Machine Learning in Practice Report

Custom-X-vector Deep Neural Network

Floris Rossel (s1010794)
Guido Miggelenbrink (s1061928)
Gijs Thuis (s4490444)
Stan Vissers (s1092712)

March 22, 2023

1 Introduction

In recent years, speaker recognition has received much attention due to increased availability of computing power and the developments of deep learning techniques. Deep neural network models, in particular, have proven to be highly effective at recognising patterns in various types of data, including speech signals.

This paper focuses on the sub-task of speaker verification using short audio speech signals. Our goal is to create and optimise a machine learning model that distinguishes whether two audio recordings belong to the same speaker or different speakers. To achieve this, we were provided a small subset of the VoxCeleb dataset containing 110 unique speakers.

We explored the state-of-the-art X-vector and VGG-M speaker verification models, starting from basic versions and iteratively refining and expanding them based on literature and the results of training evaluations. Additionally, we investigated the effectiveness of four data augmentation techniques.

2 Method

We began by conducting a literature review on state-of-the-art speaker verification models. Based on this review, we implemented, trained, optimised, and evaluated various models using the tiny-voxceleb skeleton code and data set [Vae+23].

To improve the models, we took an iterative approach, making both informed and experimental changes to the models and their corresponding parameters. Informed choices were based on the literature and suggestions from forums, while experimental adaptions were based on evaluation results from each iteration. Additionally, we investigate the effectiveness of data augmentation techniques in improving the performance.

The methods section is structured as follows: In the first subsection, we describe the provided skeleton code and data set. Next, we describe the data augmentation techniques we used. Finally, we describe the iterative process we followed to arrive at the final models.

2.1 tiny-voxceleb skeleton code and dataset

At the start of this project, we were provided with a skeleton code that implements a one-layer CNN with statistical pooling. The initial model trains and evaluates on the provided tiny-voxceleb dataset. Using the parameters described in Table 1, this model reached an Equal Error Rate (EER) of 27.22% when trained for 30 epochs. The tiny-voxceleb

Hyperparameter	**Value**
Batch size	128
Loss function	NLL
Audio length number of frames	48000
Learning rate	$3e^{-3}$
Number of MFFCs	40
Embedding size	128
Optimizer	Adam
Dropout	None
Augmentation	None

Table 1: Default settings for the models.

dataset is a subset of the VoxCeleb dataset and contains 110 unique speakers, 100 in the training set and 10 in the development set [NCZ17]. The skeleton code preprocesses these WAV audio files, stored in .tar files, via a data pipeline and outputs the data as tuples containing the sample id, waveform and ground truth.

The basic tiny-voxceleb model contains three stages. During training and inference the first two stages are equal while the last stage differs. The first stage converts audio waveforms into Mel Frequency Cepstral Coefficients (MFCCs). MFCCs represent the audio waveform as a series of vectors of a fixed length and captures relevant spectral features. This stage is referred to as feature extraction.

The MFCCs go into the second stage which is a neural network that captures speaker characteristics. The final layer of this stage creates a fixed-dimension speaker embedding. This stage is referred to as the compute embedding step.

During training, the last stage has a softmax layer that maps the embedding to a probability distribution over the speaker classes. This stage is referred to as prediction. This probability distribution is then compared to the

ground truth labels using a loss function to calculate the model error. The error is then back-propagated through the model to adjust the model weights and minimise the loss.

During inference, the last stage uses speaker embeddings as the representation of input speech signals. The two speaker embeddings are compared using a similarity score, such as cosine similarity. By applying a decision threshold it is determined whether two speech signals are from the same speaker. This stage is referred to as classification.

2.2 Data augmentation

In our approach, data augmentation was randomly applied to 50% of the samples in the data loader pipeline, right after loading the data, and before chunking the data on the first 3 seconds. Four methods were used for data augmentation, namely adding Gaussian noise, applying random time stretching or compression, reverberation, and random time masking.

