.path.join(output_filepath, file) +'.*'):
107            file_infos = read_record_infos(record_path)
108            print("read record infos")
109            hz_frq_in = file_infos['signal_sampling_frequency']
110            record_signal = read_signal(record_path, physical=physical)
111            print("read record signal", record_signal.shape)
112
113        try:
114            if resample_annotation:
115                annotation_object = read_annotation_object(record_path, physical=physical, return_label_elements=[
116                    'symbol'])#, 'label_store']) cannot use, wfdb is buggy
117                print("read record annotation")
118                resampled_signal, resampled_ann = resample_frequency(record_signal, annotation_object, hz_frq_in,
119                hz_frq_out) #Throws random assertionerrors at times
120                print("computed resampled signal", resampled_signal.shape)
121                wfdb.wrann(file+"resampled", 'atr', sample=resampled_ann.sample, label_store=annotation_object.
122                label_store, write_dir=output_filepath)
123                a_temp = wfdb.Annotation(file, 'atr', resampled_ann.sample, symbol=annotation_object.symbol, fs=
124                hz_frq_out, label_store=annotation_object.label_store)
125                a_temp.wrann(write_fs=True, write_dir=output_filepath)
126            else:
127                resampled_signal, _ = resample_sig(record_signal, hz_frq_in, hz_frq_out)
128            try:
129                if physical:
130                    wfdb.wrsamp(file, fs=hz_frq_out, p_signal=resampled_signal, sig_name=file_infos['
131                    signal_channel_names'], units=file_infos['signal_channel_units'], comments=file_infos['record_comments'],
132                    write_dir=output_filepath)
133                else:
134                    wfdb.wrsamp(file, fs=hz_frq_out, d_signal=resampled_signal, sig_name=file_infos['
135                    signal_channel_names'], units=file_infos['signal_channel_units'], comments=file_infos['record_comments'],
136                    write_dir=output_filepath)
137                    print('finished', output_filepath)
138            except (IndexError, ValueError) as ive: #wfdb = bug
139                print(ive)
140            except AssertionError: #https://stackoverflow.com/questions/11587223/how-to-handle-assertionerror-in-
141                python-and-find-out-which-line-or-statement-it-
142                ___, tb = sys.exc_info()
143                traceback.print_tb(tb) # Fixed format
144                tb_info = traceback.extract_tb(tb)
145                filename, line, func, text = tb_info[-1]
146
147                print('An error occurred on line {} in statement {}'.format(line, text))
148
149        else:
150            print("skipping", file)
151
152 # ### Download the database
153
154 # In[3]:
155
156 db_dir = 'ptb-xl' #The Database to download (see get_dbs())
157 temp_directory = 'out'
158 save_to_drive = False
159 download_annotation = False #If the annotation/labels should be downloaded
160 hz_frq_out = 1000
161 wfdb.dl_database(db_dir, temp_directory, records='all', annotators='all' if download_annotation else None)
162
163 # ### Resample all downloaded files
164
165 # In[ ]:
166
167 record_files = get_file_path_list('/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography
168 -dataset-1.0.1/records500')
169 resample_frequency_all(record_files, hz_frq_out=hz_frq_out, output_filepath=os.path.join('/media/julian/Volume/data/
170 ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/generated', str(hz_frq_out)),
171     resample_annotation=download_annotation)
172
173 # In[ ]:
174
175

```

```

168 def generate_RECORDS_file(record_file_root_dir):
169     record_files = get_file_path_list(record_file_root_dir, file_endings=['.dat', '.hea'])
170     output_path = record_file_root_dir
171     with open(os.path.join(output_path, "RECORDS"), 'w') as f:
172         f.write('\n'.join([os.path.basename(p) for p in record_files]))
173 generate_RECORDS_file('/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset
174 -1.0.1/generated/1000')
175
176 # In[5]:
177
178
179 import numpy as np
180 from scipy import signal
181 import pandas as pd
182 def resample_sig(x, fs, fs_target): #https://github.com/MIT-LCP/wfdb-python/blob/master/wfdbprocessing/basic.py
183 """
184     Resample a signal to a different frequency.
185     Parameters
186     -----
187     x : ndarray
188         Array containing the signal.
189     fs : int, float
190         The original sampling frequency.
191     fs_target : int, float
192         The target frequency.
193     Returns
194     -----
195     resampled_x : ndarray
196         Array of the resampled signal values.
197     resampled_t : ndarray
198         Array of the resampled signal locations.
199     """
200     t = np.arange(x.shape[0]).astype('float64')
201
202     if fs == fs_target:
203         return x, t
204
205     new_length = int(x.shape[0]*fs_target/fs)
206     # Resample the array if NaN values are present
207     if np.isnan(x).any():
208         x = pd.Series(x.reshape((-1,))).interpolate().values
209     resampled_x, resampled_t = signal.resample(x, num=new_length, t=t)
210     assert np.all(np.diff(resampled_t) > 0)
211     #deleted assert to make it work again
212     return resampled_x, resampled_t
213
214
215 # In[24]:
216
217
218 p = os.path.join('/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
219     generated', str(hz_frg_out))
220 dat_files = set([os.path.splitext(d)[0] for d in glob.glob(os.path.join(p, '*.dat'))])
221 hea_files = set([os.path.splitext(d)[0] for d in glob.glob(os.path.join(p, '*.hea'))])
222
223 # In[22]:
224
225
226 for f in (dat_files - hea_files) | (hea_files - dat_files):
227     for ff in glob.glob(f+'*.hea') + glob.glob(f+'*.dat'):
228         print('removing:', ff)
229         os.remove(ff)
230
231
232 # In[ ]:

```

B.7.14 jupyter_notebooks -> Precision_Recall.py (code)

```

1#!/usr/bin/env python
2# coding: utf-8
3
4# In[1]:
5
6
7import pandas as pd
8import numpy as np
9import os
10import glob
11
12
13# In[2]:
14
15

```

```

16 label_index_mapping = {'10370003': 0, '111288001': 1, '11157007': 2, '111975006': 3, '164861001': 4, '164865005': 5, '164867002': 6, '164873001': 7, '164884008': 8, '164889003': 9, '164890007': 10, '164895002': 11, '164896001': 12, '164909002': 13, '164917005': 14, '164921003': 15, '164930006': 16, '164931005': 17, '164934002': 18, '164937009': 19, '164947007': 20, '164951009': 21, '17338001': 22, '195042002': 23, '195060002': 24, '195080001': 25, '195101003': 26, '195126007': 27, '233917008': 28, '251120003': 29, '251139008': 30, '251146004': 31, '251164006': 32, '251170000': 33, '251180001': 34, '251200008': 35, '251259000': 36, '251266004': 37, '251268003': 38, '253339007': 39, '253352002': 40, '266249003': 41, '270492004': 42, '27885002': 43, '284470004': 44, '29320008': 45, '370365005': 46, '39732003': 47, '413444003': 48, '413844008': 49, '425419005': 50, '425623009': 51, '426177001': 52, '426434006': 53, '426627000': 54, '426648003': 55, '426664006': 56, '426749004': 57, '426761007': 58, '426783006': 59, '426995002': 60, '427084000': 61, '427172004': 62, '427393009': 63, '428417006': 64, '428750005': 65, '429622005': 66, '445118002': 67, '445211001': 68, '446358003': 69, '446813000': 70, '47665007': 71, '49578007': 72, '54329005': 73, '55930002': 74, '59118001': 75, '59931005': 76, '63593006': 77, '6374002': 78, '65778007': 79, '67198005': 80, '67741000119109': 81, '698252002': 82, '704997005': 83, '713422000': 84, '713426002': 85, '713427006': 86, '7439002': 87, '74615001': 88, '75532003': 89, '77867006': 90, '81898007': 91, '82226007': 92, '89792004': 93}
17
18
19 # In[14]:
20
21
22 #model_folder = 'models/02_03_21-15/architectures_cpc.cpc_combined.CPCCombined'
23 #model_folder = 'models/09_03_21-18/architectures_cpc.cpc_combined.CPCCombined'
24 #model_folder = 'models/10_03_21-18/architectures_cpc.cpc_combined.CPCCombined0'
25 model_folder = 'models/10_03_21-18/architectures_cpc.cpc_combined.CPCCombined1'
26
27
28 # In[15]:
29
30
31 pred_path = glob.glob(os.path.join(model_folder ,'*output.csv'))[0]
32 dfp = pd.read_csv(pred_path)
33 pred = dfp.values[:, 1:]
34 dfp
35
36
37 # In[16]:
38
39
40 label_path = glob.glob(os.path.join(model_folder ,'*labels*.csv'))[0]
41 dfl = pd.read_csv(label_path)
42 labels = dfl.values[:, 1:].astype(int)
43 dfl
44
45
46 # In[17]:
47
48
49 np.set_printoptions(suppress=True)
50 np.round(np.sum(pred, axis=0).astype(float), 0)
51
52
53 # In[18]:
54
55
56 np.sum(labels, axis=0)
57
58
59 # In[19]:
60
61
62 n_classes = labels.shape[1]
63 n_classes
64
65
66 # ## Calculate ROC
67
68 # In[28]:
69
70
71 from sklearn.metrics import precision_recall_curve
72 from sklearn.metrics import average_precision_score
73
74 # For each class
75 precision = dict()
76 recall = dict()
77 average_precision = dict()
78 for i in range(n_classes):
79     precision[i], recall[i], _ = precision_recall_curve(labels[:, i],
80                                                       pred[:, i])
81     average_precision[i] = average_precision_score(labels[:, i], pred[:, i])
82
83 # A "micro-average": quantifying score on all classes jointly
84 precision["micro"], recall["micro"], _ = precision_recall_curve(labels.ravel(),
85                                                               pred.ravel())
86 average_precision["micro"] = average_precision_score(labels, pred,
87                                                       average="micro")
88 print('Average precision score, micro-averaged over all classes: {0:0.2f})'
89       .format(average_precision["micro"]))
90
91

```

```

92 # In[29]:
93
94
95 import matplotlib.pyplot as plt
96 def plot_precision_recall_microavg(recall, precision):
97     plt.figure()
98     plt.step(recall['micro'], precision['micro'], where='post')
99
100    plt.xlabel('Recall')
101    plt.ylabel('Precision')
102    plt.ylim([0.0, 1.05])
103    plt.xlim([0.0, 1.0])
104    plt.title(
105        'Average precision score, micro-averaged over all classes: AP={0:0.2f}\n'
106        .format(average_precision["micro"]))
107
108
109 # In[30]:
110
111
112 plot_precision_recall_microavg(recall, precision)
113
114
115 # In[31]:
116
117
118 from itertools import cycle
119 def plot_precision_recall_multiclass(recall, precision, selection=None):
120     cm = plt.get_cmap('gist_rainbow')
121     if selection is None:
122         selection = range(n_classes)
123     # setup plot details
124     colors = cycle([cm(1.*i/n_classes) for i in selection])
125
126     plt.figure(figsize=(7, 8))
127     f_scores = np.linspace(0.2, 0.8, num=4)
128     lines = []
129     labels = []
130
131     for f_score in f_scores:
132         x = np.linspace(0.01, 1)
133         y = f_score * x / (2 * x - f_score)
134         l, = plt.plot(x[y >= 0], y[y >= 0], color='gray', alpha=0.2)
135         plt.annotate('f1={0:0.1f}'.format(f_score), xy=(0.9, y[45] + 0.02))
136
137     lines.append(l)
138     labels.append('iso-f1 curves')
139     l, = plt.plot(recall["micro"], precision["micro"], color='gold', lw=2)
140     lines.append(l)
141     labels.append('micro-average Precision-recall (area = {0:0.2f})\n'
142                   ''.format(average_precision["micro"]))
143
144     for i, color in zip(selection, colors):
145         l, = plt.plot(recall[i], precision[i], color=color, lw=2)
146         lines.append(l)
147         labels.append('Precision-recall for class {0} (area = {1:0.2f})\n'
148                       ''.format(i, average_precision[i]))
149
150     fig = plt.gcf()
151     fig.subplots_adjust(bottom=0.25)
152     plt.xlim([0.0, 1.0])
153     plt.ylim([0.0, 1.05])
154     plt.xlabel('Recall')
155     plt.ylabel('Precision')
156     plt.title('Extension of Precision-Recall curve to multi-class')
157     plt.legend(lines, labels, loc="lower right", bbox_to_anchor=(1.1, 1.05))
158
159     plt.show()
160
161
162 # In[32]:
163
164
165 physionetchallenge_sel = '
166     270492004,164889003,164890007,426627000,713427006,713426002,445118002,39732003,164909002,251146004,698252002,10370003,284470004,427172004,164
166     .split(',')
166 plot_precision_recall_multiclass(recall, precision, selection=[label_index_mapping[snomed] for snomed in
167     physionetchallenge_sel])
168
169 # In[33]:
170
171
172 precision.keys()
173
174
175 # Below taken (and changed) from: https://github.com/physionetchallenges/evaluation-2020/blob/master/
175     evaluate_12ECG_score.py
176
177 # In[34]:

```

```

178
179
180 #!/usr/bin/env python
181
182 # This file contains functions for evaluating algorithms for the 2020 PhysioNet/
183 # Computing in Cardiology Challenge. You can run it as follows:
184 #
185 #     python evaluate_12ECG_score.py labels outputs scores.csv
186 #
187 # where 'labels' is a directory containing files with the labels, 'outputs' is a
188 # directory containing files with the outputs from your model, and 'scores.csv'
189 # (optional) is a collection of scores for the algorithm outputs.
190 #
191 # Each file of labels or outputs must have the format described on the Challenge
192 # webpage. The scores for the algorithm outputs include the area under the
193 # receiver-operating characteristic curve (AUROC), the area under the recall-
194 # precision curve (AUPRC), accuracy (fraction of correct recordings), macro F-
195 # measure, and the Challenge metric, which assigns different weights to
196 # different misclassification errors.
197
198 import numpy as np, os, os.path, sys
199
200 def evaluate_12ECG_score(label_scores_data, label_binary_data, output_data):
201     # Define the weights, the SNOMED CT code for the normal class, and equivalent SNOMED CT codes.
202     weights_file = 'weights.csv'
203     normal_class = '426783006'
204     equivalent_classes = [['713427006', '59118001'], ['284470004', '63593006'], ['427172004', '17338001']]
205
206     # Load the scored classes and the weights for the Challenge metric.
207     print('Loading weights...')
208     classes, weights = load_weights(weights_file, equivalent_classes)
209
210     # Load the label and output files.
211     print('Loading label and output files...')
212
213     binary_outputs, scalar_outputs = label_binary_data, label_scores_data
214
215     # Evaluate the model by comparing the labels and outputs.
216     print('Evaluating model...')
217
218     print(' - AUROC and AUPRC... ')
219     auroc, auprc, auroc_classes, auprc_classes = compute_auc(labels, scalar_outputs)
220
221     print(' - Accuracy... ')
222     accuracy = compute_accuracy(labels, binary_outputs)
223
224     print(' - F-measure... ')
225     f_measure, f_measure_classes = compute_f_measure(labels, binary_outputs)
226
227     print(' - F-beta and G-beta measures... ')
228     f_beta_measure, g_beta_measure = compute_beta_measures(labels, binary_outputs, beta=2)
229
230     print(' - Challenge metric...(skipped)')
231     challenge_metric = -1.0 #compute_challenge_metric(weights, labels, binary_outputs, classes, normal_class)
232
233     print('Done.')
234
235     # Return the results.
236     return classes, auroc, auprc, auroc_classes, auprc_classes, accuracy, f_measure, f_measure_classes, f_beta_measure,
237         , g_beta_measure, challenge_metric
238
239 # Check if the input is a number.
240 def is_number(x):
241     try:
242         float(x)
243         return True
244     except ValueError:
245         return False
246
247 # Load weights.
248 def load_weights(weight_file, equivalent_classes):
249     # Load the weight matrix.
250     rows, cols, values = load_table(weight_file)
251     assert(rows == cols)
252
253     # For each collection of equivalent classes, replace each class with the representative class for the set.
254     rows = replace_equivalent_classes(rows, equivalent_classes)
255
256     # Check that equivalent classes have identical weights.
257     for j, x in enumerate(rows):
258         for k, y in enumerate(rows[j+1:]):
259             if x==y:
260                 assert(np.all(values[j, :]==values[j+k, :]))
261                 assert(np.all(values[:, j]==values[:, j+k]))
262
263     # Use representative classes.
264     classes = [x for j, x in enumerate(rows) if x not in rows[:j]]
265     indices = [rows.index(x) for x in classes]
266     weights = values[np.ix_(indices, indices)]

```

```

267     return classes, weights
268
269 # Compute recording-wise accuracy.
270 def compute_accuracy(labels, outputs):
271     num_recordings, num_classes = np.shape(labels)
272
273     num_correct_recordings = 0
274     for i in range(num_recordings):
275         if np.all(labels[i, :] == outputs[i, :]):
276             num_correct_recordings += 1
277
278     return float(num_correct_recordings) / float(num_recordings)
279
280 # Compute confusion matrices.
281 def compute_confusion_matrices(labels, outputs, normalize=False):
282     # Compute a binary confusion matrix for each class k:
283     #
284     # [TN_k FN_k]
285     # [FP_k TP_k]
286     #
287     # If the normalize variable is set to true, then normalize the contributions
288     # to the confusion matrix by the number of labels per recording.
289     num_recordings, num_classes = np.shape(labels)
290
291     if not normalize:
292         A = np.zeros((num_classes, 2, 2))
293         for i in range(num_recordings):
294             for j in range(num_classes):
295                 if labels[i, j] == 1 and outputs[i, j] == 1: # TP
296                     A[j, 1, 1] += 1
297                 elif labels[i, j] == 0 and outputs[i, j] == 1: # FP
298                     A[j, 1, 0] += 1
299                 elif labels[i, j] == 1 and outputs[i, j] == 0: # FN
300                     A[j, 0, 1] += 1
301                 elif labels[i, j] == 0 and outputs[i, j] == 0: # TN
302                     A[j, 0, 0] += 1
303                 else: # This condition should not happen.
304                     raise ValueError('Error in computing the confusion matrix.')
305             else:
306                 A = np.zeros((num_classes, 2, 2))
307                 for i in range(num_recordings):
308                     normalization = float(max(np.sum(labels[i, :]), 1))
309                     for j in range(num_classes):
310                         if labels[i, j] == 1 and outputs[i, j] == 1: # TP
311                             A[j, 1, 1] += 1.0/normalization
312                         elif labels[i, j] == 0 and outputs[i, j] == 1: # FP
313                             A[j, 1, 0] += 1.0/normalization
314                         elif labels[i, j] == 1 and outputs[i, j] == 0: # FN
315                             A[j, 0, 1] += 1.0/normalization
316                         elif labels[i, j] == 0 and outputs[i, j] == 0: # TN
317                             A[j, 0, 0] += 1.0/normalization
318                         else: # This condition should not happen.
319                             raise ValueError('Error in computing the confusion matrix.')
320
321         return A
322
323 # Compute macro F-measure.
324 def compute_f_measure(labels, outputs):
325     num_recordings, num_classes = np.shape(labels)
326
327     A = compute_confusion_matrices(labels, outputs)
328
329     f_measure = np.zeros(num_classes)
330     for k in range(num_classes):
331         tp, fp, fn, tn = A[k, 1, 1], A[k, 1, 0], A[k, 0, 1], A[k, 0, 0]
332         if 2 * tp + fp + fn:
333             f_measure[k] = float(2 * tp) / float(2 * tp + fp + fn)
334         else:
335             f_measure[k] = float('nan')
336
337     macro_f_measure = np.nanmean(f_measure)
338
339     return macro_f_measure, f_measure
340
341 # Compute F-beta and G-beta measures from the unofficial phase of the Challenge.
342 def compute_beta_measures(labels, outputs, beta):
343     num_recordings, num_classes = np.shape(labels)
344
345     A = compute_confusion_matrices(labels, outputs, normalize=True)
346
347     f_beta_measure = np.zeros(num_classes)
348     g_beta_measure = np.zeros(num_classes)
349     for k in range(num_classes):
350         tp, fp, fn, tn = A[k, 1, 1], A[k, 1, 0], A[k, 0, 1], A[k, 0, 0]
351         if (1+beta**2)*tp + fp + beta**2*fn:
352             f_beta_measure[k] = float((1+beta**2)*tp) / float((1+beta**2)*tp + fp + beta**2*fn)
353         else:
354             f_beta_measure[k] = float('nan')
355         if tp + fp + beta*fn:
356

```

```

357         g_beta_measure[k] = float(tp) / float(tp + fp + beta*fn)
358     else:
359         g_beta_measure[k] = float('nan')
360
361     macro_f_beta_measure = np.nanmean(f_beta_measure)
362     macro_g_beta_measure = np.nanmean(g_beta_measure)
363
364     return macro_f_beta_measure, macro_g_beta_measure
365
366 # Compute macro AUROC and macro AUPRC.
367 def compute_auc(labels, outputs):
368     num_recordings, num_classes = np.shape(labels)
369
370     # Compute and summarize the confusion matrices for each class across at distinct output values.
371     auroc = np.zeros(num_classes)
372     auprc = np.zeros(num_classes)
373
374     for k in range(num_classes):
375         # We only need to compute TPs, FPs, FNs, and TNs at distinct output values.
376         thresholds = np.unique(outputs[:, k])
377         thresholds = np.append(thresholds, thresholds[-1]+1)
378         thresholds = thresholds[::-1]
379         num_thresholds = len(thresholds)
380
381         # Initialize the TPs, FPs, FNs, and TNs.
382         tp = np.zeros(num_thresholds)
383         fp = np.zeros(num_thresholds)
384         fn = np.zeros(num_thresholds)
385         tn = np.zeros(num_thresholds)
386         fn[0] = np.sum(labels[:, k]==1)
387         tn[0] = np.sum(labels[:, k]==0)
388
389         # Find the indices that result in sorted output values.
390         idx = np.argsort(outputs[:, k])[::-1]
391
392         # Compute the TPs, FPs, FNs, and TNs for class k across thresholds.
393         i = 0
394         for j in range(1, num_thresholds):
395             # Initialize TPs, FPs, FNs, and TNs using values at previous threshold.
396             tp[j] = tp[j-1]
397             fp[j] = fp[j-1]
398             fn[j] = fn[j-1]
399             tn[j] = tn[j-1]
400
401             # Update the TPs, FPs, FNs, and TNs at i-th output value.
402             while i < num_recordings and outputs[idx[i], k] >= thresholds[j]:
403                 if labels[idx[i], k]:
404                     tp[j] += 1
405                     fn[j] -= 1
406                 else:
407                     fp[j] += 1
408                     tn[j] -= 1
409                 i += 1
410
411         # Summarize the TPs, FPs, FNs, and TNs for class k.
412         tpr = np.zeros(num_thresholds)
413         tnr = np.zeros(num_thresholds)
414         ppv = np.zeros(num_thresholds)
415         for j in range(num_thresholds):
416             if tp[j] + fn[j]:
417                 tpr[j] = float(tp[j]) / float(tp[j] + fn[j])
418             else:
419                 tpr[j] = float('nan')
420             if fp[j] + tn[j]:
421                 tnr[j] = float(tn[j]) / float(fp[j] + tn[j])
422             else:
423                 tnr[j] = float('nan')
424             if tp[j] + fp[j]:
425                 ppv[j] = float(tp[j]) / float(tp[j] + fp[j])
426             else:
427                 ppv[j] = float('nan')
428
429         # Compute AUROC as the area under a piecewise linear function with TPR/
430         # sensitivity (x-axis) and TNR/specificity (y-axis) and AUPRC as the area
431         # under a piecewise constant with TPR/recall (x-axis) and PPV/precision
432         # (y-axis) for class k.
433         for j in range(num_thresholds-1):
434             auroc[k] += 0.5 * (tpr[j+1] - tpr[j]) * (tnr[j+1] + tnr[j])
435             auprc[k] += (tpr[j+1] - tpr[j]) * ppv[j+1]
436
437         # Compute macro AUROC and macro AUPRC across classes.
438         macro_auroc = np.nanmean(auroc)
439         macro_auprc = np.nanmean(auprc)
440
441     return macro_auroc, macro_auprc, auroc, auprc
442
443 # Compute modified confusion matrix for multi-class, multi-label tasks.
444 def compute_modified_confusion_matrix(labels, outputs):
445     # Compute a binary multi-class, multi-label confusion matrix, where the rows
446     # are the labels and the columns are the outputs.

```

```

447 num_recordings, num_classes = np.shape(labels)
448 A = np.zeros((num_classes, num_classes))
449
450 # Iterate over all of the recordings.
451 for i in range(num_recordings):
452     # Calculate the number of positive labels and/or outputs.
453     normalization = float(max(np.sum(np.any((labels[i, :], outputs[i, :]), axis=0)), 1))
454     # Iterate over all of the classes.
455     for j in range(num_classes):
456         # Assign full and/or partial credit for each positive class.
457         if labels[i, j]:
458             for k in range(num_classes):
459                 if outputs[i, k]:
460                     A[j, k] += 1.0/normalization
461
462 return A
463
464 # Compute the evaluation metric for the Challenge.
465 def compute_challenge_metric(weights, labels, outputs, classes, normal_class):
466     num_recordings, num_classes = np.shape(labels)
467     normal_index = classes.index(normal_class)
468
469     # Compute the observed score.
470     A = compute_modified_confusion_matrix(labels, outputs)
471     observed_score = np.nansum(weights * A)
472
473     # Compute the score for the model that always chooses the correct label(s).
474     correct_outputs = labels
475     A = compute_modified_confusion_matrix(labels, correct_outputs)
476     correct_score = np.nansum(weights * A)
477
478     # Compute the score for the model that always chooses the normal class.
479     inactive_outputs = np.zeros((num_recordings, num_classes), dtype=np.bool)
480     inactive_outputs[:, normal_index] = 1
481     A = compute_modified_confusion_matrix(labels, inactive_outputs)
482     inactive_score = np.nansum(weights * A)
483
484     if correct_score != inactive_score:
485         normalized_score = float(observed_score - inactive_score) / float(correct_score - inactive_score)
486     else:
487         normalized_score = 0.0
488
489 return normalized_score
490
491
492 # In[ ]:
```

B.7.15 jupyter_notebooks -> ROC.py (code)

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[14]:
5
6
7 import pandas as pd
8 import numpy as np
9 import os
10 import glob
11 import seaborn as sns
12 import matplotlib.pyplot as plt
13 import matplotlib
14
15
16 # In[40]:
17
18
19 #model_folder = 'models/02_03_21-15/architectures_cpc.cpc_combined.CPCCombined'
20 #model_folder = 'models/02_03_21-15/architectures_cpc.cpc_combined.CPCCombined'
21 #model_folder = 'models/09_03_21-18/architectures_cpc.cpc_combined.CPCCombined'
22 #model_folder = 'models/10_03_21-18/architectures_cpc.cpc_combined.CPCCombined1'
23 #model_folder = '/home/julian/Downloads/Github/contrastive-predictive-coding/models/16_03_21-17/architectures_cpc.
24 cpc_combined.CPCCombined0'
24 #model_folder = '/home/julian/Downloads/Github/contrastive-predictive-coding/models/16_08_21-10-16-test|(40x)cpc/14
25 _08_21-15_36-train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined2|train-test-splits-fewer-labels60|
25 use_weights|unfrozen|C|m:all|cpc_downstream_cnn'
25 model_folder = '/home/julian/Downloads/Github/contrastive-predictive-coding/models/16_08_21-10-16-test|(40x)cpc/14
25 _08_21-15_36-train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined1|train-test-splits-fewer-labels60|
25 use_weights|strided|frozen|C|m:all|cpc_downstream_cnn'
26
27
28 # In[41]:
29
30
31 pred_path = glob.glob(os.path.join(model_folder ,'*0-output.csv'))[0]
32 dfp = pd.read_csv(pred_path)
```

```

33 pred = dfp.values[:, 1:]
34 dfp
35
36
37 # In[50]:
38
39
40 label_path = glob.glob(os.path.join(model_folder , 'labels-dataloader-0.csv'))[0]
41 print(label_path)
42 df1 = pd.read_csv(label_path)
43 LABELS = df1.values[:, 1:].astype(int)
44 df1
45
46
47 # In[51]:
48
49
50 np.set_printoptions(suppress=True)
51 pred_sum = np.round(np.sum(pred, axis=0).astype(float), 0).astype(int)
52 pred_sum
53
54
55 # In[52]:
56
57
58 label_sum = np.sum(LABELS, axis=0)
59 label_sum
60
61
62 # In[53]:
63
64
65 fig = plt.figure(dpi=600)
66 width = 0.6
67 plt.bar(np.arange(len(label_sum)), label_sum, width=width, label='Sum of labels')
68 plt.bar(np.arange(len(label_sum))+(1-width), pred_sum, width=width, label='Sum of label prediction probabilties')
69 plt.ylabel('Log count')
70 plt.xlabel('Class')
71 plt.yscale('log')
72 plt.legend(prop={'size': 6})
73 plt.show()
74
75
76 # ## Calculate ROC
77
78 # In[54]:
79
80
81 import numpy as np
82 import matplotlib.pyplot as plt
83 from itertools import cycle
84
85 from sklearn import svm, datasets
86 from sklearn.metrics import roc_curve, auc
87 from sklearn.model_selection import train_test_split
88 from sklearn.preprocessing import label_binarize
89 from sklearn.multiclass import OneVsRestClassifier
90 from scipy import interp
91 from sklearn.metrics import roc_auc_score
92
93
94 # In[55]:
95
96
97 def plot_roc_singleclass(tpr, fpr, roc_auc, n, fill_area=False):
98     plt.figure(dpi=400, figsize=(6,4))
99     lw = 2
100    plt.plot(fpr[n], tpr[n], color='darkorange',
101              lw=lw, label='ROC curve')
102    if fill_area:
103        plt.fill_between(fpr[n], tpr[n], label='(Area Under Curve = AUC = %0.2f)' % roc_auc[n])
104    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
105    plt.xlim([0.0, 1.0])
106    plt.ylim([0.0, 1.05])
107    plt.xlabel('False Positive Rate')
108    plt.ylabel('True Positive Rate')
109    plt.title('Receiver operating characteristic example')
110    plt.legend(loc="lower right")
111    plt.show()
112
113
114 # In[61]:
115
116
117 def plot_roc_multiclass(tpr, fpr, roc_auc, selection=None, plot_averages=True, plot_legend=True):
118     selection = range(n_classes) if selection is None else selection
119     all_fpr = np.unique(np.concatenate([fpr[i] for i in selection]))
120     lw = 1
121     # Then interpolate all ROC curves at this points
122     mean_tpr = np.zeros_like(all_fpr)

```

```

123     for i in selection:
124         mean_tpr += interp(all_fpr, fpr[i], tpr[i])
125
126     # Finally average it and compute AUC
127     mean_tpr /= len(selection)
128
129     fpr["macro"] = all_fpr
130     tpr["macro"] = mean_tpr
131     roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])
132
133     # Plot all ROC curves
134     plt.figure(dpi=400, figsize=(6,4))
135     if plot_averages:
136         plt.plot(fpr["micro"], tpr["micro"],
137                   label='micro-average ROC curve (area = {0:0.2f})'
138                   ''.format(roc_auc["micro"]),
139                   color='deeppink', linestyle=':', linewidth=4)
140
141         plt.plot(fpr["macro"], tpr["macro"],
142                   label='macro-average ROC curve (area = {0:0.2f})'
143                   ''.format(roc_auc["macro"]),
144                   color='navy', linestyle=':', linewidth=4)
145
146     colors = sns.color_palette("Paired", len(selection)) #cycle(['aqua', 'darkorange', 'cornflowerblue'])
147     for i, color in zip(selection, colors):
148         plt.plot(fpr[i], tpr[i], color=color, lw=lw,
149                   label='ROC curve of class {0} (area = {1:0.2f})'
150                   ''.format(i, roc_auc[i]))
151
152     plt.plot([0, 1], [0, 1], 'k--', lw=lw)
153     plt.xlim([0.0, 1.0])
154     plt.ylim([0.0, 1.05])
155     plt.xlabel('False Positive Rate')
156     plt.ylabel('True Positive Rate')
157     plt.title('Some extension of Receiver operating characteristic to multi-class')
158     plt.tight_layout()
159     if plot_legend:
160         #plt.legend(loc="upper left", bbox_to_anchor=(1.1, 1))
161         plt.legend(fontsize=2)
162     plt.show()
163
164
165 # In[57]:
166
167
168 n_classes = LABELS.shape[1]
169 fpr = dict()
170 tpr = dict()
171 thresholds = dict() #no dict because always the same
172 roc_auc = dict()
173 for i in range(n_classes):
174     fpr[i], tpr[i], thresholds[i] = roc_curve(LABELS[:, i], pred[:, i])
175     roc_auc[i] = auc(fpr[i], tpr[i])
176 fpr["micro"], tpr["micro"], _ = roc_curve(LABELS.ravel(), pred.ravel())
177 roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
178
179
180 # In[58]:
181
182
183 plot_roc_singleclass(tpr, fpr, roc_auc, 43, fill_area=True) #normal class
184
185
186 # In[62]:
187
188
189 plot_roc_multiclass(tpr, fpr, roc_auc, selection=None, plot_averages=False, plot_legend=True)
190
191
192 # In[59]:
193
194
195 physionetchallenge_sel = '270492004,427393009,426177001,426783006,427084000,17338001'.split(',')
196 physionetchallenge_sel = [df1.columns.get_loc(s)-1 for s in physionetchallenge_sel]
197 plot_roc_multiclass(tpr, fpr, roc_auc, selection=physionetchallenge_sel, plot_averages=False, plot_legend=True)
198
199
200 # ### Calculate best thresholds for each class
201
202 # In[63]:
203
204
205 def calculate_best_thresholds(tpr, fpr, thresholds, n_classes):
206     best_threshold = dict()
207     for i in range(n_classes):
208         if np.isnan(fpr[i]).all() or np.isnan(tpr[i]).all():
209             best_threshold[i]=np.nan
210         else:
211             ix = np.argmax(tpr[i]-fpr[i]) #youden J
212             #ix = np.argmax(np.sqrt(tpr[i] * (1-fpr[i]))) #gmeans

```

```

213         best_threshold[i] = thresholds[i][ix]
214     return best_threshold
215
216
217 # In[64]:
218
219
220 calculate_best_thresholds(tpr, fpr, thresholds, n_classes) #returns nan if tpr or fpr is nan
221
222
223 # In[65]:
224
225
226 best_thresholds = calculate_best_thresholds(tpr, fpr, thresholds, n_classes)
227 plt.scatter(best_thresholds.keys(), best_thresholds.values())
228 best_thresholds = np.array(list(best_thresholds.values()))
229
230
231 # ### Convert probability scores to binary predictions
232
233 # In[66]:
234
235
236 def convert_score_to_binary(pred_scores, decision_thresholds):
237     return (pred_scores > decision_thresholds).astype(int)
238
239
240 # In[115]:
241
242
243 binary_pred = convert_score_to_binary(pred, best_thresholds)#+0.01
244 binary_pred_sum = np.round(np.sum(binary_pred, axis=0).astype(float), 0).astype(int)
245 binary_pred_sum
246
247
248 # In[68]:
249
250
251 fig = plt.figure(dpi=600)
252 width = 0.6
253 plt.bar(np.arange(len(label_sum)), label_sum, width=width, label='Sum of labels')
254 plt.bar(np.arange(len(label_sum))+(1-width), binary_pred_sum, width=width, label='Sum of binary predictions')
255 plt.ylabel('Log count')
256 plt.xlabel('Class')
257 plt.yscale('log')
258 plt.legend(prop={'size': 6})
259 plt.show()
260
261
262 # Below taken (and changed) from: https://github.com/physionetchallenge/evaluation-2020/blob/master/
263     evaluate_12ECG_score.py
264 # In[121]:
265
266
267 #!/usr/bin/env python
268
269 # This file contains functions for evaluating algorithms for the 2020 PhysioNet/
270 # Computing in Cardiology Challenge. You can run it as follows:
271 #
272 #     python evaluate_12ECG_score.py labels outputs scores.csv
273 #
274 # where 'labels' is a directory containing files with the labels, 'outputs' is a
275 # directory containing files with the outputs from your model, and 'scores.csv'
276 # (optional) is a collection of scores for the algorithm outputs.
277 #
278 # Each file of labels or outputs must have the format described on the Challenge
279 # webpage. The scores for the algorithm outputs include the area under the
280 # receiver-operating characteristic curve (AUROC), the area under the recall-
281 # precision curve (AUPRC), accuracy (fraction of correct recordings), macro F-
282 # measure, and the Challenge metric, which assigns different weights to
283 # different misclassification errors.
284
285 import numpy as np, os, os.path, sys
286
287 def evaluate_12ECG_score(label_scores_data, label_binary_data, labels):
288     # Define the weights, the SNOMED CT code for the normal class, and equivalent SNOMED CT codes.
289     weights_file = 'weights.csv'
290     normal_class = '426783006'
291     equivalent_classes = [['713427006', '59118001'], ['284470004', '63593006'], ['427172004', '17338001']]
292
293     # Load the scored classes and the weights for the Challenge metric.
294     print('Loading weights...')
295     classes, weights = load_weights(weights_file, []) #equivalent_classes
296     #
297     classes =
298         '270492004,164889003,164890007,426627000,713427006,713426002,445118002,39732003,164909002,251146004,698252002,10370003,284470004,4271
299     # Load the label and output files.
300     print('Loading label and output files...')
```

```

300     binary_outputs = label_binary_data
301     scalar_outputs = label_scores_data
302
303     # Evaluate the model by comparing the labels and outputs.
304     print('Evaluating model...')
305
306     print('- AUROC and AUPRC...')
307     auroc, auprc, auroc_classes, auprc_classes = compute_auc(labels, scalar_outputs)
308
309     print('- Accuracy...')
310     accuracy = compute_accuracy(labels, binary_outputs)
311
312     print('- F-measure...')
313     f_measure, f_measure_classes = compute_f_measure(labels, binary_outputs)
314
315     print('- F-beta and G-beta measures...')
316     f_beta_measure, g_beta_measure = compute_beta_measures(labels, binary_outputs, beta=2)
317
318     print('- Challenge metric...(skipped)')
319     challenge_metric = compute_challenge_metric(weights, labels, binary_outputs, classes, normal_class)
320
321     print('Done.')
322
323     # Return the results.
324     return classes, auroc, auprc, auroc_classes, auprc_classes, accuracy, f_measure, f_measure_classes, f_beta_measure
325     , g_beta_measure, challenge_metric
326
327 # Check if the input is a number.
328 def is_number(x):
329     try:
330         float(x)
331         return True
332     except ValueError:
333         return False
334
335 # Load weights.
336 def load_weights(weight_file, equivalent_classes):
337     # Load the weight matrix.
338     rows, cols, values = load_table(weight_file)
339     assert(rows == cols)
340
341     # For each collection of equivalent classes, replace each class with the representative class for the set.
342     rows = replace_equivalent_classes(rows, equivalent_classes)
343
344     # Check that equivalent classes have identical weights.
345     for j, x in enumerate(rows):
346         for k, y in enumerate(rows[j+1:]):
347             if x==y:
348                 assert(np.all(values[j, :]==values[j+k, :]))
349                 assert(np.all(values[:, j]==values[:, j+k]))
350
351     # Use representative classes.
352     classes = [x for j, x in enumerate(rows) if x not in rows[:j]]
353     indices = [rows.index(x) for x in classes]
354     weights = values[np.ix_(indices, indices)]
355
356     return classes, weights
357
358 def load_table(table_file):
359     # The table should have the following form:
360     #
361     # ,   a,   b,   c
362     # a, 1.2, 2.3, 3.4
363     # b, 4.5, 5.6, 6.7
364     # c, 7.8, 8.9, 9.0
365     #
366     table = list()
367     with open(table_file, 'r') as f:
368         for i, l in enumerate(f):
369             arrs = [arr.strip() for arr in l.split(',')]
370             table.append(arrs)
371
372     # Define the numbers of rows and columns and check for errors.
373     num_rows = len(table)-1
374     if num_rows<1:
375         raise Exception('The table {} is empty.'.format(table_file))
376
377     num_cols = set(len(table[i])-1 for i in range(num_rows))
378     if len(num_cols)!=1:
379         raise Exception('The table {} has rows with different lengths.'.format(table_file))
380     num_cols = min(num_cols)
381     if num_cols<1:
382         raise Exception('The table {} is empty.'.format(table_file))
383
384     # Find the row and column labels.
385     rows = [table[0][j+1] for j in range(num_rows)]
386     cols = [table[i+1][0] for i in range(num_cols)]
387
388     # Find the entries of the table.

```

```

389     values = np.zeros((num_rows, num_cols), dtype=np.float64)
390     for i in range(num_rows):
391         for j in range(num_cols):
392             value = table[i+1][j+1]
393             if is_number(value):
394                 values[i, j] = float(value)
395             else:
396                 values[i, j] = float('nan')
397
398     return rows, cols, values
399
400 # Compute recording-wise accuracy.
401 def compute_accuracy(labels, outputs):
402     num_recordings, num_classes = np.shape(labels)
403
404     num_correct_recordings = 0
405     for i in range(num_recordings):
406         if np.all(labels[i, :] == outputs[i, :]):
407             num_correct_recordings += 1
408
409     return float(num_correct_recordings) / float(num_recordings)
410
411 # Compute confusion matrices.
412 def compute_confusion_matrices(labels, outputs, normalize=False):
413     # Compute a binary confusion matrix for each class k:
414     #
415     #   [TN_k FN_k]
416     #   [FP_k TP_k]
417     #
418     # If the normalize variable is set to true, then normalize the contributions
419     # to the confusion matrix by the number of labels per recording.
420     num_recordings, num_classes = np.shape(labels)
421
422     if not normalize:
423         A = np.zeros((num_classes, 2, 2))
424         for i in range(num_recordings):
425             for j in range(num_classes):
426                 if labels[i, j]==1 and outputs[i, j]==1: # TP
427                     A[j, 1, 1] += 1
428                 elif labels[i, j]==0 and outputs[i, j]==1: # FP
429                     A[j, 1, 0] += 1
430                 elif labels[i, j]==1 and outputs[i, j]==0: # FN
431                     A[j, 0, 1] += 1
432                 elif labels[i, j]==0 and outputs[i, j]==0: # TN
433                     A[j, 0, 0] += 1
434                 else: # This condition should not happen.
435                     raise ValueError('Error in computing the confusion matrix.')
436
437     else:
438         A = np.zeros((num_classes, 2, 2))
439         for i in range(num_recordings):
440             normalization = float(max(np.sum(labels[i, :]), 1))
441             for j in range(num_classes):
442                 if labels[i, j]==1 and outputs[i, j]==1: # TP
443                     A[j, 1, 1] += 1.0/normalization
444                 elif labels[i, j]==0 and outputs[i, j]==1: # FP
445                     A[j, 1, 0] += 1.0/normalization
446                 elif labels[i, j]==1 and outputs[i, j]==0: # FN
447                     A[j, 0, 1] += 1.0/normalization
448                 elif labels[i, j]==0 and outputs[i, j]==0: # TN
449                     A[j, 0, 0] += 1.0/normalization
450                 else: # This condition should not happen.
451                     raise ValueError('Error in computing the confusion matrix.')
452
453     return A
454
455 # Compute macro F-measure.
456 def compute_f_measure(labels, outputs):
457     num_recordings, num_classes = np.shape(labels)
458
459     A = compute_confusion_matrices(labels, outputs)
460
461     f_measure = np.zeros(num_classes)
462     for k in range(num_classes):
463         tp, fp, fn, tn = A[k, 1, 1], A[k, 1, 0], A[k, 0, 1], A[k, 0, 0]
464         if 2 * tp + fp + fn:
465             f_measure[k] = float(2 * tp) / float(2 * tp + fp + fn)
466         else:
467             f_measure[k] = float('nan')
468
469     macro_f_measure = np.nanmean(f_measure)
470
471     return macro_f_measure, f_measure
472
473 # Compute F-beta and G-beta measures from the unofficial phase of the Challenge.
474 def compute_beta_measures(labels, outputs, beta):
475     num_recordings, num_classes = np.shape(labels)
476
477     A = compute_confusion_matrices(labels, outputs, normalize=True)
478     f_beta_measure = np.zeros(num_classes)

```

```

479     g_beta_measure = np.zeros(num_classes)
480     for k in range(num_classes):
481         tp, fp, fn, tn = A[k, 1, 1], A[k, 1, 0], A[k, 0, 1], A[k, 0, 0]
482         if (1+beta**2)*tp + fp + beta**2*fn:
483             f_beta_measure[k] = float((1+beta**2)*tp) / float((1+beta**2)*tp + fp + beta**2*fn)
484         else:
485             f_beta_measure[k] = float('nan')
486         if tp + fp + beta*fn:
487             g_beta_measure[k] = float(tp) / float(tp + fp + beta*fn)
488         else:
489             g_beta_measure[k] = float('nan')
490
491     macro_f_beta_measure = np.nanmean(f_beta_measure)
492     macro_g_beta_measure = np.nanmean(g_beta_measure)
493
494     return macro_f_beta_measure, macro_g_beta_measure
495
496 # Compute macro AUROC and macro AUPRC.
497 def compute_auc(labels, outputs):
498     num_recordings, num_classes = np.shape(labels)
499
500     # Compute and summarize the confusion matrices for each class across at distinct output values.
501     auroc = np.zeros(num_classes)
502     auprc = np.zeros(num_classes)
503
504     for k in range(num_classes):
505         # We only need to compute TPs, FPs, FNs, and TNs at distinct output values.
506         thresholds = np.unique(outputs[:, k])
507         thresholds = np.append(thresholds, thresholds[-1]+1)
508         thresholds = thresholds[::-1]
509         num_thresholds = len(thresholds)
510
511         # Initialize the TPs, FPs, FNs, and TNs.
512         tp = np.zeros(num_thresholds)
513         fp = np.zeros(num_thresholds)
514         fn = np.zeros(num_thresholds)
515         tn = np.zeros(num_thresholds)
516         fn[0] = np.sum(labels[:, k]==1)
517         tn[0] = np.sum(labels[:, k]==0)
518
519         # Find the indices that result in sorted output values.
520         idx = np.argsort(outputs[:, k])[::-1]
521
522         # Compute the TPs, FPs, FNs, and TNs for class k across thresholds.
523         i = 0
524         for j in range(1, num_thresholds):
525             # Initialize TPs, FPs, FNs, and TNs using values at previous threshold.
526             tp[j] = tp[j-1]
527             fp[j] = fp[j-1]
528             fn[j] = fn[j-1]
529             tn[j] = tn[j-1]
530
531             # Update the TPs, FPs, FNs, and TNs at i-th output value.
532             while i < num_recordings and outputs[idx[i], k] >= thresholds[j]:
533                 if labels[idx[i], k]:
534                     tp[j] += 1
535                     fn[j] -= 1
536                 else:
537                     fp[j] += 1
538                     tn[j] -= 1
539                 i += 1
540
541         # Summarize the TPs, FPs, FNs, and TNs for class k.
542         tpr = np.zeros(num_thresholds)
543         tnr = np.zeros(num_thresholds)
544         ppv = np.zeros(num_thresholds)
545         for j in range(num_thresholds):
546             if tp[j] + fn[j]:
547                 tpr[j] = float(tp[j]) / float(tp[j] + fn[j])
548             else:
549                 tpr[j] = float('nan')
550             if fp[j] + tn[j]:
551                 tnr[j] = float(tn[j]) / float(fp[j] + tn[j])
552             else:
553                 tnr[j] = float('nan')
554             if tp[j] + fp[j]:
555                 ppv[j] = float(tp[j]) / float(tp[j] + fp[j])
556             else:
557                 ppv[j] = float('nan')
558
559         # Compute AUROC as the area under a piecewise linear function with TPR/
560         # sensitivity (x-axis) and TNR/specificity (y-axis) and AUPRC as the area
561         # under a piecewise constant with TPR/recall (x-axis) and PPV/precision
562         # (y-axis) for class k.
563         for j in range(num_thresholds-1):
564             auroc[k] += 0.5 * (tpr[j+1] - tpr[j]) * (tnr[j+1] + tnr[j])
565             auprc[k] += (tpr[j+1] - tpr[j]) * ppv[j+1]
566
567     # Compute macro AUROC and macro AUPRC across classes.
568     macro_auroc = np.nanmean(auroc)

```

```

569     macro_auprc = np.nanmean(auprc)
570
571     return macro_auroc, macro_auprc, auroc, auprc
572
573 # Compute modified confusion matrix for multi-class, multi-label tasks.
574 def compute_modified_confusion_matrix(labels, outputs):
575     # Compute a binary multi-class, multi-label confusion matrix, where the rows
576     # are the labels and the columns are the outputs.
577     num_recordings, num_classes = np.shape(labels)
578     A = np.zeros((num_classes, num_classes))
579
580     # Iterate over all of the recordings.
581     for i in range(num_recordings):
582         # Calculate the number of positive labels and/or outputs.
583         normalization = float(max(np.sum(np.any((labels[i, :], outputs[i, :]), axis=0)), 1))
584         # Iterate over all of the classes.
585         for j in range(num_classes):
586             # Assign full and/or partial credit for each positive class.
587             if labels[i, j]:
588                 for k in range(num_classes):
589                     if outputs[i, k]:
590                         A[j, k] += 1.0/normalization
591
592     return A
593
594 # Compute the evaluation metric for the Challenge.
595 def compute_challenge_metric(weights, labels, outputs, classes, normal_class):
596     num_recordings, num_classes = np.shape(labels)
597     normal_index = classes.index(normal_class)
598
599     # Compute the observed score.
600     A = compute_modified_confusion_matrix(labels, outputs)
601     observed_score = np.nansum(weights * A)
602
603     # Compute the score for the model that always chooses the correct label(s).
604     correct_outputs = labels
605     A = compute_modified_confusion_matrix(labels, correct_outputs)
606     correct_score = np.nansum(weights * A)
607
608     # Compute the score for the model that always chooses the normal class.
609     inactive_outputs = np.zeros((num_recordings, num_classes), dtype=np.bool)
610     inactive_outputs[:, normal_index] = 1
611     A = compute_modified_confusion_matrix(labels, inactive_outputs)
612     inactive_score = np.nansum(weights * A)
613
614     if correct_score != inactive_score:
615         normalized_score = float(observed_score - inactive_score) / float(correct_score - inactive_score)
616     else:
617         normalized_score = 0.0
618
619     return normalized_score
620
621 def replace_equivalent_classes(classes, equivalent_classes):
622     for j, x in enumerate(classes):
623         for multiple_classes in equivalent_classes:
624             if x in multiple_classes:
625                 classes[j] = multiple_classes[0] # Use the first class as the representative class.
626
627     return classes
628
629 # In[35]:
630
631
632 np.sum(binary_pred, axis=0)
633
634
635 # ### only use selection of classes and replace rest by normal (weird workaround)
636 #
637
638 # In[118]:
639
640
641 def replace_unwanted_labels_by_normal(labels, selection_ixs, replace_ix=None):
642     #Replaces all labels that are not in the selection_ixs with the normal label
643     if replace_ix is None:
644         new_labels = np.zeros((len(labels), len(selection_ixs)+1))
645         replace_ix = len(selection_ixs)
646     else:
647         new_labels = np.zeros((len(labels), len(selection_ixs)))
648         replace_ix = selection_ixs.index(replace_ix)
649         inverted_selection = list(set(list(range(labels.shape[1])))-set(selection_ixs))
650         new_labels[:, 0:len(selection_ixs)] = labels[:, selection_ixs]
651         new_labels[:, replace_ix] = np.where(np.sum(labels[:, inverted_selection], axis=1)>0, 1, labels[:, replace_ix])
652
653     return new_labels
654
655
656 # In[119]:
657
658

```

```

659 physionetchallenge_sel = '
660     270492004,164889003,164890007,426627000,713427006,713426002,445118002,39732003,164909002,251146004,698252002,10370003,284470004,427172004,164
661     '.split(',')
660 physionetchallenge_sel = [dfl.columns.get_loc(s)-1 for s in physionetchallenge_sel]
661 replace_unwanted_labels_by_normal(LABELS, physionetchallenge_sel, dfl.columns.get_loc('426783006')-1)
662
663
664 # ### Claculate challenge metrics
665
666 # In[122]:
667
668
669 predr = replace_unwanted_labels_by_normal(pred, physionetchallenge_sel, dfl.columns.get_loc('426783006')-1)
670 binary_predr = replace_unwanted_labels_by_normal(binary_pred, physionetchallenge_sel, dfl.columns.get_loc('426783006')-1)
671 labelsr = replace_unwanted_labels_by_normal(LABELS, physionetchallenge_sel, dfl.columns.get_loc('426783006')-1)
672 out = evaluate_12ECG_score(predr, binary_predr, labelsr)
673 classes, auroc, auprc, auroc_classes, auprc_classes, accuracy, f_measure, f_measure_classes, f_beta_measure,
674     g_beta_measure, challenge_metric = out
675 output_string = 'AUROC,AUPRC,Accuracy,F-measure,Fbeta-measure,Gbeta-measure,Challenge metric\n{:.3f},{:.3f},{:.3f}
676     ,{:.3f},{:.3f},{:.3f}'.format(auroc, auprc, accuracy, f_measure, f_beta_measure, g_beta_measure,
677     challenge_metric)
678 class_output_string = 'Classes,{}\nAUROC,{}\nAUPRC,{}\nF-measure,{}'.format(
679     '/'.join(classes),
680     '/'.join('{:.3f}'.format(x) for x in auroc_classes),
681     '/'.join('{:.3f}'.format(x) for x in auprc_classes),
682     '/'.join('{:.3f}'.format(x) for x in f_measure_classes))
683 print('#####')
684 print(class_output_string)

```

B.7.16 jupyter_notebooks -> Sinus Generator.py (code)

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[4]:
5
6
7 import numpy as np
8 import matplotlib.pyplot as plt
9 import h5py as h5
10 import os
11
12
13 # In[5]:
14
15
16 def generate_sinus_data(n, hz_anim, add_channels_with_shift=0, add_noise=False):
17     #hz_anim: a function that takes a single int as input and outputs a hz number. (Must be defined for 0 to n)
18     signal = np.empty((n, add_channels_with_shift+1))
19     if add_channels_with_shift > 0:
20         shift = np.random.randint(1, 30, add_channels_with_shift)
21     for i in range(n):
22         hz = hz_anim(i)
23         signal[i, 0] = np.sin((i)/hz*2*np.pi) + add_noise*np.random.normal(0, 0.03, 1)
24         for j in range(add_channels_with_shift):
25             signal[i, 1+j] = np.sin((i+shift[j])/hz*2*np.pi) + add_noise*np.random.normal(0, 0.03, 1)
26     print(signal.shape)
27     signal = (signal-np.min(signal))/(np.max(signal)-np.min(signal))
28
29     return signal
30
31 def save_h5(outpath, fname, data):
32     with h5.File(os.path.join(outpath, fname), 'w') as f:
33         f['data'] = data
34         f.flush()
35
36
37
38 # In[73]:
39
40
41 signal = generate_sinus_data(115000, lambda x: (200/115000)*x+800, add_channels_with_shift=11, add_noise=False)
42 #plt.plot(signal)
43
44
45
46 # In[ ]:
47
48
49 out = '/media/julian/Volume/data/sinus'
50 n_low, n_high = 115000, 150000
51 n_files = 100

```

