# Learnable MFCCs for Speaker Verification

Xuechen Liu[1,2], Md Sahidullah[2], Tomi Kinnunen[1]
[1]School of Computing, University of Eastern Finland, Joensuu, Finland
[2]Université de Lorraine, CNRS, Inria, LORIA, F-54000, Nancy, France
Email:{xuechen.liu,md.sahidullah}@inria.fr, tkinnu@cs.uef.fi

***Abstract*—We propose a learnable mel-frequency cepstral coefficients (MFCCs) front-end architecture for deep neural network (DNN) based automatic speaker verification. Our architecture retains the simplicity and interpretability of MFCC-based features while allowing the model to be adapted to data flexibly. In practice, we formulate data-driven version of four linear transforms in a standard MFCC extractor — windowing, discrete Fourier transform (DFT), mel filterbank and discrete cosine transform (DCT). Results reported reach up to 6.7% (VoxCeleb1) and 9.7% (SITW) relative improvement in term of equal error rate (EER) from static MFCCs, without additional tuning effort.**

***Index Terms*—Speaker verification, feature extraction, mel-frequency cesptral coefficients (MFCCs).**

## I. Introduction

*Automatic speaker verification* (ASV) [1] is used in forensic voice comparison, personalization of voice-based services and, more recently, smart home electronic devices. A typical ASV system can be broken down into three elementary components: (i) a frame-level feature extractor, (ii) a speaker embedding extractor, and (iii) a speaker comparator. Their functions are, respectively, to transform a waveform into a sequence of feature vectors, to extract fixed-sized speaker embedding vectors, and to compare two speaker embeddings (one from an enrollment and the other one from a test utterance).

While previous generations of ASV technology relied largely on statistical approaches such as i-vectors [2], state-of-the-art ASV leverages from *deep neural networks* (DNNs) to extract speaker embeddings. Representative examples include *d-vector* [3] and *x-vector* [4]. Numbers of extensions from them have been proposed as well [5], [6]. Common to most of those speaker embedding extractors is using either *mel-frequency cepstral coefficients* (MFCCs) [7] or raw spectrum as frame-level features. Different from the speaker embedding extractor, whose parameters are obtained through numerical optimization, raw spectra and MFCCs are obtained with fixed/static operations. In this work, our main goal is to formulate a lightweight, data-driven version of a standard MFCC extractor.

Related recent work includes so-called *end-to-end* [8] [9] front-end solutions. Using DNN-based components that are optimized jointly, such end-to-end solutions process the raw waveform to produce either detection scores or intermediate features to be used with other components. Despite promising results, the end-to-end approaches tend to require substantial engineering efforts, making them potentially inflexible for adaptation to new applications or data. Additionally, unless some prior domain knowledge is used in designing the DNN components, such models can be difficult to interpret. Meanwhile, analysis and assessment of relative importance of different signal processing components is important in speech-related research. Interpretability is also demanded in high-stake applications, such as forensic voice comparison.

This work was partially supported by Academy of Finland (project 309629) and Inria Nancy Grand Est.

For reasons above, we advocate a novel architecture whose design is guided by one the most successful fixed feature extractor, MFCCs. Even if an MFCC extractor is typically not viewed as a neural network, it can be seen as a DNN consisting of a number of linear layers (and some non-linearities). It is therefore a natural idea to expand the speaker embedding extractor to include MFCC-specific layers to be optimized in a data-driven manner. This is enabled by defining a computational graph and the associated automatic differentiation procedures available in standard DNN toolboxes. Though we draw inspiration from similar ideas in other tasks (e.g. [10], [11]), our aim is an initial formulation and an experimental feasibility study in the context of ASV.

## II. Learnable MFCC extractor

### A. Front-end MFCC extractor

A typical MFCC extractor consists of a cascade of linear and non-linear transformations originally motivated [7] from signal processing and human auditory system considerations. Typical steps (after pre-processing) include windowing, power spectrum computation using *discrete Fourier transform* (DFT), smoothing by a bank of triangular-shaped filters, logarithmic compression and *discrete cosine transform* (DCT).

