Project 1 - ECG Time Series

by Martin Tschechne, Han Bai, Nora Moser

Introduction

We first trained the Recurrent Neural Network model (RNNs) on ECG time signals on two different data sets independently, Arrhythmia Dataset with multi-class labels and PTB Diagnostic ECG Database with binary-class labels (See section *Data* in the Appendix). Secondly, transfer learning was used in this task. We pre-trained a model on the larger multi-class data set, and then used the pre-trained model as feature-extractor for the smaller binary-class data set and retrained with frozen layers and unfrozen layers. We then explored the performance of transfer learning by comparing Accuracy, F1-Score, AUCOR and AUCPR is studied. Further, more sophisticated model architectures such as combinations of Convolutional Neural Networks (CNNs) and LSTM cells and gradient boosting instead of fully-connected layers were tested in our project.

Models

- **Base Models**:
    - LSTM + FC: Three LSTM cells with decreasing number of units followed by a relu-activated fully-connected layer and a softmax output layer.
    - CNN + LSTM + FC: A combination of 1D max-pooled convolutional layers with one succeeding LSTM cell, followed by a relu-activated fully-connected layer and a softmax output layer.

  We refer to all layers before the fully connected layers as *base layers* or *feature extractors*. For a detailed visualization of the model architectures see section *Model Architectures* in the appendix.

- **Transfer Learning Models**:

  The idea of transfer learning is to pre-train a model such as CNNs on a data set, preferably a very large one, and then use the pre-trained model either as an initialization or a fixed feature extractor for a similar targeted task on a different data set that usually is smaller. In case of ECG classification, we did transfer learning by feature extraction from pre-trained model and then train a classifier in top of it. Specifically, we first used the large MIT-BIH data set to train a model. Then we used the pre-trained model and retrained it on the much smaller PTBDB data set. We further differentiate between two methods of training here: *frozen* base layers and *unfrozen* base layers. Frozen base layers refers to not updating the base layers of the pre-trained model, whereas in the unfrozen method we train all layers.

- **XGBoosted Models**:

  Gradient Boosting has been proved many times in the past to be a very effective algorithm for a large variety of machine learning task by winning multiple Kaggle competitions. In this study we compare the performance of fully connected layers and gradient boosting by using the pre-trained base layers as feature extractors and replace the fully connected layers with a gradient boosting model, XGBoost[3].
  We further examined the transfer learning performance of XGBoost by using a feature extractor trained on the MIT-BIH data set and training the XGBoost model on the extracted features from the PTBDB data set.

Training Protocol

First the data was split in a stratified fashion into 3 disjunct subsets, training set, validation set and testing set. Where the validation set was used for early-stopping, while the testing set for final evaluation of the model. Due to the heavy class imbalance in both datasets (NIR of 0.828 for MIT-BIH and 0.722 for PTBDB) minority classes in the training datasets were upsampled without any data augmentation techniques to the same size of the majority class. Base models were trained using Adam optimizer with default parameters and clipped gradients. Further a batch-size of 128 was selected. The training continued until validation loss did not improved significantly for 3 epochs in order to avoid over-fitting. During training of transfer learning models a exponentially decaying learning rate was used, which started from the default Adam learning rate. All XGBoost models were trained with the same parameters: `max_depth=10`, `n_estimators=256`, `learning rate=0.1` until no further improvement on the validation data was achieved for 3 epochs.

Results

Based on performance metrics, we could conclude that transfer learning models result in a better performance than models trained on a single data set. Our models were able to reach similar accuracy and F1 scores in 3 out of 4 times compared to their counterparts trained only on one data set. The results also suggested using convolutional layers in front of recurrent cells. In this case convolutional layers extract one-dimensional features, which correspond to characteristic shapes within the the signal. An interesting results is the fact that the performance of almost all models can be further enhanced by removing the fully connected layers and pass the intermediate output to a XGBoosting model.
One shortcoming is the lack of interpretability of the obtained classifications. Models do not indicate why certain samples are labeled as abnormal, which would be critical for actual usage. A possible less critical use case for these algorithms could be the usage in fitness tracking devices which show a notification to go see a physician if multiple samples are labeled as abnormal over a period of time.