Gaussian noise was added to the audio samples by sampling noise from a standard normal distribution and scaling it by 0.4 times the standard deviation of the audio sample. Random time stretching or compression was applied by randomly sampling a 'tempo' value from a uniform distribution between 0.7 and 0.9 (with a chance of 50%) or between 1.1 and 1.3 (also with a 50% chance), and then using the 'tempo' effect from the SoX library to apply the tempo value to the audio sample. Reverberation was applied using the SoX 'reverb' effect command with the following parameters: reverberance 60%, HF-damping 50%, room-scale 100%, and stereo-depth 0%. The pre-delay was set to 0 ms, and the wet-gain was set to 0 dB. Random time masking was applied by first converting the waveform to a spectrogram and then truncating it to 242 frames, which corresponds to 3 seconds. The Torch TimeMasking method was then applied to the spectrogram with a max possible length of 70 frames, which corresponds to roughly 0.87 seconds. This cut out a segment of the spectrogram on the time axis at a random location. Finally, the spectrogram was converted back to a waveform using the Griffin-Lim algorithm in the TorchAudio library, using default parameters [Pas+19].

2.3 Experimental process

2.3.1 X-vector

X-vector models have been shown to achieve state-of-the-art results for speaker verification tasks [Sny+18]. Since we have a small dataset available, we used the stage-wise X-vector model because it requires less training data according to Snyder er al. (2017), as opposed to end-to-end models which require larger amounts of training data to be effective [Sny+17]. Additionally adapting only the embedding stage in the skeleton code is less error-prone than implementing an end-to-end model. Table 2 shows the stage-wise X-vector architecture. Note that our initial model extracts embeddings at layer 6 before the non-linearity and that the in bold rows are not implemented. These belong to the X-vector architecture that extracts embedding at layer 7.

We experimented with different hyperparameters, starting with the default hyperparameters from Table 1, and comparing changes with respect to this baseline with EER 24.50%, found in run 23 in Appendix A.1 . Increasing the embedding size and number of MFCCs improved performance, whereas increasing the amount of audio frames negatively impacted the results. We found a learning rate of $3e^{-3}$ to be optimal.

Sieun Park et al. (2021) suggests that stochastic gradient descent (SGD) generalises better than Adam, but converges slower [Par21]. We found that the SGD optimiser with a momentum of 0.9 performed worse than Adam for our experiments.

X-vectors embeddings are initially extracted at layer 6, directly after the pooling layer. However, adding additional layers of neural networks could help maximising the distance between speakers in the embedding space. [Sny+17] shows that an additional fully connected layer after layer 6 can improve performance . The results section of the paper presents a table indicating that the extraction of embeddings from layer 7 has a lower EER compared to layer 6 for audio lengths of 5 seconds. Conversely, for audio signals exceeding a duration of 10 seconds, the EER is comparatively better for embeddings extracted at layer 6. Based on this evidence and since our experiments use 3-second audio signals, we chose to extract embeddings from layer 7. After changing the X-vector model, we evaluated the updated one with the default hyperparameters and improved the EER to 22.07%. Again increasing the number of MFCCs to 100 with an embedding size of 128 further improved performance to 21.03%.

Two augmentation strategies were tested on the best performing model and hyperparameters. The first strategy randomly applies augmentation to individual samples with a 50% probability, resulting in an EER of 20.14%, which is the best score we achieved. This setup can be found on our GitLab repository [Vis+23]. Experiment 10 in Appendix A.1 shows the corresponding parameters. The second strategy involves duplicating the dataset and applying augmentation effects to the duplicated data, which achieved an EER of 23.78%. This result contradicts the general expectation that increasing the volume of training data would improve model performance. Nonetheless, our results