```

52 hr_low , hr_high = 600, 1200
53 hr_fluc_low, hr_fluc_high = 100, 300
54 add_channels_with_shift=11
55 noise_chance = 0.1
56 for i in range(n_files):
57     length = np.random.randint(n_low, n_high)
58     noise = np.random.random() < noise_chance
59     hr_fluc = np.random.randint(hr_fluc_low, hr_fluc_high)
60     hr = np.random.randint(hr_low+hr_fluc, hr_high-hr_fluc)
61     signal = generate_sinus_data(length, lambda x: hr+np.sin(x/length*2)*hr_fluc, add_channels_with_shift=
62         add_channels_with_shift, add_noise=noise)
63     save_h5(out, 'sinus'+str(i)+'.h5', signal)
64
65 # In[16]:
66
67
68 plt.figure(figsize=(100, 4))
69 plt.plot(signal[:, 0])
70
71 # In[ ]:

```

B.7.17 jupyter_notebooks -> TSNE.py (code)

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[24]:
5
6
7 import pandas as pd
8 import numpy as np
9 import os
10 import glob
11 import seaborn as sns
12 import matplotlib.pyplot as plt
13 import matplotlib
14 import seaborn as sns
15 from sklearn.manifold import TSNE
16 import matplotlib as mpl
17 from itertools import cycle
18
19 from sklearn import svm, datasets
20 from sklearn.metrics import roc_curve, auc
21 from sklearn.model_selection import train_test_split
22 from sklearn.preprocessing import label_binarize
23 from sklearn.multiclass import OneVsRestClassifier
24 from scipy import interp
25 from sklearn.metrics import roc_auc_score
26 import colorcet as cc
27
28
29 # ## Import latent and context data
30
31 # In[38]:
32
33
34 import pickle
35 # with open('/home/julian/Downloads/Github/contrastive-predictive-coding/experiments/data_output/17_08_21-18/14_08_21
36 # -15_36-train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined1|train-test-splits-fewer-labels60|use_weights|
37 # strided|frozen[C|m|all|cpc_downstream_cnn/latent_List.pickle', 'rb') as f:
38 #     latent_list = pickle.load(f)
39 with open('/home/julian/Downloads/Github/contrastive-predictive-coding/experiments/data_output/05_02_22-15/28_01_22
40 # -16-23-train|(12x)cpc/architectures_cpc.cpc_combined.CPCCombined2|train-test-splits|use_weights|frozen[C|L|m|all
41 # |cpc_downstream_latent_maximum/latent_list.pickle', 'rb') as f:
42     latent_list = pickle.load(f)
43
44 class_idx = {'10370003': 0, '11157007': 1, '111975006': 2, '164861001': 3, '164865005': 4, '164867002': 5, '164873001'
45 : 6, '164884008': 7, '164889003': 8, '164890007': 9, '164909002': 10, '164917005': 11, '164930006': 12, '164931005': 13, '164934002': 14, '164947007': 15, '164951009': 16, '17338001': 17, '195042002': 18, '195080001': 19, '195126007': 20, '233917008': 21, '251120003': 22, '251146004': 23, '251200008': 25, '251266004': 26, '251268003': 27, '253352002': 28, '266249003': 29, '270492004': 30, '27885002': 31, '284470004': 32, '39732003': 33, '413844008': 34, '425419005': 35, '425623009': 36, '426177001': 37, '426434006': 38, '426627000': 39, '426761007': 40, '426783006': 41, '427084000': 42, '427172004': 43, '427393009': 44, '428417006': 45, '428750005': 46, '429622005': 47, '445118002': 48, '445211001': 49, '446358003': 50, '446813000': 51, '47665007': 52, '54329005': 53, '55930002': 54, '59118001': 55, '59931005': 56, '63593006': 57, '6374002': 58, '67198005': 59, '67741000119109': 60, '698252002': 61, '713422000': 62, '713426002': 63, '713427006': 64, '74390002': 65, '89792004': 66}
46 class_idx_r = {v:k for k,v in class_idx.items()}
47 palette = sns.color_palette(cc.glasbey, n_colors=len(class_idx))
48
49 # In[12]:
50
51
52 print('shape of one latent:', latent_list[0]['latents'].shape)

```

```

48
49
50 # In[13]:
51
52
53 print('shape of one context:', latent_list[0]['context'].shape)
54
55
56 # In[14]:
57
58
59 all_latents = np.concatenate([np.squeeze(l['latents']) for l in latent_list], axis=0)
60 print('got latents:', all_latents.shape)
61
62
63 # In[15]:
64
65
66 all_contexts = np.stack([np.squeeze(l['context']) for l in latent_list], axis=0)
67 print('got contexts:', all_contexts.shape)
68
69
70 # ## Calc TSNE representation (latent)
71
72 # In[8]:
73
74
75 np.random.seed(0)
76 X_embedded = TSNE(n_components=2, n_jobs=6).fit_transform(all_latents)
77 X_embedded.shape
78
79
80 # In[48]:
81
82
83 selected_class = '426783006'
84 normal_class_ix = labels = [1 if normal_class_ix in l['labels'][1] else 0 for l in latent_list]
85
86 first_labels = [item for sublist in [[1]*26 if normal_class_ix in l['labels'][1] else [0]*26 for l in latent_list]
87     for item in sublist]
88 first_labels = [item for sublist in [[l['labels'][1][0]]*26 for l in latent_list] for item in sublist]
89 df = pd.DataFrame()
90 df["y"] = first_labels
91 df["comp-1"] = X_embedded[:,0]
92 df["comp-2"] = X_embedded[:,1]
93 fig = plt.figure(dpi=600)
94 plt.tight_layout()
95 ax = sns.scatterplot(x="comp-1", y="comp-2", hue=df.y.tolist(),
96                      data=df, s=0.5, cmap=palette)
97 ax.set(title="Latent data T-SNE projection (true labels)")
98 handles, labels = ax.get_legend_handles_labels()
99 ax.legend(handles, [f'class {selected_class} is not part of label', f'class {selected_class} is part of label'],
100           fontsize='xx-small')
101 fig.savefig('/home/julian/Downloads/tsne-latent-41.png', dpi=fig.dpi)
102
103 # In[19]:
104
105
106 import random
107 df = pd.DataFrame()
108 df["y"] = first_labels
109 df["comp-1"] = X_embedded[:,0]
110 df["comp-2"] = X_embedded[:,1]
111 fig = plt.figure(dpi=600)
112 plt.tight_layout()
113 random_labels = first_labels.copy()
114 random.shuffle(random_labels)
115 sns.scatterplot(x="comp-1", y="comp-2", hue=random_labels,
116                      data=df, s=0.5).set(title="Latent data T-SNE projection (random labels)")
117 fig.savefig('/home/julian/Downloads/tsne-latent-(random).png', dpi=fig.dpi)
118
119
120 # ## Calc TSNE representation (context)
121
122 # In[9]:
123
124
125 np.random.seed(0)
126 C_embedded = TSNE(n_components=2, n_jobs=6).fit_transform(all_contexts)
127 C_embedded.shape
128
129
130 # In[52]:
131
132
133 counts = dict(zip(range(len(counts)), counts))
134 def get_lowest_count_class(idx_list):
135     return min([(i, counts[i]) for i in idx_list], key=lambda x:x[1])[0]

```

```

136
137
138 # In[54]:
139
140
141 normal_class_ix = class_idx['426783006']
142 #labels = [1 if normal_class_ix in l['labels'][1] else 0 for l in latent_list]
143 labels = [get_lowest_count_class(l['labels'][1]) for l in latent_list]
144 df = pd.DataFrame()
145 df["y"] = labels
146 df["comp-1"] = C_embedded[:,0]
147 df["comp-2"] = C_embedded[:,1]
148 fig = plt.figure(dpi=600)
149 sns.scatterplot(x="comp-1", y="comp-2", hue=df.y.tolist(),
150                   data=df, s=5).set(title="Context data T-SNE projection (true labels)")
151 fig.savefig('/home/julian/Downloads/tsne-context-41.png', dpi=fig.dpi)
152
153
154 # In[24]:
155
156
157 df = pd.DataFrame()
158 df["y"] = labels
159 df["comp-1"] = C_embedded[:,0]
160 df["comp-2"] = C_embedded[:,1]
161 fig = plt.figure(dpi=600)
162 plt.tight_layout()
163 random_labels = labels.copy()
164 random.shuffle(random_labels)
165 sns.scatterplot(x="comp-1", y="comp-2", hue=random_labels,
166                   data=df, s=5).set(title="Context data T-SNE projection (random)")
167 fig.savefig('tsne-context-(random).png', dpi=fig.dpi)
168
169
170 # ## Calculate for all classes
171
172 # In[36]:
173
174
175 def get_label_for_label_index(all_labels, index, repeat=1):
176     labels = [item for sublist in [[1]*repeat if index in l['labels'][1] else [0]*repeat for l in latent_list] for item in sublist]
177     return np.array(labels)
178
179
180 # ## order by count
181
182 # In[34]:
183
184
185 LABELS = []
186 for l in [l['labels'][1] for l in latent_list]:
187     temp = np.zeros(len(class_idx))
188     temp[l] = 1
189     LABELS.append(temp)
190 LABELS = np.stack(LABELS)
191 LABELS.shape
192
193
194 # In[ ]:
195
196
197 all_labels = [l['labels'][1] for l in latent_list]
198 fig, axes = plt.subplots(8, 8, dpi=1200, sharex='all', gridspec_kw={'wspace':0, 'hspace':0})
199 cmap = matplotlib.colors.ListedColormap(['red', 'black'])
200 label_index_order = np.argsort(-1*np.sum(LABELS, axis=0))
201 plt.axis('on')
202 plt.tight_layout()
203 for i in range(8):
204     for j in range(8):
205         ax = axes[i, j]
206         ax.set_yticklabels([])
207         ax.set_xticklabels([])
208         ax.tick_params(axis='both', which='both', bottom=False, top=False, left=False, right=False, labelbottom=False)
209         n = i*8+j
210         labels = get_label_for_label_index(all_labels, label_index_order[n], repeat=26)
211         ax.scatter(X_embedded[:,0], X_embedded[:,1], c=labels, s=(labels+0.3)/2, linewidths=0, marker='.', label=
212             class_idx_r[label_index_order[n]])
213         ax.legend(prop={'size':12}, loc=1, frameon=False)
214 fig.savefig('/home/julian/Downloads/tsne-latent-all.png', dpi=fig.dpi)
215
216
217 # ## order by auc score
218
219 # In[78]:
220
221 classes = "10370003 & 11157007 & 111975006 & 164861001 & 164865005 & 164867002 & 164873001 & 164884008 &
222     164889003 & 164890007 & 164909002 & 164917005 & 164930006 & 164931005 & 164934002 & 164947007 &
223     164951009 & 17338001 & 195042002 & 195080001 & 195126007 & 233917008 & 251120003 & 251146004 & 251180001

```

```

    & 251200008 & 251266004 & 251268003 & 253352002 & 266249003 & 270492004 & 27885002 & 284470004 &
39732003 & 413844008 & 425419005 & 425623009 & 426177001 & 426434006 & 426627000 & 426761007 & 426783006
& 427084000 & 427172004 & 427393009 & 428417006 & 428750005 & 429622005 & 445118002 & 445211001 &
446358003 & 446813000 & 47665007 & 54329005 & 55930002 & 59118001 & 59931005 & 63593006 & 6374002 &
67198005 & 67741000119109 & 698252002 & 713422000 & 713426002 & 713427006 & 74390002 & 89792004".replace(
', '').split('&')
222 aucs = '0.940 & 0.935 & 0.743 & 0.875 & 0.773 & 0.772 & 0.828 & 0.794 & 0.834 &
0.805 & 0.965 & 0.766 & 0.694 & 0.814 & 0.794 & 0.828 & 0.785 &
0.740 & 0.751 & 0.802 & 0.823 & 0.751 & 0.920 & 0.813 & 0.719 & 0.850 &
0.912 & 0.844 & 0.914 & 0.708 & 0.685 & 0.886 & 0.638 & 0.899 & 0.835
& 0.825 & 0.874 & 0.951 & 0.817 & 0.840 & 0.909 & 0.784 & 0.954 &
0.783 & 0.707 & 0.864 & 0.770 & 0.789 & 0.924 & 0.905 & 0.857 & 0.946 &
0.944 & 0.812 & 0.757 & 0.921 & 0.796 & 0.821 & 0.973 & 0.823 & 0.751 &
0.707 & 0.828 & 0.810 & 0.956 & 0.992 & 0.860'.replace(' ', '').split('&')
223 aucs = list(map(float, aucs))
224 class_auc_map = dict(zip(classes, aucs))
225 sorted_classes = sorted(class_auc_map.items(), key=lambda x: -x[1]) #sort by auc score
226 label_index_order = [df1.columns[1:][get_loc(s[0]) for s in sorted_classes]
227 color_norm = matplotlib.colors.Normalize(.5, 1.)
228
229 cmap = sns.color_palette("magma", as_cmap=True)
230
231 all_labels = [l['labels'][1] for l in latent_list]
232 fig, axs = plt.subplots(8, 8, dpi=1200, sharex='all', gridspec_kw={'wspace':0, 'hspace':0})
233 plt.axis('on')
234 for i in range(8):
235     for j in range(8):
236         ax = axs[i, j]
237         ax.set_yticklabels([])
238         ax.set_xticklabels([])
239         ax.tick_params(axis='both', which='both', bottom=False, top=False, left=False, right=False, labelbottom=False)
240         n = i*8+j
241         labels = get_label_for_label_index(all_labels, label_index_order[n], repeat=9)
242         class_name = df1.columns[1:][label_index_order[n]]
243         pcm = ax.scatter(X_embedded[:,0], X_embedded[:,1], c=labels, s=(labels+0.2)/2, linewidths=0, marker='.', label=
class_name)
244         ax.legend(prop={'size':2}, loc=1, frameon=False)
245         ax.set_facecolor(cmap(color_norm(class_auc_map[class_name]), alpha=0.7))
246 plt.tight_layout()
247 cax_kw = mpl.colorbar.make_axes([ax for ax in axs.flat], aspect=40, pad=0.0)
248 cbar = plt.colorbar(mpl.cm.ScalarMappable(norm=color_norm, cmap=cmap), cax=cax, **kw)
249 cbar.set_label('AUC score for given class', rotation=90, fontsize='xx-small')
250 cbar.ax.tick_params(labelsize=5)
251 fig.savefig('/home/julian/Downloads/tsne-latent-all(ordered by auc).png', dpi=fig.dpi, bbox_inches='tight')
252
253
254 # ## order by count
255
256 # In[57]:
257
258
259 all_labels = [l['labels'][1] for l in latent_list]
260 fig, axs = plt.subplots(8, 8, dpi=1200, sharex='all', gridspec_kw={'wspace':0, 'hspace':0})
261 cmap = matplotlib.colors.ListedColormap(['red', 'black'])
262 label_index_order = np.argsort(-1*np.sum(LABELS, axis=0))
263 color_norm = matplotlib.colors.Normalize(0.5, 1.0)
264 cmap = sns.color_palette("magma", as_cmap=True)
265 plt.axis('on')
266 #fig.suptitle('t-SNE context embedding for all classes sorted by occurrences in data')
267 for i in range(8):
268     for j in range(8):
269         ax = axs[i, j]
270         ax.set_yticklabels([])
271         ax.set_xticklabels([])
272         ax.tick_params(axis='both', which='both', bottom=False, top=False, left=False, right=False, labelbottom=False)
273         n = i*8+j
274         labels = get_label_for_label_index(all_labels, label_index_order[n], repeat=1)
275         class_name = df1.columns[1:][label_index_order[n]]
276         ax.scatter(C_embedded[:,0], C_embedded[:,1], c=labels, s=(labels+1), linewidths=0, marker='.', label=
class_name)
277         #ax.set_facecolor('xkcd:salmon')
278         ax.legend(prop={'size':2}, loc=1, frameon=False)
279         ax.set_facecolor(cmap(color_norm(class_auc_map[class_name]), alpha=0.7))
280 plt.tight_layout()
281 cax_kw = mpl.colorbar.make_axes([ax for ax in axs.flat], aspect=40, pad=0.0)
282 cbar = plt.colorbar(mpl.cm.ScalarMappable(norm=color_norm, cmap=cmap), cax=cax, **kw)
283 cbar.set_label('AUC score for given class', rotation=90, fontsize='xx-small')
284 cbar.ax.tick_params(labelsize=5)
285 fig.savefig('/home/julian/Downloads/tsne-context-all(ordererd by count).png', dpi=fig.dpi)
286
287
288 # In[58]:
289
290
291 classes = "10370003 & 11157007 & 111975006 & 164861001 & 164865005 & 164867002 & 164873001 & 164884008 &
164889003 & 164890007 & 164909002 & 164917005 & 164930006 & 164931005 & 164934002 & 164947007 &
164951009 & 17338001 & 195042002 & 195080001 & 195126007 & 233917008 & 251120003 & 251146004 & 251180001
& 251200008 & 251266004 & 251268003 & 253352002 & 266249003 & 270492004 & 27885002 & 284470004 &
39732003 & 413844008 & 425419005 & 425623009 & 426177001 & 426434006 & 426627000 & 426761007 & 426783006
& 427084000 & 427172004 & 427393009 & 428417006 & 428750005 & 429622005 & 445118002 & 445211001 &

```

```

446358003 & 446813000 & 47665007 & 54329005 & 55930002 & 59118001 & 59931005 & 63593006 & 6374002 &
67198005 & 67741000119109 & 698252002 & 713422000 & 713426002 & 713427006 & 74390002 & 89792004".replace(
', '').split('&')
292 aucs = '0.940 & 0.935 & 0.743 & 0.875 & 0.773 & 0.772 & 0.828 & 0.794 & 0.834 &
0.805 & 0.965 & 0.766 & 0.694 & 0.814 & 0.794 & 0.828 & 0.785 &
0.740 & 0.751 & 0.802 & 0.823 & 0.751 & 0.920 & 0.813 & 0.719 & 0.850 &
0.912 & 0.844 & 0.914 & 0.708 & 0.685 & 0.886 & 0.638 & 0.899 & 0.835
& 0.825 & 0.874 & 0.951 & 0.817 & 0.840 & 0.909 & 0.784 & 0.954 &
0.783 & 0.707 & 0.864 & 0.770 & 0.789 & 0.924 & 0.905 & 0.857 & 0.946 &
0.944 & 0.812 & 0.757 & 0.921 & 0.796 & 0.821 & 0.973 & 0.823 & 0.751 &
0.707 & 0.828 & 0.810 & 0.956 & 0.992 & 0.860'.replace(' ', '').split('&')
293 aucs = list(map(float, aucs))
294 class_auc_map = dict(zip(classes, aucs))
295 sorted_classes = sorted(class_auc_map.items(), key=lambda x: -x[1]) #sort by auc score
296 label_index_order = [class_idx[s[0]] for s in sorted_classes]
297 all_labels = [l['labels'][1] for l in latent_list]
298 fig, axs = plt.subplots(8, 8, dpi=1200, sharex='all', gridspec_kw={'wspace':0, 'hspace':0})
299 plt.axis('on')
300 color_norm = matplotlib.colors.Normalize(0.5, 1.0)
301 cmap = sns.color_palette("magma", as_cmap=True)
302 #fig.suptitle('t-SNE context embedding for all classes sorted by occurrences in data')
303 for i in range(8):
304     for j in range(8):
305         ax = axs[i, j]
306         ax.set_yticklabels([])
307         ax.set_xticklabels([])
308         ax.tick_params(axis='both', which='both', bottom=False, top=False, left=False, right=False, labelbottom=False)
309         n = i*8+j
310         class_name = class_idx_r[label_index_order[n]]
311         labels = np.array([get_lowest_count_class(l['labels'][1]) for l in latent_list])#get_label_for_label_index(
312         all_labels, label_index_order[n], repeat=1)
313         ax.scatter(C_embedded[:,0], C_embedded[:,1], c=labels, s=1, linewidths=0, marker='.', label=class_name)
314         ax.set_facecolor(cmap(color_norm(class_auc_map[class_name])), alpha=0.7)
315         ax.legend(prop={'size':2}, loc=1, bbox_to_anchor=(1.02, 1.02), frameon=False)
316
316 cax, kw = mpl.colorbar.make_axes([ax for ax in axs.flat], aspect=40, pad=0.0)
317 cbar = plt.colorbar(mpl.cm.ScalarMappable(norm=color_norm, cmap=cmap), cax=cax, **kw)
318 cbar.set_label('AUC score for given class', rotation=90, fontsize='xx-small')
319 cbar.ax.tick_params(labelsize=5)
320
321 #fig.tight_layout()
322 fig.savefig('/home/julian/Downloads/tsne-context-all(ordered by auc).png', dpi=fig.dpi, bbox_inches='tight')
323
324
325 # ## Umap
326
327 # In[15]:
328
329
330 import umap
331 from sklearn.preprocessing import StandardScaler
332
333
334 # In[16]:
335
336
337 np.random.seed(0)
338 reducer = umap.UMAP()
339 L_embedded = reducer.fit_transform(StandardScaler().fit_transform(all_latents))
340
341
342 # In[18]:
343
344
345 normal_class_ix = 41
346 labels = [item for sublist in [[1]*9 if normal_class_ix in l['labels'][1] else [0]*9 for l in latent_list] for item in
347             sublist]
348 df = pd.DataFrame()
349 df["y"] = labels
350 df["comp-1"] = L_embedded[:,0]
351 df["comp-2"] = L_embedded[:,1]
352 plt.figure(dpi=600)
353 sns.scatterplot(x="comp-1", y="comp-2", hue=df.y.tolist(),
354                  data=df, s=0.5).set(title="Latent data UMAP projection (real labels)")
355
356
357 # In[22]:
358
359
360 def get_label_for_label_index(all_labels, index, repeat=1):
361     labels = [item for sublist in [[1]*repeat if index in l['labels'][1] else [0]*repeat for l in latent_list] for
362               item in sublist]
363     return labels
364 all_labels = [l['labels'][1] for l in latent_list]
365 fig, axs = plt.subplots(8, 8, dpi=600, sharex='all', gridspec_kw={'wspace':0, 'hspace':0})
366 cmap = matplotlib.colors.ListedColormap(['red', 'black'])
367 label_index_order = np.argsort(-1*np.sum(LABELS, axis=0))
368 plt.axis('on')
369 for i in range(8):

```

```

369     for j in range(8):
370         ax = axs[i, j]
371         ax.set_yticklabels([])
372         ax.set_xticklabels([])
373         ax.tick_params(axis='both', which='both', bottom=False, top=False, left=False, right=False, labelbottom=False)
374         n = i*8+j
375         labels = np.array(get_label_for_label_index(all_labels, label_index_order[n], repeat=9))
376         ax.scatter(L_embedded[:,0], L_embedded[:,1], c=labels, s=(labels+0.6)/3., linewidths=0, marker=',')
377 #fig.savefig('latent-tsne-all.png', dpi=fig.dpi)
378
379
380 # In[ ]:
```

B.8 logs (folder)

B.9 models_evaluated_filtered_csv (folder)

B.9.1 models_evaluated_filtered_csv -> old (folder)

B.10 util (folder)

B.10.1 util -> data (folder)

B.10.1.1 util -> data -> clear_h5_files.py (code)

```

1 import glob
2 import os
3
4 if __name__ == '__main__':
5     path = '/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
6         generated/1000/normalized-labels'
7     for f in glob.glob(path + '/**/*.h5'):
8         # print("clearing flags:", f)
9         os.system('h5clear -s ' + f)
```

B.10.1.2 util -> data -> dataframe_factory.py (code)

```

1 import re
2 import natsort
3 import pandas as pd
4 import os
5
6
7 class DataFrameFactory():
8     def __init__(self, data=None, column_labels=None, dataframe: pd.DataFrame = None, index_name=None):
9         self.columns = column_labels
10        self.dataframe = dataframe
11        if data is None:
12            self.dataframe = pd.DataFrame()
13        else:
14            self.dataframe = pd.DataFrame(data, columns=column_labels)
15
16    def append(self, data):
17        if self.dataframe is None or len(self.dataframe) == 0:
18            self.dataframe = pd.DataFrame(data, self.columns)
19        else:
20            dftemp = pd.DataFrame(data, columns=self.columns)
21            self.dataframe = self.dataframe.append(dftemp)
22
23    def get_dataframe(self):
24        return self.dataframe
25
26    def __str__(self):
27        return self.dataframe.__str__()
28
29    def to_csv(self, output_folder, filename):
30        p = os.path.join(output_folder, filename)
```

```

31     self.dataframe.to_csv(p)
32
33     def natsort_single_index(self, key=None):
34         convert = lambda text: int(text) if text.isdigit() else text.lower()
35         alphanum_key = lambda key: [[convert(c) for c in re.split('([0-9]+)', k)] for k in key]
36         self.dataframe.sort_index(key=alphanum_key, inplace=True)
37
38     def natsort_by_column(self, column, key=None):
39         print('Called')
40         convert = lambda text: int(text) if text.isdigit() else text.lower()
41         alphanum_key = lambda key: [[convert(c) for c in re.split('([0-9]+)', k)] for k in key]
42         self.dataframe.sort_values(by=column, key=alphanum_key, inplace=True, ignore_index=True)
43         self.dataframe.reset_index(drop=True, inplace=True)
44
45     def natsort_multi_index(self, key=None):
46         raise NotImplementedError
47         # self.dataframe.sort_index(inplace=True)
48         print(self.dataframe.index)
49         self.dataframe.set_index(natsort.natsorted(self.dataframe.index, key=key), inplace=True)
50
51     def to_latex(self, output_folder, filename, caption="", label="", description="", long_tables=False,
52                 only_tabular_environment=False):
53         p = os.path.join(output_folder, filename)
54         print(len(self.dataframe.columns.unique()), len(self.dataframe.columns))
55         print(len(self.dataframe.index.unique()), len(self.dataframe.index))
56         with pd.option_context("max_colwidth", 1000):
57             # latex_string = self.dataframe.to_latex(label=label, caption=caption, na_rep='-', float_format=".3f",
58             #                                         bold_rows=True, longtable=long_tables)
59             s = self.dataframe.style
60             s = s.highlight_max(subset=['micro', 'macro'], axis=0, props='bfseries; ; underline: --rwrap;')
61             s = s.format(formatter=lambda x: '{:.3f}'.format(x) if isinstance(x, float) else x, na_rep='-', escape='
62             latex')
63             s = s.set_caption(caption)
64             s = s.hide_index()
65             latex_string = s.to_latex()
66             print(latex_string)
67             typ = 'longtable' if long_tables else 'tabular'
68             latex_string = latex_string.replace(f"\begin{{{{typ}}}}", f"\begin{{adjustbox}}{{width={textwidth}}}\n\begin{{{{typ}}}}")
69             latex_string = latex_string.replace(f"\end{{{{typ}}}}", f"\end{{{{typ}}}}\n\end{{adjustbox}}")
70             latex_lines = latex_string.split('\n')
71             if description != "":
72                 latex_lines = latex_lines[:-1] + [f"\caption{{{description}}}] + latex_lines[-1]
73             if only_tabular_environment:
74                 start = 0
75                 end = len(latex_lines)
76                 for i, v in enumerate(latex_lines):
77                     if 'begin{adjustbox}' in v:
78                         start = i
79                     if 'end{adjustbox}' in v:
80                         end = i
81                 latex_lines = latex_lines[start:end + 1]
82             latex_string = '\n'.join(latex_lines)
83             with open(p, 'w') as f:
84                 f.write(latex_string)
85
86     def put_columns_last(self, columns=[]):
87         self.dataframe = self.dataframe[[c for c in self.dataframe.columns if c not in columns] + columns]

```

B.10.1.3 util -> data -> dataset_container.py (code)

```

1 from util.data import ecg_datasets2
2 import os
3
4
5 class DatasetContainer():
6     def __init__(self, data_dirs: list, window_size=4500, pad_to_size=4500, return_labels=True,
7                  normalize_fn=ecg_datasets2.normalize_minmax_scaling):
8         self.databases = {}
9         for dir in data_dirs:
10             name = os.path.split(dir)[1]
11             count = 0
12             while name + str(count) in self.databases: count += 1
13             name = name + str(count)
14             self.databases[name] = ecg_datasets2.ECGChallengeDatasetBaseline(dir, window_size, pad_to_size,
15                                     return_labels, normalize_fn)
16
17             self.georgia_challenge = ecg_datasets2.ECGChallengeDatasetBaseline(
18                 '/media/julian/data/data/ECG/georgia_challenge/',
19                 window_size=4500, pad_to_size=4500, return_labels=True,
20                 normalize_fn=ecg_datasets2.normalize_minmax_scaling)
21             self.cpsc_challenge_train = ecg_datasets2.ECGChallengeDatasetBaseline(
22                 '/media/julian/data/data/ECG/cps2018_challenge/',
23                 window_size=4500, pad_to_size=4500, return_labels=True,
24                 normalize_fn=ecg_datasets2.normalize_minmax_scaling)
25             self.cpsc_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/china_challenge',

```

```

26                                         window_size=4500,
27                                         pad_to_size=4500, return_labels=True,
28                                         normalize_fn=ecg_datasets2.
29                                         normalize_minmax_scaling)
30                                         self.ptbxl_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/ptbxl_challenge',
31                                         ,
32                                         window_size=4500,
33                                         pad_to_size=4500, return_labels=True,
34                                         normalize_fn=ecg_datasets2.
35                                         normalize_minmax_scaling)
36                                         self.nature = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/nature_database',
37                                         window_size=4500,
38                                         pad_to_size=4500, return_labels=True,
39                                         normalize_fn=ecg_datasets2.normalize_minmax_scaling)

```

B.10.1.4 util -> data -> `ecg_data.py` (code)

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[11]:
5
6  import os
7  from collections import defaultdict
8
9  import h5py
10 import numpy as np
11 import wfdb
12 # In[18]:
13 from sklearn.preprocessing import normalize, OneHotEncoder
14
15
16 # ### Read MIT format .dat ecg data files and .hea headers
17
18
19 class ECGData():
20     def __init__(self, base_directory='/home/julian/Datasets/ptb-diagnostic-ecg-database-1.0.0/'):
21         self.BASE_DIR = base_directory
22         self.label_mappings = {}
23         self.default_key_function_dict = defaultdict(lambda y: (lambda x: True, 2))
24         # self.default_key_function_dict['age'] = lambda x: int(x) > 50 if x.isnumeric() else False, 2 # age $TODO:
25         more_onehots
26         # self.default_key_function_dict['smoker'] = lambda x: x == 'yes', 2
27         self.default_key_function_dict['reason for admission'] = lambda x: self.filter_comment('reason for admission',
28                                         x), 10
28         self.label_encoder = None
29         self.pca = None
30
31     def search_files(self, file_endings=None):
32         root = self.BASE_DIR
33         if file_endings is None:
34             file_endings = ['.dat', '.hea', '.xyz']
35         record_files = []
36         with open(os.path.join(root, 'RECORDS')) as recs:
37             record_files = recs.read().splitlines()
38         print(len(record_files), 'record files found in ', root)
39         return record_files
40
41     def read_signal(self, record_path, physical=True):
42         record = wfdb.rdrecord(record_path, physical=physical)
43         if physical:
44             data = record.p_signal
45         else:
46             data = record.d_signal
47         return data
48
49     def read_header(self, record_path):
50         record = wfdb.rdheader(record_path)
51         return record.comments
52
53     def parse_comment_dict(self, wfdb_comment):
54         comment_map = {}
55         for c in wfdb_comment:
56             e = c.lower().split(';')
57             comment_map[e[0]] = e[1].strip()
58         return comment_map
59
60     def filter_comment(self, key, comment_string):
61         c = comment_string
62         if key == 'reason for admission':
63             if 'cardiomyopathy' in c or 'heart failure' in c:
64                 return 'cardiomyopathy'
65             elif 'n/a' in c or 'palpitation' in c:
66                 return 'miscellaneous'
67             elif 'angina' in c:
68                 return 'angina'

```

```

69     else:
70         print('given key not found', key)
71     return comment_string
72
73 def onehot_comment(self, wfdb_comment, terms: list = [], key_function_dict: dict = None):
74     values = []
75     column_names = []
76     if terms:
77         terms_encoded = False
78         for i, term in enumerate(terms):
79             column_names.append(term)
80             for c in wfdb_comment:
81                 if term in c:
82                     terms_encoded = True
83                     break
84             values.append(terms_encoded)
85     comment_dict = self.parse_comment_dict(wfdb_comment)
86     for key, (func, n) in key_function_dict.items():
87         if key in comment_dict:
88             value = func(comment_dict[key])
89             values.append(value)
90             column_names.append(key)
91     print('values', values)
92     return self.label_encoder.transform([values]).toarray(), column_names
93
94 def init_label_encoder(self, files, terms, key_function_dict: dict = None):
95     self.label_encoder = OneHotEncoder()
96     X = []
97     for i, file in enumerate(files):
98         absolute = os.path.join(self.BASE_DIR, file)
99         wfdb_comment = self.read_header(absolute)
100        comment_dict = self.parse_comment_dict(wfdb_comment)
101        temp = [i % 2 == 0] * len(terms) # Show possible values
102        for key, (func, n) in key_function_dict.items():
103            if key in comment_dict:
104                value = func(comment_dict[key])
105                temp.append(value)
106        X.append(temp)
107    print(X)
108
109    self.label_encoder.fit(X)
110    print(self.label_encoder.categories_)
111
112 def read_all_files(self, record_path_list):
113     file_data = []
114     for f in record_path_list:
115         d = self.read_signal(os.path.join(self.BASE_DIR, f))
116         file_data.append(d)
117     return file_data
118
119 def partition_data(self, data, window_size=3000, overlap=0.5, store_in_array=True, align_right=True,
120                     verbose=True): # maybe allow non float overlap too
121     samples, channels = data.shape
122     if samples < window_size:
123         print('too few samples (%d) to support window size of %d' % (samples, window_size))
124         return None
125     if verbose: print('Input data has shape:', data.shape)
126     shift = window_size * (1 - overlap)
127     offset = int(samples % shift)
128     if align_right:
129         used_data = data[offset:]
130     else:
131         used_data = data[:-offset]
132     samples, _ = used_data.shape
133     if verbose:
134         print('The window of size %d will be shifted by %f. The total data used is %d' % (
135             window_size, shift, samples))
136     partitioned = np.empty((int(samples / shift) - 1, window_size, channels))
137     if verbose: print('The partitioned data now has shape:', partitioned.shape)
138     for i in range(len(partitioned)):
139         index = int(i * shift)
140         partitioned[i, :, :] = used_data[index:index + window_size, :]
141     return partitioned
142
143 def generate_context_task(self, partitioned_data, N, observations=5, predictions=3, verbose=True,
144                           shuffle_all=False): # maybe use shuffle?
145     # generate N-1 negative samples and 1 positive sample:
146     windows, samples, channels = partitioned_data.shape
147     positive_samples_x = []
148     positive_samples_y = []
149     negative_samples_x = []
150     negative_samples_y = []
151     for i in range(0, windows - observations - predictions):
152         i_cur = i + observations
153         positive_samples_x += [partitioned_data[i:i_cur, :, :]]
154         positive_samples_y += [partitioned_data[i_cur:i_cur + predictions]]
155         possible_choices = list(range(i)) + list(range(i_cur + predictions, windows))
156         for _ in range(N - 1):
157             choices = np.random.choice(possible_choices, predictions,
158                                       replace=False) # advanced use p param for different probabilities

```

```

159     negative_samples_x += [partitioned_data[i:i_cur, :, :]]
160     negative_samples_y += [partitioned_data[choices, :, :]]
161
162     return positive_samples_x, positive_samples_y, negative_samples_x, negative_samples_y
163
164 def convert_dat_to_h5_partitioned(self, storage_path, dat_file_paths, window_size=3000, overlap=0.5,
165                                 align_right=True, overwrite=True, verbose=True):
166     if not os.path.exists(storage_path):
167         os.makedirs(storage_path)
168     for f in dat_file_paths:
169         data = self.read_signal(os.path.join(self.BASE_DIR, f))
170         print(f)
171         partitioned = self.partition_data(data, window_size, overlap, True, align_right, verbose)
172         target = os.path.join(storage_path, f.replace('/', '-') + '.h5')
173         with h5py.File(target, 'w') as wf:
174             wf['data'] = partitioned
175             wf.flush()
176             if verbose: print(target, 'file created and written. %d windows saved.' % (len(partitioned)))
177
178 def convert_dat_to_h5(self, storage_path, dat_file_paths, channels=None, normalize_data=False, pca_components=0,
179                      use_labels=False, use_header=False, terms=None, key_function_dict=None, verbose=True):
180     if not os.path.exists(storage_path):
181         os.makedirs(storage_path)
182     for f in dat_file_paths:
183         absolute = os.path.join(self.BASE_DIR, f)
184         if not channels:
185             data = self.read_signal(absolute)
186         else:
187             data = self.read_signal(absolute)[:, channels]
188         if normalize_data:
189             data = normalize(data, norm='l2')
190         if pca_components > 0:
191             print('performing pca with:', data.shape)
192             cov = np.cov(data.T)
193             eig, ev = np.linalg.eigh(cov)
194             evecs = ev[::-1][:, 0:pca_components] # order is ascending, so descend, then take first two
195             data = np.dot(data, evecs)
196             print('output shape pca:', data.shape)
197         target = os.path.join(storage_path, f.replace('/', '-') + '.h5')
198         with h5py.File(target, 'w') as wf:
199             wf['data'] = data # TODO: Make this a parameter (drop last 3 channels)
200             wf.flush()
201             if use_header:
202                 comments = self.read_header(absolute)
203                 onehot, column_names = self.onehot_comment(comments, terms, key_function_dict)
204                 wf['label'] = onehot
205                 wf['label'].attrs.create('names', column_names)
206                 wf.flush()
207             if verbose: print(target, 'file created and written')
208
209 def convert_dat_to_h5_context_task(self, storage_path, dat_file_paths, window_size=3000, overlap=0.5, time_in=5,
210                                    time_out=3, align_right=True, verbose=True):
211     if not os.path.exists(storage_path):
212         os.makedirs(storage_path)
213     for f in dat_file_paths:
214         data = self.read_signal(os.path.join(self.BASE_DIR, f))
215         print(f)
216         partitioned = self.partition_data(data, window_size, overlap, True, align_right, verbose)
217         p_x, p_y, n_x, n_y = self.generate_context_task(partitioned, 100, time_in, time_out, verbose, False)
218
219         target = os.path.join(storage_path, f.replace('/', '-') + '.h5')
220         with h5py.File(target, 'w') as wf:
221             wf['windows'] = partitioned
222             wf.flush()
223             if verbose: print(target, 'file created and written. %d windows saved.' % (len(partitioned)))

```

B.10.1.5 util -> data -> ecg_datasets2.py (code)

```

1 import os
2 from random import shuffle
3
4 import h5py
5 import numpy as np
6 import torch
7 # from torch._six import container_abcs, string_classes, int_classes
8 from torch.utils.data import IterableDataset
9
10 from torch.utils.data._utils.collate import np_str_obj_array_pattern, default_collate_err_msg_format
11
12 from external import helper_code
13 from util.utility import timestamp
14
15
16 class ECGDataset(IterableDataset):
17     def __init__(self, BASE_DIR, window_size, n_windows, files=None, channels=None, use_labels=False,
18                  preload_windows=0): # TODO: option to load into ram

```

```

19     super(ECGDataset).__init__()
20     self.BASE_DIR = BASE_DIR
21     self.n_windows = n_windows
22     self.window_size = window_size
23     if files:
24         self.files = files
25     else:
26         self.files = self.search_files()
27     self.print_file_attributes()
28     self.channels = channels
29     self.total_length = 1 # Trying a weird approach (calculated in __iter__)
30     self.use_labels = use_labels
31     self.preload_windows = preload_windows
32
33     def __iter__(self):
34         # multiple workers?
35         file_index = 0
36         self.total_length = 0
37         while file_index < len(self.files):
38             self.current_file = self.files[file_index]
39             with h5py.File(self.current_file, 'r') as f:
40                 index = 0
41                 offset = np.random.randint(0, self.window_size) # Random offset
42                 if 'data' in f.keys():
43                     data = f.get('data') # Make sure this exists in your dataset
44                 if 'labels' in f.keys():
45                     labels = f.get('labels')
46                 if data is None:
47                     print(self.current_file, "'data' table is None.")
48                 else:
49                     if not self.use_labels:
50                         while (index + 1) * self.window_size * self.n_windows + offset <= len(data):
51                             preloaded = data[index * self.window_size * self.n_windows + offset: (
52                                         self.preload_windows) * self.window_size * self.n_windows + offset].copy() # maybe copy
53                             preload_index = 0
54                             while (preload_index + 1) * self.window_size * self.n_windows + offset <= len(preloaded):
55                                 if self.channels:
56                                     yield np.swapaxes(preloaded[
57                                         preload_index * self.window_size * self.n_windows + offset: (
58                                         preload_index + 1) * self.window_size * self.n_windows + offset,
59                                         self.channels].reshape((self.n_windows, self.window_size, -1)),
60                                         1,
61                                         2) # TODO: pad data in between with 0?
62                                 else:
63                                     yield np.swapaxes(preloaded[
64                                         preload_index * self.window_size * self.n_windows + offset: (
65                                         preload_index + 1) * self.window_size * self.n_windows + offset,
66                                         :].reshape((self.n_windows, self.window_size, -1)), 1, 2)
67                                     preload_index += 1
68                                     index += 1
69                     else:
70                         while (index + 1) * self.window_size * self.n_windows + offset <= len(data):
71                             preloaded = data[index * self.window_size * self.n_windows + offset: (
72                                         self.preload_windows) * self.window_size * self.n_windows + offset].copy()
73                             preload_index = 0
74                             while (preload_index + 1) * self.window_size * self.n_windows + offset <= len(preloaded):
75                                 if self.channels:
76                                     yield np.swapaxes(preloaded[
77                                         preload_index * self.window_size * self.n_windows + offset: (
78                                         preload_index + 1) * self.window_size * self.n_windows + offset,
79                                         self.channels].reshape((self.n_windows, self.window_size, -1)),
80                                         1,
81                                         2) # TODO: pad data in between with 0?
82                                 else:
83                                     yield np.swapaxes(preloaded[
84                                         preload_index * self.window_size * self.n_windows + offset: (
85                                         preload_index + 1) * self.window_size * self.n_windows + offset,
86                                         :].reshape((self.n_windows, self.window_size, -1)), 1, 2)
87                                     preload_index += 1
88                                     index += 1
89                                     del preloaded
90                                     self.total_length += len(data)
91                                     file_index += 1
92     def __len__(self):
93         return self.total_length
94
95     def search_files(self):
96         record_files = []
97         file_endings = ('.h5')
98         for root, dirs, files in os.walk(self.BASE_DIR):
99             for file in files:
100                 if file.endswith(file_endings):

```

```

101         record_files.append(os.path.join(root, file))
102     print(len(record_files), 'record files found in ', self.BASE_DIR)
103     return record_files
104
105 def print_file_attributes(self):
106     with h5py.File(self.files[0], 'r') as f: # only look at first file
107         print('Keys contained in dataset:', f.keys())
108         for k in f.keys():
109             data = f.get(k)
110             print('Data with key [%s] has shape:' % k, data.shape)
111
112
113 class ECGMultipleDatasets(torch.utils.data.IterableDataset):
114     def __init__(self, torch_datasets):
115         super(ECGMultipleDatasets).__init__()
116         self.torch_datasets = torch_datasets
117
118     def __iter__(self): # desired shape: (batches, windows
119         # TODO: how long? Also: custom batching?
120         dataset_iterators = [iter(t) for t in self.torch_datasets]
121         available = list(range(len(self.torch_datasets)))
122         available2 = list(range(len(self.torch_datasets)))
123         while available:
124             for i in available:
125                 try:
126                     yield next(dataset_iterators[i])
127                 except StopIteration:
128                     available2.remove(i)
129             available = available2.copy()
130
131     def __len__(self):
132         return sum(map(len, self.torch_datasets)) # Whatever
133
134     def print_file_attributes(self):
135         for dataset in self.torch_datasets:
136             with h5py.File(dataset.files[0], 'r') as f: # only look at first file
137                 print('Keys contained in dataset:', f.keys())
138                 for k in f.keys():
139                     data = f.get(k)
140                     print('Data with key [%s] has shape:' % k, data.shape)
141
142
143 class ECGDatasetBatching(ECGDataset): # TODO: make preload available?
144     def __init__(self, BASE_DIR, window_size, n_windows, window_gap=0, files=None, channels=None, use_labels=False,
145                  use_random_offset=False, preload_windows=0, batch_size=1,
146                  data_max_len=10000): # TODO: option to load into ram
147         super(ECGDatasetBatching, self).__init__(BASE_DIR, window_size, n_windows, files, channels, use_labels,
148                                                 preload_windows)
149         self.batch_size = batch_size
150         self.window_gap = window_gap
151         self.use_random_offsets = use_random_offset
152         self.data_max_len = data_max_len
153
154     def __iter__(self):
155         # multiple workers?
156         file_index = 0
157         self.total_length = 0
158         available_files = list(range(len(self.files)))
159         if available_files:
160             strike = 0
161             selected_files = np.random.choice(available_files, self.batch_size, replace=False)
162             opened_files = [h5py.File(self.files[f_ind], 'r') for f_ind in selected_files] # open all selected files
163             data_indices = np.zeros(self.batch_size, dtype=int)
164             half_heartrate = 800 // 2
165             data_offsets = np.random.randint(0, self.data_max_len - self.window_size * self.n_windows, self.batch_size
166                                         , dtype=int)
166
167             for sel in selected_files:
168                 available_files.remove(sel) # remove selected for future draw
169             while opened_files:
170                 for i, f in enumerate(opened_files):
171                     data_index = data_indices[i]
172                     offset = data_offsets[i]
173                     if 'data' in f.keys():
174                         data = f.get('data') # Make sure this exists in your dataset
175                     if 'multilabel' in f.keys():
176                         labels = f.get('multilabel')
177                     elif 'label' in f.keys():
178                         labels = f.get('label')
179                     if (data_index + 1) * self.window_size * self.n_windows + offset <= len(data): # + self.
180                         window_gap
181                         # print('using:', i, opened_files[i])
182                         if not self.use_labels:
183                             if self.channels:
184                                 yield np.swapaxes(data[data_index * self.window_size * self.n_windows + offset: (
185                                     data_index + 1) * self.window_size * self.n_windows + offset,
186                                     self.channels].reshape((self.n_windows, self.window_size, -1)), 1,
187                                     2) # TODO: pad data in between with 0?
188                         else:
189

```

```

188         yield np.swapaxes(data[data_index * self.window_size * self.n_windows + offset: (
189             data_index + 1) * self.window_size * self.n_windows + offset,
190                                         :].reshape((self.n_windows, self.window_size, -1)), 1, 2)
191     else:
192         if self.channels:
193             yield np.swapaxes(data[data_index * self.window_size * self.n_windows + offset: (
194                 data_index + 1) * self.window_size * self.n_windows + offset,
195                                         self.channels].reshape((self.n_windows, self.window_size, -1)), 1,
196                                         2), np.array(
197                 (data_index + 2) * self.window_size * self.n_windows + offset > len(data)), labels
198 [
199     :
200     else:
201         yield np.swapaxes(data[data_index * self.window_size * self.n_windows + offset: (
202             data_index + 1) * self.window_size * self.n_windows + offset,
203                                         :].reshape((self.n_windows, self.window_size, -1)), 1, 2), np.array(
204                 (data_index + 2) * self.window_size * self.n_windows + offset > len(data)), labels
205 [
206     :
207     data_indices[i] += 1
208     self.total_length += self.window_size * self.n_windows
209     # Replace used up files:
210     if (data_index + 2) * self.window_size * self.n_windows + offset > len(
211         data): # Remove this from opened. TODO: what about unused data?
212         if available_files:
213             # print('opening new for', i)
214             opened_files[i] = h5py.File(self.files[available_files.pop()], 'r')
215             f.close() # closing here
216         else:
217             strike += 1
218             data_indices[i] = 0
219             data_offsets[i] = np.random.randint(0, self.window_size - 1)
220             if strike >= self.batch_size: # Recycle until all files have been fully used
221                 return
222
223 class ECGLabelDataset(torch.utils.data.IterableDataset):
224     def __init__(self, BASE_DIR, window_size, n_windows, files=None):
225         super(ECGLabelDataset).__init__()
226         self.BASE_DIR = BASE_DIR
227         if files:
228             self.files = files
229         else:
230             self.files = self.search_files()
231         self.window_size = window_size
232         self.n_windows = n_windows
233         self.print_file_attributes()
234
235     def __iter__(self):
236         # multiple workers?
237         file_index = 0
238         while file_index < len(self.files):
239             self.current_file = self.files[file_index]
240             with h5py.File(self.current_file, 'r') as f:
241                 index = 0
242                 offset = np.random.randint(self.window_size) # Use a random offset
243                 data = f.get('data')
244                 labels = f.get('labels') # Uses data and labels group in .h5py file. make sure those exist
245                 while (index + 1) * self.window_size * self.n_windows <= len(data):
246                     # tmp_data = data[index*self.window_size*self.n_windows:(index+1)*self.window_size*self.n_windows]
247                     yield np.swapaxes(data[index * self.window_size * self.n_windows + offset: (
248                         index + 1) * self.window_size * self.n_windows + offset].reshape(
249                                         (self.n_windows, self.window_size, -1)), 1, 2), \
250                         np.swapaxes(labels[index * self.window_size * self.n_windows + offset: (
251                             index + 1) * self.window_size * self.n_windows + offset].reshape(
252                                         (self.n_windows, self.window_size, -1)), 1,
253                                         2) # labels[self.n_windows:(index+1)*self.window_size*self.n_windows] #TODO: Try
254             returning as list
255             index += 1
256             file_index += 1
257
258     def search_files(self):
259         record_files = []
260         file_endings = ('.h5')
261         for root, dirs, files in os.walk(self.BASE_DIR):
262             for file in files:
263                 if file.endswith(file_endings):
264                     record_files.append(os.path.join(root, file))
265         print(len(record_files), 'record files found in ', self.BASE_DIR)
266         return record_files
267
268     def print_file_attributes(self):
269         print('Printing attributes for dataset at dir:', self.BASE_DIR)
270         with h5py.File(self.files[0], 'r') as f: # only look at first file
271             print('looking at file', f)

```

```

270     print('Keys contained in dataset:', f.keys())
271     for k in f.keys():
272         data = f.get(k)
273         print('Data with key [%s] has shape:' % k, data.shape)
274
275     labels = f.get('labels') # Uses data and labels group in .h5py file. make sure those exist
276     print(np.min(labels), np.max(labels))
277
278
279 class ECGDatasetBaseline(torch.utils.data.IterableDataset):
280
281     def __init__(self, BASE_DIR, window_size, files=None):
282         super(ECGDataset).__init__()
283         self.BASE_DIR = BASE_DIR
284         self.window_size = window_size
285         if files:
286             self.files = files
287         else:
288             self.files = self.search_files()
289         self.print_file_attributes()
290
291     def __iter__(self):
292         file_index = 0
293         shuffle(self.files)
294         while file_index < len(self.files):
295             self.current_file = self.files[file_index]
296             with h5py.File(self.current_file, 'r') as f:
297                 index = 0
298                 data = f.get('data')
299                 labels = f.get('label')
300                 offset = np.random.randint(len(data) - self.window_size) # Random offset
301                 if not data is None and not labels is None:
302                     yield data[offset:self.window_size + offset, :], labels[:, :]
303             file_index += 1
304
305     def __len__(self):
306         return 1 # Whatever
307
308     def search_files(self):
309         record_files = []
310         file_endings = ('.h5')
311         for root, dirs, files in os.walk(self.BASE_DIR):
312             for file in files:
313                 if file.endswith(file_endings):
314                     record_files.append(os.path.join(root, file))
315         print(len(record_files), 'record files found in ', self.BASE_DIR)
316         return record_files
317
318     def print_file_attributes(self):
319         with h5py.File(self.files[0], 'r') as f: # only look at first file
320             print('Keys contained in dataset:', f.keys())
321             for k in f.keys():
322                 data = f.get(k)
323                 print('Data with key [%s] has shape:' % k, data.shape)
324
325
326 class ECGDatasetBaselineMulti(torch.utils.data.IterableDataset):
327
328     def __init__(self, BASE_DIR, window_size, files=None):
329         super(ECGDataset).__init__()
330         self.BASE_DIR = BASE_DIR
331         self.window_size = window_size
332         if files:
333             self.files = files
334         else:
335             self.files = self.search_files()
336         self.print_file_attributes()
337
338     def __iter__(self):
339         file_index = 0
340         shuffle(self.files)
341         while file_index < len(self.files):
342             self.current_file = self.files[file_index]
343             with h5py.File(self.current_file, 'r') as f:
344                 index = 0
345                 data = f.get('data')
346                 labels = f.get('multilabel')
347                 offset = np.random.randint(len(data) - self.window_size) # Random offset
348                 if not data is None and not labels is None:
349                     yield data[offset:self.window_size + offset, :], labels[:, :]
350             file_index += 1
351
352     def __len__(self):
353         return 1 # Whatever
354
355     def search_files(self):
356         record_files = []
357         file_endings = ('.h5')
358         for root, dirs, files in os.walk(self.BASE_DIR):
359             for file in files:

```