Performance on test data set:
MIT-BIH No Information Rate: **0.828**
PTBDB No Information Rate: **0.722**

Models	**MIT-BIH**[*]	**PTBDB**[*]	**PTBDB**[°]	**PTBDB**[†]
LSTM + FC	F1: 0.184 Acc: 0.823	F1: 0.787 Acc: 0.776 AUROC: 0.808 AUPRC: 0.934	F1: 0.419 Acc: 0.722 AUROC: 0.5 AUPRC: 0.861	F1: 0.371 Acc: 0.565 AUROC: 0.397 AUPRC: 0.805
CNN + LSTM + FC	F1: 0.868 Acc: 0.971	F1: 0.940 Acc: 0.951 AUROC: 0.947 AUPRC: 0.982	F1: **0.988** Acc: **0.990** AUROC: **0.988** AUPRC: **0.996**	F1: 0.992 Acc: **0.994** AUROC: **0.990** AUPRC: **0.996**
LSTM + XGB[‡]	F1: 0.875 Acc: 0.976	F1: 0.971 Acc: 0.977 AUROC: 0.968 AUPRC: 0.988	F1: 0.963 Acc: 0.970 AUROC: 0.955 AUPRC: 0.983	-
CNN + LSTM + XGB[‡]	F1: **0.916** Acc: **0.985**	F1: 0.983 Acc: **0.986** AUROC: **0.980** AUPRC: **0.993**	F1: 0.981 Acc: **0.990** AUROC: 0.977 AUPRC: 0.991	-
XGB	F1: 0.896 Acc: 0.979	F1: 0.970 Acc: 0.976 AUROC: 0.966 AUPRC: 0.987	-	-
Kachuee, et al.[1]	Acc: 0.934	-	F1: 0.951 Acc: 0.959	-
Baseline[2]	F1: 0.915 Acc: **0.985**	F1: **0.988** Acc: 0.983	F1: 0.969 Acc: 0.956	F1: **0.994** Acc: 0.992

[*] Only trained on this dataset
[°] Transfer Learning, pre-trained model trained on MIT-BIH, retrained with **frozen** base layers
[†] Transfer Learning, pre-trained model trained on MIT-BIH, retrained with **unfrozen** base layers
[‡] Base layers always frozen to train XGBoost

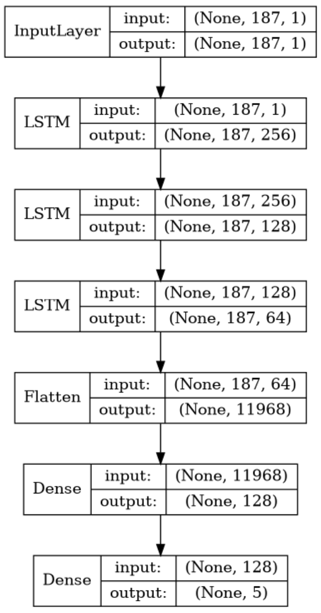
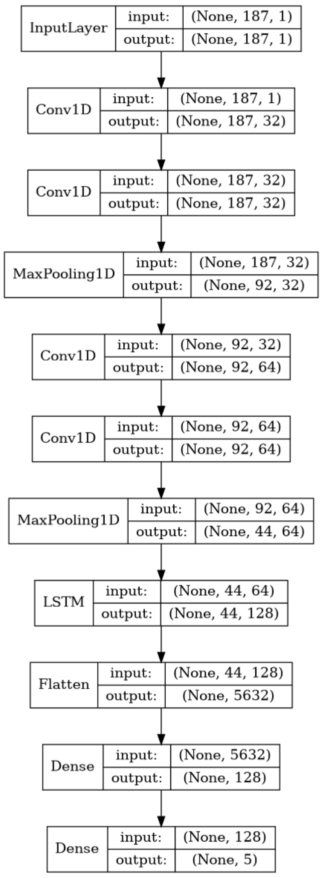
Appendix

Data

Arrhythmia Dataset	The PTB Diagnostic ECG Database
Number of Samples: 109446 Number of Categories: 5 Sampling Frequency: 125Hz Data Source: [Physionet's MIT-BIH Arrhythmia Dataset] Classes: 0: Normal (82.8%) 1: Supraventricular ectopic beat (2.5%) 2: Ventricular ectopic beat (6.6%) 3: Fusion beat (0.7%) 4: Unknown beat (7.3%)	Number of Samples: 14552 Number of Categories: 2 Sampling Frequency: 125Hz Data Source: [Physionet's PTB Diagnostic ECG Database] Classes: 0: normal (27.8%) 1: abnormal (72.2%)

Classes 1 to 4 from the Arrhythmia dataset are combined in a single class labeled 'abnormal' in the PTB data set.
Remark: All the samples are cropped, downsampled and padded with zeros if necessary to the fixed dimension of 188.