MFCCs have been used across many different speech and audio applications successfully, suggesting their generality as an application-agnostic frame-level feature. Nonetheless, the standard transformations in MFCC extractors may be improved further. For instance, [12] uses low-variance multi-taper spectrum estimation to replace Hamming-windowed DFT. Other studies employ alternative time-frequency representations, such as *constant-Q transform* (CQT) [13] and wavelet-based methods [14], [15]. Different frequency warping scales are studied in [16]. Similarly, triangular filterbank can be replaced with Gaussian and Gammatone filterbanks [17]. The commonly used logarithmic compression is also substituted with cube-root compression [18]. The suitability of block DCT as an alternative of standard DCT (i.e, DCT-II) is explored in [19].

All the above studies focus on developing other fixed operation models by overcoming some of the limitations of the existing one. Differently from those studies, we propose to optimize the parameters of MFCC pipeline in a data-driven manner. We consider making learnable components based on static MFCCs only, as dynamic (delta) coefficients were not found useful in our previous work [20].

### B. Differentiable linear transforms for MFCCs

With the above motivations, we would like to start from fixed MFCCs by making four highlighted differentiable linear transforms learnable. Three of them are real-valued, namely, windowing, mel filterbank and DCT. Therefore, when designing their learnable counterparts, for each component we simply create operators that have the same input and output as the static one so that we retain the same exact computational flow. The only difference from static feature extractor is that the gradient can now be back-propagated to update the numerical values of the linear transforms. DFT is an exception since it is a *complex*-valued linear operator. Nonetheless, when integrated as a step to produce a power spectrum, the operation can be expressed as:

$$\begin{aligned} |\boldsymbol{X}|^2 &= |\boldsymbol{F}\boldsymbol{x}|^2 = |(\boldsymbol{F}_{\text{real}} + j\boldsymbol{F}_{\text{imag}})\boldsymbol{x}|^2 \\ &= g(\boldsymbol{F}_{\text{real}}\boldsymbol{x}) + g(\boldsymbol{F}_{\text{imag}}\boldsymbol{x}), \end{aligned} \tag{1}$$

where $g(\boldsymbol{F}\boldsymbol{x}) = |\boldsymbol{F}\boldsymbol{x}|^2$. Here, $\boldsymbol{x}$ is a windowed speech frame, $\boldsymbol{X}$ is the complex-valued spectrum, $\boldsymbol{F}$ is the complex-valued DFT matrix and $|.|^2$ denotes element-wise modulus. Thus, (1) can be implemented as two real-valued linear transforms, followed by squared summation.

## III. Optimization of learnable components

We describe below three techniques to optimize the selected components. We refer to the corresponding matrices as *kernels*, denoted by specific symbols: $\boldsymbol{W}$ for the window function, $\boldsymbol{F}$ for DFT (as noted in Equation 1), $\boldsymbol{M}$ for the mel filterbank, and $\boldsymbol{D}$ for DCT.

### A. Kernelized initialization

Trainable component are often initialized using random numbers from a normal distribution [21]. In this work, however, we assert that a standard MFCC extractor serves as a reasonable starting point for further learning. Thus, our first technique initializes each kernel with its corresponding static counterpart. For windowing, we use the Hamming window [22]. For mel filterbank and DCT static correspondents are available and can be directly used in place. For DFT, we generate kernels from the DFT matrix and separate the real and imaginary parts $\boldsymbol{F}_{\text{real}}$ and $\boldsymbol{F}_{\text{imag}}$ in Eq. (1). After initialization, training proceeds the same way as for any standard DNN-based speaker embedding extractor.

The kernel initialization sets a starting point for further adaptation. We consider two additions to the training procedure. The idea in both is to promote specific numerical properties of each static component to regulate learning, discouraging overly aggressive deviation from their respective static counterparts. We detail the two ideas, loss regularization and kernel update, in the following two subsections.

### B. Loss regularization

We modify the training objective of the speaker embedding extractor as $\mathcal{L}_{\text{new}} = \mathcal{L} + \lambda * g_{\text{loss}}(\boldsymbol{K})$, where $\mathcal{L}$ is multi-class cross-entropy loss, $\mathcal{L}_{\text{new}}$ is regularized loss, $\boldsymbol{K}$ denotes the kernel, and $\lambda$ is regularization constant. For all experiments addressed in this work, we set $\lambda = 0.1$. In Section V, systems adapted with such method are marked as *name + loss.*, where *name* is the name of adapted component. We design separate regularizer $g_{\text{loss}}(\cdot)$ for each of the four linear components.

