ROBUST SPEAKER RECOGNITION USING DEEP NEURAL NETWORKS

WEIWEI LIN

PhD

The Hong Kong Polytechnic University

2020

The Hong Kong Polytechnic University

Department of Electronic and Information Engineering

ROBUST SPEAKER RECOGNITION USING DEEP NEURAL NETWORKS

WEIWEI LIN

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

April 2020

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

(Signed)

Lin Weiwei (Name of student)

Abstract

ROBUST SPEAKER RECOGNITION USING DEEP NEURAL NETWORKS

Speaker recognition refers to recognizing a person using his/her voice. Although state-of-the-art speaker recognition systems have shown remarkable performance, there are still some unsolved problems. Firstly, speaker recognition systems' performance degrades significantly when training data and test data have domain mismatch. Domain mismatch is prevalent and is expected to happen during system deployment. This could occur when the new environment has some specific noise or involves speakers speaking different languages than training speakers. Directly using the existing system in these situations could result in poor performance. Secondly, the statistics pooling layer in state-of-the-art systems does not have rich representation power to capture the complex characteristics of frame-level features. The statistics pooling layer only uses the mean and standard deviation of frame-level features. However, mean and standard deviation are insufficient for summarizing a complex distribution. Thirdly, state-of-the-art systems still rely on a PLDA backend, which makes deployment difficult and hinders the potential of the DNN frontend.

This thesis proposes several solutions to the problems mentioned above. For reducing the domain mismatch, this thesis proposes adaptation methods for both DNN frontend and PLDA backend. The proposed backend adaptation uses an auto-encoder to minimize the domain mismatch between i-vectors, while the frontend adaptation focuses on producing speaker embedding that is both discriminative and domain-invariant. Using the proposed adaptation framework, we achieve an EER of 8.69% and 7.95% in NIST SRE 2016 and 2018, respectively, which are significantly better than the previously proposed DNN adaptation methods. For better frame-level

information aggregation in the DNN, this thesis proposes an attention-based statistics pooling method, which uses an expectation–maximization (EM) like algorithm to produce multiple means and standard deviations for summarizing frame-level features distribution. Applying the proposed attention mechanism to a 121-layer Densenet, we achieve an EER of 1.1% in VoxCeleb1 and an EER of 4.77% in the VOiCES 2019 evaluation set. For facilitating end-to-end speaker recognition, this thesis proposes several strategies to eliminate the need of a backend model. Experiments on NIST SRE 2016 and 2018 show that with the proposed strategies, the DNN can achieve state-of-the-art performance using simple cosine similarity and requires only half of the computational cost of the x-vector network.

ACKNOWLEDGMENTS

I would like to express my deep gratitude to my supervisor Dr. M. W. Mak, whose patience, expertise and encouragement is the reasons this work can be done. Dr. M. W. Mak is not only an expert in the field of machine learning, bioinformatics, software engineering, but also a excellent teacher in imparting passion and insight to his students in often very challenging research works.

Besides, I would like to express appreciation to department of Electronic and Information Engineering for offering me financial support during the writing up of this thesis. I would also like to thank my course teachers in my PhD study. They have been a huge source of help, wisdom and inspiration to me.

Last but not lease, I am much indebted to my parents for their years' support and understanding.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AHC Agglomerative hierarchical clustering.

ASP Attentive statistics pooling.

ASR Automatic Speech Recognition.

BN Batch normalization.

BP Backpropagation.

CMVN Cepstral mean-variance normalization.

CNN Convolutional neural network.

CORAL Correlation alignment.

DA Domain adaptation.

DAD Domain-invariant autoencoder.

DCF Decision cost function.

DNN Deep neural network.

EER Equal error rate.

EM Expectation-maximization.

FC Fully-connected.

FGE Fast geometry emsemble.

GMac Giga multiply–accumulate operation.

GMM Gaussian mixture model.

IDVC Inter-dataset variability compensation.

JFA Joint factor analysis.

L-BFGS Limited-memory Broyden—Fletcher—Goldfarb—Shanno.

LDA Linear discriminative analysis.

LSGAN Least square GAN.

MAP Maximum a Posteriori.

MCMC Markov Chain Monte Carlo.

MFCC Mel-frequency cepstral coefficient.

MMD Maximum mean discrepancy.

MRP Mixture representation pooling.

NAD Nuisance attribute autoencoder.

NIST National Institute of Standards and Technology.

PCA Principal component analysis.

PLDA Probabilistic linear discriminative Analysis.

RBF Radial basis function.

SGD Stochastic gradient descent.

SN Score normalization.

SRE Speaker recognition evaluation.

SVM Support vector machine.

t-SNE T-distributed stochastic neighbor embedding.

TDNNs Time delay neural networks.

TI-SV Text-independent SV.

TV Total variability.

UBM Universal background model.

UP Uncertainty propagation.

VAD Voice activity detection.

VB Variational Bayes.

VI Variational inference.

WCCN Within class covariance normalization.

WGAN Wasserstein GAN.

AUTHOR'S PUBLICATIONS

Journal Papers

1. **W.W. Lin**, M.W. Mak, N. Li, D. Su and D. Yu, "A Framework for Adapting DNN Speaker Embedding Across Languages", *Submitted to IEEE/ACM Transactions on Audio, Speech and Language Processing, accepted with minor changes.*

2. **W.W. Lin**, M.W. Mak and J.T. Chien, "Multi-source I-vectors Domain Adaptation using Maximum Mean Discrepancy Based Autoencoders", *IEEE/ACM Transactions on Audio, Speech and Language Processing*, vol. 26, no. 12, pp. 2412–2422, Dec. 2018.

3. N. Li, M.W. Mak, **W.W. Lin** and J.T. Chien, "Discriminative Subspace Modeling of SNR and Duration Variabilities for Robust Speaker Verification", *Computer Speech and Language*, vol. 45, pp. 87–103, 2017.

4. **W.W. Lin**, M.W. Mak, and J.T. Chien, "Fast Scoring for PLDA with Uncertainty Propagation via I-vector Grouping", *Computer Speech and Language*, vol. 45, pp. 503–515, 2017.

Conference Papers

1. **W.W. Lin**, M.W. Mak and Y. Lu, "Learning Mixture Representation for Deep Speaker Embedding Using Attention", *Speaker Odyssey 2020.*

2. **W.W. Lin**, M.W. Mak, N. Li, D. Su, D. Yu, "Multi-level Deep Neural Network Adaptation for Speaker Verification using MMD and Consistency Regularization", *ICASSP'20*, Barcelona, May 2020.

3. **W.W. Lin**, M.W. Mak, Y.Z. Tu, and J.T. Chien, "Semi-supervised Nuisance-Attribute Networks for Domain Adaptation", *ICASSP'19, Brighton*, May, 2019, pp. 6236–6240.

4. **W.W. Lin**, M.W. Mak, L.X. Li, and J.T. Chien, "Reducing Domain Mismatch by Maximum Mean Discrepancy Based Autoencoders", *Speaker Odyssey 2018*, June 2018, Les Sables d'Olonne, France, pp. 162–167.

Chapter 1

BACKGROUND

1.1 Speaker Recognition

Speaker recognition refers to recognizing a person using his/her voice. Due to both physical differences and style of speaking, speech is a unique biometric trait. Speaker recognition is widely used in commercial products such as Apple Homepod, Amazon Echo, and Nuance Gatekeeper.

A major application of speaker recognition is identity authentication. For example, in countries with a high elderly population, the identities of pensioners must be verified before any pension transactions can be proceeded. However, elderly people are unlikely to travel to the banks often, which makes identity authentication difficult. With speaker recognition technology, a simple telephone call can identify the claimer's identity. Another important application of speaker recognition is speaker diarization, also knowns as "who spoken when", where the objective is to extract the participant's identity and speech from TV broadcasts or teleconference meetings.

Figure 1.1 shows the pipeline of a typical speaker verification system.

- Feature extraction: The raw speech signal is not amenable to machine learning algorithms. Normally, time-domain signals are converted to spectra. Then, filter-bank features or Mel-frequency cepstral coefficients (MFCCs) are computed from the spectra.

- Speaker embedding network: A speaker embedding network takes MFCCs or

filter-bank features as input and produces a compact vector representation of the speaker information in the speech. The network is trained with speaker labels to minimize the cross-entropy loss.

- Backend probabilistic linear discriminant analysis (PLDA) model: Although DNN speaker embeddings are already speaker discriminative, researchers found that the embeddings can be further improved by using a probabilistic linear discriminant analysis (PLDA) model at the backend. The PLDA model is trained using the speaker embeddings extracted from full-length utterances and the corresponding speaker labels.

- PLDA scoring: Given the embeddings of two utterances, the PLDA model outputs a PLDA score, which is a likelihood radio between the hypothesis that the two utterances (embeddings) are from the same speaker and that the two utterances are from different speakers.

1.2 Evaluation Metrics for Speaker Verification

The main evaluation metrics used in this thesis are equal error rate (EER) and minimum detection cost function (minDCF).

- Equal error rate (EER): EER refers to the point where the false rejection rate (FRR) and the false acceptance rate (FAR) are equal, where FRR and FAR are defined as

$$\text{FRR} = \frac{\text{No. of true-speakers rejected}}{\text{Total no. of true-speaker trials}} \tag{1.1}$$

$$\text{FAR} = \frac{\text{No. of impostors accepted}}{\text{Total no. of impostor attempts}}. \tag{1.2}$$

- Minimum detection cost function (minDCF): minDCF refers to the minimum

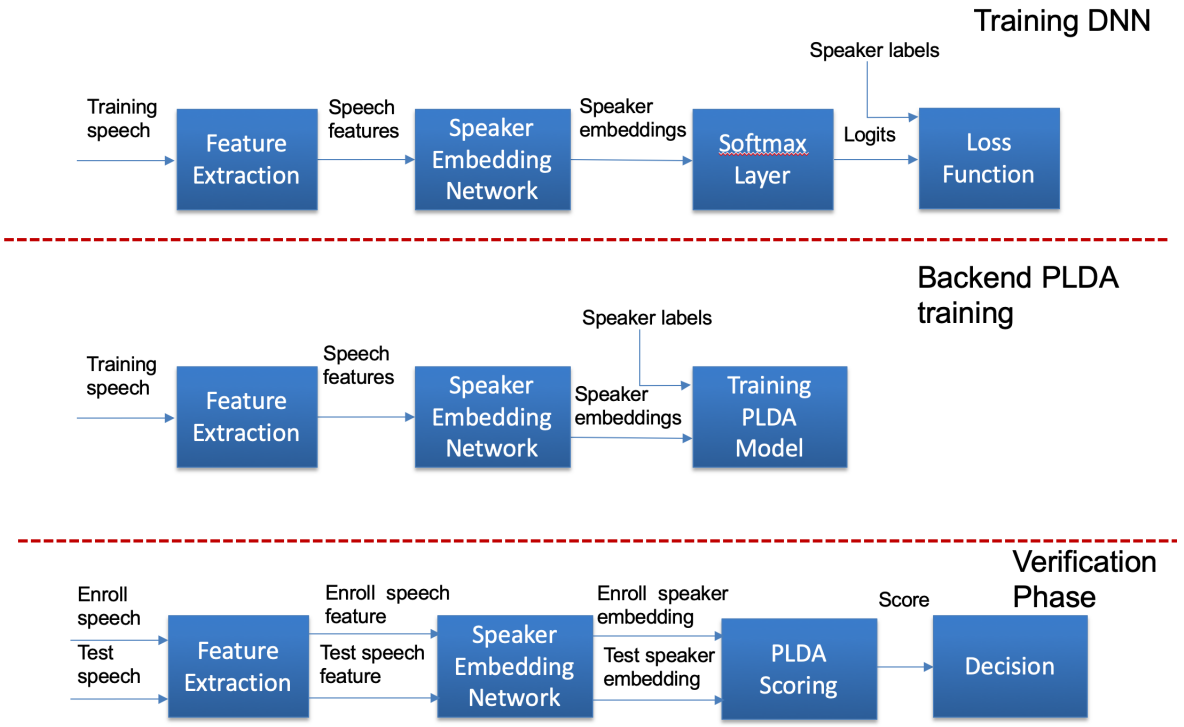

Figure 1.1: A typical speaker verification system training and test pipeline.

detection cost for a weighted sum of the false acceptance rate and the false rejection rate. Typically, the false acceptance rate has a higher weight.

1.3 Key Issues in Speaker Recognition

The i-vector/PLDA framework (see Chapter 2) had dominated the field of speaker verification (SV) for many years. Recently, deep neural network (DNN) based methods have shown remarkable results [2]. However, many important problems remain unsolved.

- Domain Adaptation: The success of machine learning relies on the assumption that training data and test data are sampled from the same distribution [3, 4]. In practice, a lot of factors can undermine this assumption. This is especially

the case when we want to deploy an existing system to a new environment, where the data have different properties than the training data. For speaker verification, this could happen when the new environment has some specific noise and channel conditions or involves speakers speaking different languages than the training speakers. Directly using the existing system in these situations could result in poor performance.

- End-to-end SV: State-of-the-art SV requires training a PLDA backend to achieve excellent performance. An end-to-end approach using an integrated neural network is more attractive in several aspects. Firstly, hyperparameter optimization is easier in an end-to-end system. In an x-vector system, the DNN and the backend are optimized separately, which complicates the hyperparameter search as validation has to be done for the network and backend separately. Secondly, although training the backend itself is fairly fast when compared with training the network, the extraction process can be time-consuming. Besides, it is not clear which part of the dataset should be used for backend training. Thirdly, an end-to-end system is easier to deploy and debug.

- Aggregation of Frame-level Information: The process of converting an acoustic sequence to a fixed-dimensional representation is referred to as statistics pooling in the literature [5]. It has been shown that the statistics pooling layer has a significant impact on speaker embeddings' performance. In [5], it was shown that using both the mean and standard deviation of frame-level features has significant improvement over using the mean alone. Although attentive statistics pooling shows promising results [6], we believe that weighting frame-level features is not enough to produce ideal speaker representation. The statistics pooling layer needs to produce a good statistical summary of the frame-level features. However, means and standard deviations are insufficient for summarizing a complex distribution.

1.4 Key Contributions

This thesis proposes several solutions to the aforementioned problems:

- *MMD-based Domain Adaptation.* MMD (maximum mean discrepancy [7]) is a popular measure for quantifying the distance between two probabilisty distributions. This thesis proposes using an MMD-based autoencoder to adapt i-vectors and a powerful MMD-based framework for adapting deep speaker embeddings.

