On-the-Fly Data Loader and Utterance-Level Aggregation for Speaker and Language Recognition

Weicheng Cai, *Student Member, IEEE*, Jinkun Chen, Jun Zhang, and Ming Li, *Senior Member, IEEE*

***Abstract*—In this article, our recent efforts on directly modeling utterance-level aggregation for speaker and language recognition is summarized. First, an on-the-fly data loader for efficient network training is proposed. The data loader acts as a bridge between the full-length utterances and the network. It generates mini-batch samples on the fly, which allows batch-wise variable-length training and online data augmentation. Second, the traditional dictionary learning and Baum-Welch statistical accumulation mechanisms are applied to the network structure, and a learnable dictionary encoding (LDE) layer is introduced. The former accumulates discriminative statistics from the variable-length input sequence and outputs a single fixed-dimensional utterance-level representation. Experiments were conducted on four different datasets, namely NIST LRE 2007, AP17-OLR, SITW, and NIST SRE 2016. Experimental results show the effectiveness of the proposed batch-wise variable-length training with online data augmentation and the LDE layer, which significantly outperforms the baseline methods.**

***Index Terms*—Speaker and language recognition, deep neural network, variable-length training, online data augmentation, utterance-level aggregation.**

I. Introduction

SPEAKER and language recognition can be defined as an utterance-level "variable-length sequence classification" task. In this problem, our aim is to retrieve attributes about an entire utterance rather than specific word content [1], [2]. Moreover, for the text-independent task, the lexical words are not constrained; thus, the training utterances and testing segments may have completely different contents [3]. Therefore, given the variable-length nature of speech utterances, our goal is to aggregate them into fixed-dimensional representations, where the inter-class variability is maximized while the intra-class variability is minimized [4].

Manuscript received April 10, 2019; revised August 27, 2019, November 25, 2019, and February 15, 2020; accepted March 2, 2020. Date of publication March 16, 2020; date of current version April 3, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61773413, in part by the Key Research and Development Program of Jiangsu Province (BE2019054), in part by Six Talent Peaks Project in Jiangsu Province under Grant JY-074, and in part by Guangzhou Municipal People's Livelihood Science and Technology Plan (201903010040). The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Kong Aik Lee. *(Corresponding author: Ming Li.)*

Weicheng Cai, Jinkun Chen, and Jun Zhang are with the School of Electronics and Information Technology, Sun Yat-sen University, Guangzhou 510275, China (e-mail: caiwch3@mail2.sysu.edu.cn; chenjk8@mail2.sysu.edu.cn; zhangj266@mail.sysu.edu.cn).

Ming Li is with the Data Science Research Center, Duke Kunshan University, Suzhou 215316, China, and also with the School of Computer Science, Wuhan University, Wuhan 430081, China (e-mail: ming.li369@dukekunshan.edu.cn).

Digital Object Identifier 10.1109/TASLP.2020.2980991

There are two major categories of methods to obtain utterance-level representation. The first comprises stacking self-contained algorithmic components, and a representative of this category is the classical i-vector approach [5]. Any variable-length speech utterance can be represented as a low-dimensional i-vector by accumulating the statistics over time. The process to extract the i-vector contains a series of separated models, and they are commonly trained in an unsupervised manner. Moreover, the front-end i-vector extractor and back-end classifier are optimized separately.

The second category relies on the model trained by a downstream procedure through deep neural networks (DNNs). In the early stages, the speaker and language recognition system based on a DNN commonly performs at the frame level. The representative is the d-vector approach [6]. In this kind of method, a set of fixed-length short training samples should be prepared in advance [6]–[10], and these DNNs only give representations or posteriors at the frame level. A further aggregating operation is required to obtain the final utterance-level representation.

However, variable-length sequences occur naturally in speech due to variations in speaker, speaking rate, and phonetic context [11]. The fixed-length frame-level DNNs are not a natural way to deal with real-world speech utterances with arbitrary durations. Recently, the utterance-level framework has gained more attention in the speaker and language recognition community. Among others, the state-of-the-art x-vector system shows superior performance [12]. In the test phase, fixed-dimensional x-vector embeddings can be extracted from utterances with arbitrary durations.

In this paper, similar to the x-vector approach, our focus is on modeling utterance-level aggregation for speaker and language recognition. Here, a new DNN training scheme is introduced and its detailed implementations are presented. The main contributions are summarized as follows.

A. Data Preparation

In many previous DNN training implementations, fixed-dimensional feature representations are extracted and stored on disk ahead of time. Taking the Kaldi [13] implementation of the x-vector recipe as an example, several archive files containing data chunks with different frame lengths and augmentation types are carefully generated before the DNN training. This procedure can be regarded as offline data preparation. In this paper, a new solution with a flexible online workflow is presented. Instead of dumping all the necessary data chunks into the disk, a data loader to produce materials for DNN training dynamically is

introduced. All the operations within the data loader are configurable and composable, which makes it more adjustable to any desired transformation or augmentation algorithms. Based on the on-the-fly data loader, a batch-wise variable-length training strategy and an online data augmentation mechanism for efficient and robust DNN training are further proposed.

B. Network Structure

First, we present a ResNet-style neural network [14] for speaker and language recognition rather than using the time-delay neural network (TDNN) [15] which is adopted for the x-vector architecture. Based upon the ResNet backbone, several pooling layers are investigated, including the temporal average pooling (TAP) and self-attentive pooling (SAP) layers, which have been shown to be effective in previous literature [16]–[18]. Further introduced is a learnable dictionary encoding (LDE) layer for utterance-level aggregation. This layer receives a variable-length sequence as input and produces a single supervector-size utterance-level representation. It integrates the traditional dictionary learning and statistical accumulation steps into a single layer, which is suitable for DNN learning.

The data loader acts as a "producer," and the network plays the role of a "consumer." They are mutually complementary and form a new learning scheme for DNN-based speaker and language recognition. Its implementation details are presented on a modern multiprocessing computation platform, and its performance is validated on several benchmark datasets.

This paper is an extension of our previous works [18], [19] with the following extensions. First, a comprehensive description of the data loader for DNN training is presented. Second, the effect of variable-length training and an online data augmentation strategy for speaker and language recognition is validated. Third, a more systematic exploration of the LDE layer is conducted, and its robustness under different types of data-preparation conditions is also validated. Finally, the performance on additional evaluation sets is further verified and compared with the state-of-the-art methods.

The rest of this paper is organized as follows. An overview of the utterance-level DNN framework is presented in Section II. The general data-preparation paradigm, along with the proposed data loader, is then explained in Section III. In Section IV, several context-independent pooling layers (including the LDE layer) for dealing with the variable-length sequence are elaborated. Experimental results and discussions are presented in Section V, and conclusions are provided in Section VI.

II. Utterance-Level DNN Framework

A general framework of the utterance-level aggregation in the DNN is described in this section. As depicted in Fig. 1, the network accepts variable-length input and produces an utterance-level result.

First, the raw waveform can be effectively represented as a short-term spectral feature sequence based on several hand-crafted filters [20]. Since DNNs typically require a multi-dimensional array as their input, this kind of full-length sequence of arbitrary duration is not very suitable for DNN training.

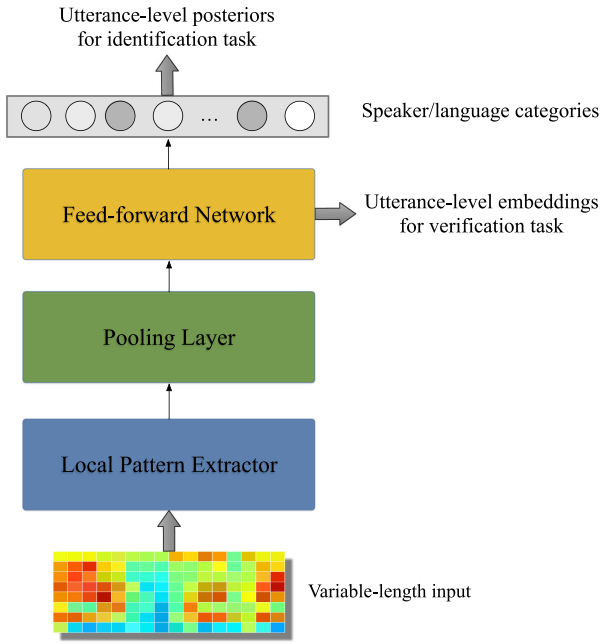


Fig. 1. Utterance-level DNN framework for speaker and language recognition. It accepts input data sequences with variable length, and produces an utterance-level result.

Therefore, an automatic data loader is implemented to generate mini-batch training samples efficiently. Detailed information is presented in Section III.

Given the input feature sequences, sufficient statistics could be accumulated and transformed into an utterance-level representation as done in the classical i-vector approach [5]. However, recent studies have shown that a front-end DNN module works well before aggregating the utterance-level representation. Any hierarchical network module able to handle variable-length inputs can be utilized as the front-end local pattern learning module, including the CNN [17], [21], TDNN [16], LSTM network [8], [22], or even CNN-Bidirectional LSTM (CNN-BLSTM) [23].

The representations learned by the front-end local pattern extractor follow still their temporal order. The remaining question is how to aggregate the whole sequence over the entire and potentially long duration. Therefore, several forms of context-independent pooling layers are presented in Section IV.

The utterance-level representation can then be processed through a standard feed-forward network, and an output layer can be built on top. Each unit in the output layer is represented as a target speaker or language category. All the components in the pipeline are trained jointly in an end-to-end manner with a unified loss function.

For the closed-set identification task, the front-end deep feature extractor and back-end classifier can be jointly trained, and the utterance-level posterior for each category can be directly derived from the DNN output. The entire identification system can be optimized with a softmax-based cross-entropy loss. This is referred to as the softmax loss for simplicity.