No.	Layer	Context frames	Context	Data size
Compute embedding stage				
1	TDDN-LeakyRelu	$\{t-2, t-1, t, t+1, t+2\}$	5	120x128
2	TDDN-LeakyRelu	$\{t-2, t, t+2\}$	9	384x128
3	TDDN-LeakyRelu	$\{t-3, t, t+3\}$	15	384x128
4	TDDN-LeakyRelu	{t}	15	128x128
5	TDDN-LeakyRelu	{t}	15	128x384
-	MeanStdPool	-	-	384x768
6	Linear	-	-	768x128
-	**LeakyReLU**	**-**	**-**	**128x128**
7	**Linear**	**-**	**-**	**128x128**
Compute prediction stage				
-	LeakyReLU	-	-	128x128
-	BatchNorm1d	-	-	128x128
-	Dropout	-	-	128x128
8	Linear	-	-	128x128
-	LeakyReLU	-	-	128x128
-	BatchNorm1d	-	-	128x128
9	Linear	-	-	128xnum_speakers
-	LogSoftmax	-	-	

Table 2: Stage-wise X-vector architecture, extract embeddings at layer 7. Leave out the bold rows to extract embeddings at layer 6.

show that data augmentation is an effective and easy-to-implement technique for improving the performance of the X-vector model.

2.3.2 VGG-M

We implement a VGG-M model according to the architecture described in [NCZ17], adapted to our specific data size and problem and using 1d convolutions instead of 2d. This means that the fc6 layer now uses a kernel size of 3, to account for the input size at that point in the model, while the number of output channels will be set to the embedding size we define during the experiments, instead of 4096. Additionally, the apool6 layer has been changed into the provided MeanStatPool1D layer. Lastly, we use 100 output channels in the fc8 layer, since that is the number of speakers in our training set. The architecture is given in Table 3. Note that ReLU activation layers, batch normalization layers and the final LogSoftMax layer have been omitted from the table.

Layer	Kernel size	Number of input channels	Number of output channels	Stride	Data size
Compute embedding stage					
conv1	7	number of MFCCs	96	2	241
mpool1	3	-	-	2	119
conv2	5	96	256	2	59
mpool2	3	256	384	2	29
conv3	3	384	256	1	14
conv4	3	256	256	1	14
conv5	3	-	-	1	14
mpool5	5	-	-	3	14
fc6	3	256	embedding size	1	3
Compute prediction stage					
MeanStatPool1D	1	-	-	1	1
fc7	1	embedding size	1024	1	1
fc8	1	1024	100	1	1

Table 3: VGG-M architecture, adapted from [NCZ17].

3 Results

3.1 X-vector

Tables 4-7 in appendix section A.1 show the results of all experiments of our X-vector model variants with different hyperparameters. Our best-performing model achieved an EER of 20.14% on the evaluation set [Vis+23]. This model was trained with half of the data augmented. In comparison to the default X-vector model, which achieved an EER of 24.50% in run 23, and to the improved X-vector model without data augmentation, which achieved an EER of 21.03% [1] in run 4, it is clear that our model adaptation provides a substantial performance improvement. Furthermore, our results also show that data augmentation leads to a significant improvement.

3.2 VGG-M

Training the VGG-M model without any additions and using the default parameters, a baseline EER of 26.58% was obtained. Since we started out with an EER a little over 27%, this improvement is not spectacular. To fine-tune the model, we varied our hyperparameters and experimented with loss functions, optimizers and data augmentation. The best VGG-M model was trained using the default settings and adding only data augmentation, yielding an EER of 24.69%. A full list of experiments using VGG-M can be found in Appendix A.2. Ultimately VGG-M experiments were discontinued due to the model severely overfitting on the training data.