- *Strategies Empowering end-to-end SV.* This thesis proposes several strategies to modify the x-vector system, making end-to-end SV possible. The strategies include: (1) applying a sampling scheme to produce diverse training samples by randomly splicing several speech segments from the original utterances, (2) adding extra convolutional layers designed to reduce the temporal resolution so as to save computational cost, and (3) introducing a mask-pooling layer that augments the utterance-level representation in the speaker embedding network by randomly masking the frames-level activations before statistics pooling.

- *Mixture Representation Pooling.* This thesis proposes a novel statistics pooling method that can produce more descriptive statistics through a mixture representation. The method is inspired by the expectation-maximization (EM) algorithm in Gaussian mixture models (GMMs). However, unlike the GMMs, the mixture assignments are given by an attention mechanism instead of the Euclidean distances between frame-level features and explicit centers.

Chapter 2

THE CLASSIC I-VECTOR/PLDA FRAMEWORK

2.1 I-vector

A milestone in the speaker recognition research is the discovery that speaker information within an utterance of arbitrary duration can be represented by a low dimension vector called i-vector [8]. Different from joint factor analysis, the i-vector framework does not attempt to suppress channel variation. Rather, it views all utterances as if they come from different speakers and maps them from the supervector space to a total variability space. Because each utterance is represented by a single point in the i-vector space, we can suppress unwanted variability by classical statistic methods such as LDA or its probabilistic counterpart PLDA.

2.1.1 Generative Model

I-vector [8] is based on the following factor analysis model:

$$\boldsymbol{\beta} = \mathbf{m} + \mathbf{T}\boldsymbol{\eta} \qquad \boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{2.1}$$

where $\boldsymbol{\beta}$ is a speaker-dependent GMM-supervector, $\mathbf{m}$ is formed by stacking the mean vectors of a Universal Background Model (UBM), $\boldsymbol{\eta}$ is a total-variability factor that has a standard normal prior distribution, and $\mathbf{T}$ is a low-rank total variability matrix mapping $\boldsymbol{\eta}$ from the total variability space to the supervector space. The i-vector is the maximum-a-posteriori (MAP) estimate of $\boldsymbol{\eta}$, which we denote as $\boldsymbol{\omega}$.

2.1.2 I-vector Extraction

Let the D-dimensional acoustic vectors of an utterance be

$$\mathcal{O} = \{\mathbf{o}_1, \ldots, \mathbf{o}_T\},$$

where T is the number of frames in the utterance. Acoustic feature vectors are usually modeled by a mixture of Gaussian densities. Assuming that a UBM with C mixtures is given, the mixture weights, mean vectors, and covariance matrices are denoted as λ_c, $\mathbf{m}_c$, and $\mathbf{\Sigma}_c$, respectively, where $c = 1, \ldots, C$. An $R \times 1$ vector $\boldsymbol{\eta}$ and a $D \times R$ matrix $\mathbf{T}_c$ are introduced to account for the inter-utterance difference in the c-th mixture. If the alignments of $\{\mathbf{o}_t\}_{t=1}^{T}$ with C mixture components are explicitly stated, then the log-likelihood of an utterance can be written as:

$$\mathcal{L}(\mathcal{O}) = \sum_c \left(n_c \ln \frac{1}{(2\pi)^{D/2}|\mathbf{\Sigma}_c|^{1/2}} - \frac{1}{2} \sum_{t \in \mathcal{T}_c} (\mathbf{o}_t - \mathbf{T}_c\boldsymbol{\eta} - \mathbf{m}_c)^{\mathsf{T}} \mathbf{\Sigma}_c^{-1} (\mathbf{o}_t - \mathbf{T}_c\boldsymbol{\eta} - \mathbf{m}_c) \right), \tag{2.2}$$

where $\mathcal{T}_c$ contains the indexes of the frames that are aligned to mixture c and n_c is the size of $\mathcal{T}_c$. Because the alignments are usually not given, their probabilistic estimates are used instead. Let C_t denotes which of the c mixtures is responsible for generating $\mathbf{o}_t$. By making use of the Bayes rule, we have the posterior probability of mixture components c given $\mathbf{o}_t$

$$\Pr(C_t = c|\mathbf{o}_t) = \frac{\lambda_c \mathcal{N}(\mathbf{o}_t; \mathbf{m}_c, \mathbf{\Sigma}_c)}{\sum_{c=1}^{C} \lambda_c \mathcal{N}(\mathbf{o}_t; \mathbf{m}_c, \mathbf{\Sigma}_c)}, \quad c = 1, \ldots, C. \tag{2.3}$$

We denote this probability as $\gamma_t(c)$ in the following for simplicity. Now the log-likelihood can be rewritten as

$$\mathcal{L}(\mathcal{O}) = \sum_c \Bigg(\sum_{t \in \mathcal{T}_c} \gamma_t(c) \ln \frac{1}{(2\pi)^{D/2} |\mathbf{\Sigma}_c|^{1/2}} - \frac{1}{2} \sum_{t \in \mathcal{T}_c} \gamma_t(c) [\mathbf{o}_t - \mathbf{T}_c \boldsymbol{\eta} - \mathbf{m}_c]^\mathsf{T} \mathbf{\Sigma}_c^{-1} [\mathbf{o}_t - \mathbf{T}_c \boldsymbol{\eta} - \mathbf{m}_c] \Bigg). \quad (2.4)$$

In order to estimate the i-vector, we need to compute the sufficient statistics:

$$N_c = \sum_{t=1}^{T} \gamma_t(c) \quad (2.5a)$$

$$\tilde{\boldsymbol{F}}_c = \sum_{t=1}^{T} \gamma_t(c)(\mathbf{o}_t - \mathbf{m}_c). \quad (2.5b)$$

The posterior covariance and posterior mean associated with the utterance is given by:

$$\mathrm{Cov}(\boldsymbol{\eta}, \boldsymbol{\eta} | \mathcal{O}) = \mathbf{L}^{-1}, \quad (2.6)$$

and

$$\boldsymbol{\omega} = \mathbf{L}^{-1} \sum_c \mathbf{T}_c^\mathsf{T} {\mathbf{\Sigma}_c}^{-1} \tilde{\boldsymbol{F}}_c, \quad (2.7)$$

respectively, where

$$\mathbf{L} = \mathbf{I} + \sum_c \mathbf{T}_c^\mathsf{T} \mathbf{\Sigma}_c^{-1} N_c \mathbf{T}_c \quad (2.8)$$

is a precision matrix [9] and $\mathbf{I}$ is the identity matrix. Eq. 2.7 and Eq. 2.8 can be written in a more compact form

$$\boldsymbol{\omega} = \mathbf{L}^{-1} \mathbf{T}^\mathsf{T} (\mathbf{\Sigma}^{(b)})^{-1} \tilde{\boldsymbol{F}} \quad (2.9)$$

$$\mathbf{L} = \mathbf{I} + \mathbf{T}^\mathsf{T} (\mathbf{\Sigma}^{(b)})^{-1} \mathbf{N} \mathbf{T}, \quad (2.10)$$

where $\mathbf{N}$ is a $CD \times CD$ diagonal matrix with diagonal blocks $N_c\mathbf{I}$, and $\tilde{\boldsymbol{F}}$ is $CD \times 1$ supervector formed by stacking $\tilde{\boldsymbol{F}}_c$, $c = 1, \ldots, C$. $\boldsymbol{\Sigma}^{(b)}$ is composed of the covariance matrices of the UBM, i.e., $\boldsymbol{\Sigma}^{(b)} = \text{diag}\{\boldsymbol{\Sigma}_1^{(b)}, \ldots, \boldsymbol{\Sigma}_C^{(b)}\}$, which represents the variability missed by the $CD \times R$ total variability matrix $\mathbf{T}$.

2.1.3 I-vector Pre-processing

PLDA assumes that i-vectors follow a Gaussian distribution. Therefore, proper pre-processing of i-vectors is needed before PLDA modeling. Garcia-Romero and Espy-Wilson [10] proposed to apply length normalization to gaussianize i-vectors, which has proved to be very effective in practice. An alternative way is to assume that i-vectors follow a non-Gaussian distribute. However, the computation of the model is more expensive [11].

The Gaussianization process includes two steps. In the first step, i-vectors are subjected to whitening. It can be written as:

$$\boldsymbol{\omega}^{\mathsf{whiten}} = \mathbf{A}^{\mathsf{T}}(\boldsymbol{\omega} - \bar{\boldsymbol{\omega}}), \tag{2.11}$$

where $\bar{\boldsymbol{\omega}}$ is the mean of the training i-vectors, $\boldsymbol{\omega}^{\mathsf{whiten}}$ is the whitened i-vector and $\mathbf{A}$ is a transformation matrix, which is computed by applying Cholesky decomposition of the within-class covariance matrix of the training i-vectors. The second step of Gaussianization involves applying length normalization to the whitened i-vectors:

$$\boldsymbol{\omega}^{\mathsf{l\text{-}norm}} = \frac{\boldsymbol{\omega}^{\mathsf{whiten}}}{\|\boldsymbol{\omega}^{\mathsf{whiten}}\|}. \tag{2.12}$$

It was found that this simple non-linear transformation can significantly reduces the mismatch between the i-vectors in the training set and the test set [10].

The next step of pre-processing is session-variability compensation. This is done by applying linear discriminant analysis (LDA) to the length-normalized i-vectors. LDA

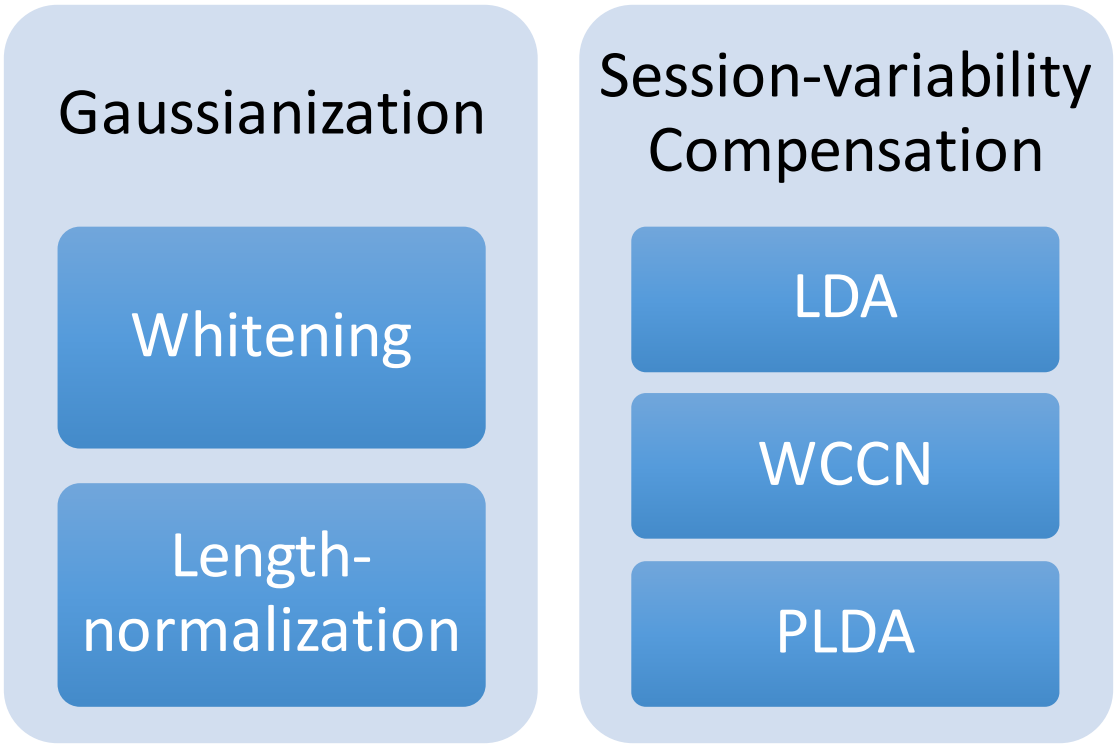

Figure 2.1: Pre-processing steps of i-vector/PLDA systems

finds a subspace that maximizes the between-class scatter while minimizes the within-class scatter [12]. Because LDA assumes that data follow a Gaussian distribution, it is necessary to apply Gaussianization to the i-vectors first. Let i be the speaker label and j indexes the session of a speaker. The between-class covariance matrix $\mathbf{S}_b$ and the within-class covariance matrix $\mathbf{S}_w$ are defined as:

$$\mathbf{S}_b = \frac{1}{I}\sum_i (\bar{\boldsymbol{\omega}}_i^{\text{l-norm}} - \bar{\boldsymbol{\omega}}^{\text{l-norm}})(\bar{\boldsymbol{\omega}}_i^{\text{l-norm}} - \bar{\boldsymbol{\omega}}^{\text{l-norm}})^{\mathsf{T}} \tag{2.13}$$

$$\mathbf{S}_w = \frac{1}{I}\sum_i \sum_j \frac{1}{N_i}(\boldsymbol{\omega}_{i,j}^{\text{l-norm}} - \bar{\boldsymbol{\omega}}_i^{\text{l-norm}})(\boldsymbol{\omega}_{i,j}^{\text{l-norm}} - \bar{\boldsymbol{\omega}}_i^{\text{l-norm}})^{\mathsf{T}}, \tag{2.14}$$

where $\bar{\boldsymbol{\omega}}_i^{\text{l-norm}}$ is the mean of length-normalized i-vectors of speaker i, $\bar{\boldsymbol{\omega}}^{\text{l-norm}}$ is the mean of all length-normalized i-vectors in the training set. The LDA projection is

found by solving the eigen-decomposition problem:

$$\mathbf{S}_b\mathbf{v} = \lambda\mathbf{S}_w\mathbf{v}. \tag{2.15}$$

The number of eigenvectors that we chose to compose the LDA projection matrix corresponds to the dimension of the LDA subspace. After LDA projection, we apply within-class covariance normalization (WCCN) [13]. The WCCN seeks to attenuate the high within-class variance in the LDA-projected vectors. We combine this sequence of projections and denote the resulting projection matrix as $\mathbf{P}$. Then the i-vector after the third step of pre-processing can be written as:

$$\mathbf{x} = \mathbf{P}\boldsymbol{\omega}^{\text{l-norm}}. \tag{2.16}$$

For simplicity in notation, we denote $\mathbf{x}$ as the i-vector after pre-processing in this thesis. We refer the Gaussianization and session-variability compensation together to as pre-processing.

