ASVtorch Toolkit: Speaker Verification with Deep Neural Networks

Kong Aik Lee[1,3], Ville Vestman[2], Tomi Kinnunen[2]

[1] *Institute for Infocomm Research, A*STAR, Singapore*
[2] *Computational Speech Group, University of Eastern Finland, Finland*
[3] *Biometrics Research Laboratories, NEC Corporation, Japan*
lee_kong_aik@i2r.a-star.edu.sg, vvestman@cs.uef.fi, tkinnu@cs.uef.fi

Abstract

The human voice differs substantially between individuals. This facilitates *automatic speaker verification* (ASV) — recognizing a person from his/her voice. ASV accuracy has substantially increased throughout the past decade due to recent advances in machine learning, particularly deep learning methods. An unfortunate downside has been substantially increased complexity of ASV systems. To help non-experts to kick-start reproducible ASV development, a state-of-the-art toolkit implementing various ASV pipelines and functionalities is required. To this end, we introduce a new open-source toolkit, ASVtorch, implemented in Python using the widely used PyTorch machine learning framework.

Keywords: Speaker recognition, PyTorch, Deep learning

1. Motivation and significance

Automatic speaker verification (ASV) systems [1] compare a pair of speech utterances (enrollment and test utterance) to decide whether or not the same speaker is present in the two. Modern ASV systems involve three broad tasks: (i) extraction of features from short segments of speech (frames); (ii) forming a fixed-dimensional vector representation (*speaker embedding*) per utterance; and (iii) comparison of the enrollment and test embeddings to assess the degree of speaker similarity.

To help non-experts to develop ASV systems and to make research results reproducible, a number of ASV bundle software has been introduced in the

past (Alize [2], Kaldi [3], Bob [4], Sidekit [5]). Further, multiple implementations focusing on speaker modeling using deep neural networks have been published recently [6, 7, 8, 9]. We introduce **ASVtorch** – an up-to-date ASV toolkit that leverages from GPU acceleration available in the PyTorch library [10]. ASVtorch provides state-of-the-art implementation of classic i-vector [11] based speaker recognition besides various deep learning methods [12, 13]. We provide training and evaluation recipes for the VoxCeleb [14, 15] and Speakers in the Wild (SITW) [16] datasets, both of which are widely used by ASV researchers.

Figure 1 shows the processing pipeline in modern ASV systems. The three main components are feature extractor, speaker embedding extractor, and scoring back-end. A brief description of these components are given below. We refer the interested reader to [17, 18, 19], and references therein, for further details.

Feature extraction. The function of a feature extractor is to produce a meaningful, compact representation of the input speech signal. The signal is first segmented into overlapped frames of 20 to 30 ms. From each frame, relevant features are extracted, for instance, mel-frequency cepstral coefficients (MFCCs) [20] in ASVtorch.

Speaker embedding is a fixed-dimensional representation of variable-length utterances. Utterances of the same speaker are close to each other in the embedding space. The idea is similar to *word* embeddings [21, 22] in natural language processing, but in our case for acoustic data. Popular examples are x-vector [12] and i-vector [11] embeddings.

Scoring back-end. Given a pair of enrollment and a test utterance embeddings, ϕ_e and ϕ_t, a similarity score is computed. It might be a simple cosine similarity, or a statistical back-end like *probabilistic linear discriminant analysis* (PLDA) [23, 24]. It is also common to pre-process the embeddings through dimension reduction with linear discriminant analysis (LDA) [25], whitening and length normalization [26]. ASVtorch implements both embedding pre-processing and PLDA based scoring.

2. Software Architecture

ASVtorch comprises many Python packages (see Figure 2) detailed in this Section. Figure 3 illustrates how the different components interact with each other.

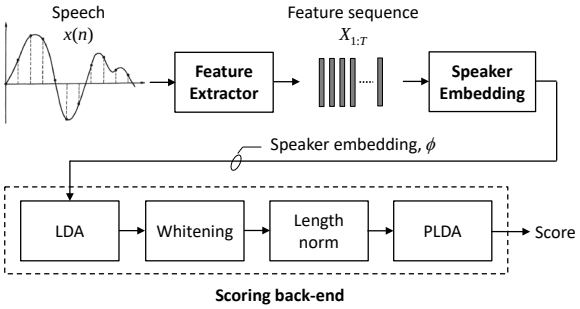

Figure 1: An example of a modern speaker recognition system consisting of a feature extraction front-end, speaker embedding, and a scoring back-end.

