Technical documentation

ZAGREB, MAY 2023

1 Project setup

Project structure for easy management:

```
lightning-instrument-classifier/
├── configs
│   └── config.yaml
├── data
│   ├── raw
│   ├── processed
│   └── test_dataset
├── deploy
│   ├── static
│   ├── templates
│   └── rest_api.py
├── models
├── notebooks
├── pictures
├── solution
│   ├── solution.py
│   ├── solution.json
│   └── test_dataset_solution.json
├── src
│   ├── data
│   │   └── data_preprocessing.py
│   ├── models
│   │   └── train.py
│   ├── fusion
│   │   └── fusion.py
│   └── run.sh
└── environment.yml
```

Detailed explanation of directories and files:

- configs/
  - config.yaml file with all of the parameters needed for preprocessing the data, training the models and fusion of the models in the end.
- data/
  - Raw data from the IRMAS dataset.
  - Processed data that was transformed from audio signals to melspectrograms, modgd-grams and pitchgrams.
  - test_dataset is the folder containing all the audio files. wanted for prediction. You should put all of your testing audio data in this folder before starting the prediction.
- deploy/
  - The main REST API endpoint script 'rest_api.py' that listens to the incoming requests, processes them and returns the prediction as a JSON object. We built that script using **Flask** framework.
  - The folder 'templates/' containing file 'index.html' that renders a web page with the REST API functionality.
  - The folder 'static/' containing CSS file 'styles.css' and similar files needed to render the web page.
- models/
  - cnn_mel.ckpt that was trained on melspectrograms.
  - cnn_modgd.ckpt that was trained on modgd-grams.
  - cnn_pitch.ckpt that was trained on CQT-chromagrams.
- notebooks/
  - feature_extraction.ipynb - demonstrating mel- spectrogram, modgdgram and CQT-chromagram extraction from an audio file.
  - dataset.ipynb - demonstrating dataset statistics.
  - visualizations.ipynb - used for creating graphs and barplots for the project documentation.
- pictures/
  - Screenshots from the notebooks that we used in the project documentation.
- solution/
  - test_dataset_solution.json that was the required output for the model prediction based on the test_dataset folder that we received.
  - An empty file solution.json in which the prediction will be generated from your test_dataset folder.

  - solution.py script that generates the solution.json.

- src/
  - data/
    → data_preprocessing.py script used for preprocessing the data from raw audio signal into mel-spectrograms, modgdgrams and CQT-chromagrams.
  - models/
    → train.py script used for training the models.
  - fusion/
    → fusion.py script used for combining the predictions of the three models into a single prediction.
  - run.sh bash script used for running all of the mentioned scripts from the 'src' folder all at once reconstructing the whole pipeline from dataset to the finished model.

- environment.yml
  - used for conda environment setup

2 Instructions to start and test the program

In the project we used **Pytorch Lightning** which provided us with a lightweight interface on top of **PyTorch** that simplified and standardized the process of training and fusing our deep learning models, allowing for faster development and easier scaling.
Navigate to the project directory pytorch-instrument-classifier-api. Install the conda environment by running 'conda env create -f environment.yml'.

2.1 Model pipeline

To run the whole pipeline with the bash scripts, follow the steps:

- Navigate to the /src/ folder.
- Run the bash script with 'bash run.sh'.

To run the pipeline separately:

- For data preprocessing, navigate to /src/data/ and run 'python data_preprocessing.py'. This step saves the processed data in /data/processed/.
- To train the models navigate to /src/models/ and run 'python train.py'.
- To calculate fusion thresholds navigate to /src/fusion/ and run 'python fusion.py'. This will save fusion thresholds to .npy files in the working directory.

As mentioned above, we have config.yaml configuration file that contains all the hyperparameters used for the data preprocessing, training, aggregation, and fusion stages. This file allows us to keep track of all the hyperparameters used throughout the project, and it can be easily modified if required. We used **hydra** framework for configuring our application.

2.2 Get the solution.json

- Navigate to the deploy folder of the project by typing 'cd deploy'.
- Run the python script 'rest_api.py' by typing 'python rest_api.py' which will run the Flask server.
- Place all the audio files for which you want the prediction in the data/test_dataset folder.
- Go back one step up in the directory with 'cd ..' and go into the solution folder with 'cd solution'.
- Run the command 'python solution.py'.
- When the 'solution.py' program ends, you will have all the predictions in the solution.json file in the same folder as solution.py in an ascending order by file name number.

2.3 Get to the web page

- Navigate to the deploy folder of the project by typing 'cd deploy'.
- Run the python script 'rest_api.py' by typing 'python rest_api.py' which will run the Flask server.
- Open your browser and type 'http://localhost:5050'. This will display the web interface of the app.
- Click on the 'Upload file' button and select an audio file with a .wav extension.
- Once the file has been uploaded, click on the 'Predict' button and wait for a few seconds until the web page displays all the instruments that appear in the uploaded audio file.