Review of different robust x-vector extractors for speaker verification

Mickael Rouvier
LIA - Avignon University
Avignon, France
mickael.rouvier@univ-avignon.fr

Richard Dufour
LIA - Avignon University
Avignon, France
richard.dufour@univ-avignon.fr

Pierre-Michel Bousquet
LIA - Avignon University
Avignon, France
pierre-michel.bousquet@univ-avignon.fr

***Abstract*—Recently, the x-vector framework, extracted with deep neural network architectures, became the state-of-the-art method for speaker verification. Although another level of performance has been overcome with this approach, fine-tuning and optimizing the hyper-parameters of a deep neural network to obtain a robust x-vector extractor is cost- and time-consuming. Several approaches have been proposed to train robust x-vector extractors. In this paper, we propose to review and analyse the impact of the most significant x-vector related approaches, including variations in terms of data augmentation, number of epochs, size of mini-batch, acoustic features and frames per iteration. By applying these approaches to the default recipe provided in the Kaldi toolkit, we observed a significant relative gain of more than 50% in terms of EER on Speaker in the Wild and Voxceleb1-E datasets.**

***Index Terms*—x-vector, deep neural network, speaker verification**

I. Introduction

Speaker recognition refers to the task of verifying the identity claimed by a speaker from that person's voice [1]. It has been shown useful, for example, in logical and physical access control, forensics [2] and speaker diarization [3].

Since many years, the i-vector/PLDA framework has been the state-of-the-art in text-independent Speaker Verification [4]. The i-vector approach provides an elegant way for extracting a small-dimensional feature vector from a variable length speech signal that preserves the speaker-specific information. Then, a Probabilistic Linear Discriminant Analysis (PLDA) is used to verify whether two i-vectors (and more generally two vectors) correspond to the same speaker or not [5]–[7].

Recently, novel deep learning approaches, which outperform the traditional i-vector framework, have emerged. In [8], the authors propose to learn high-level speaker identity features with deep neural network (DNN) models through speaker identification task, *i.e.* by classifying speech segments into one of n speaker identities. In that context, the different layers of the DNN are trained to extract information relevant for discriminating between different speakers. The main idea is that an hidden layer (layer *segment6* in [8]) is considered as the speech segment feature vector, called here an x-vector.

Several methods have been proposed to build x-vector systems [9]–[12], as well as different deep neural architectures [13], [14], and data augmentation strategies [15]–[17]. In this article, we propose to review the most significant approaches and empirically evaluate the effectiveness of each topology and variations compared to the standard widely used Kaldi version [8]. In this article, we chose to call *tricks* the variations in the x-vector extractor that do not challenge the architecture of the system, since it integrates optimization of the parameters, fine-tuning, or method variants (i.e. a different type of data augmentation).

Our experiments on the Speaker In The Wild (SITW) and Voxceleb1-E corpus based on the standard Kaldi x-vector/PLDA approach obtains respectively 4.37% and 3.65% Equal Error Rate (EER), whereas the new neural network architecture, including several optimization tricks, obtains respectively 2.01% and 1.79% EER. A relative gain of more than 50% has then been observed in terms of EER, which confirms the need to focus on the different variants of x-vector approaches, including a fine-tune and an optimization of the different parameters of DNN architectures to a targeted task. This article will then focus on the detailed review of each of the different approaches and optimized parameters, and their potential gains expected at each stage.

After presenting the standard x-vector Kaldi recipe in Section II, the new topology and tricks are presented in Section III. The corpus on which experiments are carried as well as the detailed results of our experiments are presented in Sections IV and V respectively. A conclusion is finally provided in Section VI.

II. Standard Kaldi recipes

The x-vector method is based on the system described in [8]. A speaker discriminative DNN is trained to produce speaker embeddings, called x-vectors.

The DNN uses 30-dimensional MFCC features as input, extracted from 25ms audio signal, mean-normalized over a sliding window of up to 3 seconds. Unvoiced frames are filtered out from the utterances using a Voice Activity Detection (VAD) based on signal energy.

The DNN architecture used in the x-vector system is described in Table I. In Figure 1, we observed that the DNN is composed of three components: *frame-level*, *statistics-level*, and *segment-level* components.

