Brno University of Technology
Faculty of information technology

SUR – Machine Learning and Recognition

Target Person Detection

April 22, 2024 Maxim Plička (xplick04), Šimon Šmída (xsmida03)

1 General Information

This project was developed with Python. It is recommended to use a virtual environment to manage dependencies, which are listed in the `requirements.txt`. For step-by-step usage of the project, please refer to the `README.md` in our public GitHub repository.

2 Dataset

The dataset used for this project consists of both the original data (acquired as a part of the assignment) and augmented data (described below), distributed across four main categories: `non_target_train`, `target_train`, `non_target_dev`, and `target_dev`, which are further organized into two primary sets: `dev` (development set) and `train` (training set).

Each set contains a mix of audio files (`.wav`) and images (`.png`) featuring target and nontarget individuals. To obtain the augmented data, run the `src/dataAugmentation.py` (requires original data folder in the same path). The resulting directory structure is on the right.

```
./augmented_data/
├── dev/
│   ├── non_target_dev
│   └── target_dev
└── train/
    ├── non_target_train
    └── target_train
```

2.1 Augmentation Techniques

To enhance the model's robustness and improve generalization, we applied *data augmentation* techniques to both audio and image files.

Audio Augmentation

- **Noise addition** to simulate different environmental conditions.
- **Time shifting** to represent temporal variations.
- **Speed** and **pitch adjustments** to cover a range of voice modulations.
- **Volume changes** to ensure the model's sensitivity across different sound levels.

Image Augmentation

- **Geometric transformations** such as rotation, flipping, and translation to mimic different orientations and positions.
- **Photometric transformations** including greyscale conversion, color jittering, and lighting adjustments to simulate various lighting conditions.
- **Noise** addition and **blurring** to test the model's performance under low-quality and unclear visual inputs.

3 Person Detection Using Speech Recording - MLP

For person detection from speech recordings, we employed the MultiLayer Perceptron (MLP) architecture as proposed in the referenced article [1]. The model operates on preprocessed data, where the original audio is transformed into MFCC feature matrices. Each row of these matrices represents a 10ms segment of the original audio, with each column representing an MFCC feature. These segments are then concatenated into a single long segment, typically spanning about 2 seconds of input audio. Subsequently, the matrix is reshaped into a vector of size 8000 (400 segments multiplied by 20 MFCC features), serving as the input for the MLP. Although we experimented with using a logarithmic filter bank as input, it failed to yield satisfactory results. One potential improvement for this model could involve segmenting sections of audio that do not include the speaker's voice.

3.1 Model Architecture

The MLP architecture consists of three hidden linear layers, each comprising 64 nodes and employing the sigmoid activation function. To enhance model generalization, a dropout layer has been incorporated. After exploring various dropout configurations and assessing the distance between training and validation accuracies, we settled on a rate of 0.2. The final layer, with an output dimension of 1.

3.2 Model Training

During training, we employed the Adam optimizer with a learning rate of 0.0001. To address the class imbalance issue (the target class having less training data than the non-target class), we utilized weighted binary cross-entropy with logits as the loss function.

Throughout the evaluation of the model architecture, we experimented with different numbers of epochs, ultimately settling on 50 epochs with early stopping. Additionally, we tested various batch sizes, with the optimal batch size being determined as 64.

For the final training phase, we trained the model on the entire dataset for 100 epochs, incorporating early stopping to prevent overfitting. This decision was made considering that the entire dataset is larger than one-tenth of the dataset, equivalent to one-fold in cross-validation.

3.3 Evaulation Method

The model's performance was rigorously assessed using k-fold cross-validation, which provides a robust estimate of its generalization ability. Additionally, after the final model selection, we conducted a thorough evaluation on a separate development set to estimate real-world performance. The table below showcases experiments conducted through 10-fold validation, exploring various settings of model parameters significant in determining the final parameter configuration.