For the open-set speaker verification (SV) task, the DNN acts as a feature extractor, and another model is needed to generate

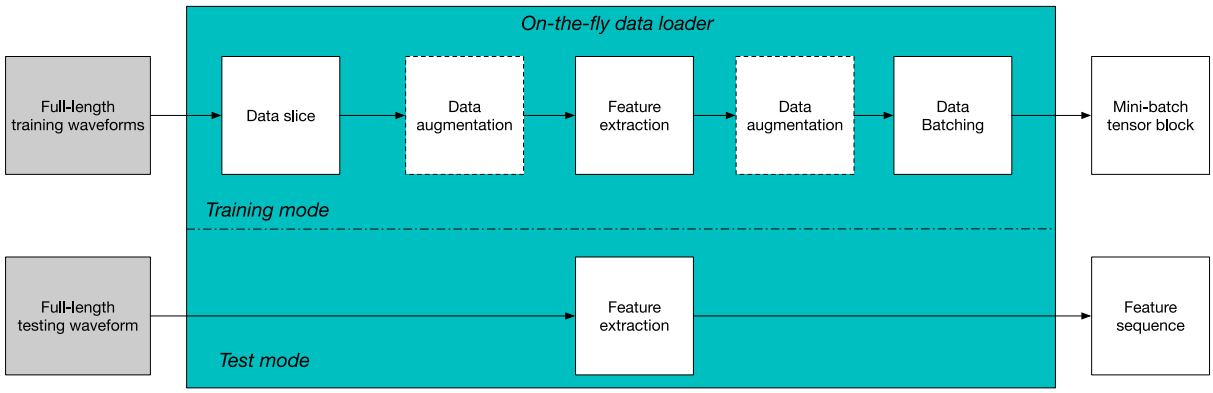


Fig. 2. Block diagram of data-loader DNN-based speaker and language recognition. In the training phase, multiple full-length training waveforms are converted into a mini-batch of tensor blocks. In the test phase, the full-length test waveform is converted into a sequence of feature vectors.

the pairwise scores for the extracted speaker embeddings. Probabilistic linear discriminant analysis (PLDA) [24] scoring has been the de facto standard in both i-vector [5] and x-vector [12] frameworks.

Regarding the SV task, since it is impractical to gather all possible target identities for training, the extracted speaker embedding needs to be not only separable but also discriminative. Cai *et al.* [18] showed the superiority of center loss [25] and angular softmax (A-Softmax) loss [26] for training more discriminative speaker embeddings.

III. On-the-Fly Data Loader

Data preparation in conventional speech and language recognition systems typically follows an off-line paradigm. Feature vectors are generated and stored on the disk first, and utterances are processed one at a time. There is no need to resize the length of utterances, and the variable-length feature sequence can finally be transformed as fixed-dimensional utterance-level representations like the i-vector. Modern GPUs process batches of arithmetic operations significantly faster than sequential processing, and grouping data into batches can make full use of the hardware resources for DNN training. To this end, many previous works train DNNs with fixed-length sequences, which do not take into the variable-length nature of speech.

In this work, our aim is to find the balance between the variable-length nature of speech signal and parallel computing power. More specifically, an on-the-fly data loader for DNN-based speaker and language recognition is introduced. The data loader is adopted to maintain an online processing workflow by generate training samples on the fly. As shown in Fig. 2, there are multiple real-time operations within the data loader, including the data slicing, data transformation (i.e., feature extraction and data augmentation), and data batching operations.

This design principle allows one to efficiently perform a batch-wise random perturbation, such as variable-length data slicing and online data augmentation. All operations are executed on the fly, and the training samples are generated right before feeding them into the DNNs. In other words, what the actual training sample is not precisely known until a specific DNN training step begins. This uncertainty and randomness let the network see different training segments derived from the same utterance in different epochs. Supposing a total number of U training utterances, the DNN is trained with E epochs. Using the proposed data loader with a variable-length and online data augmentation strategy, $U \times E$ samples are obtained for DNN training.

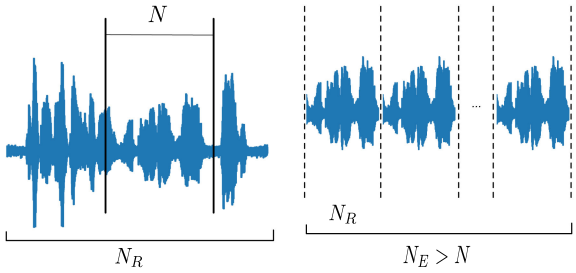
In theory, arbitrary combinations of operations can be made within the data loader to greatly augment the data. All training samples are generated on the fly, which allows one to focus on the operations themselves, without the need to store temporary data chunks that are not required for the final model inference. This flexibility allows the re-design of experiments with different configurations by adjusting the data loader with new specifications, without the need to prepare new sets of feature vectors.

Since the data flow from the raw waveform to the DNN output is maintained, it also promotes model inference and deployment with ease. After the DNN has been trained, as shown in Fig. 2, the data loader can be easily be turned into the "test" mode by setting the batch size to one and removing the data slicing, data augmentation, and data batching operations. The corresponding utterance-level result can be generated directly by feeding the arbitrary-duration test waveform into the data loader and then inferred by the DNN model.

The three primary operations within the data loader are now described in detail.

A. Data Slicing

Although the training length can be different for each mini-batch, for a specific training step a fixed-length tensor block is still needed. To this end, a truncate operation is first implemented in the data slicing module. For each training step, a random integer N is first generated in the interval $[N_{\min}, N_{\max}]$ from a uniform distribution. The goal is to sample a continuous data chunk of length N from the raw utterance of length N_R. To this end, a random starting point S is first regulated from the interval $[0, N_R - N]$. The continuous data block of length N is then truncated from the interval $[S, S + N]$, as demonstrated in Fig. 3(a). Since the variables S and N are not the same for each



(a) Truncate operation in data loader (b) Extend operation in data loader

Fig. 3. Two core operations on raw utterance within data loader.

Algorithm 1: The Proposed Variable-Length Data Loader in Training Phase.

Output: Mini-batch tensor $\mathbf{Z} \in \mathbb{R}^{B\times F\times L}$ for each step
Input: Indexed list of full-length utterances: $\mathcal{X} = [x_1, x_2, x_3, \ldots, x_{N-1}, x_N]$. Each utterance is given as a one-dimensional waveform.
Input: Total number of utterances U
Input: Total number of training epochs P
Input: Batch size B
Input: Minimum sample points N_{min}
Input: Maximum sample points N_{max}

procedure Dataloader$\mathcal{X}$, U, E, B, L_{min}, L_{max}
  Number of batches in an epoch $Q \leftarrow \lceil U/B \rceil$
  for $i = 1$ to P **do**
    Shuffle indexed list randomly $\mathcal{Y} \leftarrow$ Rand($\mathcal{X}$)
    for $j = 1$ to Q **do**
      Generate N_j from range $[N_{\text{min}}, N_{\text{max}}]$ randomly
      $\mathbf{Z} \in \mathbb{R}^{B\times F\times L} \leftarrow$ random initialization
      for $k = 1$ to B **do**
        Load G with N_{Rj} points from $\mathcal{Y}_{k+j\times B}$
        if $N_{Rj} > N_j$ **then**
          $G \leftarrow$ TRUNCATE(G)
        else
          $G_E \leftarrow$ EXTEND(G)
          $G \leftarrow$ TRUNCATE(G_E)
        end if
        $G \leftarrow$Time Domain AUGMENTATION(G)
        $G \in \mathbb{R}^{F\times L} \leftarrow$ TRANSFROM(G)
        $\mathbf{Z}(k) \leftarrow$ Frequency or Feature Domain AUGMENTATION(G)
      end for
      returnZ
    end for
  end for
end procedure

training step, the data chunk obtained at each epoch might be different.

This truncate operation simulates the real-world scenario that a test segment might belong to any part of the original audio. Furthermore, where the starting point is or how long it will last is often not known. The goal is to train a robust speaker and language recognition system that gives a consistent and correct decision on not only the full-length raw utterance, but also on its appropriate sub-segments.

Regarding the short utterances with length N_R being less than the generated random integer N, an extending operation is implemented before truncation that is inspired by the phenomenon in which the speaker/language information does not change when the same content is spoken multiple times. The short utterance is spoken several times until the extended length of N_E is larger than the random integer N, as shown in Fig. 3(b). After the extended data of shape N_E is obtained, the same truncate operation as described in the previous paragraph is performed.

B. Data Transform

For a given slice of data chunk from a raw full-length utterance, one can first perform several time-domain data-augmentation operations on the waveform. For example, one can change the speed of the audio signal [27] or add some noise, or music background as done in the x-vector approach. One can then perform a typical feature extraction operation by transforming the augmented waveform into a two-dimensional feature tensor of size $F \times L$, where F denotes the feature coefficients and L denotes the frame length. Moreover, a frequency-domain data augmentation can be performed directly on the extracted feature sequence.

As given in Algorithm 1, for each training step, all of the extracted feature sequences are grouped together to construct a mini-batch tensor of shape $B \times F \times L$, where B, F and L denote the batch size, the feature dimension, and the frame length, respectively. Since L is different for each mini-batch, the training samples for each step may have different lengths.

C. Design Details

The full algorithm is shown in Algorithm 1. In summary, the data loader acts as a "producer" that generates mini-batch samples for DNN training from the full-length utterances. Meanwhile, the DNN behaves as a "consumer" that uses these mini-batches of samples to optimize the model parameters. This is considered a "producer-consumer" scheme. A solution is thus proposed that separates the data loading and DNN training modules by placing a queue in the middle, letting the producers and consumers execute in different threads. This decouples the data preparation and DNN training stages, letting the two procedures run asynchronously and in parallel. Furthermore, the data loader is accelerated with a multi-thread processing strategy. Multiple workers are employed to process different batches of the data and push them to a shared queue for consumption, as illustrated in Fig. 4.

In modern computation platforms, multi-core CPUs are utilized to implement the data loader task and employ multiple GPUs to train the network. Under the "producer-consumer" scheme, data loading for the next several batches on CPUs is done in parallel with the forward-backward pass on the GPUs. The ideal status is that GPUs can directly fetch the next batch of data from the queue immediately after finishing the current step. In this case, the data loader runs in the background at all time, the latency of which can be minimized.