`frontend`

The front-end package contains Python wrappers for Kaldi's shell scripts [3] to extract MFCCs [20] and to perform voice activity detection (VAD) and data augmentation. The package contains a feature loader to load the features to NumPy arrays, to compute delta features, and to perform cepstral mean and variance normalization of the features.

`backend`

The back-end consists of an embedding processor, PLDA and score normalization. The embedding processor can be used to center, whiten, and length-normalize the speaker embeddings, as is the common practice.

`networks`

The network package contains all deep learning related functionalities, including many architectures for speaker embedding extraction, training loop for DNNs, and many custom neural network components. It also contains two data loaders; one to load short-term features during DNN training and the other to load features of full-length segments to perform validation and embedding extraction.

`ivector`

The i-vector package implements the i-vector pipeline. This includes the fast computation of GMM frame alignments and training of i-vector extractors, both of which utilize GPU acceleration detailed in [27]. It also contains necessary feature and Baum-Welch statistics data loaders needed in frame alignment computation, i-vector extraction and model training.

`settings`
The hyper-parameters of different experiments are defined in a configuration file managed with the `settings` package. This allows convenient parameter optimization without a need to modify the code between the experiments. Configuration files have a custom format that allows inheritance of settings from other experiments. This minimizes the need to copy-paste settings between different experiments.

`misc`
This package contains miscellaneous functions and classes of the toolkit. The most notable of these utilities are the ones specifying file and folder structure of the output folder.

`utterances`
This package contains classes for representing utterances (recordings) and lists of utterances. `Utterance` objects contain utterance and speaker identifiers as well as pointers to disk to stored features and frame alignments. In addition, the `Utterance` objects contain VAD labels that are loaded to RAM immediately when the utterance object is constructed. This reduces disk usage during later on an allows to quickly filter the utterances within a dataset based on utterance duration (after VAD).

`evaluation`
The `evaluation` package contains functions for scoring trial lists and for computing performance metrics, which include *equal error rate* (EER) and *minimum of detection cost function* (minDCF) [17]. The package also contains a class to plot *detection error tradeoff* (DET) curves [17].

`recipes/*`
The sub-packages of the `recipes` package contain main scripts and configurations to run specific ASV systems targeted for specific evaluations. These recipes are detailed in Section 4. The sub-package `detplot` contains an example of how to draw DET curve using the `evaluation` package.

3. Software Functionalities

3.1. Data loaders

As shown in Figure 3, different operations require different variants of loaders of acoustic features. Common to all data loaders is that they operate on multiple CPU cores in parallel while GPU is concurrently using the loaded