Folds Mean Accuracy	Epochs	Batch size	Dropout	Segment Size	Learning Rate	Activations
99.02%	50	64	0.2	400	0.0001	Sigmoid
98.54%	50	32	0.2	200	0.0001	Sigmoid
98.31%	50	64	0.2	200	0.0001	Relu
97.8%	50	32	0.5	200	0.0001	Sigmoid
97.78%	50	128	0.2	200	0.0001	Sigmoid
97.36%	50	64	0.2	100	0.0001	Sigmoid
95.94%	50	64	0.2	200	0.0001	Tanh
95.84%	100	128	0.2	200	0.0001	Sigmoid
95.67%	50	64	0.2	50	0.0001	Sigmoid
90.2%	50	64	0.2	10	0.0001	Sigmoid
85.54%	50	64	0.2	200	0.001	Sigmoid

To run training, cross-validation or get predictions of evaluation data, see the commands below:

```
# Perform cross-validation
python3 src/audioDetectionNN.py --train --evaluate

# Perform training on whole dataset (final model in trainedModels/)
python3 src/audioDetectionNN.py --train

# Get predictions from the final model (evaluation data are expected to be in data/eval)
python3 src/audioDetectionNN.py
```

4 Person Detection Using Speech Recording - GMM

We additionally implemented a Gaussian mixture model for this task. The model utilizes preprocessed audio in the form of MFCC coefficients as input, without the concatenation into one long segment step. It employs two Gaussian mixtures to approximate the distribution of two classes: one class represents the target individual we aim to detect, while the other class represents all other individuals present in the audio recordings.

4.1 Model Architecture

As previously mentioned, the model comprises two Gaussian mixtures, each of which can be decomposed into individual weighted Gaussian distributions. Through experimentation, we determined that the optimal configuration for modeling the target data consists of three Gaussian distributions, while the non-target data is best represented by ten Gaussian distributions. The model uses maximum aposteriori classification to classify data.

4.2 Model Training

The model parameters are initialized as follows:

- Centers are randomly selected from data points corresponding to each class.
- Covariance matrices are approximated from the variance within the specified class.
- Weights are uniformly chosen.

Training is conducted using the Expectation-Maximization (EM) algorithm, terminating when the logarithmic likelihoods of the model mixtures no longer change significantly. Additionally, we incorporated a check for a minimum covariance matrix to prevent the overfitting of any Gaussian distribution to the data.

4.3 Model Evaluation

We employed cross-validation to validate the model, splitting the data into two parts: training data and evaluation data. Through this approach, we achieved an accuracy of 94.52% on the validation dataset.

To run training, validation or get predtiction of evaluation data, see the commands below:

```
# Perform cross validation
python3 src/audioDetectionGMM.py --train --evaluate

# Perform training on whole dataset (obtaining final model in trainedModels/)
python3 src/audioDetectionGMM.py --train

# Get predictions from final model (evaluation data are expected to be in data/eval)
python3 src/audioDetectionGMM.py
```

5 Person Detection Using Face Images - CNN

For person detection using image data, we utilized a convolutional neural network (CNN). The architecture was heavily inspired by the VGG architecture [2] because of its simplicity and proven efficacy for image-related tasks. Our model was designed to capture essential features for the detection task. Additionally, we experimented with the architecture itself, adding batch normalization and dropout layers (discussed below).

5.1 Model Architecture

The model is structured into two primary sections: **feature extraction** and **classification**.

The **feature extraction** part employs *convolutional* layers, which use 32 and 64 kernels, and *max pooling*, a standard approach in CNNs for reducing spatial dimensions and emphasizing important features. This setup helps in efficiently processing the input images.

In the **classification** part, the network transitions to a *fully connected* layer that integrates the high-level features into a format suitable for classification. This section uses a dense layer with 128 units followed by ReLU activation, typical of CNNs for classification tasks.

To enhance the original VGG-inspired design, we incorporated *batch normalization* into each convolutional layer. Although not originally part of VGG, batch normalization has proven effective in reducing internal covariate shifts, stabilizing and accelerating the training process [3]. Additionally, *dropout* layers were introduced after max pooling and in the classifier to mitigate overfitting by randomly omitting subset features during training. We discuss the results of these techniques in Section 5.3. We experimented with different values of the dropout, and the best results were obtained when we differentiated the dropout for the convolutional layers (set to 0.1) and fully-connected layers (set to 0.3). The CNN architecture can be found in `src/imageModel.py`.