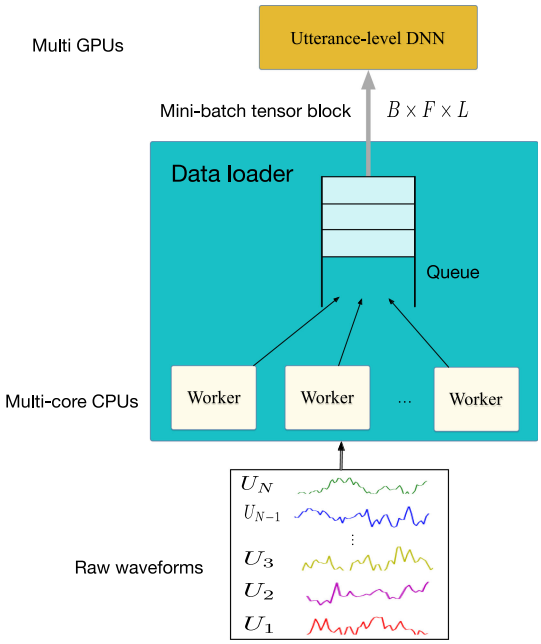


Fig. 4. Producer-consumer design paradigm of data loader.

IV. Utterance-Level Aggregation

In this section, several utterance-level aggregation layers in the deep speaker and language recognition system are presented, including the TAP layer, the self-attentive pooling (SAP) layer, and the proposed LDE layer.

A. Temporal Average Pooling (TAP) Layer

The front-end local pattern extractor module is implemented with a deep CNN architecture. For a given frame-level feature matrix of size $F \times L$ (where F denotes the feature dimension along the frequency axis and L is the length along the time axis), the CNN feature maps are typically three-dimensional tensor blocks of size $D \times H \times T$, where D denotes the number of channels and H and T denote the height and width of the feature maps, respectively. Generally, H and T are much smaller than the original F and L, due to the down-sampling operations within the CNN structure.

A TAP layer expects its inputs to be two-dimensional sequences. Therefore, the output of the CNN along its height (or frequency) axis is pooled into a two-dimensional $D \times T$ tensor. Here, D not only denotes the number of channels of the CNN, but also represents the input feature dimension of the TAP layer. Similarly, T not only denotes the width of the CNN feature maps, but also represents the time steps in the TAP layer. Note that T is a variable considering different input frames L.

The TAP layer aggregates the $D \times T$ feature matrix by accumulating the mean statistics along the time axis.

B. Self-Attentive Pooling (SAP) Layer

The TAP layer pools the CNN feature maps uniformly along the time axis. However, not all frames contribute equally to the utterance-level representation. Therefore, attention-based pooling is an active area of research in the speaker and language recognition community. Intuitively, an attention model allows the DNN to focus on frames that are informative to the utterance-level representation.

The SAP layer is implemented in a manner similar to [18], [28]–[30]; that is, first the variable-length sequence $\{\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T\}$ is fed into an multi-layer perceptron (MLP) to obtain $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_T\}$ as a hidden representation. In this paper, a one-layer perceptron is simply adopted:

$$\mathbf{h}_t = \tanh(\mathbf{W}\mathbf{x}_t + \mathbf{b}). \tag{1}$$

Then, the importance of each frame is measured as the similarity of $\mathbf{h}_t$ with a learnable context vector $\boldsymbol{\mu}$, and a normalized importance weight γ_t is obtained through a softmax function:

$$\gamma_t = \frac{\exp(\mathbf{h}_t^\top \boldsymbol{\mu})}{\sum_{t=1}^{T} \exp(\mathbf{h}_t^\top \boldsymbol{\mu})}. \tag{2}$$

The context vector $\boldsymbol{\mu}$ can be seen as a high-level representation of a fixed query "what is the informative frame over the whole frame" [28]. It is randomly initialized and jointly learned during the training process.

After that, the utterance-level representation $\mathbf{e}$ is generated as a weighted sum of the input feature sequence based on the learned weights:

$$\mathbf{e} = \sum_{t=1}^{T} \gamma_t \mathbf{o}_t. \tag{3}$$

C. Learnable Dictionary Encoding (LDE) Layer

The idea of the LDE layer was initially proposed for the task of texture recognition in computer vision [31]. It treats the two dimensions of a given image equally and accumulates statistics along both the width and height axes. Considering the fact that speech is translation-invariant along the time axis only, the original LDE layer is modified to aggregate feature descriptors along the time axis rather than both the time and frequency axes.

Given a variable-length feature matrix of shape $D \times T$, the LDE layer aggregates them over time. More specifically, it transforms them into a single $CD \times 1$-dimensional utterance-level representation, which is independent of the length T.

Fig. 5 illustrates the forward pass of the LDE layer. Here, a set of learnable parameters is introduced first: $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \ldots \boldsymbol{\mu}_C\}$.

Each vector $\boldsymbol{\mu}_c$ represents the center of a dictionary component, and it acts in a way similar to the mean of a Gaussian component in the conventional Gaussian Mixter Model (GMM). Similar to the soft-alignment in the GMM, features are independently assigned to dictionary components with non-negative weights given by a softmax function:

$$\gamma_t(c) = \frac{\exp(-s\|\mathbf{o}_t - \boldsymbol{\mu}_c\|^2)}{\sum_{m=1}^{C} \exp(-s\|\mathbf{o}_t - \boldsymbol{\mu}_m\|^2)}. \tag{4}$$

Here, the smoothing factor s controls the decay of the response with the magnitude of the distance.

One can fix s to be a positive constant. By expanding the squares operation in (4), it is easy to see that the term $e^{-s\|o_i\|^2}$

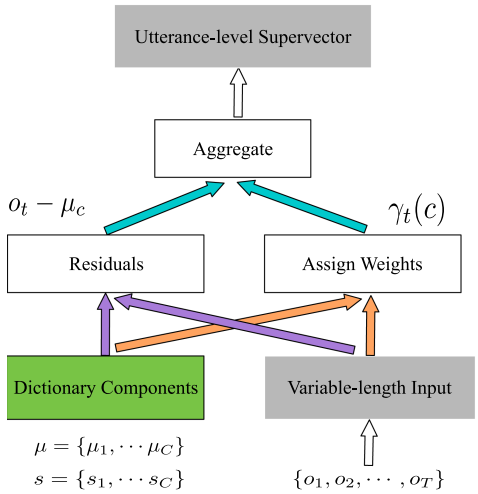


Fig. 5. Block diagram illustrating the forward pass through the LDE layer.

cancels the numerator and denominator, resulting in a soft assignment of the following form:

$$\bar{\gamma}_t(c) = \frac{e^{\mathbf{w}_c^\top \mathbf{x}_i + b_c}}{\sum_m e^{\mathbf{w}_m^\top \mathbf{x}_i + b_m}}, \tag{5}$$

where $\mathbf{w}_c = 2\, s\boldsymbol{\mu}_c$ is a vector and $b_k = -\alpha\|\boldsymbol{\mu}_c\|^2$ a scalar.

Inspired by the component weights in the conventional GMM, the s_c for each cluster center $\boldsymbol{\mu}_c$ is further allowed to be learnable:

$$\gamma_t(c) = \frac{\exp(-s_c\|\mathbf{o}_t - \boldsymbol{\mu}_c\|^2)}{\sum_{m=1}^{C} \exp(-s_m\|\mathbf{o}_t - \boldsymbol{\mu}_m\|^2)}. \tag{6}$$

Given a set of T frames and a dictionary, each frame is assigned a weight $\gamma_t(c)$ with respect to each component $\mu_\mathbf{c}$, where $t = 1, 2 \ldots T$ and $c = 1, 2 \ldots C$.

Therefore, the zeroth-order Baum-Welch statistics with respect to each mixture component are calculated as follows:

$$N_c = \sum_{t=1}^{T} \gamma_t(c). \tag{7}$$

In a similar manner, the centered first-order Baum–Welch statistics with respect to each mixture component are calculated as follows:

$$\mathbf{F}_c = \sum_{t=1}^{T} \gamma_t(c)(\mathbf{o}_t - \boldsymbol{\mu}_c). \tag{8}$$

This represents the aggregated residual between o_t and $\boldsymbol{\mu}_c$ over time. The aggregated residual is then normalized by the zeroth-order statistics:

$$\tilde{\mathbf{F}}_c = \frac{\mathbf{F}(c)}{N(c)}. \tag{9}$$

Note that $N(c)$ is a constant, and in practice, one can replace this scale normalization operation by a L_2 normalization operation. Therefore, (9) can be replaced as

$$\tilde{\mathbf{F}}_c = \frac{\mathbf{F}(c)}{\|\mathbf{F}(c)\|_2}. \tag{10}$$

Similar to the case in the conventional GMM supervector/i-vector approach, $\mathbf{F}_c$ is concatenated from all dictionary components. Thus, the LDE layer outputs a supervector-sized representation $\tilde{\mathbf{F}} = \{\tilde{\mathbf{F}}_1^\top, \tilde{\mathbf{F}}_2^\top \ldots \tilde{\mathbf{F}}_C^\top\}^\top$.

In summary, a set of learnable parameters $\lambda_c = \{\boldsymbol{\mu}_c, s_c, \}, c = 1, \ldots, C$ is introduced in the LDE layer, where $\boldsymbol{\mu}_c$ is the center of the dictionary component and s_c denotes the scaling factor. The LDE layer is a directed acyclic graph, and all the components are differentiable $w.r.t$ the input $\boldsymbol{O}$ and the learnable parameters. Therefore, the LDE layer can be trained in an end-to-end manner with the standard stochastic gradient descent algorithm.

The parameter set $\boldsymbol{\mu}$ represents the Gaussian mean vectors in the GMM, and s_c acts as the mixture weight. In this sense, the LDE layer integrates the powerful Baum-Welch statistical accumulation procedure into DNNs. The output $CD \times 1$ supervector-sized representation has the same role as the supervector in the i-vector GMM, and the supervised supervector is obtained from the LDE layer. This supervector is projected into the low-dimensional speaker embedding.

The LDE layer can be considered a combination of traditional dictionary learning and the pooling layer. Here, the relationship between the LDE layer and other methods is presented.

