Contents

Introduction

Speaker recognition is the task of identifying a person from his/her voice. In this lab report, we will examine a speaker recognition system that will use *x-vectors* embeddings to classify the speakers. The system has been developed to solve the VoxCeleb challenge.

The system *extracts* the fixed-size embeddings trying to include in these representations as much discriminative information as possible for the speaker recognition task. A few architectures of neural networks can be used with this purpose, as well as different loss functions.

The code that I used to test the model, which was provided by Alicia Lozano, can be found in this link.

Examining the code

In this section, we will examine the provided code in order to find the most relevant parts of it. Let us begin by examining the `trainSpeakerNet.py` file, since it is the main script. The code is properly commented so identifying the different parts is quite simple.

Questions.

- *Which line of the code trains the model?*

  We found `line 169`:

  ```
  loss, traineer = s.train_network(loader=trainLoader);
  ```

  which trains the code. The variable `s` was previously initialized as `s = SpeakerNet(**vars(args));`, creating an instance of the SpeakerNet defined model class. This class has the called `train_network` function, which performs (vaguely explained) a classic forward pass with loss computation and returns the average loss and the `train eer`.

- *Which command loads a pre-trained model?*

  Having a look from lines $112-121$, we find that in lines 116 and 120, the function `s.loadParameters(args)` is called. This function loads the pretrained model from a specific file. As we can see in the code, a distinction is made in our case, being able to load the model from the `save_path` or from the argument `args.initial_model`.

- *Which line evaluates the performance of the neural network?*

  A few lines below the training line, in line 176 we find:

  ```
  sc, lab, _ = s.evaluateFromList(args.test_list, print_interval=100,
                            test_path=args.test_path, eval_frames=args.eval_frames)
  ```

  which evaluates the model in the `test_list`. In this function, all the features for each of the test file audios (*.wav*) files are extracted and then the scores are computed using `pairwise_distance`.

- *Which variable contains the scores of the trials list? In which file are those scores saved?*

  Firstly, we find in line 176 that the scores of the trials are stored in the variable `sc`. Then, in line 177 we observe that the scores are thresholded and stored in the variable `result`. Then, they are saved in the `scorefile` file. This is a variable that, in line 150 opened a file called `result_save_path/scores.txt`, where `result_save_path` came as an argument from the execution of the script.

- *Which variable controls the number of epochs(iterations) that the model is trained?*

  We find in line 198 and 201 that the variable `it` is modified and checks the `max_epoch` indicated as an argument. Then, is the variable `max_epoch` the one that controls the number of epochs that the model is trained.

- *Which parameter determines the loss function?*

  We find that in line 29 the argument `--trainfunc` with description *'Loss function'* is declared. Then, it is passed to the model class in the creation of the instance of the model (passing all the arguments at the same time).

- *What is the purpose of the parameter `test_interval`?*

  Having a look at the description of the parameters, in `line 27` we read that this parameter controls how many train epochs we make before every evaluation. For instance, if $test\_interval = 10$, then every 10 training epoch we perform an evaluation on the `test_list` (`line 176`) and save the results.

- *Which argument has to be modified in order to change the architecture of the model?*

  The architecture of the model is defined in the argument `--model`. As we can see, in the folder `models`, there are a few implementations of `ResNet` and `VGG` that could be used to change the architecture of the model.

- *What is the purpose of the parameter `nClasses`? How would you obtain this value from the dataset? Write the command that you would use.*

  The parameter `nClasses` controls the number of speakers that are used in the softmax layer. Let us explain how we would obtain this from the dataset. As we are explained in the provided jupyter notebook, the training data is stored in the folder `data/voxceleb2`. Inside this folder, we find a large number of subfolders, each one containing the audios from a different speaker, which **represents a class** in our problem. So, to obtain the parameter `nClasses`, we would have to count the number of folders stored in the directory `data/voxceleb2`. We can do this in linux by using a `pipe`, listing the folders in the directory and then counting the number of lines (using the linux command `wc -l`). The execution provides the following result:

  ```
  !ls data/voxceleb2/ | wc -l

  4953
  ```

  It has to be remarked that the validation set does not have the same number of unique classes. When we execute this same command in the folder `data/voxceleb1`, we obtain that there are only 40 classes.

Evaluating pretrained models

We are given two pretrained models using the *VoxCeleb2* dataset (the dev partition). In this section, we evaluate the models and compare the results. To evaluate the models, we have to execute the previous script, using the argument `--eval` as well as the dataset path, the model type and some extra parameters. We show the execution output (removing the `warning` lines).