**Windowing**. Many window functions (e.g. Hamming and Blackman) are generated using sinusoids [22]. Thus, our regularizer measures distance from the learnable window to a cosine function: $g_{\text{loss}}(\boldsymbol{W}) = ||\boldsymbol{W}_{\text{norm}} - \boldsymbol{C}||$, where $\boldsymbol{C}(n) = -\cos(2\pi n/M)$, $n \in [0, M-1]$ is a cosine function, $M$ being equal to frame length (i.e. length of window vector), $\boldsymbol{W}_{\text{norm}}$ is a mean-normalized window, and $||.||$ denotes Frobenius norm [23]. Therefore, when the constraint equals zero, the window equals a cosine function.

**DFT**. A DFT matrix is squared and symmetric. It can be split into real and imaginary parts, both of which are real-valued, squared and symmetric. We therefore introduce such property when implementing regularization by computing matrix-wise distance of the kernel to its symmetric version: $\boldsymbol{F}_{\text{dist.}} = \boldsymbol{F}_{\text{norm}} - \boldsymbol{F}_{\text{norm}}\boldsymbol{F}_{\text{norm}}^{\top}$, where $\boldsymbol{F}_{\text{dist.}}$ is the difference matrix and $\boldsymbol{F}_{\text{norm}}$ is the normalized version of $\boldsymbol{F}$. This applies to both $\boldsymbol{F}_{\text{real}}$ and $\boldsymbol{F}_{\text{imag}}$ in Eq. 1. The Frobenius norm of $\boldsymbol{F}_{\text{dist.}}$ is then used for regularization: $g_{\text{loss}}(\boldsymbol{F}) = ||\boldsymbol{F}_{\text{dist.}}||$. Therefore, when the constraint is perfectly met ($g_{\text{loss}}(\boldsymbol{F}) = 0$), we see that $\boldsymbol{F}_{\text{norm}}^{\top} = \boldsymbol{I}$, where $\boldsymbol{I}$ is an identity matrix.

**Mel filterbank**. Mel filterbank is a set of overlapped triangular filters with scaled peak magnitude, which can be either constant across all filters (our case) or varied via different frequency bins [24]. Computationally, it is a matrix with non-negative elements with high sparsity. In order to control the level of sparsity of the kernel, we adopt $L_2$ regularization [25] on the filterbank kernel to avoid over-fitting, instead of $L_1$, which tends to have a more enhancing effect on sparsity of model as a loss regularizer. Formally, $g_{\text{loss}}(\boldsymbol{M}) = ||\boldsymbol{M}||^2$.

**DCT**. A DCT matrix is orthonormal, i.e. $\boldsymbol{D}\boldsymbol{D}^{\top} = \boldsymbol{D}^{\top}\boldsymbol{D} = \boldsymbol{I}$. We employed a recently-proposed soft orthonormality loss function [26], expressed as $g_{\text{loss}}(\boldsymbol{D}) = ||\boldsymbol{D}^{\top}\boldsymbol{D} - \boldsymbol{I}||^2$, where $\boldsymbol{I}$ is the identity matrix. Optimizing such loss function minimizes the distance between the Gram matrix of $\boldsymbol{D}$ and $\boldsymbol{I}$ to encourage orthonormality.

### C. Kernel update

Aside from loss regularization, the other optimization technique performs direct update on the kernel operators every time after gradient update. Compared to loss regularization, it is a more 'brute-force' approach. The updated kernel matrix or vector is then directly used for next iteration: $\boldsymbol{K}_{\text{new}} = g_{\text{kernel}}(\boldsymbol{K})$, where, $\boldsymbol{K}$ is kernel matrix after gradient update and $\boldsymbol{K}_{\text{new}}$ is the directly-updated one used for next iteration.

In Section V, systems adapted with this method are marked as *name + kernel.*, where *name* is the name of adapted component. Design of updater $g_{\text{kernel}}(.)$ for each component is as follows.