1) Relationship to Conventional Dictionary Learning: A dictionary is usually learned from the distribution of descriptors in an unsupervised manner. The K-means method learns the dictionary using hard-assignment grouping. The GMM is a probabilistic version of K-means, which allows more delicate modeling of the feature distributions. Each cluster is modeled with a Gaussian component with its mean vector, covariance matrix, and mixture weight. The LDE layer makes the inherent dictionary differentiable and learns the dictionary in a supervised manner. To see the similarities of the LDE layer to K-means, consider Fig. 5 with the omission of residual vectors and let the smoothing factor $s \to \infty$. With these modifications, the LDE layer acts like K-means. Meanwhile, the LDE layer can also be regarded as a simplified version of the GMM, which neglects the covariance matrix.

2) Relationship to TAP and SAP Layers: Letting $C = 1$ and $\boldsymbol{\mu} = 0$, the LDE layer simplifies to a TAP layer. Note that the context vector in the SAP layer plays a role similar to the dictionary centroid in the LDE layer. If one uses a multiple context vector to compute multiple groups of attentive weights, it becomes multi-head attention [32]. However, the LDE layer normalizes the component posteriors to let them sum to unity across all Gaussian components, rather than normalizing the posteriors along the time axis. In a loose sense, the LDE layer can be seen an extension of the SAP layer with multiple context vectors.

V. Experimental Results and Discussion

A. Datasets and Evaluation Metrics

Experiments were conducted using two language identification databases (NIST Language Recognition Evaluation (LRE) 2007 [33] and AP17-OLR [34]) and two speaker verification databases (SITW [35] and NIST SRE 2016 [36]).

1) NIST LRE 2007: The task of interest is closed-set language detection. This corpus has 14 target languages: Arabic, Bengali, Chinese, English, Hindustani, Spanish, Farsi, German, Japanese, Korean, Russian, Tamil, Thai, and Vietnamese. There are 7530 utterances in total, with three nominal durations of 3 s (2–4 s actual), 10 s (7–13 s actual), and 30 s (25–35 s actual). Each of the three durations has 2510 segments. The training corpus employed includes the Callfriend, LRE 2003, LRE 2005, and SRE 2008 datasets, and the development set of LRE 2007. The total number of training utterances is 37095. System performance is reported in terms of equal error rate (EER) and C_{avg} as defined in [33].

2) AP17-OLR: The dateset consists of 10 oriental languages: Mandarin, Cantonese, Indonesian, Japanese, Russian, Korean, Vietnamese, Kazakh, Tibetan, and Uyghur [34]. Three test conditions were defined according to the length of the test utterances: 1-s, 3-s, and full-utterance. The 1- and 3-s test utterances are random excerpts of the original ones. Each test condition has 22051 utterances. AP16-OL7 and AP17-OL3 were used as training data, totalling 72234 utterances for training. System performance is reported in terms of EER and C_{avg} as defined in [34].

3) Speakers in the Wild (SITW): The SITW dataset consists of unconstrained audio-visual data of English speakers, which naturally includes noises, reverberation, as well as device and codec variability [35]. Our focus is on its core-core protocol for both the development and evaluation sets. The duration of enrollment and testing utterances in the test set varies in length from 6 to 240 s. Following the Kaldi recipe, the pooled VoxCeleb1 and VoxCeleb2 datasets were used as our training set and speakers overlapping with the evaluation set were removed. Finally, a training set of 1236567 utterances from 7323 celebrities was obtained. System performance is reported in terms of equal error rate (EER) and C_{det} as defined in [35].

4) NIST SRE 2016: Our training corpus includes data from NIST SRE 2004-2010, Mixer 6, and Switchboard 2 Phase 1, 2, and 3 as well as Switchboard Cellular. The CallMyNet test set is composed of telephone conversations collected outside North America, spoken in Tagalog and Cantonese. The duration of the enrollment segments is approximately 60 s, while the test segments are uniformly sampled, ranging from 10 to 60 s. An unlabeled development set of approximately 2200 calls from the CallMyNet collection is also available. System performance is evaluated in terms of EER and C_{det} as defined in [36].

B. Network Setup

To obtain a higher-level abstract representation, a deep CNN based on the well-known ResNet [14] architecture was designed. The primary network structure and parameters are described in Table I. Batch normalization followed by ReLU activation functions after every convolutional layer was used. The kernels in the convolutional layer are 3×3. A down-sampling operation was performed on the last three blocks (Res2, Res3, and Res4) using a stride factor of 2 in the first convolutional layer. Therefore, the input feature images are down-sampled to $(1/8 \times 1/8)$ "images" along the time-frequency axes. Our network is slightly "thin" because the output channels of the front-end ResNet are only up to 128. Experiments with different types of pooling layers were conducted to accumulate statistics over time. A fully connected layer then processed the utterance-level representation produced by the pooling layers, which was then connected to a classification output layer. All the components in the pipeline were jointly trained with a typical softmax loss.

TABLE I
RESNET-LDE STRUCTURE AND THE NUMBER OF PARAMETERS IN EACH MODULE

Layer	Input size	Output size	Structure	#Parameters
Conv1	$1\times64\times L$	$16\times64\times L$	3×3, stride 1	144
Res1	$16\times64\times L$	$16\times64\times L$	$\begin{bmatrix} 3\times3,\ 16 \\ 3\times3,\ 16 \end{bmatrix}\times3$	$4K\times3$
Res2	$16\times64\times L$	$32\times32\times\frac{L}{2}$	$\begin{bmatrix} 3\times3,\ 32 \\ 3\times3,\ 32 \end{bmatrix}\times4$	$18K\times4$
Res3	$32\times32\times\frac{L}{2}$	$64\times16\times\frac{L}{4}$	$\begin{bmatrix} 3\times3,\ 64 \\ 3\times3,\ 64 \end{bmatrix}\times6$	$72K\times6$
Res4	$64\times16\times\frac{L}{4}$	$128\times8\times\frac{L}{8}$	$\begin{bmatrix} 3\times3,\ 128 \\ 3\times3,\ 128 \end{bmatrix}\times3$	$288K\times3$
Squeeze	$128\times8\times\frac{L}{8}$	$128\times\frac{L}{8}$	8×1 Avgpool	0
LDE	$128\times\frac{L}{8}$	$128C$	C components	$C\times(128+1)$
FC1(embd)	$128C$	128	FC	$128\times(128C+1)$
FC2(output)	128	Classes	FC	$129\times$ Classes

The widely used stochastic gradient descent algorithm with a momentum of 0.9 and weight decay of 1e-4 was used. The learning rate was set to 0.1, 0.01, and 0.001, and it was switched when the training loss plateaued. Since there was no separate validation set, the converged model after the last optimization step was used for evaluation.

In the training phase, the model was trained with a mini-batch size of 128. First, a continuous data chunk was sliced from the raw full-length utterance, and it was then converted into 64-dimensional log Mel-filterbank energies (Fbanks) within the data loader. The frame length is 25 ms, with a frame shift of 10s ms. A cepstral mean subtraction was applied over the sliced data chunk. After the batching operations, a dynamic mini-batch tensor with a shape of $128 \times 64 \times L$ was generated on-the-fly for each training step, where L is a batch-wise variable number ranging from L_{low} to L_{high}. How to set the suitable number for different datasets is discussed in the following section.

In the test phase, all the data with different durations were evaluated on a single unified model. Because the duration is arbitrary, the test utterances were fed to the trained DNNs one by one. For the identification task, the utterance-level posterior can be directly retrieved from the DNN outputs. For the verification task, the utterance-level embedding can be extracted from the penultimate layer, and a simple cosine similarity or PLDA was adopted to obtain the final pairwise score.

C. Effect of Variable-Length Training Mechanism

In this section, the baseline TAP layer is used for temporal pooling, and the effect of variable-length training for speaker and language recognition is investigated.

The proposed data loader can flexibly produce samples of any specific length. Although it is designed primarily for variable-length training, it can be adapted to generate fixed-length segments over the entire training stage so as to simulate the conventional fixed-length training scenario.

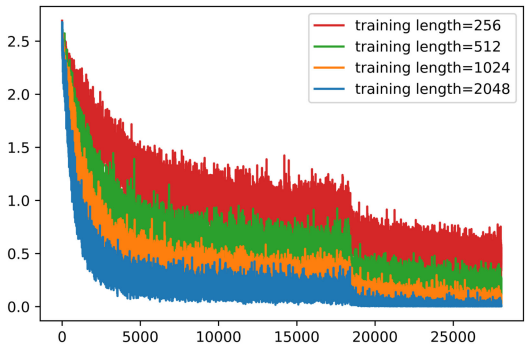


Fig. 6. Comparison of learning curves for different lengths of training segments on NIST LRE 2007.

TABLE II
PERFORMANCE COMPARISON OF FIXED- AND VARIABLE-LENGTH TRAINING ON 2007 NIST LRE. BOLDFACE DENOTES BEST PERFORMANCE FOR EACH COLUMN. [EER(%)]

ID	Training length (frames)	3-s condition	10-s condition	30s condition
1	Fixed 256	14.65	10.51	9.33
2	Fixed 512	13.73	8.54	6.81
3	Fixed 1024	13.96	7.83	5.74
4	Fixed 2048	16.23	8.63	6.23
5	Batch-wise [100, 300]	14.48	9.36	8.01
6	Batch-wise [200, 400]	14.29	8.01	7.96
7	Batch-wise [300, 800]	**11.28**	5.76	3.96
8	Batch-wise [800, 1200]	12.57	**5.42**	**3.74**
8	Batch-wise [1200, 2000]	15.71	8.19	5.01
9	Epoch-wise [300, 800]	11.95	5.92	4.03

Several numbers of typical lengths, including 256, 512, 1024, and 2048, were tried for NIST LRE 2007. Fig. 6 shows the learning curves. The configuration used for all the curves are the same, except for the length of the training segments. The training loss not only fluctuates wildly during training, but also converges to a rather high value when the DNN is fed with short training segments of 256. As the training length increases, the loss curve becomes much smoother and more stable. There is clear distance between these learning curves, and one can find a distinct gap among them. The network trained with longer segments produces more stable and smoother training curves and eventually achieves a lower loss value, which indicates a higher training accuracy. The converged loss value is approximately 0.45 for length 256, and 0.012 for length 2048. This demonstrates the difficulty of learning information from training samples with short durations. DNNs tend to be optimized well for a long-duration dataset and much easier to perfectly fit the long-duration training samples. This behavior is the same as that seen with the traditional i-vector, which is known to obtain a better representation for long-duration utterances.