The *frame-level* component is composed of the first five layers (layer 1 to 5). The layers are constructed with a Time-

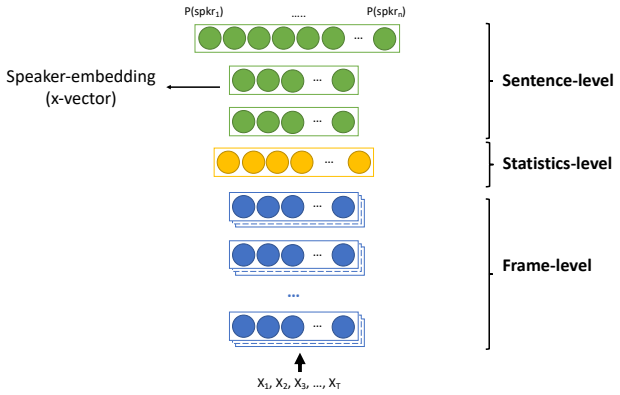

Fig. 1. Architecture of a standard x-vector extractor.

Delay Neural Network (TDNN). Suppose t is the current time step. Frames from $(t-2)$ to $(t+2)$ are spliced together at the input layer (layer 1). The next two layers splice the output of the previous layer at time steps $t-2, t, t+2$ (layer 2) and $t-3, t, t+3$ (layer 3), respectively. No temporal contexts are added to the fourth and fifth layers. Thus, the total temporal context after the third layer is 15 frames.

The *statistics-level* component is an essential component that converts from a variable length speech signal into a single fixed-dimensional vector. The statistics-level is composed of one layer: the statistics-pooling, which aggregates over frame-level output vectors of the DNN (layer 5) and computes their mean and standard deviation.

The *segment-level* component maps the segment-level vector to speaker identities. The mean and standard deviation are concatenated together and forward to two additional hidden layers (layers 7 and 8), and finally a softmax output layer (layer 9).

TABLE I
TOPOLOGY OF THE STANDARD x-VECTOR KALDI DNN ARCHITECTURE [8], INCLUDING A TDNN IN THE FIRST LAYERS.

Layer	Layer type	Context	Size
1	TDNN-ReLU	t-2:t+2	512
2	TDNN-ReLU	t-2, t, t+2	512
3	TDNN-ReLU	t-3, t, t+3	512
4	Dense-ReLU	t	512
5	Dense-ReLU	t	1500
6	Pooling (mean+stddev)	t	3000
7	Dense(Embedding)-ReLU	t	512
8	Dense-ReLU	t	512
9	Dense-Softmax	t	Nb spks

The DNN is trained to classify speakers contained in the training set. The layer 6 is chosen as the speaker embedding (*i.e.* the x-vector), which forces the information brought from the preceding layer into a low-dimensional representation.

All neural units are Rectified Linear Uniters (ReLUs). The DNN is trained with 3 epochs using mini-batch of size 64.

In order to increase the diversity of the acoustic conditions in the training set, a data augmentation strategy is used, that adds two corrupted copies of the original recordings to the training list. The recordings are corrupted by adding noise, music and mixed speech (babble) drawn from the MUSAN database [18] and adding reverberation by using simulated Room Impulse Responses (RIR).

III. EXTENDED TOPOLOGY AND TRICKS

This section describes several tricks for training a robust x-vector extractor. As a reminder, a trick can be seen as a different configuration or a non-architectural modification to the x-vector extractor. We note that we focus only on neural network training part of the x-vector pipeline.

A. Topology: Extended-TDNN

In [14], the authors proposed an extended version of the TDNN (called Extended-TDNN or E-TDNN). Table II summarizes the Extended-TDNN architecture. The two main differences are a slightly wider temporal context of the TDNN (due to the addition of layer 7) and interleaving dense layers between the TDNN layers (equivalent to the 1×1 convolution used in computer vision architectures). In addition to this topology, we proposed to increase the number of neurons (referred by K in Table II) only for the frame-level and statistics-pooling.

TABLE II
TOPOLOGY OF THE EXTENDED-TDNN x-VECTOR ARCHITECTURE [14].