2.2 PLDA Modeling

Because i-vectors contain both speaker and channel information, channel compensation in i-vector space is needed. Probabilistic linear discriminant analysis (PLDA) is one of such techniques. A PLDA model [14] is a generative model that uses low-dimensional latent variables to explain the observed i-vectors. Given an utterance of speaker i in session j, the PLDA model explains the variability of the corresponding i-vector $\mathbf{x}_{i,j}$ through the speaker factor $\mathbf{h}_i$, channel factor $\mathbf{z}_{i,j}$, and residue $\boldsymbol{\epsilon}_{i,j}$

$$\mathbf{x}_{i,j} = \boldsymbol{\mu} + \mathbf{V}\mathbf{h}_i + \mathbf{U}\mathbf{z}_{i,j} + \boldsymbol{\epsilon}_{i,j} \qquad \boldsymbol{\epsilon}_{i,j} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}), \tag{2.17}$$

where $\boldsymbol{\mu}$ is the global offset and the column vectors of $\mathbf{V}$ define the bases of the speaker subspace. The column vectors of $\mathbf{U}$ define the bases of the channel subspace. The prior of both $\mathbf{h}$ and $\mathbf{z}$ are assumed to follow a standard normal distribution, and $\boldsymbol{\epsilon}$ is assumed to follow a Gaussian distribution with zero mean and diagonal covariance matrix $\boldsymbol{\Sigma}$. The matrix ($\mathbf{V}\mathbf{V}^{\mathsf{T}}$) models between-speaker variability. The channel component $\mathbf{U}\mathbf{z}_{i,j} + \boldsymbol{\epsilon}_{i,j}$ models session variability. It changes from utterance to utterance. In practice, i-vectors are typically of low dimension. Therefore, we may assume that the covariance of channel ($\mathbf{U}\mathbf{U}^{\mathsf{T}}$) can be absorbed by the residual covariance $\boldsymbol{\Sigma}$ if the latter is a full covariance matrix. The modified model is

$$\mathbf{x}_{i,j} = \boldsymbol{\mu} + \mathbf{V}\mathbf{h}_i + \boldsymbol{\epsilon}_{i,j} \qquad \boldsymbol{\epsilon}_{i,j} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}). \tag{2.18}$$

The parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \mathbf{V}, \boldsymbol{\Sigma}\}$ of the model can be learnt via the EM algorithm over large amount of i-vectors.

Authentication is achieved by computing the ratio of two likelihoods, which is the likelihood of a test utterance and a target utterance generated by the same latent variable against the likelihood of the two utterances independently generated by two different latent variables. Given a test i-vector $\mathbf{x}_t$ and a target-speaker's i-vector $\mathbf{x}_s$, the log-likelihood ratio score is [10]

$$\log S_{LR}(\mathbf{x}_s, \mathbf{x}_t) = \frac{1}{2}\mathbf{x}_s^{\mathsf{T}}\mathbf{Q}\mathbf{x}_s + \mathbf{x}_s^{\mathsf{T}}\mathbf{P}\mathbf{x}_t + \frac{1}{2}\mathbf{x}_t^{\mathsf{T}}\mathbf{Q}\mathbf{x}_t + \text{const}, \tag{2.19}$$

where

$$\mathbf{Q} = \boldsymbol{\Sigma}_{tot}^{-1} - (\boldsymbol{\Sigma}_{tot} - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_{tot}^{-1}\boldsymbol{\Sigma}_{ac})^{-1} \tag{2.20}$$

$$\mathbf{P} = \boldsymbol{\Sigma}_{tot}^{-1}\boldsymbol{\Sigma}_{ac}(\boldsymbol{\Sigma}_{tot} - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_{tot}^{-1}\boldsymbol{\Sigma}_{ac})^{-1}, \tag{2.21}$$

where

$$\mathbf{\Sigma}_{ac} = \mathbf{V}\mathbf{V}^{\mathsf{T}} \tag{2.22}$$

$$\mathbf{\Sigma}_{tot} = \mathbf{V}\mathbf{V}^{\mathsf{T}} + \mathbf{\Sigma}. \tag{2.23}$$

Note that $\mathbf{Q}$ and $\mathbf{P}$ in Eq. 2.19 can be pre-computed before verification.

2.3 Coupling between I-Vector Extraction and PLDA Modeling

In the standard PLDA/I-vector framework, an utterance of arbitrary duration is mapped to an i-vector of fixed dimension. Then, in PLDA modeling, we ignore how i-vectors are extracted and pretend that they are generated by a PLDA model. This decoupling between i-vector extraction and PLDA modeling enables very fast authentication and performs very well when the durations of speech segments are long enough. However, the performance of this approach degrades severely when utterances are short. This is not surprising, because i-vectors are essentially a point estimate calculated by evaluating the posterior expectation of latent variables using Baum-Welch statistics extracted from the utterances. When the utterances are short, there is a huge amount of uncertainty associated with such estimates. In other words, the point estimates are no longer accurate and adequate for representing the speaker information incorporated in speech segments.

Kenny *et al.* [15] proposed using both posterior expectations and posterior covariance matrices to represent the speaker information in speech segments when utterances are very short. The posterior covariance matrices of the latent variable can be interpreted as the reliability of i-vectors estimates. We can also see from Eq. 2.8 that the shorter the utterance is, the larger this covariance matrix would be. To incorporate the uncertainty of i-vectors, the PLDA model is written as:

$$\mathbf{x}_{i,j} = \boldsymbol{\mu} + \mathbf{V}\mathbf{h}_i + \mathbf{U}_{i,j}\mathbf{z}_{i,j} + \boldsymbol{\epsilon}_{i,j}, \tag{2.24}$$

where $\mathbf{U}_{i,j}$ is the Cholesky decomposition of the posterior covariance matrix of the corresponding i-vector and $\mathbf{z}_{i,j}$ is a hidden variable assumed to have a standard Gaussian prior. $\mathbf{U}_{i,j}\mathbf{z}_{i,j}$ quantifies the deviation of the i-vector from its point estimate due to the uncertainty of the i-vector. When linear transformation $\mathbf{P}$ is used for pre-processing, we have:

$$\mathbf{U}\mathbf{U}^{\mathsf{T}} = \mathrm{cov}(\mathbf{P}\boldsymbol{\eta}, \mathbf{P}\boldsymbol{\eta}|\mathcal{O}) = \mathbf{P}\mathbf{L}^{-1}\mathbf{P}^{\mathsf{T}}, \tag{2.25}$$

where $\mathbf{L}$ is the posterior precision matrix in Eq. 2.8 and $\mathcal{O}$ is the set of acoustic vectors in the utterance. But if length-normalization is applied to the i-vectors, Eq. 2.25 cannot be applied. This is because length-normalization is not a linear transformation. As a result, the pre-processing step cannot be represented by a single transformation matrix $\mathbf{P}$. Nevertheless, Kenny *et al.* [15] suggested the following method to approximate $\mathbf{U}$:

$$\mathbf{U} \leftarrow \frac{\mathbf{U}}{\|\boldsymbol{\omega}^{\mathsf{whiten}}\|}. \tag{2.26}$$

2.4 EM Formulations for PLDA with Uncertainty Propagation

The expectation maximization (EM) algorithm is a general purpose algorithm for finding the maximum-likelihood solution of a generative model that involves latent variables. The EM algorithm defines a lower bound of the true likelihood. It alternately maximizes the lower-bound by updating the model parameters and the posterior distribution of latent variables [12].

Assuming that we have a set of pre-processed [10] i-vectors $\mathcal{X} = \{\mathbf{x}_{i,j}; i = 1, \ldots, I; j = 1, \ldots, H_i\}$, Eq. 2.24 can be written as [16]

$$\mathbf{x}_{i,j} = \boldsymbol{\mu} + [\mathbf{V} \quad \mathbf{U}_{i,j}] \begin{bmatrix} \mathbf{h}_i \\ \mathbf{z}_{i,j} \end{bmatrix} + \boldsymbol{\epsilon}_{i,j} = \boldsymbol{\mu} + \mathbf{B}_{i,j}\boldsymbol{\beta}_{i,j} + \boldsymbol{\epsilon}_{i,j}, \tag{2.27}$$

where $\mathbf{B}_{i,j} = [\mathbf{V}\ \ \mathbf{U}_{i,j}]$ and $\boldsymbol{\beta}_{i,j} = [\mathbf{h}_i^\mathsf{T}\ \ \mathbf{z}_{i,j}^\mathsf{T}]^\mathsf{T}$. The model parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \mathbf{V}, \boldsymbol{\Sigma}\}$ are estimated via the EM algorithm and matrix $\mathbf{U}_{i,j}$ can be estimated from the posterior covariance matrix of the i-vector $\mathbf{x}_{i,j}$ using Eq. 2.26.

2.4.1 E-Step

In the E-step, we want to compute the posterior distribution of latent variables $\mathbf{h}_i$ and $\mathbf{z}_{i,j}$. The posterior distribution of latent variable $\mathbf{h}_i$ can be written as [17]:

$$\begin{aligned}
p(\mathbf{h}_i|\mathbf{x}_{i,j}, \boldsymbol{\theta}) &\propto p(\mathbf{x}_{i,j}|\mathbf{h}_i, \boldsymbol{\theta})p(\mathbf{h}_i) \\
&= \int p(\mathbf{x}_{i,j}, \mathbf{z}_{i,j}|\mathbf{h}_i, \boldsymbol{\theta})p(\mathbf{h}_i)\mathrm{d}\mathbf{z}_{i,j} \\
&= \int p(\mathbf{x}_{i,j}|\mathbf{h}_i, \mathbf{z}_{i,j}, \boldsymbol{\theta})p(\mathbf{z}_{i,j})p(\mathbf{h}_i)\mathrm{d}\mathbf{z}_{i,j} \\
&= \int \mathcal{N}(\mathbf{x}_{i,j}|\boldsymbol{\mu} + \mathbf{V}\mathbf{h}_i + \mathbf{U}_{i,j}\mathbf{z}_{i,j}, \boldsymbol{\Sigma})\mathcal{N}(\mathbf{z}_{i,j}|\mathbf{0}, \mathbf{I})\mathcal{N}(\mathbf{h}_i|\mathbf{0}, \mathbf{I})\mathrm{d}\mathbf{z}_{i,j} \\
&= \mathcal{N}(\mathbf{x}_{i,j}|\boldsymbol{\mu} + \mathbf{V}\mathbf{h}_i, \boldsymbol{\Phi}_{i,j})\mathcal{N}(\mathbf{h}_i|\mathbf{0}, \mathbf{I}) \\
&\propto \exp\left\{\mathbf{h}_i^\mathsf{T}\mathbf{V}^\mathsf{T}\boldsymbol{\Phi}_{i,j}^{-1}(\mathbf{x}_{i,j} - \boldsymbol{\mu}) - \frac{1}{2}\mathbf{h}_i^\mathsf{T}(\mathbf{I} + \mathbf{V}^\mathsf{T}\boldsymbol{\Phi}_{i,j}^{-1}\mathbf{V})\mathbf{h}_i\right\},
\end{aligned} \tag{2.28}$$

where $\boldsymbol{\Phi}_{i,j} = \mathbf{U}_{i,j}\mathbf{U}_{i,j}^\mathsf{T} + \boldsymbol{\Sigma}$. Assume that each training speaker provides one utterance (i.e. one i-vector only). Then, comparing Eq. 2.28 with a standard Gaussian [14] leads to

$$\begin{aligned}
\langle \mathbf{h}_i|\mathbf{x}_{i,j}\rangle &= \left(\mathbf{I} + \mathbf{V}^\mathsf{T}\boldsymbol{\Phi}_{ij}^{-1}\mathbf{V}\right)^{-1}\mathbf{V}^\mathsf{T}\boldsymbol{\Phi}_{ij}^{-1}(\mathbf{x}_{i,j} - \boldsymbol{\mu}) \\
\left\langle \mathbf{h}_i\mathbf{h}_i^\mathsf{T}|\mathbf{x}_{i,j}\right\rangle &= \left(\mathbf{I} + \mathbf{V}^\mathsf{T}\boldsymbol{\Phi}_{ij}^{-1}\mathbf{V}\right)^{-1} + \langle \mathbf{h}_i|\mathbf{x}_{i,j}\rangle\langle \mathbf{h}_i|\mathbf{x}_{i,j}\rangle^\mathsf{T}.
\end{aligned} \tag{2.29}$$

If speaker s has multiple i-vectors, the posterior expectations can be written as:

$$\begin{aligned}
\langle \mathbf{h}_i|\tilde{\mathbf{x}}_{i,\bullet}\rangle &= \left(\mathbf{I} + \sum\nolimits_{j=1}^{H_i}\mathbf{V}^\mathsf{T}\boldsymbol{\Phi}_{i,j}^{-1}\mathbf{V}\right)^{-1}\mathbf{V}^\mathsf{T}\sum\nolimits_{j=1}^{H_i}\boldsymbol{\Phi}_{i,j}^{-1}(\mathbf{x}_{i,j} - \boldsymbol{\mu}) \\
\left\langle \mathbf{h}_i\mathbf{h}_i^\mathsf{T}|\tilde{\mathbf{x}}_{i,\bullet}\right\rangle &= \left(\mathbf{I} + \sum\nolimits_{j=1}^{H_i}\mathbf{V}^\mathsf{T}\boldsymbol{\Phi}_{i,j}^{-1}\mathbf{V}\right)^{-1} + \langle \mathbf{h}_i|\tilde{\mathbf{x}}_{i,\bullet}\rangle\langle \mathbf{h}_i|\tilde{\mathbf{x}}_{i,\bullet}\rangle^\mathsf{T},
\end{aligned} \tag{2.30}$$

where $\tilde{\mathbf{x}}_{i,\bullet}$ represents a set of i-vectors from speaker i.