```

360         if file.endswith(file_endings):
361             record_files.append(os.path.join(root, file))
362     print(len(record_files), 'record files found in ', self.BASE_DIR)
363     return record_files
364
365 def print_file_attributes(self):
366     with h5py.File(self.files[0], 'r') as f: # only look at first file
367         print('Keys contained in dataset:', f.keys())
368         for k in f.keys():
369             data = f.get(k)
370             print('Data with key [%s] has shape:' % k, data.shape)
371
372 class ECGChallengeDatasetBatching(torch.utils.data.IterableDataset):
373     def __init__(self, BASE_DIR, window_size, n_windows, files=None, channels=None, use_labels=False,
374                  use_random_offset=False, batch_size=1, classes=None):
375         super(ECGDataset).__init__()
376         self.BASE_DIR = BASE_DIR
377         self.n_windows = n_windows
378         self.window_size = window_size
379         if files:
380             self.files = files
381         else:
382             self.files = self.search_files()
383         if classes is None:
384             self.classes = helper_code.get_classes(self.files)
385         else:
386             self.classes = classes
387         self.print_file_attributes()
388         self.channels = channels
389         self.total_length = 1 # Trying a weird approach (calculated in __iter__)
390         self.use_labels = use_labels
391         self.use_random_offsets = use_random_offset
392         self.batch_size = batch_size
393
394     def __iter__(self):
395         available_files = list(range(len(self.files)))
396         if available_files:
397             strike = 0
398             selected_files = np.random.choice(available_files, self.batch_size, replace=False)
399             opened_file_data = [self._read_recording_file(self.files[f_ind]) for f_ind in
400                                 selected_files] # open all selected files
401             if self.use_labels:
402                 opened_file_labels = [self._read_header_labels(self.files[f_ind]) for f_ind in selected_files]
403             data_indices = np.zeros(self.batch_size, dtype=int)
404             data_offsets = [
405                 np.random.randint(0, len(opened_file_data[i]) - self.window_size * self.n_windows, dtype=int) for i in
406                 range(self.batch_size)]
407             for sel in selected_files:
408                 available_files.remove(sel) # remove selected for future draw
409             while opened_file_data:
410                 for i, f in enumerate(opened_file_data):
411                     data_index = data_indices[i]
412                     offset = data_offsets[i]
413                     data = opened_file_data[i]
414                     if self.use_labels:
415                         labels = opened_file_labels[i]
416                         if (data_index + 1) * self.window_size * self.n_windows + offset <= len(data):
417                             if not self.use_labels:
418                                 if self.channels:
419                                     yield np.swapaxes(data[data_index * self.window_size * self.n_windows + offset: (
420                                         data_index + 1) * self.window_size * self.n_windows + offset,
421                                         self.channels].reshape((self.n_windows, self.window_size, -1)), 1,
422                                         2)
423                                     else:
424                                         yield np.swapaxes(data[data_index * self.window_size * self.n_windows + offset: (
425                                         data_index + 1) * self.window_size * self.n_windows + offset,
426                                         :].reshape((self.n_windows, self.window_size, -1)), 1, 2)
427                                     else:
428                                         if self.channels:
429                                             yield np.swapaxes(data[data_index * self.window_size * self.n_windows + offset: (
430                                         data_index + 1) * self.window_size * self.n_windows + offset,
431                                         self.channels].reshape((self.n_windows, self.window_size, -1)), 1,
432                                         2), np.array(
433                                         (data_index + 2) * self.window_size * self.n_windows + offset > len(data)), labels
434
435             [
436                 data_indices[i] += 1
437                 # Replace used up files:
438
439                 data_indices[i] += 1
440                 # Replace used up files:
441
442                 data_indices[i] += 1
443                 # Replace used up files:
444
445                 data_indices[i] += 1
446                 # Replace used up files:
447
448                 data_indices[i] += 1
449                 # Replace used up files:
450
451                 data_indices[i] += 1
452                 # Replace used up files:
453
454                 data_indices[i] += 1
455                 # Replace used up files:
456
457                 data_indices[i] += 1
458                 # Replace used up files:
459
460                 data_indices[i] += 1
461                 # Replace used up files:
462
463                 data_indices[i] += 1
464                 # Replace used up files:
465
466                 data_indices[i] += 1
467                 # Replace used up files:
468
469                 data_indices[i] += 1
470                 # Replace used up files:
471
472                 data_indices[i] += 1
473                 # Replace used up files:
474
475                 data_indices[i] += 1
476                 # Replace used up files:
477
478                 data_indices[i] += 1
479                 # Replace used up files:
480
481                 data_indices[i] += 1
482                 # Replace used up files:
483
484                 data_indices[i] += 1
485                 # Replace used up files:
486
487                 data_indices[i] += 1
488                 # Replace used up files:
489
490                 data_indices[i] += 1
491                 # Replace used up files:
492
493                 data_indices[i] += 1
494                 # Replace used up files:
495
496                 data_indices[i] += 1
497                 # Replace used up files:
498
499                 data_indices[i] += 1
500                 # Replace used up files:
501
502                 data_indices[i] += 1
503                 # Replace used up files:
504
505                 data_indices[i] += 1
506                 # Replace used up files:
507
508                 data_indices[i] += 1
509                 # Replace used up files:
510
511                 data_indices[i] += 1
512                 # Replace used up files:
513
514                 data_indices[i] += 1
515                 # Replace used up files:
516
517                 data_indices[i] += 1
518                 # Replace used up files:
519
520                 data_indices[i] += 1
521                 # Replace used up files:
522
523                 data_indices[i] += 1
524                 # Replace used up files:
525
526                 data_indices[i] += 1
527                 # Replace used up files:
528
529                 data_indices[i] += 1
530                 # Replace used up files:
531
532                 data_indices[i] += 1
533                 # Replace used up files:
534
535                 data_indices[i] += 1
536                 # Replace used up files:
537
538                 data_indices[i] += 1
539                 # Replace used up files:
540
541                 data_indices[i] += 1
542                 # Replace used up files:
543
544                 data_indices[i] += 1
545                 # Replace used up files:
546
547                 data_indices[i] += 1
548                 # Replace used up files:
549
550                 data_indices[i] += 1
551                 # Replace used up files:
552
553                 data_indices[i] += 1
554                 # Replace used up files:
555
556                 data_indices[i] += 1
557                 # Replace used up files:
558
559                 data_indices[i] += 1
560                 # Replace used up files:
561
562                 data_indices[i] += 1
563                 # Replace used up files:
564
565                 data_indices[i] += 1
566                 # Replace used up files:
567
568                 data_indices[i] += 1
569                 # Replace used up files:
570
571                 data_indices[i] += 1
572                 # Replace used up files:
573
574                 data_indices[i] += 1
575                 # Replace used up files:
576
577                 data_indices[i] += 1
578                 # Replace used up files:
579
580                 data_indices[i] += 1
581                 # Replace used up files:
582
583                 data_indices[i] += 1
584                 # Replace used up files:
585
586                 data_indices[i] += 1
587                 # Replace used up files:
588
589                 data_indices[i] += 1
590                 # Replace used up files:
591
592                 data_indices[i] += 1
593                 # Replace used up files:
594
595                 data_indices[i] += 1
596                 # Replace used up files:
597
598                 data_indices[i] += 1
599                 # Replace used up files:
600
601                 data_indices[i] += 1
602                 # Replace used up files:
603
604                 data_indices[i] += 1
605                 # Replace used up files:
606
607                 data_indices[i] += 1
608                 # Replace used up files:
609
610                 data_indices[i] += 1
611                 # Replace used up files:
612
613                 data_indices[i] += 1
614                 # Replace used up files:
615
616                 data_indices[i] += 1
617                 # Replace used up files:
618
619                 data_indices[i] += 1
620                 # Replace used up files:
621
622                 data_indices[i] += 1
623                 # Replace used up files:
624
625                 data_indices[i] += 1
626                 # Replace used up files:
627
628                 data_indices[i] += 1
629                 # Replace used up files:
630
631                 data_indices[i] += 1
632                 # Replace used up files:
633
634                 data_indices[i] += 1
635                 # Replace used up files:
636
637                 data_indices[i] += 1
638                 # Replace used up files:
639
640                 data_indices[i] += 1
641                 # Replace used up files:
642
643                 data_indices[i] += 1
644                 # Replace used up files:
645
646                 data_indices[i] += 1
647                 # Replace used up files:
648
649                 data_indices[i] += 1
650                 # Replace used up files:
651
652                 data_indices[i] += 1
653                 # Replace used up files:
654
655                 data_indices[i] += 1
656                 # Replace used up files:
657
658                 data_indices[i] += 1
659                 # Replace used up files:
660
661                 data_indices[i] += 1
662                 # Replace used up files:
663
664                 data_indices[i] += 1
665                 # Replace used up files:
666
667                 data_indices[i] += 1
668                 # Replace used up files:
669
670                 data_indices[i] += 1
671                 # Replace used up files:
672
673                 data_indices[i] += 1
674                 # Replace used up files:
675
676                 data_indices[i] += 1
677                 # Replace used up files:
678
679                 data_indices[i] += 1
680                 # Replace used up files:
681
682                 data_indices[i] += 1
683                 # Replace used up files:
684
685                 data_indices[i] += 1
686                 # Replace used up files:
687
688                 data_indices[i] += 1
689                 # Replace used up files:
690
691                 data_indices[i] += 1
692                 # Replace used up files:
693
694                 data_indices[i] += 1
695                 # Replace used up files:
696
697                 data_indices[i] += 1
698                 # Replace used up files:
699
699             ]

```

```

444     if (data_index + 2) * self.window_size * self.n_windows + offset > len(data):
445         # Remove this from opened.
446     if available_files:
447         new_f = self.files[available_files.pop()]
448         opened_file_data[i] = self._read_recording_file(new_f) # Open new available file
449         if self.use_labels:
450             opened_file_labels[i] = self._read_header_labels(new_f)
451         else: # if no available file use old and count a strike
452             strike += 1
453         data_indices[i] = 0 # reset data indice
454         data_offsets[i] = np.random.randint(0, len(
455             opened_file_data[i]) - self.window_size * self.n_windows) # reset offset randomly
456         if strike >= self.batch_size: # Recycle until all files have been fully used
457             return
458
459     def _read_recording_file(self, path_without_ext):
460         fp = path_without_ext + '.mat'
461         return helper_code.load_recording(fp, key='val').transpose()
462
463     def _read_header_file(self, path_without_ext):
464         fp = path_without_ext + '.hea'
465         return helper_code.load_header(fp)
466
467     def _read_header_labels(self, path_without_ext):
468         header = self._read_header_file(path_without_ext)
469         return helper_code.encode_header_labels(header, self.classes)
470
471     def __len__(self):
472         return self.total_length
473
474     def search_files(self):
475         headers, records = helper_code.find_challenge_files(self.BASE_DIR)
476         print(len(records), 'record files found in ', self.BASE_DIR)
477         return list(map(lambda x: os.path.splitext(x)[0], records)) # remove extension
478
479     def print_file_attributes(self):
480         f = self.files[0]
481         print('Information for file', f)
482         data = self._read_recording_file(f)
483         print('Data has shape:', data.shape)
484         header = self._read_header_file(f)
485         print('Header is:', header, end='#####\n')
486         print('Classes found in data folder:', self.classes)
487         labels = self._read_header_labels(f)
488         print('Labels have shape', labels.shape)
489
490
491 class ECGChallengeDatasetBaseline(torch.utils.data.IterableDataset):
492     def __init__(self, BASE_DIR, window_size, pad_to_size=None, files=None, channels=None, return_labels=False,
493                  return_filename=False, classes=None, normalize_fn=None, verbose=False, randomize_order=True):
494         super(ECGdataset).__init__()
495         self.BASE_DIR = BASE_DIR
496         self.window_size = window_size
497         self.pad_to_size = pad_to_size or window_size
498         self.files = files or self.search_files()
499         self.classes = classes or helper_code.get_classes(self.files)
500         if verbose:
501             self.print_file_attributes()
502         self.channels = channels
503         self.total_length = 1 # Trying a weird approach (calculated in __iter__)
504         self.return_labels = return_labels
505         self.return_filename = return_filename
506         self.normalize_fn = normalize_fn if not normalize_fn is None else lambda x: x
507         self.randomize_order = randomize_order
508
509     def generate_datasets_from_split_file(self, ttsfile='train-test-splits.txt'):
510         splits = load_train_test_split(os.path.join(self.BASE_DIR, ttsfile))
511         return tuple(ECGChallengeDatasetBaseline(self.BASE_DIR, self.window_size, self.pad_to_size, files=s,
512                                                 channels=self.channels, return_labels=self.return_labels,
513                                                 return_filename=self.return_filename, classes=self.classes,
514                                                 normalize_fn=self.normalize_fn, randomize_order=self.randomize_order)
515                                         for s in splits)
516
517     def generate_datasets_from_split_list(self, trainf, valf, testf):
518         splits = [trainf, valf, testf]
519         return tuple(ECGChallengeDatasetBaseline(self.BASE_DIR, self.window_size, self.pad_to_size, files=s,
520                                                 channels=self.channels, return_labels=self.return_labels,
521                                                 return_filename=self.return_filename, classes=self.classes,
522                                                 normalize_fn=self.normalize_fn, randomize_order=self.randomize_order)
523                                         for s in splits)
524
525     def __iter__(self):
526         file_index = 0
527         if self.randomize_order:
528             shuffle(self.files)
529         while file_index < len(self.files):
530             current_file = self.files[file_index]
531             data = self.normalize_fn(self._read_recording_file(current_file))
532             if self.channels:
533                 data = data[:, self.channels]

```

```

534         if self.return_labels:
535             labels = self._read_header_labels(current_file)
536         if len(data) - self.window_size > 0:
537             if self.randomize_order:
538                 offset = np.random.randint(len(data) - self.window_size) # Random offset
539             else:
540                 offset = 0
541             data = data[offset:self.window_size + offset]
542         else:
543             offset = 0
544             data = np.pad(data, ((max(0, self.pad_to_size - min(self.window_size, len(data))), 0), (0, 0)))
545         if not any([self.return_filename, self.return_labels]):
546             yield data
547         else:
548             yield [data] + ([labels] if self.return_labels else [None]) + (
549                 [current_file] if self.return_filename else [])
550
551         file_index += 1
552
553     def _read_recording_file(self, path_without_ext):
554         fp = path_without_ext + '.mat'
555         return helper_code.load_recording(fp, key='val').transpose()
556
557     def _read_header_file(self, path_without_ext):
558         fp = path_without_ext + '.hea'
559         return helper_code.load_header(fp)
560
561     def _read_header_labels(self, path_without_ext, onerror_class='426783006'):
562         header = self._read_header_file(path_without_ext)
563         return helper_code.encode_header_labels(header, self.classes, onerror_class)
564
565     def __len__(self):
566         return self.total_length
567
568     def search_files(self):
569         headers, records = helper_code.find_challenge_files(self.BASE_DIR)
570         print(len(records), 'record files found in ', self.BASE_DIR)
571         return list(map(lambda x: os.path.splitext(x)[0], records)) # remove extension
572
573     def random_train_split(self, train_fraction=0.7, val_fraction=0.2, test_fraction=0.1, save=True,
574                           save_path_overwrite=None, filename_overwrite=None):
575         assert train_fraction + val_fraction + test_fraction <= 1
576         N = len(self.files)
577         shuffle(self.files)
578         train_slice = slice(0, int(train_fraction * N))
579         val_slice = slice(train_slice.stop, train_slice.stop + int(val_fraction * N))
580         test_slice = slice(val_slice.stop, val_slice.stop + int(test_fraction * N))
581         if save:
582             p = save_path_overwrite or self.BASE_DIR
583             fname = filename_overwrite or 'train-test-splits.txt'
584             save_train_test_split(os.path.join(p, fname), self.files[train_slice], self.files[val_slice],
585                                   self.files[test_slice])
586         return self.files[train_slice], self.files[val_slice], self.files[test_slice]
587
588     def random_train_split_with_class_count(self, train_fraction=0.7, val_fraction=0.2, test_fraction=0.1, save=True,
589                                           save_path_overwrite=None, filename_overwrite=None):
590         assert train_fraction + val_fraction + test_fraction <= 1
591         class_buckets = [[] for x in range(len(self.classes))] # Creates classes buckets
592         for i, f in enumerate(self.files):
593             label = self._read_header_labels(f)
594             label_idx = np.argwhere(label).flatten()
595             for l in label_idx: # can return more than one (multi-label)
596                 class_buckets[l].append(f) # Put this file into class l bucket
597         train_files, val_files, test_files = [], [], []
598         sorted_idx = list(sorted(range(len(class_buckets)),
599                               key=lambda x: len(class_buckets[x]))) # make index sorted by class count low->high
600         print(sorted_idx)
601         while len(sorted_idx) > 0:
602             idx = sorted_idx[0]
603             shuffle(class_buckets[idx])
604             b = class_buckets[idx]
605             c_N = len(b)
606             train_slice = slice(0, int(train_fraction * c_N))
607             val_slice = slice(train_slice.stop, train_slice.stop + int(val_fraction * c_N))
608             test_slice = slice(val_slice.stop, val_slice.stop + int(test_fraction * c_N))
609             train_files += b[train_slice]
610             val_files += b[val_slice]
611             test_files += b[test_slice]
612             used_set = set(b[train_slice] + b[val_slice] + b[test_slice])
613             for j in sorted_idx[1:]:
614                 class_buckets[j] = [x for x in class_buckets[j] if
615                                     x not in used_set] # REMOVE THIS FILE FROM ALL OTHER BUCKETS
616             sorted_idx = list(
617                 sorted(sorted_idx[1:], key=lambda x: len(class_buckets[x]))) # sort again (removal may change order)
618
619         train_files = list(set(train_files))
620         val_files = list(set(val_files))
621         test_files = list(set(test_files))
622         if save:
623             p = save_path_overwrite or self.BASE_DIR

```

```

624         fname = filename_overwrite or 'train-test-splits.txt'
625         save_train_test_split(os.path.join(p, fname), train_files, val_files, test_files)
626     return train_files, val_files, test_files
627
628     def random_train_split_with_class_count_mins(self, train_counts, val_counts, test_counts, save=True,
629                                                 save_path_overwrite=None, filename_overwrite=None):
630         class_buckets = [[] for x in range(len(self.classes))] # Creates classes buckets
631         for i, f in enumerate(self.files):
632             label = self._read_header_labels(f)
633             label_idx = np.argwhere(label).flatten()
634             for l in label_idx: # can return more than one (multi-label)
635                 class_buckets[l].append(f) # Put this file into class l bucket
636         file_splits = [[], [], []]
637         for i, split_counts in enumerate([test_counts, val_counts, train_counts]):
638             sorted_class_idxs = np.argsort(split_counts)
639             sorted_class_idxs = sorted_class_idxs[np.argwhere(split_counts[sorted_class_idxs] > 0)].ravel()
640             while (len(sorted_class_idxs) > 0): # TODO:add bad counter
641                 for class_idx in sorted_class_idxs:
642                     if len(class_buckets[class_idx]) == 0:
643                         split_counts[class_idx] = 0
644                         continue
645                     shuffle(class_buckets[class_idx])
646                     file_splits[i].append(class_buckets[class_idx].pop())
647                     split_counts[class_idx] -= 1
648
649             sorted_class_idxs = np.argsort(split_counts)
650             sorted_class_idxs = sorted_class_idxs[np.argwhere(split_counts[sorted_class_idxs] > 0)].ravel()
651
652         train_files = list(set(file_splits[2]))
653         val_files = list(set(file_splits[1]))
654         test_files = list(set(file_splits[0]))
655         if save:
656             p = save_path_overwrite or self.BASE_DIR
657             fname = filename_overwrite or 'train-test-splits.txt'
658             save_train_test_split(os.path.join(p, fname), train_files, val_files, test_files)
659         return train_files, val_files, test_files
660
661     def count_classes(self):
662         counts = np.zeros(len(self.classes), dtype=int)
663         for i, f in enumerate(self.files):
664             labels = self._read_header_labels(f).astype(float)
665             counts += labels != 0.0 # Count where label isnt 0
666         return counts
667
668     def train_split_with_function(self, file_mapping_function, save_path=None):
669         splits = [[], [], []]
670         for f in self.files:
671             i = file_mapping_function(f)
672             splits[i].append(f)
673         if not save_path is None:
674             save_train_test_split(os.path.join(save_path, 'train-test-splits.txt'), splits[0], splits[1], splits[2])
675         return splits[0], splits[1], splits[2]
676
677     def print_file_attributes(self):
678         f = self.files[0]
679         print('Information for file', f)
680         data = self._read_recording_file(f)
681         print('Data has shape:', data.shape)
682         header = self._read_header_file(f)
683         print('Header is:', header, end='#####\n')
684         print('Classes found in data folder:', self.classes)
685         labels = self._read_header_labels(f)
686         print('Labels have shape', labels.shape)
687
688     def merge_and_update_classes(self, datasets):
689         all_classes = set()
690         for d in datasets:
691             all_classes = all_classes | set(d.classes.keys())
692         all_classes = dict(zip(sorted(all_classes), range(len(all_classes))))
693         for d in datasets:
694             d.classes = all_classes
695         print('Labels for datasets set to:', all_classes)
696
697     def remove_unknown_label_files(self):
698         for f in self.files[:]:
699             if self._read_header_labels(f, onerror_class=None) is None:
700                 print('removed', f)
701                 self.files.remove(f)
702
703
704     def filter_update_classes_by_count(datasets, min_count, add_unknown=False):
705         counts, all_classes = count_merged_classes(datasets)
706         filtered_classes = set()
707         for k, v in all_classes.items():
708             if counts[v] >= min_count:
709                 filtered_classes.add(k)
710         if add_unknown:
711             filtered_classes.add('-1')
712         filtered_classes = dict(zip(sorted(filtered_classes), range(len(filtered_classes))))
713         for d in datasets:

```

```

714     d.classes = filtered_classes
715     d.remove_unknown_label_files()
716     return filtered_classes
717
718
719 def count_merged_classes(datasets):
720     all_classes = set()
721     for d in datasets:
722         all_classes = all_classes | set(d.classes.keys())
723     all_classes = sorted(all_classes)
724     all_classes = dict(zip(all_classes, range(len(all_classes))))
725     counts = np.zeros(len(all_classes), dtype=int)
726     for d in datasets:
727         temp_classes = d.classes.copy() # set back later
728         d.classes = all_classes
729         counts += d.count_classes()
730         d.classes = temp_classes # set back
731     return counts, all_classes
732
733
734 def load_train_test_split(tts_file_path: str):
735     splits = [[], [], []]
736     with open(tts_file_path, 'r') as f:
737         line_count = 0
738         for line in f:
739             if not line.strip().startswith('#') or line == '\n': # Comment line or empty line
740                 splits[line_count] = [f.strip() for f in line.split(',')]
741                 line_count += 1
742             if line_count >= 3:
743                 break
744     return splits[0], splits[1], splits[2]
745
746
747 def save_train_test_split(tts_file: str, trainf=[], valf=[], testf=[]):
748     if os.path.isfile(tts_file): # make a backup just in case
749         sf = os.path.split(tts_file)
750         print(os.path.join(sf[0], timestamp.string_timestamp_minutes()) + sf[1])
751         os.rename(tts_file, os.path.join(sf[0], timestamp.string_timestamp_minutes()) + sf[1])
752
753     with open(tts_file, 'w') as f:
754         f.write('#train files\n')
755         f.write(".".join(trainf) + '\n')
756         f.write('#val files\n')
757         f.write(".".join(valf) + '\n')
758         f.write('#test files\n')
759         f.write(".".join(testf))
760
761
762 def collate_fn(batch): # https://github.com/pytorch/pytorch/blob/master/torch/utils/data/_utils/collate.py
763     r"""Puts each data field into a tensor with outer dimension batch size"""
764
765     elem = batch[0]
766     elem_type = type(elem)
767     if isinstance(elem, torch.Tensor):
768         out = None
769         if torch.utils.data.get_worker_info() is not None:
770             # If we're in a background process, concatenate directly into a
771             # shared memory tensor to avoid an extra copy
772             numel = sum([x.numel() for x in batch])
773             storage = elem.storage()._new_shared(numel)
774             out = elem.new(storage)
775         return torch.stack(batch, 0, out=out)
776     elif elem_type.__module__ == 'numpy' and elem_type.__name__ != 'str_':
777         and elem_type.__name__ != 'string_':
778         if elem_type.__name__ == 'ndarray' or elem_type.__name__ == 'memmap':
779             # array of string classes and object
780             if np_str_obj_array_pattern.search(elem.dtype.str) is not None:
781                 raise TypeError(default_collate_err_msg_format.format(elem.dtype))
782
783             return collate_fn([torch.as_tensor(b) for b in batch])
784     elif elem.shape == (): # scalars
785         return torch.as_tensor(batch)
786     elif isinstance(elem, float):
787         return torch.tensor(batch)
788     elif issubclass(type(elem), int):
789         return torch.tensor(batch)
790     elif issubclass(type(elem), str):
791         return batch
792     elif issubclass(type(elem), dict):
793         return {key: collate_fn([d[key] for d in batch]) for key in elem}
794     elif isinstance(elem, tuple) and hasattr(elem, '_fields'): # namedtuple
795         return elem_type(*(collate_fn(samples) for samples in zip(*batch)))
796     elif issubclass(type(elem), list):
797         # check to make sure that the elements in batch have consistent size
798         transposed = zip(*batch)
799         return [collate_fn(samples) for samples in transposed]
800
801
802 def normalize_minmax_scaling(data, low: int = 0, high: int = 1, axis=0):
803     data = data - data.min(axis=axis, keepdims=True)

```

```

804     data = data / np.maximum(data.max(axis=axis, keepdims=True), 1e-12)
805     data = data * (high-low)
806     data = data + low
807     return data
808
809 def normalize_mean_scaling(data, axis=0):
810     data = data - data.mean(axis=axis, keepdims=True)
811     data = data / np.maximum(data.std(axis=axis, keepdims=True), 1e-12)
812     return data

```

B.10.1.6 util -> data -> ecg_datasets3.py (code)

```

1 import os
2 from random import shuffle
3
4 import torch
5 from torch.utils.data import IterableDataset
6
7 import numpy as np
8
9 from external import helper_code
10 from util.utility import timestamp
11 from torch.utils.data._utils.collate import np_str_obj_array_pattern, default_collate_err_msg_format
12
13
14 class ECGChallengeDatasetBaseline(IterableDataset):
15     def __init__(self, BASE_DIR, window_size=None, pad_to_size=None, files=None, channels=None, return_labels=False,
16                  return_filename=False, classes=None, normalize_fn=None, verbose=False, randomize_order=True):
17         super(IterableDataset).__init__()
18         self.BASE_DIR = BASE_DIR
19         self.window_size = window_size
20         self.pad_to_size = pad_to_size or window_size
21         self.files = files or self.search_files()
22         self.classes = classes or helper_code.get_classes(self.files)
23         if verbose:
24             self.print_file_attributes()
25         self.channels = channels
26         self.total_length = 1 # Trying a weird approach (calculated in __iter__)
27         self.return_labels = return_labels
28         self.return_filename = return_filename
29         self.normalize_fn = normalize_fn if not normalize_fn is None else lambda x: x
30         self.randomize_order = randomize_order
31         self.loaded_data = {}
32         self.loaded_labels = {}
33
34     def generate_datasets_from_split_file(self, ttsfile='train-test-splits.txt'):
35         splits = load_train_test_split(os.path.join(self.BASE_DIR, ttsfile))
36         return tuple(ECGChallengeDatasetBaseline(self.BASE_DIR, self.window_size, self.pad_to_size, files=s,
37                                                 channels=self.channels, return_labels=self.return_labels,
38                                                 return_filename=self.return_filename, classes=self.classes,
39                                                 normalize_fn=self.normalize_fn, randomize_order=self.randomize_order)
39
40             for s in splits)
41
42     def generate_datasets_from_split_list(self, trainf, valf, testf):
43         splits = [trainf, valf, testf]
44         return tuple(ECGChallengeDatasetBaseline(self.BASE_DIR, self.window_size, self.pad_to_size, files=s,
45                                                 channels=self.channels, return_labels=self.return_labels,
46                                                 return_filename=self.return_filename, classes=self.classes,
47                                                 normalize_fn=self.normalize_fn, randomize_order=self.randomize_order)
47
48             for s in splits)
49
50     def preload(self, preload_frac=0.):
51         self.loaded_data = {}
52         self.loaded_labels = {}
53         for f in self.files[:int(len(self.files)*preload_frac)]:
54             d = torch.from_numpy(self.normalize_fn(self._read_recording_file(f)))
55             if len(d) - self.window_size < 0: #pad
56                 d = torch.nn.ReflectionPad1d((0, max(0, self.pad_to_size - min(self.window_size, len(d)))))(d.T.
56
57                 unsqueeze(0)).squeeze(0).T #WTF is this pytorch
58             self.loaded_data[f] = d
59             #del d
60             self.loaded_labels[f] = torch.from_numpy(self._read_header_labels(f))
61
62     def __iter__(self):
63         file_index = 0
64         if self.randomize_order:
65             shuffle(self.files)
66         while file_index < len(self.files):
67             current_file = self.files[file_index]
68             if current_file in self.loaded_data:
69                 data = self.loaded_data[current_file]
70             else:
71                 data = torch.Tensor(self.normalize_fn(self._read_recording_file(current_file)))
72                 if self.channels:
73                     data = data[:, : self.channels]

```

```

74         if self.return_labels:
75             if current_file in self.loaded_labels:
76                 labels = self.loaded_labels[current_file]
77             else:
78                 labels = torch.Tensor(self._read_header_labels(current_file))
79             if len(data) - self.window_size >= 0:
80                 if self.randomize_order and (len(data) - self.window_size) > 0:
81                     offset = np.random.randint(len(data) - self.window_size) # Random offset
82                 else:
83                     offset = 0
84                 data = data[offset:self.window_size + offset]
85             else:
86                 offset = 0
87                 data = torch.nn.ReflectionPad1d((0, max(0, self.pad_to_size - min(self.window_size, len(data))))) (data
88 .T.unsqueeze(0)).squeeze(0).T
89             if not any([self.return_filename, self.return_labels]):
90                 yield data
91             else:
92                 yield [data] + ([labels] if self.return_labels else [None]) + (
93                     [current_file] if self.return_filename else []))
94
95             file_index += 1
96
97     def _read_recording_file(self, path_without_ext):
98         fp = path_without_ext + '.mat'
99         return helper_code.load_recording(fp, key='val').transpose()
100
101    def _read_header_file(self, path_without_ext):
102        fp = path_without_ext + '.hea'
103        return helper_code.load_header(fp)
104
105    def _read_header_labels(self, path_without_ext, onerror_class='426783006'):
106        header = self._read_header_file(path_without_ext)
107        return helper_code.encode_header_labels(header, self.classes, onerror_class)
108
109    def __len__(self):
110        return self.total_length
111
112    def search_files(self):
113        headers, records = helper_code.find_challenge_files(self.BASE_DIR)
114        print(len(records), 'record files found in ', self.BASE_DIR)
115        return list(map(lambda x: os.path.splitext(x)[0], records)) # remove extension
116
117    def random_train_split(self, train_fraction=0.7, val_fraction=0.2, test_fraction=0.1, save=True,
118                           save_path_overwrite=None, filename_overwrite=None):
119        assert train_fraction + val_fraction + test_fraction <= 1
120        N = len(self.files)
121        shuffle(self.files)
122        train_slice = slice(0, int(train_fraction * N))
123        val_slice = slice(train_slice.stop, train_slice.stop + int(val_fraction * N))
124        test_slice = slice(val_slice.stop, val_slice.stop + int(test_fraction * N))
125        if save:
126            p = save_path_overwrite or self.BASE_DIR
127            fname = filename_overwrite or 'train-test-splits.txt'
128            save_train_test_split(os.path.join(p, fname), self.files[train_slice], self.files[val_slice],
129                                  self.files[test_slice])
130        return self.files[train_slice], self.files[val_slice], self.files[test_slice]
131
132    def random_train_split_with_class_count(self, train_fraction=0.7, val_fraction=0.2, test_fraction=0.1, save=True,
133                                           save_path_overwrite=None, filename_overwrite=None):
134        assert train_fraction + val_fraction + test_fraction <= 1
135        class_buckets = [[] for x in range(len(self.classes))] # Creates classes buckets
136        for i, f in enumerate(self.files):
137            label = self._read_header_labels(f)
138            label_idx = np.argwhere(label).flatten()
139            for l in label_idx: # can return more than one (multi-label)
140                class_buckets[l].append(f) # Put this file into class l bucket
141        train_files, val_files, test_files = [], [], []
142        sorted_idx = list(sorted(range(len(class_buckets)),
143                               key=lambda x: len(class_buckets[x]))) # make index sorted by class count low->high
144        print(sorted_idx)
145        while len(sorted_idx) > 0:
146            idx = sorted_idx[0]
147            shuffle(class_buckets[idx])
148            b = class_buckets[idx]
149            c_N = len(b)
150            train_slice = slice(0, int(train_fraction * c_N))
151            val_slice = slice(train_slice.stop, train_slice.stop + int(val_fraction * c_N))
152            test_slice = slice(val_slice.stop, val_slice.stop + int(test_fraction * c_N))
153            train_files += b[train_slice]
154            val_files += b[val_slice]
155            test_files += b[test_slice]
156            used_set = set(b[train_slice] + b[val_slice] + b[test_slice])
157            for j in sorted_idx[1:]:
158                class_buckets[j] = [x for x in class_buckets[j] if
159                                   x not in used_set] # REMOVE THIS FILE FROM ALL OTHER BUCKETS
160            sorted_idx = list(
161                sorted(sorted_idx[1:], key=lambda x: len(class_buckets[x]))) # sort again (removal may change order)
162
163        train_files = list(set(train_files))

```

```

163     val_files = list(set(val_files))
164     test_files = list(set(test_files))
165     if save:
166         p = save_path_overwrite or self.BASE_DIR
167         fname = filename_overwrite or 'train-test-splits.txt'
168         save_train_test_split(os.path.join(p, fname), train_files, val_files, test_files)
169     return train_files, val_files, test_files
170
171 def random_train_split_with_class_count_mins(self, train_counts, val_counts, test_counts, save=True,
172                                              save_path_overwrite=None, filename_overwrite=None):
173     class_buckets = [[] for x in range(len(self.classes))] # Creates classes buckets
174     for i, f in enumerate(self.files):
175         label = self._read_header_labels(f)
176         label_idx = np.argwhere(label).flatten()
177         for l in label_idx: # can return more than one (multi-label)
178             class_buckets[l].append(f) # Put this file into class l bucket
179     file_splits = [[], [], []]
180     for i, split_counts in enumerate([test_counts, val_counts, train_counts]):
181         sorted_class_idxs = np.argsort(split_counts)
182         sorted_class_ids = sorted_class_idxs[np.argwhere(split_counts[sorted_class_ids] > 0)].ravel()
183         while (len(sorted_class_ids) > 0): # TODO:add bad counter
184             for class_idx in sorted_class_ids:
185                 if len(class_buckets[class_idx]) == 0:
186                     split_counts[class_idx] = 0
187                     continue
188                 shuffle(class_buckets[class_idx])
189                 file_splits[i].append(class_buckets[class_idx].pop())
190                 split_counts[class_idx] -= 1
191
192             sorted_class_ids = np.argsort(split_counts)
193             sorted_class_ids = sorted_class_ids[np.argwhere(split_counts[sorted_class_ids] > 0)].ravel()
194
195     train_files = list(set(file_splits[2]))
196     val_files = list(set(file_splits[1]))
197     test_files = list(set(file_splits[0]))
198     if save:
199         p = save_path_overwrite or self.BASE_DIR
200         fname = filename_overwrite or 'train-test-splits.txt'
201         save_train_test_split(os.path.join(p, fname), train_files, val_files, test_files)
202     return train_files, val_files, test_files
203
204 def count_classes(self):
205     counts = np.zeros(len(self.classes), dtype=int)
206     for i, f in enumerate(self.files):
207         labels = self._read_header_labels(f).astype(float)
208         counts += labels != 0.0 # Count where label isn't 0
209     return counts
210
211 def train_split_with_function(self, file_mapping_function, save_path=None):
212     splits = [[], [], []]
213     for f in self.files:
214         i = file_mapping_function(f)
215         splits[i].append(f)
216     if not save_path is None:
217         save_train_test_split(os.path.join(save_path, 'train-test-splits.txt'), splits[0], splits[1], splits[2])
218     return splits[0], splits[1], splits[2]
219
220 def print_file_attributes(self):
221     f = self.files[0]
222     print('Information for file', f)
223     data = self._read_recording_file(f)
224     print('Data has shape:', data.shape)
225     header = self._read_header_file(f)
226     print('Header is:', header, end='#####\n')
227     print('Classes found in data folder:', self.classes)
228     labels = self._read_header_labels(f)
229     print('Labels have shape', labels.shape)
230
231 def merge_and_update_classes(self, datasets):
232     all_classes = set()
233     for d in datasets:
234         all_classes = all_classes | set(d.classes.keys())
235     all_classes = dict(zip(sorted(all_classes), range(len(all_classes))))
236     for d in datasets:
237         d.classes = all_classes
238     print('Labels for datasets set to:', all_classes)
239
240 def remove_unknown_label_files(self):
241     for f in self.files[:]:
242         if self._read_header_labels(f, onerror_class=None) is None:
243             print('removed', f)
244             self.files.remove(f)
245
246
247 def filter_update_classes_by_count(datasets, min_count, add_unknown=False):
248     counts, all_classes = count_merged_classes(datasets)
249     filtered_classes = set()
250     for k, v in all_classes.items():
251         if counts[v] >= min_count:
252             filtered_classes.add(k)

```

```

253     if add_unknown:
254         filtered_classes.add('-1')
255     filtered_classes = dict(zip(sorted(filtered_classes), range(len(filtered_classes))))
256     for d in datasets:
257         d.classes = filtered_classes
258         d.remove_unknown_label_files()
259     return filtered_classes
260
261
262 def count_merged_classes(datasets):
263     all_classes = set()
264     for d in datasets:
265         all_classes = all_classes | set(d.classes.keys())
266     all_classes = sorted(all_classes)
267     all_classes = dict(zip(all_classes, range(len(all_classes))))
268     counts = np.zeros(len(all_classes), dtype=int)
269     for d in datasets:
270         temp_classes = d.classes.copy() # set back later
271         d.classes = all_classes
272         counts += d.count_classes()
273         d.classes = temp_classes # set back
274     return counts, all_classes
275
276
277 def load_train_test_split(tts_file_path: str):
278     splits = [[], [], []]
279     with open(tts_file_path, 'r') as f:
280         line_count = 0
281         for line in f:
282             if not line.strip().startswith('#') or line == '\n': # Comment line or empty line
283                 splits[line_count] = [f.strip() for f in line.split(',')]
284                 line_count += 1
285             if line_count >= 3:
286                 break
287     return splits[0], splits[1], splits[2]
288
289
290 def save_train_test_split(tts_file: str, trainf=[], valf=[], testf=[]):
291     if os.path.isfile(tts_file): # make a backup just in case
292         sf = os.path.split(tts_file)
293         print(os.path.join(sf[0], timestamp.string_timestamp_minutes()) + sf[1])
294         os.rename(tts_file, os.path.join(sf[0], timestamp.string_timestamp_minutes()) + sf[1])
295
296     with open(tts_file, 'w') as f:
297         f.write('#train files\n')
298         f.write(",".join(trainf) + '\n')
299         f.write('#val files\n')
300         f.write(",".join(valf) + '\n')
301         f.write('#test files\n')
302         f.write(",".join(testf))
303
304
305 def collate_fn(batch): # https://github.com/pytorch/pytorch/blob/master/torch/utils/data/_utils/collate.py
306     r"""Puts each data field into a tensor with outer dimension batch size"""
307
308     elem = batch[0]
309     elem_type = type(elem)
310     if isinstance(elem, torch.Tensor):
311         out = None
312         if torch.utils.data.get_worker_info() is not None:
313             # If we're in a background process, concatenate directly into a
314             # shared memory tensor to avoid an extra copy
315             numel = sum([x.numel() for x in batch])
316             storage = elem.storage()._new_shared(numel)
317             out = elem.new(storage)
318         return torch.stack(batch, 0, out=out)
319     elif elem_type.__module__ == 'numpy' and elem_type.__name__ != 'str_':
320         and elem_type.__name__ != 'string__':
321         if elem_type.__name__ == 'ndarray' or elem_type.__name__ == 'memmap':
322             # array of string classes and object
323             if np_str_obj_array_pattern.search(elem.dtype.str) is not None:
324                 raise TypeError(default_collate_err_msg_format.format(elem.dtype))
325
326             return collate_fn([torch.as_tensor(b) for b in batch])
327     elif elem.shape == (): # scalars
328         return torch.as_tensor(batch)
329     elif isinstance(elem, float):
330         return torch.tensor(batch)
331     elif issubclass(type(elem), int):
332         return torch.tensor(batch)
333     elif issubclass(type(elem), str):
334         return batch
335     elif issubclass(type(elem), dict):
336         return {key: collate_fn([d[key] for d in batch]) for key in elem}
337     elif isinstance(elem, tuple) and hasattr(elem, '_fields'): # namedtuple
338         return elem_type(*collate_fn(samples) for samples in zip(*batch))
339     elif issubclass(type(elem), list):
340         # check to make sure that the elements in batch have consistent size
341         transposed = zip(*batch)
342         return [collate_fn(samples) for samples in transposed]
```

```

343
344
345 def normalize_minmax_scaling(data, low: int = 0, high: int = 1, axis=0):
346     data = data - data.min(axis=axis, keepdims=True)
347     data = data / np.maximum(data.max(axis=axis, keepdims=True), 1e-12)
348     data = data * (high-low)
349     data = data + low
350     return data
351
352 def normalize_minmax_scaling_different(data, low: int = 0, high: int = 1, axis=1):
353     data = data - data.min(axis=axis, keepdims=True)
354     data = data / np.maximum(data.max(axis=axis, keepdims=True), 1e-12)
355     data = data * (high-low)
356     data = data + low
357     return data
358
359 def normalize_std_scaling(data, axis=0):
360     data = data - data.mean(axis=axis, keepdims=True)
361     data = data / np.maximum(data.std(axis=axis, keepdims=True), 1e-12)
362     return data

```

B.10.1.7 util -> data -> ptbxl_data.py (code)

```

1 import ast
2 import glob
3 import math
4 import os
5 from collections import defaultdict
6
7 import h5py
8 import numpy as np
9 import pandas as pd
10 import wfdb
11 # In[18]:
12 from sklearn.preprocessing import normalize, OneHotEncoder
13
14
15 # ### Read MIT format .dat ecg data files and .hea headers
16
17
18 class PTBXLData():
19     def __init__(self,
20                  base_directory='/media/julian/Volume/data/ECG/ptb-xl-a-large-publicly-available-electrocardiography-
21 dataset-1.0.1/'):
22         self.BASE_DIR = base_directory
23         self.label_encoder = None
24         self.pca = None
25
26     def search_files(self, file_endings=None, relative=True):
27         root = self.BASE_DIR
28         if file_endings is None:
29             file_endings = ['.dat', '.hea', '.xyz']
30         record_files = []
31         for fe in file_endings:
32             record_files += list(glob.glob(os.path.join(root, '*' + fe)))
33         if relative:
34             record_files = [os.path.basename(f) for f in record_files]
35         record_files = list(set([os.path.splitext(f)[0] for f in record_files]))
36         print(record_files)
37         print(len(record_files), 'record files found in ', root, 'matching', file_endings)
38         return record_files
39
40     def read_signal(self, record_path, physical=True):
41         print(record_path)
42         record = wfdb.rdrecord(record_path, physical=physical) #
43         if physical:
44             data = record.p_signal
45         else:
46             data = record.d_signal
47         return data
48
49     def read_header(self, record_path):
50         record = wfdb.rdheader(record_path)
51         return record.comments
52
53     def convert_dat_to_h5(self, storage_path, dat_file_paths, channels=None, normalize_data=False, pca_components=0,
54                           use_labels=False, verbose=True):
55         if not os.path.exists(storage_path):
56             os.makedirs(storage_path)
57         if use_labels:
58             ptbxl_database_dataframe = self.read_ptbxl_database()
59             spc_codes_dataframe = self.read_ptbxl_scp_statements()
60             for f in dat_file_paths:
61                 absolute = os.path.join(self.BASE_DIR, f)
62                 print(absolute)
63                 if not channels:

```

```

63     data = self.read_signal(absolute)
64
65     else:
66         data = self.read_signal(absolute)[:, channels]
67     if normalize_data:
68         data = normalize(data, norm='l2')
69     if pca_components > 0:
70         print('performing pca with:', data.shape)
71         cov = np.cov(data.T)
72         eig, ev = np.linalg.eigh(cov)
73         evcs = ev[::-1][:, 0:pca_components] # order is ascending, so descend, then take first two
74         data = np.dot(data, evcs)
75         print('output shape pca:', data.shape)
76     target = os.path.join(storage_path, f.replace('/', '-') + '.h5')
77     fail = False
78     with h5py.File(target, 'w') as wf:
79         wf['data'] = data # TODO: Make this a parameter (drop last 3 channels)
80         if use_labels:
81             temp = self.read_label(ptbxl_database_dataframe, spc_codes_dataframe, f)
82             if len(temp) > 0:
83                 labels, likelihoods = zip(*temp[0:1]) # TODO: more than 1 label
84                 onehot = self.label_encoder.transform([labels]).toarray()
85                 wf['label'] = onehot
86                 wf['label'].attrs.create('names',
87                                         [np.array(o, dtype=str) for o in self.label_encoder.categories_])
87                 wf['multilabel'] = self.file_codes_onehot[f]
88                 wf['multilabel'].attrs.create('names', [np.array(o, dtype=str) for o in self.code_list])
89             else:
90                 print(f, "has no diagnostic labels")
91                 fail = True
92             wf.flush()
93         if fail:
94             os.remove(target)
95             print('##### REMOVED', target)
96         else:
97             if verbose: print(target, 'file created and written')
98
99     def init_label_encoder(self):
100         self.label_encoder = OneHotEncoder()
101         df = self.read_ptbxl_scp_statements()
102         X = [[v] for v in df['diagnostic_class'].dropna().unique()]
103         self.label_encoder.fit(X)
104         print(self.label_encoder.categories_)
105
106     def init_multilabel_encoder(self):
107         dbdf = self.read_ptbxl_database()
108         all_labels = defaultdict(set)
109         # Collect all possible labels
110         codes = dbdf['scp_codes']
111         for row in codes:
112             lbl_dict = ast.literal_eval(row)
113             for k, v in lbl_dict.items():
114                 all_labels[k].update({v})
115
116         # Filter out labels that have only 0 probability
117         all_labels = {k: v for k, v in all_labels.items() if max(v) > 0.0}
118
119         # Build a dict with filename as key and scp code as values
120         filenames = dbdf[['filename_hr', 'scp_codes']]
121         file_codes = defaultdict(dict)
122         for i, (f, c) in filenames.iterrows():
123             lbl_dict = ast.literal_eval(c)
124             for k, v in lbl_dict.items():
125                 if k in all_labels and v > 0.0: # First check not necessary
126                     file_codes[f][k] = v / 100.0
127
128         code_indices = dict(zip(all_labels.keys(), range(len(all_labels.keys()))))
129         self.file_codes_onehot = dict()
130         for k, v in file_codes.items():
131             hot_prob = np.zeros(len(code_indices))
132             for ck, cv in v.items():
133                 hot_prob[code_indices[ck]] = cv
134             self.file_codes_onehot[os.path.basename(k)] = hot_prob
135
136         self.code_list = [a[0] for a in sorted(code_indices.items(), key=lambda x: x[1])]
137         print(self.code_list)
138         print(self.file_codes_onehot)
139
140     def read_all_files(self, record_path_list):
141         file_data = []
142         for f in record_path_list:
143             d = self.read_signal(os.path.join(self.BASE_DIR, f))
144             file_data.append(d)
145         return file_data
146
147     def read_ptbxl_database(self):
148         csvfile = os.path.join(self.BASE_DIR, 'ptbxl_database.csv')
149         dataframe = pd.read_csv(csvfile)
150         return dataframe
151
152     def read_ptbxl_scp_statements(self):

```

```

153 csvfile = os.path.join(self.BASE_DIR, 'scp_statements.csv')
154 dataframe = pd.read_csv(csvfile)
155 return dataframe
156
157 def train_test_split(self, record_files, relative=True):
158     if not relative:
159         record_files = {os.path.basename(f): f for f in record_files}
160     df = self.read_ptbxl_database()
161     selection = df[['strat_fold', 'filename_hr', 'filename_lr']]
162     print('Files found in DB:', len(selection))
163     train, val, test = [], [], []
164     rfrset = set(list(record_files.values()))
165     for i, [fold, fhr, flr] in selection.iterrows():
166         fhr = os.path.basename(fhr)
167         flr = os.path.basename(flr)
168         if fhr in rfrset:
169             fhr = fhr
170             if fold <= 8: # https://physionet.org/content/ptb-xl/1.0.1/ #Cross-validation Folds
171                 train.append(fhr)
172             elif fold == 9:
173                 val.append(fhr)
174             elif fold == 10:
175                 test.append(fhr)
176         if flr in rfrset:
177             flr = flr
178             if fold <= 8: # https://physionet.org/content/ptb-xl/1.0.1/ #Cross-validation Folds
179                 train.append(flr)
180             elif fold == 9:
181                 val.append(flr)
182             elif fold == 10:
183                 test.append(flr)
184
185     return train, val, test
186
187 def read_label(self, ptbxl_database_dataframe, spc_codes_dataframe, filename, likelihood_threshold=0.0):
188     df = ptbxl_database_dataframe
189     spc_df = spc_codes_dataframe
190     rf = filename
191     temp = rf # If you need to match files put it here
192
193     row = df.loc[(df['filename_hr'].str.contains(temp)) | (df['filename_lr'].str.contains(temp))]
194     if len(row) < 1:
195         print(rf, 'not found in dataframe')
196         return
197     code = row['scp_codes'].values[0]
198
199     labels = [(c.split(':')[0].replace('(', '').replace(")", '').strip(), c.split(':')[1].replace(')', '').strip())
200     )
201         for c in code.split(',')]
202     print(labels)
203     diagnostic_classes = []
204     for l, p in labels:
205         if float(p) > likelihood_threshold:
206             scp_row = spc_df.loc[(spc_df.iloc[:, 0] == 1) | (spc_df['diagnostic_subclass'] == 1)]
207             if len(scp_row) < 1:
208                 print(l, 'not found in scp_statements')
209                 break
210             v = scp_row['diagnostic_class'].values[0]
211             if not (type(v) == float and math.isnan(v)):
212                 diagnostic_classes.append((v, p))
213
214     return sorted(diagnostic_classes, key=lambda x: x[1], reverse=True)

```

B.10.2 util -> metrics (folder)

B.10.2.1 util -> metrics -> baseline_losses.py (code)

```

1 from torch.nn import functional as F
2 import torch
3 import numpy as np
4
5
6 def MSE_loss(pred, y, weight=None):
7     return F.mse_loss(pred, y, weight)
8
9
10 def cross_entropy(pred, y, weight=None):
11     return -torch.sum(y * torch.log(pred)) / np.prod(y.shape)
12
13
14 def bidirectional_cross_entropy(pred, y, weight=None): # same as BCELoss
15     pred_inv = 1.0 - pred
16     y_inv = 1.0 - y
17     return -(torch.nansum(y * torch.log(pred)) + torch.nansum(y_inv * torch.log(pred_inv))) / (np.prod(y.shape))

```

```

18
19
20 def binary_cross_entropy(pred, y, weight=None):
21     return F.binary_cross_entropy(pred, y, weight=weight)
22
23
24 def multi_loss_function(loss_fns):
25     def fn(pred, y, weight=None):
26         return sum([lfn(pred, y, weight) for lfn in loss_fns]) / len(loss_fns)
27
28     return fn
29
30
31 def nllloss(pred, y, weight=None):
32     F.nll_loss(pred, y, weight=weight)

```

B.10.2.2 util -> metrics -> metrics.py (code)

```

1 import numpy as np
2 from sklearn.metrics import roc_curve, auc, multilabel_confusion_matrix, roc_auc_score
3 import sklearn
4 import pandas as pd
5 import glob
6 import os
7 from sklearn.metrics import precision_recall_curve
8 from sklearn.metrics import average_precision_score
9
10
11 def zero_fit_score(labels, predictions, average: str = None):
12     # print(labels.shape) #== samples x n_classes
13     _, n_classes = labels.shape
14     labels = labels.astype(float)
15     mask = labels == 0.0
16     dists = np.square(np.where(mask, labels - predictions, 0.0))
17     if average == 'micro': # Average over all samples
18         return 1.0 - np.sum(dists) / np.sum(mask) # zero fit goal
19     if average == 'macro':
20         class_average = np.sum(dists, axis=0) / np.sum(mask, axis=0)
21         return 1.0 - np.nanmean(class_average)
22     else: # Dont take average
23         return 1.0 - np.sum(dists, axis=0) / np.sum(mask, axis=0)
24
25
26 def class_fit_score(labels, predictions, average: str = None) -> float:
27     # print(labels.shape) #== samples x n_classes
28     _, n_classes = labels.shape
29     labels = labels.astype(float)
30     mask = labels != 0.0
31     dists = np.sqrt(np.square(np.where(mask, labels - predictions, 0.0)))
32     if average == 'micro': # Average over all samples
33         return 1.0 - np.sum(dists) / np.sum(mask) # zero fit goal
34     if average == 'macro':
35         class_average = np.sum(dists, axis=0) / np.sum(mask, axis=0)
36         return 1.0 - np.nanmean(class_average)
37     else: # Dont take average
38         return 1.0 - np.sum(dists, axis=0) / np.sum(mask, axis=0)
39
40
41 def top1_score(labels, predictions):
42     correct = 0
43     total = 0
44     for l_i in range(len(labels)):
45         label = labels[l_i]
46         pred = predictions[l_i]
47         unique_label_probs = list(reversed(sorted(set(label) - {0.0})))
48         pred_prob_idxs = np.argsort(pred)[::-1]
49         total_local = 0
50         for prob in unique_label_probs:
51             label_prob_idxs = np.argwhere(label == prob) # Get idxs of all occurrences
52             for i in range(len(label_prob_idxs)):
53                 correct += (pred_prob_idxs[i + total_local] in label_prob_idxs) and pred[
54                     pred_prob_idxs[i + total_local]] > 0 # add total
55             total_local += len(label_prob_idxs)
56         total += total_local
57     return 1.0 * correct / total
58
59
60 def confusion_matrix(binary_labels, binary_predictions):
61     return np.array(multilabel_confusion_matrix(binary_labels, binary_predictions), dtype=int)
62
63
64 def ROC(labels: np.ndarray, predictions: np.ndarray): # see scikit learn doc
65     n_classes = labels.shape[1]
66     fpr = dict()
67     tpr = dict()
68     thresholds = dict() # no dict because always the same

```