Layer	Layer type	Context	Size
1	TDNN-ReLU	t-2:t+2	K
2	Dense-ReLU	t	K
3	TDNN-ReLU	t-2, t, t+2	K
4	Dense-ReLU	t	K
5	TDNN-ReLU	t-3, t, t+3	K
6	Dense-ReLU	t	K
7	TDNN-ReLU	t-4, t, t+4	K
8	Dense-ReLU	t	K
9	Dense-ReLU	t	3*K
10	Pooling (mean+stddev)	t	6*K
11	Dense(Embedding)-ReLU	t	512
12	Dense-ReLU	t	512
13	Dense-Softmax	t	Nb spks

B. Tricks: Data augmentation, number of epochs and size of mini-batch

Data augmentation: The strategy of data augmentation consists of corrupting recordings by adding noise, music and mixed speech drawn from the MUSAN database [18] and adding reverberation by using simulated RIR. Data augmentation is a crucial step to increase the amount and the diversity of the training data. In [16], the authors propose a much more aggressive data augmentation than the classical recipe. The authors propose not to limit the data augmentation to two corrupted copies of the original recordings but to use all the corrupted copies. Even if the learning time is increased, using all the corrupted versions can improve the amount of diversity of data without saturating the neural network.

Number of epochs and size of mini-batch: In the standard Kaldi recipe, the number of epochs is set to 3 and each mini-batch is composed of 64 examples. For speaker recognition

tasks, the authors in [16], [19] observed that the augmentation of the number of epochs and of the minibatch size improve the overall performance. Then, they propose to set the number of epochs to 9 while, in the meantime, each minibatch must be composed of 128 examples.

C. Tricks: Acoustic features and normalization

Normalization: Classically, the input acoustic features are mean-normalized over a sliding window of up to 3 seconds. We propose to mean-normalize over the entire segment (per utterance). The mean idea is to normalize more finely over the entire segment than what was performed on a 3-seconds length.

Acoustic features: Classically, MFCCs are used as input acoustic features. But while DNNs can handle highly correlated features, it is easy to think of a direct use in DNNs of log Filter-bank energies instead of MFCCs. The role of Discrete Cosine Transform (DCT) in MFCC extraction is to de-correlate features. This is important for Gaussian Mixture Model (GMM) modeling but it becomes unnecessary in DNNs. Moreover, DCT reduces the feature dimension and this may lead to information loss. In this experiment, we propose to compare Filter-bank features to MFCCs ones.

D. Tricks: Fine-tuning with multiple GPUs

In order to train a neural network using multiple GPUs, Kaldi propose to learn neural networks following the fork-join method. It means that each GPU is trained separately with randomized subsets of the training data. Once the subsets for all GPUs are trained, the models are averaged across all the GPU. This new model is re-distributed for the other GPU. This process is repeated until all the data are processed and for a specified number of epochs. The size of subsets is fixed by a number of samples K (this parameter is called *frames per iteration*). The larger this size is, the less averaged are the models (and vice-versa). We propose to vary this parameter in order to see the impact of the frames per iteration on the performance.

IV. Experimental protocol

This section describes the experimental setup in terms of dataset and evaluation protocol.

A. Training and Evaluation datasets

The x-vector extractors, and their variations, are trained on the VoxCeleb2 dataset [20], only on the development partition, which contains speech from 5,994 speakers with 16 Khz sampling rate. The trained extractors are evaluated on Speakers in the Wild (SITW) core-core task [21] and Voxceleb1-E Cleaned [22] dataset with 16 KHz sampling rate.

Note that the development set of VoxCeleb2 is completely disjoint from the VoxCeleb1 dataset (*i.e.* no speakers in common).

B. Performance criterion

Equal Error Rate (EER) and Detection Cost Function (DCF) are used as the performance criterion of speaker verification. EER is the threshold value such that false acceptance rate and miss rate are equals. DCF is defined as a weighted sum:

$$C_{det} = C_{Miss} \times P_{Miss|Target} \times P_{Target} + C_{FalseAlarm} \times P_{FalseAlarm|NonTarget} \times P_{NonTarget} \quad (1)$$

with the prior probabilities P_{Target} and $P_{NonTarget} = 1 - P_{Target}$ of target and impostor speakers, respectively. The relative costs of detection errors in this function are the costs of miss C_{Miss} and false alarm errors $C_{FalseAlarm}$. These parameters were set as follows: $P_{Target} = 0.01$, $C_{Miss} = 1$ and $C_{FalseAlarm} = 1$.

V. Experiments and results

In this section, we report the results obtained from the review of different x-vector extractors architectures as well as different configurations (*tricks*). Results are compared to the standard Kaldi recipe [8].