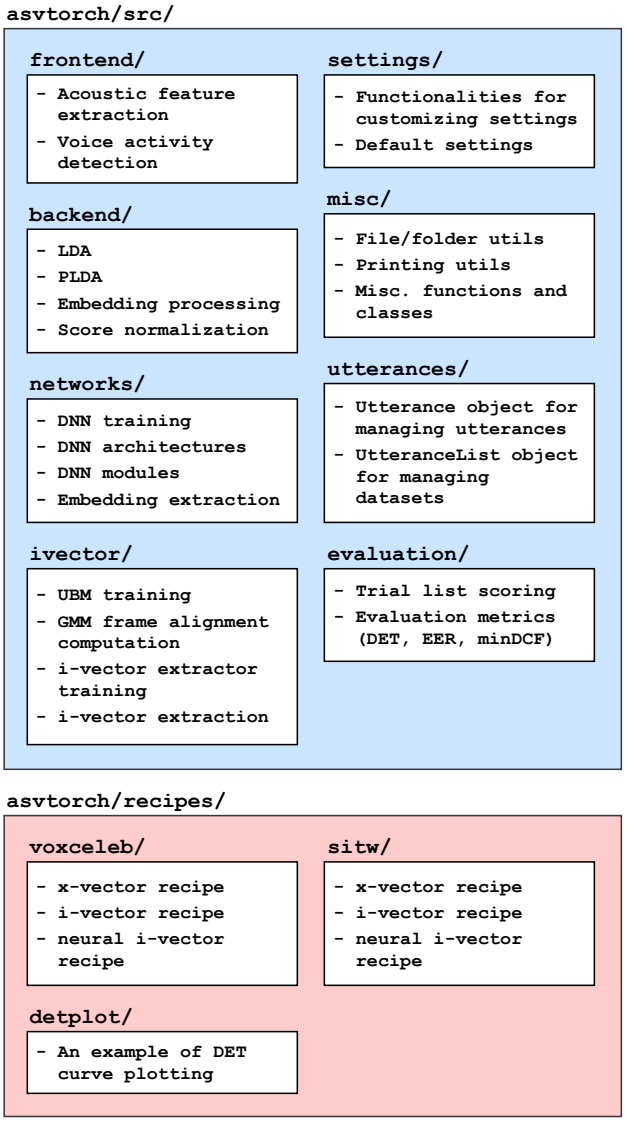

Figure 2: Python packages of the ASVtorch toolkit.

data to perform the primary computation work. This subsection describes the functionality of each data loader.

The data loader for DNN training crops training utterances within a minibatch to enforce the same duration before feeding the minibatch to the network. Although duration *within* a minibatch is fixed, it may vary *across*

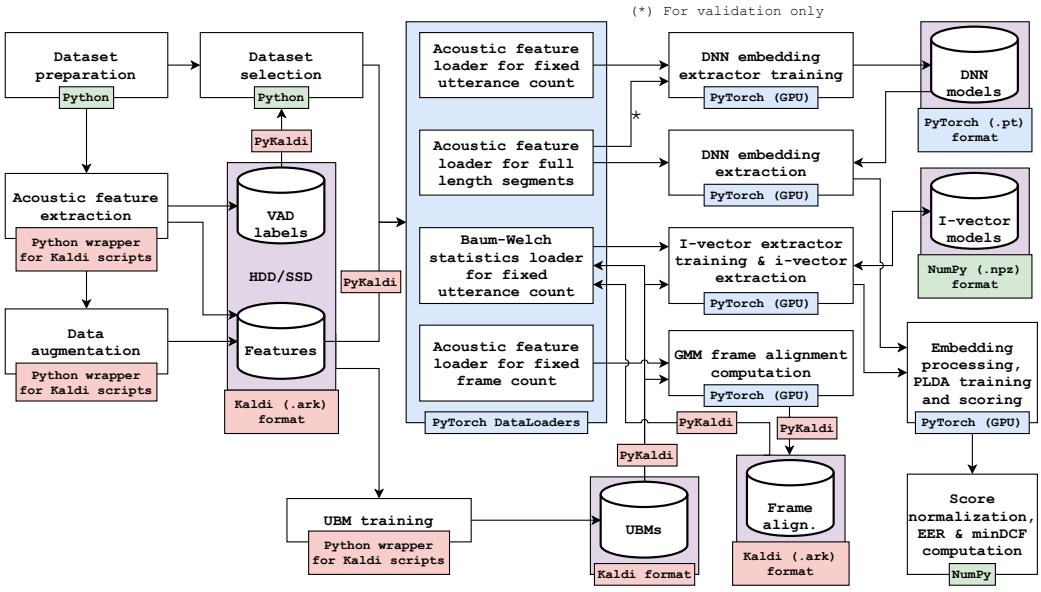

Figure 3: Computation pipelines of i-vector and DNN embedding systems in ASVtorch.

minibatches. Training minibatches for an epoch are created as follows:

1. For each training speaker, U utterances are chosen and cropped into fixed duration segments by choosing starting positions of the segments randomly.
2. Using the cropped utterances, minibatches of M utterances are created.

As a result of random sampling, no two epochs use exactly the same data.

For accelerated embedding extraction with GPU, we designed separate data loader that organizes enrollment and testing utterances into minibatches. The requirement of constant duration within a minibatch is handled by grouping utterances with similar durations into a minibatch by cropping them to the length of the shortest utterance. I-vector based systems also utilize two different kinds of data loaders. The first one loads acoustic feature vectors in batches of fixed frame count to be used in computation of frame alignments with respect to the UBM components. The second data loader loads acoustic features and frame alignments and turns them into Baum-Welch statistics used for i-vector extraction.