However, the test results in Table II are not consistent with the training loss tendency. While the performance consistently improved when the length increased from 256 to 1024, it decreases as the training length increases from 1024 to 2048. This means that, although DNNs are apt to reach higher training accuracy on training data with longer durations, an overfitting problem might occur. The same phenomenon appears more distinctly in Table III for the AP17-OLR evaluation set. When it comes to the 1- or 3-s duration test condition, one might intuitively think that better testing performance could be obtained if the training sample length matches the testing duration. However, in our experiments, when DNNs were fed with training samples of 1-s duration, performance on the 1-s task was inferior to those systems with a variable-length training strategy. Results on the 3-s task also confirm our assumption that variable-length training is also superior to duration-matched fixed-length training.

TABLE III
PERFORMANCE COMPARISON OF FIXED- AND VARIABLE-LENGTH TRAINING ON AP17-OLR. BOLDFACE DENOTES BEST PERFORMANCE FOR EACH COLUMN. [EER(%)]

ID	Training length (frames)	1-s condition	3-s condition	Full-length
1	Fixed 100	14.93	11.36	9.25
2	Fixed 300	13.92	10.98	8.93
3	Fixed 800	13.31	9.36	7.59
4	Fixed 1200	13.31	10.07	8.45
5	Batch-wise [100, 300]	14.25	11.21	9.12
6	Batch-wise [200, 400]	13.01	9.11	7.48
7	Batch-wise [300, 800]	**12.93**	**9.05**	**7.17**
8	Batch-wise [800, 1200]	13.22	9.59	8.23
9	Epoch-wise [300, 800]	13.09	9.17	7.30

Since significant differences exist among DNNs trained with different lengths, one might wonder whether it is feasible to perform a batch-wise training strategy by pooling all the training samples with profoundly different lengths. For a more in-depth analysis of this issue, multiple curves are visualized in Fig. 7. An interesting phenomenon can be seen from these figures.

First, since the length of each training step is switched arbitrarily, the batch-wise learning curve demonstrated in Fig. 7(a) looks somewhat noisy. After performing the moving average operation, an epoch-wise loss curve was obtained, as shown in Fig. 7(b). The curve is almost perfectly smooth and decreases quite steadily. It can be inferred that although variable-length training leads to local instability, the downward trend of the loss curve is still very stable over the long term.

The last 1000 steps were extracted from Fig. 7(a), and are shown separately in Fig. 7(c). Even if this data block locates in the latest area near the convergence region, it can be found that the loss is dithering drastically. While these points all seem to be in disarray along the time axis, valuable information is revealed when the data are switched to their corresponding scatter plots (Fig. 7(d)). In this plot, the randomly selected training length is shown along the horizontal axis between [300, 800], and loss value along the Y axis and each time step is represented by one dot. The dots form a dispersed plot, yet there is undoubtedly a trend for the time steps with long lengths to have lower loss values.

Although the long- and short-duration samples lead to somewhat different training tendencies and performances, they can be combined with a variable-length training strategy. The variable-length training strategy makes it possible to incorporate diverse duration conditions, which makes it converge to a better optimization point and also more robust against overfitting.

When performing variable-length training, a key factor affecting the performance is the selection of the training length

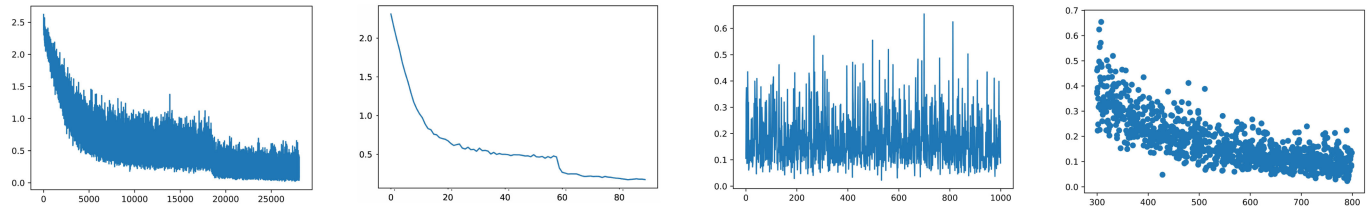


Fig. 7. Learning curves indicating softmax loss at each step in training the neural network for NIST LRE 2007. (a) Batch-wise learning curve; (b) epoch-wise learning curve; (c) learning curve for last 1000 steps; (d) scattered graph for last 1000 steps.

TABLE IV
PERFORMANCE COMPARISON OF FIXED- AND VARIABLE-LENGTH TRAINING ON SITW. BOLDFACE DENOTES BEST PERFORMANCE FOR EACH COLUMN. [EER(%)]

ID	Training length (frames)	Development set	Evaluation set
1	Fixed 100	5.41	6.51
2	Fixed 300	4.72	5.64
3	Fixed 800	4.74	5.58
4	Fixed 1200	4.95	5.71
5	Batch-wise [100, 300]	4.81	5.75
6	Batch-wise [200, 400]	**4.69**	**5.37**
7	Batch-wise [300, 800]	4.73	5.45
8	Batch-wise [800, 1200]	4.94	5.65
9	Epoch-wise [300, 800]	4.98	5.61

TABLE V
PERFORMANCE COMPARISON OF FIXED- AND VARIABLE-LENGTH TRAINING ON NIST SRE 2016. BOLDFACE DENOTES BEST PERFORMANCE FOR EACH COLUMN. [EER(%)]

ID	Training length (frames)	Pooled	Tagalog	Cantonese
1	Fixed 100	16.21	22.10	11.08
2	Fixed 300	14.46	19.64	10.27
3	Fixed 800	12.96	16.46	8.41
4	Fixed 1200	12.01	15.90	7.95
5	Fixed 2000	11.05	**14.08**	7.06
6	Batch-wise [100, 300]	14.80	19.84	10.92
7	Batch-wise [200, 400]	13.01	17.98	9.57
8	Batch-wise [300, 800]	11.25	16.04	8.01
9	Batch-wise [800, 1200]	10.64	15.25	7.04
10	Batch-wise [1200, 2000]	**9.87**	14.75	**6.59**
11	Epoch-wise [300, 800]	11.39	16.27	8.14

interval, i.e., the upper and lower bounds. A smaller lower bound introduces more short-duration training samples. This may lead to anti-over-fitting, but it could also provoke a non-convergence issue. For the upper bound, in order to have a wider range of training length, one may look for a larger upper bound. However, A larger upper bound introduces more long-duration training samples, which might lead to over-fitting. Moreover, the GPU memory usage grows linearly as longer training length is used. For example, training DNNs with data of 1000 frames requires a frame count five times larger than 200 frames.

Tables IV and III show that a training interval of [300, 800] or even [200, 400] could achieve satisfying performance on SITW and AP17-OLR test sets. Larger training length leads to over-fitting and brings worse results. In contrast, for NIST LRE 2007 and NIST SRE 2016, the DNNs have difficulty to converge when training with short-duration data. To achieve a better performance, DNNs need to be trained with longer-duration data. For NIST SRE 2016, DNNs trained with data length within the interval [1200, 2000] achieves the best performance.

The choice of the training length range depends on the specific training data, and one could set proper lower and upper bounds, considering the training-data fitting degree. There is also a trade-off between system performance and GPU memory usage. In our implementation, it was found that a training range of [300, 800] can achieve good performance in most cases while not consuming many hardware resources. In the following experiments, L_{low} and L_{high} are fixed as 300 and 800, respectively.

Different implementations of the proposed data loader were also investigated, namely an epoch-wise strategy and a batch-wise strategy. For the epoch-wise strategy, the length of the training segments was changed epoch by epoch; for the batch-wise strategy, the length of the training segments was changed batch by batch. From the results shown in Tables II and III, it is evident that the batch-wise strategy achieves better performance.

TABLE VI
PERFORMANCE COMPARISON OF DIFFERENT IMPLEMENTATIONS OF LDE LAYER ON NIST LRE 2007. BOLDFACE DENOTES BEST PERFORMANCE FOR EACH COLUMN. [EER(%)]

ID	Scale factor	Components	3-s condition	10s condition	30-s condition
1	Learnable s	C=16	8.89	2.73	1.13
2	Learnable s	C=32	8.12	2.45	0.98
3	Learnable s	C=64	**7.75**	**2.31**	**0.96**
4	Learnable s	C=128	8.20	2.49	1.12
5	Learnable s	C=256	8.59	2.87	1.38
6	Fixed s	C=64	8.23	2.35	1.07

D. Exploration of LDE-Layer Implementation

1) Number of Components: In the conventional GMM-UBM approach, as the number of Gaussian components increases, the performance varies. The number of mixtures is typically up to 1024 or even 2048. Several settings were attempted in our LDE-layer implementation. The performance on NIST LRE 2007 with different sizes of LDE dictionaries is presented in Table VI. When the number of dictionary components increases from 16 to 64, the performance improves. However, once the number of dictionary components exceeds 64, the performance decreases, perhaps because of over-fitting.

TABLE VII
PERFORMANCE COMPARISON OF DIFFERENT DATA-PREPARATION MECHANISMS ON AP17-OLR. DA DENOTES DATA AUGMENTATION AND BOLDFACE DENOTES BEST PERFORMANCE FOR EACH COLUMN. [EER(%)]

ID	Data preparation	1-s condition	3-s condition	Full-length
1	Offline mode	10.69	7.35	4.29
2	Offline mode+ DA	9.70	6.49	3.92
3	Online mode	10.03	6.98	4.04
4	Online mode+ DA	**8.98**	**6.30**	**3.58**

2) Scale Factor: Two types of LDE layers were implemeted with different setups of the scale factor s. For the first one, the scale factor was fixed as a constant for all the dictionary components. In the second setup, s is a group of component-wise learnable parameters. Here, the number of dictionary components was fixed to 64 in order to compare the aforementioned two setups of the scale factor s. The experimental results on NIST LRE 2007 are shown in Table VI, showing that the learnable scale factor performs better than the constant one.

