Acknowlegment

We expect everything to work on an isolated python environment created as per the instructions below, but in case you face any issues running the code please feel free to contact us by email or on MS-Teams (irodrigu@student.ethz.ch, kumarsh@student.ethz.ch, neumannam@ethz.ch).

We have tested our code in an environment with the following specifications:

- Machine:
  - CPU: `11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz`
    - `x86_64`
  - RAM: 16 GB
- OS: `Ubuntu 20.04.4 LTS`
- Python Version: `3.7.11`

Creating isolated execution environment

- Go to the `src` directory
- Execute the following in sequence (enter yes when prompted):

```
conda create -n ml4hc_ais python==3.7.11
conda activate ml4hc_ais
pip install -r requirements.txt
```

- Now the environment should be ready
- Make sure to check that the environment is activated before running the code

> **Please make the datasets available in the `input` folder (Also pelase check section: Overview of Code Structure)**

```
├── input
│   ├── mitbih_test.csv
│   ├── mitbih_train.csv
│   ├── ptbdb_abnormal.csv
│   └── ptbdb_normal.csv
│
└── src
```

Overview of Code Structure

- The modules containing models are prefixed with `model_` except for `model_factory.py`, which contains classes to resolve and load models by name. We experimented with different models and all of them can be found in these files (however we have mentioned only the relevant ones in the report.)
- `data_loader.py` contains dataloaders for the two datasets and some extensions thereof, to provide support for loading , splitting and subsampling the data.

- `trainingutil.py` : This contains the trainer (e.g. `BaseTrainer`, `CnnTrainer`) classes to abstract training loop; and experiment pipeline classes (*e.g.* `ExperimentPipelineForClassifier`) to abstract away logging, data loading, optimizer & cost_function iniatilazion *etc.* and provide and interface to run the training based on a single configuration file (refer below).

Please note: the directory structure must be as follows:

Please make sure to put the datasets (as they are not added in the submission)

```
├── input
│   ├── mitbih_test.csv
│   ├── mitbih_train.csv
│   ├── ptbdb_abnormal.csv
│   └── ptbdb_normal.csv
├── README.md
├── README.pdf
├── resources
│   ├── ...
│
└── src
    ├── data_loader.py
    ├── ensemble.py
    ├── eval.py
    ├── experiment_configs
    │   ├── archive
    │   │   ├── experiment_3_a_-PTB-_cnn_auto_encoder.yaml
    │   │   ├── ....
    │   │
    │   ├── experiment_0_a_vanilla_cnn_mitbih.yaml
    │   ├── .....
    │
    ├── main.py
    ├── metric_auroc_auprc.py
    ├── model_cnn_2d.py
    ├── model_cnn_ae.py
    ├── model_cnn.py
    ├── model_cnn_res.py
    ├── model_factory.py
    ├── model_lstm.py
    ├── model_rnn_b.py
    ├── model_transformer.py
    ├── requirements.txt
    ├── saved_models
    │   ├── 2022-03-28_000731__CnnWithResidualConnection
    │   │   ├── best_model.ckpt
    │   │   └── config.yaml
    │   ├── ...
    ├── svm.py
    ├── trainingutil.py
    └── util.py
```

How to run training?

For SVM models

In order to train SVM models you can call:

```
python svm.py train [mitbih|ptbdb] [OPTIONS]
```

specifying the dataset over which to train the model. Running the script will train several SVMs with different hyperparameter selections, and output test metrics for the SVM that performed better in F1 score over cross validation. The script offers as options:

- iter: indicates how many different randomly chosen hyperparameter configurations to train on (default 5)
- jobs: number of threads to use for training SVM models in parallel (default 1)

We provide the SVM models with best performance we found in files:

- src/saved_models/svm_mitbih.pickle for MIBIH
- src/saved_models/svm_ptbdb.pickle for PTBDB

For PyTorch based Deep Learning Models:

- N.B. Most of our models are implemented in PyTorch and the steps below apply to all such models

> Steps for training:

- Make sure you are inside the `src` directory
- To start training execute:
    - ```
      python trainingutil.py --config <path-to-run-config-file>
      ```
    - *e.g.*

      ```
      python trainingutil.py --config
      experiment_configs/experiment_0_a_vanilla_cnn_mitbih.yaml
      ```
    - The **src/experiment_configs** directory contains many configs, that we have used for running our experiments. You can choose any of those or create your own.