The posterior probability of latent variable $\mathbf{z}_{i,j}$ can be written as:

$$\begin{aligned}
p(\mathbf{z}_{i,j}|\mathbf{x}_{i,j},\boldsymbol{\theta}) &\propto p(\mathbf{x}_{i,j}|\mathbf{z}_{i,j},\boldsymbol{\theta})p(\mathbf{z}_{i,j}) \\
&= \int p(\mathbf{x}_{i,j},\mathbf{h}_i|\mathbf{z}_{i,j},\boldsymbol{\theta})p(\mathbf{z}_{i,j})\mathrm{d}\mathbf{h}_i \\
&= \int p(\mathbf{x}_{i,j}|\mathbf{h}_i,\mathbf{z}_{i,j},\boldsymbol{\theta})p(\mathbf{z}_{i,j})p(\mathbf{h}_i)\mathrm{d}\mathbf{h}_i \\
&= \int \mathcal{N}(\mathbf{x}_{i,j}|\boldsymbol{\mu}+\mathbf{V}\mathbf{h}_i+\mathbf{U}_{i,j}\mathbf{z}_{i,j},\boldsymbol{\Sigma})\mathcal{N}(\mathbf{z}_{i,j}|\mathbf{0},\mathbf{I})\mathcal{N}(\mathbf{h}_i|\mathbf{0},\mathbf{I})\mathrm{d}\mathbf{h}_i \\
&= \mathcal{N}(\mathbf{x}_{i,j}|\boldsymbol{\mu}+\mathbf{U}_{i,j}\mathbf{z}_{i,j},\boldsymbol{\Psi})\mathcal{N}(\mathbf{z}_{i,j}|\mathbf{0},\mathbf{I}) \\
&\propto \exp\left\{\mathbf{z}_{i,j}^{\mathsf{T}}\mathbf{U}_{i,j}^{\mathsf{T}}\boldsymbol{\Psi}^{-1}(\mathbf{x}_{i,j}-\boldsymbol{\mu})-\frac{1}{2}\mathbf{z}_{i,j}^{\mathsf{T}}(\mathbf{I}+\mathbf{U}_{i,j}^{\mathsf{T}}\boldsymbol{\Psi}^{-1}\mathbf{U}_{i,j})\mathbf{z}_{i,j}\right\}, \qquad (2.31)
\end{aligned}$$

where $\boldsymbol{\Psi}=\mathbf{V}\mathbf{V}^{\mathsf{T}}+\boldsymbol{\Sigma}$. Comparing this posterior density with a standard Gaussian, we have

$$\begin{aligned}
\langle\mathbf{z}_{i,j}|\mathbf{x}_{i,j}\rangle &= \left(\mathbf{I}+\mathbf{U}_{i,j}^{\mathsf{T}}\boldsymbol{\Psi}^{-1}\mathbf{U}_{i,j}\right)^{-1}\mathbf{U}_{i,j}^{\mathsf{T}}\boldsymbol{\Psi}^{-1}(\mathbf{x}_{i,j}-\boldsymbol{\mu}) \\
\left\langle\mathbf{z}_{i,j}\mathbf{z}_{i,j}^{\mathsf{T}}|\mathbf{x}_{i,j}\right\rangle &= \left(\mathbf{I}+\mathbf{U}_{i,j}^{\mathsf{T}}\boldsymbol{\Psi}^{-1}\mathbf{U}_{i,j}\right)^{-1}+\langle\mathbf{z}_{i,j}|\mathbf{x}_{i,j}\rangle\langle\mathbf{z}_{i,j}|\mathbf{x}_{i,j}\rangle^{\mathsf{T}}.
\end{aligned} \qquad (2.32)$$

2.4.2 M-Step

In the M-step, we maximize the lower bound of the likelihood [17] given the posterior expectation of the latent variables $\mathbf{h}_i$ and $\mathbf{z}_{i,j}$ that we have estimated in the E-step:

$$\begin{aligned}
B(\boldsymbol{\theta}) &= \mathbb{E}_{\mathcal{H},\mathcal{Z}}\left\{\sum\nolimits_{i,j}\ln\mathcal{N}\left(\mathbf{x}_{i,j}\middle|\boldsymbol{\mu}+\mathbf{V}\mathbf{h}_i+\mathbf{U}_{i,j}\mathbf{z}_{i,j},\boldsymbol{\Sigma}\right)\mathcal{N}(\mathbf{h}_i|\mathbf{0},\mathbf{I})\mathcal{N}(\mathbf{z}_{i,j}|\mathbf{0},\mathbf{I})\middle|\mathcal{X},\boldsymbol{\theta}\right\} \\
&= -\frac{1}{2}\sum_{i,j}\mathbb{E}_{\mathcal{H},\mathcal{Z}}\Big\{\log|\boldsymbol{\Sigma}|+(\mathbf{x}_{i,j}-\boldsymbol{\mu}-\mathbf{V}\mathbf{h}_i-\mathbf{U}_{i,j}\mathbf{z}_{i,j})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1} \\
&\qquad\times(\mathbf{x}_{i,j}-\boldsymbol{\mu}-\mathbf{V}\mathbf{h}_i-\mathbf{U}_{i,j}\mathbf{z}_{i,j})+\mathbf{h}_i^{\mathsf{T}}\mathbf{h}_i+\mathbf{z}_{i,j}^{\mathsf{T}}\mathbf{z}_{i,j}\big|\mathcal{X},\boldsymbol{\theta}\Big\},
\end{aligned} \qquad (2.33)$$

where $\mathcal{H}=\{\mathbf{h}_i;i=1,\ldots,I\}$ and $\mathcal{Z}=\{\mathbf{z}_{i,j};i=1,\ldots,I;j=1,\ldots,H_i\}$. Differentiating Eq. 2.33 with respect to $\mathbf{V}$ and $\boldsymbol{\Sigma}$ and set the resulting derivatives to zero, we

obtain

$$\mathbf{V} = \left[\sum\nolimits_{i,j} \left(\mathbf{x}_{i,j} - \boldsymbol{\mu} - \mathbf{U}_{i,j}\langle \mathbf{z}_{i,j}|\mathbf{x}_{i,j}\rangle\right)\langle \mathbf{h}_i|\tilde{\mathbf{x}}_{i,\bullet}\rangle^{\mathsf{T}}\right]\left[\sum\nolimits_{i,j}\langle \mathbf{h}_i\mathbf{h}_i^{\mathsf{T}}|\tilde{\mathbf{x}}_{i,\bullet}\rangle\right]^{-1} \tag{2.34}$$

$$\boldsymbol{\Sigma} = \frac{1}{\sum_i H_i}\left\{\sum\nolimits_{i,j}\left[(\mathbf{x}_{i,j}-\boldsymbol{\mu})(\mathbf{x}_{i,j}-\boldsymbol{\mu})^{\mathsf{T}} - \left(\mathbf{V}\langle \mathbf{h}_i|\tilde{\mathbf{x}}_{i,\bullet}\rangle + \mathbf{U}_{i,j}\langle \mathbf{z}_{i,j}|\mathbf{x}_{i,j}\rangle\right)(\mathbf{x}_{i,j}-\boldsymbol{\mu})^{\mathsf{T}}\right]\right\}. \tag{2.35}$$

2.5 PLDA Scoring with UP

Given a test i-vector $\mathbf{x}_t$ and target-speaker's i-vector $\mathbf{x}_s$, the log likelihood-ratio score is:

$$\begin{aligned}
\log S_{\mathrm{LR}}(\mathbf{x}_s,\mathbf{x}_t) &= \log\left[\frac{p(\mathbf{x}_s,\mathbf{x}_t|\text{same-speaker})}{p(\mathbf{x}_s,\mathbf{x}_t|\text{different-speakers})}\right] \\
&= \log\left[\frac{\int p(\mathbf{x}_s,\mathbf{x}_t,\boldsymbol{\eta}|\boldsymbol{\theta})\mathrm{d}\boldsymbol{\eta}}{\int p(\mathbf{x}_s,\boldsymbol{\eta}_s|\boldsymbol{\theta})\mathrm{d}\boldsymbol{\eta}_s \int p(\mathbf{x}_t,\boldsymbol{\eta}_t|\boldsymbol{\theta})\mathrm{d}\boldsymbol{\eta}_t}\right] \\
&= \log\left[\frac{\int p(\mathbf{x}_s,\mathbf{x}_t|\boldsymbol{\eta},\boldsymbol{\theta})p(\boldsymbol{\eta})\mathrm{d}\boldsymbol{\eta}}{\int p(\mathbf{x}_s|\boldsymbol{\eta}_s,\boldsymbol{\theta})p(\boldsymbol{\eta}_s)\mathrm{d}\boldsymbol{\eta}_s \int p(\mathbf{x}_t|\boldsymbol{\eta}_t,\boldsymbol{\theta})p(\boldsymbol{\eta}_t)\mathrm{d}\boldsymbol{\eta}_t}\right] \\
&= \log\mathcal{N}\left(\begin{bmatrix}\mathbf{x}_s\\ \mathbf{x}_t\end{bmatrix};\begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}\end{bmatrix},\begin{bmatrix}\boldsymbol{\Sigma}_s & \boldsymbol{\Sigma}_{ac}\\ \boldsymbol{\Sigma}_{ac} & \boldsymbol{\Sigma}_t\end{bmatrix}\right) \\
&\quad - \log\mathcal{N}\left(\begin{bmatrix}\mathbf{x}_s\\ \mathbf{x}_t\end{bmatrix};\begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}\end{bmatrix},\begin{bmatrix}\boldsymbol{\Sigma}_s & \mathbf{0}\\ \mathbf{0} & \boldsymbol{\Sigma}_t\end{bmatrix}\right) .
\end{aligned} \tag{2.36}$$

The global mean vector of all i-vectors is usually removed from i-vectors to save computation. $\boldsymbol{\Sigma}_s$, $\boldsymbol{\Sigma}_t$, and $\boldsymbol{\Sigma}_{ac}$ are respectively defined by:

$$\boldsymbol{\Sigma}_s = \mathbf{V}\mathbf{V}^{\mathsf{T}} + \mathbf{U}_s\mathbf{U}_s^{\mathsf{T}} + \boldsymbol{\Sigma}, \tag{2.37}$$

$$\boldsymbol{\Sigma}_t = \mathbf{V}\mathbf{V}^{\mathsf{T}} + \mathbf{U}_t\mathbf{U}_t^{\mathsf{T}} + \boldsymbol{\Sigma}, \tag{2.38}$$

and

$$\boldsymbol{\Sigma}_{ac} = \mathbf{V}\mathbf{V}^{\mathsf{T}}. \tag{2.39}$$

The log likelihood-ratio is:[1]

$$\begin{aligned}
\log S_{LR}(\mathbf{x}_s, \mathbf{x}_t) &= -\frac{1}{2}\begin{bmatrix}\mathbf{x}_s \\ \mathbf{x}_t\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}\boldsymbol{\Sigma}_s & \boldsymbol{\Sigma}_{ac} \\ \boldsymbol{\Sigma}_{ac} & \boldsymbol{\Sigma}_t\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{x}_s \\ \mathbf{x}_t\end{bmatrix} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_s \\ \mathbf{x}_t\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}\boldsymbol{\Sigma}_s & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_t\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{x}_s \\ \mathbf{x}_t\end{bmatrix} \\
&\qquad - \frac{1}{2}\log\begin{vmatrix}\boldsymbol{\Sigma}_s & \boldsymbol{\Sigma}_{ac} \\ \boldsymbol{\Sigma}_{ac} & \boldsymbol{\Sigma}_t\end{vmatrix} + \frac{1}{2}\log\begin{vmatrix}\boldsymbol{\Sigma}_s & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_t\end{vmatrix} \\
&= -\frac{1}{2}\begin{bmatrix}\mathbf{x}_s \\ \mathbf{x}_t\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}(\boldsymbol{\Sigma}_s - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_t^{-1}\boldsymbol{\Sigma}_{ac})^{-1} & -\boldsymbol{\Sigma}_s^{-1}\boldsymbol{\Sigma}_{ac}(\boldsymbol{\Sigma}_t - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_s^{-1}\boldsymbol{\Sigma}_{ac})^{-1} \\ -(\boldsymbol{\Sigma}_t - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_s^{-1}\boldsymbol{\Sigma}_{ac})^{-1}\boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_s^{-1} & (\boldsymbol{\Sigma}_t - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_s^{-1}\boldsymbol{\Sigma}_{ac})^{-1}\end{bmatrix}\begin{bmatrix}\mathbf{x}_s \\ \mathbf{x}_t\end{bmatrix} \\
&\qquad + \frac{1}{2}\begin{bmatrix}\mathbf{x}_s \\ \mathbf{x}_t\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}\boldsymbol{\Sigma}_s^{-1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_t^{-1}\end{bmatrix}\begin{bmatrix}\mathbf{x}_s \\ \mathbf{x}_t\end{bmatrix} - \frac{1}{2}\log\begin{vmatrix}\boldsymbol{\Sigma}_s & \boldsymbol{\Sigma}_{ac} \\ \boldsymbol{\Sigma}_{ac} & \boldsymbol{\Sigma}_t\end{vmatrix} + \frac{1}{2}\log\begin{vmatrix}\boldsymbol{\Sigma}_s & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_t\end{vmatrix} \\
&= \frac{1}{2}\begin{bmatrix}\mathbf{x}_s \\ \mathbf{x}_t\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}\boldsymbol{\Sigma}_s^{-1} - (\boldsymbol{\Sigma}_s - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_t^{-1}\boldsymbol{\Sigma}_{ac})^{-1} & \boldsymbol{\Sigma}_s^{-1}\boldsymbol{\Sigma}_{ac}(\boldsymbol{\Sigma}_t - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_s^{-1}\boldsymbol{\Sigma}_{ac})^{-1} \\ (\boldsymbol{\Sigma}_t - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_s^{-1}\boldsymbol{\Sigma}_{ac})^{-1}\boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_s^{-1} & \boldsymbol{\Sigma}_t^{-1} - (\boldsymbol{\Sigma}_t - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_s^{-1}\boldsymbol{\Sigma}_{ac})^{-1}\end{bmatrix}\begin{bmatrix}\mathbf{x}_s \\ \mathbf{x}_t\end{bmatrix} \\
&\qquad - \frac{1}{2}\log\begin{vmatrix}\boldsymbol{\Sigma}_s & \boldsymbol{\Sigma}_{ac} \\ \boldsymbol{\Sigma}_{ac} & \boldsymbol{\Sigma}_t\end{vmatrix} + \frac{1}{2}\log\begin{vmatrix}\boldsymbol{\Sigma}_s & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_t\end{vmatrix} \\
&= \frac{1}{2}\begin{bmatrix}\mathbf{x}_s^{\mathsf{T}} & \mathbf{x}_t^{\mathsf{T}}\end{bmatrix}\begin{bmatrix}\mathbf{A}_{s,t} & \mathbf{B}_{s,t} \\ \mathbf{B}_{s,t} & \mathbf{C}_{s,t}\end{bmatrix}\begin{bmatrix}\mathbf{x}_s \\ \mathbf{x}_t\end{bmatrix} + D_{s,t} \\
&= \frac{1}{2}\left[\mathbf{x}_s^{\mathsf{T}}\mathbf{A}_{s,t}\mathbf{x}_s + \mathbf{x}_s^{\mathsf{T}}\mathbf{B}_{s,t}\mathbf{x}_t + \mathbf{x}_t^{\mathsf{T}}\mathbf{B}_{s,t}\mathbf{x}_s + \mathbf{x}_t^{\mathsf{T}}\mathbf{C}_{s,t}\mathbf{x}_t\right] + D_{s,t} \\
&= \frac{1}{2}\left[\mathbf{x}_s^{\mathsf{T}}\mathbf{A}_{s,t}\mathbf{x}_s + 2\mathbf{x}_s^{\mathsf{T}}\mathbf{B}_{s,t}\mathbf{x}_t + \mathbf{x}_t^{\mathsf{T}}\mathbf{C}_{s,t}\mathbf{x}_t\right] + D_{s,t}
\end{aligned} \tag{2.40}$$

[1]To simplify notation, we assume that the i-vectors have been subjected to mean removal.

where

$$\mathbf{A}_{s,t} = \boldsymbol{\Sigma}_s^{-1} - (\boldsymbol{\Sigma}_s - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_t^{-1}\boldsymbol{\Sigma}_{ac})^{-1} \tag{2.41a}$$

$$\mathbf{B}_{s,t} = \boldsymbol{\Sigma}_s^{-1}\boldsymbol{\Sigma}_{ac}(\boldsymbol{\Sigma}_t - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_s^{-1}\boldsymbol{\Sigma}_{ac})^{-1} \tag{2.41b}$$

$$\mathbf{C}_{s,t} = \boldsymbol{\Sigma}_t^{-1} - (\boldsymbol{\Sigma}_t - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_s^{-1}\boldsymbol{\Sigma}_{ac})^{-1} \tag{2.41c}$$

$$D_{s,t} = -\frac{1}{2}\log\begin{vmatrix}\boldsymbol{\Sigma}_s & \boldsymbol{\Sigma}_{ac}\\ \boldsymbol{\Sigma}_{ac} & \boldsymbol{\Sigma}_t\end{vmatrix} + \frac{1}{2}\log\begin{vmatrix}\boldsymbol{\Sigma}_s & \mathbf{0}\\ \mathbf{0} & \boldsymbol{\Sigma}_t\end{vmatrix}. \tag{2.41d}$$

Chapter 3

DEEP NEURAL NETWORKS FOR SPEAKER EMBEDDING

This chapter introduces the deep neural network architectures that were used in this work. Unlike, regular regression and classification tasks, speaker verification systems are typically composed of a deep neural network (DNN) for extracting speaker-dependent vectors and a backend classifier for decision making. The backend typically is a PLDA model or cosine scoring. The feature vectors from the DNN are referring to as embeddings in the literature.