Model Architectures

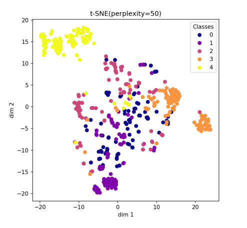
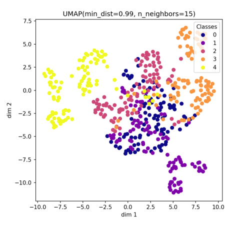
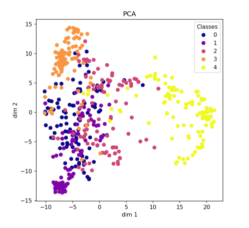
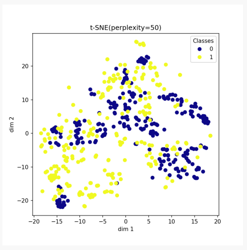
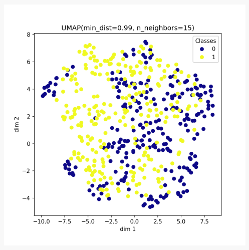
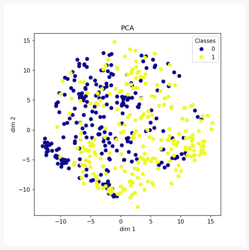
LSTM + FC	CNN + LSTM + FC
	

InputLayer input: (None, 187, 1) output: (None, 187, 1)
LSTM input: (None, 187, 1) output: (None, 187, 256)
LSTM input: (None, 187, 256) output: (None, 187, 128)
LSTM input: (None, 187, 128) output: (None, 187, 64)
Flatten input: (None, 187, 64) output: (None, 11968)
Dense input: (None, 11968) output: (None, 128)
Dense input: (None, 128) output: (None, 5)

InputLayer input: (None, 187, 1) output: (None, 187, 1)
Conv1D input: (None, 187, 1) output: (None, 187, 32)
Conv1D input: (None, 187, 32) output: (None, 187, 32)
MaxPooling1D input: (None, 187, 32) output: (None, 92, 32)
Conv1D input: (None, 92, 32) output: (None, 92, 64)
Conv1D input: (None, 92, 64) output: (None, 92, 64)
MaxPooling1D input: (None, 92, 64) output: (None, 44, 64)
LSTM input: (None, 44, 64) output: (None, 44, 128)
Flatten input: (None, 44, 128) output: (None, 5632)
Dense input: (None, 5632) output: (None, 128)
Dense input: (None, 128) output: (None, 5)

Embedding Visualizations

Displayed are the learned representations of the base layers from the CNN+LSTM model, i.e. best performing model, trained on the MIT-BIH data mapped into 2 dimensions. Here used three famous dimension reduction algorithms, t-distributed Stochastic Neighbor Embedding (t-SNE), Uniform Manifold Approximation (UMAP) and Principal Component Analysis (PCA) to map the 5632-dimensional embedding into a 2-dimensional space. A subsample of 500 data points for each dataset was selected randomly in a class-balanced way, i.e. each class has the same number of samples.

	t-SNE	**UMAP**	**PCA**
MIT-BIH			
PTBDB			

Reproducibility

To reproduce the results, download the zipped data form the sources mentioned above. Create the folders `data` and `data/raw` inside the project folder. Extract the zip-file inside `data/raw`.

select the configuration file corresponding the the model configuration in the table above from the `config-file` directory and run

```
python train.py --config ./config-files/config.yaml
```

for base and transfer models,

```
python train_base_xgb.py --config ./config-files/gxb-config.yaml
```

XGBoosted models and

```
python train_xgb.py --config ./config-files/gxb-config.yaml
```

to reproduce reference results of pure XGBoost models.

Performance metrics for each model as well as weights, architectures and test set predictions are saved in the `results` folder in a single folder for each model.

References

[1] Mohammad Kachuee, Shayan Fazeli, and Majid Sarrafzadeh. "ECG Heartbeat Classification: A Deep Transferable Representation." arXiv preprint arXiv:1805.00794 (2018) .

[2] CVxTz's GitHub implementation: ECG_Heartbeat_Classification (link)

[3] XGBoost (link)