```

69     roc_auc = dict()
70
71     for i in range(n_classes):
72         fpr[i], tpr[i], thresholds[i] = roc_curve(labels[:, i], predictions[:, i])
73         roc_auc[i] = auc(fpr[i], tpr[i])
74
75     fpr["micro"], tpr["micro"], _ = roc_curve(labels.ravel(), predictions.ravel())
76     roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
77
78     return tpr, fpr, roc_auc, thresholds
79
80
80 def precision_recall(labels: np.ndarray, predictions: np.ndarray, n_classes): # see scikit learn doc
81     precision = dict()
82     recall = dict()
83     average_precision = dict()
84     for i in range(n_classes): # For each class
85         precision[i], recall[i], _ = precision_recall_curve(labels[:, i],
86                                                               predictions[:, i])
87         average_precision[i] = average_precision_score(labels[:, i], predictions[:, i])
88
89     # A "micro-average": quantifying score on all classes jointly
90     precision["micro"], recall["micro"], _ = precision_recall_curve(labels.ravel(),
91                                                                     predictions.ravel())
92     average_precision["micro"] = average_precision_score(labels, predictions,
93                                                           average="micro")
94     average_precision["macro"] = average_precision_score(labels, predictions,
95                                                           average="macro")
96     print('Average precision score, micro-averaged over all classes: {0:0.5f}'.
97           format(average_precision["micro"]))
98     print('Average precision score, macro-averaged over all classes: {0:0.5f}'.
99           format(average_precision["macro"]))
100    return precision, recall, average_precision
101
102
103 def auc_scores(labels, predictions, average=None):
104     auc = roc_auc_score(labels, predictions, average=average)
105
106     return auc
107
108 def precision_scores(binary_labels, binary_predictions, average=None):
109     cm = confusion_matrix(binary_labels, binary_predictions) # n_classes x 2 x 2
110     tn = cm[:, 0, 0] # TN
111     fn = cm[:, 1, 0] # FN
112     fp = cm[:, 0, 1] # FP
113     tp = cm[:, 1, 1] # TP
114     if average == 'macro':
115         return np.mean(tp / (tp + fp))
116     elif average == 'micro':
117         return np.sum(tp) / (np.sum(tp) + np.sum(fp)) # Micro
118     else:
119         return tp / (tp + fp)
120
121
122 def recall_scores(binary_labels, binary_predictions, average=None):
123     cm = confusion_matrix(binary_labels, binary_predictions) # n_classes x 2 x 2
124     tn = cm[:, 0, 0] # TN
125     fn = cm[:, 1, 0] # FN
126     fp = cm[:, 0, 1] # FP
127     tp = cm[:, 1, 1] # TP
128     if average == 'macro':
129         return np.mean(tp / (tp + fn))
130     elif average == 'micro':
131         return np.sum(tp) / (np.sum(tp) + np.sum(fn)) # Micro
132     else:
133         return tp / (tp + fn)
134
135
136 def accuracy_scores(binary_labels, binary_predictions, average=None):
137     cm = confusion_matrix(binary_labels, binary_predictions) # n_classes x 2 x 2
138     tn = cm[:, 0, 0] # TN
139     fn = cm[:, 1, 0] # FN
140     fp = cm[:, 0, 1] # FP
141     tp = cm[:, 1, 1] # TP
142     if average == 'macro':
143         return np.mean((tp + tn) / (tp + tn + fp + fn))
144     elif average == 'micro':
145         return (np.sum(tp) + np.sum(tn)) / (np.sum(tp) + np.sum(fp) + np.sum(tn) + np.sum(fn)) # Micro
146     else:
147         return (tp + tn) / (tp + tn + fp + fn)
148
149
150 def average_precision_scores(binary_labels, score_predictions, average=None):
151     return sklearn.metrics.average_precision_score(binary_labels, score_predictions, average=average)
152
153
154 def balanced_accuracy_scores(binary_labels, binary_predictions, average=None):
155     cm = confusion_matrix(binary_labels, binary_predictions) # n_classes x 2 x 2
156     tn = cm[:, 0, 0] # TN
157     fn = cm[:, 1, 0] # FN
158     fp = cm[:, 0, 1] # FP

```

```

159     tp = cm[:, 1, 1] # TP
160     # TPR = (tp/(tp+fn))
161     # TNR = (tn/(tn+fp))
162     if average == 'macro':
163         return np.mean(((tp / (tp + fn)) + (tn / (tn + fp))) / 2)
164     elif average == 'micro':
165         return ((np.sum(tp) / (np.sum(tp) + np.sum(fn))) + (np.sum(tn) / (np.sum(tn) + np.sum(fp)))) / 2
166     else:
167         return ((tp / (tp + fn)) + (tn / (tn + fp))) / 2
168
169
170 def f1_scores(binary_labels, binary_predictions, average=None):
171     cm = confusion_matrix(binary_labels, binary_predictions) # n_classes x 2 x 2
172     tn = cm[:, 0, 0] # TN
173     fn = cm[:, 1, 0] # FN
174     fp = cm[:, 0, 1] # FP
175     tp = cm[:, 1, 1] # TP
176     if average == 'macro':
177         return np.mean(tp / (tp + (fp + fn) / 2.0))
178     elif average == 'micro':
179         return np.sum(tp) / (np.sum(tp) + (np.sum(fp) + np.sum(fn)) / 2.0) # Micro
180     else:
181         return tp / (tp + (fp + fn) / 2.0)
182
183
184 def brier_score(labels, predictions, average=None):
185     print(labels.shape)
186     labels = labels.astype(float)
187     dists = np.square(labels - predictions)
188     if average == 'micro': # Average over all samples
189         return 1.0 - np.nanmean(dists) # zero fit goal
190     if average == 'macro':
191         class_average = np.nanmean(dists, axis=0)
192         return 1.0 - np.nanmean(class_average)
193     else: # Dont take average
194         return 1.0 - np.nanmean(dists, axis=0)
195
196
197 def select_best_thresholds(tpr, fpr, thresholds, n_classes):
198     best_threshold = dict()
199     tps = dict()
200     fps = dict()
201     for i in range(n_classes):
202         if np.isnan(fpr[i]).all() or np.isnan(tpr[i]).all():
203             best_threshold[i] = np.nan
204         else:
205             ix = np.argmax(tpr[i] - fpr[i]) # youden J
206             # ix = np.argmax(np.sqrt(tpr[i] * (1-fpr[i]))) #gmeans
207             best_threshold[i] = thresholds[i][ix]
208             tps[i] = tpr[i][ix]
209             fps[i] = fpr[i][ix]
210     return tps, fps, best_threshold
211
212
213 def class_count_table(labels: np.ndarray, binary_predictions: np.ndarray, n_classes=94):
214     counts = np.zeros((n_classes, n_classes))
215     for i in range(n_classes):
216         for j in range(n_classes):
217             counts[i, j] = np.sum(labels[:, i] == binary_predictions[:, j])
218     return counts
219
220
221 def convert_pred_to_binary(predictions, thresholds):
222     thresholds_np = np.array([t for t in thresholds.values()])
223     return (predictions >= thresholds_np).astype(int)
224
225
226 def read_output_csv_from_model_folder(model_folder, data_loader_index=0):
227     pred_path = glob.glob(os.path.join(model_folder, f"model-*-{data_loader_index}-output.csv"))
228     if len(pred_path) == 0:
229         raise FileNotFoundError
230     dfp = pd.read_csv(pred_path[0])
231     return dfp.values[:, 1:].astype(float), dfp.columns[1:].values # 1 is file
232
233
234 def read_label_csv_from_model_folder(model_folder, data_loader_index=0):
235     label_path = glob.glob(os.path.join(model_folder, f"labels-dataloader-{(data_loader_index)}.csv"))
236     if len(label_path) == 0:
237         raise FileNotFoundError
238     df1 = pd.read_csv(label_path[0])
239     labels = df1.values[:, 1:1].astype(float)
240     return labels, df1.columns[1:1].values
241
242
243 def read_binary_label_csv_from_model_folder(model_folder, data_loader_index=0):
244     l, c = read_label_csv_from_model_folder(model_folder, data_loader_index)
245     l = (l > 0).astype(int)
246     return l, c
247
248

```

```

249 if __name__ == '__main__': # usage example
250     model_folder = '/home/julian/Downloads/Github/contrastive-predictive-coding/models/22_04_21-17/architectures_cpc.
251     cpc_combined.CPCCombined0'
251     labels, _ = read_binary_label_csv_from_model_folder(model_folder)
252     print(labels[0])
253     pred, _ = read_output_csv_from_model_folder(model_folder)

```

B.10.2.3 util -> metrics -> training_metrics.py (code)

```

1 import torch as t
2
3
4 def tp_score_global(y: t.Tensor, pred: t.Tensor, pred_threshold=0.5):
5     """
6         True Positive accuracy
7         :param y: ground truth
8         :param pred: prediction of same shape
9         :param pred_threshold: the pred_threshold at what probability a class is considered "True"
10        :return: The True Positive sum/score
11    """
12    mask = y >= pred_threshold
13    return t.sum(pred[mask] >= pred_threshold) / t.sum(mask)
14
15
16 def fp_score_global(y: t.Tensor, pred: t.Tensor, pred_threshold=0.5):
17     """
18         False Positive accuracy
19         :param y: ground truth
20         :param pred: prediction of same shape
21         :param pred_threshold: the pred_threshold at what probability a class is considered "True"
22         :return: The True Positive sum/score
23    """
24    mask = y < pred_threshold
25    return t.sum(pred[mask] >= pred_threshold) / t.sum(mask)
26
27
28 def tn_score_global(y: t.Tensor, pred: t.Tensor, pred_threshold=0.5):
29     """
30         :param y: ground truth
31         :param pred: prediction of same shape
32         :param pred_threshold: the pred_threshold at what probability a class is considered "True"
33         :return: The True Positive sum/score
34    """
35    mask = y < pred_threshold
36    return t.sum(pred[mask] < pred_threshold) / t.sum(mask)
37
38
39 def fn_score_global(y: t.Tensor, pred: t.Tensor, pred_threshold=0.5):
40     """
41         :param y: ground truth
42         :param pred: prediction of same shape
43         :param pred_threshold: the pred_threshold at what probability a class is considered "True"
44         :return: The True Positive score
45    """
46    mask = y >= pred_threshold
47    return t.sum(pred[mask] < pred_threshold) / t.sum(mask)
48
49
50 def tp_score_label(y: t.Tensor, pred: t.Tensor, y_threshold=0.5, pred_threshold=0.5):
51     """
52         Returns the TP score for each class in a vector
53         :param y:
54         :param pred:
55         :param pred_threshold:
56         :return:
57    """
58    mask = (y >= y_threshold)
59    return t.sum(t.where(mask, pred, t.tensor(-0.1, dtype=pred.dtype, device=pred.device)) >= pred_threshold,
60                dim=0) / t.sum(mask, dim=0)
61
62
63 def fp_score_label(y: t.Tensor, pred: t.Tensor, y_threshold=0.5, pred_threshold=0.5):
64     """
65         Returns the TP score for each class in a vector
66         :param y:
67         :param pred:
68         :param pred_threshold:
69         :return:
70    """
71    mask = y < y_threshold
72    return t.sum(t.where(mask, pred, t.tensor(-0.1, dtype=pred.dtype, device=pred.device)) >= pred_threshold,
73                dim=0) / t.sum(mask, dim=0)
74
75
76 def tn_score_label(y: t.Tensor, pred: t.Tensor, y_threshold=0.5, pred_threshold=0.5):
77     """

```

```

78     Returns the TP score for each class in a vector
79     :param y:
80     :param pred:
81     :param pred_threshold:
82     :return:
83     """
84     mask = y < y_threshold
85     return t.sum(t.where(mask, pred, t.tensor(1.1, dtype=pred.dtype, device=pred.device)) < pred_threshold,
86                 dim=0) / t.sum(mask, dim=0)
87
88
89 def fn_score_label(y: t.Tensor, pred: t.Tensor, y_threshold=0.5, pred_threshold=0.5):
90     """
91     Returns the TP score for each class in a vector
92     :param y:
93     :param pred:
94     :param pred_threshold:
95     :return:
96     """
97     mask = y >= y_threshold
98     return t.sum(t.where(mask, pred, t.tensor(1.1, dtype=pred.dtype, device=pred.device)) < pred_threshold,
99                 dim=0) / t.sum(mask, dim=0)
100
101
102 def micro_avg_precision_score(y: t.Tensor, pred: t.Tensor, y_threshold=0.5, pred_threshold=0.5):
103     tps = tp_score_label(y, pred, y_threshold, pred_threshold)
104     fps = fp_score_label(y, pred, y_threshold, pred_threshold)
105     return t.nansum(tps) / t.nansum(tps + fps)
106
107
108 def micro_avg_recall_score(y: t.Tensor, pred: t.Tensor, y_threshold=0.5, pred_threshold=0.5):
109     tps = tp_score_label(y, pred, y_threshold, pred_threshold)
110     fns = fn_score_label(y, pred, y_threshold, pred_threshold)
111     return t.nansum(tps) / (t.nansum(tps) + t.nansum(fns))
112
113
114 def f1_score(y: t.Tensor, pred: t.Tensor, y_threshold=0.5, pred_threshold=0.5):
115     precision = micro_avg_precision_score(y, pred, y_threshold, pred_threshold)
116     recall = micro_avg_recall_score(y, pred, y_threshold, pred_threshold)
117     return 2 * precision * recall / (precision + recall)
118
119
120 def accuracy(y: t.Tensor, pred: t.Tensor, y_threshold=0.5, pred_threshold=0.5):
121     tps = tp_score_label(y, pred, y_threshold, pred_threshold)
122     fps = fp_score_label(y, pred, y_threshold, pred_threshold)
123     tns = tn_score_label(y, pred, y_threshold, pred_threshold)
124     fns = fn_score_label(y, pred, y_threshold, pred_threshold)
125     return (t.nansum(tps) + t.nansum(tns)) / (t.nansum(fps) + t.nansum(fns) + t.nansum(tps) + t.nansum(tns))
126
127
128 def class_count_prediction(y: t.Tensor, pred: t.Tensor, y_threshold=0.5, pred_threshold=0.5):
129     return t.sum(pred > pred_threshold, dim=0)
130
131
132 def class_count_truth(y: t.Tensor, pred: t.Tensor, y_threshold=0.5, pred_threshold=0.5):
133     return t.sum(y, dim=0)
134
135
136 def zero_fit_score(y: t.Tensor, pred: t.Tensor, y_threshold=0.5, pred_threshold=0.5) -> float:
137     mask = y != 0.0
138     inverse_mask = ~mask
139     return 1.0 - t.sqrt(t.sum(t.square(y[inverse_mask] - pred[inverse_mask]))) / t.sum(inverse_mask) # zero fit goal
140
141
142 def class_fit_score(y: t.Tensor, pred: t.Tensor, y_threshold=0.5, pred_threshold=0.5) -> float:
143     mask = y != 0.0
144     return 1.0 - t.sqrt(t.sum(t.square(y[mask] - pred[mask]))) / t.sum(mask) # zero fit goal

```

B.10.3 util -> utility (folder)

B.10.3.1 util -> utility -> dict_utils.py (code)

```

1 def count_key_in_dict(dictionary, search_key, decision_fn=None):
2     count = 0
3     if type(dictionary) == dict:
4         for k, v in dictionary.items():
5             if decision_fn is None:
6                 count += k == search_key
7             else:
8                 count += decision_fn(k, search_key)
9             count += count_key_in_dict(v, search_key)
10
11     return count

```

```

12 def extract_values_for_key_in_dict(dictionary, search_key, decision_fn=None):
13     values = []
14     if type(dictionary) == dict:
15         for k, v in dictionary.items():
16             if decision_fn is None:
17                 if k == search_key:
18                     if type(v) == list:
19                         values += v
20                     else:
21                         values += [v]
22             elif decision_fn(k, search_key):
23                 if type(v) == list:
24                     values += v
25                 else:
26                     values += [v]
27     values += extract_values_for_key_in_dict(v, search_key)
28
29 return values

```

B.10.3.2 util -> utility -> extract_trained_model_attributes.py (code)

```

1 import itertools
2 import json
3 import os
4 import glob
5
6 import matplotlib
7 import numpy as np
8 import pandas as pd
9 import matplotlib.pyplot as plt
10 from matplotlib import ticker
11 from pandas.plotting import parallel_coordinates
12 import plotly.express as px
13 from scipy.interpolate import make_interp_spline
14 from util.metrics import metrics as m
15 import util.visualize.plot_metrics as plotm
16 import seaborn as sns
17 from util.utility.dict_utils import count_key_in_dict, extract_values_for_key_in_dict
18
19
20 def extract_baseline_model_attributes(train_folders, extract_baselines=True, extract_cpcs=False, exclude_models=[]):
21     model_descriptions = {}
22     collision_counter = 1
23     for f in train_folders:
24         if not ('test' in f):
25             print(f"Checking {f}...")
26             for root, dirs, files in os.walk(f, followlinks=True):
27                 print(root)
28                 if len(dirs) == 0 and len(files) == 0:
29                     continue
30                 if len(dirs) > 0: # Traverse more
31                     continue
32                 if len(dirs) == 0 and len(files) > 0: # leaf dir (model?)
33                     name = os.path.basename(root).split('/')[-1]
34                     print(root)
35                     if 'architectures_cpc.' in name and not extract_cpcs: # its a cpc model but dont extract
36                         continue
37                     if 'architectures_baseline_challenge.' in name and not extract_baselines: # its a cpc model but
38                         dont extract
39                         continue
40                     if any([em in name for em in exclude_models]): # exclude this model
41                         continue
42                     name = name.replace('architectures_baseline_challenge.', '')
43                     name = name.replace('architectures_cpc.', '')
44                     print(name)
45                     if name in model_descriptions:
46                         name = name + f'{collision_counter}'
47                         collision_counter += 1
48                     model_descriptions[name] = {}
49                     if 'model_arch.txt' in files:
50                         with open(os.path.join(root, 'model_arch.txt'), 'r') as file:
51                             content = file.read()
52                             # print(content)
53
54                     if 'model_variables.txt' in files:
55                         with open(os.path.join(root, 'model_variables.txt'), 'r') as file:
56                             content = file.readlines()
57                             for i, line in enumerate(content):
58                                 if '{' in line:
59                                     content = '\n'.join(content[i:])
60                                     break
61                             data = json.loads(content)
62                         model_descriptions[name]['ConvId'] = count_key_in_dict(data, 'torch.nn.modules.conv.ConvId')
63                         model_descriptions[name]['MaxPoolId'] = count_key_in_dict(data,
64                                         'torch.nn.modules.pooling.MaxPoolId'
65 )

```

```

model_descriptions[name]['AdaptiveAvgPoolId'] = count_key_in_dict(data,
                                                               'torch.nn.modules.pooling.')
AdaptiveAvgPoolId')
model_descriptions[name]['Linear'] = count_key_in_dict(data, 'torch.nn.modules.linear.Linear')
model_descriptions[name]['LSTM'] = count_key_in_dict(data, 'torch.nn.modules.rnn.LSTM')
model_descriptions[name]['BatchNormId'] = count_key_in_dict(data,
                                                               'torch.nn.modules.batchnorm.')
BatchNormId')
model_descriptions[name]['stride product'] = np.array(
    extract_values_for_key_in_dict(data, 'stride')).prod()
model_descriptions[name]['dilation sum'] = np.array(
    extract_values_for_key_in_dict(data, 'dilation')).sum()
model_descriptions[name]['padding sum'] = np.array(
    extract_values_for_key_in_dict(data, 'padding')).sum()
model_descriptions[name]['kernelsize sum'] = np.array(
    extract_values_for_key_in_dict(data, 'kernel_size')).sum()
if 'params.txt' in files:
    with open(os.path.join(root, 'params.txt'), 'r') as file:
        content = file.read()
    model_descriptions[name]['uses class weights'] = 'use_class_weights=True' in content

if model_descriptions[name] == {}:
    del model_descriptions[name]
return model_descriptions

def extract_cpc_model_attributes_from_test(test_folders, exclude_models=[], check_collision=False):
    model_descriptions = []
    for f in test_folders:
        print(f"Checking {f}...")
        if '-test' in f:
            for root, dirs, files in os.walk(f, followlinks=True):
                print(root)
                if len(dirs) == 0 and len(files) == 0:
                    continue
                if len(dirs) > 0 and len(files) > 0: # Contains auc?
                    continue
                if len(dirs) == 0 and len(files) > 0: # leaf dir (model?)
                    name = os.path.basename(root).split('!')[0]
                    print(root)
                    if 'BaselineNet' in name: # its a baseline model so dont extract
                        continue
                    if any(em in name for em in exclude_models): # exclude this model
                        continue
                    name = name.replace('architectures_baseline_challenge.', '')
                    name = name.replace('architectures_cpc.', '')
                    name = ''.join([i for i in name if not i.isdigit()])
                    print(name)
                    model_description = {}
                    try:
                        binary_labels, classes = m.read_binary_label_csv_from_model_folder(root,
                                                                                           data_loader_index=0)
                        predictions, pred_classes = m.read_output_csv_from_model_folder(root,
                                                                                           data_loader_index=0)
                        model_description['micro-auc'] = m.auc_scores(binary_labels, predictions, average='micro')
                        model_description['macro-auc'] = m.auc_scores(binary_labels, predictions, average='macro')
                    except FileNotFoundError:
                        print(root, "Model wasnt really tested!")
                        continue
                    if 'model_arch.txt' in files:
                        with open(os.path.join(root, 'model_arch.txt'), 'r') as file:
                            content = file.read()
                            # print(content)

                    if 'model_variables.txt' in files:
                        with open(os.path.join(root, 'model_variables.txt'), 'r') as file:
                            content = file.readlines()
                            for i, line in enumerate(content):
                                if '(' in line:
                                    content = '\n'.join(content[i:])
                                    break
                            data = json.loads(content)
                            model_description['weights frozen'] = '"freeze_cpc": true' in content
                            model_description['use context'] = '"use_context": true' in content
                            model_description['use latents'] = '"use_latents": true' in content
                            model_description['normalized latents'] = '"normalize_latents": true' in content

                            if "sampling_mode" in content:
                                model_description['sampling mode'] = content.split('"sampling_mode": ')[1].split(',')[-1]
                            else:
                                model_description['sampling mode'] = 'same'
                    if '"downstream_model"' in content:
                        model_description['downstream model'] = \
                            content.split('"downstream_model": ')[1].split(':')[0].strip().lstrip('"').split('\"')

                .)[
                    -2]
            else:
                model_description['downstream model'] = 'none'
            model_description['strided'] = 'cpc_encoder_as_strided.StridedEncoder' in content

```

```

151         if len(model_description.keys()) > 2:
152             model_descriptions.append(model_description)
153     return model_descriptions
154
155 def extract_cpc_model_attributes(train_folders, exclude_models=[]):
156     model_descriptions = {}
157     for f in train_folders:
158         print(f"Checking {f}...")
159         for root, dirs, files in os.walk(f, followlinks=True):
160             print(root)
161             if len(dirs) == 0 and len(files) == 0:
162                 continue
163             if len(dirs) > 0: # Traverse more
164                 continue
165             if len(dirs) == 0 and len(files) > 0: # leaf dir (model?)
166                 name = os.path.basename(root).split('/')[-1]
167                 print(root)
168                 if 'architectures_baseline_challenge.' in name: # its a baseline model so dont extract
169                     continue
170                 if any([em in name for em in exclude_models]): # exclude this model
171                     continue
172                 name = name.replace('architectures_baseline_challenge.', '')
173                 name = name.replace('architectures_cpc.', '')
174                 print(name)
175                 model_descriptions[name] = {}
176                 if 'model_arch.txt' in files:
177                     with open(os.path.join(root, 'model_arch.txt'), 'r') as file:
178                         content = file.read()
179                         # print(content)
180
181                 if 'model_variables.txt' in files:
182                     with open(os.path.join(root, 'model_variables.txt'), 'r') as file:
183                         content = file.readlines()
184                         for i, line in enumerate(content):
185                             if '{' in line:
186                                 content = '\n'.join(content[i:])
187                                 break
188                         data = json.loads(content)
189                         model_descriptions[name]['weights frozen'] = not '"freeze_cpc": false' in content
190                         model_descriptions[name]['use context'] = '"use_context": true' in content
191                         model_descriptions[name]['use latents'] = '"use_latents": true' in content
192                         model_descriptions[name]['normalized latents'] = '"normalize_latents": true' in content
193
194                 if "sampling_mode" in content:
195                     model_descriptions[name]['sampling mode'] = content.split('"sampling_mode": ')[1].split(',')[
196                         0][1:-1]
197                     else:
198                         model_descriptions[name]['sampling mode'] = 'none'
199                     if '"downstream_model":' in content:
200                         model_descriptions[name]['downstream model'] = \
201                             content.split('"downstream_model": ')[1][1].split(':')[1][0].strip().lstrip('"').split('.')[
202                             -2]
203                     else:
204                         model_descriptions[name]['downstream model'] = 'none'
205
206                 if 'params.txt' in files:
207                     with open(os.path.join(root, 'params.txt'), 'r') as file:
208                         content = file.read()
209                         model_descriptions[name]['uses class weights'] = 'use_class_weights=True' in content
210                     if model_descriptions[name] == {}:
211                         del model_descriptions[name]
212     return model_descriptions
213
214
215 def plot_parallel_coordinates(df, name_column):
216     # import packages
217     import numpy as np
218     import matplotlib.pyplot as plt
219     plt.tight_layout()
220     model_names = df[name_column].values
221     df = df.drop(name_column, axis='columns')
222     data = df.values
223     n_categories = data.shape[1]
224     x = np.arange(n_categories)
225     x_labels = df.columns
226
227     fig, axs = plt.subplots(1, n_categories - 1, sharey='none')
228
229     # plot subplots and set xlim
230     for i in range(len(axs)):
231         for j in range(len(data)):
232             axs[i].plot(x_labels, data[j])
233             axs[i].set_xlim(x_labels[i], x_labels[i + 1])
234             axs[i].set_ylim(data[:, i].min(), data[:, i].max())
235             axs[i].tick_params('x', labelrotation=90)
236
237     # set width space to zero
238     plt.subplots_adjust(wspace=0)
239     # show plot
240     plt.show()

```

```

240
241
242 def parallel_coordinates_custom(df, name_column, style=None): # https://stackoverflow.com/q/8230638
243
244     model_names = df[name_column].values
245     df = df.drop(name_column, axis='columns')
246     data_sets = df.values
247     n_categories = data_sets.shape[1]
248     x = np.arange(n_categories)
249     x_labels = list(df.columns)
250
251     dims = len(data_sets[0])
252     x = range(dims) # x_labels#
253     fig, axes = plt.subplots(1, dims - 1, sharey='none')
254
255     if style is None:
256         style = ['r-' * len(data_sets)]
257
258     # Calculate the limits on the data
259     min_max_range = list()
260     for m in zip(*data_sets):
261         if type(m) == str:
262             sorted(m)
263         else:
264             mn = min(m)
265             mx = max(m)
266             if mn == mx:
267                 mn -= 0.5
268                 mx = mn + 1.
269             r = float(mx - mn)
270             min_max_range.append((mn, mx, r))
271
272     # Normalize the data sets
273     norm_data_sets = list()
274     for ds in data_sets:
275         nds = [(value - min_max_range[dimension][0]) /
276                 min_max_range[dimension][2]
277                 for dimension, value in enumerate(ds)]
278     norm_data_sets.append(nds)
279     data_sets = norm_data_sets
280
281     # Plot the datasets on all the subplots
282     for i, ax in enumerate(axes):
283         for dsi, d in enumerate(data_sets):
284             ax.plot(x, d, style[dsi])
285             ax.set_xlim([x[i], x[i + 1]])
286
287     # Set the x axis ticks
288     for dimension, (axx, xx) in enumerate(zip(axes, x[:-1])):
289         axx.xaxis.set_major_locator(ticker.FixedLocator([xx]))
290         ticks = len(axx.get_yticklabels())
291         labels = list()
292         step = min_max_range[dimension][2] / (ticks - 1)
293         mn = min_max_range[dimension][0]
294         for i in range(ticks):
295             v = mn + i * step
296             labels.append('%4.2f' % v)
297         axx.set_yticklabels(labels)
298
299     # Move the final axis' ticks to the right-hand side
300     axx = plt.twinx(axes[-1])
301     dimension += 1
302     axx.xaxis.set_major_locator(ticker.FixedLocator([x[-2], x[-1]]))
303     ticks = len(axx.get_yticklabels())
304     step = min_max_range[dimension][2] / (ticks - 1)
305     mn = min_max_range[dimension][0]
306     labels = ['%4.2f' % (mn + i * step) for i in range(ticks)]
307     axx.set_yticklabels(labels)
308
309     # Stack the subplots
310     plt.subplots_adjust(wspace=0)
311
312     plt.show()
313
314
315 def factorize_dataframe(df, column):
316     print(df[column])
317     codes = df[column].astype("category").cat.codes
318     print('after fac', df[column])
319     df.update(pd.DataFrame({column: codes}))
320
321
322 # https://github.com/jraine/parallel-coordinates-plot-dataframe/blob/master/parallel_plot.py
323 def parallel_plot(df, cols, rank_attr, cmap='magma', spread=None, curved=False, curvedextend=0.1, save=None):
324     '''Produce a parallel coordinates plot from pandas dataframe with line colour with respect to a column.
325     Required Arguments:
326         df: dataframe
327         cols: columns to use for axes
328         rank_attr: attribute to use for ranking
329     Options:

```

```

330     cmap: Colour palette to use for ranking of lines
331     spread: Spread to use to separate lines at categorical values
332     curved: Spline interpolation along lines
333     curvedextend: Fraction extension in y axis, adjust to contain curvature
334 Returns:
335     x coordinates for axes, y coordinates of all lines''
336 colmap = sns.color_palette(cmap, as_cmap=True)
337 cols = cols + [rank_attr]
338
339 fig, axes = plt.subplots(1, len(cols) - 1, sharey=False, figsize=(3 * len(cols) + 3, 5), dpi=800)
340 valmat = np.ndarray(shape=(len(cols), len(df)))
341 x = np.arange(0, len(cols), 1)
342 ax_info = {}
343 for i, col in enumerate(cols):
344     vals = df[col]
345     if (vals.dtype == float) and (len(np.unique(vals)) > 20):
346         minval = np.min(vals)
347         maxval = np.max(vals)
348         rangeval = maxval - minval
349         vals = np.true_divide(vals - minval, maxval - minval)
350         nticks = 5
351         tick_labels = [round(minval + i * (rangeval / nticks), 4) for i in range(nticks + 1)]
352         ticks = [0 + i * (1.0 / nticks) for i in range(nticks + 1)]
353         valmat[i] = vals
354         ax_info[col] = [tick_labels, ticks]
355     else:
356         vals = vals.astype('category')
357         cats = vals.cat.categories
358         c_vals = vals.cat.codes
359         minval = 0
360         maxval = len(cats) - 1
361         if maxval == 0:
362             c_vals = 0.5
363         else:
364             c_vals = np.true_divide(c_vals - minval, maxval - minval)
365         tick_labels = cats
366         ticks = np.unique(c_vals)
367         ax_info[col] = [tick_labels, ticks]
368         if spread is not None:
369             offset = np.arange(-1, 1, 2. / (len(c_vals))) * 2e-2
370             np.random.shuffle(offset)
371             c_vals = c_vals + offset
372             valmat[i] = c_vals
373
374 extendfrac = curvedextend if curved else 0.05
375 for i, ax in enumerate(axes):
376     for idx in range(valmat.shape[-1]):
377         if curved:
378             x_new = np.linspace(0, len(x), len(x) * 20)
379             a_BSpline = make_interp_spline(x, valmat[:, idx], k=3, bc_type='clamped')
380             y_new = a_BSpline(x_new)
381             ax.plot(x_new, y_new, color=colmap(valmat[-1, idx]), alpha=0.3)
382         else:
383             ax.plot(x, valmat[:, idx], color=colmap(valmat[-1, idx]), alpha=0.3)
384         ax.set_ylim(0 - extendfrac, 1 + extendfrac)
385         ax.set_xlim(i, i + 1)
386
387 for dim, (ax, col) in enumerate(zip(axes, cols)):
388     ax.xaxis.set_major_locator(ticker.FixedLocator([dim]))
389     ax.yaxis.set_major_locator(ticker.FixedLocator(ax_info[col][1]))
390     ax.set_yticklabels(ax_info[col][0])
391     ax.set_xticklabels([cols[dim]])
392
393 plt.subplots_adjust(wspace=0)
394 norm = matplotlib.colors.Normalize(0, 1) # *axes[-1].get_ylim())
395 sm = plt.cm.ScalarMappable(cmap=colmap, norm=norm)
396 cbar = plt.colorbar(sm, pad=0, ticks=ax_info[rank_attr][1], extend='both', extendrect=True, extendfrac=extendfrac)
397 if curved:
398     cbar.ax.set_ylim(0 - curvedextend, 1 + curvedextend)
399     cbar.ax.set_yticklabels(ax_info[rank_attr][0])
400     cbar.ax.set_xlabel(rank_attr)
401     if save:
402         fig.savefig(save, bbox_inches='tight', dpi=fig.dpi)
403     else:
404         plt.show()
405
406 return x, valmat
407
408
409 if __name__ == '__main__':
410     # model_descriptions = extract_baseline_model_attributes(
411     #     ['/home/julian/Downloads/Github/contrastive-predictive-coding/models_symbolic_links/train'],
412     #     exclude_models=['v14'])
413     #model_descriptions = extract_cpc_model_attributes_from_test(glob.glob('/home/julian/Downloads/Github/contrastive-
414     #predictive-coding/models/*'))
415     #df = pd.DataFrame(model_descriptions)
416     # model_descriptions = extract_cpc_model_attributes(['/home/julian/Downloads/Github/contrastive-predictive-coding/
417     # models_symbolic_links/train/correct-age/no_class_weights/cpc'])
418     df = pd.read_csv('/home/julian/Desktop/cpc-descriptions.csv')
419     df = df.dropna()

```

```

418     #df.to_csv('/home/julian/Desktop/cpc-descriptions.csv', index=False)
419     df.index.name = "Model"
420     cols = list(df.columns)
421     #df['micro_auc'] = np.random.randn(len(df))
422     parallel_plot(df.reset_index(), list(set(df.columns) - {'micro-auc', 'macro-auc'}), 'micro-auc', curved=True, save
423     ='~/home/julian/Desktop/parallel-cpc-micro.png')
424     parallel_ploc(df.reset_index(), list(set(df.columns) - {'macro-auc', 'micro-auc'}), 'macro-auc', curved=True, save
425     ='~/home/julian/Desktop/parallel-cpc-macro.png')
426     #plt.show()
427     savepath = '/home/julian/Desktop/'
428     # plotm.plot_parallel_coordinates(df, 'micro-auc', savepath + 'micro.png',
429     #                                     drop_columns=['use_class_weights'], put_last_columns=['micro-auc'])
430     # plotm.plot_parallel_coordinates(df, 'macro-auc', savepath + 'macro.png',
431     #                                     drop_columns=['use_class_weights'], put_last_columns=['macro-auc'])
432     # plot_parallel_coordinates(df.reset_index(), 'Model')
433     # parallel_coordinates_custom(df.reset_index(), 'Model')
434     # fig = px.parallel_coordinates(df.reset_index(), color_continuous_scale=px.colors.diverging.Tealrose,
435     #                               color_continuous_midpoint=2)
436     # fig.write_image('tmp.png')

```

B.10.3.3 util -> utility -> full_class_name.py (code)

```

1 # https://stackoverflow.com/a/2020083/3620718
2 def fullname(o):
3     # o.__module__ + "." + o.__class__.__qualname__ is an example in
4     # this context of H.L. Mencken's "neat, plausible, and wrong."
5     # Python makes no guarantees as to whether the __module__ special
6     # attribute is defined, so we take a more circumspect approach.
7     # Alas, the module name is explicitly excluded from __qualname__
8     # in Python 3.
9
10    module = o.__class__.__module__
11    if module is None or module == str.__class__.__module__:
12        return o.__class__.__name__ # Avoid reporting __builtin__
13    else:
14        return module + '.' + o.__class__.__name__

```

B.10.3.4 util -> utility -> layer_calculations.py (code)

```

1 import torch
2 from torch.nn import functional as F
3
4 import util.visualize.layer_visualization as lv
5
6
7 def calc_conv1ds_output_length(input_length, kernel_sizes: list, paddings: list = None, dilations: list = None,
8                                 strides: list = None):
9     assert (paddings is None or len(paddings) == len(kernel_sizes)) and (
10            dilations is None or len(dilations) == len(kernel_sizes)) and (
11            strides is None or len(strides) == len(kernel_sizes))
12
13     if paddings is None:
14         paddings = [0] * len(kernel_sizes)
15     if dilations is None:
16         dilations = [1] * len(kernel_sizes)
17     if strides is None:
18         strides = [1] * len(kernel_sizes)
19
20     output_length = input_length
21     for padding, dilation, kernel_size, stride in zip(paddings, dilations, kernel_sizes, strides):
22         print('In', output_length, end='->')
23         output_length = _calc_conv1d_output_length(output_length, padding, dilation, kernel_size, stride)
24         print('out', output_length)
25
26     return output_length
27
28
29
30 def _calc_conv1d_output_length(input_length, padding, dilation, kernel_size, stride):
31     return int((input_length + 2 * padding - dilation * (kernel_size - 1) - 1) / stride + 1)
32
33
34
35 def calc_conv1ds_input_length(output_length, kernel_sizes: list, paddings: list = None, dilations: list = None,
36                               strides: list = None):
37     assert (paddings is None or len(paddings) == len(kernel_sizes)) and (
38            dilations is None or len(dilations) == len(kernel_sizes)) and (
39            strides is None or len(strides) == len(kernel_sizes))
40
41     if paddings is None:
42         paddings = [0] * len(kernel_sizes)
43     if dilations is None:
44         dilations = [1] * len(kernel_sizes)
45     if strides is None:
46         strides = [1] * len(kernel_sizes)
47
48     input_length = output_length
49     for padding, dilation, kernel_size, stride in reversed(list(zip(paddings, dilations, kernel_sizes, strides))):
50         print('In', input_length, end='->')

```

```

44     input_length = _calc_convld_input_length(input_length, padding, dilation, kernel_size, stride)
45     print('out', input_length)
46     return input_length
47
48
49 def calc_convlds_input_length_range(output_length, kernel_sizes: list, paddings: list = None, dilations: list = None,
50                                     strides: list = None):
51     assert (paddings is None or len(paddings) == len(kernel_sizes)) and (
52            dilations is None or len(dilations) == len(kernel_sizes)) and (
53            strides is None or len(strides) == len(kernel_sizes))
54     if paddings is None:
55         paddings = [0] * len(kernel_sizes)
56     if dilations is None:
57         dilations = [1] * len(kernel_sizes)
58     if strides is None:
59         strides = [1] * len(kernel_sizes)
60     in_min = in_max = output_length
61     for padding, dilation, kernel_size, stride in reversed(list(zip(paddings, dilations, kernel_sizes, strides))):
62         print('In', in_min, in_max, end='->')
63         in_min = _calc_convld_input_length(in_min, padding, dilation, kernel_size, stride)
64         in_max = _calc_convld_input_length(in_max + ((stride - 1) / stride), padding, dilation, kernel_size,
65                                         stride) # assumes maximum possible error
65     print('out', in_min, in_max)
66     return in_min, in_max
67
68
69
70 def _calc_convld_input_length(output_length, padding, dilation, kernel_size, stride):
71     return int((output_length - 1) * stride - 2 * padding + dilation * (kernel_size - 1) + 1)
72
73
74 def calc_convld_input_receptive_field(output_length, kernel_sizes: list, paddings: list = None, dilations: list = None,
75                                     strides: list = None, weights='balanced'):
76     assert (paddings is None or len(paddings) == len(kernel_sizes)) and (
77            dilations is None or len(dilations) == len(kernel_sizes)) and (
78            strides is None or len(strides) == len(kernel_sizes))
79     if paddings is None:
80         paddings = [0] * len(kernel_sizes)
81     if dilations is None:
82         dilations = [1] * len(kernel_sizes)
83     if strides is None:
84         strides = [1] * len(kernel_sizes)
85     if type(output_length) == type(1):
86         receptive_out = torch.ones((1, 1, output_length))
87     else:
88         receptive_out = output_length # Assume array
89     for padding, dilation, kernel_size, stride in reversed(list(zip(paddings, dilations, kernel_sizes, strides))):
90         print('In:', receptive_out, end='->')
91         receptive_out = _calc_convld_input_receptive_field(receptive_out=receptive_out, kernel_size=kernel_size,
92                                                       padding=padding, dilation=dilation, stride=stride,
93                                                       kernel_weights=weights)
94     # print('out:', receptive_out)
95     print(receptive_out)
96     return receptive_out.squeeze().numpy()
97
98
99 def _calc_convld_input_receptive_field(receptive_out, kernel_size, padding, dilation, stride, kernel_weights='balanced',
100                                         pad_mode='both'):
101     if kernel_weights is None or kernel_weights == 'balanced': # Good to see how many each value is used in the final
102         output
103         kernel_weights = torch.ones(1, 1, kernel_size)
104     elif kernel_weights == 'left': # Good to see start of new value in outputmap
105         kernel_weights = torch.zeros(1, 1, kernel_size)
106         kernel_weights[0, 0, 0] = 1
107     elif kernel_weights == 'right': # Good to see end of new value in outputmap
108         kernel_weights = torch.zeros(1, 1, kernel_size)
109         kernel_weights[0, 0, -1] = 1
110     elif kernel_weights == 'center':
111         kernel_weights = torch.zeros(1, 1, kernel_size)
112         kernel_weights[0, 0, kernel_size // 2] = 1
113     elif kernel_weights == 'tails': # Good to see end of new value in outputmap
114         kernel_weights = torch.zeros(1, 1, kernel_size)
115         kernel_weights[0, 0, -1] = 1
116     receptive_input_map = F.pad(
117         F.conv_transposed(input=receptive_out, weight=kernel_weights, stride=stride, dilation=dilation),
118         [padding, padding] if pad_mode == 'both' else [padding, 0], value=0)
119
120     return receptive_input_map
121
122
123 if __name__ == '__main__':
124     print(calc_convlds_output_length(4500, kernel_sizes=[3,3,3,3,3], strides=[3, 2, 2, 1, 1]))
125
126     print(calc_convlds_input_length(1, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2]))
127     # To quickly run calculations:
128     # print(calc_convlds_output_length(512, kernel_sizes=[3,3,3,3,3], dilations=[1,3,9,27,27*3,27*3*3], paddings
129     # =[3,3,3,3,3]))
129     # print(calc_convlds_output_length(465, kernel_sizes=[10, 8, 4, 4, 4], strides = [5, 4, 2, 2, 2]))
```

```

130 # print(calc_convlds_output_length(4500, kernel_sizes=[11, 7, 5, 5, 3], strides=[3, 2, 2, 1, 1]))
131 # print(calc_convlds_output_length(4500, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2]))
132 # print(calc_convlds_output_length(20480, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2]))
133 # print(calc_convlds_input_length(27, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2]))
134 # # print(calc_convlds_input_length(1, kernel_sizes = [8, 6, 3, 3, 3], strides = [4, 2, 1, 1, 1], dilations = [1,
1, 1, 3, 9]))
135 # # print(calc_convlds_input_length(1,kernel_sizes = [7, 3, 3, 3, 3], strides = [2, 1, 1, 1, 1], dilations = [1,
1, 3, 9, 27]))
136 # # print(calc_convlds_output_length(9500, kernel_sizes=[7, 3, 3, 3, 3], strides=[2, 1, 1, 1, 1], dilations=[1, 1,
3, 9, 27]))
137 # print(calc_convlds_output_length(4625, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2]))
138 # print(calc_convlds_output_length(5000, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2]))
139 # print(calc_convlds_input_length(1, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2]))
140 #
141 # rf = calc_convld_input_receptive_field(1, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2],
#                                         weights='balanced')
142 # lv.plot_receptivefield_plot_repeated(rf, 'standard1', repeat_shift=[160])
143 #
144 # print('experiment', calc_convlds_output_length(4500, kernel_sizes=[7, 3, 2, 2, 2], strides=[1, 1, 1, 2, 2],
#                                                 dilations=[1, 7, 7 * 3, 7 * 3 * 2, 7 * 3 * 2 * 2]))
145 # rf = calc_convld_input_receptive_field(4333, kernel_sizes=[7, 3, 2, 2, 2], strides=[1, 1, 1, 1, 1],
#                                         dilations=[1, 7, 7 * 3, 7 * 3 * 2, 7 * 3 * 2 * 2],
#                                         weights='balanced')
146 # lv.plot_receptivefield_plot(rf, 'experiment')
147 # rf = calc_convld_input_receptive_field(1, kernel_sizes=[7, 3, 2, 2, 2], strides=[1, 1, 1, 1, 1],
#                                         dilations=[1, 7, 7 * 3, 7 * 3 * 2, 7 * 3 * 2 * 2],
#                                         weights='balanced')
148 # lv.plot_receptivefield_plot(rf, 'experiment')
149 # rf = calc_convld_input_receptive_field(1, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2],
#                                         weights='balanced')
150 # lv.plot_receptivefield_plot(rf, 'experiment')
151 # rf = calc_convld_input_receptive_field(1, kernel_sizes=[7, 3, 2, 2, 2], strides=[1, 1, 1, 1, 1],
#                                         dilations=[1, 7, 7 * 3, 7 * 3 * 2, 7 * 3 * 2 * 2],
#                                         weights='balanced')
152 # lv.plot_receptivefield_plot(rf, 'experiment')
153 # print(calc_convlds_input_length(1, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2]))
154 # # print(calc_convlds_input_length(57, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2]))
155 # # print(calc_convlds_output_length(9500, kernel_sizes=[3, 3, 3, 3], strides=[1, 1, 1, 1], dilations=[1, 3,
9, 27, 27*3]))
156 # # print(calc_convlds_input_length(1, kernel_sizes=[3, 3, 3, 3, 3], strides=[1, 1, 1, 1, 1],
#                                         dilations=[1, 3, 9, 27, 27 * 3]))
157 # # print(calc_convld_input_receptive_field(1, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2]),
# rf = calc_convld_input_receptive_field(20480, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2],
#                                         weights='balanced')
158 # lv.plot_receptivefield_plot(rf)
159 #
160 # rf = calc_convld_input_receptive_field(torch.tensor([[1., 1.]]), kernel_sizes=[10, 8, 4, 4, 4],
#                                         strides=[5, 4, 2, 2, 2], weights='balanced')
161 # lv.plot_receptivefield_plot(rf)
162 # # rfr = calc_convld_input_receptive_field(57, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2],
#                                         weights='balanced')
163 # # rfl = calc_convld_input_receptive_field(57, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2],
#                                         weights='left')
164 # # lv.plot_multiple_receptivefield_plot(rfr, rfl)
165 # # rf = calc_convld_input_receptive_field(1, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2],
#                                         weights='balanced')
166 # # lv.plot_receptivefield_plot(rf, 'Receptive field for 1 pixel in output')
167 # # rf = calc_convld_input_receptive_field(57, kernel_sizes=[10, 8, 4, 4, 4], strides=[5, 4, 2, 2, 2],
#                                         weights='balanced')
168 # # lv.plot_receptivefield_plot(rf, 'Receptive field for 57 pixel in output')
169 # # rf = calc_convld_input_receptive_field(1, kernel_sizes=[3, 3, 3, 3], strides=[1, 1, 1, 1, 1], dilations=[1,
3, 9, 27, 27*3],
170 #                                         weights='balanced')
171 # # lv.plot_receptivefield_plot(rf, 'Receptive field for 1 pixel in output')
172 # # rf = calc_convld_input_receptive_field(57, kernel_sizes=[3, 3, 3, 3, 3], strides=[1, 1, 1, 1, 1],
#                                         dilations=[1, 3, 9, 27, 27 * 3],
173 #                                         weights='balanced')
174 # # lv.plot_receptivefield_plot(rf, 'Receptive field for 57 pixel in output')
175 # # rf = calc_convld_input_receptive_field(1, kernel_sizes=[3, 3, 3, 3], strides=[1, 1, 1, 1, 1], dilations=[1,
3, 9, 27, 27*3],
176 #                                         weights='balanced')
177 # #
178 # # lv.plot_receptivefield_plot(rf, 'Receptive field for 1 pixel in output')
179 # # rf = calc_convld_input_receptive_field(57, kernel_sizes=[3, 3, 3, 3, 3], strides=[1, 1, 1, 1, 1],
#                                         dilations=[1, 3, 9, 27, 27 * 3],
180 #                                         weights='balanced')
181 # # lv.plot_receptivefield_plot(rf, 'Receptive field for 57 pixel in output')
182 # # rf = calc_convld_input_receptive_field(1, kernel_sizes=[3, 3, 3, 3], strides=[1, 1, 1, 1, 1], dilations=[1,
3, 9, 27, 27*3],
183 #                                         weights='balanced')
184 # #
185 # ks = 5
186 # ln = 6
187 # print(calc_convlds_output_length(4500, kernel_sizes=[ks] * ln,
#                                         dilations=[ks ** i for i in range(ln)])) # , paddings=[ks**i for i in range(ln
188 # ])
189 # print(calc_convlds_input_length(1, kernel_sizes=[ks] * ln,
#                                         dilations=[ks ** i for i in range(ln)])) # , paddings=[ks ** i for i in range(ln
189 # ])
190 # # rf1 = calc_convld_input_receptive_field(5405, kernel_sizes=[ks]*ln, dilations=[ks**i for i in range(ln)],
#                                         paddings=[ks*i for i in range(ln)], weights='balanced')
191 # rf2 = calc_convld_input_receptive_field(1, kernel_sizes=[ks] * ln, dilations=[ks ** i for i in range(ln)],
#                                         # , paddings=[ks**i for i in range(ln)]
192 #                                         weights='balanced')
193 # lv.plot_receptivefield_plot(rf2, 'Receptive field for 1 pixel in output')
194 #
195 # print(calc_convlds_output_length(9500, kernel_sizes=[ks] * ln + [1], dilations=[ks ** i for i in range(ln)] +
196 #                                         [1],
#                                         strides=ln * [1] + [729]))
197 # # rf1 = calc_convld_input_receptive_field(5405, kernel_sizes=[ks]*ln, dilations=[ks**i for i in range(ln)],
#                                         paddings=[ks*i for i in range(ln)], weights='balanced')
198 # rf2 = calc_convld_input_receptive_field(1, kernel_sizes=[ks] * ln + [1],
#                                         dilations=[ks ** i for i in range(ln)] + [1], strides=ln * [1] + [729],
#                                         weights='balanced')
199 # lv.plot_receptivefield_plot(rf2, 'Receptive field for 1 pixel in output')

```

B.10.3.5 util -> utility -> print_layer.py (code)

```

1 from torch import nn
2
3
4 class PrintLayer(nn.Module):
5     def __init__(self, name=''):
6         super().__init__()
7         self.name = name
8
9     def forward(self, x):
10        print(self.name, x.shape)
11        return x

```

B.10.3.6 util -> utility -> sparse_print.py (code)

```

1 class SparsePrint:
2     def __init__(self):
3         self.counter = 0
4
5     def print(self, messages, nmod=1):
6         self.counter += 1
7         if self.counter % nmod == 0:
8             print(messages)
9             self.counter = 0
10
11    def __call__(self, *args, **kwargs):
12        self.print(*args)

```

B.10.3.7 util -> utility -> timestamp.py (code)

```

1 import datetime
2
3
4 def string_timestamp_hours():
5     return str(datetime.datetime.now().strftime("%d_%m_%y-%H"))
6
7
8 def string_timestamp_minutes():
9     return str(datetime.datetime.now().strftime("%d_%m_%y-%H-%M"))
10
11
12 def string_timestamp_seconds():
13     return str(datetime.datetime.now().strftime("%d_%m_%y-%H-%M-%S"))

```

B.10.4 util -> visualize (folder)

B.10.4.1 util -> visualize -> layer_visualization.py (code)

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4
5 def plot_kernel_weights(kernel_weights):
6     kw = np.repeat(kernel_weights, 1000, axis=0)
7     plt.yticks([])
8     plt.imshow(kw, interpolation='none', cmap=plt.cm.get_cmap('Blues'))
9     plt.show()
10
11
12 def plot_kernel_histogram(kernel_weights):
13     unique_vals, counts = np.unique(kernel_weights, return_counts=True)
14     n_bins = np.arange(min(kernel_weights), max(kernel_weights))
15     plt.xticks(n_bins)
16     plt.xlim(min(kernel_weights) - 1, max(kernel_weights) + 1)
17     n, bin, patches = plt.hist(kernel_weights, bins=n_bins, rwidth=0.5)
18     for i in range(len(n)):
19         plt.text(bin[i], n[i], str(n[i]))
20     plt.show()
21
22
23 def plot_receptivefield_baridiagram(kernel_weights):
24     plt.bar(np.arange(len(kernel_weights)), kernel_weights) # height=h, align='center'
25     plt.show()

```

```

27
28 def plot_receptivefield_plot(kernel_weights, name=''):
29     plt.title(name)
30     plt.plot(np.arange(len(kernel_weights)), kernel_weights) # height=h, align='center'
31     plt.show()
32
33
34 def plot_receptivefield_plot_repeated(kernel_weights, name='', repeat_shift: list = None):
35     if not repeat_shift is None:
36         plt.title(name)
37         for i in range(len(repeat_shift) + 1):
38             plt.plot(np.arange(len(kernel_weights)) + sum(repeat_shift[0:i]),
39                     kernel_weights) # height=h, align='center'
40     plt.show()
41
42
43 def plot_multiple_receptivefield_plot(kernel_weights_list: list, name=''):
44     plt.title(name)
45     for kw in kernel_weights_list:
46         plt.plot(np.arange(len(kw)), kw) # height=h, align='center'
47     plt.show()