The early successes in DNN-based SV include the d-vector [18] and the triplet network [19]. In [18], the authors proposed using a fully-connected network to process contextual filter-bank features. The averaged activations at the last layer were used for cosine-distance scoring. An advanced DNN architecture for SV was proposed in [19], where a network comprising several inception blocks was trained by minimizing the triplet loss. The most successful DNN-based SV is the x-vector network proposed in [5]. The x-vector network consists of time-delay neural networks (TDNNs) followed by a statistics pooling layer and fully-connection layers. The x-vector network is also the first to include standard deviations in statistics pooling. More recently, ResNets and DenseNets have been successfully applied to SV [20, 21]. The ResNets in [20] consist of 2D convolutional layers instead of 1D TDNNs as in x-vector networks. Compared with the x-vector variants, the ResNets are more compact. However, the inference time of ResNets is significantly longer than that of x-vector networks.

3.1 X-vector Network

The x-vector network consists of three parts: Frame-level time-delay neural networks (TDNNs), utterance-level fully-connected (FC) layers, and a statistics pooling layer [5, 22]. A TDNN is a special form of convolutional neural networks (CNNs). It skips the computation at chosen temporal positions while maintaining the same receptive field size as a CNN. A statistics pooling layer concatenates the mean and standard deviation of the activations from the last convolutional layer. The concatenated mean and standard deviation are passed to two FC layers. The network is trained to minimize the standard cross-entropy loss using small chunks of acoustic sequences derived from the clean and augmented utterances. Typical chunk length for training an x-vector network ranges from 200 ms to 400 ms.

After the network is trained, the embedding of each utterance is extracted from the first affine layer after the statistics pooling layer. A backend consisting of LDA and PLDA models is trained using the embeddings as input [14]. Although the network is trained to minimize classification error, the ultimate goal is to produce embeddings that perform well for the speakers that do not appear in the training set. As the speaker information should not be limited to individual frames, the embeddings should be extracted anywhere after statistics pooling. The pre-softmax affine layer is too cumbersome to work with due to its large size. The best choice is usually the last affine layer. However, it was found that the embeddings extracted from the penultimate layer perform better than the ones extracted from the last layer in both SRE16 and SRE18. It could be that both SRE16 and SRE18 have severe domain mismatch between training and test data.

This thesis also used a wide x-vector network in which the number of channel per layer is increased. Table 3.1 shows the architectures of the x-vector network and a larger x-vector network used in this thesis. The number of floating-point operations and model sizes of the x-vector networks are presented in Table 3.3.

3.2 Residual Networks

Residual networks were proposed in [23] to solve the gradient vanishing problem in deep neural networks. Deep neural networks are difficult to train with standard gradient descent. In a network with over a hundred layers, the bottom layers (layers closed to the input) can only receive a very weak gradient signal, which makes training difficult. The authors in [23] found that compared with their shallow counterparts, deeper neural networks are more prone to under-fitting instead of overfitting. They argued that a deeper neural network should be at least as powerful as the shallow one if optimization is successful. They introduce a skip connection unit to help training very deep networks.

Let $\mathbf{x}$ be the input to a few stacked layers and $\mathbf{y}$ be the output of the stacked layers. In conventional neural networks, the goal of learning is to learn an input-to-output mapping that minimizes a loss function. In a residual network, however, skip connections are introduced so that the residual unit learns the difference between the input and output. The output of a residual block is:

$$\mathbf{y} = F(\mathbf{x}) + \mathbf{x}, \tag{3.1}$$

where $F(\mathbf{x})$ is the mapping shown in Fig. 3.1. The author in [23] found that it is easier to learn the residue than the direct mapping from $\mathbf{x}$ to $\mathbf{y}$. Fig. 3.1 illustrates a residual block in a ResNet.

3.3 DenseNets

DenseNets were proposed in [24] for computer vision. A Densenet comprises two types of blocks, namely, dense block and transition block. In a dense block, each layer is connected by the outputs from all of the previous layers. To prevent the number of feature maps from growing excessively, a transition block is introduced to reduce the

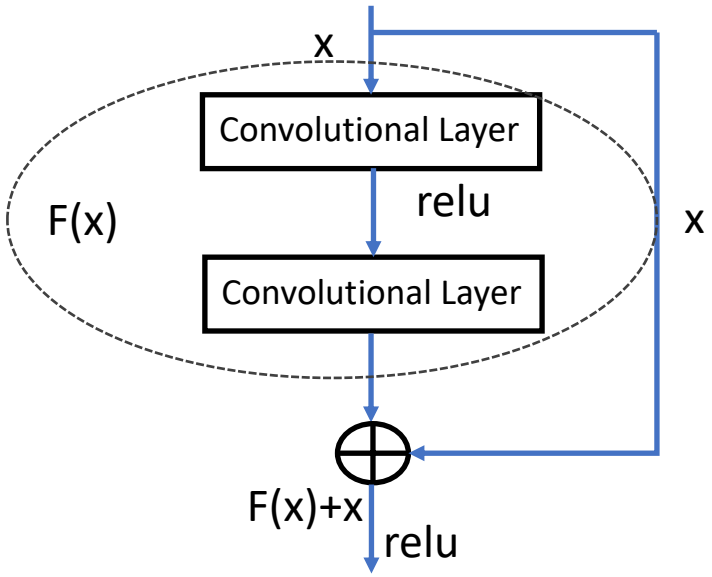

Figure 3.1: A residual block in a ResNet.

feature map size. Suppose each convolutional layer produces k feature maps, then the l-th layer inside the block has $k_0 + k \times (l-1)$ feature maps, where k_0 is the number of feature maps in the input layer. The parameter k is referred to as the growth rate.

In this thesis, we used a dense network composed of 1-dimensional convolution layers. We used the same statistics pooling layer as that of the x-vector network. Because max-pooling and average pooling do not work well in speaker recognition, we replaced the max-pooling layers by stride-2 convolution layers. Table 3.2 shows our network architectures. Table 3.3 summarizes the number of parameters and the number of floating-point operations in a forward pass for our Densenet80 and Densenet121 for an input size of 23×400.

3.4 Additive Margin Softmax

Margin-based loss has been very successful in face recognition [25] and speaker recognition [26]. Additive margin loss enforces a minimum margin m between the target

class and non-target classes:

$$\begin{aligned}\mathcal{L}_{AMS} &= -\frac{1}{n}\sum_{i=1}^{n}\log\frac{e^{s\cdot\left(\cos\theta_{y_i}-m\right)}}{e^{s\cdot\left(\cos\theta_{y_i}-m\right)}+\sum_{j=1,j\neq y_i}^{c}e^{s\cdot\cos\theta_j}} \\ &= -\frac{1}{n}\sum_{i=1}^{n}\log\frac{e^{s\cdot\left(\mathbf{W}_{y_i}^{T}\mathbf{x}_i-m\right)}}{e^{s\cdot\left(\mathbf{W}_{y_i}^{T}\mathbf{x}_i-m\right)}+\sum_{j=1,j\neq y_i}^{c}e^{s\mathbf{W}_j^{T}\mathbf{x}_i}},\end{aligned} \tag{3.2}$$

where $\mathbf{W}$ is a weight matrix ($\mathbf{W}_j$ is the j-th column of $\mathbf{W}$) and $\mathbf{x}$ is an embedding vector, both of which are normalized to have unit length. s is a scaling constant.

Table 3.1: Summary of our neural network architectures. The kernel is specified as kernel size, stride, and dilation.

	X-vector network	
Layer	**Kernel**	**Channel_in × Channel_out**
Conv1	5,1,1	MFCC_DIM × 512
Conv2	3,1,2	512 × 512
Conv3	3,1,3	512 × 512
Conv4	1,1,1	512 × 512
Conv5	1,1,1	512 × 1536
Statistics pooling		1536 × 3072
FC6	–	3072 × 512
FC7	–	512 × 512
AM-softmax	–	512 × N

	Wide x-vector network	
Layer	**Kernel**	**Channel_in × Channel_out**
Conv1	5,1,1	MFCC_DIM × 1024
Conv2	3,1,2	1024 × 1024
Conv3	3,1,3	1024 × 1024
Conv4	1,1,1	1024 × 1024
Conv5	1,1,1	1024 × 2000
Statistics pooling		2000 × 4000
FC6	–	4000 × 512
FC7	–	512 × 512
AM-softmax	–	512 × N

Table 3.2: Densenet architectures for speaker embedding. The growth rate for the networks is 40. Note that each "conv" layer shown in the table corresponds to the sequence BN-ReLU-Conv. "conv 3" denotes 1-D convolution with kernel size 3.

Layers	Densenet-80	Densenet-121
Convolution	conv 3	conv 3
Dense Block (1)	$\begin{bmatrix} \text{conv } 1 \\ \text{conv } 3 \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv } 1 \\ \text{conv } 3 \end{bmatrix} \times 6$
Transition Layer (1)	conv 2 stride 2	conv 2 stride 2
Dense Block (2)	$\begin{bmatrix} \text{conv } 1 \\ \text{conv } 3 \end{bmatrix} \times 10$	$\begin{bmatrix} \text{conv } 1 \\ \text{conv } 3 \end{bmatrix} \times 12$
Transition Layer (2)	conv 2 stride 2	conv 2 stride 2
Dense Block (3)	$\begin{bmatrix} \text{conv } 1 \\ \text{conv } 3 \end{bmatrix} \times 14$	$\begin{bmatrix} \text{conv } 1 \\ \text{conv } 3 \end{bmatrix} \times 24$
Transition Layer (3)	conv 2 stride 2	conv 2 stride 2
Dense Block (4)	$\begin{bmatrix} \text{conv } 1 \\ \text{conv } 3 \end{bmatrix} \times 10$	$\begin{bmatrix} \text{conv } 1 \\ \text{conv } 3 \end{bmatrix} \times 16$
Stats-pooling Layer	-	-
FC1	492×512 Linear	2560×512 Linear
FC2	512×256 Linear	512×256 Linear
Classification Layer	$256 \times$ # of classes AM-Softmax	$256 \times$ # of classes AM-Softmax

Table 3.3: Summary of the number of parameters and the number of floating-point operations in a forward pass for our Densenet121, the x-vector network and the wide x-vector network for an input size of 23×400.

Model	# of flops	# of parameters
X-vector network	1000.043M	4.276M
Wide x-vector network	3000.705M	11.639M
Densenet80	572.172M	5.256M
Densenet121	958.143M	10.334M

Chapter 4

DOMAIN ADAPTATION FOR SPEAKER RECOGNITION

4.1 Domain Adaptation as a Robust Speaker Verification Problem

A fundamental assumption of machine learning is that training and test data are sampled from the same distribution. However, a lot of factors can invalidate this assumption. For speaker verification, this could happen when the new environment has some specific noise and channel conditions or involves speakers speaking different languages than the training speakers. We want the speaker verification systems to be robust to these factors, which can be achieved through domain adaptation.

4.2 Domain Mismatch in Speaker Recognition

Using i-vector as an unsupervised feature extraction method and PLDA as a supervised channel compensation technique have been very successful in speaker verification [8,14]. However, like many machine learning algorithms, i-vector/PLDA assumes that the training data and test data are independently sampled from the same distribution. When training data and test data have a severe mismatch, the performance degrades rapidly [27–32]. The mismatch between training data and test data is not uncommon, as it can be caused by a lot of factors such as languages, channels, noises, and genders. Collecting more data to retrain the system is time-consuming and computationally-expensive. Such a solution is unrealistic in some scenarios. It is desirable to use the existing data and a small amount of target-specific data to modify the system to meet the need, which is essentially what domain adaptation (DA) does.

To highlight the problems caused by domain mismatch in real-world scenarios, we

use the gender and language mismatch in NIST 2016 speaker recognition (SRE16) as an example. SRE16 introduces various new challenges to speaker recognition [33, 34], among which the multilingual setup brought the most attention. Unlike the previous SREs, both development (Dev) and evaluation (Eval) data in SRE16 comprise utterances spoken in non-English languages. Table 4.1 shows the composition of SRE16 data. Because all of the SRE16 data are non-English, training using data from the previous SREs results in poor performance. Training using only SRE16 development data is also not feasible, as there are only 2,472 segments in total and a very small number of them are labeled. Besides, the labeled development data are of different languages than the evaluation data.

Table 4.1: The composition of SRE16 data. "Labeled" means speaker labels are provided. "unLabeled" means speaker labels are not provided.

Dataset	Category	Language
Dev	Unlabelled	Cantonese and Tagalog
Dev	Unlabelled	Mandarin and Cebuano
Dev	Labelled	Mandarin and Cebuano
Eval	Enrollment	Cantonese and Tagalog
Eval	Test	Cantonese and Tagalog

Fig. 4.3(a) shows the t-distributed stochastic neighbor embedding (t-SNE) [35] of i-vectors from SRE16 development data and the previous SRE data. In the figure, datasets are colored according to speakers' genders and languages. The gender- and language-dependent clusters are clearly visible in this 2-dimensional embedded space. Also, the multi-source mismatch occurs not only between the English data (ENG_F and ENG_M) and SRE16 data but also within SRE16 data (CAN_F, CAN_M, TGL_F, and TGL_M).