E. Impact of Data-Augmentation Strategy

In this section, the LDE layer is used for temporal pooling and the impact of the data-augmentation strategy for speaker and language recognition tasks is investigated.

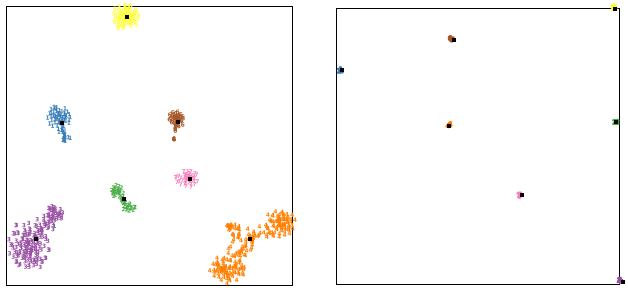
As a control group, the offline data preparation strategy was followed as in many previous DNN training procedures. Several files containing training segments with various frame lengths were dumped into the disk before DNN training. Further, a system with the same data augmentation described in the x-vector approach was built by adding the reverberation, noise, music, and babble noise into the audio files. The script provided by the Kaldi SITW recipe [37] was used, and, finally, feature data with twice the number of original audios were obtained, one-half of which are the clean data and the other half the augmented data.

Furthermore, the data loader was employed to generate the training samples on the fly. Given the significant performance gains of data augmentation in the x-vector approach, the same data- augmentation operation was also employed in our proposed on-the-fly data loader. The difference is that the augmented feature files are not generated physically in the hard drive; all the operations were processed on the fly in memory. Different noisy types, noise source waves, clips in the noise source, and signal-to-noise ratio (SNR) were randomly selected to generate the training segments. All these permutations were randomly performed on the variable-length segments, so the network sees different noisy segments from the same clean speech. To guarantee the proper partition of the clean data, for each audio file a [0-1] uniform distributed random variable was sampled. If the value was below 0.5, data augmentation was not performed for this file. If the value was equal or higher than 0.5, one random data-augmentation operation was added when preparing the mini-batch tensors.

From Tables VII and VIII, it can be seen that the online data loader performs slightly better than the offline one. For both the offline and online procedures, data augmentation significantly improves the performance. Finally, the on-the-fly data loader with online data augmentation achieves the best performance.

TABLE VIII
PERFORMANCE COMPARISON OF DIFFERENT DATA-PREPARATION MECHANISMS ON SITW. DA DENOTES DATA AUGMENTATION AND BOLDFACE DENOTES BEST PERFORMANCE FOR EACH COLUMN. [EER(%)]

ID	Data preparation	Development set	Evaluation set
1	Offline mode	3.99	4.71
2	Offline mode + DA	3.74	4.25
3	Online mode	3.95	4.52
4	Online mode + DA	**3.62**	**4.10**



(a) t-SNE plot of speaker embeddings from full-length utterances and their sub-segments as shown in 3(a). Scatter points of the same color mean they are from the same utterance.

(b) t-SNE plot of speaker embeddings from full-length utterances and their rolled or repeated variants as shown in 3(b). Scatter points of the same color mean they are from the same utterance.

Fig. 8. Embedding visualization of the two operations in Fig. 3 using t-SNE on SITW.

F. Validation of Truncate and Extend Operations

In our data loader, there are two core data slicing operations: a truncate operation and an extend operation. The truncate operation produces a random training sample by taking a random sub-segment from a full-length utterance. The extend operation produces a training sample by repeating a full-length utterance multiple times. Our goal is to validate whether our network has the capability to robustly make correct predictions on these two variants in the test phase.

Seven audio files were randomly selected from the SITW test set. For each utterance, its corresponding speaker embeddings were extracted. In addition, speaker embeddings for their random sub-segments were also extracted. The t-distributed Stochastic Neighbor Embedding (t-SNE) is plotted in Fig. 8(a). The black dots are the embeddings of the raw full-length test utterances and the others are the speaker embeddings of their sub-segments. It can be seen that the speaker embeddings from the same utterance are naturally clustered together.

Fig. 8(b) visualizes the speaker embeddings from the full-length utterance and its repeated as well as rolled variants. It can also be seen that those speaker embeddings from the same utterance are naturally clustered together.

This property is important when developing text-independent speaker verification or language identification systems. The extracted embeddings are robust against lexical variabilities.

G. Comparison With TAP and SAP Baselines

In this section, the proposed LDE layer is compared with the baseline TAP and SAP layers. From Tables IX–XII, it can be seen

TABLE IX
SYSTEM PERFORMANCE ON 2007 NIST LRE CLOSED-SET TASK. NR DENOTES NOT REPORTED AND ResNet-LDE V2 DENOTES ResNet-LDE NETWORK WITH DROPOUT WHILE ResNet-LDE V3 DENOTES A LARGER SIZE ResNet-LDE NETWORK WITH DROPOUT

ID	System	$C_{avg}(\%)$/EER(%)		
		3-s task	10-s task	30s Task
1	ResNet-TAP	9.01/10.19	2.98/5.30	1.60/3.56
2	ResNet-SAP	7.83/9.01	2.58/3.94	1.31/1.20
3	ResNet-LDE	7.52/6.99	2.34/2.12	1.09/0.91
4	ResNet-LDE V2	7.39/6.84	2.35/2.00	1.01/0.87
5	ResNet-LDE V3	**6.98/6.11**	**2.18/1.94**	**0.94/0.81**
6	GMM i-vector [21]	20.46/8.29	3.02/17.71	3.02/2.27
7	DNN i-vector [21]	14.64/12.04	6.20/3.74	2.60/1.29
8	DNN PPP feature [21]	**8.00/6.90**	**2.20/1.43**	**0.61/0.32**
9	DNN Tandem Feature [21]	9.85/7.96	3.16/1.95	0.97/0.51
10	DNN Phonotactic [38]	18.59/12.79	6.28/4.21	1.34/0.79
11	RNN D&C [38]	22.67/15.57	9.45/6.81	3.28/3.25
12	LSTM-Attention [39]	NR/14.72	NR	NR
13	ResNet-GRU [21]	11.31/10.74	5.49/6.40	NR
14	ResNet-LSTM [21]	10.17/9.80	4.66/4.26	NR

TABLE X
SYSTEM PERFORMANCE ON AP17-OLR. ResNet-LDE V2 DENOTES ResNet-LDE NETWORK WITH DROPOUT WHILE ResNet-LDE V3 DENOTES A LARGER-SIZE ResNet-LDE NETWORK WITH DROPOUT

ID	System	$C_{avg}(\%)$/EER(%)		
		1-s task	3-s task	Full-length
1	ResNet-TAP	11.38/11.90	8.19/8.30	6.30/6.51
2	ResNet-SAP	9.71/10.20	6.80/7.39	4.80/5.98
3	ResNet-LDE	8.80/8.98	5.51/6.30	3.41/3.58
4	ResNet-LDE V2	8.82/8.87	5.57/6.21	3.35/3.58
5	ResNet-LDE V3	**7.21/7.63**	**4.81/5.02**	**2.81/3.20**
6	i-vector	**14.85/14.43**	**6.24/6.07**	**4.69/4.58**
7	TDNN-LID [34]	16.04/15.63	15.23/15.43	14.51/14.65
8	LSTM-LID [34]	15.69/16.77	15.25/16.99	14.68/16.03
9	PTN-LID [34]	11.53/11.88	7.27/8.24	6.89/8.15

that both the SAP layer and LDE layer outperform the baseline TAP layer on all four speaker and language recognition tasks. Furthermore, systems with a LDE layer show better performance compared to a SAP layer.

H. Additional Performance Improvement With Dropout and Large-Size Network

In previous reports, dropout has been shown to reduce over-fitting and improve the performance of speaker recognition tasks [40], [41]. This motivates us to propose a **ResNet-LDE V2** system that adds a dropout layer with 0.5 probability. From Tables IX and XII, it can be observed that significant performance improvements are achieved in both the speaker verification and language identification tasks.

Furthermore, since the proposed on-the-fly data loader can provide a large amount of augmented data and the usage of dropout could further reduce the risk of over-fitting, a more extensive network was also explored for better performance. The 'thin ResNet' in Table I might not be sufficient, and thus

TABLE XI
SYSTEM PERFORMANCE ON SITW DATASET. ResNet-LDE V2 DENOTES ResNet-LDE NETWORK WITH DROPOUT WHILE ResNet-LDE V3 DENOTES A LARGER-SIZED ResNet-LDE NETWORK WITH DROPOUT

ID	System	Scoring	min C_{det}/EER(%)	
			Development set	Evaluation set
1	ResNet-TAP	Cosine	0.347/4.58	0.419/5.22
2	ResNet-SAP	Cosine	0.308/4.00	0.373/4.76
3	ResNet-LDE	Cosine	0.273/3.62	0.320/4.14
4	ResNet-LDE V2	Cosine	0.245/3.26	0.292/3.70
5	ResNet-LDE V3	Cosine	0.206/2.30	**0.223**/2.79
6	ResNet-LDE V3	PLDA	**0.201/2.20**	0.237/**2.61**
7	i-vector	PLDA	0.425/4.81	0.463/5.65
8	x-vector	Cosine	0.777/15.05	0.818/ 15.30
9	x-vector	PLDA	**0.313/ 3.08**	**0.348/ 3.41**

TABLE XII
SYSTEM PERFORMANCE ON NIST SRE 2016 EVALUATION SET. ResNet-LDE V2 DENOTES ResNet-LDE NETWORK WITH DROPOUT WHILE ResNet-LDE V3 DENOTES A LARGER-SIZED ResNet-LDE NETWORK WITH DROPOUT

ID	System	Scoring	min C_{det}/EER(%)		
			Pooled	Tagalog	Cantonese
1	ResNet-TAP	Cosine	0.89/19.21	0.92/23.31	0.78/15.08
2	ResNet-TAP	PLDA	0.68/10.83	0.82/14.99	0.52/6.78
3	ResNet-SAP	PLDA	0.66/9.88	0.78/14.34	0.49/5.68
4	ResNet-LDE	PLDA	0.64/9.33	0.77/13.59	0.46/5.24
5	ResNet-LDE V2	PLDA	0.62/8.60	0.75/12.31	0.41/4.81
6	ResNet-LDE V3	PLDA	**0.56/7.98**	**0.71/11.62**	**0.35/3.90**
7	i-vector	PLDA	0.73/12.98	0.87/17.80	0.59/8.35
8	x-vector	Cosine	0.95/34.21	0.96/37.68	0.89/31.66
9	x-vector	PLDA	**0.61/8.57**	**0.76/12.29**	**0.42/4.89**

a **ResNet-LDE V3** system that modifies the widths (number of channels) of the residual blocks from {16, 32, 64, 128} to {32, 64, 128, 256} is proposed. This ResNet-LDE V3 system also adopts a dropout layer with 0.5 probability.