```

B.10.4.2 util -> visualize -> plot_metrics.py (code)

```

1 import os
2 from itertools import cycle
3
4 import pandas as pd
5 import numpy as np
6 import plotly.graph_objects as go
7 import plotly.express as px
8 import matplotlib.pyplot as plt
9 import seaborn
10 import torch
11 from matplotlib.font_manager import FontProperties
12 from numpy import interp
13 from sklearn.metrics import auc, ConfusionMatrixDisplay
14
15
16 def plot_roc_singleclass(tpr, fpr, roc_auc, class_name, class_i, savepath=None, plot_name='', plot_legends=True):
17     plt.figure()
18     lw = 2
19     plt.plot(fpr[class_i], tpr[class_i], color='darkorange',
20              lw=lw, label='ROC curve: ' + class_name + ' (area = %0.2f)' % roc_auc[class_i])
21     plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
22     plt.xlim([0.0, 1.0])
23     plt.ylim([0.0, 1.05])
24     plt.xlabel('False Positive Rate')
25     plt.ylabel('True Positive Rate')
26     plt.title(plot_name + '\nReceiver operating characteristic for single class: ' + class_name)
27     if plot_legends:
28         plt.legend(loc="lower right")
29     if savepath:
30         plt.savefig(os.path.join(savepath, f'roc-{class_name}.png'), bbox_inches='tight')
31     plt.show()
32
33
34 def plot_roc_multiclass(tpr, fpr, roc_auc, classes: list, selection=None, savepath=None, plot_name='',
35                         plot_legends=True):
36     n_classes = len(classes)
37     selection = range(n_classes) if selection is None else selection
38     all_fpr = np.unique(np.concatenate([fpr[i] for i in selection]))
39     lw = 1
40     # Then interpolate all ROC curves at this points
41     mean_tpr = np.zeros_like(all_fpr)
42     for i in selection:
43         mean_tpr += interp(all_fpr, fpr[i], tpr[i])
44
45     # Finally average it and compute AUC
46     mean_tpr /= n_classes
47
48     fpr["macro"] = all_fpr
49     tpr["macro"] = mean_tpr
50     roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])
51
52     # Plot all ROC curves
53     plt.figure()
54     plt.plot(fpr["micro"], tpr["micro"],
55               label='micro-average ROC curve (area = {0:0.2f})'
56               ''.format(roc_auc["micro"]),
57               color='deeppink', linestyle=':', linewidth=4)
58
59     plt.plot(fpr["macro"], tpr["macro"],
60               label='macro-average ROC curve (area = {0:0.2f})'
61               ''.format(roc_auc["macro"]),
62               color='navy', linestyle=':', linewidth=4)

```

```

63     cm = plt.get_cmap('gist_rainbow')
64     colors = cycle([cm(1. * i / n_classes) for i in selection])
65
66     for i, color in zip(selection, colors):
67         plt.plot(fpr[i], tpr[i], color=color, lw=lw,
68                   label='{0} (area = {1:0.2f})'
69                   ''.format(classes[i], roc_auc[i]))
70
71     plt.plot([0, 1], [0, 1], 'k--', lw=lw)
72     plt.xlim([0.0, 1.0])
73     plt.ylim([0.0, 1.05])
74     plt.xlabel('False Positive Rate')
75     plt.ylabel('True Positive Rate')
76     parts = plot_name.split('|')
77     plot_name = parts[0] + '\n' + '|'.join(parts[1:])
78     plt.title(plot_name + '\nReceiver operating characteristic for multi-class')
79
80     if plot_legends:
81         plt.legend(bbox_to_anchor=(1.04, 1), loc='upper left', fontsize=6)
82     if savepath:
83         plt.savefig(os.path.join(savepath, 'ROC-multiclass.png'), bbox_inches='tight')
84     plt.show()
85
86
87 def plot_precision_recall_microavg(recall, precision, average_precision, savepath=None):
88     plt.figure()
89     plt.step(recall['micro'], precision['micro'], where='post')
90
91     plt.xlabel('Recall')
92     plt.ylabel('Precision')
93     plt.ylim([0.0, 1.05])
94     plt.xlim([0.0, 1.0])
95     plt.title(
96         'Average precision score, micro-averaged over all classes: AP={0:0.2f}'
97         ''.format(average_precision["micro"]))
98
99     if savepath:
100        plt.savefig(os.path.join(savepath, 'precision-recall-microavg.png'))
101
102 def plot_precision_recall_multiclass(precision, recall, average_precision, classes, selection=None, savepath=None,
103                                     plot_name='', plot_legends=True):
104     n_classes = len(classes)
105     if selection is None:
106         selection = range(n_classes)
107     # setup plot details
108     cm = plt.get_cmap('gist_rainbow')
109     colors = cycle([cm(1. * i / n_classes) for i in selection])
110
111     plt.figure(figsize=(12, 10))
112     f_scores = np.linspace(0.2, 0.8, num=4)
113     lines = []
114     labels = []
115     for f_score in f_scores:
116         x = np.linspace(0.01, 1)
117         y = f_score * x / (2 * x - f_score)
118         l, = plt.plot(x[y >= 0], y[y >= 0], color='gray', alpha=0.2)
119         plt.annotate('f1={0:0.1f}'.format(f_score), xy=(0.9, y[45] + 0.02))
120     lines.append(l) # yes only one
121     labels.append('iso-f1 curves')
122
123     l, = plt.plot(recall["micro"], precision["micro"], color='gold', lw=2)
124     lines.append(l)
125     labels.append('micro-average Precision-recall (area = {0:0.2f})'
126                   ''.format(average_precision["micro"]))
127
128     for i, color in zip(selection, colors):
129         l, = plt.plot(recall[i], precision[i], color=color, lw=2)
130     lines.append(l)
131     labels.append('{0} (area = {1:0.2f})'
132                   ''.format(classes[i], average_precision[i]))
133
134     fig = plt.gcf()
135     plt.xlim([0.0, 1.0])
136     plt.ylim([0.0, 1.05])
137     plt.xlabel('Recall')
138     plt.ylabel('Precision')
139     parts = plot_name.split('|')
140     plot_name = parts[0] + '\n' + '|'.join(parts[1:])
141     plt.title(plot_name + '\nExtension of Precision-Recall curve to multi-class')
142
143     if plot_legends:
144         plt.legend(lines, labels, bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=6)
145     if savepath:
146         plt.savefig(os.path.join(savepath, 'precision-recall-multiclass.png'), bbox_inches='tight')
147     plt.show()
148
149 def plot_confusion_matrix(confusion_matrix: np.ndarray, classes):
150     disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=classes)
151     disp.plot()
152

```

```

153
154
155 def plot_parallel_coordinates(df: pd.DataFrame, color_column, save_to, drop_columns: list = [], put_last_columns: list = [], exclude_color_column=True):
156     """Put last columns at the end of the list if they are not in drop_columns"""
157     def _make_plotly_dict(column_name, data):
158         d = dict()
159         t = data.dtype
160         if t == bool:
161             d['range'] = [-0.5, 1.5]
162             d['tickvals'] = [True, False]
163             d['tickettext'] = ['True', 'False']
164             d['values'] = data
165         elif t == int:
166             d['range'] = [data.min(), data.max()]
167             d['tickformat'] = 'd'
168             d['values'] = data
169         elif t == str or t == object:
170             da = data.astype('category').cat
171             d['tickvals'] = da.codes
172             d['tickettext'] = da.categories
173             d['values'] = da.codes
174         else:
175             d['range'] = [data.min(), data.max()]
176             d['values'] = data
177         d['label'] = column_name
178     return d
179
180 cols = df.columns.tolist()
181 # cols = list(set(cols)-set(put_last_columns)-set(exclude_columns))+put_last_columns
182 cols = [c for c in cols if not (c in drop_columns or c in put_last_columns)] + put_last_columns
183 df = df[cols]
184 dimensions = [_make_plotly_dict(column_name, data) for column_name, data in df.iteritems() if
185                 not (exclude_color_column and column_name == color_column)]
186 fig = go.Figure(data=
187     go.Parcoords(
188         line=dict(color=df[color_column],
189                   colorscale=px.colors.diverging.Tealrose,
190                   # autocolorscale=True,
191                   showscale=True,
192                   cmin=df[color_column].min(),
193                   cmax=df[color_column].max()),
194         dimensions=dimensions
195     )
196 )
197 fig.update_traces(labelangle=-90, selector=dict(type='parcoords'))
198 fig.show()
199 if not save_to is None:
200     fig.write_image(save_to)
201
202
203 def plot_lowlabel_availability(df_groups, title, save_to, filename, data_col='micro', save_legend_seperate=False):
204     fractions = {'fewer-labels_0_001': 0.0012756005952802778,
205                  'fewer-labels_0_005': 0.009378026598634634,
206                  'fewer-labels_0_05': 0.10032362459546926,
207                  'fewer-labels_0_01': 0.02010252049228734,
208                  'fewer-labels_0_10': 0.18834006566980843,
209                  'fewer-labels_14': 0.18829282120331656,
210                  'fewer-labels_20': 0.3308057543760187,
211                  'fewer-labels_30': 0.44031842770415514,
212                  'fewer-labels_40': 0.5257600453546878,
213                  'fewer-labels_50': 0.5950912999314956,
214                  'fewer-labels_60': 0.6526823045850755,
215                  'train-test-splits': 0.7017220608036284,
216                  'train-test-splits_min_cut10': 0.015330829376609265,
217                  'train-test-splits_min_cut25': 0.037252261828833295,
218                  'train-test-splits_min_cut50': 0.06798478728178962,
219                  'train-test-splits_min_cut100': 0.116174143103489,
220                  'train-test-splits_min_cut150': 0.15470200552760258,
221                  'train-test-splits_min_cut200': 0.18864715470200552}
222 ordered_splits = [k for k, v in sorted(fractions.items(), key=lambda item: item[1])]
223
224 def get_fraction_x_for_splitsname(name):
225     return fractions[name]
226
227 fontP = FontProperties()
228 fontP.set_size('xx-small')
229 fig, ax = plt.subplots(figsize=(20, 10))
230 seaborn.set_palette("hls", len(df_groups))
231 plt.title(title)
232 for name, group in df_groups:
233     g = group.reset_index()
234     g.insert(loc=0, column='splitfraction', value=[get_fraction_x_for_splitsname(sp) for sp in g['level_0']])
235     g = g.sort_values(by='splitfraction')
236     if 'cpc' in name.lower():
237         ax.plot(g['splitfraction'], g[data_col], '--o', label=name)
238     else:
239         ax.plot(g['splitfraction'], g[data_col], '-o', label=name)
240 plt.ylabel('average AUC score')
241
242 plt.xlabel('fraction of files used (i.r.t. all files)')

```

```

243 handles, labels = ax.get_legend_handles_labels()
244 if save_legend_seperate:
245     legend = plt.legend(handles, labels, loc=3, framealpha=1, frameon=False)
246     export_legend(legend, save_to, 'legend-' + filename)
247 else:
248     ax.legend(handles, labels, loc='lower right', prop=fontP, handlelength=3) # bbox_to_anchor=(1.05, 1)
249 fig.savefig(os.path.join(save_to, filename), bbox_inches='tight')
250 plt.show()
251
252
253 def export_legend(legend, save_to, filename="legend.png"):
254     fig = legend.figure
255     fig.canvas.draw()
256     bbox = legend.get_window_extent().transformed(fig.dpi_scale_trans.inverted())
257     fig.savefig(os.path.join(save_to, filename), dpi="figure", bbox_inches=bbox)
258
259
260 def plot_prediction_scatterplots(labels, pred, model_thresholds, filename=None, save=False, show=True):
261     binary_pred = pred[0] >= model_thresholds
262     diff = (pred[0] - model_thresholds).numpy()
263     x = np.arange(len(model_thresholds))
264     fig, (ax1, ax2, ax3) = plt.subplots(1, 3, sharex=True, figsize=(15, 5))
265     title = "Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].numpy()))
266     # title += "\n Predicted classes (binary): " + ", ".join(map(str, np.nonzero(binary_pred.numpy()) [0]))
267
268     fig.suptitle(title)
269     ##First Plot
270     ax1.set_aspect = 'equal'
271     ax1.scatter(x[binary_pred], pred[0][binary_pred], c='yellow', label='true')
272     ax1.scatter(x[~binary_pred], pred[0][~binary_pred], c='purple', label='false')
273     ax1.set_xlabel('Class Nr.')
274
275     ax1.set_ylabel('Model output')
276     for class_i in x:
277         ax1.annotate(str(class_i), (class_i, pred[0][class_i]))
278     ax1.legend()
279     ##Second Plot
280     ax2.set_aspect = 'equal'
281     ax2.scatter(x[binary_pred], diff[binary_pred], c='yellow', label='true')
282     ax2.scatter(x[~binary_pred], diff[~binary_pred], c='purple', label='false')
283     ax2.set_xlabel('Class Nr.')
284     ax2.set_ylabel('model output minus specific class thresholds')
285     for class_i in x:
286         ax2.annotate(str(class_i), (class_i, diff[class_i]))
287     ax2.legend()
288     ##Third Plot
289     normed_pos = diff / (1 - model_thresholds)
290     normed_neg = diff / model_thresholds
291     # probs =
292     probs = (np.where(diff >= 0, normed_pos, normed_neg) + 1) / 2
293     ax3.scatter(x[binary_pred], probs[binary_pred], c='yellow', label='true')
294     ax3.scatter(x[~binary_pred], probs[~binary_pred], c='purple', label='false')
295     ax3.set_ylim(top=1.05)
296
297     ax3.set_aspect = 'equal'
298     ax3.set_xlabel('Class Nr.')
299     ax3.set_ylabel(r'"Probabilities" obtained by weighting output with threshold$^{-1}$')
300     for class_i in x:
301         ax3.annotate(str(class_i), (class_i, probs[class_i]))
302     ax3.legend()
303     filename = filename or ""
304     fig.tight_layout()
305     if save:
306         plt.savefig(filename + 'scatterplot-predictions(combined).png', dpi=fig.dpi)
307     if show:
308         plt.show()
309     plt.close(fig)
310
311 if __name__ == '__main__':
312     all_df = pd.read_csv('/home/julian/Desktop/All_attributes_with_scores.csv', index_col=0, parse_dates=['train timestamp'])
313     all_df = all_df.dropna(subset=['train timestamp', 'Uses Classweights'])
314     all_df = all_df[['train timestamp', 'Model Path'] + [c for c in all_df.columns if not c in ['train timestamp', 'Model Path', 'micro', 'macro']] + ['macro', 'micro']]
315     all_df[['Train Normalization Function', 'Test Normalization Function']] = all_df[['Train Normalization Function', 'Test Normalization Function']].apply(lambda s:s.str.replace("", ""))
316     all_df[['model']] = all_df[['model']].apply(lambda s:s.str.replace(":\\d*", "", regex=True))
317     all_df = all_df.drop(columns=['train timestamp'])
318     cpc_df = all_df[all_df['normalizes latents'].isin([False, True])].dropna(axis=1, how='all')
319     #print(cpc_df)
320     plot_parallel_coordinates(cpc_df, 'macro', save_to='/home/julian/Downloads/Multi-Download/lowlabel/parcoords-cpc.png', drop_columns=['Model Path', 'timesteps in', 'timesteps out', "Train Normalization Function", "Test Normalization Function", 'Train splitsfile', 'Pretrain Epochs'], )

```

B.10.4.3 util -> visualize -> timeseries_to_image_converter.py (code)

```

1 import io
2 from typing import Optional, Dict, Any, List
3
4 import av
5 import matplotlib.pyplot as plt
6 import numpy as np
7 import torch
8 from PIL import Image
9 from torchvision.io import write_video
10 import seaborn as sns
11 import matplotlib.patches as mpatches
12 from matplotlib.colors import Normalize
13
14
15 # from torch.nn.functional import relu
16 from util.data import ecg_datasets3
17
18
19 class VideoWriter():
20     def __init__(self, filename: str, fps: float, video_codec: str = "libx264",
21                  options: Optional[Dict[str, Any]] = None):
22         self.filename = filename
23         self.fps = fps
24         if isinstance(fps, float):
25             self.fps = np.round(fps)
26         self.video_codec = video_codec
27         self.options = options
28         self.is_open = False
29
30     def tensor_to_video_continuous(self, video_array: torch.Tensor, convert_timeseries=False) -> None:
31         """
32         https://pytorch.org/docs/stable/_modules/torchvision/io.html#VideoReader
33         """
34         video_array = torch.as_tensor(video_array, dtype=torch.uint8).numpy()
35         if convert_timeseries:
36             video_array = timeseries_to_image(video_array)
37         if not self.is_open:
38             self._container = av.open(self.filename, mode="w")
39             self._stream = self._container.add_stream(self.video_codec, rate=self.fps)
40             self._stream.width = video_array.shape[2]
41             self._stream.height = video_array.shape[1]
42             self._stream.pix_fmt = "yuv420p" if self.video_codec != "libx264rgb" else "rgb24"
43             self._stream.options = self.options or {}
44             self.is_open = True
45
46         for img in video_array:
47             frame = av.VideoFrame.from_ndarray(img, format="rgb24")
48             frame.pict_type = "NONE"
49             for packet in self._stream.encode(frame):
50                 self._container.mux(packet)
51
52     def close(self):
53         for packet in self._stream.encode():
54             self._container.mux(packet)
55         self._container.flush()
56         self._container.close()
57         self.is_open = False
58
59
60     def tensor_to_video(filename: str,
61                        video_array: torch.Tensor,
62                        fps: float,
63                        video_codec: str = "libx264",
64                        options: Optional[Dict[str, Any]] = None):
65         return write_video(filename, video_array, fps, video_codec, options)
66
67
68     def timeseries_to_image(data: torch.Tensor, title='ECG-data visualization',
69                            ground_truth: list = None, filename: str = None, show=False, save=True):
70         if len(data.shape) == 2:
71             data = data[np.newaxis]
72         batches, width, height = data.shape
73         if not filename:
74             filename = title.replace(' ', '_')+'.png'
75         for batch in range(batches):
76             fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30,20))
77             plt.xlim((0, width))
78             plt.ylim((0, 1))
79
80             if not ground_truth is None:
81                 title += '| Correct classes: ' + ", ".join(map(str, np.nonzero(ground_truth[batch].numpy())[0])) + ". "
82             fig.suptitle(title, fontsize=24)
83             fig.tight_layout()
84             for i, ax in enumerate(axs):
85                 ax.set_xlim((0, width))
86                 ax.axis('off')
87                 d = data[batch, :, i]
88                 ax.set_ylim((d.min()-0.1, d.max()+0.1))
89                 ax.plot(range(width), d, color='red')
90                 ax.axis('on')
```

```

91     ax.spines['top'].set_visible(False)
92     ax.spines['left'].set_visible(False)
93     ax.spines['right'].set_visible(False)
94
95     axs[-1].get_yaxis().set_visible(False)
96     plt.xlabel('Time (500 steps = 1 second)', fontsize=18)
97
98     if save:
99         plt.savefig(f"{{filename.split('.')[{-2}]}{{batch}}.{{filename.split('.')[{-1}]}}", dpi=fig.dpi)
100    if show:
101        plt.show()
102    plt.close()
103    print('Finished')
104
105
106 def timeseries_to_image_with_gradient(data: torch.Tensor, labels: torch.Tensor, grad: torch.Tensor,
107                                         pred: torch.Tensor = None, model_thresholds=None, grad_alteration='none',
108                                         title=None, class_name=None, filenames: list = None, show=False, save=True):
109     batches, width, height = data.shape
110     for batch in range(batches):
111         cmap_green = sns.light_palette("seagreen", as_cmap=True)
112         fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
113         plt.xlim((0, width))
114         plt.ylim((0, 1))
115         if grad_alteration is None or grad_alteration == 'none':
116             gradient = grad[batch]
117         elif grad_alteration == 'abs':
118             gradient = grad[batch].abs()
119         elif grad_alteration == 'relu':
120             gradient = torch.nn.functional.relu(grad[batch]) ** 2 # all values >= 0
121             grad_norm = (gradient - gradient.min()) / (gradient.max() - gradient.min())
122             title = title or f'Gradient visualization for class:{(class_name)} as label\n'
123
124         if not labels is None:
125             title += "Correct classes: " + ", ".join(map(str, np.nonzero(labels[batch].numpy())[0])) + ". "
126         if not (pred is None or model_thresholds is None):
127             binary_pred = pred[batch] >= model_thresholds
128             title += "Predicted classes: " + ", ".join(map(str, np.nonzero(binary_pred.numpy())[0])) + ". "
129         fig.suptitle(title)
130         fig.tight_layout()
131
132         for i, ax in enumerate(axs):
133             ax.set_xlim((0, width))
134             ax.set_ylim((-0.1, 1.1))
135             ax.axis('off')
136             d = data[batch, :, i]
137             g = grad_norm[:, i:i + 1].T
138             ax.plot(range(width), d, color='red')
139             ax.autoscale(False)
140             ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_green)
141
142         if save:
143             plt.savefig(
144                 filenames[batch] + '-class:' + str(class_name) + '-alt:' + grad_alteration + '-gradient-vis.png',
145                 dpi=fig.dpi)
146         if show:
147             plt.show()
148         plt.close()
149
150
151 def timeseries_to_image_with_gradient_joined(data: torch.Tensor, labels: torch.Tensor, grad_list: List[torch.Tensor],
152                                              pred: torch.Tensor = None, model_thresholds=None, grad_alteration='none',
153                                              cutoff=0.1, title=None, class_name_list=None, filenames: list = None,
154                                              show=False, save=True):
155     batches, width, height = data.shape
156     n_preds = len(grad_list)
157     base_colors = sns.color_palette("hls", n_preds)
158     cmaps = [sns.light_palette(c, as_cmap=True) for c in base_colors]
159     for i in range(len(cmaps)):
160         cmaps[i].set_gamma(1.)
161         cmaps[i]._lut[0, :] = 0. # Set all initial values to transparent
162     for batch in range(batches):
163         fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
164         plt.xlim((0, width))
165         plt.ylim((0, 1))
166         title = title or f'Gradient visualization for class:{(class_name_list)} as label\n'
167         title += f"Gradient Alteration: '{grad_alteration}'\n"
168         if not labels is None:
169             title += "Correct classes: " + ", ".join(map(str, np.nonzero(labels[batch].numpy())[0])) + ". "
170         if not (pred is None or model_thresholds is None):
171             binary_pred = pred[batch] >= model_thresholds
172             title += "Predicted classes: " + ", ".join(map(str, np.nonzero(binary_pred.numpy())[0])) + ". "
173         fig.suptitle(title)
174         fig.tight_layout()
175         for i, ax in enumerate(axs):
176             ax.set_xlim((0, width))
177             ax.set_ylim((-0.1, 1.1))
178             ax.axis('off')
179             d = data[batch, :, i]
180             ax.plot(range(width), d, color='red')
legend_handles = []

```

```

181     grad_norms = []
182     gradients = []
183     color_values = []
184     for n in range(n_preds):
185         if grad_alteration == 'abs':
186             gradient = grad_list[n][batch].abs()
187         elif grad_alteration == 'relu':
188             gradient = torch.nn.functional.relu(grad_list[n][batch]) # all values >= 0
189         elif grad_alteration == 'abs_neg':
190             gradient = grad_list[n][batch].abs()
191         elif grad_alteration == 'relu_neg':
192             gradient = torch.nn.functional.relu(-grad_list[n][batch]) # all values >= 0
193         else:
194             gradient = grad_list[n][batch]
195         gradients.append(gradient.numpy())
196         grad_norm = (gradient - gradient.min()) / (gradient.max() - gradient.min()).numpy()
197         grad_norm[grad_norm < cutoff] = 0.
198         grad_norms.append(grad_norm)
199         color_values.append(cmaps[n](grad_norm))
200         legend_handles.append(mpatches.Patch(color=base_colors[n], label=f"Class: {class_name_list[n]}"))
201         #####NEXT IDEA: use custom color map (choose depending on argmax over n_preds)
202     grad_norms = np.stack(grad_norms)
203     gradients = np.stack(gradients)
204     color_values = np.stack(color_values)
205     # ix = np.argmax(grad_norms, axis=0)
206     ix = np.argmax(gradients, axis=0)
207
208     print(grad_norms.shape, ix.shape, color_values.shape)
209     for i, ax in enumerate(axes):
210         g = color_values[ix[:, i], np.arange(width), i][np.newaxis, :, :]
211         ax.autoscale(False)
212         ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', alpha=1, interpolation='none')
213     fig.legend(handles=legend_handles)
214     if save:
215         plt.savefig(
216             filenames[batch] + '-class:' + str(class_name_list) + '-alt:' + grad_alteration + '-gradient-vis.png',
217             dpi=fig.dpi)
218     if show:
219         plt.show()
220     plt.close()
221
222 def timeseries_to_image_with_gradient_cam(data: torch.Tensor, labels: torch.Tensor, grad_list: List[torch.Tensor],
223                                            pred: torch.Tensor = None, model_thresholds=None, cut_off=0.4, title=None,
224                                            class_name_list=None, filenames: list = None,
225                                            batches, width, height = data.shape
226                                            show=False, save=True):
227     n_preds = len(grad_list)
228     base_colors = sns.color_palette("hls", n_preds)
229     cmaps = [sns.light_palette(c, as_cmap=True) for c in base_colors]
230     for batch in range(batches):
231         fig, axes = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
232         plt.xlim((0, width))
233         plt.ylim((0, 1))
234         title = title or f'Gradient visualization for class:{class_name_list} as label\n'
235         if not labels is None:
236             title += "Correct classes: " + ", ".join(map(str, np.nonzero(labels[batch].numpy())[0])) + ". "
237         if not (pred is None or model_thresholds is None):
238             binary_pred = pred[batch] >= model_thresholds
239             title += "Predicted classes: " + ", ".join(map(str, np.nonzero(binary_pred.numpy())[0])) + ". "
240         fig.suptitle(title)
241         fig.tight_layout()
242         for i, ax in enumerate(axes):
243             ax.set_xlim((0, width))
244             ax.axis('off')
245             d = data[batch, :, i]
246             ax.set_ylim((d.min()-0.1, d.max()+0.1))
247             ax.plot(range(width), d, color='red')
248             legend_handles = []
249             for n in range(n_preds):
250                 legend_handles.append(mpatches.Patch(color=base_colors[n], label=f"Class: {class_name_list[n]}"))
251             for i, ax in enumerate(axes):
252                 ax.autoscale(False)
253                 bottom, top = ax.get_ylim()
254                 r = (top-bottom)/n_preds
255                 if len(grad_list[n][batch].shape) == 1:
256                     g = grad_list[n][batch][np.newaxis, :]
257                     ax.imshow(g, extent=[0, width, top-r*(n+1), top-r*n], cmap=cmaps[n], aspect='auto', alpha=0.8,
258                               interpolation='bilinear')
259                 elif len(grad_list[n][batch].shape) == 2:
260                     g = grad_list[n][batch][:, i:i+1].T
261                     g[g<cut_off]=0
262                     ax.imshow(g, extent=[0, width, top-r*(n+1), top-r*n], cmap=cmaps[n], norm=Normalize(vmin=0, vmax
263 =1), aspect='auto', alpha=1, interpolation='bilinear')
264                 else:
265                     print("Wrong shape for gradient:", grad_list[n][batch].shape)
266             fig.legend(handles=legend_handles)
267             if save:
268                 plt.savefig(
269                     filenames[batch] + '-class:' + str(class_name_list) + '-gradient-cam.png',
270                     dpi=fig.dpi)

```

```

268     if show:
269         plt.show()
270     plt.close()
271     print('Finished')
272
273
274 def kernel_to_image(layer_name, layer_weights: torch.Tensor):
275     out_channels, in_channels, kernel_size = layer_weights.shape
276
277     fig, axs = plt.subplots(1, in_channels, figsize=(out_channels // 2, in_channels // 2))
278     fig.suptitle(layer_name)
279     mins = layer_weights.min(dim=2)[0].unsqueeze(2)
280     maxs = layer_weights.max(dim=2)[0].unsqueeze(2)
281     layer_weights = (layer_weights - mins) / (maxs - mins)
282     for index, ax in enumerate(axs):
283         ax.imshow(layer_weights[:, index, :], cmap='gray')
284         ax.set_xticks([])
285         ax.set_yticks([])
286     plt.show()
287
288
289 if __name__ == '__main__': # Usage example
290     # model_f = '../models/18_01_21-14/baseline_modelstate_epoch200.pt'
291     # model_state_dict = torch.load(model_f) ['model_state_dict']
292     # print(model_state_dict.keys())
293     # for k in model_state_dict.keys():
294     #     if 'weight' in k:
295     #         conv = model_state_dict[k].cpu()
296     #         print('layer weight shape:', conv.shape)
297     #         kernel_to_image(k, conv)
298
299     path_without_ext = '/media/julian/data/data/ECG/ptbxl_challenge/HR14099'
300     d = ecg_datasets3.ECGChallengeDatasetBaseline(None, None)
301     data = ecg_datasets3.normalize_minmax_scaling(d._read_recording_file(path_without_ext), axis=0)
302     labels = d._read_header_file(path_without_ext)
303     print(labels)
304     timeseries_to_image(data, 'ECG-data of patient HR14099 in the PTBXL dataset', filename='/home/julian/Documents/
    projekt-master/bilder/ecg-example-minmax.png', show=True, save=True)

```

B.10.5 util -> store_models.py (code)

```

1 import json
2 import os
3 import pickle
4
5 import torch
6
7 from util.utility.full_class_name import fullname
8
9
10 def save_model_checkpoint(output_path, epoch, model, optimizer=None, name=""):
11     print("saving model at epoch:", epoch)
12     checkpoint = {
13         'epoch': epoch,
14         'model_state_dict': model.state_dict()
15     }
16     if not optimizer is None:
17         checkpoint['optimizer_state_dict'] = optimizer.state_dict()
18         torch.save(checkpoint, os.path.join(output_path, name + '_checkpoint_epoch_{}.pt'.format(epoch)))
19
20 def save_model_state_dict_checkpoint(output_path, epoch, model_statedict, optimizer=None, name="", additional_name=""):
21     print("saving model at epoch:", epoch)
22     checkpoint = {
23         'epoch': epoch,
24         'model_state_dict': model_statedict
25     }
26     if not optimizer is None:
27         checkpoint['optimizer_state_dict'] = optimizer.state_dict()
28         torch.save(checkpoint, os.path.join(output_path, f'{name}_checkpoint_epoch_{epoch}{additional_name}.pt'))
29
30
31 def save_model_architecture(output_path, model, name=""):
32     print("Saving full model...")
33     torch.save(model, os.path.join(output_path, name + '_full_model.pt'))
34     save_model_architecture_text_only(output_path, model, name)
35
36
37 def save_model_architecture_text_only(output_path, model, name=""):
38     with open(os.path.join(output_path, 'model_arch.txt'), 'w') as f:
39         print(name, file=f)
40         print(fullname(model), file=f)
41         print(model, file=f)
42
43
44 def save_model_variables_text_only(output_path, model, name=""):

```

```

45     with open(os.path.join(output_path, 'model_variables.txt'), 'w') as f:
46         print(name, file=f)
47         print(fullname(model), file=f)
48         print(json.dumps(extract_params_from_model(model), sort_keys=True, indent=2), file=f)
49
50
51 def load_model_architecture(full_model_file, device_id='cuda:0'): #
52     try:
53         model = torch.load(full_model_file, map_location=device_id)
54     except ModuleNotFoundError as e:
55         print(e)
56         return None
57     return model
58
59
60 def load_model_checkpoint(model_checkpoint_file, model, optimizer=None, device_id='cuda:0'):
61     checkpoint = torch.load(model_checkpoint_file, map_location=device_id)
62     model.load_state_dict(checkpoint['model_state_dict'])
63     if not optimizer is None:
64         optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
65     epoch = checkpoint['epoch']
66     model.eval()
67     return model, optimizer, epoch
68
69
70 def load_model(full_model_file, model_checkpoint_file, optimizer):
71     model = load_model_architecture(full_model_file)
72     model, optimizer, epoch = load_model_checkpoint(model_checkpoint_file, model, optimizer)
73     return model, optimizer, epoch
74
75
76 def extract_model_files_from_dir(directory):
77     print("Looking for model in abs path:", os.path.abspath(directory))
78     files = []
79     for root, dirs, dir_files in os.walk(directory, followlinks=True):
80         fm_temp, ch_temp = [], []
81         for file in dir_files:
82             if 'full_model' in file and file.endswith('.pt'):
83                 fm_temp.append(os.path.join(root, file))
84             elif 'checkpoint' in file and file.endswith('.pt'):
85                 ch_temp.append(os.path.join(root, file))
86         if len(fm_temp) > 0 and len(ch_temp) > 0:
87             files.append((fm_temp, ch_temp, root))
88     return files
89
90
91 def extract_params_from_model(obj, prefix=''):
92     if issubclass(type(obj), torch.nn.Module):
93         return extract_params_from_model(obj.__dict__, fullname(obj))
94     elif issubclass(type(obj), dict):
95         params = {}
96         for k, v in obj.items():
97             if not k.startswith('_') or k == '_modules':
98                 params.update({k: extract_params_from_model(v)})
99         return {prefix: params} if params else {}
100    elif issubclass(type(obj), list):
101        return [extract_params_from_model(l) for l in obj]
102    elif not hasattr(obj, '__dict__'):
103        return obj
104    else:
105        return None
106
107
108 @DeprecationWarning
109 def save_model_state(output_path, epoch, name=None, model=None, optimizer=None, accuracies=None, losses=None,
110                      full=False):
111     if name is None:
112         name = fullname(model)
113     with open(os.path.join(output_path, 'model_arch.txt'), 'w') as f:
114         print(fullname(model), file=f)
115         print(model, file=f)
116     if full:
117         print("Saving full model...")
118         name = 'model_full.pt'
119         torch.save(model, os.path.join(output_path, name))
120         with open(os.path.join(output_path, 'model_arch.txt'), 'w') as f:
121             print(fullname(model), file=f)
122             print(model, file=f)
123     else:
124         print("saving model at epoch:", epoch)
125         if not (model is None and optimizer is None):
126             name = name + '_modelstate_epoch' + str(epoch) + '.pt'
127             torch.save({
128                 'epoch': epoch,
129                 'model_state_dict': model.state_dict(),
130                 'optimizer_state_dict': optimizer.state_dict()
131             }, os.path.join(output_path, name))
132         if not (accuracies is None and losses is None):
133             with open(os.path.join(output_path, 'losses.pkl'), 'wb') as pickle_file:
134                 pickle.dump(losses, pickle_file)

```

```

135         with open(os.path.join(output_path, 'accuracies.pkl'), 'wb') as pickle_file:
136             pickle.dump(accuracies, pickle_file)
137
138
139     @DeprecationWarning
140     def load_model_state(model_path, model=None, optimizer=None):
141         if model is None:
142             model = torch.load(model_path)
143             epoch = 1
144         else:
145             checkpoint = torch.load(model_path)
146             if 'model_state_dict' in checkpoint:
147                 model.load_state_dict(checkpoint['model_state_dict'])
148                 if not optimizer is None:
149                     optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
150                 epoch = checkpoint['epoch']
151             else:
152                 model.load_state_dict(checkpoint)
153             epoch = 1
154
155     model.eval()
156     return model, optimizer, epoch

```

B.11 main_cpc_benchmark_test.py (code)

```

1 import argparse
2 import datetime
3 import os
4 import pickle
5 import re
6 import shutil
7 import time
8 from collections import defaultdict
9 from pathlib import Path
10 import pandas as pd
11
12 import numpy as np
13 import torch
14 from torch.optim import Adam
15 from torch.utils.data import DataLoader, ChainDataset
16
17 from util import store_models
18 from util.metrics import training_metrics
19 from external import helper_code
20 from util.data import ecg_datasets3
21 from util.store_models import load_model_checkpoint, load_model_architecture, extract_model_files_from_dir
22
23
24 def main(args):
25     np.random.seed(args.seed)
26     # os.environ['CUDA_LAUNCH_BLOCKING'] = '1'
27     torch.cuda.set_device(args.gpu_device)
28     print(f'Device set to : {torch.cuda.current_device()}. Selected was {args.gpu_device}')
29     torch.cuda.manual_seed(args.seed)
30     print(f'Seed set to : {args.seed}.')
31     print(f'Model outputpath: {args.out_path}')
32     Path(args.out_path).mkdir(parents=True, exist_ok=True)
33     with open(os.path.join(args.out_path, 'params.txt'), 'w') as cfg:
34         cfg.write(str(args))
35     # georgia = ecg_datasets3.ECGChallengeDatasetBaseline('/home/juwin106/data/georgia/WFDB', window_size=4500,
36     # pad_to_size=4500, use_labels=True)
37     # cpsc_train = ecg_datasets3.ECGChallengeDatasetBaseline('/home/juwin106/data/cpsc_train', window_size=4500,
38     # pad_to_size=4500, use_labels=True)
39     # cpsc = ecg_datasets3.ECGChallengeDatasetBaseline('/home/juwin106/data/cpsc', window_size=4500, pad_to_size=4500,
40     # use_labels=True)
41     # ptbxl = ecg_datasets3.ECGChallengeDatasetBaseline('/home/juwin106/data/ptbxl/WFDB', window_size=4500,
42     # pad_to_size=4500, use_labels=True)
43
44     if hasattr(ecg_datasets3, args.norm_fn):
45         norm_fn = getattr(ecg_datasets3, args.norm_fn)
46     else:
47         norm_fn = ecg_datasets3.normalize_std_scaling
48     #norm_fn = ecg_datasets3.normalize_minmax_scaling_different
49
50     georgia_challenge = ecg_datasets3.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/georgia_challenge/',
51                                                               window_size=args.crop_size,
52                                                               pad_to_size=args.crop_size,
53                                                               return_labels=True, return_filename=True,
54                                                               normalize_fn=norm_fn)
55     cpsc_challenge = ecg_datasets3.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/cps2018_challenge/',
56                                                               window_size=args.crop_size, pad_to_size=args.crop_size,
57                                                               return_labels=True, return_filename=True,
58                                                               normalize_fn=norm_fn)
59     cpsc2_challenge = ecg_datasets3.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/china_challenge',
60                                                               window_size=args.crop_size, pad_to_size=args.crop_size)
61
62

```

```

57                                         return_labels=True, return_filename=True,
58                                         normalize_fn=norm_fn)
59 ptbxl_challenge = ecg_datasets3.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/ptbxl_challenge',
60                                         window_size=args.crop_size, pad_to_size=args.crop_size
61                                         ,
62                                         return_labels=True, return_filename=True,
63                                         normalize_fn=norm_fn)
64 a1, b1, ptbxl_test = ptbxl_challenge.generate_datasets_from_split_file()
65 a2, b2, georgia_test = georgia_challenge.generate_datasets_from_split_file()
66 a3, b3, cpdc_challenge = cpdc_challenge.generate_datasets_from_split_file()
67 a4, b4, cpdc2_challenge = cpdc2_challenge.generate_datasets_from_split_file()
68
69 ptbxl_test.randomize_order = False
70 georgia_test.randomize_order = False
71 cpdc_challenge.randomize_order = False
72 cpdc2_challenge.randomize_order = False
73
74 b1.randomize_order = False
75 b2.randomize_order = False
76 b3.randomize_order = False
77 b4.randomize_order = False
78
79 classes = ecg_datasets3.filter_update_classes_by_count(
80     [a1, b1, ptbxl_test, a2, b2, georgia_test, a3, b3, cpdc_challenge, a4, b4, cpdc2_challenge],
81     1) # Set classes if specified in split files (filter out classes with no occurrence)
82 print(classes)
83 print(classes == {'10370003': 0, '11157007': 1, '111975006': 2, '164861001': 3, '164865005': 4, '164867002': 5,
84     '164873001': 6, '164884008': 7, '164889003': 8, '164890007': 9, '164909002': 10, '164917005':
85     11,
86     '164930006': 12, '164931005': 13, '164934002': 14, '164947007': 15, '164951009': 16,
87     '17338001': 17, '195042002': 18, '195080001': 19, '195126007': 20, '233917008': 21,
88     '251120003': 22, '251146004': 23, '251180001': 24, '251200008': 25, '251266004': 26,
89     '251268003': 27, '253352002': 28, '266249003': 29, '270492004': 30, '27885002': 31,
90     '284470004': 32, '39732003': 33, '413844008': 34, '425419005': 35, '425623009': 36,
91     '426177001': 37, '426434006': 38, '426627000': 39, '426761007': 40, '426783006': 41,
92     '427084000': 42, '427172004': 43, '427393009': 44, '428417006': 45, '428750005': 46,
93     '429622005': 47, '445118002': 48, '445211001': 49, '446358003': 50, '446813000': 51,
94     '47665007': 52, '54329005': 53, '55930002': 54, '59118001': 55, '59931005': 56, '63593006': 57,
95     '6374002': 58, '67198005': 59, '67741000119109': 60, '698252002': 61, '713422000': 62,
96     '713426002': 63, '713427006': 64, '74390002': 65, '89792004': 66})
97
98 if args.preload_fraction > 0. and getattr(b1, 'preload'):
99     print("Preloading data...")
100    b1.preload(args.preload_fraction)
101    b2.preload(args.preload_fraction)
102    b3.preload(args.preload_fraction)
103    b4.preload(args.preload_fraction)
104    ptbxl_test.preload(args.preload_fraction)
105    georgia_test.preload(args.preload_fraction)
106    cpdc_challenge.preload(args.preload_fraction)
107    cpdc2_challenge.preload(args.preload_fraction)
108
109 train_dataset_challenge = ChainDataset([a1, a2, a3, a4])
110 val_dataset_challenge = ChainDataset([b1, b2, b3, b4])
111 test_dataset_challenge = ChainDataset([ptbxl_test, georgia_test, cpdc_challenge, cpdc2_challenge])
112 # all_dataset_challenge = ChainDataset([ptbxl_challenge, georgia_challenge, cpdc_challenge, cpdc2_challenge])
113 model_folders = [
114     #'home/julian/Downloads/Github/contrastive-predictive-coding/models_symbolic_links/train/class_weights/14_05_21-15-train|(8x)cpc',
115     #'home/julian/Downloads/Github/contrastive-predictive-coding/models_symbolic_links/train/class_weights/19_05_21-16-train|cpc',
116     #'home/julian/Downloads/Github/contrastive-predictive-coding/models_symbolic_links/train/class_weights/19_05_21-17-train|cpc',
117     #'home/julian/Downloads/Github/contrastive-predictive-coding/models_symbolic_links/train/class_weights/20_05_21-18-train|(2x)bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v14+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4+bl_cnn_v5+bl_cnn_v6+bl_cnn_v8+bl_cnn_v9',
118     #'home/julian/Downloads/Github/contrastive-predictive-coding/models_symbolic_links/train/class_weights/25_05_21-13-train|bl_FCN' #used class weights
119     #'models_symbolic_links/train/correct-age/class_weights/'
120     #     ****'/home/julian/Downloads/Github/contrastive-predictive-coding/models/14_08_21-15_36-train|(4x)cpc
121     # /home/julian/Downloads/Github/contrastive-predictive-coding/models/14_08_21-14_22-train|(4x)cpc
122     # /home/julian/Downloads/Github/contrastive-predictive-coding/models/14_08_21-13_13-train|(4x)cpc
123     # /home/julian/Downloads/Github/contrastive-predictive-coding/models/14_08_21-12_13-train|(4x)cpc
124     # /home/julian/Downloads/Github/contrastive-predictive-coding/models/14_08_21-11_22-train|(4x)cpc
125     # /home/julian/Downloads/Github/contrastive-predictive-coding/models/14_08_21-10_44-train|(4x)cpc
126     # /home/julian/Downloads/Github/contrastive-predictive-coding/models/13_08_21-15_02-train|(4x)cpc
127     # /home/julian/Downloads/Github/contrastive-predictive-coding/models/13_08_21-14_42-train|(4x)cpc
128     # /home/julian/Downloads/Github/contrastive-predictive-coding/models/13_08_21-14_12-train|(4x)cpc
129     # /home/julian/Downloads/Github/contrastive-predictive-coding/models/13_08_21-13_51-train|(4x)cpc''''.split('\n')
130
131     #'home/julian/Downloads/Github/contrastive-predictive-coding/models/11_11_21-16-train|(2x)cpc'
132     #'['/home/julian/Downloads/Github/contrastive-predictive-coding/models/11_11_21-16-train|(2x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/15_11_21-13-train|cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/12_11_21-11-train|(8x)cpc'
133     #'home/julian/Downloads/Github/contrastive-predictive-coding/models/14_12_21-10-train|bl_MLP+bl_alex_v2+bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v2+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+bl_cnn_v8+bl_cnn_v9'

```

```

132     # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/20_12_21-15-49-train|bl_cnn_v14+
133     bl_cnn_v8',
134     # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/20_12_21-15-32-train|bl_cnn_v14+
135     bl_cnn_v8',
136     # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/20_12_21-15-17-train|bl_cnn_v14+
137     bl_cnn_v8',
138     # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/20_12_21-15-03-train|bl_cnn_v14+
139     bl_cnn_v8',
140     # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/11_02_22-13-48-train| (8x) cpc'
141     #'/home/julian/Downloads/Github/contrastive-predictive-coding/models/17_02_22-09-49-train| (3x) cpc'
142 ]
143
144 # infer class from model-arch file
145 model_dicts = []
146 for train_folder in model_folders:
147     model_files = extract_model_files_from_dir(train_folder)
148     for mfile in model_files:
149         fm_fs, cp_fs, root = mfile
150         fm_f = fm_fs[0]
151         if not args.checkpoint_file_ending is None:
152             temp = list(filter(lambda x: x.endswith(args.checkpoint_file_ending), cp_fs))
153             if len(temp) == 1:
154                 cp_f = temp[0]
155             elif len(temp) > 1:
156                 cp_f = sorted(cp_fs, key=lambda text: [int(c) if c.isdigit() else c for c in re.split(r'(\d+)', text)])[0]
157                 print(f"WARNING! multiple checkpoint files fitting '{args.checkpoint_file_ending}': {temp}. Taking first")
158             else:
159                 print(f"WARNING! No files found matching {args.checkpoint_file_ending}. Selecting latest.")
160                 cp_f = sorted(cp_fs, key=lambda text: [int(c) if c.isdigit() else c for c in re.split(r'(\d+)', text)])[-1]
161         else:
162             cp_f = sorted(cp_fs, key=lambda text: [int(c) if c.isdigit() else c for c in re.split(r'(\d+)', text)])[-1]
163         model = load_model_architecture(fm_f)
164         if model is None:
165             continue
166         model, _, epoch = load_model_checkpoint(cp_f, model, optimizer=None, device_id=f'cuda:{args.gpu_device}')
167         print(
168             f'Found architecturefile {os.path.basename(fm_f)}, checkpointfile {os.path.basename(cp_f)}, epochs:({epoch}), in folder {root}. Appending model for testing.')
169         model_dicts.append({'model': model, 'model_folder': root, 'trained_epochs': epoch})
170 if len(model_dicts) == 0:
171     print(f"Could not find any models in {model_folders}.")
172 loaders = [
173     DataLoader(test_dataset_challenge, batch_size=args.batch_size, drop_last=False, num_workers=1,
174                 collate_fn=ecg_datasets3.collate_fn),
175     DataLoader(val_dataset_challenge, batch_size=args.batch_size, drop_last=False, num_workers=1,
176                 collate_fn=ecg_datasets3.collate_fn),
177     # DataLoader(train_dataset_challenge, batch_size=args.batch_size, drop_last=False, num_workers=1,
178     #             collate_fn=ecg_datasets3.collate_fn), #Train usually not required
179 ]
180 metric_functions = [ # Functions that take two tensors as argument and give score or list of score
181     training_metrics.micro_avg_precision_score,
182     training_metrics.micro_avg_recall_score,
183 ]
184 for model_i, model_dict in enumerate(model_dicts):
185     model = model_dict['model']
186     model_name = os.path.split(model_dict['model_folder'])[1]
187     train_folder = os.path.split(os.path.split(model_dict['model_folder'])[0])[1]
188     output_path = os.path.join(args.out_path, train_folder, model_name)
189     print("Evaluating {}. Output will be saved to dir: {}".format(model_name, output_path))
190     # Create dirs and model info
191     Path(output_path).mkdir(parents=True, exist_ok=True)
192     try:
193         print('Copying params.txt to train_params.txt')
194         train_params = os.path.join(model_dict['model_folder'], 'params.txt')
195         with open(train_params, 'r') as tp:
196             txt = tp.read()
197             txt = re.sub("downstream_epochs=\d+", f"downstream_epochs={model_dict['trained_epochs']}", txt)
198             with open(os.path.join(output_path, 'train_params.txt'), 'w') as tpc:
199                 tpc.write(txt)
200             #shutil.copyfile(train_params, os.path.join(output_path, 'train_params.txt'))
201     except FileNotFoundError as e:
202         print(e)
203     store_models.save_model_variables_text_only(output_path, model)
204     try:
205         store_models.save_model_architecture_text_only(output_path, model)
206     except AttributeError as e:
207         print(e)
208         print('Older/Newer Model maybe?')
209         with open(os.path.join(output_path, 'params.txt'), 'w') as cfg:
210             cfg.write(str(args))
211         model.cuda()
212         # first = True
213         # init optimizer
214         optimizer = Adam(model.parameters(), lr=3e-4)

```

```

213 metrics = defaultdict(lambda: defaultdict(list))
214 for epoch in range(1, 2):
215     starttime = time.time()
216     for loader_i, loader in enumerate(loaders):
217         pred_dataframe = pd.DataFrame(columns=classes)
218         pred_dataframe.index.name = 'filename'
219         label_dataframe = pd.DataFrame(columns=classes)
220         label_dataframe.index.name = 'filename'
221         for dataset_tuple in loader:
222             data, labels, filenames = dataset_tuple
223             data = data.float().cuda()
224             labels = labels.float().cuda()
225             # print(torch.any(torch.isnan(data)), data.shape, torch.any(torch.isnan(labels)), labels.shape)
226             # if first:
227             #     dummy = torch.randn_like(data, requires_grad=True)
228             #     dummy_p = model(dummy)
229             #     make_dot(dummy_p, params=dict(list(model.named_parameters()) + [('x', dummy)])).render(
230                 model_name+'-viz', output_path, format="png")
231             #     first = False
232             optimizer.zero_grad()
233             pred = model(data, y=None)  # makes model return prediction instead of loss
234             # print(pred.shape)
235             if len(pred.shape) == 1:  # hack for squeezed batch dimension
236                 pred = pred.unsqueeze(0)
237             if torch.any(torch.isnan(pred)):
238                 print('nan encountered')
239                 print('nan in data:', torch.any(torch.isnan(data)))
240                 print(pred)
241             pred = pred.detach().cpu()
242
243             labels = labels.cpu()
244             labels_numpy = parse_tensor_to_numpy_or_scalar(labels)
245             pred_numpy = parse_tensor_to_numpy_or_scalar(pred)
246             pred_dataframe.append(pd.DataFrame(pred_numpy, columns=classes, index=filenames))
247             label_dataframe.append(
248                 pd.DataFrame(labels_numpy, columns=classes, index=filenames))
249             helper_code.save_challenge_predictions(output_path, filenames, classes=classes, scores=pred_numpy,
250                                                     labels=labels_numpy)
251             with torch.no_grad():
252                 for i, fn in enumerate(metric_functions):
253                     metrics[epoch][f'acc_{i}'].append(
254                         parse_tensor_to_numpy_or_scalar(fn(y=labels, pred=pred)))
255             if args.dry_run:
256                 break
257             del data, pred, labels
258             csv_pred_name = "model-{}-dataloader-{}-output.csv".format(model_name, loader_i)
259             csv_label_name = "labels-dataloader-{}.csv".format(loader_i)
260             print("\tFinished dataset {}. Progress: {} / {}".format(loader_i, loader_i + 1, len(loaders)))
261             print("\tSaving prediction and label to csv.")
262             pred_dataframe.to_csv(os.path.join(output_path, csv_pred_name))
263             label_dataframe.to_csv(os.path.join(output_path, csv_label_name))
264             print("\tSaved files {} and {}".format(csv_pred_name, csv_label_name))
265             torch.cuda.empty_cache()
266
267             elapsed_time = str(datetime.timedelta(seconds=time.time() - starttime))
268             metrics[epoch]['elapsed_time'].append(elapsed_time)
269             print("Done. Elapsed time: {}".format(elapsed_time))
270             if args.dry_run:
271                 break
272
273             pickle_name = "model-{}-test.pickle".format(model_name)
274             # Saving metrics in pickle
275             with open(os.path.join(output_path, pickle_name), 'wb') as pick_file:
276                 pickle.dump(dict(metrics), pick_file)
277             print("Finished model {}. Progress: {} / {}".format(model_name, model_i + 1, len(model_dicts)))
278             del model  # delete and free
279             torch.cuda.empty_cache()
280
281 def save_dict_to_csv_file(filepath, data, column_names=None):
282     with open(filepath) as f:
283         if not column_names is None:
284             f.write(','.join(column_names) + '\n')
285         for dc in data:
286             f.write(','.join(dc) + '\n')
287
288 def parse_tensor_to_numpy_or_scalar(input_tensor):
289     if type(input_tensor) == torch.Tensor:
290         arr = input_tensor.detach().cpu().numpy() if input_tensor.is_cuda else input_tensor.numpy()
291         if arr.size == 1:
292             return arr.item()
293         return arr
294     return input_tensor
295
296
297 if __name__ == "__main__":
298     import sys
299
300     print(sys.argv)

```

```

302 parser = argparse.ArgumentParser(description='Contrastive Predictive Coding')
303 # datapath
304 # Other params
305
306 parser.add_argument('--seed', type=int, help='The seed used', default=0)
307
308 parser.add_argument('--out_path', help="The output directory for losses and models",
309                     default='models/' + str(datetime.datetime.now().strftime("%d_%m_%y-%H-%M")) + '-test', type=
310                     str)
311
312 parser.add_argument('--forward_classes', type=int, default=67,
313                     help="The number of possible output classes (only relevant for downstream)")
314
315 parser.add_argument('--checkpoint_file_ending', type=str, default=None)
316
317 parser.add_argument('--batch_size', type=int, default=24, help="The batch size")
318
319 parser.add_argument('--latent_size', type=int, default=128,
320                     help="The size of the latent encoding for one window")
321
322 parser.add_argument('--crop_size', type=int, default=4500,
323                     help="The size of the data that it is cropped to. If data is smaller than this number, data
324                     gets padded with zeros")
325
326 parser.add_argument('--norm_fn', type=str, default='normalize_std_scaling',
327                     help="The Normalization function to use (from ecg_datasets3")
328
329 parser.add_argument("--gpu_device", type=int, default=0)
330
331 parser.add_argument('--dry_run', dest='dry_run', action='store_true',
332                     help="Only run minimal samples to test all models functionality")
333 parser.set_defaults(dry_run=False)
334
335 parser.add_argument("--preload_fraction", type=float, default=1.)
336
337 args = parser.parse_args()
338 main(args)