To compare the closeness between different language- and gender-clusters in the

original i-vector space, we computed the squared Euclidean distances between the means of these clusters and normalized the pairwise distances by their maximum. Fig. 4.3(b) shows these normalized distances. Apparently, some gender-language combinations (e.g., English-male) are more distinct from the others. To get a sense of the degree of dispersion of these clusters, we may compare the maximum pairwise distance (= 0.0021) with the trace of the total covariance matrix (= 0.0068). This means that the maximum distance (occurs between CAN_F and ENG_M) is about 31% of the total variance, which is not a small value because if the i-vectors are domain-independent, this value should be zero.

To compare the variances of these clusters, Fig. 4.3(c) shows the pairwise differences between the covariance matrices of the i-vectors from the six language- and gender-dependent groups. The differences are measured in terms of Frobenius norm [36]. Again, the differences are normalized by the maximum difference. We can see that the covariance matrices of these clusters are fairly different from each other. In particular, even the minimum difference (ENG_M–ENG_F) is 55% of the maximum difference (CAN_M–TGL_F). Interestingly, although CAN_M and ENG_F are closest in terms of their means, the difference between their covariance matrix is the second largest. Overall speaking, Fig. 4.3 shows that the i-vectors of these six groups differ from each other not only by their means but also by their covariance matrices.

4.3 Domain Adaptation Methods

State-of-the-art SV systems are comprised of a deep neural network and a backend model [5]. DA is typically carried out in the PLDA backend or directly during the training of the speaker embedding network. We refer to adaptation in the PLDA backend as backend domain adaptation and adaptation during the training of the speaker embedding network as DNN adaptation. Figure 4.1 shows the processing pipeline for adapting the PLDA backend.

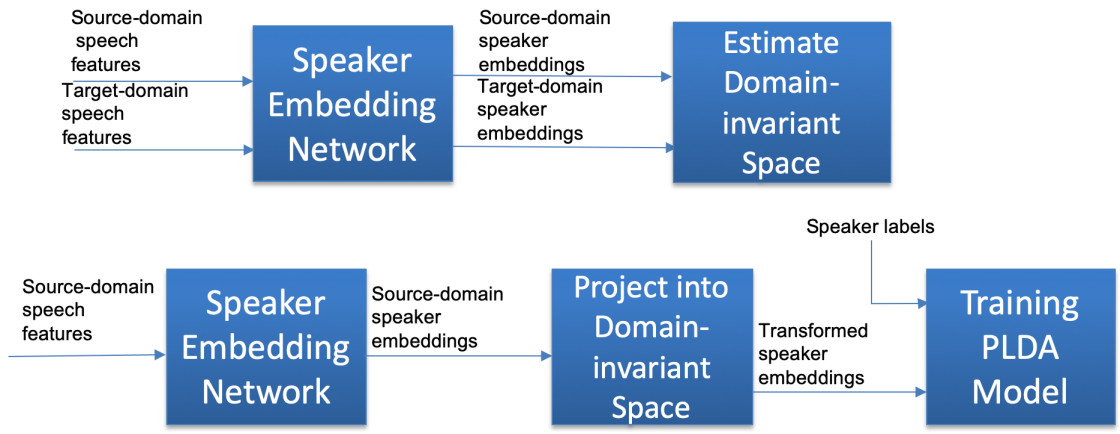

Figure 4.1: Backend adaptation pipeline

4.3.1 Backend Domain Adaptation

Garcia-Romero and McCree [27] found that the mismatch between the out-of-domain PLDA model and the in-domain test data contributes to most of the performance degradation. Therefore, it is important to apply domain adaptation to reduce the mismatch between in-domain and out-domain i-vectors before training the PLDA model. Alternatively, a PLDA model trained on out-of-domain data can be adapted to fit the in-domain data.

Earlier attempts in i-vector based DA require the in-domain data to have speaker labels. For example, Garcia-Romero and McCree [27] computed the MAP-estimates of the in-domain within-speaker and across-speaker covariance matrices in the i-vector space using the speaker labels from the in-domain data. In [29], these matrices are treated as latent variables and their joint posterior distribution is factorized using variational Bayes so that MAP point estimates of the matrices can be computed from the factorized distributions. The point estimates are then used for scoring in the target environment. More recent approaches attempt to obviate the need for speaker labels. For instance, Villalba and Lleida [37] extended their Bayesian adaptation in [29] by treating the *unknown* speaker labels in the in-domain data as latent variables. Another

approach is to generate *hypothesized* speaker labels via unsupervised clustering [28, 38, 39]. Given the hypothesized labels, the covariance matrices of in-domain data can be computed as usual and can be interpolated with the out-of-domain covariance matrices to obtain an adapted PLDA model. Of course, correctly inferring all of the missing labels is even harder than performing speaker verification. However, as is shown in [28], even imperfect labels can achieve performance almost as good as the correct labels. Still, cluster-based approaches require a lot of heuristics to set the number of clusters.

It is also possible to carry out the unsupervised DA without inferring the missing labels at all. Most of the methods in this category assume that there is a common feature space in which in-domain and out-domain have a minimum mismatch. DA aims to project data on such feature space and uses the projected data to train a classifier. Fig 4.2 shows the process of i-vector based domain adaptation using the common feature-space approach. As mismatch can be caused by multiple sources, it is helpful to divide the training data into homogenous subsets according to their sources before finding a common feature space. This is called multi-source domain adaptation in the literature [40]. In addition to the robustness to heterogeneous sources, this approach also has the potential to generalize to unseen domains, as it does not assume a particular in-domain environment. Over the years, several unsupervised DA techniques have been proposed to find a common feature space that is less domain-dependent. These techniques include inter-dataset variability compensation (IDVC) [30, 41, 42], source normalization (SN) [31] and discriminative multi-domain PLDA [32]. IDVC divides the training data into several subsets, and for each subset, the mean is computed. The means of these subsets are used to find the directions of maximum inter-dataset variability; then the subspace corresponding to these directions is removed from the i-vectors.

In [30], the author successfully used IDVC to reduce the mismatch between NIST telephone data and switchboard data. In several NIST 2016 submissions [2, 43],

IDVC is also found to be very helpful in boosting system performance. Recently, domain-adversarial training of neural networks has also been successfully applied to unsupervised domain adaptation for speaker recognition [44]. In domain-adversarial learning [45], a feature extraction network is trained to produce embedded features that are indistinguishable to a domain classifier but are highly speaker discriminative to a speaker classifier. After training, the embedded features are believed to be domain-invariant.

Specifically, IDVC follows the subspace removal approach proposed in [46]. It aims to find the directions in the i-vector space with the largest inter-dataset variability and remove the variability in these directions. This is achieved by projecting the i-vectors $\mathbf{x}$'s as follows:

$$\hat{\mathbf{x}} = (\mathbf{I} - \mathbf{W}\mathbf{W}^{\mathsf{T}})\mathbf{x}, \tag{4.1}$$

where the columns of $\mathbf{W}$ span the subspace of unwanted variability. $\mathbf{W}$ comprises the eigenvectors of the covariance matrix of the subset means. Recently, it has been extended to reduce the inaccuracy of PLDA scores caused by domain mismatch [41].

Note that in IDVC the domain mismatch is defined by the covariances of the subset means. However, the mismatch of datasets may not only manifest in the dataset means but also in the higher-order statistics of these datasets. The limitation of IDVC will become apparent when we consider some Gaussian distributions (one for each dataset) with identical means but different covariance matrices. Despite the severe mismatches among these Gaussians, IDVC considers these Gaussians to be identical and will not remove any subspace ($\mathbf{W}$ in Eq. 4.1 is a null matrix) to reduce the mismatches.

Another very popular DA method for the backend is correlation alignment (CORAL), which essentially whitens the source-domain data and recolors them with a whitening matrix estimated from the target-domain data [47]. In [48], the author proposed a

Algorithm 1 CORAL for Unsupervised Domain Adaptation

Input: Source Data D_S, Target Data D_T
Output: Adjusted Source Data D_s^*
$C_S = cov(D_S)$
$C_T = cov(D_T)$
$D_S = D_S * C_S^{\frac{-1}{2}}$
$D_S^* = D_S * C_T^{\frac{1}{2}}$

hybrid method combining PLDA model adaptation and CORAL and showed that it is superior to the individual methods. Specifically, CORAL aims to minimize the distance between the second-order statistics (covariance) of the source features $\mathbf{C}_S$ and target features $\mathbf{C}_T$. When using speaker embeddings, CORAL has the advantage of fast adaptation without re-training the whole network for a new domain. CORAL aims to find a transformation matrix $\mathbf{A}$ that minimizes the distance:

$$\left\|\mathbf{A}^\top \mathbf{C}_S \mathbf{A} - \mathbf{C}_T\right\|_F^2. \tag{4.2}$$

Eq. 4.2 has a closed form solution. Algorithm 1 shows the pseudo-code of CORAL [47]. In the case of PLDA adaptation, CORAL is typically preceded by shifting the mean of the PLDA model by the mean of the in-domain data, i.e., centering. In [49], the author proposed to combine a variational auto-encoder (VAE) [50] with a domain adversarial neural network (DANN) for x-vectors domain adaptation. The DANN [45] part aims to retain speaker identity information and learn a feature space that is robust against domain mismatch, while the VAE part is to impose variational regularization on the learned features so that they follow a Gaussian distribution. In [51], the authors extended their variational DANN by incorporating an information maximization criterion [52] into the objective function.

4.3.2 DNN Domain Adaptation

DNN adaptation is relatively new in SV. Because DNN provides a larger parameter space to explore, it is potentially more powerful than backend adaptation methods. Most of the DA methods aim to minimize the divergence between the source-domain data and the target-domain data. In the context of speaker verification, this means minimizing the discrepancy between the source-domain speaker embeddings and the target-domain speaker embeddings. In [1], the authors proposed to use adversarial learning to adapt the speaker embeddings. Specifically, Wasserstein GANs [53] were used to minimize the discrepancy between the source-domain and the target-domain speaker embeddings. The authors also explored using other information such as language labels and phone numbers and found that they are beneficial. However, their method requires speaker labels to perform well, which limits the method's applicability. In [54], several GAN variants were proposed. Both adaptation and verification were carried out end-to-end. However, the performance of the system is not as good as the x-vector system with PLDA adaptation in Kaldi.

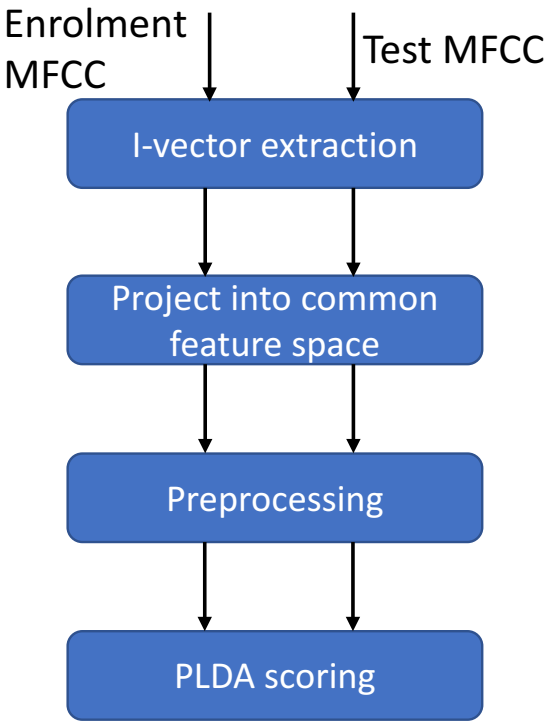

Figure 4.2: A flow chart showing the process of i-vector based domain adaptation using the common feature-space approach.

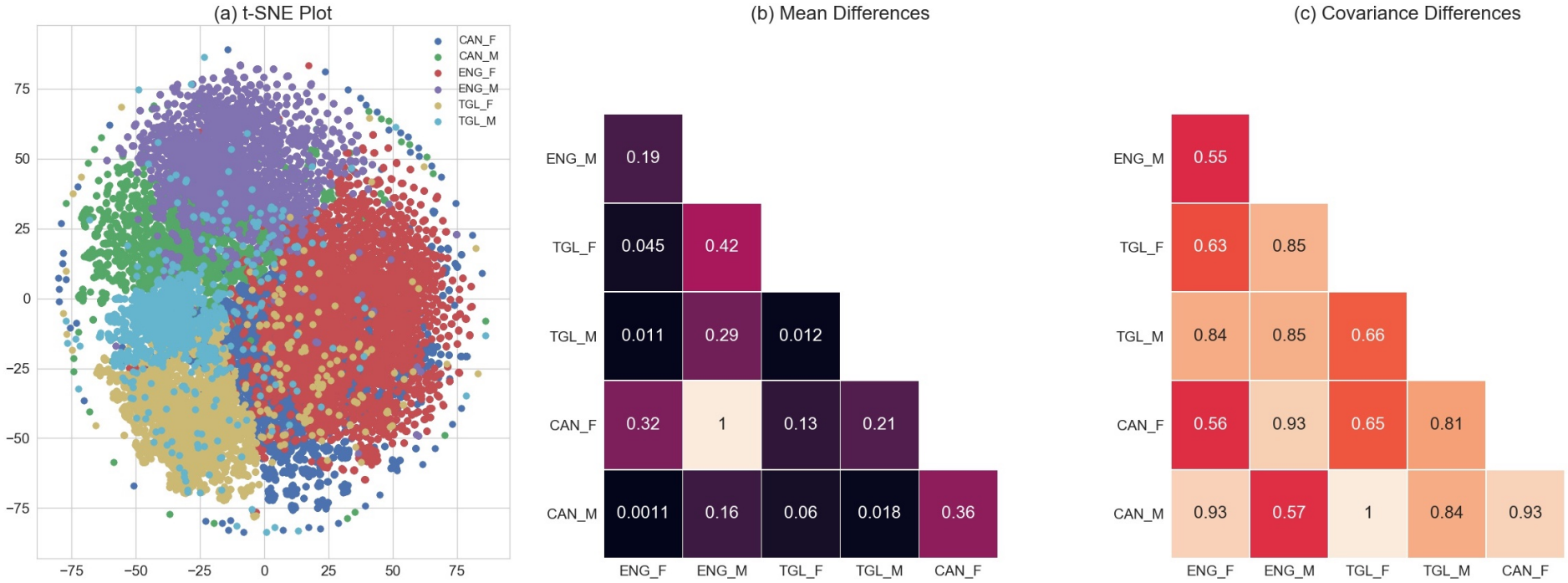

Figure 4.3: (a) Scatter plot of 2-dimensional t-SNE embedded i-vectors. In the legend, "M" and "F" stand for male and female, respectively, and "ENG", "CAN" and "TGL" stand for English, Cantonese, and Tagalog, respectively. (b) Pairwise differences between the means. (c) Pairwise differences between the covariance matrices. The means and covariances differences are measured in the *original* space and are normalized to the range between 0 and 1 for ease of comparison.