Experiments were continued with a smaller convolutional network similar to VGG-M, containing just three convolutional layers scaling the number of channels from the number of MFCCs to 64, 128 and finally the number of embeddings. The first linear layer takes the number of embeddings as input and outputs 256 features. Using this smaller network in combination with a cross entropy loss function and data augmentation, an EER of 21.77% was obtained. Full experiments are presented in Appendix A.3

4 Discussion

The main challenge we faced during our research was to find a model that generalises well when trained on a small data set. We found that increasing the number of speaker characteristics, such as MFCCs, or the embedding size did not always translate well to our test evaluation set. We attempted several methods to improve the model's ability to generalise, including comparing the effectiveness of the SGD over the Adam optimiser. While Sieun Park et al. (2021) suggested that SGD generalises better than Adam, we did not observe this in our project [Par21]. We suspect that the differences would have been smaller or even reversed if we had trained the model with the SGD optimiser for a longer period, as we only trained it for 30 epochs.

Another approach we took to improve the model's generalisation ability was data augmentation. Although augmenting the dataset by adding new augmented data did not improve results as much as expected, adding augmentation on top of some of the data improved performance considerably.

We found that the X-vector model worked well for our small data set, significantly better than more standard convolutional models. In future work, we recommend further enhancing the X-vector model with additional forms of data augmentation. For example, we could experiment with techniques such as shifting the pitch of the speaker or passing the audio files through band filters. We anticipate that these approaches will help us decrease the EER even further.

5 Conclusion

In this study, we trained two speaker verification models on a small subset of the VoxCeleb dataset. Our initial results showed that the X-vector model outperformed more traditional convolutional models, such as VGG-M, in terms of accuracy. We found that the X-vector model was able to adapt effectively to the limited amount of data available by incorporating augmented versions of some of the input data. Our evaluation results demonstrated that the X-vector model continued to perform well on an independent evaluation set, indicating its ability to generalise effectively.

Overall, our findings suggest that the X-vector model is a promising approach for speaker verification tasks involving small datasets. Further investigation is warranted to explore additional methods for augmenting the dataset and enhancing the model's performance. Nevertheless, the results of this study provide valuable insights into

[1] Model available at `https://gitlab.science.ru.nl/stanvissers/tiny-voxceleb-skeleton-2023.git` in the Custom-X-vector branch.

the development of more effective and efficient speaker verification models, with important implications for a wide range of applications.

6 Author Contributions

Initially, our work was divided into two groups: one group consisting of Gijs and Stan who focused on searching for relevant literature and implementing effective speaker recognition models, and another group consisting of Guido and Floris who focused on developing and integrating data augmentation techniques into the data loading pipeline. In the final stage of the project, Gijs was searching the relevant literature for suitable model parameters, Guido and Stan were busy running the X-vector and VGG-M models respectively with the different parameters and collecting and processing the data, and Floris began writing the report. After data collection was completed, we all finished writing the report.

7 Evaluation of the Process

At the beginning, the clear division of tasks helped us structure and set up the project. Ultimately however, due to the business of the cluster and time constraints of group members, we could not carry out the same experiments for both models. It would have been nice to compare these models more by varying one parameter per comparison to study the effect of individual parameters more precisely. Time was also lost by submitting jobs to the cluster and finding out hours later they had failed after a few seconds, effectively causing us to wait a few more hours for results.

Despite these issues, we managed to present a finished work that clearly shows how our methods give improvements and managed to test a large number of model variations, and are quite satisfied with the results.

8 Evaluation of the Supervision

The main supervision we received was during the check-up with the TA. This conversation helped us steer in the right direction to finalise the project and start writing the report. Additionally, the Discord server has been very helpful in solving both issues during model coding that also arose for other groups and helping to solve cluster issues. We do think one more check-up two weeks earlier might have been nice to get an idea of our progress earlier.

References

[NCZ17] Arsha Nagrani, Joon Son Chung, and Andrew Zisserman. "VoxCeleb: a large-scale speaker identification dataset". In: *CoRR* abs/1706.08612 (2017). arXiv: 1706.08612. URL: http://arxiv.org/abs/1706.08612.

[Sny+17] David Snyder et al. "Deep Neural Network Embeddings for Text-Independent Speaker Verification". In: *Proc. Interspeech 2017*. 2017, pp. 999–1003. DOI: 10.21437/Interspeech.2017-620.