**Windowing**. Commonly-used window functions are non-negative and symmetric. Inspired by such properties, our kernel update is $g_{\text{kernel}}(\boldsymbol{W}) = |\text{cat}(\boldsymbol{W}_{[:\text{size}/2]}, \boldsymbol{W}_{\text{flip}})|$, where $\boldsymbol{W}_{[:\text{size}/2]}$ denotes the half-size truncated version of window vector $\boldsymbol{W}$ while $\boldsymbol{W}_{\text{flip}}$ is its flipped (time-reversed) version. Here $\text{cat}.$ performs column-wise concatenation and $|.|$ denotes absolute values.

**DFT**. As noted above, DFT matrices are square and symmetric. To enhance such properties, we perform a simple update on the kernel: $g_{\text{kernel}}(\boldsymbol{F}) = \boldsymbol{F}\boldsymbol{F}^\top$, where $\boldsymbol{F}$ and $\boldsymbol{F}_{\text{new}}$ denote kernel at the end of the current iteration and the next iteration, respectively. It is easy to see that $\boldsymbol{F}_{\text{new}}$ is indeed symmetric[1]. Similar to the loss regularization scheme, this update is applied to both $\boldsymbol{F}_{\text{real}}$ and $\boldsymbol{F}_{\text{imag}}$.

**Mel filterbank**. As mel filterbank is a set of overlapped triangular filters with non-negative values, we force positivity by replacing negative elements with a small value: $\forall i, j,\ g_{\text{kernel}}(\boldsymbol{M}_{i,j}) = \epsilon$ if $\boldsymbol{M}_{i,j} \leq 0$, otherwise $\boldsymbol{M}_{i,j}$, where $\epsilon = 10^{-4}$, $i$ and $j$ denote row and column indices of the filterbank $\boldsymbol{M}$.

**DCT**. For DCT, we again capture its orthonormality requirement from its static correspondent by performing QR decomposition [23] on the learnt kernel matrix: $g_{\text{kernel}}(\boldsymbol{D}) = QR(\boldsymbol{D})$, where $QR(.)$ decomposes $\boldsymbol{D} = \boldsymbol{Q}\boldsymbol{R}$ and outputs only the orthogonal matrix $\boldsymbol{Q}$. Such an operation can be performed because the kernel learnt corresponds to DCT is set to be a square matrix, which means number of mel filters is same as number of output cepstral coefficients. We acquire such design choice because setting number of filters same as final static feature dimension can bring competitive performance, as shown in [20]. This applies to all experiments in this work, including baseline.

## IV. Data and experimental protocol

### A. Data

We trained baseline x-vector model on *dev* [27] partition of VoxCeleb1, which consists of 1211 speakers. We used the same dataset for additional training steps on learnable linear components. For evaluation, we considered one matched and another relatively mismatched condition. For the former, we used the *test* partition of VoxCeleb1 that consists of 40 speakers, 18860 genuine trials and same number of impostor trials [27]. The latter was composed of the *development* part of *speakers-in-the-wild* [28] (SITW) corpus "core-core" condition, containing 119 speakers. It contains 2597 genuine and 335629 impostor trials. We refer to the two datasets as *Voxceleb1-E* and *SITW-DEV*, respectively.

### B. System configuration

For the baseline system, we used 30 static MFCCs as the input features and replicated x-vector configuration from [4] as the speaker embedding extractor. We trained the model using VoxCeleb1 without any data augmentation and Adam [29] as optimizer. During test time, we extracted 512 dimensional speaker embedding from the first fully-connected layer after statistics pooling.

We adapted each of the four learnable front-end systems at a time, using same data as for training. In order to prevent distractions in terms of joint optimization from scratch and meet the aim of providing light-weighted interface for adaptation, the selected component was jointly optimized with the pre-trained baseline x-vector. Speaker embeddings for all systems with learnt front-end components were extracted in same manner as baseline after adaptation.

For all systems, we applied energy-based *speech activity detection* (SAD) before feature processor and *cepstral mean normalization* (CMN). All embeddings extracted at inference time were length-normalized and centered prior to being transformed by a 200-dimensional *linear discriminant analysis* (LDA). Scoring was implemented through *probabilistic linear discriminant analysis* (PLDA) [30] classifier. We used Kaldi[2] for data preparation and PLDA training and PyTorch [31] for all DNN-related training and inference experiments.