```

B.12 main_cpc_benchmark_train.py (code)

```

1 import argparse
2 import copy
3 import datetime
4 import os
5 import pickle
6 import re
7 import time
8 from collections import defaultdict
9 from pathlib import Path
10
11 import numpy as np
12 import torch
13 from torch.optim import Adam
14 from torch.optim.lr_scheduler import ReduceLROnPlateau
15 from torch.utils.data import DataLoader, ChainDataset
16
17 from architectures_baseline_challenge import baseline_cnn_v14, baseline_cnn_v15, baseline_cnn_v8, baseline_TCN_down,
18     baseline_cnn_v2, \
19     baseline_FCN, baseline_MLP, baseline_TCN_block, baseline_TCN_flatten, baseline_TCN_last, baseline_alex_v2,
20     baseline_cnn_v0, baseline_cnn_v0_1, \
21     baseline_cnn_v0_2, baseline_cnn_v0_3, baseline_cnn_v1, baseline_cnn_v3, baseline_cnn_v4, baseline_cnn_v5,
22     baseline_cnn_v6, baseline_cnn_v7, \
23     baseline_cnn_v9, baseline_resnet
24
25
26 from architectures_cpc import cpc_autoregressive_v0, cpc_combined, cpc_encoder_v0, cpc_intersect, \
27     cpc_predictor_v0, cpc_encoder_as_strided, cpc_downstream_cnn, cpc_downstream_only, \
28     cpc_downstream_latent_maximum, cpc_downstream_twolinear_v2, cpc_downstream_latent_average, \
29     cpc_intersect_manylatents, cpc_encoder_small, cpc_autoregressive_hidden, cpc_encoder_likev8, \
30     cpc_predictor_nocontext, cpc_predictor_nocontext
31
32 from util import store_models
33 from util.data import ecg_datasets3
34 from util.utility.full_class_name import fullname
35 from util.metrics import training_metrics, baseline_losses as bl
36 from util.store_models import save_model_architecture, save_checkpoint, save_model_variables_text_only, \
37     save_model_state_dict_checkpoint
38
39 def main(args):
40     np.random.seed(args.seed)
41     torch.cuda.set_device(args.gpu_device)
42     print(f'Device set to : {torch.cuda.current_device()}. Selected was {args.gpu_device}')
43     torch.cuda.manual_seed(args.seed)

```

```

43     print(f'Seed set to : {args.seed}.')
44
45     print(f'Model outputpath: {args.out_path}')
46     Path(args.out_path).mkdir(parents=True, exist_ok=True)
47     # norm_fn = ecg_datasets3.normalize_minmax_scaling_different
48     if hasattr(ecg_datasets3, args.norm_fn):
49         norm_fn = getattr(ecg_datasets3, args.norm_fn)
50     else:
51         norm_fn = ecg_datasets3.normalize_std_scaling
52
53     # georgia_challenge = ecg_datasets3.ECGChallengeDatasetBaseline('/home/juwin106/data/georgia/WFDB',
54     #                                                               window_size=args.crop_size,
55     #                                                               pad_to_size=args.crop_size, return_labels=True,
56     #                                                               normalize_fn=norm_fn)
57     # cpsc_challenge = ecg_datasets3.ECGChallengeDatasetBaseline('/home/juwin106/data/cpsc_train/',
58     #                                                               window_size=args.crop_size, pad_to_size=args.
59     #                                                               crop_size,
60     #                                                               return_labels=True,
61     #                                                               normalize_fn=norm_fn)
62     # cpsc2_challenge = ecg_datasets3.ECGChallengeDatasetBaseline('/home/juwin106/data/cpsc',
63     #                                                               window_size=args.crop_size,
64     #                                                               pad_to_size=args.crop_size, return_labels=True,
65     #                                                               normalize_fn=norm_fn)
66     # ptbxl_challenge = ecg_datasets3.ECGChallengeDatasetBaseline('/home/juwin106/data/ptbxl/WFDB',
67     #                                                               window_size=args.crop_size,
68     #                                                               pad_to_size=args.crop_size, return_labels=True,
69     #                                                               normalize_fn=norm_fn)
70     # nature = ecg_datasets3.ECGChallengeDatasetBaseline('/home/juwin106/data/nature',
71     #                                                               window_size=args.crop_size,
72     #                                                               pad_to_size=args.crop_size, return_labels=True,
73     #                                                               normalize_fn=norm_fn)
74
75     georgia_challenge = ecg_datasets3.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/georgia_challenge/',
76                                                               window_size=args.crop_size,
77                                                               pad_to_size=args.crop_size, return_labels=True,
78                                                               normalize_fn=norm_fn)
79     cpsc_challenge = ecg_datasets3.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/cpsc2018_challenge/',
80                                                               window_size=args.crop_size, pad_to_size=args.crop_size,
81                                                               return_labels=True,
82                                                               normalize_fn=norm_fn)
83     cpsc2_challenge = ecg_datasets3.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/china_challenge',
84                                                               window_size=args.crop_size,
85                                                               pad_to_size=args.crop_size, return_labels=True,
86                                                               normalize_fn=norm_fn)
87     ptbxl_challenge = ecg_datasets3.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG(ptbxl_challenge',
88                                                               window_size=args.crop_size,
89                                                               pad_to_size=args.crop_size, return_labels=True,
90                                                               normalize_fn=norm_fn)
91     nature = ecg_datasets3.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/nature_database',
92                                                               window_size=args.crop_size,
93                                                               pad_to_size=args.crop_size, return_labels=True,
94                                                               normalize_fn=norm_fn)
95
96     if args.redo_splits:
97         ecg_datasets3.filter_update_classes_by_count(
98             [georgia_challenge, cpsc_challenge, ptbxl_challenge, cpsc2_challenge, nature], min_count=20)
99         print("Warning! Redoing splits!")
100        ptbxl_challenge.random_train_split_with_class_count()
101        cpsc_challenge.random_train_split_with_class_count()
102        cpsc2_challenge.random_train_split_with_class_count()
103        georgia_challenge.random_train_split_with_class_count()
104
105        print("Loading splits file:", args.splits_file)
106        ptbxl_train, ptbxl_val, t1 = ptbxl_challenge.generate_datasets_from_split_file(args.splits_file)
107        georgia_train, georgia_val, t2 = georgia_challenge.generate_datasets_from_split_file(args.splits_file)
108        cpsc_train, cpsc_val, t3 = cpsc_challenge.generate_datasets_from_split_file(args.splits_file)
109        cpsc2_train, cpsc2_val, t4 = cpsc2_challenge.generate_datasets_from_split_file(args.splits_file)
110
111        ecg_datasets3.filter_update_classes_by_count(
112            [nature, ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train,
113             cpsc2_val, t4], 1)
114        print('Classes after last update', len(ptbxl_train.classes), ptbxl_train.classes)
115        if args.use_class_weights:
116            counts, counted_classes = ecg_datasets3.count_merged_classes(
117                [nature, ptbxl_train, ptbxl_val, t1, georgia_train, georgia_val, t2, cpsc_train, cpsc_val, t3, cpsc2_train
118
119                cpsc2_val, t4])
120            class_weights = torch.Tensor(max(counts) / counts).to(device=f'cuda:{(args.gpu_device)}')
121            print('Using the following class weights:', class_weights)
122        else:
123            class_weights = None
124
125        if args.preload_fraction > 0. and getattr(ptbxl_train, 'preload'):
126            print("Preloading data...")
127            ptbxl_train.preload(args.preload_fraction)
128            ptbxl_val.preload(args.preload_fraction)
129            georgia_train.preload(args.preload_fraction)
130            georgia_val.preload(args.preload_fraction)
131            cpsc_train.preload(args.preload_fraction)

```

```

131     cpsc_val.preload(args.preload_fraction)
132     cpsc2_train.preload(args.preload_fraction)
133     cpsc2_val.preload(args.preload_fraction)
134
135     pretrain_train_dataset = ChainDataset([nature, ptbxl_train, georgia_train, cpsc_train, cpsc2_train]) # CPC TRAIN
136     pretrain_val_dataset = ChainDataset([ptbxl_val, georgia_val, cpsc_val, cpsc2_val]) # CPC VAL
137     downstream_train_dataset = ChainDataset([ptbxl_train, georgia_train, cpsc_train, cpsc2_train])
138     downstream_val_dataset = ChainDataset([ptbxl_val, georgia_val, cpsc_val, cpsc2_val])
139     pretrain_models = [
140         # cpc_intersect.CPC(
141             # cpc_encoder_v0.Encoder(args.channels, args.latent_size),
142             # cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
143             # cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_in),
144             # args.timesteps_in, args.timesteps_out, args.latent_size,
145             # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='all'
146         # ),
147         # cpc_intersect.CPC(
148             # cpc_encoder_as_strided.StridedEncoder(cpc_encoder_v0.Encoder(args.channels, args.latent_size),
149             #                                         args.window_size),
150             # cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
151             # cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_in // 4),
152             # args.timesteps_in // 4, args.timesteps_out // 4, args.latent_size,
153             # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='all'
154         # ),
155         # cpc_intersect.CPC(
156             # cpc_encoder_v0.Encoder(args.channels, args.latent_size),
157             # cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
158             # cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
159             # args.timesteps_in, args.timesteps_out, args.latent_size,
160             # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='same'
161         # ),
162         # cpc_intersect.CPC(
163             # cpc_encoder_as_strided.StridedEncoder(cpc_encoder_v0.Encoder(args.channels, args.latent_size),
164             #                                         args.window_size),
165             # cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
166             # cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out//4),
167             # args.timesteps_in//4, args.timesteps_out//4, args.latent_size,
168             # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='same'
169         # ),
170         # cpc_intersect.CPC(
171             # cpc_encoder_v0.Encoder(args.channels, args.latent_size),
172             # cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
173             # cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
174             # args.timesteps_in, args.timesteps_out, args.latent_size,
175             # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='all'
176         # ),
177         # cpc_intersect.CPC(
178             # cpc_encoder_as_strided.StridedEncoder(cpc_encoder_v0.Encoder(args.channels, args.latent_size),
179             #                                         args.window_size),
180             # cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
181             # cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out//4),
182             # args.timesteps_in//4, args.timesteps_out//4, args.latent_size,
183             # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='all'
184         # ),
185         # cpc_intersect_manylatents.CPC(
186             # cpc_encoder_likev8.Encoder(args.channels, args.latent_size),
187             # cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
188             # cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
189             # args.timesteps_in, args.timesteps_out, args.latent_size,
190             # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='multisame'
191         # ),
192         # cpc_intersect_manylatents.CPC(
193             # cpc_encoder_likev8.Encoder(args.channels, args.latent_size),
194             # cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
195             # cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
196             # args.timesteps_in, args.timesteps_out, args.latent_size,
197             # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='multisame'
198         # ),
199         # cpc_intersect_manylatents.CPC(
200             # cpc_encoder_likev8.Encoder(args.channels, args.latent_size),
201             # cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
202             # cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
203             # args.timesteps_in, args.timesteps_out, args.latent_size,
204             # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='same'
205         # ),
206         # cpc_intersect.CPC(
207             # cpc_encoder_as_strided.StridedEncoder(cpc_encoder_v0.Encoder(args.channels, args.latent_size),
208             #                                         args.window_size),
209             # cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
210             # cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out//4),
211             # args.timesteps_in//4, args.timesteps_out//4, args.latent_size,
212             # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='future'
213         # ),
214         # cpc_intersect_manylatents.CPC(
215             # cpc_encoder_small.Encoder(args.channels, args.latent_size),
216             # cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
217             # cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
218             # args.timesteps_in, args.timesteps_out, args.latent_size,
219             # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='all'
220         # ),

```

```

221
222     # cpc_intersect_manylatents.CPC(
223     #     cpc_encoder_likev8.Encoder(args.channels, args.latent_size),
224     #     cpc_autoregressive_hidden.AutoRegressor(args.latent_size, args.hidden_size, 1), #With hidden instead of
225     #         context
226     #         # cpc_predictor_stacked.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
227     #         # args.timesteps_in, args.timesteps_out, args.latent_size,
228     #         # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='crossentropy'
229     #     ),
230     #     cpc_intersect_manylatents.CPC(
231     #         cpc_encoder_likev8.Encoder(args.channels, args.latent_size),
232     #         cpc_autoregressive_hidden.AutoRegressor(args.latent_size, args.hidden_size, 1), #With hidden instead of
233     #             context
234     #             # cpc_predictor_nocontext.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
235     #             # args.timesteps_in, args.timesteps_out, args.latent_size,
236     #             # timesteps_ignore=0, normalize_latents=True, verbose=False, sampling_mode='crossentropy-nocontext'
237     #         ),
238     #         cpc_intersect_manylatents.CPC(
239     #             cpc_encoder_likev8.Encoder(args.channels, args.latent_size),
240     #             cpc_autoregressive_hidden.AutoRegressor(args.latent_size, args.hidden_size, 1), #With hidden instead of
241     #                 context
242     #                 # cpc_predictor_nocontext.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
243     #                 # args.timesteps_in, args.timesteps_out, args.latent_size,
244     #                 # timesteps_ignore=0, normalize_latents=True, verbose=False, sampling_mode='crossentropy-nocontext'
245     #             ),
246     #             cpc_intersect_manylatents.CPC(
247     #                 cpc_encoder_as_strided.StridedEncoder(cpc_encoder_likev8.Encoder(args.channels, args.latent_size), args.
248     #                     window_size),
249     #                     # cpc_autoregressive_hidden.AutoRegressor(args.latent_size, args.hidden_size, 1), #With hidden instead of
250     #                     context
251     #                     # cpc_predictor_nocontext.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
252     #                     # args.timesteps_in, args.timesteps_out, args.latent_size,
253     #                     # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='crossentropy-nocontext'
254     #                 ),
255     #                 cpc_intersect_manylatents.CPC(
256     #                     cpc_encoder_small.Encoder(args.channels, args.latent_size),
257     #                     cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
258     #                     cpc_predictor_nocontext.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
259     #                     # args.timesteps_in, args.timesteps_out, args.latent_size,
260     #                     # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='crossentropy-nocontext'
261     #                 ),
262     #                 cpc_intersect_manylatents.CPC(
263     #                     cpc_encoder_v0.Encoder(args.channels, args.latent_size),
264     #                     cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
265     #                     cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_in),
266     #                     # args.timesteps_in, args.timesteps_out, args.latent_size,
267     #                     # timesteps_ignore=0, normalize_latents=True, verbose=False, sampling_mode='crossentropy'
268     #                 ),
269     #                 cpc_intersect_manylatents.CPC(
270     #                     cpc_encoder_as_strided.StridedEncoder(cpc_encoder_v0.Encoder(args.channels, args.latent_size),
271     #                         args.window_size),
272     #                         # cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
273     #                         # cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out//4),
274     #                         # args.timesteps_in//4, args.timesteps_out//4, args.latent_size,
275     #                         # timesteps_ignore=0, normalize_latents=True, verbose=False, sampling_mode='crossentropy'
276     #                 ),
277     #                 cpc_intersect_manylatents.CPC(
278     #                     cpc_encoder_v0.Encoder(args.channels, args.latent_size),
279     #                     cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
280     #                     cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_in),
281     #                     # args.timesteps_in, args.timesteps_out, args.latent_size,
282     #                     # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='crossentropy'
283     #                 ),
284     #                 cpc_intersect_manylatents.CPC(
285     #                     cpc_encoder_as_strided.StridedEncoder(cpc_encoder_v0.Encoder(args.channels, args.latent_size),
286     #                         args.window_size),
287     #                         # cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
288     #                         # cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out//4),
289     #                         # args.timesteps_in//4, args.timesteps_out//4, args.latent_size,
290     #                         # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='crossentropy'
291     #                 ),
292     #                 cpc_intersect_manylatents.CPC(
293     #                     cpc_encoder_likev8.Encoder(args.channels, args.latent_size),
294     #                     cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
295     #                     cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_in),
296     #                     # args.timesteps_in, args.timesteps_out, args.latent_size,
297     #                     # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='crossentropy'
298     #                 ),
299     #                 cpc_intersect_manylatents.CPC(
300     #                     cpc_encoder_likev8.Encoder(args.channels, args.latent_size),
301     #                     cpc_autoregressive_hidden.AutoRegressor(args.latent_size, args.hidden_size, 1),
302     #                     cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_in),
303     #                     # args.timesteps_in, args.timesteps_out, args.latent_size,
304     #                     # timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='crossentropy'
305     #                 ),
306     #                 cpc_intersect_manylatents.CPC(
307     #                     cpc_encoder_as_strided.StridedEncoder(cpc_encoder_likev8.Encoder(args.channels, args.latent_size),
308     #                         window_size=545),
309     #                         # cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),

```

```

306     #     cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
307     #     args.timesteps_in//4, args.timesteps_out//4, args.latent_size,
308     #     timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='crossentropy'
309     # ),
310     # cpc_intersect_manylatents.CPC(
311     #     cpc_encoder_as_strided.StridedEncoder(cpc_encoder_v0.Encoder(args.channels, args.latent_size),
312     #                                         args.window_size),
313     #     cpc_autoregressive_hidden.AutoRegressor(args.latent_size, args.hidden_size, 1),
314     #     cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out//4),
315     #     args.timesteps_in//4, args.timesteps_out//4, args.latent_size,
316     #     timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='crossentropy'
317     # ),
318     # cpc_intersect_manylatents.CPC(
319     #     cpc_encoder_as_strided.StridedEncoder(cpc_encoder_v0.Encoder(args.channels, args.latent_size),
320     #                                         args.window_size),
321     #     cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size, 1),
322     #     cpc_predictor_nocontext.Predictor(args.hidden_size, args.latent_size, args.timesteps_out//4),
323     #     args.timesteps_in//4, args.timesteps_out//4, args.latent_size,
324     #     timesteps_ignore=0, normalize_latents=False, verbose=False, sampling_mode='crossentropy-nocontext'
325     # ),
326   ],
327   downstream_models = [
328     # cpc_downstream_only.DownstreamLinearNet(
329     #     latent_size=args.latent_size, context_size=args.hidden_size, out_classes=args.forward_classes,
330     #     use_latents=False, use_context=True, verbose=False
331     # ),
332     # cpc_downstream_only.DownstreamLinearNet(
333     #     latent_size=args.latent_size, context_size=args.hidden_size, out_classes=args.forward_classes,
334     #     use_latents=True, use_context=False, verbose=False
335     # ),
336     # cpc_downstream_only.DownstreamLinearNet(
337     #     latent_size=args.latent_size, context_size=args.hidden_size, out_classes=args.forward_classes,
338     #     use_latents=True, use_context=True, verbose=False
339     # ),
340     # cpc_downstream_cnn.DownstreamLinearNet(
341     #     latent_size=args.latent_size, context_size=args.hidden_size, out_classes=args.forward_classes,
342     #     use_latents=True, use_context=False, verbose=False
343     # ),
344     cpc_downstream_latent_maximum.DownstreamLinearNet(
345       latent_size=args.latent_size, context_size=args.hidden_size, out_classes=args.forward_classes,
346       use_latents=True, use_context=True, verbose=False
347     ),
348     cpc_downstream_latent_average.DownstreamLinearNet(
349       latent_size=args.latent_size, context_size=args.hidden_size, out_classes=args.forward_classes,
350       use_latents=True, use_context=True, verbose=False
351     ),
352     # cpc_downstream_latent_maximum.DownstreamLinearNet(
353     #     latent_size=args.latent_size, context_size=args.hidden_size, out_classes=args.forward_classes,
354     #     use_latents=True, use_context=False, verbose=False
355     # ),
356     # cpc_downstream_latent_average.DownstreamLinearNet(
357     #     latent_size=args.latent_size, context_size=args.hidden_size, out_classes=args.forward_classes,
358     #     use_latents=True, use_context=False, verbose=False
359     # ),
360     cpc_downstream_twolinear_v2.DownstreamLinearNet(
361       latent_size=args.latent_size, context_size=args.hidden_size, out_classes=args.forward_classes,
362       use_latents=True, use_context=True, verbose=False
363     ),
364     # cpc_downstream_twolinear_v2.DownstreamLinearNet(
365     #     latent_size=args.latent_size, context_size=args.hidden_size, out_classes=args.forward_classes,
366     #     use_latents=True, use_context=False, verbose=False
367     # ),
368     # cpc_downstream_twolinear_v2.DownstreamLinearNet(
369     #     latent_size=args.latent_size, context_size=args.hidden_size, out_classes=args.forward_classes,
370     #     use_latents=False, use_context=True, verbose=False
371     # ),
372   ],
373   trained_model_dicts = [  # continue training for these in some way
374     {
375       'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/15_12_21-21-train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined0|train-test-splits|use_weights|frozen|C|L|m:same|cpc_downstream_latent_maximum',
376       'model': None  # standard same
377     },
378     {
379       'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/15_12_21-21-train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined1|train-test-splits|use_weights|strided|frozen|C|L|m:same|cpc_downstream_latent_maximum',
380       'model': None  # standard same strided
381     },
382     {
383       'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/15_12_21-21-train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined2|train-test-splits|use_weights|frozen|C|L|m:all|cpc_downstream_latent_maximum',
384       'model': None  # standard all
385     },
386     {
387       'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/15_12_21-21-train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined3|train-test-splits|use_weights|strided|frozen|C|L|m:all|cpc_downstream_latent_maximum',
388     }
  ]

```

```

388         'model': None # standard all strided
389     },
390     # {
391     #   'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/02_02_22-16-12-train| (2x)
392     cpc/architectures_cpc.cpc_combined.CPCCCombined0|train-test-splits|use_weights|unfrozen|C|L|m:crossentropy|
393     cpc_downstream_latent_maximum',
394     #   'model': None # many latents v0s
395     # },
396     # {
397     #   'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/02_02_22-16-12-train| (2x)
398     cpc/architectures_cpc.cpc_combined.CPCCCombined1|train-test-splits|use_weights|strided|unfrozen|C|L|m:
399     crossentropy|cpc_downstream_latent_maximum',
400     #   'model': None # many latents v0 strided
401     # },
402     # {
403     #   'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/04_02_22-17-52-train| (2x)
404     cpc/architectures_cpc.cpc_combined.CPCCCombined0|train-test-splits|use_weights|unfrozen|C|L|m:multisame|
405     cpc_downstream_latent_maximum',
406     #   'model': None # normal multisame
407     # },
408     # {
409     #   'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/04_02_22-17-52-train| (2x)
410     cpc/architectures_cpc.cpc_combined.CPCCCombined1|train-test-splits|use_weights|unfrozen|C|L|m:same|
411     cpc_downstream_latent_maximum',
412     #   'model': None # normal same
413     # },
414     # {
415     #   'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/26_01_22-14-14-train| (4x)
416     cpc/architectures_cpc.cpc_combined.CPCCCombined0|train-test-splits|use_weights|unfrozen|C|L|m:crossentropy|
417     cpc_downstream_latent_maximum',
418     #   'model': None # latent max cpc, down only 120
419     # },
420     # {
421     #   'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/02_02_22-17-29-train| (6x)
422     cpc/architectures_cpc.cpc_combined.CPCCCombined3|train-test-splits|use_weights|frozen|C|L|m:crossentropy|
423     cpc_downstream_latent_maximum',
424     #   'model': None # latent max cpc, pretrained 100
425     # },
426     # #### new archs test
427     # {
428     #   'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/02_02_22-16-12-train| (2x)
429     cpc/architectures_cpc.cpc_combined.CPCCCombined0|train-test-splits|use_weights|unfrozen|C|L|m:crossentropy|
430     cpc_downstream_latent_maximum',
431     #   'model': None # v0
432     # },
433     # {
434     #   'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/23_12_21-19-52-train| (2x)cpc
435     /architectures_cpc.cpc_combined.CPCCCombined0|train-test-splits|use_weights|frozen|C|m:crossentropy|
436     cpc_downstream_only',
437     #   'model': None # v8
438     # },
439     # {
440     #   'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/22_12_21-12-train|cpc/
441     architectures_cpc.cpc_combined.CPCCCombined0|train-test-splits|use_weights|frozen|C|m:all|cpc_downstream_only',
442     #   'model': None # v0 hidden
443     # },
444     # {
445     #   'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/07_02_22-12-07-train|cpc/
446     architectures_cpc.cpc_combined.CPCCCombined0|train-test-splits|use_weights|unfrozen|C|L|m:crossentropy|
447     cpc_downstream_latent_maximum',
448     #   'model': None # v8 hidden
449     # },
450     # {
451     #   'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/05_01_22-18-16-train| (2x)
452     cpc/architectures_cpc.cpc_combined.CPCCCombined0|train-test-splits|use_weights|unfrozen|C|m:crossentropy-
453     nocontext|cpc_downstream_only',
454     #   'model': None # v0 nocontext
455     # },
456     # {
457     #   'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/04_01_22-16-51-train|cpc/
458     architectures_cpc.cpc_combined.CPCCCombined0|train-test-splits|use_weights|frozen|C|m:crossentropy-nocontext|
459     cpc_downstream_only',
460     #   'model': None # v8 nocontext
461     # },
462     # {
463     #   'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/27_01_22-14-49-train| (2x)
464     cpc/architectures_cpc.cpc_combined.CPCCCombined0|train-test-splits|use_weights|strided|unfrozen|C|LNORM|m:
465     crossentropy|cpc_downstream_only',
466     #   'model': None # strided normalized
467     # },
468     # {
469     #   'folder': '/home/julian/Downloads/Github/contrastive-predictive-coding/models/27_01_22-14-49-train| (2x)
470     cpc/architectures_cpc.cpc_combined.CFCCCombined1|train-test-splits|use_weights|unfrozen|C|LNORM|m:crossentropy|
471     cpc_downstream_only',
472     #   'model': None # normalized
473     # },
474   ]
475   for model_i, trained_model_dict in enumerate(trained_model_dicts): # hack bad
476     model_path = trained_model_dict['folder']
477     model_files = store_models.extract_model_files_from_dir(model_path)

```

```

451     if len(model_files) == 0:
452         print(f"Could not find specified model {model_path}")
453     for mfile in model_files:
454         fm_fs, cp_fs, root = mfile
455         fm_f = fm_fs[0]
456         if not args.checkpoint_file_ending is None:
457             temp = list(filter(lambda x: x.endswith(args.checkpoint_file_ending), cp_fs))
458             if len(temp) == 1:
459                 cp_f = temp[0]
460             elif len(temp) > 1:
461                 cp_f = sorted(cp_fs, key=lambda text: [int(c) if c.isdigit() else c for c in re.split(r'(\d+)', text)])[0]
462                 print(f"WARNING! multiple checkpoint files fitting '{args.checkpoint_file_ending}': {temp}. Taking first")
463             else:
464                 print(f"WARNING! No files found matching {args.checkpoint_file_ending}. Selecting latest.")
465                 cp_f = sorted(cp_fs, key=lambda text: [int(c) if c.isdigit() else c for c in re.split(r'(\d+)', text)])[-1]
466             else:
467                 cp_f = sorted(cp_fs, key=lambda text: [int(c) if c.isdigit() else c for c in re.split(r'(\d+)', text)])[-1]
468             model = store_models.load_model_architecture(fm_f)
469             model, _, epoch = store_models.load_model_checkpoint(cp_f, model, optimizer=None,
470                                                       device_id=f'cuda:{(args.gpu_device)}')
471             trained_model_dict['model'] = model # load model into dict
472             trained_model_dict['pretrained_epochs'] = epoch
473             break # only take first you find
474     models = [
475         # {'model': cpc_combined.CPCCombined(trained_model_dicts[0]['model'].cpc_model, downstream_models[1],
476         freeze_cpc=False),
477         # 'pretrained_epochs':trained_model_dicts[0]['pretrained_epochs'],
478         # 'will_pretrain': False, 'will_downtrain': True},
479         # {'model': cpc_combined.CPCCombined(trained_model_dicts[0]['model'].cpc_model, downstream_models[2],
480         freeze_cpc=False),
481         # 'pretrained_epochs':trained_model_dicts[0]['pretrained_epochs'],
482         # 'will_pretrain': False, 'will_downtrain': True},
483         # # {'model': cpc_combined.CPCCombined(trained_model_dicts[0]['model'].cpc_model, downstream_models[1],
484         freeze_cpc=True),
485         # # 'pretrained_epochs':trained_model_dicts[0]['pretrained_epochs'],
486         # # 'will_pretrain': False, 'will_downtrain': True},
487         # {'model': cpc_combined.CPCCombined(trained_model_dicts[0]['model'].cpc_model, downstream_models[2],
488         freeze_cpc=True),
489         # 'pretrained_epochs':trained_model_dicts[0]['pretrained_epochs'],
490         # 'will_pretrain': False, 'will_downtrain': True},
491         # # #
492         # # # {'model': cpc_combined.CPCCombined(trained_model_dicts[1]['model'].cpc_model, downstream_models[1],
493         freeze_cpc=False),
494         # # # 'pretrained_epochs':trained_model_dicts[1]['pretrained_epochs'],
495         # # # 'will_pretrain': False, 'will_downtrain': True},
496         # {'model': cpc_combined.CPCCombined(trained_model_dicts[1]['model'].cpc_model, downstream_models[2],
497         freeze_cpc=False),
498         # 'pretrained_epochs':trained_model_dicts[1]['pretrained_epochs'],
499         # 'will_pretrain': False, 'will_downtrain': True},
500         # # #
501         # # # {'model': cpc_combined.CPCCombined(trained_model_dicts[1]['model'].cpc_model, downstream_models[1],
502         freeze_cpc=True),
503         # # # 'pretrained_epochs':trained_model_dicts[1]['pretrained_epochs'],
504         # # # 'will_pretrain': False, 'will_downtrain': True},
505         # {'model': cpc_combined.CPCCombined(trained_model_dicts[1]['model'].cpc_model, downstream_models[2],
506         freeze_cpc=True),
507         # 'pretrained_epochs':trained_model_dicts[1]['pretrained_epochs'],
508         # 'will_pretrain': False, 'will_downtrain': True},
509         # # #
510         # # # {'model': cpc_combined.CPCCombined(trained_model_dicts[2]['model'].cpc_model, downstream_models[1],
511         freeze_cpc=False),
512         # # # 'pretrained_epochs':trained_model_dicts[2]['pretrained_epochs'],
513         # # # 'will_pretrain': False, 'will_downtrain': True},
514         # {'model': cpc_combined.CPCCombined(trained_model_dicts[2]['model'].cpc_model, downstream_models[2],
515         freeze_cpc=True),
516         # 'pretrained_epochs':trained_model_dicts[2]['pretrained_epochs'],
517         # 'will_pretrain': False, 'will_downtrain': True},
518         # # #
519         # # # {'model': cpc_combined.CPCCombined(trained_model_dicts[2]['model'].cpc_model, downstream_models[1],
520         freeze_cpc=False),
521         # # # 'pretrained_epochs':trained_model_dicts[2]['pretrained_epochs'],
522         # # # 'will_pretrain': False, 'will_downtrain': True},
523         # {'model': cpc_combined.CPCCombined(trained_model_dicts[3]['model'].cpc_model, downstream_models[2],
524         freeze_cpc=False),
525         # 'pretrained_epochs':trained_model_dicts[2]['pretrained_epochs'],
526         # 'will_pretrain': False, 'will_downtrain': True}
527     ]

```

```

523     # 'will_pretrain': False, 'will_downtrain': True},
524     # #
525     # # ('model': cpc_combined.CPCCombined(trained_model_dicts[2]['model'].cpc_model, downstream_models[1],
526     freeze_cpc=True),
527     # # 'pretrained_epochs':trained_model_dicts[2]['pretrained_epochs'],
528     # # 'will_pretrain': False, 'will_downtrain': True},
529     # ('model': cpc_combined.CPCCombined(trained_model_dicts[3]['model'].cpc_model, downstream_models[2],
530     freeze_cpc=True),
531     # 'pretrained_epochs':trained_model_dicts[2]['pretrained_epochs'],
532     # 'will_pretrain': False, 'will_downtrain': True}
533
532 {'model': cpc_combined.CPCCombined(
533     cpc_intersect.CPC(cpc_encoder_v0.Encoder(12, args.latent_size),
534         cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size),
535         cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
536         args.timesteps_in, args.timesteps_out, args.latent_size, normalize_latents=False),
537     downstream_models[2], freeze_cpc=True), 'will_pretrain': False, 'will_downtrain': True},
538 {'model': cpc_combined.CPCCombined(
539     cpc_intersect.CPC(cpc_encoder_v0.Encoder(12, args.latent_size),
540         cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size),
541         cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
542         args.timesteps_in, args.timesteps_out, args.latent_size, normalize_latents=False),
543     downstream_models[1], freeze_cpc=True),
544     'will_pretrain': False, 'will_downtrain': True},
545 {'model': cpc_combined.CPCCombined(
546     cpc_intersect.CPC(cpc_encoder_v0.Encoder(12, args.latent_size),
547         cpc_autoregressive_v0.AutoRegressor(args.latent_size, args.hidden_size),
548         cpc_predictor_v0.Predictor(args.hidden_size, args.latent_size, args.timesteps_out),
549         args.timesteps_in, args.timesteps_out, args.latent_size, normalize_latents=False),
550     downstream_models[0], freeze_cpc=True), 'will_pretrain': False, 'will_downtrain': True},
551 # baseline_FCN.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
552 # baseline_MLP.BaselineNet(in_features=args.crop_size, out_classes=args.forward_classes, verbose=False),
553 # baseline_alex_v2.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
554 # baseline_resnet.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
555 # baseline_cnn_v0.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
556 # baseline_cnn_v0_1.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
557 # baseline_cnn_v0_2.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
558 # baseline_cnn_v0_3.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
559 # baseline_cnn_v1.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
560 # baseline_cnn_v2.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
561 # baseline_cnn_v3.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
562 # baseline_cnn_v4.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
563 # baseline_cnn_v5.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
564 # baseline_cnn_v6.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
565 # baseline_cnn_v7.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
566 # baseline_cnn_v8.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
567 # baseline_cnn_v9.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
568 # baseline_TCN_last.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
569 # # baseline_TCN_flatten.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=
args.forward_classes, verbose=False),
570 # baseline_TCN_down.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
571 # baseline_TCN_block.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
572 # baseline_cnn_v14.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
573 # baseline_cnn_v15.BaselineNet(in_channels=args.channels, out_channels=args.latent_size, out_classes=args.
forward_classes, verbose=False),
574 # ('model':cpc_combined.CPCCombined(pretrain_models[0], downstream_models[1], freeze_cpc=False), '
will_pretrain':False, 'will_downtrain':True},
575 # ('model':cpc_combined.CPCCombined(pretrain_models[0], downstream_models[0], freeze_cpc=True), 'will_pretrain
':False, 'will_downtrain':True),
576 # ('model':cpc_combined.CPCCombined(pretrain_models[0], downstream_models[1], freeze_cpc=True), 'will_pretrain
':False, 'will_downtrain':True),
577 # ('model':cpc_combined.CPCCombined(pretrain_models[2], downstream_models[0], freeze_cpc=True), 'will_pretrain
':True, 'will_downtrain':False),
578 # ('model':cpc_combined.CPCCombined(pretrain_models[3], downstream_models[0], freeze_cpc=True), 'will_pretrain
':True, 'will_downtrain':False),
579 # ('model': cpc_combined.CPCCombined(pretrain_models[0], downstream_models[1], freeze_cpc=True),
will_pretrain': False, 'will_downtrain': True),
580 # ('model': cpc_combined.CPCCombined(trained_model_dicts[0]['model'].cpc_model, downstream_models[0]),
will_pretrain': False, 'will_downtrain': True, 'desc':
# 'blabla'),
581

```

```

582     # baseline_rnn_simplest_lstm.BaselineNet(in_channels=args.channels, out_channels=None, out_classes=args.
583     forward_classes, verbose=False),
584     # baseline_rnn_simplest_gru.BaselineNet(in_channels=args.channels, out_channels=None, out_classes=args.
585     forward_classes, verbose=False),
586 ]
587
588 pretrain_train_loaders = [
589     DataLoader(pretrain_train_dataset, batch_size=args.batch_size, drop_last=False, num_workers=1,
590                 collate_fn=ecg_datasets3.collate_fn, pin_memory=True)
591 ]
592 pretrain_val_loaders = [
593     DataLoader(pretrain_val_dataset, batch_size=args.batch_size, drop_last=False, num_workers=1,
594                 collate_fn=ecg_datasets3.collate_fn, pin_memory=True)
595 ]
596 downstream_train_loaders = [
597     DataLoader(downstream_train_dataset, batch_size=args.batch_size, drop_last=False, num_workers=1,
598                 collate_fn=ecg_datasets3.collate_fn)
599 ]
600 downstream_val_loaders = [
601     DataLoader(downstream_val_dataset, batch_size=args.batch_size, drop_last=False, num_workers=1,
602                 collate_fn=ecg_datasets3.collate_fn)
603 ]
604 metric_functions = [
605     # Functions that take two tensors as argument and give score or list of score #TODO: maybe use dict with name
606     # accuracy_metrics.fn_score_label,
607     # accuracy_metrics.tn_score_label,
608     # accuracy_metrics.tp_score_label,
609     # accuracy_metrics.fl_score,
610     # accuracy_metrics.micro_avg_recall_score,
611     # accuracy_metrics.micro_avg_precision_score,
612     # accuracy_metrics.accuracy,
613     # training_metrics.zero_fit_score,
614     # training_metrics.class_fit_score
615     # accuracy_metrics.class_count_prediction,
616     # accuracy_metrics.class_count_truth
617 ]
618
619 def pretrain(model_i, model):
620     model_name = model.name if hasattr(model, 'name') else fullname(model)
621     pretrain_fun = getattr(model, 'pretrain', None)
622     if not callable(pretrain_fun): # this is not a CPC model!
623         print(f'{model_name} is not a CPC model (needs to implement pretrain)... Skipping pretrain call')
624         return
625     output_path = os.path.join(args.out_path, model_name + str(model_i))
626     print("Begin pretraining of {}. Output will be saved to dir: {}".format(model_name, output_path))
627     # Create dirs and model info
628     Path(output_path).mkdir(parents=True, exist_ok=True)
629     with open(os.path.join(output_path, 'params.txt'), 'w') as cfg:
630         temp = args.downstream_epochs #copy
631         args.downstream_epochs = 0 #change for save because model hasnt been downtrained yet and maybe is not
632         cfg.write(str(args))
633         args.downstream_epochs = temp
634     save_model_architecture(output_path, model, model_name)
635     save_model_variables_text_only(output_path, model)
636     model.cuda()
637     model.train()
638     # init optimizer
639     optimizer = Adam(model.parameters(), lr=3e-3)#3e-4
640     scheduler = ReduceLROnPlateau(optimizer, 'min', patience=5, min_lr=3e-10, verbose=True)
641     metrics = defaultdict(lambda: defaultdict(list))
642     update_count = 0
643     train_mean_loss = torch.Tensor([0.]).cuda()
644     train_mean_acc = torch.Tensor([0.]).cuda()
645     val_mean_loss = torch.Tensor([0.]).cuda()
646     val_mean_acc = torch.Tensor([0.]).cuda()
647     for epoch in range(1, args.pretrain_epochs + 1):
648         starttime = time.time() # train
649         moving_average = 0
650         for train_loader_i, train_loader in enumerate(pretrain_train_loaders):
651             for dataset_tuple in train_loader:
652                 data, _ = dataset_tuple
653                 data = data.float().cuda()
654                 optimizer.zero_grad()
655                 acc, loss, hidden = model.pretrain(data, y=None, hidden=None)
656                 loss.backward()
657                 optimizer.step()
658                 update_count += 1
659                 # saving metrics
660                 if not args.no_metrics:
661                     metrics[epoch]['trainloss'].append(parse_tensor_to_numpy_or_scalar(loss))
662                     metrics[epoch]['trainacc'].append(parse_tensor_to_numpy_or_scalar(acc))
663                 train_mean_loss += loss
664                 train_mean_acc += acc
665                 moving_average += 1
666                 if args.dry_run:
667                     break
668                 del data, loss, hidden
669             print(f'\tFinished training dataset {train_loader_i}. Progress: {train_loader_i + 1}/{len(pretrain_train_loaders)}')
670             # torch.cuda.empty_cache()

```

```

669     train_mean_loss = parse_tensor_to_numpy_or_scalar(train_mean_loss)/moving_average
670     train_mean_acc = parse_tensor_to_numpy_or_scalar(train_mean_acc)/moving_average
671     moving_average=0
672     with torch.no_grad():
673         for val_loader_i, val_loader in enumerate(pretrain_val_loaders): # validate
674             for dataset_tuple in val_loader:
675                 data, _ = dataset_tuple
676                 data = data.float().cuda()
677                 acc, loss, hidden = model.pretrain(data, y=None, hidden=None)
678                 # saving metrics
679                 if not args.no_metrics:
680                     metrics[epoch]['valloss'].append(parse_tensor_to_numpy_or_scalar(loss))
681                     metrics[epoch]['valacc'].append(parse_tensor_to_numpy_or_scalar(acc))
682                 val_mean_loss += loss
683                 val_mean_acc += acc
684                 moving_average += 1
685                 if args.dry_run:
686                     break
687                 print("\tFinished validation dataset {}. Progress: {} / {}".format(val_loader_i, val_loader_i + 1,
688                                                                                   len(pretrain_val_loaders)))
689                 del data, loss, hidden
690                 val_mean_loss = parse_tensor_to_numpy_or_scalar(val_mean_loss)/moving_average
691                 val_mean_acc = parse_tensor_to_numpy_or_scalar(val_mean_acc)/moving_average
692                 scheduler.step(val_mean_loss)
693                 elapsed_time = str(datetime.timedelta(seconds=time.time() - starttime))
694                 metrics[epoch]['elapsed_time'].append(elapsed_time)
695                 print(
696                     "Epoch {} / {} done. Avg train loss: {:.4f}. Avg val loss: {:.4f}. Avg train acc: {:.4f}. Avg val acc: {:.4f}. Elapsed time: {}. Total optimizer steps: {}.".format(
697                         epoch, args.pretrain_epochs, train_mean_loss, val_mean_loss,
698                         train_mean_acc, val_mean_acc, elapsed_time, update_count))
699                 if args.no_metrics:
700                     metrics[epoch]['trainloss'].append(train_mean_loss)
701                     metrics[epoch]['trainacc'].append(train_mean_acc)
702                     metrics[epoch]['valloss'].append(val_mean_loss)
703                     metrics[epoch]['valacc'].append(val_mean_acc)
704                 if args.dry_run:
705                     break
706                 if epoch in args.save_at_epoch_pre:
707                     save_model_checkpoint(output_path, epoch=epoch, model=model, optimizer=optimizer,
708                                           name=model_name)
709                 pickle_name = "pretrain-model-{}-epochs-{}.pickle".format(model_name, args.pretrain_epochs)
710                 # Saving metrics in pickle
711                 with open(os.path.join(output_path, pickle_name), 'wb') as pick_file:
712                     pickle.dump(dict(metrics), pick_file)
713                 # Save model + model weights + optimizer state
714                 save_model_checkpoint(output_path, epoch=args.pretrain_epochs, model=model, optimizer=optimizer,
715                                       name=model_name)
716                 print("Finished model {}. Progress: {} / {}".format(model_name, model_i + 1, len(models)))
717
718                 del model # delete and free
719                 torch.cuda.empty_cache()
720
721             def downstream(model_i, model, pretrained_epochs=None):
722                 model_name = model.name if hasattr(model, 'name') else fullname(model)
723                 output_path = os.path.join(args.out_path, model_name + str(model_i))
724                 print("Begin training of {}. Output will be saved to dir: {}".format(model_name, output_path))
725                 # Create dirs and model info
726                 Path(output_path).mkdir(parents=True, exist_ok=True)
727
728                 save_model_architecture(output_path, model, model_name)
729                 save_model_variables_text_only(output_path, model)
730                 model.cuda()
731                 model.train()
732                 # init optimizer
733                 optimizer = Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=3e-4)
734                 scheduler = ReduceLROnPlateau(optimizer, 'min', patience=5, min_lr=3e-10, verbose=True)
735                 best_val_model = None
736                 best_val_loss = None
737                 best_val_epoch = None
738                 metrics = defaultdict(lambda: defaultdict(list))
739                 update_count = 0
740
741                 train_mean_loss = torch.Tensor([0.]).cuda()
742                 val_mean_loss = torch.Tensor([0.]).cuda()
743                 moving_average = 0
744                 if args.downstream_updates_limit > 0:
745                     args.downstream_epochs = 99999999
746                 for epoch in range(1, args.downstream_epochs + 1):
747                     starttime = time.time() # train
748                     for train_loader_i, train_loader in enumerate(downstream_train_loaders):
749                         for dataset_tuple in train_loader:
750                             data, labels = dataset_tuple
751                             data = data.float().cuda()
752                             labels = labels.float().cuda()
753                             optimizer.zero_grad()
754                             pred = model(data, y=None) # makes model return prediction instead of loss
755                             loss = bl.binary_cross_entropy(pred=pred, y=labels,
756                                                       weight=class_weights) # bl.multi_loss_function([bl.
757                                         binary_cross_entropy, bl.MSE_loss])(pred=pred, y=labels)

```

```

757         loss.backward()
758         optimizer.step()
759         update_count += 1
760         train_mean_loss += loss
761         moving_average += 1
762         if not args.no_metrics:
763             with torch.no_grad():
764                 for i, fn in enumerate(metric_functions):
765                     metrics[epoch]['acc_' + str(i)].append(
766                         parse_tensor_to_numpy_or_scalar(fn(y=labels, pred=pred)))
767             # saving metrics
768             metrics[epoch]['trainloss'].append(parse_tensor_to_numpy_or_scalar(loss))
769             del data, pred, labels, loss
770         if args.dry_run:
771             break
772         if args.downstream_updates_limit > 0 and args.downstream_updates_limit <= update_count:
773             break
774         print(f"\tFinished training dataset {train_loader_i}. Progress: {train_loader_i + 1}/{len(
775             downstream_train_loaders)}")
775         torch.cuda.empty_cache()
776         if args.downstream_updates_limit > 0 and args.downstream_updates_limit <= update_count:
777             break
778         train_mean_loss = parse_tensor_to_numpy_or_scalar(train_mean_loss)/moving_average
779         moving_average=0
780         with torch.no_grad():
781             for val_loader_i, val_loader in enumerate(downstream_val_loaders): # validate
782                 for dataset_tuple in val_loader:
783                     data, labels = dataset_tuple
784                     data = data.float().cuda()
785                     labels = labels.float().cuda()
786                     pred = model(data, y=None) # makes model return prediction instead of loss
787                     loss = bl.binary_cross_entropy(pred=pred, y=labels, weight=class_weights)
788                     val_mean_loss += loss
789                     moving_average += 1
790                     if not args.no_metrics:
791                         for i, fn in enumerate(metric_functions):
792                             metrics[epoch]['val_acc_' + str(i)].append(
793                                 parse_tensor_to_numpy_or_scalar(fn(y=labels, pred=pred)))
794                         # saving metrics
795                         metrics[epoch]['valloss'].append(parse_tensor_to_numpy_or_scalar(loss))
796                         del data, pred, labels, loss
797                         if args.dry_run:
798                             break
799                     val_mean_loss = parse_tensor_to_numpy_or_scalar(val_mean_loss)/moving_average
800                     if args.no_metrics:
801                         metrics[epoch]['trainloss'].append(train_mean_loss)
802                         metrics[epoch]['valloss'].append(val_mean_loss)
803
804                     print("\tFinished validation dataset {}. Progress: {}/{}/{}/{}.".format(val_loader_i, val_loader_i + 1,
805                                         len(downstream_val_loaders)))
806                     if epoch in args.save_at_epoch_down:
807                         save_model_checkpoint(output_path, epoch=epoch, model=model, optimizer=optimizer,
808                                   name=model_name)
809                     if best_val_loss is None or np.mean(metrics[epoch]['valloss']) < best_val_loss:
810                         best_val_loss = np.mean(metrics[epoch]['valloss'])
811                         best_val_model = copy.deepcopy(model.state_dict())
812                         best_val_epoch = epoch
813                         scheduler.step(np.mean(metrics[epoch]['valloss']))
814                         elapsed_time = str(datetime.timedelta(seconds=time.time() - starttime))
815                         metrics[epoch]['elapsed_time'].append(elapsed_time)
816                         print("Epoch {} done. Avg train loss: {:.4f}. Avg val loss: {:.4f} Elapsed time: {}. Total optimizer
817 steps: {}.".format(
818                             epoch, args.downstream_epochs, np.mean(metrics[epoch]['trainloss']), np.mean(metrics[epoch]['valloss']),
819                             ), elapsed_time, update_count)
820                     if args.dry_run:
821                         break
822                     if args.downstream_updates_limit > 0 and args.downstream_updates_limit <= update_count:
823                         break
824                     pickle_name = "model-{}-epochs-{}.pickle".format(model_name, epoch)
825                     # Saving metrics in pickle
826                     with open(os.path.join(output_path, pickle_name), 'wb') as pick_file:
827                         pickle.dump(dict(metrics), pick_file)
828                     with open(os.path.join(output_path, 'params.txt'), 'w') as cfg:
829                         if not pretrained_epochs is None:
830                             temp = args.pretrain_epochs #copy
831                             args.pretrain_epochs = pretrained_epochs #change for save
832                             cfg.write(str(args))
833                             args.pretrain_epochs = temp #Change back
834                         else:
835                             cfg.write(str(args))
836                     # Save model + model weights + optimizer state
837                     save_model_checkpoint(output_path, epoch=args.downstream_epochs, model=model, optimizer=optimizer,
838                                   name=model_name)
839                     save_model_state_dict_checkpoint(output_path, epoch=best_val_epoch, model_statedict=best_val_model, optimizer=
None,
840                                   name=model_name, additional_name='_best_val_model')
841                     print("Finished model {}. Output saved to dir: {} Progress: {}/{}/{}/{}.".format(model_name, output_path, model_i +
1,

```

```

842                                         len(models)))
843
844     del model  # delete and free
845     torch.cuda.empty_cache()
846
847     print("Going to train", len(models), 'models')
848     for model_i, model_dict in enumerate(models):  # TODO: easily select what training is necessary!
849         if type(model_dict) == dict:
850             model = copy.deepcopy(model_dict['model'])
851             trained_epochs = None
852             if 'desc' in model_dict:
853                 model.description = model_dict['desc']
854             if 'name' in model_dict:
855                 model.name = model_dict['name']
856             if 'pretrained_epochs' in model_dict:
857                 trained_epochs = model_dict['pretrained_epochs']
858             elif 'will_pretrain' in model_dict and not model_dict['will_pretrain']: #its not an already trained model
859             so...
860                 trained_epochs = 0
861             if model_dict['will_pretrain']:
862                 pretrain(model_i, model)
863             if model_dict['will_downtrain']:
864                 downstream(model_i, model, trained_epochs)
865             else: # assume its a model and the dev was to lazy to make it a dict
866                 model = model_dict
867                 pretrain(model_i, model)
868                 downstream(model_i, model)
869
870     def parse_tensor_to_numpy_or_scalar(input_tensor):
871         try:
872             return input_tensor.item()
873         except:
874             return input_tensor.detach().cpu().numpy()
875
876
877 if __name__ == "__main__":
878     import sys
879
880     print(sys.argv)
881
882     parser = argparse.ArgumentParser(description='Contrastive Predictive Coding')
883     # datapath
884     # Other params
885     parser.add_argument('--saved_model', type=str,
886                         help='Model path to load weights from. Has to be given for downstream mode.')
887
888     parser.add_argument('--pretrain_epochs', type=int, help='The number of Epochs to pretrain', default=100)
889
890     parser.add_argument('--downstream_epochs', type=int, help='The number of Epochs to downtrain', default=100)
891
892     parser.add_argument('--seed', type=int, help='The seed used', default=0)
893
894     parser.add_argument('--forward_mode', help="The forward mode to be used.", default='context',
895                         type=str) # , choices=['context', latents, all']
896
897     parser.add_argument('--out_path', help="The output directory for losses and models",
898                         default='models/' + str(datetime.datetime.now().strftime("%d_%m_%y-%H-%M")) + '-train', type=
899                         str)
900
901     parser.add_argument('--forward_classes', type=int, default=52,
902                         help="The number of possible output classes (only relevant for downstream)")
903
904     parser.add_argument('--warmup_steps', type=int, default=0, help="The number of warmup steps")
905
906     parser.add_argument('--batch_size', type=int, default=24, help="The batch size")
907
908     parser.add_argument('--latent_size', type=int, default=128,
909                         help="The size of the latent encoding for one window")
910
911     parser.add_argument('--timesteps_in', type=int, default=6,
912                         help="The number of windows being used to form a context for prediction")
913
914     parser.add_argument('--timesteps_out', type=int, default=6,
915                         help="The number of windows being predicted from the context (cpc task exclusive)")
916
917     parser.add_argument('--channels', type=int, default=12,
918                         help="The number of channels the data will have") # TODO: auto detect
919
920     parser.add_argument('--window_size', type=int, default=512,
921                         help="The number of datapoints per channel per window")
922
923     parser.add_argument('--crop_size', type=int, default=4500,
924                         help="The size of the data that it is cropped to. If data is smaller than this number, data
925                         gets padded with zeros")
926
927     parser.add_argument('--hidden_size', type=int, default=512,
928                         help="The size of the cell state/context used for predicting future latents or solving
929                         downstream tasks")

```

```

928 parser.add_argument('--dry_run', dest='dry_run', action='store_true',
929                     help="Only run minimal samples to test all models functionality")
930 parser.set_defaults(dry_run=False)
931
932 parser.add_argument('--use_class_weights', dest='use_class_weights', action='store_true',
933                     help="Use class weights determined by datasets class count")
934 parser.set_defaults(use_class_weights=False)
935
936 parser.add_argument('--splits_file', type=str, default='train-test-splits.txt',
937                     help="The Train val test split file to use")
938
939 parser.add_argument('--norm_fn', type=str, default='normalize_std_scaling',
940                     help="The Normalization function to use (from ecg_datasets3")
941
942 parser.add_argument('--save_at_epoch_down', nargs='*', help='Selects additional downstream epochs to save the
943     model weights at.', default=[], type=int)
944
945 parser.add_argument('--save_at_epoch_pre', nargs='*', help='Selects additional pretraining epochs to save the
946     model weights at.', default=[], type=int)
947
948 parser.add_argument('--redo_splits', dest='redo_splits', action='store_true',
949                     help="Redo splits. Warning! File will be overwritten!")
950 parser.set_defaults(redo_splits=False)
951
952 parser.add_argument('--no_metrics', dest='no_metrics', action='store_true',
953                     help="No metrics (loss etc.) will be saved during training")
954 parser.set_defaults(no_metrics=False)
955
956 parser.add_argument('--checkpoint_file_ending', type=str, default=None)
957
958 parser.add_argument("--gpu_device", type=int, default=0)
959
960 parser.add_argument("--comment", type=str, default=None)
961
962 parser.add_argument('--downstream_updates_limit', type=int, default=0)
963
964 parser.add_argument('--downstream_updates_minimum', type=int, default=0)
965
966 parser.add_argument("--preload_fraction", type=float, default=1.)
967
968 args = parser.parse_args()
969 main(args)