Chapter 5

MULTI-SOURCE MMD-BASED DOMAIN ADAPTATION

5.1 Limitation of IDVC

Several theoretical works in DA [3, 4, 55] and practical applications [56] suggest that minimizing the divergence between the in-domain and out-domain distributions is very important for obtaining a good representation. From this perspective, approaches based solely on the differences among the domain-means, such as IDVC, are not enough for finding a good representation. The reason is that even if the means of the distributions are exactly the same, there could still be a severe mismatch between the data distributions if their variances are very different. Thus, to reduce inter-dataset mismatch, it is important to consider the statistics beyond the means.

To better utilize the statistics of multi-source data, this thesis applies maximum mean discrepancy (MMD) [57] for domain adaptation. MMD is a nonparametric method for measuring the distance between two distributions [7, 58, 59]. With a properly chosen kernel, MMD can utilize all moments of data. This thesis generalizes MMD to measure the discrepancies among multiple distributions. By formulating MMD as a part of the objective function for training autoencoders, the autoencoders will learn the features that contain less domain-specific information but are still relevant to the classification task. It also applies MMD to force an autoencoder to capture the inter-data set variabilities. By subtracting out these variabilities, the i-vectors become more domain-independent.

5.2 Maximum Mean Discrepancy Autoencoder

5.2.1 Maximum Mean Discrepancy

The theoretical works in DA [3, 4, 55] suggest that it is important to have a good measurement of the divergence between the data distributions of different domains. The maximum mean discrepancy is a distance measure in the space of probability. Given two sets of samples $\{\mathbf{x}_i\}_{i=1}^N$ and $\{\mathbf{y}_j\}_{j=1}^M$, MMD computes the mean squared difference of the statistics of the two datasets:

$$\mathcal{D}_{\text{MMD}} = \left\| \frac{1}{N}\sum_{i=1}^{N}\phi(\mathbf{x}_i) - \frac{1}{M}\sum_{j=1}^{M}\phi(\mathbf{y}_j) \right\|^2, \tag{5.1}$$

where ϕ is a feature map. When ϕ is the identity function, the MMD distance simply computes the discrepancy between the sample means.

Eq. 5.1 can be expanded as:

$$\begin{aligned}\mathcal{D}_{\text{MMD}} = &\frac{1}{N^2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\phi(\mathbf{x}_i)^\mathsf{T}\phi(\mathbf{x}_{i'}) \\ &- \frac{2}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\phi(\mathbf{x}_i)^\mathsf{T}\phi(\mathbf{y}_j) + \frac{1}{M^2}\sum_{j=1}^{M}\sum_{j'=1}^{M}\phi(\mathbf{y}_j)^\mathsf{T}\phi(\mathbf{y}_{j'}).\end{aligned} \tag{5.2}$$

As each term in Eq. 5.2 involves dot products only, the kernel trick can be applied:

$$\begin{aligned}\mathcal{D}_{\text{MMD}} = &\frac{1}{N^2}\sum_{i=1}^{N}\sum_{i'=1}^{N}k(\mathbf{x}_i,\mathbf{x}_{i'}) \\ &- \frac{2}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}k(\mathbf{x}_i,\mathbf{y}_j) + \frac{1}{M^2}\sum_{j=1}^{M}\sum_{j'=1}^{M}k(\mathbf{y}_j,\mathbf{y}_{j'}),\end{aligned} \tag{5.3}$$

where $k(\cdot,\cdot)$ is a kernel function. In the case of quadratic (Quad) kernels, we have:

$$k(\mathbf{x},\mathbf{y}) = (\mathbf{x}^\mathsf{T}\mathbf{y} + c)^2. \tag{5.4}$$

Then, the MMD becomes:

$$\mathcal{D}_{\text{MMD}} = 2c \left\| \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i - \frac{1}{M} \sum_{j=1}^{M} \mathbf{y}_j \right\|^2 + \left\| \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^\mathsf{T} - \frac{1}{M} \sum_{j=1}^{M} \mathbf{y}_j \mathbf{y}_j^\mathsf{T} \right\|_F^2, \tag{5.5}$$

where $\|\cdot\|_F$ represents the Frobenius norm. We can see that with a quadratic kernel, MMD can match up to the second-order statistics and c can be adjusted to control the trade-off of the matching between the first-order and the second-order moments.

Another popular kernel is the Gaussian kernel:

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{x} - \mathbf{y}\|^2 \right), \tag{5.6}$$

where σ is the width parameter. With the Gaussian kernel, the feature space is of infinite dimension and contains all moments of data. Minimizing MMD using the Gaussian kernel is equivalent to matching all moments of two distributions [58]. It is also possible to use a mixture of the Gaussian kernels [59]:

$$k(\mathbf{x}, \mathbf{y}) = \sum_{q=1}^{K} \exp\left(-\frac{1}{2\sigma_q^2} \|\mathbf{x} - \mathbf{y}\|^2 \right), \tag{5.7}$$

where σ_q is the width parameter of the q-th Gaussian kernel.

5.2.2 Domain-invariant Autoencoder

Assume that we have in-domain data $\{\mathbf{x}_i^{\text{in}}\}_{i=1}^{N_{\text{in}}}$ and out-domain data $\{\mathbf{x}_j^{\text{out}}\}_{j=1}^{N_{\text{out}}}$. We want to learn a transformation $\mathbf{h} = f(\mathbf{x})$ such that the transformed data $\{\mathbf{h}_i^{\text{in}}\}_{i=1}^{N_{\text{in}}}$ and $\{\mathbf{h}_j^{\text{out}}\}_{j=1}^{N_{\text{out}}}$ are as similar as possible. The mismatch between the transformed data

can be measured by MMD:

$$\mathcal{D}_{\text{MMD}} = \frac{1}{N_{\text{in}}^2}\sum_{i=1}^{N_{\text{in}}}\sum_{i'=1}^{N_{\text{in}}} k(\mathbf{h}_i^{\text{in}}, \mathbf{h}_{i'}^{\text{in}}) - \frac{2}{N_{\text{in}}N_{\text{out}}}\sum_{i=1}^{N_{\text{in}}}\sum_{j=1}^{N_{\text{out}}} k(\mathbf{h}_i^{\text{in}}, \mathbf{h}_j^{\text{out}}) + \frac{1}{N_{\text{out}}^2}\sum_{j=1}^{N_{\text{out}}}\sum_{j'=1}^{N_{\text{out}}} k(\mathbf{h}_j^{\text{out}}, \mathbf{h}_{j'}^{\text{out}}). \tag{5.8}$$

When the data come from multiple sources, we want the transformed data to be as similar to each other as possible. To this end, we define a domain-wise MMD measure. Specifically, given D sets of data $\{\mathbf{x}_i^d\}_{i=1}^{N_d}$, where $d = 1, 2, \ldots, D$, we want the transformed data $\{\mathbf{h}_i^d\}_{i=1}^{N_d}$ to have small loss as defined by the following equation:

$$\mathcal{L}_{\text{mismatch}} = \sum_{d=1}^{D}\sum_{\substack{d'=1\\d'\neq d}}^{D}\left(\frac{1}{N_d^2}\sum_{i=1}^{N_d}\sum_{i'=1}^{N_d} k(\mathbf{h}_i^d, \mathbf{h}_{i'}^d) - \frac{2}{N_d N_{d'}}\sum_{i=1}^{N_d}\sum_{j=1}^{N_{d'}} k(\mathbf{h}_i^d, \mathbf{h}_j^{d'}) + \frac{1}{N_{d'}^2}\sum_{j=1}^{N_{d'}}\sum_{j'=1}^{N_{d'}} k(\mathbf{h}_j^{d'}, \mathbf{h}_j^{d'})\right). \tag{5.9}$$

Of course, we also want to retain as much non-domain related information as possible. Assume that another transformation can reconstruct the input from $\mathbf{h}$:

$$\tilde{\mathbf{x}} = g(\mathbf{h}), \tag{5.10}$$

where $\tilde{\mathbf{x}}$ is the reconstruction of the input $\mathbf{x}$. We want to make $\tilde{\mathbf{x}}$ as close to $\mathbf{x}$ as possible by minimizing:

$$\mathcal{L}_{\text{recons}} = \frac{1}{2}\sum_{d=1}^{D}\sum_{i=1}^{N_d}\left\|\mathbf{x}_i^d - \tilde{\mathbf{x}}_i^d\right\|^2. \tag{5.11}$$

Both objectives can be achieved by an antoencoder comprising an encoder network f

and a decoder network g, with the total loss:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{mismatch}} + \lambda \mathcal{L}_{\text{recons}}, \tag{5.12}$$

where λ is a parameter controlling the importance of the reconstruction loss. Note that both f and g can be multilayer neural networks. As the autoencoder encodes domain-invariant information, we call it **d**omain-invariant **a**uto**e**ncoder (DAE).[1]

Fig. 5.1 shows the architecture of DAE for the case of three domains ($D = 3$), with each row corresponding to one domain. Note that the weights in the rows are shared across all domains. Fig. 5.2 shows a scatter plot of 2-dimensional t-SNE embedding

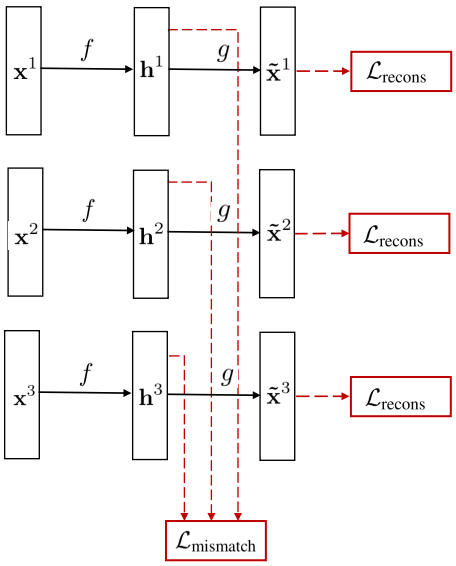

Figure 5.1: Architecture of the proposed domain-invariant autoencoder (DAE) when data are from three different domains. Solid black arrows represent the connections between neurons. Dashed red arrows represent the hidden nodes' outputs for computing the domain-mismatch loss or autoencoder's outputs for computing the reconstruction loss.

of the hidden activations of a DAE. Compared with the t-SNE plot in Fig 4.3(a), we

[1]Not to be confused with the denoising autoencoder of Vincent *et al* [60].

can see that the embedding of the hidden activations have apparently less domain-clustering effect than the embedding of i-vectors, which shows that the DAE indeed learns a domain-invariant representation.

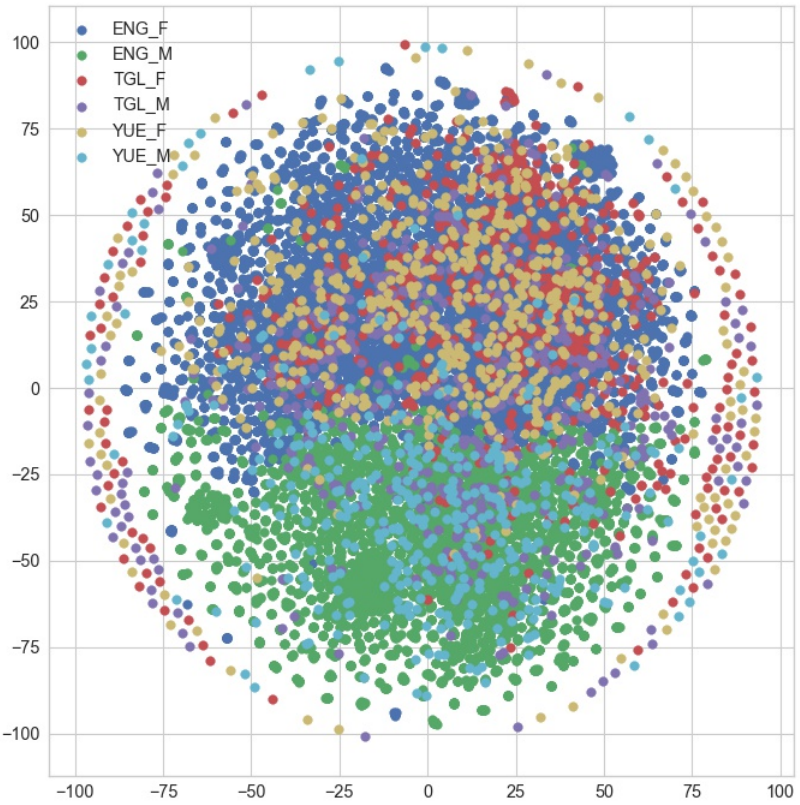

Figure 5.2: Scatter plot of 2-dimensional t-SNE embedding of the hidden activations of DAE. In the legend, "M" and "F" stand for male and female, respectively, and "ENG", "CAN" and "TGL" stand for English, Cantonese, and Tagalog, respectively.

5.2.3 Nuisance-attribute Autoencoder

In addition to directly learning domain-invariant features, we can also train an autoencoder to remove the domain-specific features similar to the idea of IDVC. Note

that Eq. 4.1 can be written as:

$$\hat{\mathbf{x}} = \mathbf{x} - \mathbf{W}\mathbf{W}^{\mathsf{T}}\mathbf{x}. \tag{5.13}$$

The columns of matrix $\mathbf{W}^{\mathsf{T}}$ span the nuisance subspace. Thus, $\mathbf{W}^{\mathsf{T}}\mathbf{x}$ represents a nuisance vector in the subspace. $\mathbf{W}\mathbf{W}^{\mathsf{T}}\mathbf{x}$ represents the nuisance vector in the original space. Instead of using principal component analysis (PCA) as in IDVC, we may use an autoencoder to estimate the nuisance components. Specifically, Eq. 5.13 can be replaced by:

$$\begin{aligned} \hat{\mathbf{x}} &= \mathbf{x} - \tilde{\mathbf{x}} \\ &= \mathbf{x} - g(f(\mathbf{x})), \end{aligned} \tag{5.14}$$

where $\tilde{\mathbf{x}}$ is the output of the autoencoder. $f(.)$ and $g(.)$ are implemented by an encoder and a decoder network, respectively. The parameters of $f(.)$ and $g(.)$ are learned using backpropagation. Similar to the DAE, the autoencoder has an encoder $\mathbf{h} = f(\mathbf{x})$ and a decoder $\tilde{\mathbf{x}} = g(\mathbf{h})$. But unlike the DAE, the autoenconder is trained to make $\hat{\mathbf{x}}$ as close to $\mathbf{x}$ as possible. The subtraction in Eq. 5.14 will make $\tilde{\mathbf{x}}$'s to contain all of the domain-specific information and $\hat{\mathbf{x}}$ to become less domain-dependent.