A. Tricks: Data augmentation, number of epochs and size of mini-batch

Table III shows the results concerning the tricks about data augmentation (using four corrupted copies of the original recordings), number of epochs (set to 9), and number of examples in mini-batch (set to 128). The *Baseline* system is the system obtained by the standard Kaldi recipe (*i.e.* using two corrupted copies of the original recordings, number of epochs is set to 3 and number of examples in mini-batch is set to 64). We can see that the baseline x-vector extractor obtains a performance of 3.65% and 4.37% in terms of EER, on SITW and VoxCeleb1-E respectively. We observe that increasing the data-augmentation, mini-batch and number of epochs considerably enhances the results and allows to reach a performance of 3.61% and 3.03% in terms of EER, on SITW and VoxCeleb1-E datasets respectively. This error decrease is also observed in terms of DCF, where results outperform the baseline Kaldi recipe. We note that increasing the number of epochs (set to 12) and of examples in mini-batch (set to 256) does not improve the results.

TABLE III
Comparison of performance when increasing number of epochs and size of mini-batch, and using a larger data augmentation.

	SITW$_{core-core}$		**Voxceleb1-E**$_{Cleaned}$	
	EER	DCF^{-10}	EER	DCF^{-10}
Baseline (Kaldi)	4.37	0.409	3.65	0.400
+ Epoch	4.01	0.381	3.25	0.362
+ Mini-batch	3.80	0.379	3.23	0.365
+ Data-augmentation	3.61	0.353	3.03	0.342

B. Topology: Extended-TDNN

Table IV shows the results concerning the Extended-TDNN architecture (E-TDNN). We remind that K refers to the number of neurons in the frame-level and statistic-pooling. Note that the first line (TDNN) refers to the best results previously obtained in Subsection V-A (baseline + epoch + mini-batch + data augmentation).

First, we compare TDNN and E-TDNN$_{(K=512)}$ architectures, because these two architectures have the same number of neurons per layer in the frame-level and statistics-pooling. We observe that E-TDNN$_{(K=512)}$ outperforms the TDNN architecture. The TDNN architecture obtains 3.61% and 3.65% of EER on SITW and VoxCeleb1-E respectively, whereas E-TDNN$_{(K=512)}$ architecture obtains 3.23% and 2.77% of EER on SITW and VoxCeleb1-E respectively. This can be explained by the fact that E-TDNN is deeper and has a slightly wider temporal context. Same tendencies are observed with the DCT metric.

Moreover, we observe that E-TDNN$_{(K=1536)}$ obtains the best results. The E-TDNN$_{(K=1536)}$ architecture reached 2.68% and 2.32% of EER on SITW and VoxCeleb1-E respectively. Unfortunately, this kind of models requests more than 14GB of memory in GPU cards. In our experiments, this model with 2 Tesla P100 GPUs takes 18 days to train. So for the rest of the experiments we will limit the experiments to the E-TDNN$_{(K=1024)}$ architecture.

TABLE IV
COMPARISON OF TDNN AND E-TDNN NEURAL NETWORK ARCHITECTURES FOR x-VECTOR EXTRACTION.

	SITW$_{core-core}$		**Voxceleb1-E**$_{Cleaned}$	
	EER	DCF^{-10}	EER	DCF^{-10}
TDNN (subsec. V-A)	3.61	0.353	3.03	0.342
E-TDNN$_{(K=512)}$	3.23	0.322	2.77	0.319
E-TDNN$_{(K=1024)}$	2.84	0.287	2.45	0.289
E-TDNN$_{(K=1536)}$	2.68	0.269	2.32	0.282

C. Tricks: Acoustic features and normalization

Normalization: Table V compares the system where the input acoustic features are mean-normalized over a sliding window of up to 3 seconds (MFCC$_{sliding-window}$) and the system where the input acoustic features are mean-normalized over the entire segment (MFCC$_{per-utterance}$). Note that MFCC$_{sliding-window}$ corresponds to E-TDNN$_{(K=1024)}$ (see Subsection V-B), which is the baseline in this subsection.

We observed for the SITW corpus that the MFCC$_{per-utterance}$ system obtains the best results, with an EER of 2.84%. On the other side, we observed for the Voxceleb1-E Cleaned corpus that the results are exactly the same for MFCC$_{sliding-window}$ and MFCC$_{per-utterance}$ systems (2.45% of EER). Indeed the training data is similar to the testing data (Voxceleb), and we think that DNNs have been trained to do a normalization in the DNN.

Acoustic features: Table VI gives a comparison between an x-vector extractor that uses MFCCs in input and a system that uses Filter-bank features. For the Filter-bank features, we vary the dimensional acoustic features. We observed that the system that uses Filter-bank features as input obtains better results than the one with MFCC features. The Filter-Bank$_{(Dim=60)}$ architecture obtained the best results in terms of EER, with 2.51% and 2.28% on Voxceleb1-E and SITW respectively.