```

B.13 main_cpc_explain.py (code)

```

1 import argparse
2 import datetime
3 import functools
4 import os
5 from pathlib import Path
6
7 import numpy as np
8 import torch
9 from torch.optim import Adam
10 from torch.utils.data import DataLoader, ChainDataset
11
12 from architectures_various.explain_network2 import ExplainLabel, ExplainLabelLayer
13 from main_produce_plots_for_tested_models import calculate_best_thresholds
14 from util.data import ecg_datasets2
15 from util.store_models import load_model_checkpoint, load_model_architecture, extract_model_files_from_dir
16
17 from util.visualize.timeseries_to_image_converter import timeseries_to_image_with_gradient_joined,
18     timeseries_to_image_with_gradient_cam
19
20
21 def main(args):
22     np.random.seed(args.seed)
23     torch.cuda.set_device(args.gpu_device)
24     print(f'Device set to : {torch.cuda.current_device()}. Selected was {args.gpu_device}')
25     torch.cuda.manual_seed(args.seed)
26     print(f'Seed set to : {args.seed}')
27     print(f'Model outputpath: {args.out_path}')
28     Path(args.out_path).mkdir(parents=True, exist_ok=True)
29     with open(os.path.join(args.out_path, 'params.txt'), 'w') as cfg:
30         cfg.write(str(args))
31     # georgia = ecg_datasets2.ECGChallengeDatasetBaseline('/home/juwin106/data/georgia/WFDB', window_size=4500,
32     # pad_to_size=4500, use_labels=True)
33     # cpsc_train = ecg_datasets2.ECGChallengeDatasetBaseline('/home/juwin106/data/cpsc_train', window_size=4500,
34     # pad_to_size=4500, use_labels=True)
35     # cpsc = ecg_datasets2.ECGChallengeDatasetBaseline('/home/juwin106/data/cpsc', window_size=4500, pad_to_size=4500,
36     # use_labels=True)
37     # ptbxl = ecg_datasets2.ECGChallengeDatasetBaseline('/home/juwin106/data/ptbxl/WFDB', window_size=4500,
38     # pad_to_size=4500, use_labels=True)
39
40     georgia_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/georgia_challenge/',
41                     window_size=args.crop_size,

```

```

38         pad_to_size=args.crop_size,
39         return_labels=True, return_filename=True,
40         normalize_fn=ecg_datasets2.normalize_mean_scaling)
41 cpdc_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/cps2018_challenge',
42                                         window_size=args.crop_size, pad_to_size=args.crop_size,
43                                         return_labels=True, return_filename=True,
44                                         normalize_fn=ecg_datasets2.normalize_mean_scaling)
45 cpdc2_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/china_challenge',
46                                         window_size=args.crop_size, pad_to_size=args.crop_size
47                                         ,
48                                         return_labels=True, return_filename=True,
49                                         normalize_fn=ecg_datasets2.normalize_mean_scaling)
50 ptbxl_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/ptbxl_challenge',
51                                         window_size=args.crop_size, pad_to_size=args.crop_size
52                                         ,
53                                         return_labels=True, return_filename=True,
54                                         normalize_fn=ecg_datasets2.normalize_mean_scaling)
55
56 a1, b1, ptbxl_test = ptbxl_challenge.generate_datasets_from_split_file()
57 a2, b2, georgia_test = georgia_challenge.generate_datasets_from_split_file()
58 a3, b3, cpdc_challenge = cpdc_challenge.generate_datasets_from_split_file()
59 a4, b4, cpdc2_challenge = cpdc2_challenge.generate_datasets_from_split_file()
60
61 ptbxl_test.randomize_order = False
62 georgia_test.randomize_order = False
63 cpdc_challenge.randomize_order = False
64 cpdc2_challenge.randomize_order = False
65
66 classes = ecg_datasets2.filter_update_classes_by_count(
67     [a1, b1, ptbxl_test, a2, b2, georgia_test, a3, b3, cpdc_challenge, a4, b4, cpdc2_challenge],
68     1) # Set classes if specified in split files (filter out classes with no occurrence)
69 classes_by_index = {v: k for k, v in classes.items()}
70 train_dataset_challenge = ChainDataset([a1, a2, a3, a4])
71 val_dataset_challenge = ChainDataset([b1, b2, b3, b4])
72 test_dataset_challenge = ChainDataset([ptbxl_test, georgia_test, cpdc_challenge, cpdc2_challenge])
73
74 if args.use_class_weights:
75     counts, counted_classes = ecg_datasets2.count_merged_classes(
76         [a1, b1, ptbxl_test, a2, b2, georgia_test, a3, b3, cpdc_challenge, a4, b4, cpdc2_challenge])
77     class_weights = torch.Tensor(max(counts) / counts).to(device=f'cuda:{args.gpu_device}')
78     print('Using the following class weights:', class_weights)
79 else:
80     class_weights = None
81
82 # all_dataset_challenge = ChainDataset[ptbxl_challenge, georgia_challenge, cpdc_challenge, cpdc2_challenge]
83 model_train_folders = [
84     '#v14
85     #'home/julian/Downloads/Github/contrastive-predictive-coding/models/10_08_21-14_42-train/bl_TCN_down+
86     bl_cnn_v1+bl_cnn_v14+bl_cnn_v15+bl_cnn_v8/architectures_baseline_challenge.baseline_cnn_v14.BaselineNet3|train-
87     test-splits-fewer-labels60|use_weights|ConvLyrs:29|MaxPool|Linear|BatchNorm|stride_sum:71|dilation_sum:30|
88     padding_sum:22|krnls_sum:143'
89     #'home/julian/Downloads/Github/contrastive-predictive-coding/models/14_08_21-15_36-train|(4x)cpc/
90     architectures_cpc.cpc_combined.CPCCombined0|train-test-splits-fewer-labels60|use_weights|frozen|C|m:all|
91     cpc_downstream_cnn'
92     #'home/julian/Downloads/Github/contrastive-predictive-coding/models/21_05_21-11-train/bl_cnn_v0+bl_cnn_v0_1+
93     bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v14+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4+bl_cnn_v5+bl_cnn_v6+bl_cnn_v8+
94     bl_cnn_v9/architectures_baseline_challenge.baseline_cnn_v14.BaselineNet12|ConvLyrs:29|MaxPool|Linear|BatchNorm|
95     stride_sum:71|dilation_sum:30|padding_sum:22|krnls_sum:143'# v14
96     #'home/julian/Downloads/Github/contrastive-predictive-coding/models/10_08_21-14_42-train/bl_TCN_down+
97     bl_cnn_v1+bl_cnn_v14+bl_cnn_v15+bl_cnn_v8/architectures_baseline_challenge.baseline_cnn_v15.BaselineNet4|train-
98     test-splits-fewer-labels60|use_weights|ConvLyrs:5|MaxPool|Linear|BatchNorm|stride_sum:14|dilation_sum:96|
99     padding_sum:0|krnls_sum:26'|v15
    #new v14
    #'home/julian/Downloads/Github/contrastive-predictive-coding/models/30_11_21-18-train/bl_cnn_v14/
    architectures_baseline_challenge.baseline_cnn_v14.BaselineNet0|use_weights|ConvLyrs:29|MaxPool|Linear|BatchNorm|
    stride_sum:71|dilation_sum:30|padding_sum:22|krnls_sum:143'
    #newtcn
    #'home/julian/Downloads/Github/contrastive-predictive-coding/models/02_12_21-12-train|bl_TCN_down+bl_cnn_v2/
    architectures_baseline_challenge.baseline_TCN_down.BaselineNet1|use_weights|ConvLyrs:15|Linear|stride_sum:15|
    dilation_sum:31|padding_sum:56|krnls_sum:39'
    #new cpc
    #'home/julian/Downloads/Github/contrastive-predictive-coding/models/02_12_21-16-train|(4x)cpc/
    architectures_cpc.cpc_combined.CPCCombined2|use_weights|frozen|C|m:same|cpc_downstream_only'
    #b1v8
    #'home/julian/Downloads/Github/contrastive-predictive-coding/models/14_12_21-10-train|bl_FCN+bl_MLP+
    bl_TCN_block+bl_TCN_down+bl_TCN_last+bl_alex_v2+bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+
    bl_cnn_v14+bl_cnn_v15+bl_cnn_v2+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+bl_cnn_v8+bl_cnn_v9/
    architectures_baseline_challenge.baseline_cnn_v8.BaselineNet11|use_weights|ConvLyrs:5|MaxPool|Linear|BatchNorm|
    stride_sum:14|dilation_sum:19|padding_sum:0|krnls_sum:22'
]
model_test_folders = [
    '#v14
    #'home/julian/Downloads/Github/contrastive-predictive-coding/models/11_08_21-15_58-test|(10x)bl_TCN_down+(10x)
    )bl_cnn_v1+(10x)bl_cnn_v14+(10x)bl_cnn_v15+(10x)bl_cnn_v8/10_08_21-14_42-train|bl_TCN_down+bl_cnn_v1+bl_cnn_v14+
    bl_cnn_v15+bl_cnn_v8/architectures_baseline_challenge.baseline_cnn_v14.BaselineNet3|train-test-splits-fewer-
    labels60|use_weights|ConvLyrs:29|MaxPool|Linear|BatchNorm|stride_sum:71|dilation_sum:30|padding_sum:22|krnls_sum
    :143'
    #'home/julian/Downloads/Github/contrastive-predictive-coding/models/16_08_21-10-16-test|(40x)cpc/14_08_21-15
    _36-train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined0|train-test-splits-fewer-labels60|use_weights|
    frozen|C|m:all|cpc_downstream_cnn'
]

```

```

100     # v14
101     #'./home/julian/Downloads/Github/contrastive-predictive-coding/models/26_06_21-15-test|(2x)bl_MLP+bl_FCN+
102     bl_TCN_block+bl_TCN_down+bl_TCN_flatten+bl_TCN_last+bl_alex_v2+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+
103     bl_cnn_v1+bl_cnn_v14+bl_cnn_v15+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4+bl_cnn_v5+bl_cnn_v6+bl_cnn_v7+bl_cnn_v8+bl_cnn_v9+21
104     _05_21-11-train|bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v14+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4
105     +bl_cnn_v5+bl_cnn_v6+bl_cnn_v8+bl_cnn_v9/architectures_baseline_challenge.baseline_cnn_v14.BaselineNet12|dte
106     :120'
107     #cpc
108     #'./home/julian/Downloads/Github/contrastive-predictive-coding/models/13_08_21-10-47-test|(80x)cpc/11_08_21-19
109     _56-train|(8x)cpc/architectures_cpc.cpc_combined.CPCCombined6|train-test-splits-fewer-labels60|use_weights|
110     frozen|L|m:all|cpc_downstream_cnn'
111     #v15
112     #'./home/julian/Downloads/Github/contrastive-predictive-coding/models/11_08_21-15-58-test|(10x)bl_TCN_down+(10x)
113     )bl_cnn_v1+(10x)bl_cnn_v14+(10x)bl_cnn_v15+(10x)bl_cnn_v8/10_08_21-14_42-train|bl_TCN_down+bl_cnn_v1+bl_cnn_v14+
114     bl_cnn_v15+bl_cnn_v8/architectures_baseline_challenge.baseline_cnn_v15.BaselineNet4|train-test-splits-fewer-
115     labels60|use_weights|ConvLyrs:5|MaxPool|Linear|BatchNorm|stride_sum:14|dilation_sum:96|padding_sum:0|krnls_sum
116     :26'
117     #new v14
118     #'./home/julian/Downloads/Github/contrastive-predictive-coding/models/30_11_21-20-25-test|bl_cnn_v14/30_11_21
119     -18-train|bl_cnn_v14/architectures_baseline_challenge.baseline_cnn_v14.BaselineNet0|use_weights|ConvLyrs:29|
120     MaxPool|Linear|BatchNorm|stride_sum:71|dilation_sum:30|padding_sum:22|krnls_sum:143'
121     #new tcn
122     #'./home/julian/Downloads/Github/contrastive-predictive-coding/models/02_12_21-14-58-test|bl_TCN_down+bl_cnn_v2
123     /02_12_21-12-train|bl_TCN_down+bl_cnn_v2/architectures_baseline_challenge.baseline_TCN_down.BaselineNet1|
124     use_weights|ConvLyrs:15|Linear|stride_sum:15|dilation_sum:31|padding_sum:56|krnls_sum:39'
125     #new cpc
126     #'./home/julian/Downloads/Github/contrastive-predictive-coding/models/02_12_21-17-56-test|(4x)cpc/02_12_21-16-
127     train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined2|use_weights|frozen|C|m:same|cpc_downstream_only'
128     #blv8
129     #'./home/julian/Downloads/Github/contrastive-predictive-coding/models/15_12_21-15-57-test|bl_FCN+bl_MLP+
130     bl_TCN_block+bl_TCN_down+bl_TCN_last+bl_alex_v2+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+
131     bl_cnn_v14+bl_cnn_v15+bl_cnn_v2+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+bl_cnn_v8+bl_cnn_v9/14_12_21-10-train|bl_MLP+
132     bl_alex_v2+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v2+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+
133     bl_cnn_v8+bl_cnn_v9/architectures_baseline_challenge.baseline_cnn_v8.BaselineNet11|use_weights|ConvLyrs:5|
134     MaxPool|Linear|BatchNorm|stride_sum:14|dilation_sum:19|padding_sum:0|krnls_sum:22'
135
136 # infer class from model-arch file
137 model_dicts = []
138 for train_folder, test_folder in zip(model_train_folders, model_test_folders):
139     model_files = extract_model_files_from_dir(train_folder)
140     for mfile in model_files:
141         fm_fs, cp_fs, root = mfile
142         fm_f = fm_fs[0]
143         cp_f = sorted(cp_fs)[-1]
144         model = load_model_architecture(fm_f)
145         if model is None:
146             continue
147         model, _, epoch = load_model_checkpoint(cp_f, model, optimizer=None, device_id=f'cuda:{args.gpu_device}')
148         if hasattr(model, 'freeze_cpc'):
149             model.freeze_cpc = False
150         thresholds = calculate_best_thresholds([test_folder])[0]
151         print(thresholds)
152         print()
153         if len(model_dicts) == 0:
154             print(f"Found architecturefile {os.path.basename(fm_f)}, checkpointfile {os.path.basename(cp_f)} in folder {root}. Appending model for testing.")
155             explain_model = ExplainLabel(model, class_weights=class_weights)
156             model_dicts.append({'model': explain_model, 'model_folder': root, 'thresholds': thresholds,
157                                 'name': train_folder.split(os.path.sep)[-1].split('|')[0].split('.')[1]})
```

```

168         N_SAMPLES -= 1
169         data, labels, filenames = dataset_tuple
170         data = data.float().cuda()
171         labels = labels.float().cuda()
172         # for
173         # NORMAL GRADIENT
174         optimizer.zero_grad()
175         pred, _ = model(data, y=labels) # obtain pred once
176         pred = pred.detach()
177         optimizer.zero_grad()
178         # _, grad1 = model(data, y=labels)# explain_class=
179         # timeseries_to_image_with_gradient(data.detach().cpu(), labels.detach().cpu(), grad1.detach().cpu
180         (), pred.detach().cpu(), model_thresholds=thresholds, title='Gradient for actual label\n', filenames=filenames,
181         save=True, show=True)
182         filenames = list(
183             map(lambda x: os.path.join(output_dir, x), map(os.path.basename, filenames)))
184         # plot_prediction_scatterplots(labels.detach().cpu(), pred.detach().cpu(), model_thresholds=
185         thresholds, filename=filenames[0], save=True, show=False)
186         # print(labels)
187         # with open('temp-grad.pickle', 'wb') as f:
188         #     pickle.dump([data.detach().cpu(), labels.detach().cpu(), grad1.detach().cpu(), pred.detach()
189         .cpu()], f, protocol=pickle.HIGHEST_PROTOCOL)
190         #     print("saved")
191
192         # INVERSE GRADIENT
193         # optimizer.zero_grad()
194         # _, grad2 = model(data, y=1-labels)# explain_class=
195         # # print(1-labels)
196         # # with open('temp-grad-inverse.pickle', 'wb') as f:
197         # #     pickle.dump([data.detach().cpu(), (1-labels).detach().cpu(), grad2.detach().cpu(), pred.
198         detach().cpu()], f, protocol=pickle.HIGHEST_PROTOCOL)
199         # #     print("done")
200         # # timeseries_to_image_with_gradient(data.detach().cpu(), labels.detach().cpu(), grad2.detach().cpu
201         (), pred.detach().cpu(), model_thresholds=thresholds, title='Gradient for inverted label\n', save=False, show=
202         True)
203
204         # for class_i in torch.nonzero(labels, as_tuple=False): #only do for first image in batch
205         #     optimizer.zero_grad()
206         #     l = labels.clone() #torch.zeros_like(labels, device=labels.device)
207         #     l[tuple(class_i)] = 0.
208         #     _, gradi = model(data, y=1)
209         #     class_name = class_i[1].item()
210         #     timeseries_to_image_with_gradient(data.detach().cpu(), labels.detach().cpu(), gradi.detach()
211         .cpu(), pred.detach().cpu(), model_thresholds=thresholds, title=f'Gradient for class:{class_name} as label\n',
212         save=False, show=True)
213         # with open(f'temp-grad{class_i}.pickle', 'wb') as f:
214         #     pickle.dump([data.detach().cpu(), l.detach().cpu(), gradi.detach().cpu(), pred.detach().cpu
215         ()], f, protocol=pickle.HIGHEST_PROTOCOL)
216         #     print("saved")
217         grads = []
218         class_names = []
219         for class_i in torch.nonzero(labels, as_tuple=False): # only do for first image in batch
220             optimizer.zero_grad()
221             l = torch.zeros_like(labels, device=labels.device) # pred.clone()
222             l[tuple(class_i)] = 1.
223             _, gradi = model(data, y=1)
224             gradi = gradi.abs()
225             #gradi[gradi < 0] = 0#comment out after
226             gradi = gradi - gradi.min(keepdim=True, dim=1)[0]
227             gradi = gradi / (1e-7 + gradi.max(keepdim=True, dim=1)[0])
228             print('data shape', data.shape, 'grad shape', gradi.shape)
229             class_names.append(class_i[1].item())
230             grads.append(gradi.detach().cpu())
231             timeseries_to_image_with_gradient_cam(data.detach().cpu(), labels.detach().cpu(), grads,
232             pred.detach().cpu(), model_thresholds=thresholds,
233             class_name_list=class_names,
234             filenames=filenames, save=True, show=False)
235             # timeseries_to_image_with_gradient_joined(data.detach().cpu(), labels.detach().cpu(), grads,
236             # pred.detach().cpu(), model_thresholds=thresholds,
237             # grad_alteration='abs', class_name_list=class_names,
238             # filenames=filenames, save=True, show=False)
239             # timeseries_to_image_with_gradient_joined(data.detach().cpu(), labels.detach().cpu(), grads,
240             # pred.detach().cpu(), model_thresholds=thresholds,
241             # grad_alteration='relu', class_name_list=class_names,
242             # filenames=filenames, save=True, show=False)
243             # timeseries_to_image_with_gradient_joined(data.detach().cpu(), labels.detach().cpu(), grads,
244             # pred.detach().cpu(), model_thresholds=thresholds,
245             # grad_alteration='relu_neg', class_name_list=class_names
246
247             #
248             filenames=filenames, save=True, show=False)
249             # with open(f'temp-grad{class_i}.pickle', 'wb') as f:
250             #     pickle.dump([data.detach().cpu(), l.detach().cpu(), gradi.detach().cpu(), pred.detach().cpu
251             ()], f, protocol=pickle.HIGHEST_PROTOCOL)
252             #     print("saved")
253
254             # for class_i in range(labels.shape[1]):
255             #     optimizer.zero_grad()
256             #     l = pred.clone() #torch.zeros_like(labels, device=labels.device)
257             #     l[:, class_i] = 1
258             #     _, gradi = model(data, y=1)# explain_class=

```

```

246         #     print(l)
247         #     timeseries_to_image_with_gradient(data.detach().cpu(), labels.detach().cpu(), gradi.detach()
248         .cpu(), pred.detach().cpu(), model_thresholds=thresholds, save=False, show=True)
249         #     with open(f'temp-grad(class_{i}).pickle', 'wb') as f:
250         #         pickle.dump([data.detach().cpu(), l.detach().cpu(), gradi.detach().cpu(), pred.detach()]
251         .cpu(), f, protocol=pickle.HIGHEST_PROTOCOL)
252         #     print("saved")
253         # for class_i in range(labels.shape[1]):
254         #     optimizer.zero_grad()
255         #     l = pred.clone()
256         #     l[:, class_i] = 0
257         #     _, gradi = model(data, y=l) # explain_class=
258         #     print(l)
259         #     with open(f'temp-grad-inverse(class_{i}).pickle', 'wb') as f:
260         #         pickle.dump([data.detach().cpu(), labels.detach().cpu(), gradi.detach().cpu(), pred.
261         detach().cpu()], f, protocol=pickle.HIGHEST_PROTOCOL)
262         #     print("saved")
263         if args.dry_run:
264             break
265         del data, pred, labels
266         print("\tFinished dataset {}. Progress: {}/{}/{}.".format(loader_i, loader_i + 1, len(loaders)))
267         torch.cuda.empty_cache()
268     if args.dry_run:
269         break
270     del model # delete and free
271     torch.cuda.empty_cache()
272
273 def save_dict_to_csv_file(filepath, data, column_names=None):
274     with open(filepath) as f:
275         if not column_names is None:
276             f.write(','.join(column_names) + '\n')
277         for dc in data:
278             f.write(','.join(dc) + '\n')
279
280 def parse_tensor_to_numpy_or_scalar(input_tensor):
281     if type(input_tensor) == torch.Tensor:
282         arr = input_tensor.detach().cpu().numpy() if input_tensor.is_cuda else input_tensor.numpy()
283         if arr.size == 1:
284             return arr.item()
285         return arr
286     return input_tensor
287
288 if __name__ == "__main__":
289     import sys
290
291     print(sys.argv)
292
293     parser = argparse.ArgumentParser(description='Contrastive Predictive Coding')
294     # datapath
295     # Other params
296     parser.add_argument('--saved_model', type=str,
297                         help='Model path to load weights from. Has to be given for downstream mode.')
298
299     parser.add_argument('--seed', type=int, help='The seed used', default=0)
300
301     parser.add_argument('--forward_mode', help="The forward mode to be used.", default='context',
302                         type=str) # , choices=['context, latents, all']
303
304     parser.add_argument('--out_path', help="The output directory for losses and models",
305                         default='models/' + str(datetime.datetime.now().strftime("%d_%m_%y-%H")) + '-explain', type=
306                         str)
307
308     parser.add_argument('--forward_classes', type=int, default=52,
309                         help="The number of possible output classes (only relevant for downstream)")
310
311     parser.add_argument('--warmup_steps', type=int, default=0, help="The number of warmup steps")
312
313     parser.add_argument('--batch_size', type=int, default=24, help="The batch size")
314
315     parser.add_argument('--latent_size', type=int, default=128,
316                         help="The size of the latent encoding for one window")
317
318     parser.add_argument('--timesteps_in', type=int, default=6,
319                         help="The number of windows being used to form a context for prediction")
320
321     parser.add_argument('--timesteps_out', type=int, default=6,
322                         help="The number of windows being predicted from the context (cpc task exclusive)")
323
324     parser.add_argument('--crop_size', type=int, default=4500,
325                         help="The size of the data that it is cropped to. If data is smaller than this number, data
326                         gets padded with zeros")
327
328     parser.add_argument('--channels', type=int, default=12,
329                         help="The number of channels the data will have") # TODO: auto detect
330
331     parser.add_argument('--window_size', type=int, default=512,
332                         help="The number of datapoints per channel per window")

```

```

331 parser.add_argument('--hidden_size', type=int, default=512,
332     help="The size of the cell state/context used for predicting future latents or solving
333     downstream tasks")
334
335 parser.add_argument('--dry_run', dest='dry_run', action='store_true',
336     help="Only run minimal samples to test all models functionality")
337 parser.set_defaults(dry_run=False)
338
339 parser.add_argument('--use_class_weights', dest='use_class_weights', action='store_true',
340     help="Use class weights determined by datasets class count")
341 parser.set_defaults(use_class_weights=False)
342
343 parser.add_argument("--gpu_device", type=int, default=0)
344
345 args = parser.parse_args()
346 main(args)

```

B.14 main_cpc_explain_gradcam.py (code)

```

1 import argparse
2 import datetime
3 import random
4 import functools
5 import os
6 from pathlib import Path
7
8 import numpy as np
9 import torch
10 from torch.optim import Adam
11 from torch.utils.data import DataLoader, ChainDataset
12
13 from architectures_various.explain_network2 import ExplainLabel, ExplainLabelLayer
14 from main_produce_plots_for_tested_models import calculate_best_thresholds
15 from util.data import ecg_datasets2
16 from util.store_models import load_model_checkpoint, load_model_architecture, extract_model_files_from_dir
17
18 from util.visualize.timeseries_to_image_converter import timeseries_to_image_with_gradient_joined, \
19     timeseries_to_image_with_gradient_cam
20
21
22 def main(args):
23     np.random.seed(args.seed)
24     random.seed(args.seed)
25     torch.cuda.set_device(args.gpu_device)
26     print(f'Device set to : {torch.cuda.current_device()}. Selected was {args.gpu_device}')
27     torch.cuda.manual_seed(args.seed)
28     print(f'Seed set to : {args.seed}.')
29     print(f'Model outputpath: {args.out_path}')
30     Path(args.out_path).mkdir(parents=True, exist_ok=True)
31     with open(os.path.join(args.out_path, 'params.txt'), 'w') as cfg:
32         cfg.write(str(args))
33     # georgia = ecg_datasets2.ECGChallengeDatasetBaseline('/home/juwin106/data/georgia/WFDB', window_size=4500,
34     # pad_to_size=4500, use_labels=True)
35     # cpsc_train = ecg_datasets2.ECGChallengeDatasetBaseline('/home/juwin106/data/cpsc_train', window_size=4500,
36     # pad_to_size=4500, use_labels=True)
37     # cpsc = ecg_datasets2.ECGChallengeDatasetBaseline('/home/juwin106/data/cpsc', window_size=4500, pad_to_size=4500,
38     # use_labels=True)
39     # ptbxl = ecg_datasets2.ECGChallengeDatasetBaseline('/home/juwin106/data/ptbxl/WFDB', window_size=4500,
40     # pad_to_size=4500, use_labels=True)
41
42     georgia_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/georgia_challenge/',
43                     window_size=args.crop_size,
44                     pad_to_size=args.crop_size,
45                     return_labels=True, return_filename=True,
46                     normalize_fn=ecg_datasets2.normalize_mean_scaling)
47     cpsc_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/cps2018_challenge/',
48                     window_size=args.crop_size, pad_to_size=args.crop_size,
49                     return_labels=True, return_filename=True,
50                     normalize_fn=ecg_datasets2.normalize_mean_scaling)
51     cpsc2_challenge = ecg_datasets2.ECGChallengeDatasetBaseline('/media/julian/data/data/ECG/china_challenge',
52                     window_size=args.crop_size, pad_to_size=args.crop_size
53
54                     return_labels=True, return_filename=True,
55                     normalize_fn=ecg_datasets2.normalize_mean_scaling)
56
57     a1, b1, ptbxl_test = ptbxl_challenge.generate_datasets_from_split_file()
58     a2, b2, georgia_test = georgia_challenge.generate_datasets_from_split_file()
59     a3, b3, cpsc_test = cpsc_challenge.generate_datasets_from_split_file()
60     a4, b4, cpsc2_test = cpsc2_challenge.generate_datasets_from_split_file()

```

```

61 ptbxl_test.randomize_order = False
62 georgia_test.randomize_order = False
63 cpsc_test.randomize_order = False
64 cpsc2_test.randomize_order = False
65
66 classes = ecg_datasets2.filter_update_classes_by_count(
67     [a1, b1, ptbxl_test, a2, b2, georgia_test, a3, b3, cpsc_test, a4, b4, cpsc2_test],
68     1) # Set classes if specified in split files (filter out classes with no occurrence)
69 classes_by_index = {v: k for k, v in classes.items()}
70 train_dataset_challenge = ChainDataset([a1, a2, a3, a4])
71 val_dataset_challenge = ChainDataset([b1, b2, b3, b4])
72 test_dataset_challenge = ChainDataset([ptbxl_test, georgia_test, cpsc_test, cpsc2_test])
73 print("Use class weights:", args.use_class_weights)
74 if args.use_class_weights:
75     counts, counted_classes = ecg_datasets2.count_merged_classes(
76         [a1, b1, ptbxl_test, a2, b2, georgia_test, a3, b3, cpsc_test, a4, b4, cpsc2_test])
77     class_weights = torch.Tensor(max(counts) / counts).to(device='cuda:(args.gpu_device)')
78     print('Using the following class weights:', class_weights)
79 else:
80     class_weights = None
81
82 # all_dataset_challenge = ChainDataset[ptbxl_challenge, georgia_challenge, cpsc_challenge, cpsc2_challenge]
83 model_train_folders = [
84     #v14
85     #'home/julian/Downloads/Github/contrastive-predictive-coding/models/10_08_21-14_42-train|bl_TCN_down+
86     bl_cnn_v1+bl_cnn_v14+bl_cnn_v15+bl_cnn_v8/architectures_baseline_challenge.baseline_cnn_v14.BaselineNet3|train-
87     test-splits-fewer-labels60|use_weights|ConvLyrs:29|MaxPool|Linear|BatchNorm|stride_sum:71|dilation_sum:30|
88     padding_sum:22|krnls_sum:143'
89     #cpc
90     # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/14_08_21-15_36-train|(4x)cpc/
91     architectures_cpc.cpc_combined.CPCCombined0|train-test-splits-fewer-labels60|use_weights|frozen|C|m:all|
92     cpc_downstream_cnn'
93     #'home/julian/Downloads/Github/contrastive-predictive-coding/models/21_05_21-11-train|bl_cnn_v0+bl_cnn_v0_1+
94     bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v14+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4+bl_cnn_v5+bl_cnn_v6+bl_cnn_v8+
95     bl_cnn_v9/architectures_baseline_challenge.baseline_cnn_v14.BaselineNet12|ConvLyrs:29|MaxPool|Linear|BatchNorm|
96     stride_sum:71|dilation_sum:30|padding_sum:22|krnls_sum:143'# v14
97     #'home/julian/Downloads/Github/contrastive-predictive-coding/models/10_08_21-14_42-train|bl_TCN_down+
98     bl_cnn_v1+bl_cnn_v14+bl_cnn_v15+bl_cnn_v8/architectures_baseline_challenge.baseline_cnn_v15.BaselineNet4|train-
99     test-splits-fewer-labels60|use_weights|ConvLyrs:5|MaxPool|Linear|BatchNorm|stride_sum:14|dilation_sum:96|
100    padding_sum:0|krnls_sum:26'#v15
101    #new CPC
102    # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/01_12_21-10-train|(12x)cpc/
103    architectures_cpc.cpc_combined.CPCCombined9|use_weights|strided|frozen|L|m:same|cpc_downstream_cnn'
104    #new bl14
105    # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/14_12_21-10-train|bl_FCN+bl_MLP+
106    bl_TCN_block+bl_TCN_down+bl_TCN_last+bl_alex_v2+bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+
107    bl_cnn_v14+bl_cnn_v15+bl_cnn_v2+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+bl_cnn_v8+bl_cnn_v9/
108    architectures_baseline_challenge.baseline_cnn_v14.BaselineNet3|use_weights|ConvLyrs:29|MaxPool|Linear|BatchNorm|
109    stride_sum:71|dilation_sum:30|padding_sum:22|krnls_sum:143'
110    #new bl v8
111    # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/14_12_21-10-train|bl_FCN+bl_MLP+
112    bl_TCN_block+bl_TCN_down+bl_TCN_last+bl_alex_v2+bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+
113    bl_cnn_v14+bl_cnn_v15+bl_cnn_v2+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+bl_cnn_v8+bl_cnn_v9/
114    architectures_baseline_challenge.baseline_cnn_v8.BaselineNet11|use_weights|ConvLyrs:5|MaxPool|Linear|BatchNorm|
115    stride_sum:14|dilation_sum:19|padding_sum:0|krnls_sum:22'
116    #new blvtn
117    # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/14_12_21-10-train|bl_FCN+bl_MLP+
118    bl_TCN_block+bl_TCN_down+bl_TCN_last+bl_alex_v2+bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+
119    bl_cnn_v14+bl_cnn_v15+bl_cnn_v2+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+bl_cnn_v8+bl_cnn_v9/
120    architectures_baseline_challenge.baseline_TCN_down.BaselineNet1|use_weights|ConvLyrs:15|Linear|stride_sum:15|
121    dilation_sum:31|padding_sum:56|krnls_sum:39'
122    #new cpc v8
123    # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/23_12_21-18-09-train|(8x)cpc/
124    architectures_cpc.cpc_combined.CPCCombined0|train-test-splits|use_weights|frozen|C|L|m:all|
125    cpc_downstream_twolinear_v2'
126    #new cpc v0
127    # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/02_12_21-16-train|(4x)cpc/
128    architectures_cpc.cpc_combined.CPCCombined2|use_weights|frozen|C|m:same|cpc_downstream_only'
129    #cpc v8 nocontext
130    # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/07_02_22-13-29-train|(30x)cpc/
131    architectures_cpc.cpc_combined.CPCCombined2|train-test-splits|use_weights|unfrozen|C|L|m:crossentropy-nocontext|
132    cpc_downstream_twolinear_v2'
133    #cpc v8
134    # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/07_02_22-13-29-train|(30x)cpc/
135    architectures_cpc.cpc_combined.CPCCombined17|train-test-splits|use_weights|frozen|C|L|m:crossentropy|
136    cpc_downstream_twolinear_v2'
137 ]
138 model_test_folders = [
139     #v14
140     #'home/julian/Downloads/Github/contrastive-predictive-coding/models/11_08_21-15_58-test|(10x)bl_TCN_down+(10x
141     )bl_cnn_v1+(10x)bl_cnn_v14+(10x)bl_cnn_v15+(10x)bl_cnn_v8/10_08_21-14_42-train|bl_TCN_down+bl_cnn_v1+bl_cnn_v14+
142     bl_cnn_v15+bl_cnn_v8/architectures_baseline_challenge.baseline_cnn_v14.BaselineNet3|train-test-splits-fewer-
143     labels60|use_weights|ConvLyrs:29|MaxPool|Linear|BatchNorm|stride_sum:71|dilation_sum:30|padding_sum:22|krnls_sum
144     :143'
145     #cpc
146     # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/16_08_21-10-16-test|(40x)cpc/14_08_21-15
147     _36-train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined0|train-test-splits-fewer-labels60|use_weights|
148     frozen|C|m:all|cpc_downstream_cnn'
149     # v14

```

```

113      #' /home/julian/Downloads/Github/contrastive-predictive-coding/models/26_06_21-15-test | (2x)bl_MLP+bl_FCN+
114      bl_TCN_block+bl_TCN_down+bl_TCN_flatten+bl_TCN_last+bl_alex_v2+bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+
115      bl_cnn_v1+bl_cnn_v14+bl_cnn_v15+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4+bl_cnn_v5+bl_cnn_v6+bl_cnn_v7+bl_cnn_v8+bl_cnn_v21
116      _05_21-11-train|bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v14+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4
117      +bl_cnn_v5+bl_cnn_v6+bl_cnn_v8+bl_cnn_v9/architectures_baseline_challenge.baseline_cnn_v14.BaselineNet12|dte
118      :120'
119      #cpc
120      #' /home/julian/Downloads/Github/contrastive-predictive-coding/models/13_08_21-10-47-test | (80x)cpc/11_08_21-19
121      _56-train|(8x)cpc/architectures_cpc.cpc_combined.CPCCombined6|train-test-splits-fewer-labels60|use_weights|
122      frozen|L|m:all|cpc_downstream_cnn'
123      #v15
124      #' /home/julian/Downloads/Github/contrastive-predictive-coding/models/11_08_21-15-58-test | (10x)bl_TCN_down+(10x)
125      )bl_cnn_v1+(10x)bl_cnn_v14+(10x)bl_cnn_v15+(10x)bl_cnn_v8/10_08_21-14_42-train|bl_TCN_down+bl_cnn_v1+bl_cnn_v14+
126      bl_cnn_v15+bl_cnn_v8/architectures_baseline_challenge.baseline_cnn_v15.BaselineNet4|train-test-splits-fewer-
127      labels60|use_weights|ConvLyrs:5|MaxPool|Linear|BatchNorm|stride_sum:14|dilation_sum:96|padding_sum:0|krnls_sum
128      :26'
129      #new cpc
130      #' /home/julian/Downloads/Github/contrastive-predictive-coding/models/01_12_21-12-48-test | (12x)cpc/01_12_21-10-
131      train|(12x)cpc/architectures_cpc.cpc_combined.CPCCombined9|use_weights|strided|frozen|L|m:same|
132      cpc_downstream_cnn'
133      #new bl14
134      #' /home/julian/Downloads/Github/contrastive-predictive-coding/models/15_12_21-15-57-test | bl_FCN+bl_MLP+
135      bl_TCN_block+bl_TCN_down+bl_TCN_last+bl_alex_v2+bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+
136      bl_cnn_v14+bl_cnn_v15+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+bl_cnn_v8+bl_cnn_v9/14_12_21-10-train|bl_MLP+
137      bl_alex_v2+bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v2+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+
138      bl_cnn_v8+bl_cnn_v9/architectures_baseline_challenge.baseline_cnn_v14.BaselineNet3|use_weights|ConvLyrs:29|
139      MaxPool|Linear|BatchNorm|stride_sum:71|dilation_sum:30|padding_sum:22|krnls_sum:143'
140      #new bl v8
141      #' /home/julian/Downloads/Github/contrastive-predictive-coding/models/15_12_21-15-57-test | bl_FCN+bl_MLP+
142      bl_TCN_block+bl_TCN_down+bl_TCN_last+bl_alex_v2+bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+
143      bl_cnn_v14+bl_cnn_v15+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+bl_cnn_v8+bl_cnn_v9/14_12_21-10-train|bl_MLP+
144      bl_alex_v2+bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v2+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+
145      bl_cnn_v8+bl_cnn_v9/architectures_baseline_challenge.baseline_cnn_v8.BaselineNet11|use_weights|ConvLyrs:5|
146      MaxPool|Linear|BatchNorm|stride_sum:14|dilation_sum:19|padding_sum:0|krnls_sum:22'
147      #new bltn
148      #' /home/julian/Downloads/Github/contrastive-predictive-coding/models/15_12_21-15-57-test | bl_FCN+bl_MLP+
149      bl_TCN_block+bl_TCN_down+bl_TCN_last+bl_alex_v2+bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+
150      bl_cnn_v14+bl_cnn_v15+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+bl_cnn_v8+bl_cnn_v9/14_12_21-10-train|bl_MLP+
151      bl_alex_v2+bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v2+bl_cnn_v4+bl_cnn_v5+bl_cnn_v7+
152      bl_cnn_v8+bl_cnn_v9/architectures_baseline_challenge.baseline_TCN_down.BaselineNet1|use_weights|ConvLyrs:15|
153      Linear|stride_sum:15|dilation_sum:31|padding_sum:56|krnls_sum:39'
154      #new cpc
155      #' /home/julian/Downloads/Github/contrastive-predictive-coding/models/24_12_21-11-14-test | (8x)cpc/23_12_21
156      -18-09-train|(8x)cpc/architectures_cpc.cpc_combined.CPCCombined0|train-test-splits|use_weights|frozen|C|L|m:all|
157      cpc_downstream_twolinear_v2'
158      #new cpc v0
159      #' /home/julian/Downloads/Github/contrastive-predictive-coding/models/02_12_21-17-56-test | (4x)cpc/02_12_21-16-
160      train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined2|use_weights|frozen|C|m:same|cpc_downstream_only'
161      #likev8 cpc nocontext
162      #' /home/julian/Downloads/Github/contrastive-predictive-coding/models/07_02_22-15-03-test | (36x)cpc/07_02_22
163      -13-29-train|(30x)cpc/architectures_cpc.cpc_combined.CPCCombined2|train-test-splits|use_weights|unfrozen|C|L|m:
164      crossentropy-nocontext|cpc_downstream_twolinear_v2'
165      #likev8 cpc
166      #' /home/julian/Downloads/Github/contrastive-predictive-coding/models/07_02_22-15-03-test | (36x)cpc/07_02_22
167      -13-29-train|(30x)cpc/architectures_cpc.cpc_combined.CPCCombined17|train-test-splits|use_weights|frozen|C|L|m:
168      crossentropy|cpc_downstream_twolinear_v2'
169      '
170      # infer class from model-arch file
171      model_dicts = []
172      for train_folder, test_folder in zip(model_train_folders, model_test_folders):
173          model_files = extract_model_files_from_dir(train_folder)
174          for mfile in model_files:
175              fm_fs, cp_fs, root = mfile
176              fm_f = fm_fs[0]
177              cp_f = sorted(cp_fs)[-1]
178              model = load_model_architecture(fm_f)
179              if model is None:
180                  print("No Model loaded")
181                  continue
182              model, __, epoch = load_model_checkpoint(cp_f, model, optimizer=None, device_id=f'cuda:{args.gpu_device}')
183              if hasattr(model, 'freeze_cpc'):
184                  model.freeze_cpc = False
185                  model._unfreeze_cpc()
186              thresholds = calculate_best_thresholds([test_folder])[0]
187              print(thresholds)
188              print(
189                  f'Found architecturefile {os.path.basename(fm_f)}, checkpointfile {os.path.basename(cp_f)} in folder {root}. Appending model for testing.')
190              explain_model = ExplainLabelLayer(model, class_weights=class_weights,
191                                              layer='cpc_model.encoder.convolutionals.8', #'conv.12' , 'cpc_model.
192                                              encoder.convolutionals.12''tcn.network.2.net.4'
193                                              guided=args.guided) #'msresnet.layer7x7_3.0.conv2'' convs.16''msresnet.
194                                              maxpool? '#cpc_model.encoder.cpc_encoder.convolutionals.8'
195
196              model_dicts.append({'model': explain_model, 'model_folder': root, 'thresholds': thresholds,
197                                 'name': '-'.join(train_folder.split(os.path.sep)[-1].split('|')[0].split('.')[-2:])})
198      if len(model_dicts) == 0:
199          print(f'Could not find any models in {model_train_folders}.')
200      loaders = [

```

```

165     DataLoader(test_dataset_challenge, batch_size=args.batch_size, drop_last=False, num_workers=1,
166                 collate_fn=ecg_datasets2.collate_fn),
167     # DataLoader(val_dataset_challenge, batch_size=args.batch_size, drop_last=False, num_workers=1,
168     #             collate_fn=ecg_datasets2.collate_fn),
169     # DataLoader(train_dataset_challenge, batch_size=args.batch_size, drop_last=False, num_workers=1,
170     #             collate_fn=ecg_datasets2.collate_fn), #Train usually not required
171
172 ]
173 SHOW = True
174 SAVE = False
175 for model_i, model_dict in enumerate(model_dicts):
176     N_SAMPLES = 20
177     model = model_dict['model']
178     model_name = model_dict['name']
179     thresholds = torch.Tensor(list(model_dict['thresholds'].values()))
180     model.cuda()
181     model.train()
182     output_dir = os.path.join(args.out_path, model_name)
183     Path(output_dir).mkdir(parents=True, exist_ok=True)
184     with open(os.path.join(output_dir, 'thresholds.txt'), 'w+') as f:
185         f.write(','.join(map(str, thresholds)))
186     # first = True
187     # init optimizer
188     optimizer = Adam(model.parameters(), lr=3e-4)
189     for epoch in range(1, 2):
190         for loader_i, loader in enumerate(loaders):
191             for dataset_tuple in loader:
192                 if N_SAMPLES <= 0:
193                     return
194                 N_SAMPLES -= 1
195                 data, labels, filenames = dataset_tuple
196                 data = data.float().cuda()
197                 labels = labels.float().cuda()
198                 # for
199                 # NORMAL GRADIENT
200                 optimizer.zero_grad()
201                 pred = model(data, y=labels) # obtain pred once
202                 pred = pred.detach()
203                 optimizer.zero_grad()
204                 # _, grad1 = model(data, y=labels) # explain_class=
205                 # timeseries_to_image_with_gradient(data.detach().cpu(), labels.detach().cpu(), grad1.detach().cpu(),
206                 # pred.detach().cpu(), model_thresholds=thresholds, title='Gradient for actual label\n', filenames=filenames,
207                 # save=True, show=True)
208                 filenames = list(
209                     map(lambda x: os.path.join(output_dir, x), map(os.path.basename, filenames)))
210
211                 grads = []
212                 class_names = []
213                 for class_i in torch.nonzero(labels, as_tuple=False): # only do for first image in batch
214                     optimizer.zero_grad()
215                     l = torch.zeros_like(labels, device=labels.device) # pred.clone()
216                     l[tuple(class_i)] = 1.
217                     _ = model(data, y=l)
218                     if args.guided:
219                         gradi = model.get_gradcam_and_guided(target_size=None)
220                         print("minsmax", gradi.min(), gradi.max(), gradi.mean())
221                     else:
222                         gradi = model.get_gradcam(target_size=None, scale=False)
223                         print("minsmax", gradi.min(), gradi.max(), gradi.mean())
224                         print('data shape', data.shape, 'grad shape', gradi.shape)
225                         class_names.append(class_i[1].item())
226                         grads.append(gradi)
227                         timeseries_to_image_with_gradient_cam(data.detach().cpu(), labels.detach().cpu(), grads,
228                         pred.detach().cpu(), model_thresholds=thresholds,
229                         cut_off=0.0 if args.guided else 0.4, class_name_list=
230                         class_names,
231                         filenames=filenames, save=True, show=False)
232
233                 if args.dry_run:
234                     break
235                 del data, pred, labels
236                 print("\tfinished dataset {}.\tProgress: {} / {}".format(loader_i, loader_i + 1, len(loaders)))
237                 torch.cuda.empty_cache()
238                 if args.dry_run:
239                     break
240                 del model # delete and free
241                 torch.cuda.empty_cache()
242
243 def save_dict_to_csv_file(filepath, data, column_names=None):
244     with open(filepath) as f:
245         if not column_names is None:
246             f.write(','.join(column_names) + '\n')
247             for dc in data:
248                 f.write(','.join(dc) + '\n')
249
250 def parse_tensor_to_numpy_or_scalar(input_tensor):
251     if type(input_tensor) == torch.Tensor:
252         arr = input_tensor.detach().cpu().numpy() if input_tensor.is_cuda else input_tensor.numpy()

```

```

252     if arr.size == 1:
253         return arr.item()
254     return arr
255 
256 
257 
258 if __name__ == "__main__":
259     import sys
260 
261     print(sys.argv)
262 
263     parser = argparse.ArgumentParser(description='Contrastive Predictive Coding')
264     # datapath
265     # Other params
266     parser.add_argument('--saved_model', type=str,
267                         help='Model path to load weights from. Has to be given for downstream mode.')
268 
269     parser.add_argument('--seed', type=int, help='The seed used', default=0)
270 
271     parser.add_argument('--forward_mode', help="The forward mode to be used.", default='context',
272                         type=str) # , choices=['context', latents, all']
273 
274     parser.add_argument('--out_path', help="The output directory for losses and models",
275                         default='models/' + str(datetime.datetime.now().strftime("%d_%m_%y-%H")) + '-gradcam-explain',
276                         type=str)
277 
278     parser.add_argument('--forward_classes', type=int, default=52,
279                         help="The number of possible output classes (only relevant for downstream)")
280 
281     parser.add_argument('--warmup_steps', type=int, default=0, help="The number of warmup steps")
282 
283     parser.add_argument('--batch_size', type=int, default=1, help="The batch size")
284 
285     parser.add_argument('--latent_size', type=int, default=128,
286                         help="The size of the latent encoding for one window")
287 
288     parser.add_argument('--timesteps_in', type=int, default=6,
289                         help="The number of windows being used to form a context for prediction")
290 
291     parser.add_argument('--timesteps_out', type=int, default=6,
292                         help="The number of windows being predicted from the context (cpc task exclusive)")
293 
294     parser.add_argument('--crop_size', type=int, default=4500,
295                         help="The size of the data that it is cropped to. If data is smaller than this number, data
296                         gets padded with zeros")
297 
298     parser.add_argument('--channels', type=int, default=12,
299                         help="The number of channels the data will have") # TODO: auto detect
300 
301     parser.add_argument('--window_size', type=int, default=512,
302                         help="The number of datapoints per channel per window")
303 
304     parser.add_argument('--hidden_size', type=int, default=512,
305                         help="The size of the cell state/context used for predicting future latents or solving
306                         downstream tasks")
307 
308     parser.add_argument('--dry_run', dest='dry_run', action='store_true',
309                         help="Only run minimal samples to test all models functionality")
310     parser.set_defaults(dry_run=False)
311 
312     parser.add_argument('--use_class_weights', dest='use_class_weights', action='store_true',
313                         help="Use class weights determined by datasets class count")
314     parser.set_defaults(use_class_weights=False)
315 
316     parser.add_argument('--guided', dest='guided', action='store_true',
317                         help="Use class weights determined by datasets class count")
318     parser.set_defaults(guided=False)
319 
320     args = parser.parse_args()
321     main(args)

```

B.15 main_produce_plots_for_tested_models.py (code)

```

1 import glob
2 import json
3 import os
4 import pathlib
5 import re
6 from datetime import datetime
7 
8 import pandas as pd
9 import numpy as np
10 
11 from util.data.dataframe_factory import DataFrameFactory
12 from util.metrics import metrics as m

```

```

13 import util.visualize.plot_metrics as plotm
14
15 from util.utility.dict_utils import count_key_in_dict, extract_values_for_key_in_dict
16
17
18 def auto_find_tested_models_recursive(path='models/'):
19     # only works with specific file structure date/modelfolder/files
20     print(f"Looking for models at {os.path.abspath(path)}")
21     files = []
22     for root, dirs, dir_files in os.walk(path):
23         fm_temp, ch_temp = [], []
24         for file in dir_files:
25             # print('Checking', file, 'labels' in file and file.endswith('.csv'), 'output' in file and file.endswith('.csv'))
26             if 'labels' in file and file.endswith('.csv'):
27                 fm_temp.append(os.path.join(root, file))
28             if 'output' in file and file.endswith('.csv'):
29                 ch_temp.append(os.path.join(root, file))
30         if len(fm_temp) > 0 and len(ch_temp) > 0:
31             files.append(os.path.split(fm_temp[0])[0])
32         else:
33             print(f"not a model folder: {root}")
34     print(f"Found {len(files)} model test files")
35     return files
36
37
38 def auto_find_tested_models(path='models/'):
39     csvs = glob.glob(os.path.join(path, '*/*/*.csv')) # Finds all csv files with above structure
40     csv_paths = list(reversed(list(set([os.path.abspath(os.path.split(csv)[0]) for csv in csvs]))))
41     return csv_paths
42
43
44 def long_to_shortname(model_name):
45     print('modelname in ', model_name)
46     model_name = re.sub('cpc_combined.CPCCombined', 'CPC:', model_name)
47     model_name = re.sub('architectures_baseline_challenge.', '', model_name)
48     model_name = re.sub('baseline_cnn', 'BL', model_name)
49     model_name = re.sub('baseline', 'BL', model_name)
50     model_name = re.sub('.BaselineNet', ':', model_name)
51     model_name = re.sub('cpc_downstream_only', 'linear', model_name)
52     model_name = re.sub('cpc_downstream', '', model_name)
53     # model_name = re.sub('use_weights', '', model_name)
54     # model_name = re.sub('pte:\d*', '', model_name)
55     return model_name
56
57 def extract_model_attributes(model_folder, model_name, skip_cpc=True, skip_baseline=True, short_key=False):
58     print(model_folder)
59     is_cpc = True
60     attrs = {}
61     try:
62         #model_arch.txt
63         with open(os.path.join(model_folder, 'model_arch.txt'), 'r') as file:
64             content = file.read()
65             if 'BaselineNet' in content:
66                 is_cpc = False
67             attrs["Model Path"] = model_folder
68             if (skip_cpc and is_cpc) or (skip_baseline and not is_cpc):
69                 return None
70
71         fname = pathlib.Path(os.path.join(model_folder, 'params.txt'))
72         timestamp = fname.stat().st_mtime
73         attrs['train_timestamp'] = datetime.fromtimestamp(timestamp)
74         with open(os.path.join(model_folder, 'train_params.txt'), 'r') as file:
75             content = file.read()
76             attrs['Uses Classweights'] = not 'use_class_weights=False' in content
77             if 'norm_fn=' in content:
78                 attrs["Train Normalization Function"] = content.split('norm_fn=')[1].split(',')[-1]
79             else:
80                 if int(timestamp) > 1637682016:
81                     attrs["Train Normalization Function"] = 'normalize_std_scaling'
82                 else:
83                     attrs["Train Normalization Function"] = 'normalize_minmax_scaling_different'
84             if 'splits_file=' in content:
85                 attrs['Train splitsfile'] = content.split("splits_file=")[1].split("")[-1].replace('.txt', '')
86             else:
87                 attrs['Train splitsfile'] = 'train-test-splits'
88             if 'crop_size=' in content:
89                 attrs["Crop Size"] = content.split('crop_size=')[1].split(',')[-1]
90
91         #params.txt
92         with open(os.path.join(model_folder, 'params.txt'), 'r') as file:
93             content = file.read()
94             if 'norm_fn=' in content:
95                 attrs["Test Normalization Function"] = content.split('norm_fn=')[1].split(',')[-1]
96             else:
97                 if int(timestamp) > 1638140400: #oh god
98                     attrs["Test Normalization Function"] = 'normalize_std_scaling'
99                 else:
100                    attrs["Test Normalization Function"] = 'normalize_minmax_scaling_different'
```

```

1 # if 'splits_file=' in content:
2     attrs['Test splitsfile'] = content.split("splits_file'") [1].split("') [0].replace('.txt', '')
3 # else:
4     attrs['Test splitsfile'] = 'train-test-splits'
5
6 if is_cpc:
7     with open(os.path.join(model_folder, 'model_variables.txt'), 'r') as file:
8         content = file.readlines()
9     for i, line in enumerate(content):
10        if '(' in line:
11            content = '\n'.join(content[i:])
12            break
13    data = json.loads(content)
14    attrs['strided'] = 'cpc_encoder_as_strided.StridedEncoder' in content
15    attrs['Freeze CPC'] = not "'freeze_cpc': false' in content
16    attrs['uses Context'] = "'use_context": true' in content
17    attrs['uses Latents'] = "'use_latents": true' in content
18    attrs['normalizes latents'] = "'normalize_latents": true' in content
19    if "sampling_mode" in content:
20        attrs['CPC Sampling Mode'] = content.split('"sampling_mode": ')[1].split(',') [0][1:-1]
21    else:
22        attrs['CPC Sampling Mode'] = 'same'
23    try:
24        cpc_type = list(data["architectures_cpc.cpc_combined.CPCCombined"]["_modules"] [""] ["cpc_model"].keys())
25    [0]
26        attrs['CPC Type'] = cpc_type.split('.')[ -2]
27        attrs['Autoregressive'] = list(data["architectures_cpc.cpc_combined.CPCCombined"]["_modules"] [""] ["cpc_model"] [cpc_type] ["_modules"] [""] ["autoregressive"])[0].split('.')[ -2]
28        if not attrs['strided']:
29            attrs['Encoder'] = list(data["architectures_cpc.cpc_combined.CPCCombined"]["_modules"] [""] ["cpc_model"] [cpc_type] ["_modules"] [""] ["encoder"])[0].split('.')[ -2]
30        else:
31            attrs['Encoder'] = list(data["architectures_cpc.cpc_combined.CPCCombined"]["_modules"] [""] ["cpc_model"] [cpc_type] ["_modules"] [""] ["encoder"])
32                ["architectures_cpc.cpc_encoder_as_strided.StridedEncoder"] ["_modules"] ["cpc_encoder"] [0].split('.')[ -2]
33        attrs['Predictor'] = list(data["architectures_cpc.cpc_combined.CPCCombined"]["_modules"] [""] ["cpc_model"] [cpc_type] ["_modules"] [""] ["predictor"])[0].split('.')[ -2]
34    except KeyError:
35        print("Model does not follow CPC Combined architecture spec.")
36    if '"downstream_model:' in content:
37        attrs['Downstream Model'] = \
38            content.split('"downstream_model": ')[1].split(": ")[0].strip('"').split('.')[ -2]
39
40    # params.txt
41    with open(os.path.join(model_folder, 'train_params.txt'), 'r') as file:
42        content = file.read()
43    if 'pretrain_epochs' in content and is_cpc:
44        pos = content.split('pretrain_epochs=')[1].split(',') [0]
45        attrs['Pretrain Epochs'] = pos
46    if 'downstream_epochs' in content:
47        pos = content.split('downstream_epochs=')[1].split(',') [0]
48        attrs['Downstream Epochs'] = pos
49    attrs['Latent Size'] = content.split('latent_size=')[1].split(',') [0]
50    attrs['Context Size'] = content.split('hidden_size=')[1].split(',') [0]
51    attrs['timesteps in'] = content.split('timesteps_in=')[1].split(',') [0]
52    attrs['timesteps out'] = content.split('timesteps_out=')[1].split(',') [0]
53
54
55 else: # not cpc
56     with open(os.path.join(model_folder, 'train_params.txt'), 'r') as file:
57         content = file.read()
58     if 'downstream_epochs' in content:
59         pos = content.split('downstream_epochs=')[1].split(',') [0]
60         attrs['Downstream Epochs'] = pos
61
62     with open(os.path.join(model_folder, 'model_variables.txt'), 'r') as file:
63         content = file.readlines()
64     for i, line in enumerate(content):
65        if '(' in line:
66            content = '\n'.join(content[i:])
67            break
68    data = json.loads(content)
69    attrs['Convolutional Layer Number'] = str(count_key_in_dict(data, 'torch.nn.modules.conv.Convid'))
70    attrs['uses Max Pool'] = 'torch.nn.modules.pooling.MaxPool1d' in content
71    attrs['uses Adaptive Average Pooling'] = 'torch.nn.modules.pooling.AdaptiveAvgPool1d' in content
72    attrs['uses Linear'] = 'torch.nn.modules.linear.Linear' in content
73    attrs['uses LSTM'] = 'torch.nn.modules.rnn.LSTM' in content
74    attrs['uses BatchNorm'] = 'torch.nn.modules.batchnorm.BatchNorm1d' in content
75    attrs['Sum of Strides'] = str(int(np.array(extract_values_for_key_in_dict(data, 'stride')).sum()))
76    attrs['Sum of Dilation'] = str(int(np.array(extract_values_for_key_in_dict(data, 'dilation')).sum()))
77    attrs['Sum of Paddings'] = str(int(np.array(extract_values_for_key_in_dict(data, 'padding')).sum()))
78    attrs['Sum of Filters'] = str(int(np.array(extract_values_for_key_in_dict(data, 'kernel_size')).sum()))
79    final_layers = {'baseline_FC1': '3',
80                    'baseline_cnn_v0_2': '4',
81                    'baseline_cnn_v2': '3',
82                    'baseline_cnn_v6': '4',
83                    'baseline_cnn_v1': '3',
84                    'baseline_cnn_v5': '4',
85