From Tables IX–XII, it can be seen that the proposed larger network in the ResNet-LDE V3 system could be well trained and achieves state-of-the-art performance.

I. Discussions of State-of-the-Art Approaches

1) Discussion of the I-Vector Approach: As shown in Tables IX–XI, the deep speaker and language recognition system significantly outperforms the conventional i-vector approach in most conditions.

Since i-vector is an unsupervised approach, a supervised back-end is still needed. For closed-set identification, a back-end logistic regression or SVM might be adopted to obtain the final score. For the open-set verification, PLDA might be employed as the back-end. On the contrary, in our supervised DNN framework, the class posteriors can be generated directly from the network output layer in the identification task. For the open-set SITW task, a simple cosine similarity measure is sufficient for the scoring.

Nevertheless, the i-vector system still performs well in some long-duration language identification tasks, especially when phonetic information is integrated into the i-vector training phase. As shown in Table IX, the i-vector trained with phoneme

discriminant features extracted from the acoustic model achieves the best performance, especially for the 30-s condition. This indicates that the simple acoustic filter banks might not be the best choice, and we can integrate the phonetic information into DNN training as described in [34], [42], [43].

2) Discussion of Fixed-Length Training for DNNs: DNNs trained with fixed-length inputs typically perform at the frame level. Most previous studies report successful results on short-duration tasks (<3 s) [7]–[9]. Taking the AP17-OLR dataset as an example, the TDNN-LID and LSTM-LID systems reported in [34] only achieve performance comparable to that of the i-vector approach on the 1-s short-duration task.

Moreover, even for some systems utilizing a statistics pooling layer for better performance, different networks are trained and tested separately for the 30-, 10-s, and 3-s conditions [38], [44].

In our experiments, a single unified model was adopted to generate all the 3-, 10-, and 30-s condition results. Our experiments show that a unified model trained with variable-length inputs can handle all 3-, 10-, and 30-s conditions. It may not be necessary to train specific models to match various kinds of test durations.

3) Discussion of Other Utterance-Level DNNs: The most representative utterance-level DNN system using variable-length inputs is the well-known x-vector system.

As shown in Table XI, for the SITW test set, it can be seen that a simple cosine similarity back-end achieves satisfying performance. Adding a PLDA back-end achieves a marginal gain. However, direct cosine similarity scoring on the x-vectors leads to degraded results, and an additional PLDA back-end is required to achieve state-of-the-art performance. In this sense, the speaker embeddings extracted by the proposed ResNet-based system might contain more salient speaker-discriminative information. In the face recognition domain, the state-of-the-art methods also only employ a cosine similarity back-end for scoring [26]. This might reveal new insights into developing the embedding network.

For NIST SRE 2016, the change in language introduces a shift between the data distributions of the training and evaluation data. Even if one trains a perfect network on the training set, the extracted embeddings might not follow the distribution of the evaluation set. Therefore, a simple cosine scoring could not achieve satisfying performance. From Table XI, it can be seen that both the x-vector and proposed systems obtains poor performance with cosine scoring. Thus, the adaptive PLDA back-end is then adopted as implemented by the Kaldi SRE16 recipe [45]. Both of these two systems achieve significant gain.

J. Performance of Computational Efficiency

1) Data-Loading Efficiency: Here, the focus is on the data loader to measure its data loading efficiency. For each step, one should read a set of raw audio files from the disk and then transform them into a mini-batch tensor. Therefore, this procedure might need a massive file I/O and CPU computation.

First, an experiment was carried out with an Intel Xeon E6-2630 @2.2 GHz CPU on a Seagate ST2000NX0253 hard disk drive (HDD). It can be seen from Fig. 9 that it takes approximately 35 seconds to complete a full data loading procedure.

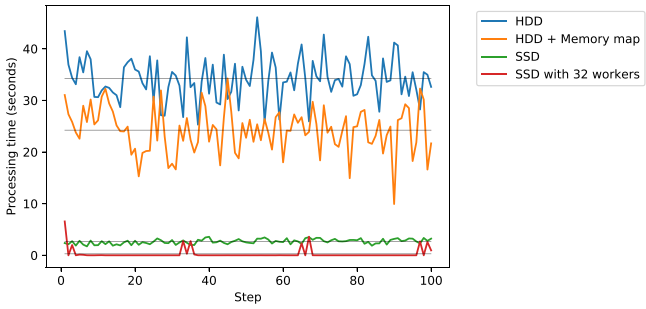


Fig. 9. Data loading efficiency in different situations.

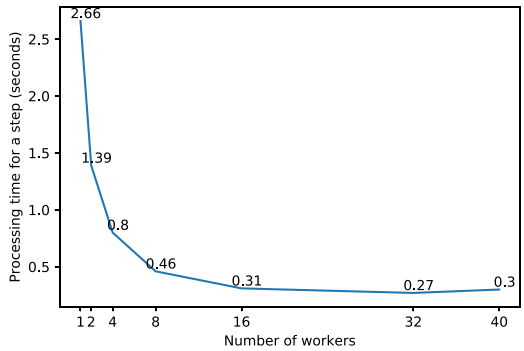


Fig. 10. Data loading time with different numbers of workers.

Note that for each time step, what is really desired is a tiny data chunk of the full-length utterance, and it is not necessary to read the entire utterance. Virtual memory techniques can be utilized to treat the I/O as routine memory accesses. This approach, known as memory mapping, allows a part of the virtual address space to be logically associated with the file. With these implementations, it can be seen from Fig. 9 that a memory map reduces data loading time in the HDD.

However, the I/O bottleneck still exists. Hence, our audio files were moved onto a Solid State Drive (SSD). It can be seen that the data loading efficiency is significantly improved.

The number of workers was further increased from 1 to 32. With 32 workers, the data loader produces 32 batches of training samples in parallel and then pushes these data into the queue. Therefore, for the following 31 steps, the data loading time is almost zero. Fig. 10 shows the trend of data loading efficiency with respect to the number of workers. As the number of workers increases, the data loading efficiency improves. Our CPU has 40 cores, and the best efficiency performance is achieved with 32 workers, which is slightly smaller than the number of CPU cores.

2) Training Efficiency: Here, the training efficiency was demonstrated using a server with two Intel Xeon E6-2630 CPUs, four NVIDIA GTX 1080Ti GPUs, and one Samsung 860 PRO SSD. The pooled Voxceleb dataset was adopted as the training corpus. The number of audio files processed per second were measured as our efficiency metric. From Table XIII, it can be observed that our optimized system processes 882 audio files per second in the network training when four GPUs are used. Moreover, the data preparation step (T1) is much faster than

TABLE XIII
TRAINING EFFICIENCY IN DIFFERENT SITUATIONS. BS DENOTES BATCH SIZE, T1 REPRESENTS DATA-PREPARATION STEP, AND T2 REPRESENTS DNN COMPUTATION STEP

BS	#Workers	#GPUs	Processing time (s)			Speed (files/s)
			Total	T1	T2	
64	1	1	0.750	0.589	0.160	85.3
64	16	1	0.172	0.004	0.168	372.1
64	32	1	0.182	0.004	0.178	351.6
128	3	2	0.308	0.006	0.302	423.8
256	32	4	**0.290**	**0.012**	**0.278**	**882.7**

TABLE XIV
RUNTIME PERFORMANCE IN THE TEST PHASE WITH A GPU. T1 REPRESENTS THE DATA PREPARATION STEP AND T2 THE DNN COMPUTATION STEP

Duration	Processing time (s)			Real-time factor		
	Total	T1	T2	Total	T1	T2
8 seconds	0.028	0.0168	0.011	0.0035	0.0021	0.0013
450 seconds	3.258	3.245	0.013	0.0072	0.0072	2e-5

the network training step (T2), which enables the GPU to run continuously.

3) Test Efficiency: Here, the testing phase efficiency was demonstrated using the same server, but only employing one CPU worker and one GPU to process test files. One hundred audio files were randomly selected from the SITW training data (the pooled VoxCeleb1 and VoxCeleb2 datasets) and the SITW evaluation data. The average duration of Voxceleb data is approximately 8 seconds and the average duration of SITW data is approximately 450 seconds. It was desired, however, to evaluate the test efficiency under different duration conditions.

From Table XIV, it can be observed that the network inference time (T2) in the proposed approach is stable as the test duration increases. Specifically, the network inference time is increased from 0.011 to 0.013 when the test duration is increased from 8 to 450 seconds. This benefit comes from the high parallelization of convolutional operations on the GPU device. The data processing time (T1) takes the most time especially for the long-duration speech utterance. Finally, the overall real-time factor is lower than 0.01 when network inference is performed on the GPU.

VI. CONCLUSIONS

In this paper, efforts were concentrated on two core issues related to directly modeling an utterance-level based DNN for speaker and language recognition: (i) how to prepare the variable-length inputs for effective and efficient DNN training, and (ii) how to aggregate the variable-length sequences into utterance-level representations.

First, regarding the data-preparation procedure, a novel data loader is proposed to generate variable-length samples in mini-batches for efficient DNN training. The data loader acts as a bridge between the training utterances and the network, and to make full use of every random segments of the training utterances. Our proposed implementation is flexible, and it produces batch-wise variable-length training samples on-the-fly. Furthermore, the data loader is equipped with an online data-augmentation strategy. The experimental results show the efficacy of the batch-wise variable-length training and online data-augmentation techniques.