[Sny+18] David Snyder et al. "X-Vectors: Robust DNN Embeddings for Speaker Recognition". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 5329–5333. DOI: 10.1109/ICASSP.2018.8461375.

[Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[Par21] Sieun Park. *A 2021 Guide to improving CNNs-Optimizers: Adam vs SGD*. 2021. URL: https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008.

[Vae+23] Nik Vaessen et al. *tiny-voxceleb-skeleton-2023*. 2023. URL: https://gitlab.science.ru.nl/imc030/tiny-voxceleb-skeleton-2023.

[Vis+23] Stan Vissers et al. *MLiP Group 6 - tiny-voxceleb*. 2023. URL: https://gitlab.science.ru.nl/stanvissers/tiny-voxceleb-skeleton-2023.

A Appendix

A.1 X-vector experimental results

File name model	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py
Run ID	1	2	3	4	5	6	7	8	9	10	11	12
Batch size	128	128	128	128	128	128	128	128	128	128	128	128
Learning rate	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003
Number of epochs	40	40	40	40	40	40	40	40	40	40	40	40
Number of frames	48000	48000	48000	48000	48000	48000	48000	48000	48000	48000	48000	48000
Number of mfcc	40	40	80	100	100	100	100	100	100	120	140	120
Embedding size	128	128	128	128	128	128	128	128	128	128	128	128
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam
Momentum												
Loss function	NLL	NLL	NLL	NLL	NLL	NLL	NLL	NLL	NLL	NLL	NLL	NLL
Gamma	1	0.8	0.8	1	1	1	1	1	1	1	1	1
Data augmentation	0	0	0	0	0	0.5	1	1	0.5	0.5	0.5	0.5
Activation	LR	LR	LR	LR	ReLU	LR	LR	LR	LR	LR	LR	LR
Test EER	0.2206	0.2428	0.24912	0.210	0.2138	0.2068	0.2378	0.2487	0.2075	0.2013	crash	0.2163
Eval. EER	22.07%	24.28%	24.91%	21.03%	x	20.68%	x	x	20.75%	20.14%	x	x

Table 4: Exhaustive tables of x-Vector results. LR = leaky ReLU. Data augmentation values indicate fraction of samples on which a augmentation is applied. (table 1/4)

File name model	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_7.py	pt_1_6.py	pt_1_6.py
Run ID	13	14	15	16	17	18	19	20	21	22	23	24
Batch size	128	128	128	128	128	128	128	128	128	128	128	128
Learning rate	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003
Number of epochs	40	40	40	30	30	40	40	30	30	30	40	30
Number of frames	48000	48000	48000	48000	48000	48000	48000	48000	48000	80000	48000	48000
Number of mfcc	100	40	40	40	40	40	40	40	80	40	40	40
Embedding size	128	64	192	512	512	128	128	128	128	128	128	512
Optimizer	Adam	Adam	Adam	Adam	SGD	SGD	Adam	Adam	Adam	Adam	Adam	Adam
Momentum					0.9	0.9						
Loss function	NLL	NLL	NLL	NLL	NLL	NLL	CEL	CEL	NLL	NLL	NLL	NLL
Gamma	0.8	0.8	0.8	1	1	0.8	0.8	1	1	1	1	1
Data augmentation	0	0	0	0	0	0	0	0	0	0	0	0
Activation	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR
Test EER	0.23688	0.25540	0.24570	0.2454	0.297	0.24542	0.2542	0.2630	0.246	0.271	0.2450	0.2606
Eval. EER	23.69%	25.54%	24.58%	24.54%	29.72%	24.54%	err	26.41%	24.63%	27.17%	24.50%	23.48%

Table 5: Continuation (table 2/4)

