SUR project 2024/25

Samuel Šimún (xsimun04), Daniel Žárský (xzarsk04)

4.4.2025

Contents

1. Introduction

This project was developed as part of the Machine Learning and Recognition course at BUT FIT. The primary objective is to recognize individuals based on their face images (in PNG format) and short voice recordings (in WAV format). The training dataset consists of 31 distinct individuals. Notably, the project constraints prohibit the use of any pre-trained models or additional external data.

2. Face recognition

2.1 Data Augmentation

Images are randomly rotated within a small range of ±5 degrees, as they are generally upright and not significantly tilted, and no horizontal or vertical flipping is applied to maintain natural face orientation. To simulate slight variations in face position, images are shifted horizontally and vertically by up to 10% of their dimensions, and minor changes in camera distance are represented by random zooming within a 10% range. Additionally, images are randomly sheared by up to 10% to introduce perspective distortions. To account for different lighting conditions, the brightness of images is randomly adjusted within the range [0.8, 1.2], and channel values are shifted by up to 50 units to simulate variations in color balance. Any empty areas created by these transformations are filled using the nearest available pixels, ensuring the output images remain valid and visually coherent.

2.2 CNN

The proposed face recognition model is a sequential convolutional neural network (CNN) designed specifically for small datasets and low-resolution images. Its architecture consists of four convolutional layers with ReLU activation, followed by a flattening layer and a dense output layer with softmax activation for classification.

2.2.1 Model Tuning

Based on that architecture we decided to utilize tensorflow keras Tuner to find the best hyperameters for model. With this tuner we are looking for best combination of *number of filters* for each convolutional layer and *learning rate*. We defined the searching space for Tuner:

- **1st Conv Layer:** 32 to 64 filters
- **2st Conv Layer:** 16 to 64 filters
- **3st Conv Layer:** 128 to 256 filters
- **4st Conv Layer:** 128 to 256 filters

- **Initial Learning Rate:** $10^{-3}, 10^{-4}, 10^{-5}$

As an algorithm for tuning, we chose **Hyperland**, which uses adaptive resource allocation and early stopping to quickly find "good" models. We run this tuner with a target to maximise validation accuracy, with a maximum of 20 epochs.

2.2.2 Layer Details

Based from results from tuner 2.2.1, we used following model with total 9,197,583 trainable params:

- **1st Conv Layer:** 64 filters, 3×3 kernel, input shape $(80, 80, 3)$, ReLU activation, 2×2 max pooling.
- **2nd Conv Layer:** 48 filters, 3×3 kernel, ReLU activation. c
- **3rd Conv Layer:** 192 filters, 3×3 kernel, ReLU activation.
- **4th Conv Layer:** 256 filters, 3×3 kernel, ReLU activation.
- **Flatten Layer:** Converts feature maps to a vector.
- **Output Layer:** Dense layer with 31 units, softmax activation.

2.2.3 Training and Regularization

Training is stabilized using:

- Early stopping (monitors validation accuracy, patience 10).
- Learning rate reduction (monitors validation loss, patience 8).

2.3 Experiments and evaluation

Narrowing the augmentation parameters—such as rotation_range=5, width_shift_range=0.1, height_shift_range=0.1, shear_range=0.1, and zoom_range=0.1—had a positive impact on model performance. Additionally, reducing the brightness_range from [0.5, 1.5] to [0.8, 1.2] improved results, and increasing the channel_shift_range to 50 also contributed positively.

We tried these architecture chnages with validation loss achieving 0.9677 accuracy.

3. Voice recognition

3.1 Architecture

This method presents a speaker recognition system based on Mel-Frequency Cepstral Coefficients (MFCCs) and Gaussian Mixture Models (GMMs). Audio samples recorded at 16 kHz are converted into 13-dimensional MFCC features, capturing spectral characteristics of speech. For each individual, features from multiple recordings are aggregated and used to train a GMM using the Expectation-Maximization algorithm. The models are initialized with random means, uniform weights, and covariance matrices regularized to prevent singularities. Each speaker is represented by a unique GMM, enabling identification through likelihood-based comparison. The employed model uses 12 components and was trained for 32 iterations.