TABLE V
COMPARISON OF PERFORMANCE USING SLIDING NORMALIZATION AND MEAN NORMALIZATION.

	SITW$_{core-core}$		**Voxceleb1-E**$_{Cleaned}$	
	EER	DCF^{-10}	EER	DCF^{-10}
MFCC$_{sliding-window}$	2.84	0.287	2.45	0.289
MFCC$_{per-utterance}$	2.76	0.283	2.45	0.289

TABLE VI
COMPARISON OF PERFORMANCE USING MFCCS AND FILTER-BANK AS INPUT FEATURES.

	SITW$_{core-core}$		**Voxceleb1-E**$_{Cleaned}$	
	EER	DCF^{-10}	EER	DCF^{-10}
MFCC	2.76	0.283	2.45	0.289
Filter-Bank$_{(Dim=40)}$	2.57	0.264	2.29	0.269
Filter-Bank$_{(Dim=60)}$	2.51	0.271	2.28	0.261
Filter-Bank$_{(Dim=80)}$	2.57	0.267	2.32	0.273

D. Tricks : Fined-tuning with multiple GPUs

Table VII presents the results obtained using different sizes of frames per iteration. In the default Kaldi recipe, the number of frames per iteration is fixed to 1.000.000.000. The system obtains for the SITW and Voxceleb1-E corpus an EER of 2.51% and 2.28% respectively. By lowering this parameter, we observe that the performances increase. For the SITW corpus the best results are obtained with the Frames per iter.$_{250.000.000}$ system (2.02% in terms of EER). And for Voxceleb1-E corpus, the best results are obtained with the Frames per iter.$_{200.000.000}$ system (1.79% in terms of EER). We can conclude that, in the Kaldi recipe, the number of frames per iteration is clearly too high. And during the training of DNNs, each GPU obtained very different models. The average of all the models across all the GPUs did not allow to converge towards a good model. Thus, Lower the parameter of frames per iteration allows to obtain more similar models along the iterations and, therefore, to converge towards a better model in a speaker verification task.

TABLE VII
COMPARISON OF PERFORMANCE USING DIFFERENT SIZES OF FRAMES PER ITERATIONS.

	SITW$_{core-core}$		**Voxceleb1-E**$_{Cleaned}$	
	EER	DCF^{-10}	EER	DCF^{-10}
Frames per iter.$_{1.000.000.000}$	2.51	0.271	2.28	0.261
Frames per iter.$_{350.000.000}$	2.32	0.244	1.87	0.213
Frames per iter.$_{300.000.000}$	2.13	0.240	1.83	0.212
Frames per iter.$_{250.000.000}$	2.02	0.238	1.82	0.210
Frames per iter.$_{200.000.000}$	2.08	0.233	1.79	0.206

E. Contributions for the different tricks

Table VIII summarizes the contributions of the different tricks. The baseline Kaldi system (standard recipe) obtains 4.37% and 3.65% in terms of EER for SITW and Voxceleb1-E respectively. The successive addition of the different tricks allows us to significantly improve the system baseline, with a final EER of 2.02% and 1.82% for SITW and Voxceleb1-E datasets respectively .

TABLE VIII
SUMMARY OF THE DIFFERENT x-VECTOR EXTRACTORS AND CONFIGURATIONS (*tricks*).

	SITW$_{core-core}$		**Voxceleb1-E**$_{Cleaned}$	
	EER	DCF^{-10}	EER	DCF^{-10}
Baseline	4.37	0.409	3.65	0.400
+ Epoch, minibatch...	3.61	0.353	3.03	0.342
+ E-TDNN	2.84	0.287	2.45	0.289
+ Filter-bank, norm.	2.51	0.271	2.28	0.261
+ Frames per iter	2.02	0.238	1.82	0.210

VI. CONCLUSION

In this paper, we presented and evaluated different DNN architectures and "tricks" in order to train a robust x-vector extractor for speaker verification tasks. We focused our review on different aspects, including architectural ones, with TDNN and E-TDNN, and aspects related to the configuration of DNNs (number of epochs, size of mini-batch, acoustic features, and sizes of frames per iteration).

Experiments performed on the SITW and Voxceleb1-E dataset showed significant improvements using the reviewed architectures and tricks in comparison to the standard Kaldi recipe, with a global gain of more than 50% of terms of EER. These experiences confirm the need to correctly define the architectures and their learning parameters to take advantage of abstract representations from deep learning architecture, in particular when these are dedicated to a given task (here, x-vectors for speaker verification).