3.2. Progress monitoring of DNN training

In ASVtorch, the training process of deep embedding extractors is monitored in a number of ways. The training and validation losses as well as training and validation classification accuracies are periodically reported. Additionally, ASVtorch reports losses and accuracies on *full-duration* besides the cropped training segments. This allows monitoring potential issues caused by duration variation.

The above metrics are not often enough to reliably determine the quality of the speaker embeddings, as the losses and accuracies are computed from the output layer of the network, whereas the embeddings are extracted from one of the preceding layers. Thus, ASVtorch runs a scaled-down version of an ASV system with a PLDA backend after every Nth epoch to monitor the progress in terms of EER and minDCF metrics.

3.3. Learning rate scheduler for DNN training

In ASVtorch, the parameters of a DNN are optimized using the minibatch *stochastic gradient descent* (SGD) algorithm. In SGD, the magnitude of the updates to the network parameters is controlled by the *learning rate* parameter. In practice, it is beneficial to begin training with a high learning rate and decrease it as training progresses. This allows the network to converge fast at the beginning while reaching closer to minimum of the loss landscape at the final stages of training. To lessen the need of manual tuning of the learning rate schedule, we use a learning rate scheduler based on the training loss. The scheduler operates as follows: if the relative decrease in training loss between two consecutive epochs is less than a predefined percentage, the learning rate is halved. The training stops when the learning rate was halved twice in a row.

3.4. 1D-CNN as time-delay neural network

Two common choices to feed audio features into DNNs are either as (a) 2-D spectrograms, or (b) 2-D feature maps. Among the two options, 1-D convolution is more commonly used in speaker recognition. The reason is that the concept resembles the way sequences of feature vectors are processed as a time series. Coupled with dilation, the temporal context of the neural network can be expanded using the same number of weights. The temporal context becomes wider when moving deeper into the network. The units in the deeper layers are indirectly connected to wider input regions, as

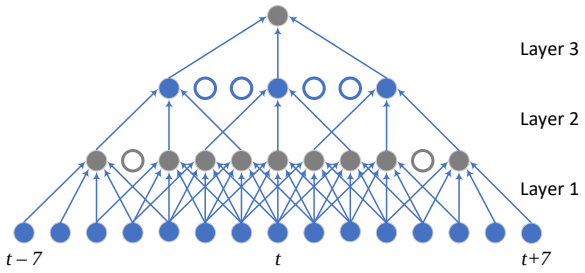

Figure 4: A stack of three-layer time-delay neural network (TDNN) produces a wide temporal context. The first layer constitutes a 1-D convolution with 1-by-5 filter and a dilation factor of one (i.e., no dilation). The second and third layers are implemented as 1-D convolution with 1-by-3 filter with dilation factor of two and three, respectively. The effective temporal context at the last layer is $7 \times 2 + 1 = 15$ frames.

illustrated in Figure 4. This type of neural network is known as a time-delay neural network (TDNN) [28]. It is extensively used in ASVtorch.

3.5. Temporal pooling layer

Speaker embedding uses temporal pooling of features to form a fixed-sized representation. This is implemented via an equal-weight averaging (or pooling) of transformed features across all time steps. A fundamental concern is to ensure that the gradient is properly back-propagated through the temporal pooling layer. The authors are unaware of such analysis being reported in earlier ASV studies. In specific, we are interested in the backward propagation of gradient through

$$\mathbf{a} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{a}_t, \tag{1}$$

where

$$\mathbf{a}_t = g\left(\mathbf{W}\mathbf{f}_t + \mathbf{b}_t\right) \tag{2}$$

is the transformed feature vector at the t-th time step. Here, $\mathbf{W}$, $\mathbf{b}$, and $\mathbf{f}_t$ are the weight matrix, bias vector, and input to a feed-forward layer followed by a non-linear activation function g. This could be a rectified linear unit (ReLU), leaky ReLU, or some other activation function.