```

```

186         'baseline_cnn_v4': '4',
187         'baseline_cnn_v14': '4',
188         'baseline_cnn_v3': '4',
189         'baseline_cnn_v0_1': '4',
190         'baseline_cnn_v0': '4',
191         'baseline_cnn_v9': '4',
192         'baseline_cnn_v8': '4',
193         'baseline_cnn_v0_3': '4',
194         'baseline_TCN_down': '4',
195         'baseline_TCN_flatten': '1',
196         'baseline_cnn_v7': '4',
197         'baseline_TCN_block': '1',
198         'baseline_rnn_simplest_lstm': '5',
199         'baseline_rnn_simplest_gru': '5',
200         'baseline_MLP': '2',
201         'baseline_alex_v2': '1',
202         'baseline_cnn_v15': '4',
203         'baseline_TCN_last': '2'}
204     try:
205         attrs['Final Layer'] = final_layers[list(data.keys())[0].split('.')[0][-2]]
206     except KeyError:
207         final_layers_short = {'BL_FCN': '3',
208             'BL_v0_2': '4',
209             'BL_v2': '3',
210             'BL_v6': '4',
211             'BL_v1': '3',
212             'BL_v5': '4',
213             'BL_v4': '4',
214             'BL_v14': '4',
215             'BL_v3': '4',
216             'BL_v0_1': '4',
217             'BL_v0': '4',
218             'BL_v9': '4',
219             'BL_v8': '4',
220             'BL_v0_3': '4',
221             'BL_TCN_down': '4',
222             'BL_TCN_flatten': '1',
223             'BL_v7': '4',
224             'BL_TCN_block': '1',
225             'BL_rnn_simplest_lstm': '5',
226             'BL_MLP': '2',
227             'BL_alex_v2': '1',
228             'BL_v15': '4',
229             'BL_TCN_last': '2'}
230     try:
231         attrs["Final Layer"] = final_layers_short[model_name]
232     except KeyError:
233         attrs["Final Layer"] = None
234         print(model_name, 'not found in final layers dict')
235     except FileNotFoundError:
236         print(model_folder, 'is not a model folder?')
237     return attrs
238
239 # def create_metric_plots(model_folder, binary_labels, pred, classes):
240 #     print("creating for:", model_folder)
241 #     model_folder_name = os.path.split(model_folder)[1]
242 #     n_classes = len(classes)
243 #     tpr, fpr, roc_auc, thresholds = m.ROC(binary_labels, pred)
244 #     tps, fps, best_thresholds = m.select_best_thresholds(tpr, fpr, thresholds, n_classes)
245 #     zero_fit = m.zero_fit_score(binary_labels, pred, 'macro')
246 #     print('zero_fit, macro', zero_fit)
247 #     zero_fit = m.zero_fit_score(binary_labels, pred, 'micro')
248 #     print('zero_fit, micro', zero_fit)
249 #     class_fit = m.class_fit_score(binary_labels, pred, 'macro')
250 #     print('class_fit, macro', class_fit)
251 #     class_fit = m.class_fit_score(binary_labels, pred, 'micro')
252 #
253 #     print('class_fit, micro', class_fit)
254 #     binary_preds = m.convert_pred_to_binary(pred, best_thresholds)
255 #
256 #     df = create_metric_score_dataframe(binary_labels, binary_preds, classes, m.f1_scores)
257 #     print(df)
258 #     df = create_metric_score_dataframe(binary_labels, binary_preds, classes, m.recall_scores)
259 #     print(df)
260 #     df = create_metric_score_dataframe(binary_labels, binary_preds, classes, m.precision_scores)
261 #     print(df)
262 #     df = create_metric_score_dataframe(binary_labels, binary_preds, classes, m.accuracy_scores)
263 #     print(df)
264 #     df = create_metric_score_dataframe(binary_labels, binary_preds, classes, m.balanced_accuracy_scores)
265 #     print(df)
266 #     #print(df.to_latex(index=False, label='', caption='LUL'))
267 #
268 #     normal_class = '426783006'
269 #     normal_class_idx = np.where(classes == normal_class)[0][0]
270 #     plotm.plot_roc_multiclass(tpr, fpr, roc_auc, classes, savepath=model_folder, plot_name=model_folder_name)
271 #     plotm.plot_roc_singleclass(tpr, fpr, roc_auc, class_name=normal_class, class_i=normal_class_idx, savepath=
272 #     model_folder, plot_name=model_folder_name)
273 #     #plotm.plot_precision_recall_multiclass(precision, recall, avg_precision, classes, savepath=model_folder,
274 #     plot_name=model_folder_name)
```

```

274 def create_metric_score_dataframe(binary_labels, binary_preds, classes, metric_function, model_name=None,
275                                     average_only=False, model_attrs=None):
276     scdf = pd.DataFrame({'model': [model_name]})
277     if not model_attrs is None:
278         scdf = pd.concat([scdf, pd.DataFrame([(k:v if type(v)==list else [v] for k,v in model_attrs.items())])], axis=1)
279     if not average_only:
280         scores = metric_function(binary_labels, binary_preds, average=None)
281         scdf = pd.concat([scdf, pd.DataFrame(scores[np.newaxis, :], columns=classes)], axis=1, )
282     avg5 = pd.DataFrame(data={
283         'micro': np.atleast_1d(metric_function(binary_labels, binary_preds, average='micro')),
284         'macro': np.atleast_1d(metric_function(binary_labels, binary_preds, average='macro'))
285     })
286     scdf = pd.concat([scdf, avg5], axis=1)
287     # scdf.insert(0, 'micro', metric_function(binary_labels, binary_preds, average='micro'), allow_duplicates=True)
288     # scdf.insert(0, 'macro', metric_function(binary_labels, binary_preds, average='macro'), allow_duplicates=True)
289     #scdf = scdf.set_index('model')
290     return scdf
291
292
293 def create_metric_confusion_matrix(model_folder, binary_labels, pred, classes: list):
294     print("creating for:", model_folder)
295     model_folder_name = os.path.split(model_folder)[1]
296     n_classes = len(classes)
297     tpr, fpr, roc_auc, thresholds = m.ROC(binary_labels, pred)
298     tps, fps, best_thresholds = m.select_best_thresholds(tpr, fpr, thresholds, n_classes)
299     # test_thresholds = {c:0.5 for c in classes}
300     binary_preds = m.convert_pred_to_binary(pred, best_thresholds)
301     print(binary_preds)
302     print(binary_labels)
303     cm = m.confusion_matrix(binary_labels, binary_preds)
304     print(cm)
305     plotm.plot_confusion_matrix(cm, classes)
306
307
308 # def create_latex_table(dataframe:pd.DataFrame, output_folder, latex_label='', caption='', filename='table.tex'):
309 #     latex_string = dataframe.to_latex(index=False, label=latex_label, caption=caption)
310 #     with open(os.path.join(output_folder, filename), 'w') as f:
311 #         f.write(latex_string)
312
313
314 def calculate_best_thresholds(model_folders, data_loader_index=1): # 0 = test, 1 = val, 2 = train
315     model_thresholds = []
316     for mi, model_folder in enumerate(model_folders):
317         print(model_folder)
318         best_thresholds = None
319         try:
320             binary_labels, classes = m.read_binary_label_csv_from_model_folder(model_folder,
321                                         data_loader_index=data_loader_index)
322             predictions, pred_classes = m.read_output_csv_from_model_folder(model_folder,
323                                         data_loader_index=data_loader_index)
324             if np.any(np.isnan(predictions)):
325                 print(f"Encountered nan value in {model_folder} prediction!")
326                 model_thresholds.append(None)
327                 continue
328             tprs, fprs, roc_auc, thresholds = m.ROC(binary_labels, predictions)
329             tpr, fpr, best_thresholds = m.select_best_thresholds(tprs, fprs, thresholds, len(classes))
330         except FileNotFoundError as e: # folder with not the correct csv?
331             print(e)
332         model_thresholds.append(best_thresholds)
333     return model_thresholds
334
335
336 def create_paper_plots(model_folders, data_loader_index=0):
337     TEST_SET = 0;
338     VAL_SET = 1;
339     TRAIN_SET = 2
340
341     model_thresholds = calculate_best_thresholds(model_folders, data_loader_index=VAL_SET)
342     for mi, model_folder in enumerate(model_folders):
343         if model_thresholds[mi] is None:
344             print(f"Encountered nan value in {model_folder} prediction!. Skipping this model.")
345             continue
346         try:
347             model_name = os.path.split(model_folder)[1]
348             model_name = '.'.join(model_name.split('.')[0:-2]) if '.' in model_name else model_name # fullname(store_models.
349             load_model_architecture(extract_model_files_from_dir(model_folder)[0][0]))
350             print(model_name)
351             binary_labels, classes = m.read_binary_label_csv_from_model_folder(model_folder,
352                                         data_loader_index=data_loader_index)
353             predictions, pred_classes = m.read_output_csv_from_model_folder(model_folder,
354                                         data_loader_index=data_loader_index)
355             binary_predictions = m.convert_pred_to_binary(predictions, model_thresholds[mi])
356
357             n_classes = len(classes)
358             tpr, fpr, roc_auc, thresholds = m.ROC(binary_labels, predictions)
359             tps, fps, best_thresholds = m.select_best_thresholds(tpr, fpr, thresholds, n_classes)
360             normal_class = '426783006'
361             normal_class_idx = np.where(classes == normal_class)[0][0]
362             plotm.plot_roc_multiclass(tpr, fpr, roc_auc, classes, savepath=model_folder, plot_name=model_name,

```

```

363         plot_legends=False)
364     plotm.plot_roc_singleclass(tpr, fpr, roc_auc, class_name=normal_class, class_i=normal_class_idx,
365                               savepath=model_folder, plot_name=model_name)
366 except FileNotFoundError:
367     print("File not found")
368
369
370 def create_model_attribute_table(model_folders, filename, save_to='/home/julian/Documents/projekt-master/tables',
371                                 skip_cpc=True, skip_baseline=False):
372     attribute_df = DataFrameFactory()
373
374     for f in model_folders:
375         for root, dirs, files in os.walk(f):
376             if len(dirs) == 0 and len(files) == 0:
377                 continue
378             if len(dirs) > 0: # Traverse more
379                 continue
380             if len(dirs) == 0 and len(files) > 0: # leaf dir (model?)
381                 name = long_to_shortname(root.split(os.sep)[-1].split('|')[0])
382                 attrs = extract_model_attributes(root, name, skip_cpc, skip_baseline)
383                 if attrs == None: #got skipped
384                     continue
385                 attrs['Model Name'] = name
386                 attrs = pd.DataFrame(attrs, index=[0])
387                 attrs = attrs.set_index('Model Name')
388                 attribute_df.append(attrs)
389     attribute_df.dataframe.drop_duplicates(inplace=True)
390     attribute_df.natsort_single_index()
391     if not filename is None:
392         attribute_df.dataframe.to_csv('/home/julian/Desktop/' + filename + '.csv')
393     return attribute_df.dataframe
394
395 def create_lowlabel_plots(model_folders, filename, title_add='', save_to='/home/julian/Documents/projekt-master/bilder'
396                           ,
397                           data_loader_index=0):
398     TEST_SET = 0;
399     VAL_SET = 1;
400     TRAIN_SET = 2
401
402     model_thresholds = calculate_best_thresholds(model_folders, data_loader_index=VAL_SET)
403
404     f1_dff = DataFrameFactory()
405     prec_dff = DataFrameFactory()
406     rec_dff = DataFrameFactory()
407     classfit_dff = DataFrameFactory()
408     zerofit_dff = DataFrameFactory()
409     auc_dff = DataFrameFactory()
410
411     average_only = True
412
413     for mi, model_folder in enumerate(model_folders):
414         try:
415             if model_thresholds[mi] is None:
416                 print("Encountered nan value in {model_folder} prediction!. Skipping this model.")
417                 continue
418             model_name = os.path.split(model_folder)[1]
419             model_name = '.'.join(model_name.split('.')[1:-2:]) if '.' in model_name else model_name # fullname(store_models.
420             load_model_architecture(extract_model_files_from_dir(model_folder)[0][0])
421             model_name = long_to_shortname(model_name)
422             print(model_name)
423             binary_labels, classes = m.read_binary_label_csv_from_model_folder(model_folder,
424                                   data_loader_index=data_loader_index)
425             predictions, pred_classes = m.read_output_csv_from_model_folder(model_folder,
426                                   data_loader_index=data_loader_index)
427             binary_predictions = m.convert_pred_to_binary(predictions, model_thresholds[mi])
428             auc_dff.append(create_metric_score_dataframe(binary_labels, predictions, classes, m.auc_scores, model_name
429
430                                         , average_only))
431
432         except FileNotFoundError as e: # folder with not the correct csv?
433             print(e)
434     auc_dff.natsort_single_index()
435
436     def find_splits_in_string(s):
437         if 'train-test-splits-' in s:
438             sfile = s.split('train-test-splits_')[1].split('|')[0]
439             return (sfile, s.replace('train-test-splits-' + sfile, '').replace('|||', '|').replace('|||', '|'))
440         elif 'train-test-splits-' in s:
441             sfile = s.split('train-test-splits-')[1].split('|')[0]
442             return (sfile, s.replace('train-test-splits-' + sfile, '').replace('|||', '|').replace('|||', '|'))
443         else:
444             return ("standard", s)
445
446     auc_dff.dataframe.index = pd.MultiIndex.from_tuples(
447         [find_splits_in_string(i) for i, _ in auc_dff.dataframe.iterrows()])
448     groups = auc_dff.dataframe.groupby(level=[1], as_index=False)
449     plotm.plot_lowlabel_availability(groups, 'Micro average AUC score with low label availability' + title_add, save_to
450
451                                         , filename + 'micro.png', data_col='micro')

```

```

449     plotm.plot_lowlabel_availability(groups, 'Macro average AUC score with low label availability' + title_add, save_to
450     ,
451         filename + 'macro.png', data_col='macro')
452
453 def create_parallel_plots(model_folders, savepath, data_loader_index=0, skip_cpc=False, skip_baseline=False):
454     TEST_SET = 0;
455     VAL_SET = 1;
456     TRAIN_SET = 2
457
458     model_thresholds = calculate_best_thresholds(model_folders, data_loader_index=VAL_SET)
459
460     auc_dff = DataFrameFactory()
461
462     average_only = True
463
464     for mi, model_folder in enumerate(model_folders):
465         try:
466             if model_thresholds[mi] is None:
467                 print(f"Encountered nan value in {model_folder} prediction!. Skipping this model.")
468                 continue
469             model_name = os.path.split(model_folder)[1]
470             model_name = '.'.join(model_name.split('.')[0:-2]) if '.' in model_name else model_name # fullname(store_models.
471             load_model_architecture(extract_model_files_from_dir(model_folder)[0][0]))
472             model_name = long_to_shortname(model_name)
473             print(model_name)
474             binary_labels, classes = m.read_binary_label_csv_from_model_folder(model_folder,
475                 data_loader_index=data_loader_index)
476             predictions, pred_classes = m.read_output_csv_from_model_folder(model_folder,
477                 data_loader_index=data_loader_index)
478             binary_predictions = m.convert_pred_to_binary(predictions, model_thresholds[mi])
479             auc_df = create_metric_score_dataframe(binary_labels, predictions, classes, m.auc_scores, model_name,
480                 average_only)
481             # TODO: get train path for call below for most accurate results: not feasible because of random structure
482             # TODO: add final layer number desc
483             attribute_df = create_model_attribute_table([model_folder], filename=None, skip_cpc=skip_cpc,
484                 skip_baseline=skip_baseline)
485             if len(attribute_df) == 0:
486                 continue
487             attribute_df.index = auc_df.index
488             # seperate_folders = model_folder.split(os.sep)
489             # train_folder_ix = seperate_folders.index('models')+2
490             # train_folder = os.sep.join(seperate_folders[train_folder_ix:])
491             auc_dff.append(pd.concat([attribute_df, auc_df], axis=1))
492
493         except FileNotFoundError as e: # folder with not the correct csv?
494             print(e)
495         auc_dff.natsort_single_index()
496         auc_dff.dataframe.to_csv('ALLMODELSATTRIBUTES' + str(skip_cpc) + '.csv')
497         plotm.plot_parallel_coordinates(auc_dff.dataframe, 'micro', savepath + 'micro.png',
498             drop_columns=['use_class_weights'], put_last_columns=['micro'])
499         plotm.plot_parallel_coordinates(auc_dff.dataframe, 'macro', savepath + 'macro.png',
500             drop_columns=['use_class_weights'], put_last_columns=['macro'])
501         # auc_dff.dataframe.to_csv('/home/julian/Desktop/attributeswithauc.csv')
502
503
504 def create_paper_metrics(model_folders, root_path, data_loader_index=0, average_only=False, long_tables=False,
505     cut_nameAttrs=False, useAttrsInName=False, includeAttrsInTable=False, saveCsv=False,
506     saveToAllDirs=True):
507     TEST_SET = 0;
508     VAL_SET = 1;
509     TRAIN_SET = 2
510
511     model_thresholds = calculate_best_thresholds(model_folders, data_loader_index=VAL_SET)
512
513     f1_dff = DataFrameFactory()
514     prec_dff = DataFrameFactory()
515     rec_dff = DataFrameFactory()
516     classfit_dff = DataFrameFactory()
517     zerofit_dff = DataFrameFactory()
518     auc_dff = DataFrameFactory()
519
520     for mi, model_folder in enumerate(model_folders):
521         try:
522             if model_thresholds[mi] is None:
523                 print(f"Encountered nan value in {model_folder} prediction!. Skipping this model.")
524                 continue
525             model_name = os.path.split(model_folder)[1]
526             model_name = '.'.join(model_name.split('.')[0:-2]) if '.' in model_name else model_name # fullname(store_models.
527             load_model_architecture(extract_model_files_from_dir(model_folder)[0][0])
528             if cut_nameAttrs:
529                 model_name = model_name.split('/')[-1]
530             model_name = long_to_shortname(model_name)
531             print('Final name:', model_name)
532             attrs = None
533             if useAttrsInName or includeAttrsInTable:
534                 attrs = extract_model_attributes(model_folder, model_name, False, False)

```

```

535     nameAttrs = model_name
536     if useAttrsInName:
537         for k, v in attrs.items():
538             if (type(v) == str) and (not v in nameAttrs):
539                 model_name += '/' + v
540     if not includeAttrsInTable:
541         attrs = None
542     with open(os.path.join(model_folder, 'thresholds.txt'), 'w+') as f:
543         f.write(','.join(map(str, model_thresholds[mi].values())))
544     binary_labels, classes = m.read_binary_label_csv_from_model_folder(model_folder,
545                                         data_loader_index=data_loader_index)
546     predictions, pred_classes = m.read_output_csv_from_model_folder(model_folder,
547                                         data_loader_index=data_loader_index)
548     binary_predictions = m.convert_pred_to_binary(predictions, model_thresholds[mi])
549     print(model_thresholds[mi])
550     # scores with binary
551     f1_dff.append(
552         create_metric_score_dataframe(binary_labels, binary_predictions, classes, m.f1_scores, model_name,
553                                         average_only, modelAttrs=attrs))
554
555     prec_dff.append(
556         create_metric_score_dataframe(binary_labels, binary_predictions, classes, m.precision_scores,
557                                         model_name, average_only, modelAttrs=attrs))
558     rec_dff.append(
559         create_metric_score_dataframe(binary_labels, binary_predictions, classes, m.recall_scores, model_name,
560                                         average_only, modelAttrs=attrs))
561     # scores with probability
562     classfit_dff.append(
563         create_metric_score_dataframe(binary_labels, predictions, classes, m.class_fit_score, model_name,
564                                         average_only, modelAttrs=attrs))
565     zeroFit_dff.append(
566         create_metric_score_dataframe(binary_labels, predictions, classes, m.zero_fit_score, model_name,
567                                         average_only, modelAttrs=attrs))
568     auc_dff.append(create_metric_score_dataframe(binary_labels, predictions, classes, m.auc_scores, model_name
569                                         average_only, modelAttrs=attrs))
570
571 except FileNotFoundError as e: # folder with not the correct csv?
572     print(e)
573 allFactories = [auc_dff, prec_dff, rec_dff, zeroFit_dff, f1_dff, classfit_dff]
574 list(map(lambda obj: obj.natsort_by_column(column='model'), allFactories)) # SINCE WHEN ARE THESE LAZY?!
575 list(map(lambda obj: obj.put_columns_last(columns=['micro', 'macro']), allFactories))
576 if saveCsv:
577     auc_dff.to_csv(root_path, 'All_attributes_with_scores2.csv')
578 try:
579     list(map(lambda obj: obj.dataframe.drop(columns=['Model Path', 'train timestamp'], inplace=True),
580             allFactories))
581 except:
582     print(['Model Path', 'train timestamp'], 'not found in df')
583 if len(modelFolders) > 0:
584     if saveToAllDirs:
585         ps = list(set([os.path.split(mf)[0] for mf in modelFolders])) # GET all basepaths
586         print(ps)
587     else:
588         ps = [rootPath]
589     """
590     If if models from multiple test sessions are run, put a csv into every of their basepath folders.
591     """
592     label = 'scores'
593     label += '-avg' if averageOnly else ''
594     label += '-long' if longTables else ''
595     for p in ps:
596         print(f'Saving metrics to: {p};')
597         # f1_dff.to_csv(p, f'f1-score-dataloader{data_loader_index}.csv')
598         # prec_dff.to_csv(p, f'precision-score-dataloader{data_loader_index}.csv')
599         # rec_dff.to_csv(p, f'recall-score-dataloader{data_loader_index}.csv')
600         f1_dff.to_latex(p, f'f1-{label}-dataloader{data_loader_index}.tex', caption='F1 Scores',
601                         label='tbl:f1' + label, longTables=longTables, onlyTabularEnvironment=True)
602         prec_dff.to_latex(p, f'precision-{label}-dataloader{data_loader_index}.tex', caption='Precision Scores',
603                         label='tbl:precision' + label, longTables=longTables, onlyTabularEnvironment=True)
604         rec_dff.to_latex(p, f'recall-{label}-dataloader{data_loader_index}.tex', caption='Precision Scores',
605                         label='tbl:recall' + label, longTables=longTables, onlyTabularEnvironment=True)
606         # cstAcc_dff.to_csv(p, f'Custom Accuracy{data_loader_index}.tex')
607         classfit_dff.to_latex(p, f'Custom Accuracy (Class Fit){label}-dataloader-{data_loader_index}.tex',
608                               caption='Class Fit Scores', label='tbl:classfit' + label, longTables=longTables,
609                               onlyTabularEnvironment=True)
610         zeroFit_dff.to_latex(p, f'Custom Accuracy (Zero Fit){label}-dataloader-{data_loader_index}.tex',
611                               caption='Zero Fit Scores', label='tbl:zeroFit' + label, longTables=longTables,
612                               onlyTabularEnvironment=True)
613         auc_dff.to_latex(p, f'AUC-{label}-dataloader{data_loader_index}.tex', caption='AUC score',
614                         label='tbl:auc' + label, longTables=longTables, onlyTabularEnvironment=True)
615     # make attribute parallel lines plot at this position
616
617 return model_thresholds
618
619 def create_latex_table_from_csv(csv_file, outPath=None, outName=None, autoStripCols=True, averageOnly=False,
620                                longTables=False, useAttrsInName=False, saveToAllDirs=True):
621     fileRoot, csvName = os.path.split(csv_file)
622     print(csv_file)
623     if outPath is None:

```

```

622     out_path = file_root
623     if out_name is None:
624         out_name = os.path.splitext(csv_name)[0]
625     dff = DataFrameFactory(pd.read_csv(csv_file))
626     if auto_strip_cols:
627         include = ['model', 'micro', 'macro',
628                    'Freeze CPC', 'uses Context', 'uses Latents', 'normalizes latents', 'CPC Sampling Mode', 'CPC Type',
629                    'Autoregressive', 'Encoder', 'Predictor',
630                    ]
631     dff.dataframe = dff.dataframe[[c for c in dff.dataframe.columns if c in include]]
632     print(dff.dataframe.columns)
633     #dff.natsort_by_column(column='model')
634     dff.put_columns_last(columns=['micro', 'macro'])
635     try:
636         dff.dataframe.drop(columns=['Model Path', 'train timestamp'], inplace=True)
637     except:
638         print(['Model Path', 'train timestamp', 'not found in df'])
639     dff.to_latex(out_path, out_name+'.tex', caption='AUC score',
640                  label='tbl:auc-' + out_name, long_tables=long_tables, only_tabular_environment=True)
641
642
643
644 def sort_naturally(data):
645     pass
646
647
648 if __name__ == '__main__':
649     TEST_SET = 0;
650     VAL_SET = 1;
651     TRAIN_SET = 2
652     paths = [
653         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/16_12_21-15-09-test|(10x)bl_TCN_down+(10x)bl_cnn_v1+(10x)bl_cnn_v14+(10x)bl_cnn_v15+(10x)bl_cnn_v8'
654         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/16_08_21-10-16-test|(40x)cpc/14_08_21-15-36-train|(4x)cpc/architectures_cpc.cpc_combined.CPCCombined2|train-test-splits-fewer-labels60|use_weights|unfrozen|C|m:all|cpc_downstream_cnn'
655         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/17_12_21-13-13-test|(160x)cpc',
656         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/16_12_21-15-09-test|(10x)bl_TCN_down+(10x)bl_cnn_v1+(10x)bl_cnn_v14+(10x)bl_cnn_v15+(10x)bl_cnn_v8',
657         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/20_12_21-13-40-test|(32x)cpc',
658         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/20_12_21-16-23-test|(4x)bl_cnn_v14+(4x)bl_cnn_v8',
659         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/20_01_22-13-43-test|(12x)cpc'
660         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/12_01_22-16-33-test|(6x)cpc'
661         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/18_01_22-15-28-test|(28x)cpc/14_01_22-13-26-train|(14x)cpc',
662         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/18_01_22-15-28-test|(28x)cpc/14_01_22-15-39-train|(14x)cpc',
663         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/',
664         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/26_01_22-16-39-test|(4x)cpc',
665         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/20_12_21-16-23-test|(4x)bl_cnn_v14+(4x)bl_cnn_v8'
666         #'',
667         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/27_01_22-13-07-test|(4x)cpc',
668         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/27_01_22-13-57-test|(8x)cpc',
669         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/26_01_22-16-39-test|(4x)cpc',
670         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/28_01_22-15-57-test|(12x)cpc'
671         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/28_01_22-17-06-test|(12x)cpc'
672         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/03_02_22-14-40-test|(2x)cpc'
673         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/11_02_22-15-10-test|(8x)cpc'
674         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/25_06_21-16-test|bl_TCN_down/24_06_21-16-train|(2x)bl_TCN_block+(2x)bl_TCN_down+(2x)bl_TCN_flatten+(2x)bl_TCN_last+(2x)bl_cnn_v15+(2x)bl_rnn_simplest+lstm+bl_MLP+bl_alex_v2+bl_cnn_v7',
675         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/26_06_21-15-test|(2x)bl_MLP+bl_FCNN+bl_TCN_block+bl_TCN_down+bl_TCN_flatten+bl_TCN_last+bl_alex_v2+bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v14+bl_cnn_v15+bl_cnn_v2+bl_cnn_v3+bl_cnn_v4+bl_cnn_v5+bl_cnn_v6+bl_cnn_v7+bl_cnn_v8+bl_cnn',
676         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/09_07_21-17-test|(34x)cpc',
677         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/15_02_22-15-09-test|(4x)cpc',
678         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/17_02_22-10-03-test|(3x)cpc'
679
680
681         #'home/julian/Downloads/Github/contrastive-predictive-coding/models/30_11_21-15-train|cpc', 'home/julian/Downloads/Github/contrastive-predictive-coding/models/02_12_21-18-train|cpc', 'home/julian/Downloads/Github/contrastive-predictive-coding/models/10_12_21-17-train|(2x)cpc', 'home/julian/Downloads/Github/contrastive-predictive-coding/models/30_11_21-20-train|(4x)cpc', 'home/julian/Downloads/Github/contrastive-predictive-coding/models/30_11_21-17-train|cpc', 'home/julian/Downloads/Github/contrastive-predictive-coding/models/06_12_21-17-train|(2x)cpc'
682     ]
683     root_path = '/home/julian/Desktop/'
684     csvs = list(glob('models_evaluated_filtered_csv/*.csv'))
685     for csv in csvs:
686         create_latex_table_from_csv(csv, auto_strip_cols=False)
687     # paths = ['home/julian/Downloads/Github/contrastive-predictive-coding/models/23_12_21-16-37-test|(4x)cpc', 'home/julian/Downloads/Github/contrastive-predictive-coding/models/27_12_21-14-01-test|(8x)cpc', 'home/julian/Downloads/Github/contrastive-predictive-coding/models/02_12_21-20-09-test|(2x)cpc', 'home/julian/Downloads/Github/contrastive-predictive-coding/models/05_01_22-17-30-test|(4x)cpc', 'home/julian/Downloads/Github/contrastive-predictive-coding/models/12_01_22-16-33-test|(6x)cpc', 'home/julian/Downloads/Github/contrastive-predictive-coding/models/30_11_21-16-28-test|cpc', 'home/julian/Downloads/Github/contrastive-predictive-coding/models/15_11_21-13-25-test|cpc', 'home/julian/Downloads/Github/contrastive-predictive-coding/models/06_12_21

```

```

-19-40-test|(2x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/23_12_21-14-17-test|(2x)cpc',
  '/home/julian/Downloads/Github/contrastive-predictive-coding/models/13_12_21-15-12-test|(4x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/06_01_22-15-01-test|(4x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/24_12_21-11-14-test|(8x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/01_12_21-18-14-test|(12x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/10_01_22-17-11-test|(2x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/30_11_21-18-25-test|cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/13_12_21-13-42-test|cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/01_12_21-19-56-test|(4x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/04_01_22-18-21-test|(2x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/12_11_21-13-23-test|(8x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/13_11_21-21-41-test|cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/16_12_21-13-17-test|(12x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/06_01_22-18-55-test|cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/06_01_22-18-04-test|(3x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/01_12_21-12-48-test|(12x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/12_11_21-16-02-test|(4x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/13_12_21-11-09-test|(2x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/13_12_21-13-13-test|(4x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/13_11_21-21-38-test|cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/20_12_21-13-40-test|(32x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/23_11_21-19-25-test|(16x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/02_12_21-17-56-test|(4x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/13_12_21-12-26-test|(4x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/13_12_21-14-27-test|cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/13_12_21-16-04-test|(3x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/17_12_21-13-13-test|(160x)cpc', '/home/julian/Downloads/Github/contrastive-predictive-coding/models/07_01_22-17-01-test|(4x)cpc'
for pl in paths:
    if type(pl) == list:
        model_folders = [a for p in pl for a in auto_find_tested_models_recursive(p)]
    else:
        model_folders = auto_find_tested_models_recursive(pl)
    # low_label_noclassweights_paths = ['/home/julian/Downloads/Github/contrastive-predictive-coding/models/20_07_21-17-50-test|(48x)cpc',
    694    # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/20_07_21-16-test|(5x)bl_TCN_down+(5x)bl_cnn_v1+(5x)bl_cnn_v14+(5x)bl_cnn_v15+(5x)bl_cnn_v8']
    695    # model_folders = [a for p in low_label_noclassweights_paths for a in auto_find_tested_models_recursive(p)]
    696    # create_lowlabel_plots(model_folders, data_loader_index=TEST_SET, filename='low_label_availability_noclassweights')
    697    # low_label_classweights_paths_more_epochs = ['models/11_08_21-15-58-test|(10x)bl_TCN_down+(10x)bl_cnn_v1+(10x)bl_cnn_v14+(10x)bl_cnn_v15+(10x)bl_cnn_v8',
    698    # '# models/13_08_21-10-47-test|(80x)cpc',
    699    # '# '/home/julian/Downloads/Github/contrastive-predictive-coding/models/16_08_21-10-16-test|(40x)cpc']
    700    # model_folders = [a for p in low_label_classweights_paths_more_epochs for a in
    auto_find_tested_models_recursive(p)]
    701    # create_lowlabel_plots(model_folders, data_loader_index=TEST_SET, filename='low_label_availability_classweights-more-epochs', title_add='(50 CPC epochs,)')
    702    # #
    703    # low_label_classweights_paths = ['models/11_08_21-15-58-test|(10x)bl_TCN_down+(10x)bl_cnn_v1+(10x)bl_cnn_v14+(10x)bl_cnn_v15+(10x)bl_cnn_v8',
    704    # '# models/13_08_21-10-47-test|(80x)cpc']
    705    # model_folders = [a for p in low_label_classweights_paths for a in auto_find_tested_models_recursive(p)]
    706    # create_lowlabel_plots(model_folders, data_loader_index=TEST_SET, filename='low_label_availability_classweights', title_add='(20 CPC epochs)', save_to=paths[0])
    707    #
    708    # low_label_classweights_paths = ['/home/julian/Downloads/Github/contrastive-predictive-coding/models/20_07_21-18-41-test|(48x)cpc',
    709    # '/home/julian/Downloads/Github/contrastive-predictive-coding/models/20_07_21-17-test|(5x)bl_TCN_down+(5x)bl_cnn_v1+(5x)bl_cnn_v14+(5x)bl_cnn_v15+(5x)bl_cnn_v8']
    710    # model_folders = [a for p in low_label_classweights_paths for a in auto_find_tested_models_recursive(p)]
    711    # create_lowlabel_plots(model_folders, data_loader_index=TEST_SET, filename='low_label_availability_classweights')
    712    #
    713    # model_folders = auto_find_tested_models_recursive('/home/julian/Downloads/Github/contrastive-predictive-coding/models/')
    714    # save csv:
    715    # create_paper_metrics(model_folders, root_path=root_path, data_loader_index=TEST_SET, average_only=True,
    cut_nameAttrs=True, useAttrsInName=False, includeAttrsInTable=True, saveCsv=True, saveToAllDirs=False)
    716    # save table in dir:
    717    # create_paper_metrics(model_folders, root_path=None, data_loader_index=TEST_SET, average_only=True,
    cut_nameAttrs=True, useAttrsInName=False, includeAttrsInTable=True, saveCsv=False, saveToAllDirs=True)
    718    # Old
    # create_paper_metrics(model_folders, root_path='', data_loader_index=TEST_SET, average_only=True,
    useAttrsInName=True, saveToAllDirs=True, includeAttrsInTable=True) # On Testset
    # create_paper_plots(model_folders, data_loader_index=TEST_SET)
    719    # create_paper_metrics(model_folders, root_path=path, data_loader_index=TEST_SET, average_only=True,
    saveToAllDirs=False) # On Testset
    720    # create_paper_metrics(model_folders, root_path=path, data_loader_index=TEST_SET, average_only=True, longTables=True,
    saveToAllDirs=False)
    721    # create_model_attribute_table(model_folders, 'baseline-attributes-full', skipCpc=True, skipBaseline=False)
    # create_model_attribute_table(model_folders, 'cpc-attributes', skipCpc=False, skipBaseline=True)
    # create_parallel_plots(model_folders, '/home/julian/Desktop/cpc-attributes-parallelcoords', skipCpc=False,
    skipBaseline=True)
    # create_parallel_plots(model_folders, '/home/julian/Desktop/bl-attributes-parallelcoords', skipCpc=True,
    skipBaseline=False)

```

References

- [1] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. "Representation Learning with Contrastive Predictive Coding". In: *CoRR* abs/1807.03748 (2018). arXiv: [1807 . 03748](https://arxiv.org/abs/1807.03748).
- [2] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. 2018. arXiv: [1803 . 01271 \[cs.LG\]](https://arxiv.org/abs/1803.01271).
- [3] Fisher Yu and Vladlen Koltun. *Multi-Scale Context Aggregation by Dilated Convolutions*. 2016. arXiv: [1511.07122 \[cs.CV\]](https://arxiv.org/abs/1511.07122).
- [4] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. "Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization". In: *CoRR* abs/1610.02391 (2016). arXiv: [1610 . 02391](https://arxiv.org/abs/1610.02391).
- [5] *Electrocardiography - Wikipedia*. Jan. 14, 2021. URL: [https : / / en . wikipedia . org / wiki / Electrocardiography](https://en.wikipedia.org/wiki/Electrocardiography) (visited on 01/14/2021).
- [6] Cables and Sensors EU. *12-Lead ECG Placement Guide with Illustrations*. Sept. 28, 2021. URL: [https : / / www . cablesandsensors . eu / pages / 12 - lead - ecg - placement - guide - with - illustrations](https://www.cablesandsensors.eu/pages/12-lead-ecg-placement-guide-with-illustrations) (visited on 09/28/2021).
- [7] *ImageNet*. Jan. 14, 2021. URL: [http : / / www . image - net . org /](http://www.image-net.org/) (visited on 01/14/2021).
- [8] *StackOverflow Question*. Jan. 14, 2021. URL: [https : / / stackoverflow . com / a / 42979315 / 3620718](https://stackoverflow.com/a/42979315/3620718) (visited on 01/14/2021).
- [9] Ralf-Dieter Bousseljot, D Kreiseler, and A Schnabel. *The PTB Diagnostic ECG Database*. 2004.
- [10] Patrick Wagner, Nils Strodthoff, Ralf-Dieter Bousseljot, Wojciech Samek, and Tobias Schaeffter. *PTB-XL, a large publicly available electrocardiography dataset*. 2020.
- [11] *Georgia 12-Lead ECG Challenge Database | Kaggle*. Mar. 15, 2021. URL: [https : / / www . kaggle . com / bjoernjostein / georgia - 12lead - ecg - challenge - database](https://www.kaggle.com/bjoernjostein/georgia-12lead-ecg-challenge-database) (visited on 03/15/2021).
- [12] Jianwei Zheng, Jianming Zhang, Sidy Danioko, Hai Yao, Hangyuan Guo, and Cyril Rakovski. "A 12-lead electrocardiogram database for arrhythmia research covering more than 10,000 patients". In: *Scientific Data* 7.1 (Feb. 2020).
- [13] Erick A Perez Alday, Annie Gu, Amit J Shah, Chad Robichaux, An-Kwok Ian Wong, Chengyu Liu, Feifei Liu, Ali Bahrami Rad, Andoni Elola, Salman Seyedi, Qiao Li, Ashish Sharma, Gari D Clifford, and Matthew A Reyna. "Classification of 12-lead ECGs: the PhysioNet/Computing in Cardiology Challenge 2020". In: *Physiological Measurement* 41.12 (Jan. 2021), p. 124003.
- [14] Ary L. Goldberger, Luis A. N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. "PhysioBank, PhysioToolkit, and PhysioNet". In: *Circulation* 101.23 (June 2000).

- [15] SNOMED CT - Classes | NCBO BioPortal. Mar. 16, 2021. URL: <https://bioportal.bioontology.org/ontologies/SNOMEDCT/?p=classes&conceptid=root> (visited on 03/16/2021).
- [16] I now call it "self-supervised learning". June 26, 2021. URL: <https://www.facebook.com/722677142/posts/10155934004262143/> (visited on 06/26/2021).
- [17] Mehdi Noroozi and Paolo Favaro. *Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles*. 2017. arXiv: [1603.09246 \[cs.CV\]](https://arxiv.org/abs/1603.09246).
- [18] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. *Unsupervised Representation Learning by Predicting Image Rotations*. 2018. arXiv: [1803.07728 \[cs.CV\]](https://arxiv.org/abs/1803.07728).
- [19] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. *Unsupervised Visual Representation Learning by Context Prediction*. 2016. arXiv: [1505.05192 \[cs.CV\]](https://arxiv.org/abs/1505.05192).
- [20] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. *A Simple Framework for Contrastive Learning of Visual Representations*. 2020. arXiv: [2002.05709 \[cs.LG\]](https://arxiv.org/abs/2002.05709).
- [21] Cheng-I Lai. "Contrastive Predictive Coding Based Feature for Automatic Speaker Verification". In: *arXiv preprint arXiv:1904.01575* (2019).
- [22] Michael Gutmann and Aapo Hyvärinen. "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. PMLR, 13–15 May 2010, pp. 297–304.
- [23] Mutual information - Wikipedia. Mar. 25, 2021. URL: https://en.wikipedia.org/wiki/Mutual_information (visited on 03/31/2021).
- [24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: [1301.3781 \[cs.CL\]](https://arxiv.org/abs/1301.3781).
- [25] Olivier J. Hénaff, Aravind Srinivas, Jeffrey De Fauw, Ali Razavi, Carl Doersch, S. M. Ali Eslami, and Aaron van den Oord. *Data-Efficient Image Recognition with Contrastive Predictive Coding*. 2020. arXiv: [1905.09272 \[cs.CV\]](https://arxiv.org/abs/1905.09272).
- [26] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: [1502.03167 \[cs.LG\]](https://arxiv.org/abs/1502.03167).
- [27] Zhiguang Wang, Weizhong Yan, and Tim Oates. *Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline*. 2016. arXiv: [1611.06455 \[cs.LG\]](https://arxiv.org/abs/1611.06455).
- [28] geekfeiw/Multi-Scale-1D-ResNet: pytorch code of multi scale 1d resnet, we hope it will help your research. May 26, 2021. URL: <https://github.com/geekfeiw/Multi-Scale-1D-ResNet> (visited on 05/26/2021).
- [29] alexnet.py pytorch. June 26, 2021. URL: <https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py> (visited on 06/26/2021).
- [30] Mohammad Kachuee, Shayan Fazeli, and Majid Sarrafzadeh. "ECG Heartbeat Classification: A Deep Transferable Representation". In: *2018 IEEE International Conference on Healthcare Informatics (ICHI)* (June 2018).

- [31] Stefan Harmeling, Guido Dornhege, David Tax, Frank Meinecke, and Klaus-Robert Müller. "From outliers to prototypes: Ordering data". In: *Neurocomputing* 69.13–15 (Aug. 2006), pp. 1608–1618.
- [32] *Brier Score: Definition, Examples - Statistics How To*. Feb. 9, 2022. URL: <https://www.statisticshowto.com/brier-score/> (visited on 02/16/2022).
- [33] Will Koehrsen. *Beyond Accuracy: Precision and Recall | by Will Koehrsen | Towards Data Science*. Mar. 22, 2021. URL: <https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c> (visited on 03/22/2021).
- [34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: [1512.03385 \[cs.CV\]](https://arxiv.org/abs/1512.03385).
- [35] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605.
- [36] *Inappropriate Sinus Tachycardia | Cedars-Sinai*. Sept. 7, 2021. URL: <https://www.cedars-sinai.org/health-library/diseases-and-conditions/i/inappropriate-sinus-tachycardia.html> (visited on 09/07/2021).
- [37] *Sinus Bradycardia | Cedars-Sinai*. Sept. 7, 2021. URL: <https://www.cedars-sinai.org/health-library/diseases-and-conditions/s/sinus-bradycardia.html> (visited on 09/07/2021).
- [38] Renu Khandelwal. *How to Visually Explain any CNN based Models | by Renu Khandelwal | Towards Data Science*. Jan. 18, 2022. URL: <https://towardsdatascience.com/how-to-visually-explain-any-cnn-based-models-80e0975ce57> (visited on 01/18/2022).
- [39] Jiaming Song and Stefano Ermon. *Multi-label Contrastive Predictive Coding*. 2020. arXiv: [2007.09852 \[cs.LG\]](https://arxiv.org/abs/2007.09852).
- [40] Jacob Gildenblat and contributors. *PyTorch library for CAM methods*. <https://github.com/jacobgil/pytorch-grad-cam>. 2021.
- [41] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. *Representation Learning with Contrastive Predictive Coding*. 2019. arXiv: [1807.03748 \[cs.LG\]](https://arxiv.org/abs/1807.03748).
- [42] *Entropy (information theory) - Wikipedia*. Mar. 29, 2021. URL: [https://en.wikipedia.org/wiki/Entropy_\(information_theory\)](https://en.wikipedia.org/wiki/Entropy_(information_theory)) (visited on 03/31/2021).
- [43] *Classification of 12-lead ECGs: the PhysioNet/Computing in Cardiology Challenge 2020 | PhysioNet/CinC Challenges*. Mar. 15, 2021. URL: <https://physionetchallenges.org/2020/> (visited on 03/16/2021).
- [44] Jan Chorowski, Grzegorz Ciesielski, Jarosław Dzikowski, Adrian Łaćucki, Richard Marxer, Mateusz Opala, Piotr Pusz, Paweł Rychlikowski, and Michał Stypulkowski. *Aligned Contrastive Predictive Coding*. 2021. arXiv: [2104.11946 \[cs.LG\]](https://arxiv.org/abs/2104.11946).
- [45] Patrick Wagner, Nils Strodthoff, Ralf-Dieter Bousseljot, Dieter Kreiseler, Fatima I. Lunze, Wojciech Samek, and Tobias Schaeffter. "PTB-XL, a large publicly available electrocardiography dataset". In: *Scientific Data* 7.1 (May 2020).

- [46] Ary L. Goldberger, Luis A. N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. "PhysioBank, PhysioToolkit, and PhysioNet". In: *Circulation* 101.23 (June 2000).
- [47] R. Bousseljot, D. Kreiseler, and A. Schnabel. "Nutzung der EKG-Signaldatenbank CARDIODAT der PTB über das Internet". In: *Biomedizinische Technik/Biomedical Engineering* (July 2009), pp. 317–318.
- [48] *MIT-LCP/wfdb-python: Native Python WFDB package*. Jan. 11, 2021. URL: <https://github.com/MIT-LCP/wfdb-python> (visited on 01/11/2021).
- [49] *Github Masterarbeit*. June 26, 2021. URL: <https://github.com/Lullatsch/ecg-cpc> (visited on 06/26/2021).

List of Figures

| | | |
|----|---|----|
| 1 | A single patients ECG with 500hz recording frequency. The classes <i>EKG: T wave abnormal</i> and <i>sinus rhythm</i> can both be found somewhere in the data. | 4 |
| 2 | All SNOMED Codes with their respective name and count in the datasets. | 5 |
| 3 | Dataset Class Distribution | 6 |
| 4 | Pearson Class Correlation Coefficient $\rho_{X,Y}$ heatmap: Given class x , what other classes y are likely to appear at the same patient?. $\rho_{X,Y} > 0$ = positive correlation, $\rho_{X,Y} < 0$ = negative correlation, $\rho_{X,Y} \approx 0$ no correlation . | 7 |
| 5 | CPC audio architecture how it is visualized in [1] | 9 |
| 6 | CPC audio architecture how it is visualized in [21] | 10 |
| 7 | "Given a set $X = x_1, \dots, x_N$ of N random samples containing one positive sample from $p(x_{t+k} c_t)$ and $N - 1$ negative samples from [...] $p(x_{t+k})$, we optimize": | 14 |
| 8 | ROC example | 34 |
| 9 | Different score metrics to measure performance. All score functions calculate the score for a given class i | 35 |
| 10 | Micro- and Macro-average calculations for a given score function. Example for TPR at bottom | 35 |
| 11 | Test scores for different models trained with a fraction of data labels | 42 |
| 12 | Test scores for different models trained with a fraction of data labels (more epochs) | 43 |
| 13 | Normalized ECG example from beginning Figure 1 | 44 |
| 14 | Parallel Coordinates Plot for all trained baseline networks from Figure 8a. | 52 |
| 15 | Parallel Coordinates Plot for all trained baseline networks from Figure 8a. | 53 |
| 16 | Two dimensional t-SNE embeddings of all context vectors | 54 |
| 17 | Two dimensional t-SNE embeddings of all latent vectors | 54 |
| 18 | Two dimensional t-SNE embeddings of all context vectors | 56 |
| 19 | Two dimensional t-SNE embeddings of all latent vectors | 57 |
| 20 | Two dimensional t-SNE embeddings of all context vectors (different model) | 58 |
| 21 | Two dimensional t-SNE embeddings of all latent vectors (different model) | 59 |
| 22 | Scatterplots showing from left to right: exact model predictions, model predictions minus specific class thresholds, reweighted prediction probabilities | 61 |
| 23 | Scatterplots showing from left to right: exact model predictions, model predictions minus specific class thresholds, reweighted prediction probabilities | 62 |

| | | |
|----|---|----|
| 24 | Scatterplot showing all exact model predictions, divided into groups depending on their ground truth label. | 63 |
| 26 | BL_v8 model gradients. Absolute gradient values utilized. Colored vertical bars/areas show "interesting/controversial" spots in the input data. | 67 |
| 27 | CPC model (with encoder_v0) gradients. Absolute gradient values utilized. Colored vertical bars/areas show "interesting/controversial" spots in the input data. | 68 |
| 28 | ECG examples, for the BL_v8 Model, Grad-Cam technique used. 147 feature map values stretched to input length of 4500. | 69 |
| 29 | ECG examples, for the CPC Model, Grad-Cam technique used. | 70 |
| 30 | ECG examples, for the TCN (down) architecture, Grad-Cam technique used. 4500 feature map values 'stretched' to input length of 4500. | 71 |

List of Tables

| | | |
|----|---|----|
| 1 | Model attributes summarized. Final Layer number refers to the enumeration <u>Final Output Layer 2.3.1</u> | 28 |
| 2 | Baseline Results | 37 |
| 3 | CPC as Baseline Results | 38 |
| 4 | CPC ROC-AUC scores, with no CPC layers updated during downstream training | 39 |
| 5 | CPC ROC-AUC scores, random weight initialization, no CPC layers updated during downstream training | 39 |
| 6 | ROC-AUC scores for standard CPC models without pretraining (random weights); trained for 20 epochs on the downstream task (with the CPC weights frozen) | 39 |
| 7 | CPC ROC-AUC scores, with all layers updated in downstream training | 40 |
| 8 | Baseline ROC-AUC scores | 45 |
| 9 | CPC ROC-AUC scores | 46 |
| 10 | Average CPC ROC-AUC scores, no pretraining | 47 |
| 11 | CPC ROC-AUC scores, with all layers updated in downstream training | 47 |
| 12 | CPC ROC-AUC scores, 40 Downstream Epochs | 49 |
| 13 | CPC ROC-AUC scores: normalized latents | 49 |
| 14 | CPC ROC-AUC scores: <code>cpc_encoder_likev8</code> | 50 |
| 15 | CPC ROC-AUC scores: <code>cpc_autoregressive_hidden</code> | 50 |
| 16 | CPC ROC-AUC scores: <code>cpc_predictor_nocontext</code> | 50 |