File name model	pt_1_6.py	pt_1_6.py	pt_1_6.py	pt_1_6.py	pt_1_6.py	pt_1_6.py	pt_1_6.py	pt_1_6.py	pt_1_6.py	pt_1_6.py	pt_1_6.py	pt_1_6.py
Run ID	25	26	27	28	29	30	31	32	33	34	35	36
Batch size	128	128	128	128	128	128	128	128	128	256	64	128
Learning rate	0,003	0,003	0,03	0,0003	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003
Number of epochs	30	30	30	30	30	30	40	30	30	30	30	30
Number of frames	80000	16000	48000	48000	48000	48000	80000	80000	48000	48000	48000	48000
Number of mfcc	40	40	40	40	80	120	40	80	80	40	40	40
Embedding size	512	512	512	512	512	512	128	128	128	512	128	64
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam	SGD	Adam	Adam	Adam
Momentum									0.9			
Loss function	NLL	NLL	NLL	NLL	NLL	NLL	NLL	NLL	NLL	NLL	NLL	NLL
Gamma	1	1	1	1	1	1	1	1	1	1	1	1
Data augmentation	0	0	0	0	0	0	0	0	0	0	0	0
Activation	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR
Test EER	0.255	0.29	0.26	0.28	0.248	0.25	0.242597	0.29	0.279	0.30	0.2574	0.2648
Eval. EER	23.59%	26.96%	25.80%	26.44%	22.73%	23.56%	x	29.76%	x	28.01%	25.74%	26.48%

Table 6: Continuation (table 3/4)

File name model	pt_1_6.py	pt_1_6.py	pt_1_8.py	pt_min3.py	pt_tryout_1.py	pt_tryout_1.py	pt_tryout_1.py	pt_tryout_1.py	pt_tryout_5.py
Run ID	37	38	39	40	41	42	43	44	45
Batch size	128	128	128	128	128	128	128	128	128
Learning rate	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003
Number of epochs	30	30	40	40	40	40	40	40	40
Number of frames	48000	48000	48000	48000	48000	48000	48000	48000	48000
Number of mfcc	40	40	40	40	40	100	100	40	40
Embedding size	128	128	128	128	128	128	128	128	128
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam
Momentum									
Loss function	CEL	NLL	NLL	NLL	NLL	NLL	NLL	NLL	NLL
Gamma	1	1	s	0.8	1	1	1	1	0.8
Data augmentation	0	0	0	0	1	1	0	0	0
Activation	LR	LR	LR	LR	LR	LR	LR	LR	LR
Test EER	0.2676	0.2398	0.2358	0.23574	0.2139	0.2198	0.2326	0.23151	0.238924
Eval. EER	x	20.82%	x	x	x	x	23.26%	23.15%	x

Table 7: Continuation (table 4/4)

A.2 VGG-M experimental results

Model	EER (%)
Default	26.58
embedding_size = 512	28.46
embedding_size = 512, learning_rate = $3e^{-2}$	28.02
embedding_size = 512, learning_rate = $3e^{-4}$	26.25
embedding_size = 512, batch_size = 256	27.07
SGD optimizer	24.72
Cross entropy loss function	27.12
Dropout layers, with $p = 0.5$	27.59
Data augmentation	**24.69**

Table 8: Experimental results for the VGG-M model. Only deviating values from the default settings are mentioned.

A.3 Small convolutional network experimental results

Model	EER (%)
Default	24.41
learning_rate = $3e^{-4}$, dropout, with $p = 0.5$, cross entropy loss function and data augmentation	22.18
learning_rate = $3e^{-4}$, cross entropy loss function and data augmentation	23.16
Cross entropy loss function	24.18
Cross entropy loss function and data augmentation	**21.77**
weight_decay = $1e^{-3}$, cross entropy loss function and data augmentation	22.71
SGD optimizer and cross entropy loss function	28.45
GELU activations, cross entropy loss function and data augmentation	24.14
Dropout, with $p = 0.5$, cross entropy loss function and data augmentation	22.98

Table 9: Experimental results for the small convolutional model. Only deviating values from the default settings are mentioned.