Let $\frac{\partial \mathcal{L}}{\partial \mathbf{a}}$ be the gradient at the output of the temporal pooling layer, where $\mathcal{L}$ denotes the loss function (typically, a multi-class cross-entropy loss) used to optimize the network. Propagating the gradient backward through the

averaging operation essentially divides the gradient to T equal parts such that

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}_t} = \frac{1}{T} \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{a}} \quad \text{for} \quad t = 1, 2, \ldots, T. \tag{3}$$

Since the same set of weights $\mathbf{W}$ is used to process the features $\mathbf{f}_t$, for $t = 1, 2, ..., T$, the gradients at all the T time steps are summed, as follows

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{a}_t} \cdot \frac{\partial \mathbf{a}_t}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}} \cdot \frac{1}{T} \sum_{t=1}^{T} \frac{\partial \mathbf{a}_t}{\partial \mathbf{W}} \tag{4}$$

From (3) and (4), we see that the net effect of gradient back-propagating through the temporal pooling layer is the smearing of gradients to frames and then summed to update the weight matrix and bias vector. From (4), it is also clear that weighted average could be used to give more *attention* to certain frames deem to be more important for speaker recognition task. We refer interested readers to [29, 30] for further details.

4. Illustrative examples

As shown in Figure 2, ASVtorch comes with two ready made pipelines (or recipes) to develop ASV systems for widely adopted VoxCeleb [14, 15] and *Speakers in the Wild* [16] (SITW) datasets. Both evaluation recipes contain i-vector, x-vector, and neural i-vector variants. The last was introduced in [13] to combine ideas of discriminative and generative embeddings.

The VoxCeleb1 and VoxCeleb2 datasets used for VoxCeleb evaluation were collected from YouTube by the authors of [14] and [15]. VoxCeleb1 consists of more than 150 000 utterances from 1251 speakers and VoxCeleb2 consists of over 1.1M utterances from 6112 speakers. The average utterance duration is about eight seconds. In the VoxCeleb repice, we train the ASV systems using VoxCeleb2 dataset, whereas VoxCeleb1 is used for testing. For training, the utterances originating from the same YouTube video are concatenated together. Testing is done using three trial lists introduced in [14] and [15]:

- **VoxCeleb1-O** — The original VoxCeleb1 trial list consisting of 37 720 trials from 40 speakers.
- **VoxCeleb1-E** — A trial list comprising 581 480 trials from the entire set of 1251 VoxCeleb1 speakers.

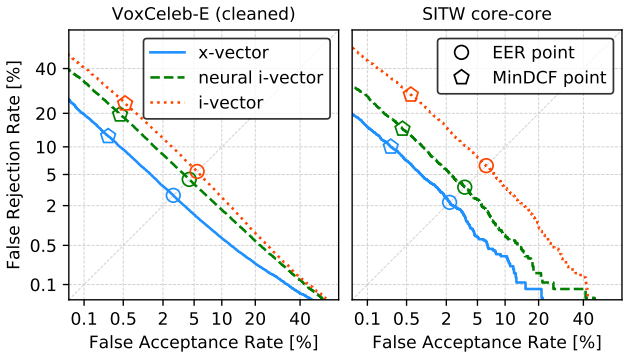

Figure 5: Detection error tradeoff curves for x-vector, neural i-vector, and i-vector systems evaluated on VoxCeleb-E (cleaned) [31] and SITW core-core trial lists.

- **VoxCeleb1-H** — A harder version of the VoxCeleb1-E trial list comprising 552 536 same gender and same nationality trials.

In addition, the results are shown for the cleaned versions of the above lists introduced in [31].

The pipelines for SITW are very similar to VoxCeleb recipes. The difference is that SITW recipes use both VoxCeleb1 and VoxCeleb2 for training. The results are computed for two trial lists:

- **SITW core-core** — A trial list comprising 721 788 trials. Each utterance contains speech from one speaker only.

- **SITW core-multi** — A trial list of 2 010 683 trials. Test utterances contain speech from one or more speakers.

As the recipes are under continuous development, we refrain from reporting the exact results from the recipes as they are expected to change over time. The current results can be found from the code repository of ASVtorch. Figure 5 shows the DET plots of the different ASV systems under different evaluations at the time of writing. The DET plots were drawn using functionalities in the ASVtorch toolkit.