3.2 Experiments and evaluation

As well as for training convolutional network we used one more file per class from `dev` directory. We empirically tested various settings, and observed that using between 8 and 20 GMM components generally produced good results, with 85%+ accuracy. The best performance, achieving cca 90% accuracy, was observed with 12 to 16 components. To balance generalization and representational capacity, we selected 12 components, as this provided sufficient space to capture diverse feature characteristics while avoiding overfitting.

The number of EM iterations was also chosen empirically. We initially started with 50 iterations and gradually reduced it. Accuracy noticeably dropped below 25 iterations, so we selected 32 iterations as a conservative yet effective choice. The final accuracy was 90.32%, using the enlarged training set.

4. Combinig scores

Inference script combines audio and visual models using a weighted sum rule, where the parameter $\alpha = 0.35$ is set to prioritize the image classifier due to its improved accuracy. The system first loads a Gaussian Mixture Model (GMM) for speaker identification from audio data, as well as a pretrained Convolutional Neural Network (CNN) for visual classification. For each input pair, it computes log-likelihoods from the audio model and class probabilities from the image model. These predictions are then combined using the weighted sum rule: combined_score = $\alpha \times$ sound_score + $(1 - \alpha) \times$ image_score, where the image classifier is given more weight to reflect its superior performance based on experiments we performed. The final class for each input is determined by selecting the highest combined score, and the results are written to a formatted output file.

5. Installation and Setup

5.1 Environment Setup

Ensure that you have Python 3.10 or higher and pip installed on your system. To create the Python virtual environment, run the following command in your terminal:

```
python3 -m venv env
```

Activate the environment by running:

```
source env/bin/activate    # On Linux/MacOS
env\Scripts\activate       # On Windows
```

5.2 Installing Dependencies

Install the required dependencies using:

```
pip install -r requirements.txt
```

5.3 Running the GMM Training

We assume your training and validation data are organized in a folder containing subfolders, where each subfolder holds the data for one class. Navigate to the `sound` directory. To train the Gaussian Mixture Model (GMM), execute the following command:

```
python3 train_gmm.py --train_dir your_train_set --eval_dir your_val_data
```

This script will display evaluation results and save the trained models as `gmm_speaker_models.joblib`, which can be subsequently used for standalone speaker recognition tasks. By default, the number of components is set to 12, and the number of expectation-maximization iterations is set to 32.

5.4 Running the CNN Training

To train the Convolutional Neural Network (CNN) for face recognition, prepare the training and validation data in the same format as for GMM. Navigate to the `image` directory from the root folder Execute the training script with your dataset paths:

```
python3 train_cnn.py --train_dir your_train_data --eval_dir your_val_data
```

This script will output two files:

- `face_recognition_model.h5`: The trained face recognition model, ready for standalone use.
- `class_indices.json`: A mapping file that associates model output class indices to the corresponding identities.

The default number of training epochs is 100, and the default batch size is 16. These parameters can be adjusted by passing additional arguments to the script if necessary.

5.5 Running combined inference

To run this code, execute the script from the command line and provide the required arguments. Specify the path to the sound model with `-s` or `--sound` (in .joblib format), the path to the image model (in .h5 format) with `-i` or `--image`, and the path to the data directory containing audio (`.wav`) and image (`.png`) files with `-d` or `--data`. Optionally, you can set the output file path with `-o` or `--output` (defaults to `output.txt`), the alpha parameter for the sum rule with `-a` or `--alpha` (defaults to `0.5`), and the path to the class indices file with `-ci` or `--class_indices` (defaults to `../class_indices.json`). For example:

```
python script.py -s gmm_speaker_models.joblib -i face_recognition_model.h5
-d eval/ -o results.txt -a 0.35 -ci class_indices.json.
```