Second, on the utterance-level aggregation, an LDE layer is introduced by integrating the traditional dictionary learning procedure into the deep speaker and language recognition system. The LDE layer acts as an enhanced pooling layer and is placed on top of the front-end local pattern extractor. It accepts a variable-length sequence and outputs a single super-vector-sized utterance-level representation. The parameters in the LDE layer are learnable so that it is suitable for supervised DNN learning under a unified loss function. The experimental results show the effectiveness of the LDE layer over TAP and SAP layers.

To summarize, the utterance-level deep speaker and language recognition system is explored from "producer" (the data loader) and "consumer" (the LDE layer) perspectives. The data loader efficiently produces variable-length training samples on the fly, and the LDE layer aggregates discriminative utterance-level representations from the random augmented variable-length input. The entire pipeline forms a new training scheme, and the system exhibits comparable performance with the state-of-the-art techniques.

ACKNOWLEDGMENT

This research was carried out when the first author was interning at Duke Kunshan University in 2018 and 2019.

REFERENCES

[1] W. M. Campbell, D. E. Sturim, and D. A. Reynolds, "Support vector machines using GMM supervectors for speaker verification," *IEEE Signal Process. Lett.*, vol. 13, no. 5, pp. 308–311, May 2006.
[2] A. D. Dileep and C. C. Sekhar, "GMM-based intermediate matching kernel for classification of varying length patterns of long duration speech using support vector machines," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 8, pp. 1421–1432, Aug. 2014.
[3] T. Kinnunen and H. Li, "An overview of text-independent speaker recognition: From features to supervectors," *Speech Commun.*, vol. 52, no. 1, pp. 12–40, 2010.
[4] J. H. L. Hansen and T. Hasan, "Speaker recognition by machines and humans: A tutorial review," *IEEE Signal Process. Mag.*, vol. 32, no. 6, pp. 74–99, Nov. 2015.
[5] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 19, no. 4, pp. 788–798, May 2011.
[6] E. Variani, X. Lei, E. McDermott, I. L. Moreno, and J. Gonzalez-Dominguez, "Deep neural networks for small footprint text-dependent speaker verification," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2014, pp. 4052–4056.
[7] I. Lopez-Moreno, J. Gonzalez-Dominguez, O. Plchot, D. Martinez, J. Gonzalez-Rodriguez, and P. Moreno, "Automatic language identification using deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2014, pp. 5337–5341.
[8] J. Gonzalez-Dominguez, I. Lopez-Moreno, H. Sak, J. Gonzalez-Rodriguez, and P. J. Moreno, "Automatic language identification using long short-term memory recurrent neural networks," in *Proc. INTERSPEECH*, 2014, pp. 2155–2159.
[9] R. Li, S. H. Mallidi, L. Burget, O. Plchot, and N. Dehak, "Exploiting hidden-layer responses of deep neural networks for language recognition," in *Proc. INTERSPEECH*, 2016, pp. 3265–3269.

[10] M. Tkachenko, A. Yamshinin, N. Lyubimov, M. Kotov, and M. Nastasenko, "Language identification using time delay neural network d-vector on short utterances," in *Proc. Int. Conf. Speech Comput.*, 2016, pp. 443–449.
[11] N. Smith and M. Gales, "Speech recognition using SVMs," in *Proc. Advances Neural Inf. Process. Syst.*, 2001, pp. 1197–1204.
[12] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, "X-vectors: Robust DNN embeddings for speaker recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2018, pp. 5329–5333.
[13] D. Povey *et al.*, "The kaldi speech recognition toolkit," in *Proc. Workshop Autom. Speech Recognit. Understanding*, 2011, pp. 1–4.
[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 770–778.
[15] A. Waibel, T. Hanazawa, G. E. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Trans. Acoust. Speech, Signal Process.*, vol. 37, no. 3, pp. 328–339, Mar. 1989.
[16] D. Snyder, P. Ghahremani, D. Povey, D. Garcia-Romero, Y. Carmiel, and S. Khudanpur, "Deep neural network-based speaker embeddings for end-to-end speaker verification," in *Proc. IEEE Spoken Lang. Technol. Workshop*, 2017, pp. 165–170.
[17] C. Li *et al.*, "Deep speaker: an end-to-end neural speaker embedding system," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1705.02304
[18] W. Cai, J. Chen, and M. Li, "Exploring the encoding layer and loss function in end-to-end speaker and language recognition system," in *Proc. ODYSSEY*, 2018, pp. 74–81.
[19] W. Cai, Z. Cai, X. Zhang, and M. Li, "A novel learnable dictionary encoding layer for end-to-end language identification," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2018, pp. 5189–5193.
[20] D. Reynolds, "Experimental evaluation of features for robust speaker identification," *IEEE Trans. Speech Audio Process.*, vol. 2, no. 4, pp. 639–643, Oct. 1994.
[21] W. Cai, Z. Cai, W. Liu, X. Wang, and M. Li, "Insights into end-to-end learning scheme for language identification," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2018, pp. 5209–5213.
[22] H. Bredin, "Tristounet: Triplet loss for speaker turn embedding," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2017, pp. 5430–5434.
[23] W. Cai, D. Cai, S. Huang, and M. Li, "Utterance-level end-to-end language identification using attention-based CNN-BLSTM," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2019, pp. 5991–5999.
[24] S. J. D. Prince and J. H. Elder, "Probabilistic linear discriminant analysis for inferences about identity," in *Proc. IEEE 11th Int. Conf. Comput. Vision*, 2007, pp. 1–8.
[25] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A discriminative feature learning approach for deep face recognition," in *Proc. Eur. Conf. Comput. Vision*, 2016, pp. 499–515.
[26] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "Sphereface: Deep hypersphere embedding for face recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2017, pp. 212–220.
[27] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, "Audio augmentation for speech recognition," in *Proc. INTERSPEECH*, 2015, pp. 3586–3589.
[28] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technologies*, 2016, pp. 1480–1489.
[29] G. Bhattacharya, J. Alam, and P. Kenny, "Deep speaker embeddings for short-duration speaker verification," in *Proc. Interspeech*, 2017, pp. 1517–1521.
[30] F. A. R. R. Chowdhury, Q. Wang, I. L. Moreno, and L. Wan, "Attention-based models for text-dependent speaker verification," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2018, pp. 5359–5363.
[31] H. Zhang, J. Xue, and K. Dana, "Deep TEN: Texture encoding network," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2017, pp. 708–717.
[32] A. Vaswani *et al.*, "Attention is All you Need," in *Proc. Advances Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.
[33] A. F. Martin and A. N. Le, "NIST 2007 language recognition evaluation," in *Proc. ODYSSEY*, 2008.
[34] Z. Tang, D. Wang, Y. Chen, and Q. Chen, "AP17-OLR challenge: Data, plan, and baseline," in *Proc. Asia-Pacific Signal Inf. Process. Assoc. Annu. Summit Conf.*, 2017, pp. 749–753.
[35] M. McLaren, L. Ferrer, D. Castan, and A. Lawson, "The speakers in the wild (SITW) speaker recognition database," in *Proc. INTERSPEECH*, 2016, pp. 818–822.
[36] S. O. Sadjadi *et al.*, "The 2016 NIST speaker recognition evaluation," in *Proc. Interspeech*, 2017, pp. 1353–1357.
[37] "Kaldi SITW x-vector recipe," 2019. [Online]. Available: https://github.com/kaldi-asr/kaldi/tree/master/egs/sitw/v2
[38] G. Gelly, J. L. Gauvain, V. B. Le, and A. Messaoudi, "A divide-and-conquer approach for language identification based on recurrent neural networks," in *Proc. INTERSPEECH*, 2016, pp. 3231–3235.
[39] W. Geng *et al.*, "End-to-end language identification using attention-based recurrent neural networks," in *Proc. INTERSPEECH*, 2016, pp. 2944–2948.
[40] H. Zeinali, L. Burget, J. Rohdin, T. Stafylakis, and J. H. Cernocky, "How to improve your speaker embeddings extractor in generic toolkits," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2019, pp. 6141–6145.
[41] M. Hajibabaei and D. Dai, "Unified hypersphere embedding for speaker recognition," *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1807.08312
[42] Y. Liu, L. He, J. Liu, and M. T. Johnson, "Speaker embedding extraction with phonetic information," in *Proc. INTERSPEECH*, 2018, pp. 2247–2251.
[43] T. Zhou, Y. Zhao, J. Li, Y. Gong, and J. Wu, "CNN with phonetic attention for text-independent speaker verification," in *Proc. Autom. Speech Recognit. Understanding Workshop*, 2019, pp. 718–725.
[44] M. Jin *et al.*, "LID-Senones and their statistics for language identification," *IEEE/ACM Trans. Audio, Speech Lang. Process.*, vol. 26, no. 1, pp. 171–183, Jan. 2018.
[45] "Kaldi SRE16 x-vector recipe," 2019. [Online]. Available: https://github.com/kaldi-asr/kaldi/tree/master/egs/sre16/v2

Weicheng Cai (Student Member, IEEE) received the M.S. degree in electrical engineering in 2016 from Sun Yat-sen University, Guangzhou, China, where he has been working toward the Ph.D. degree in computer science since August 2016. His research interests focus on machine learning and its applications on speech modeling, including speaker recognition, language identification, and anti-spoofing.

Jinkun Chen received the B.E. degree in software engineering and the M.E. degree in electronic and communication engineering from Sun Yat-sen University, Guangzhou, China, in 2016 and 2018, respectively.

Jun Zhang received the B.S. degree in physics from Fudan University, Shanghai, China, in 1992, and the Ph.D. degree in optics from Shanghai Jiaotong University, Shanghai, in 2003. He is currently a Professor with the School of Electronics and Information Technology, Sun Yat-sen University. His research interests include biomedical imaging particularly the new non-invasive and minimally-invasive optical imaging technologies and translation of these techniques to address clinical problems.

Ming Li (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in May 2013. He is currently an Associate Professor of Electrical and Computer Engineering with Duke Kunshan University, Suzhou, China, and an Adjunct Professor with the School of Computer Science, Wuhan University, Wuhan, China. His research interests are in the areas of audio, speech, and language processing as well as multimodal behavior signal analysis and interpretation.