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 ().

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

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 (`#TODO`) 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).

How to run training?

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 `experiment_configs` folder 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

How to run evaluation?

- `eval.py` is the script for running evaluations
- 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
  "runs/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`
`CnnModel2DMITBIH`	2D-CNN model for MITBIH	`mitbih`