### C. Evaluation

*Equal error rate* (EER) and *minimum detection cost function* (minDCF) were used to measure ASV performance. MinDCF was computed with target speaker prior $p = 0.001$ and detection costs $C_{\text{FA}} = C_{\text{miss}} = 1.0$. We used BOSARIS[3] to produce selected detection trade-off (DET) curves.

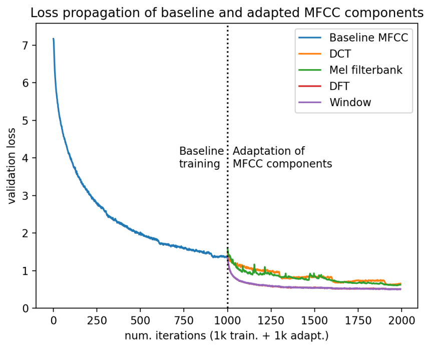


Fig. 1. Loss propagation for baseline and adapted MFCC components. Best viewed in color.

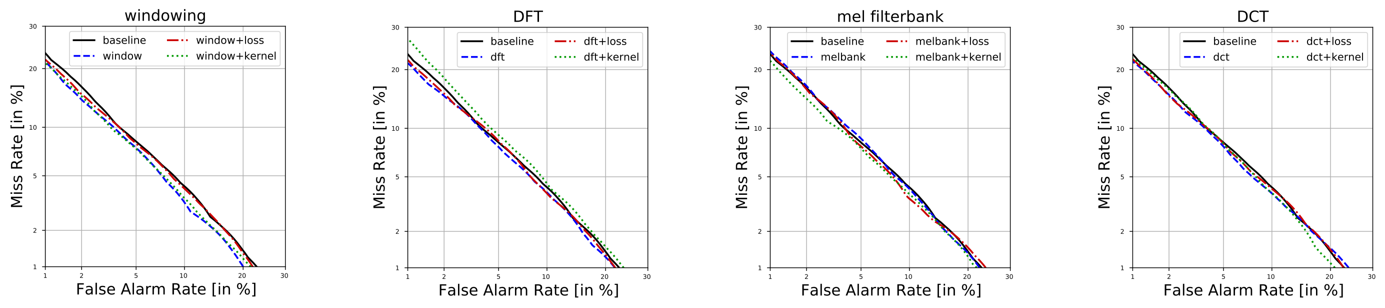
## V. Results

Before presenting ASV results, we demonstrate validation loss (on *dev* set) propagation of our baseline and adapted systems in Fig. 1. The baseline x-vector system (with fixed MFCC components) was pre-trained with 1000 iterations, followed up by another 1000 iterations to adapt the MFCC components. The adaptation, especially for window function and DFT, results in a notable decrease of validation loss. This indicates potential to make components of an MFCC extractor learnable. The ASV results are reported in Table I.

[1] $(\boldsymbol{F}\boldsymbol{F}^\top)^\top = (\boldsymbol{F}^\top)^\top\boldsymbol{F}^\top = \boldsymbol{F}\boldsymbol{F}^\top$.

[2] https://github.com/kaldi-asr/kaldi

[3] https://sites.google.com/site/bosaristoolkit/



(a) windowing (b) DFT (c) mel filterbank (d) DCT

Fig. 2. DET plots for *SITW-DEV*. Best viewed in color.

TABLE I
RESULTS ON *Voxceleb1-E* AND *SITW-DEV*. ALL SYSTEMS ASIDE FROM BASELINE ARE WITH KERNEL INITIALIZATION.