5. Impact

The complexity of state-of-the-art ASV systems has increased throughout the past decades. This increased complexity concerns both the required level of expertise required to craft such systems, the amount and variety in

speech data required to develop (and evaluate) the systems, and the number of model parameters. Crafting usable ASV systems used to be special activity reserved for experts in audio processing. This often involved combining and interfacing different scripts and tools across programming languages or computer environments; and part of the 'art' (as in 'state-of-the-art') was about being aware of hidden implementation details (not always transparent in publications), and spending substantial time to design and clean-up file-lists. ASVtorch provides functionalities and recipes aimed at lowering the barrier for non-experts, especially those from other disciplines and industries, to quickly kickstart building ASV systems.

Whereas it took relatively long time for deep learning to outperform classic modeling approaches in ASV, deep speaker embeddings are now considered the state-of-the-art and are under active studies by many research groups. While providing the state-of-the-art components, ASVtorch also implements *accelerated variants of classic methods*, like i-vector. This enables systematic comparison of new and existing algorithms on the same platform, and is consistent with our aim to promote reproducible research. In the example, we showed how to use ASVtorch in to train, test, and evaluate a speaker verification system. Users can use the toolkit as-is, make modification on top of current functionalities — or to even build commercial ASV systems.

6. Conclusions

We introduced the ASVtorch toolkit for automatic speaker verification (ASV) consisting of functionalities ranging from feature extraction to speaker embedding and scoring. These functionalities have been carefully crafted, fine-tuned, and tested on large-scale ASV tasks. Constructing a complete ASV pipeline is always a major undertaking as it involves substantial domain knowledge. Our aim is to make the ASVtorch toolkit available to a wide audience, especially those from other fields and industry, and to encourage reproducible ASV research. While providing a complete ASV pipeline, the toolkit can be used independently following library-like design. It allows a flexible, robust way to trial different ASV methods. Additionally, we believe that various functionalities provided in the toolkit are applicable for other audio, speech, and time series processing tasks, though the efficacy is yet to be tested.

7. Conflict of Interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Acknowledgements

This work was partially supported by Academy of Finland (project #309629) and by the Doctoral Programme in Science, Technology, and Computing (SCITECO) of the University of Eastern Finland (UEF). The authors at UEF were also supported by NVIDIA Corporation with the donation of Titan V GPU.

References

[1] D. A. Reynolds, T. F. Quatieri, R. B. Dunn, Speaker verification using adapted gaussian mixture models, Digital Signal Processing 10 (2000) 19 – 41. URL: http://www.sciencedirect.com/science/article/pii/S1051200499903615. doi:https://doi.org/10.1006/dspr.1999.0361.

[2] J.-F. Bonastre, F. Wils, S. Meignier, ALIZE, a free toolkit for speaker recognition, in: Proceedings.(ICASSP'05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005., volume 1, IEEE, 2005, pp. I–737.

[3] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, et al., The Kaldi speech recognition toolkit, in: IEEE 2011 workshop on automatic speech recognition and understanding, IEEE Signal Processing Society, 2011.

[4] A. Anjos, L. El-Shafey, R. Wallace, M. Günther, C. McCool, S. Marcel, Bob: a free signal processing and machine learning toolbox for researchers, in: Proceedings of the 20th ACM international conference on Multimedia, 2012, pp. 1449–1452.

[5] A. Larcher, K. A. Lee, S. Meignier, An extensible speaker identification sidekit in python, in: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2016, pp. 5095–5099.

[6] H. Zeinali, L. Burget, J. Rohdin, T. Stafylakis, J. H. Cernocky, How to improve your speaker embeddings extractor in generic toolkits, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2019, pp. 6141–6145. URL: https://github.com/hsn-zeinali/x-vector-kaldi-tf.

[7] Y. Liu, L. He, J. Liu, Large margin softmax loss for speaker verification, in: Proc. INTERSPEECH, 2019. URL: https://github.com/mycrazycracy/tf-kaldi-speaker.

[8] ASV-subtools, https://github.com/Snowdar/asv-subtools, 2020.

[9] J. S. Chung, J. Huh, S. Mun, M. Lee, H. S. Heo, S. Choe, C. Ham, S. Jung, B.-J. Lee, I. Han, In defence of metric learning for speaker recognition, arXiv preprint arXiv:2003.11982 (2020). URL: https://github.com/clovaai/voxceleb_trainer.