First Model:

```
!python ./trainSpeakerNet.py --eval --model ResNetSE34L --log_input True \
         --trainfunc angleproto --save_path exps/test_lite --eval_frames 400 \
         --test_list lists/veri_test.txt \
         --initial_model pretrained_models/baseline_lite_ap.model
```

```
Embedding size is 512, encoder SAP.
Initialised AngleProto
Initialised Adam optimizer
Initialised step LR scheduler
Model has 1437080 parameters
Model pretrained_models/baseline_lite_ap.model loaded!
Reading 4700 of 4715: 28.75 Hz, embedding size 512
Computing 37700 of 37720: 1776.69 Hz
```

```
EER 2.1792
```

Second Model:

```
!python ./trainSpeakerNet.py --eval --model ResNetSE34V2 --log_input True \
  --encoder_type ASP --n_mels 64 --trainfunc softmaxproto \
  --save_path exps/test_v2 --eval_frames 400 --test_list lists/veri_test.txt \
  --initial_model pretrained_models/baseline_v2_ap.model
```

```
Embedding size is 512, encoder ASP.
Initialised Softmax Loss
Initialised AngleProto
Initialised SoftmaxPrototypical Loss
Initialised Adam optimizer
Initialised step LR scheduler
Model has 11103416 parameters
Model pretrained_models/baseline_v2_ap.model loaded!
Reading 4700 of 4715: 4.19 Hz, embedding size 512
Computing 37700 of 37720: 1766.00 Hz
```

```
EER 1.1771
```

Questions.

- *How does each of the previous model perform?*

  We obtain the following results:

    – Model 1 EER: 2.1792.
    – Model 2 EER: 1.1771.

  Further explanation on these results will be provided in the following questions.

- *Which one is the best and why?*

  As we will explain in the following question, the lower the EER is, the better the model is. This implies that, in our case, the second model is better than the first one.

- *What is the EER and how can we interpret it?*

  EER stands for **Equal Error Rate**. To understand this concept, we assume that the concepts of *TP, TN, FP, and FN* are known. Using these concepts and considering N as the total number of examples, we can define the **False Rejection Rate (FFR)** and **False Aceptation Rate (FAR)** as the ratios:

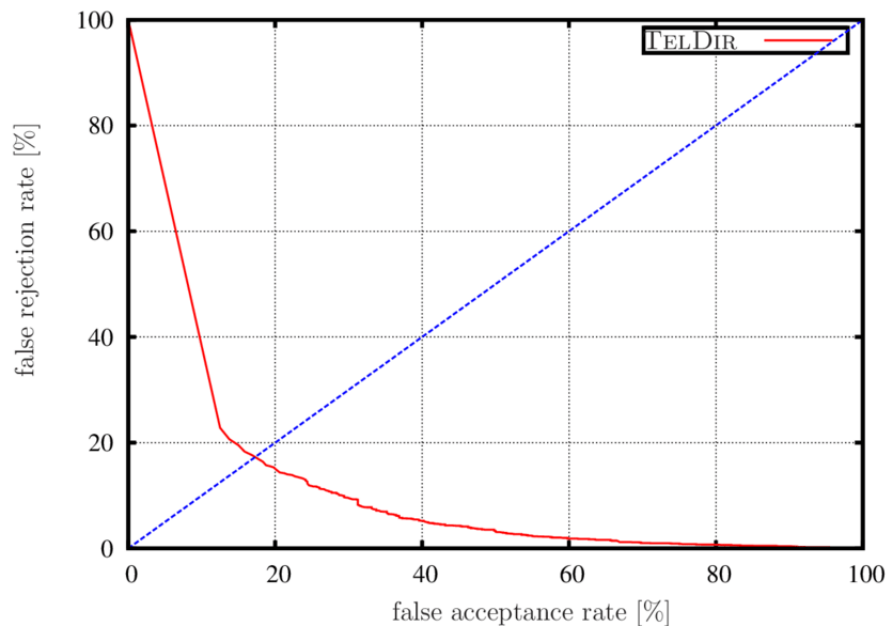$$FFR = \frac{FN}{N}, \quad FAR = \frac{FP}{N}.$$

Figure 1: Example of DET curve.

For each of these two metrics, we have to choose a *threshold* that we would like our model to fulfill, using the number of *allowed False Positives and False Negatives*. The **Dectection Error Tradeoff (DET)** curve represents this tradeoff trying to decrease both FAR and FFR. We can see an example of DET curve in Figure 1.

With all these concept, we can define the EER as the point of the DET curve where $FAR = FRR$, that is, the point where our model rejects the same amount of positive examples than the number of negative samples that it accepts.

At this point, we get to understand why a lower EER is better, since if the EER is lower it means that both our FAR and FFR (which are *negative* quantities) are low.

- *What are the differences between the models?*