| | Voxceleb1-E | | SITW-DEV | |
|---|---|---|---|---|
| Operator (+optimal.) | EER(%) | minDCF | EER(%) | minDCF |
| Baseline MFCC | 4.64 | 0.6071 | 6.72 | 0.8243 |
| Window | 4.51 | 0.5544 | **6.09** | 0.7698 |
| Window + loss. | 4.40 | 0.5508 | 6.66 | 0.7915 |
| Window + kernel. | 4.42 | 0.5459 | **6.09** | 0.7819 |
| DFT | **4.33** | 0.5933 | 6.35 | 0.7698 |
| DFT + loss. | 4.71 | 0.6156 | 6.62 | 0.8041 |
| DFT + kernel. | 5.42 | 0.6182 | 7.04 | 0.7903 |
| Melbank | 4.83 | 0.5767 | 6.52 | 0.8253 |
| Melbank + loss. | 4.63 | 0.6162 | 6.39 | 0.8011 |
| Melbank + kernel. | 4.45 | 0.5768 | 6.31 | **0.7689** |
| DCT | 4.36 | 0.5572 | 6.27 | 0.7950 |
| DCT + loss. | 4.46 | **0.4971** | 6.54 | 0.7982 |
| DCT + kernel. | 4.41 | 0.5501 | 6.54 | 0.7925 |

## A. ASV results on Voxceleb1-E

Concerning windowing, all the three adapted variants outperform the baseline. Loss regularization and kernel update are particularly more effective. The results indicate usefulness to retain symmetricity and positivity of the window.

Concerning DFT, simply letting it to be data-driven (without added regularization or kernel update) yields lowest EER among all systems, with a relative improvement of 6.7% over the baseline. In fact, the additional regularization and direct update are detrimental. This indicates potential weakness of our symmetricity constraint.

Concerning the mel filterbank, *Melbank + kernel.* yields the best performance among the three adapted variants, with best minDCF of all systems, improving baseline by relatively 6.7%. This indicates the importance of enforcing positivity of the learnt filters.

Concerning DCT, similar to windowing all the learning schemes improve upon the baseline. While QR decomposition does not bring notable positive impact, the orthonormality-enhancing loss regularization results in slightly worse EER, but improved minDCF. In fact, *DCT + loss.* results in lowest minDCF among all systems.

## B. ASV results on SITW-DEV

We now move on to discuss ASV results on the more challenging *SITW-DEV* data. Overall, the data-driven components yield now more competitive performance boost over the baseline. Adapting the window function is most effective, with a relative improvement of 9.7% in EER over the baseline. Concerning DFT, *DFT + loss.* slightly outperforms baseline in both metrics while *DFT + kernel.* is the only variant that does not reach baseline in EER. This finding is in line with *VoxCeleb1-E* results. Concerning mel filterbank, all the three systems outperform baseline. Overall, it achieves competitive performance compared with learnable DCT. It reflects its potential on being made adaptable to improve system robustness.

DET curves for single systems including baseline on *SITW-DEV* have been shown in Fig. 2. The curves agree with observations from Table I in general. Systems with window function adapted produce largest improvement gap with baseline compared with other three, which can be reflected from EER. Considering systems that are less strict on false alarms, we can see that ones like *DCT + kernel.* and *window + loss.* are exceptional and thus can be taken into concern.

## VI. CONCLUSION

We conducted an initial study on a lightweight learnable MFCC feature extractor as a compromise between complex end-to-end architectures and fixed, hand-crafted feature extractors. Our initial results on *SITW-DEV* are promising: the proposed scheme improved upon baseline MFCC extractor. Results for optimized window and mel filterbank are particularly promising. Due to our domain-specific optimization constraints, the learnt representations bear close resemblance to fixed MFCC operations. For interpretability and computational reasons, we restricted the focus on optimization of individual MFCC extractor components; joint optimization of all the four linear componentswas left as a future work. Similarly, the work can be extended with other deep models such as extended TDNN and ResNet using larger datasets and data augmentation.

## REFERENCES

[1] J. H. L. Hansen and T. Hasan, "Speaker recognition by machines and humans: A tutorial review," *IEEE Signal Processing Magazine*, vol. 32, no. 6, pp. 74–99, 2015.

[2] 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, May 2011.

[3] E. Variani et al., “Deep neural networks for small footprint text-dependent speaker verification,” in *Proc. ICASSP*, 2014, pp. 4052–4056.

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

[5] L. You, W. Guo, D. Li, and J. Du, “Multi-Task learning with high-order statistics for X-vector based text-independent speaker verification,” in *Proc. INTERSPEECH*, 2019, pp. 1158–1162.

[6] D. Snyder, D. Garcia-Romero, G. Sell, A. McCree, D. Povey, and S. Khudanpur, “Speaker recognition for multi-speaker conversations using x-vectors,” in *Proc. ICASSP*, 2019, pp. 5796–5800.