The steps above will do the following:

- It will start training
- create `runs` folder if not already present
- create a timestamped folder with `tag` value provided in the config as suffix *e.g.* : `2022-03-29_014835__exp_0_b_VanillaCnnPTB`
      ```

    - this folder will be used to output the best model
    - in this folder `logs` subfolder will be created in which tensorboard logs and AUROC and AUPRC curves will be saved.
- the best model will be saved if the validation F1 has increased when compared to the last best F1

These are the training Config File used for different experiments (training)

Config File	**Experiment description**
`experiment_0_a_vanilla_cnn_mitbih.yaml`	Vanilla CNN Model on PTBDB Dataset (Binary classification)
`experiment_0_b_vanilla_cnn_ptb.yaml`	Vanilla CNN model on MITBIH dataset (5 Class classification)
`experiment_10_a_PTB_rnn_vanilla.yaml`	Vanilla RNN Model on PTBDB
`experiment_10_b_MITBIH_rnn_vanilla.yaml`	Vanilla RNN Model on MITBIH
`experiment_4_a_-PTB-_rnn.yaml`	Bidirectional RNN on PTBDB
`experiment_5_a_MITBIH_rnn.yaml`	Bidirectional RNN on MITBIH
`experiment_2_a_-PTB-_cnn_with_residual_block.yaml`	CNN with residual blocks (for PTBDB dataset)
`experiment_1_a_cnn_with_residual_block.yaml`	CNN with residual blocks (for MITBIH dataset)

Config File	Experiment description
`experiment_6_b_cnn2d_ptb.yaml`	2D-CNN model for PTBDB
`experiment_6_a_cnn2d_mitbih.yaml`	2D-CNN model for MITBIH
`experiment_11_a_transfer_cnn_with_residual_block_mit_to_ptb.yaml`	Transfer Learning - without freezing weights
`experiment_11_c_transfer_frozen_cnn_with_residual_block_mit_to_ptb.yaml`	Transfer Learning with frozen weights

- There are more in the `experiment_configs/archive` if you wish to explore those as well. But the above mentioned are the ones we included in our report.

For SVM ensembles

In order to train the bagging SVM ensemble you can run:

```
python ensemble.py svm [OPTIONS]
```

where the following options are offered:

- -samples: number of samples to be used to train each bagging classifier
- -models: number of models to train in the ensemble
- -jobs: number of threads to use for training models in parallel
- -modelpath: path to a saved individual SVM model with the hyperparameters to be used. If not indicated, a default SVM configuration is used.

How to run evaluation?

- `eval.py` is the script for running evaluations
  - It prints out accuracy and F1 score for a given model on the given dataset
  - It also shows PRC and ROC curves for binary classification task
- We have kept the models we trained in the `src/saved_models/` directory
  - These models may be used for evaluation or you can give path to your own model

> The command to run is as follows:

```
python eval.py --model <model_class_name> --data <dataset_name>
```

This will use the model with class name `model_class_name` and select the dataset based on the `dataset_name` from **saved_models** e.g.

- ```
  python eval.py --model VanillaCnnMITBIH --data mitbih
  ```
- ```
  python eval.py --model VanillaCnnPTB --data ptbdb
  ```

In case you wish to use the model trained by you. You may specify the model path using `--model-path` like in the example below

- ```
  python eval.py --model CnnModel2DPTB --data ptbdb --model-path
  "saved_models/2022-03-28_224128__CnnModel2DPTB/best_model.ckpt"
  ```

The following is a list of Models you can try:

Model Class Name	Model Description	Dataset Name to Use
`VanillaCnnPTB`	Vanilla CNN Model for PTBDB Dataset (Binary classification)	`ptbdb`
`VanillaCnnMITBIH`	Vanilla CNN model for MITBIH dataset (5 Class classification)	`mitbih`
`VanillaRNNPTB`	RNN Model for PTBDB	`ptbdb`
`VanillaRNNMITBIH`	RNN Model for MITBIH	`mitbih`
`RnnModelPTB`	Bidirectional RNN for PTBDB	`ptbdb`
`RnnModelMITBIH`	Bidirectional RNN for MITBIH	`mitbih`
`CnnWithResidualConnectionPTB`	CNN with residual blocks (for PTBDB dataset)	`ptbdb`
`CnnWithResidualConnection`	CNN with residual blocks (for MITBIH dataset)	`mitbih`
`CnnModel2DPTB`	2D-CNN model for PTBDB	`ptbdb`
  ```

Model Class Name	Model Description	Dataset Name to Use
`CnnModel2DMITBIH`	2D-CNN model for MITBIH	`mitbih`
`CnnWithResidualConnectionTransferMitbihToPtb`	CNN with residual connection (to be used for transfer learning with unfrozen weights)	`ptbdb`
`CnnWithResidualConnectionTransferMitbihToPtbFrozen`	CNN with residual connection (but freezes some weights) (It is to be used for transfer learning)	`ptbdb`

Testing NN ensemble:

In order to test different combinations of NN ensembles using our trained models, you can run from the `src` directory:

```
python ensemble.py nn [mitbih|ptbdb]
```

which will print the performance of several ensembles.

Generating plots for data subsample:

In order to generate plots with the performance of an SVM model with respect to different downsampling rates over the training data, run:

```
python svm.py samples [mitbih|ptbdb] -model MODEL_PATH -iter N
```

indicating the path of a saved SVM model from which to take the hyperparameters (or use a default SVM if not indicated), and the number `N` of equally spaced maximum number of samples per class to use from the training data.

The plot will be generated in a .jpeg in the current directory (e.g. `sample_svm_2022-03-29_154116_.jpeg`) along with a csv file with the f1 score, accuracy, and training time for each number of samples used (e.g. `sample_svm_2022-03-29_154116__stats.csv`).

Appendix