[10] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, 2019, pp. 8024–8035.

[11] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, P. Ouellet, Front end factor analysis for speaker verification, IEEE Transactions on Audio, Speech and Language Processing (2010).

[12] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, S. Khudanpur, X-vectors: Robust DNN embeddings for speaker recognition, in: Proc. ICASSP, 2018, pp. 5329–5333.

[13] V. Vestman, K. A. Lee, T. Kinnunen, Neural i-vectors, in: Proc. Odyssey 2020 The Speaker and Language Recognition Workshop, 2020, pp. 67–74. URL: http://dx.doi.org/10.21437/Odyssey.2020-10. doi:10.21437/Odyssey.2020-10.

[14] A. Nagrani, J. S. Chung, A. Zisserman, VoxCeleb: A large-scale speaker identification dataset, Proc. Interspeech 2017 (2017) 2616–2620.

[15] J. S. Chung, A. Nagrani, A. Zisserman, VoxCeleb2: deep speaker recognition, in: Proc. Interspeech 2018, 2018, pp. 1086–1090. URL: http://dx.doi.org/10.21437/Interspeech.2018-1929. doi:10.21437/Interspeech.2018-1929.

[16] M. McLaren, L. Ferrer, D. Castan, A. Lawson, The speakers in the wild (SITW) speaker recognition database., in: Proc. Interspeech 2016, 2016, pp. 818–822.

[17] T. Kinnunen, H. Li, An overview of text-independent speaker recognition: from features to supervectors, Speech Communication 52 (2010) 12–40.

[18] J. H. L. Hansen, T. Hasan, Speaker recognition by machines and humans: a tutorial review, IEEE Signal Processing Magazine 32 (2015) 74 – 99.

[19] K. A. Lee, O. Sadjadi, H. Li, D. Reynolds, Two decades into speaker recognition evaluation - are we there yet?, Computer Speech & Language 61 (2020) 101058.

[20] S. Davis, P. Mermelstein, Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences, IEEE Transactions on Acoustics, Speech, and Signal Processing 28 (1980) 357–366. doi:10.1109/TASSP.1980.1163420.

[21] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: Advances in Neural Information Processing Systems, 2013, pp. 3111–3119.

[22] Y. Bengio, R. Ducharme, P. Vincent, A neural probabilistic language model, in: Advances in Neural Information Processing Systems, 2000, pp. 932–938.

[23] S. Ioffe, Probabilistic linear discriminant analysis, in: roceedings of the 9th European Conference on Computer Vision, 2006.

[24] S. J. D. Prince, J. H. Elder, Probabilistic linear discriminant analysis for inferences about identity, in: IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007, 2007, pp. 1–8.

[25] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer-Verlag, Berlin, Heidelberg, 2006.

[26] D. Garcia-Romero, C. Y. Espy-Wilson, Analysis of i-vector length normalization in speaker recognition systems, in: Proc. Interspeech 2011, 2011.

[27] V. Vestman, K. A. Lee, T. H. Kinnunen, T. Koshinaka, Unleashing the Unused Potential of i-Vectors Enabled by GPU Acceleration, in: Proc. Interspeech 2019, 2019, pp. 351–355. URL: http://dx.doi.org/10.21437/Interspeech.2019-1955. doi:10.21437/Interspeech.2019-1955.

[28] V. Peddinti, D. Povey, S. Khudanpur, A time delay neural network architecture for efficient modeling of long temporal contexts, in: Proc. Interspeech, 2015, pp. 3214–3218.

[29] K. Okabe, T. Koshinaka, K. Shinoda, Attentive statistics pooling for deep speaker embedding, in: Proc. Interspeech, 2018, pp. 2252–2256.

[30] Y. Zhu, T. Ko, D. Snyder, B. Mak, D. Povey, Self-attentive speaker embeddings for text-independent speaker verification, in: Proc. Interspeech 2018, 2018, pp. 3573–3577.

[31] W. Xie, A. Nagrani, J. S. Chung, A. Zisserman, Utterance-level aggregation for speaker recognition in the wild, in: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2019, pp. 5791–5795.