5.2 Model Training

The model was trained using the Adam optimizer, which was chosen for its ability to adjust the learning rate dynamically, often leading to quicker convergence. Initially, we employed the Binary Cross-Entropy (BCE) loss, suitable for binary classification tasks. Subsequently, to address the significant class imbalance—approximately 6.4 times more non-target than target data—we switched to BCEWithLogitsLoss. This allowed us to assign weights to the classes proportionally based on their prevalence. Additionally, we implemented oversampling to further mitigate the imbalance issue, which improved the model's performance. Our training approach also incorporated early stopping during validation. This technique halts training when there are no further improvements in validation loss, serving as an effective strategy to prevent overfitting. The CNN model training logic is implemented in `src/visualDetection.py`.

5.3 Evaluation Method

The model's performance was again assessed using k-fold cross-validation, similarly as in Section 3.3. After that, the model was trained on all of the available data (train and dev), to maximize its ability to generalize to unseen data. The learning rate was set to 0.0001 for all runs, based on the experiments (higher values of lr oscillated or diverged).

Folds Mean Accuracy	Epochs	Batch size	Conv. Dropout	FC Dropout	Batch Norm.
97.30%	25	32	0	0	no
98.02%	25	32	0	0	yes
98.24%	25	32	0	0.3	yes
97.51%	25	32	0.1	0.3	no
99.41%	25	32	0.1	0.3	yes
98.22%	25	64	0.1	0.3	yes
98.12%	25	32	0.3	0.5	yes

To run training, validation, or get prediction of evaluation data, see the commands below:

```
# Perform training on the whole dataset (the final model in trainedModels/)
python3 src/visualDetection.py

# Perform cross-validation
python3 src/visualDetection.py --cross_valid

# Get predictions from the final model (evaluation data are expected to be in data/eval)
python3 src/visualDetection.py --eval <path_to_cnn_model>
```

6 Techniques

This section outlines key techniques in our models and how they contributed to the performance.

6.1 Data Augmentation

Data augmentation involved applying transformations like rotation, scaling, and color adjustment to images, and adding noise to audio files, thereby enhancing the model's ability to generalize from limited training data. To generate the augmented data and store it in the structure described in Section 2 run the `src/dataAugmentation.py`.

6.2 Addressing Class Imbalance

To tackle the issue of class imbalance, where the dataset had significantly more instances of non-target data compared to target data, we implemented two main strategies:

Weighted Loss We utilized the Binary Cross-Entropy with Logits Loss and adjusted the weights for the classes proportionally to their representation in the dataset. This approach allows the loss function to penalize misclassifications of the minority class more than those of the majority class, effectively balancing the scale of influence between the classes during model training.

Oversampling In addition to weighted loss, we employed oversampling to increase the number of samples from the underrepresented class in the training data. This method involved replicating the minority class examples to balance the class distribution before training, thereby providing the model with a more balanced view of the classes and enhancing its ability to generalize across both classes effectively.

6.3 Cross-Validation

We employed k-fold cross-validation to ensure that our evaluation is robust and reliable, utilizing the entire dataset for both training and validation to prevent overfitting and to better estimate the model's performance on unseen data.

6.4 Early Stopping

Early stopping was used to halt training as soon as the validation loss ceased to decrease, preventing overfitting and saving computational resources while maintaining model effectiveness.

6.5 Dropout

Dropout layers were incorporated to randomly ignore a subset of neurons during training, which helps prevent the network from becoming too dependent on any single neuron and promotes generalization.

6.6 Batch Normalization

Batch normalization was applied to accelerate training, reduce the number of training epochs required, and stabilize the learning process by normalizing the inputs of each layer.

6.7 MFCC

Mel Frequency Cepstral Coefficients (MFCCs) were extracted from audio data to capture the timbral aspects, providing a compact representation of the sound data which is crucial for effective audio analysis and classification.

References

[1] U. Ayvaz, H. Gürüler, F. Khan, N. Ahmed, T. Whangbo, and A. Abdusalomov, "Automatic speaker recognition using mel-frequency cepstral coefficients through machine learning," *Computers, Materials & Continua*, vol. 71, pp. 5511–5521, 01 2022.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.