ACKNOWLEDGEMENT

This research was supported by the ANR agency (Agence Nationale de la Recherche), on the RoboVox project (ANR-18-CE33-0014).

REFERENCES

[1] F. Bimbot, J.-F. Bonastre, C. Fredouille, G. Gravier, I. Magrin-Chagnolleau, S. Meignier, T. Merlin, J. Ortega-García, D. Petrovska-Delacrétaz, and D. A. Reynolds, "A tutorial on text-independent speaker verification," *EURASIP Journal on Advances in Signal Processing*, vol. 2004, no. 4, p. 101962, 2004.

[2] J. P. Campbell, W. Shen, W. M. Campbell, R. Schwartz, J.-F. Bonastre, and D. Matrouf, "Forensic speaker recognition," *IEEE Signal Processing Magazine*, vol. 26, no. 2, pp. 95–103, 2009.

[3] M. Rouvier and S. Meignier, "A global optimization framework for speaker diarization," in *Odyssey*, 2012.

[4] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788–798, 2010.

[5] S. Ioffe, "Probabilistic linear discriminant analysis," in *European Conference on Computer Vision*. Springer, 2006, pp. 531–542.

[6] S. J. Prince and J. H. Elder, "Probabilistic linear discriminant analysis for inferences about identity," in *2007 IEEE 11th International Conference on Computer Vision*. IEEE, 2007, pp. 1–8.

[7] P. Kenny, "Bayesian speaker verification with heavy-tailed priors." in *Odyssey*, 2010, p. 14.

[8] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, "X-vectors: Robust dnn embeddings for speaker recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5329–5333.

[9] H. Zeinali, L. Burget, J. Rohdin, T. Stafylakis, and J. H. Cernocky, "How to improve your speaker embeddings extractor in generic toolkits," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 6141–6145.

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

[11] Y. Liu, L. He, J. Liu, and M. T. Johnson, "Introducing phonetic information to speaker embedding for speaker verification," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2019, no. 1, p. 19, 2019.

[12] Z. Gao, Y. Song, I. McLoughlin, P. Li, Y. Jiang, and L. Dai, "Improving aggregation and loss function for better embedding learning in end-to-end speaker verification system," *Proc. Interspeech 2019*, pp. 361–365, 2019.

[13] J. Villalba, N. Chen, D. Snyder, D. Garcia-Romero, A. McCree, G. Sell, J. Borgstrom, F. Richardson, S. Shon, F. Grondin *et al.*, "State-of-the-art speaker recognition for telephone and video speech: the jhu-mit submission for nist sre18," *Proc. Interspeech 2019*, pp. 1488–1492, 2019.

[14] D. Snyder, D. Garcia-Romero, G. Sell, A. McCree, D. Povey, and S. Khudanpur, "Speaker recognition for multi-speaker conversations using x-vectors," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 5796–5800.

[15] H. Yamamoto, K. A. Lee, K. Okabe, and T. Koshinaka, "Speaker augmentation and bandwidth extension for deep speaker embedding," *Proc. Interspeech 2019*, pp. 406–410, 2019.

[16] M. McLaren, D. Castan, M. K. Nandwana, L. Ferrer, and E. Yilmaz, "How to train your speaker embeddings extractor," 2018.

[17] O. Novotnỳ, O. Plchot, P. Matejka, L. Mosner, and O. Glembek, "On the use of x-vectors for robust speaker recognition." in *Odyssey*, 2018, pp. 168–175.

[18] D. Snyder, G. Chen, and D. Povey, "Musan: A music, speech, and noise corpus," *arXiv preprint arXiv:1510.08484*, 2015.

[19] L. Burget, O. Novotnỳ, and O. Glembek, "Analysis of but submission in far-field scenarios of voices 2019 challenge," 2019.

[20] J. S. Chung, A. Nagrani, and A. Zisserman, "Voxceleb2: Deep speaker recognition," *arXiv preprint arXiv:1806.05622*, 2018.

[21] M. McLaren, L. Ferrer, D. Castan, and A. Lawson, "The speakers in the wild (sitw) speaker recognition database." in *Interspeech*, 2016, pp. 818–822.

[22] A. Nagrani, J. S. Chung, and A. Zisserman, "Voxceleb: a large-scale speaker identification dataset," *arXiv preprint arXiv:1706.08612*, 2017.