[7] S. Davis and P. Mermelstein, “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences,” *ACOUSTICS, SPEECH AND SIGNAL PROCESSING, IEEE TRANSACTIONS ON*, pp. 357–366, 1980.

[8] M. Ravanelli and Y. Bengio, “Speaker recognition from raw waveform with SincNet,” in *Proc. SLT*, 2018, pp. 1021–1028.

[9] N. Zeghidour, N. Usunier, I. Kokkinos, T. Schaiz, G. Synnaeve, and E. Dupoux, “Learning filterbanks from raw speech for phone recognition,” in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 5509–5513.

[10] S. Vasquez and M. Lewis, “Melnet: A generative model for audio in the frequency domain,” *ArXiv*, vol. abs/1906.01083, 2019.

[11] M. Pariente, S. Cornell, A. Deleforge, and E. Vincent, “Filterbank design for end-to-end speech separation,” in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 6364–6368.

[12] T. Kinnunen et al., “Low-variance multitaper MFCC features: A case study in robust speaker verification,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 7, pp. 1990–2001, 2012.

[13] M. Todisco, H. Delgado, and N. Evans, “Articulation rate filtering of CQCC features for automatic speaker verification,” in *Proc. INTERSPEECH*, 2016, pp. 3628–3632.

[14] J. Andén and S. Mallat, “Deep scattering spectrum,” *IEEE Transactions on Signal Processing*, vol. 62, no. 16, pp. 4114–4128, 2014.

[15] O. Farooq and S. Datta, “Mel filter-like admissible wavelet packet structure for speech recognition,” *IEEE Signal Processing Letters*, vol. 8, no. 7, pp. 196–198, 2001.

[16] S. Umesh, L. Cohen, and D. Nelson, “Fitting the mel scale,” in *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, 1999, vol. 1, pp. 217–220 vol.1.

[17] C. Kim and R. M. Stern, “Power-normalized cepstral coefficients (PNCC) for robust speech recognition,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 7, pp. 1315–1329, 2016.

[18] H. Hermansky, “Perceptual linear predictive (plp) analysis of speech,” *The Journal of the Acoustical Society of America*, vol. 87, no. 4, pp. 1738–1752, 1990.

[19] A. K. Naveena and N. K. Narayanan, “Block dct coefficients and histogram for image retrieval,” in *2017 International Conference on Signal Processing and Communication (ICSPC)*, 2017, pp. 48–52.

[20] X. Liu, M. Sahidullah, and T. Kinnunen., “A comparative re-assessment of feature extractors for deep speaker embeddings,” in *Proc. INTERSPEECH*, 2020.

[21] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feed-forward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010, pp. 249–256, PMLR.

[22] F. J. Harris, “On the use of windows for harmonic analysis with the discrete fourier transform,” *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, 1978.

[23] G.H. Golub and C.F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, 1996.

[24] R. Lawrence and J. Hwang, *Fundamentals of Speech Recognition*, Prentice-Hall, Inc., USA, 1993.

[25] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[26] Y. Zhu and B. Mak, “Orthogonality Regularizations for End-to-End Speaker Verification,” in *Proc. Odyssey 2020 The Speaker and Language Recognition Workshop*, 2020, pp. 17–23.

[27] A. Nagrani, J. Chung, and A. Zisserman, “VoxCeleb: A large-scale speaker identification dataset,” in *Proc. INTERSPEECH*, 2017, pp. 2616–2620.

[28] M. McLaren, Luciana Ferrer, Diego Castán Lavilla, and Aaron Lawson, “The speakers in the wild (SITW) speaker recognition database,” in *Proc. INTERSPEECH*, 2016, pp. 818–822.

[29] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015*, 2015.

[30] S. Ioffe, “Probabilistic linear discriminant analysis,” in *Computer Vision – ECCV 2006*, Aleš Leonardis, Horst Bischof, and Axel Pinz, Eds., Berlin, Heidelberg, 2006, pp. 531–542, Springer Berlin Heidelberg.

[31] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, L. Zeming, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” in *NIPS 2017 Workshop on Autodiff*, 2017.