To achieve the goal mentioned above, we can use MMD to measure the discrepancy between the distribution of $\hat{\mathbf{x}}$ across different datasets:

$$\mathcal{L}_{\text{mismatch}} = \sum_{d=1}^{D} \sum_{\substack{d'=1 \\ d' \neq d}}^{D} \left(\frac{1}{N_d^2} \sum_{i=1}^{N_d} \sum_{i'=1}^{N_d} k(\hat{\mathbf{x}}_i^d, \hat{\mathbf{x}}_{i'}^d) - \frac{2}{N_d N_{d'}} \sum_{i=1}^{N_d} \sum_{j=1}^{N_{d'}} k(\hat{\mathbf{x}}_i^d, \hat{\mathbf{x}}_j^{d'}) + \frac{1}{N_{d'}^2} \sum_{j=1}^{N_{d'}} \sum_{j'=1}^{N_{d'}} k(\hat{\mathbf{x}}_j^{d'}, \hat{\mathbf{x}}_j^{d'}) \right). \tag{5.15}$$

Also, we can add reconstruction loss between $\mathbf{x}$ and $\hat{\mathbf{x}}$. As this network encodes the unwanted domain-specific information, we call it **n**uisance-attribute **a**uto**e**ncoder (NAE). Fig. 5.3 shows the architecture of NAE for the case of three domains ($D = 3$).

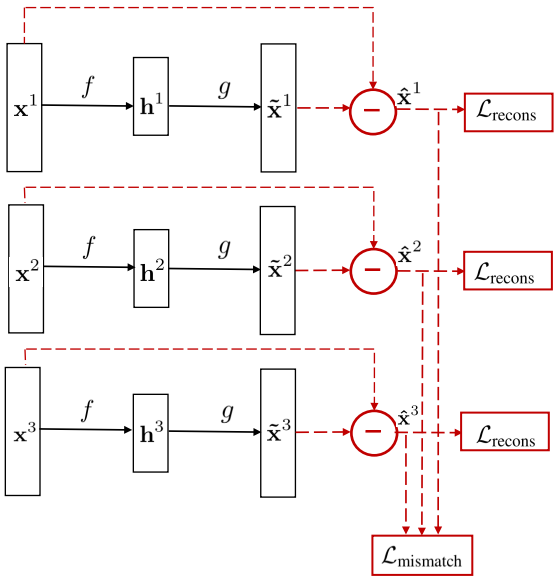

Figure 5.3: Architecture of the proposed nuisance-attribute autoencoder (NAE) when data are from three different domains. Solid black arrows represent the connections between neurons. Dashed red arrows represent the signal pathways for computing the domain-mismatch loss or reconstruction loss.

5.2.4 Semi-supervised Nuisance-attribute Networks

Supervised DA can leverage the speaker labels in the out-of-domain data through the center or triplet loss. Center loss was proposed in [61]. The motivation of center loss is that the softmax loss only considers the separation of classes but it fails to take intra-class distances into account. By introducing a center for each class and minimizing the distances between data and class centers, center loss can help networks learn more discriminative features. We refer to the nuisance-attribute networks with both unsupervised loss and supervised loss as semi-supervised nuisance-attribute networks (SNANs). To apply center loss to train an SNAN, we consider the network outputs $\hat{\mathbf{x}}_i$'s as the feature vectors, where i indexes the training samples in a mini-batch. Also denote $y_i \in \{1, \ldots, K\}$ as the class label for the i-th training sample. Then the center

loss is defined as:

$$\mathcal{L}_{\text{center}} = \frac{1}{2}\sum_{i=1}^{B_{y_i}} \|\hat{\mathbf{x}}_i - \mathbf{c}_{y_i}\|^2 , \tag{5.16}$$

where $\mathbf{c}_{y_i}$ is the center of the y_i-th class and B_{y_i} is the number of samples in the mini-batch. Note that $\mathbf{c}_{y_i}$ should be updated using the $\hat{\mathbf{x}}_i$'s in the mini-batch.

Triplet loss was proposed in [62]. Similar to center loss, triplet loss is also based on the idea of maximizing inter-class distance and minimizing intra-class distance. The key components are triplets. A triplet consists of three samples, namely, anchor sample $\hat{\mathbf{x}}_a$, positive sample $\hat{\mathbf{x}}_p$ and negative sample $\hat{\mathbf{x}}_n$. Positive sample shares the same class with the anchor while the negative sample comes from different classes. To learn discriminative features, we need to maximize inter-class distance while minimizing intra-class distance. This can be achieved by minimizing the following loss function:

$$\mathcal{L}_{\text{triplet}} = \max\{\|\hat{\mathbf{x}}_a - \hat{\mathbf{x}}_p\|^2 - \|\hat{\mathbf{x}}_a - \hat{\mathbf{x}}_n\|^2 + m,\ \ 0\}, \tag{5.17}$$

where m is a margin term. We can also incorporate supervised loss function such as center loss and triplet loss into the total loss:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{mismatch}} + \alpha\mathcal{L}_{\text{recons}} + \beta\mathcal{L}_{\text{supervised}}, \tag{5.18}$$

where β is a parameter controlling the importance of supervised loss. Eq. 5.18 enables the SNAN to leverage both labeled and unlabeled data. For labeled data, all of the three terms will be involved in the minimization, whereas for unlabeled data, the supervised loss is disabled by setting β to 0. In this way, we can make use of both labeled out-domain data and unlabeled in-domain data. Fig. 5.4 shows the architecture of SNAN for the case of three domains ($D = 3$).

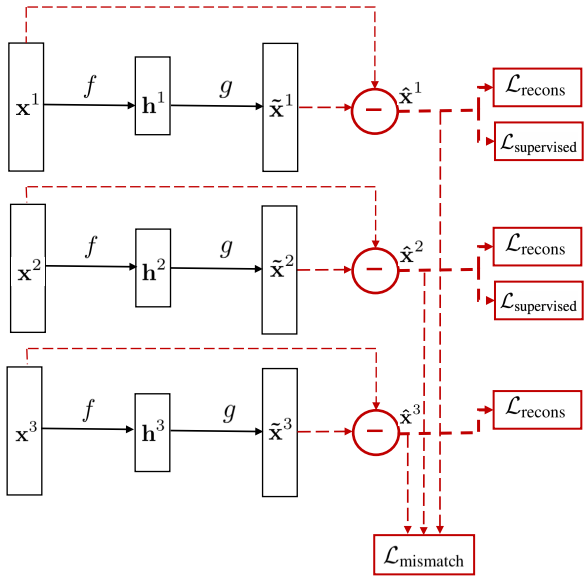

Figure 5.4: Architecture of the proposed semi-supervised nuisance-attribute network (SNAN) when data are from three different domains. Solid black arrows represent the connections between neurons. Dashed red arrows represent the signal pathways for computing the domain-mismatch loss $\mathcal{L}_{\text{mismatch}}$, reconstruction loss $\mathcal{L}_{\text{recons}}$ or supervised loss $\mathcal{L}_{\text{supervised}}$ such as center loss or triplet loss. Note that for $\mathbf{x}_3$ we do not have $\mathcal{L}_{\text{supervised}}$, which shows the semi-supervised nature of SNAN.

5.3 Experimental Setup

5.3.1 Speech Data and Acoustic Features

Speech files from NIST 2004–2010 Speaker Recognition Evaluation (hereafter, referred to as SRE04–SRE10)[2] and the development set of SRE16 (SRE16-dev) was used as the development data and speech files from the evaluation set of SRE16 (SRE16-eval) were used as test data. The speech regions in the speech files were extracted by using a two-channel voice activity detector [63]. For each speech frame, 19 MFCCs together with energy plus their 1st and 2nd derivatives were computed, followed by cepstral mean normalization and feature warping [64] with a window size of three seconds. A 60-dim acoustic vector was extracted every 10ms, using a Hamming window of 25ms.

5.3.2 I-vector Extraction and PLDA Model Training

SRE16 development data were used to train a gender-independent UBM with 512 mixture components and a total variability (TV) matrix with 300 total factors. After training the UBM and the TV matrix, i-vectors were extracted from speech files in SRE04–SRE10, SRE16–dev, and SRE16–eval.

Unless stated otherwise, i-vectors derived from SRE04–SRE10 and the SRE16 development set were used for training the DAEs, the NAEs, and the projection matrices of IDVC. Note that the non-English speech in SRE04–SRE10 were filtered out. The resulting networks and projection matrices were then used to transform i-vectors derived from SRE04–SRE10 and SRE16. Then, we computed a PCA projection matrix by using the transformed i-vectors from SRE04–SRE10 and reduced the dimension of i-vectors to 200. Length normalization was applied to the 200-dimensional i-vectors [10]. Then, we trained a gender-independent PLDA model with 200 latent variables using SRE04–SRE10 data only. PLDA scores were normalized by S-norm

[2]https://www.nist.gov/itl/iad/mig/speaker-recognition

using SRE16 development data as the cohort set [65].

Speech files with bad recordings (e.g., without speech or contain telephone tones only) as detected by the VAD and speech files shorter than 10 seconds were excluded from training any models (UBM, TV matrix, and PLDA) and networks (DAEs and NAEs).

5.3.3 MMD Autoencoders and IDVC Training Details

We used quadratic kernels and Gaussian kernels in the MMD and utilized a softmax (multi-class logistic regression) domain classifier to determine the best hyperparameters of the RBF kernel. Specifically, for each candidate RBF width, we trained a DAE and extracted vectors from the hidden nodes' activations in Fig. 5.1 and used them as the input to a softmax classifier with the number of outputs equals to the number of domains.[3] Similarly, another softmax domain classifier was trained to classify the domain-removed vectors $\hat{\mathbf{x}}$ in Fig. 5.3. Because our goal is to make these vectors as domain-invariant as possible, we selected the RBF width such that the resulting MMD vectors and domain-removed vectors lead to the lowest accuracy in the domain classifiers.

Fig. 5.5 shows the classification accuracy of the MMD vectors $\mathbf{h}$'s with respect to the RBF width. Evidently, the MMD vectors contain the least domain information when $\sigma = 1$, as they lead to the lowest domain classification accuracy. For both DAE and NAE, the weights in the decoder and encoder networks are always tied as in [66]. Unless stated otherwise, we divided SRE04–10 and SRE16 data into gender- and language-homogenous subsets to train the IDVC projection matrices, the DAEs, and the NAEs. Excluding the minor data in SRE16, we have six subsets: English male, English female, Cantonese male, Cantonese female, Tagalog male and Tagalog female. The networks were optimized using the Limited-memory Broyden—

[3]Here, we define a softmax classifier as a single-layer network with softmax output.

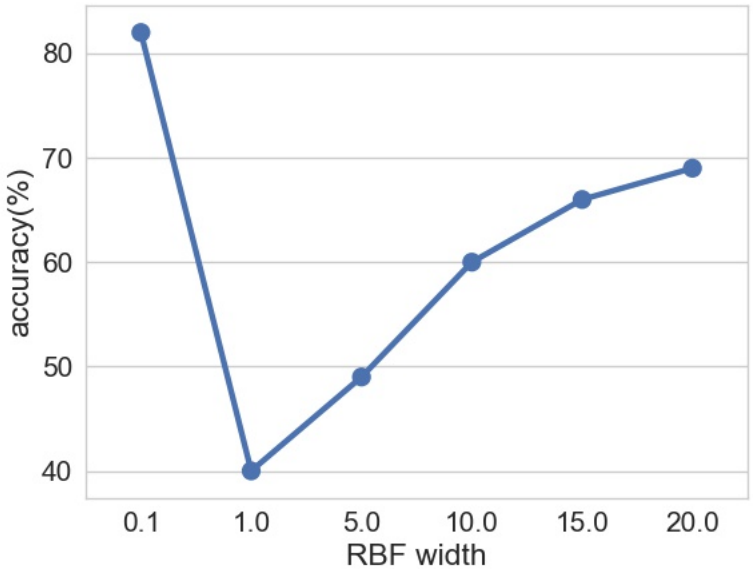

Figure 5.5: The relationship between the width σ of the radial basis function kernel and the accuracy of the softmax domain classifier on the features extracted from a DAE.

Fletcher—Goldfarb—Shanno (L-BFGS) algorithm [67, 68]. The history size of L-BFGS was set to 20 and the learning rate was set to 1. Optimizations were stopped when the difference of loss between two iterations was smaller than 0.0001.

5.4 Results and Discussions

5.4.1 General Performance Analysis

Table 5.1 shows the performance of four i-vector adaptation methods. All systems use PLDA as their backend. A classical i-vector PLDA system without domain adaptation (No Adapt) is also included for comparison. For the DAEs and NAEs, we used a quadratic kernel with $c = 1$ and λ in Eq. 5.12 was set to 1. Both linear and non-linear autoencoders were used in the experiments.

We can also see from Table 5.1 that except for non-linear DAE, all of the DA

methods boost the performance significantly in term of EER, although in terms of minimum Cprimary and actual Cprimary, the improvement is minor. We can also observe that the linear DAE and NAE have a small improvement over IDVC. Surprisingly, the non-linear DAE and NAE perform worse than their linear counterparts. When we look into the losses of these autoencoders, we found that the non-linear DAE and NAE produce higher losses than their linear counterparts. Considering that non-linear autoencoders should have higher capacity in fitting the training data (but they fail to do so), they probably got stuck in local minima [12, 69, 70]. Because of the relatively poor performance of non-linear autoencoders, we only present and discuss the results of linear autoencoders in the sequel.

In [71], the authors showed that a linear autoencoder works like PCA if their weights are found by minimizing the mean-squared error (MSE). A natural question that arises is whether the linear DAE and NAE are the same as PCA. Note that in [71], the objective function of the autoencoders comprises the MSE loss only. On the other hand, the objective function of DAE and NAE comprises both the MMD loss and MSE loss. As MMD loss is totally different from MSE loss, DAEs and NAEs will not reduce to PCA even with linear units. To demonstrate that our linear NAE and DAE are different from PCA, we also report results obtained by using PCA to preprocess the i-vectors in Table 5.1. Evidently, using PCA alone could not improve the performance significantly.

To gain more insights into the performance of IDVC, DAEs, and NAEs, we report the performance of the three systems on four gender- and language-dependent subsets in Table 5.2(a) and Fig 5.6. The results suggest that Tagalog is more challenging than Cantonese, with EERs of 20.55% and 19.89% for males and females, respectively. Because the Tagalog and the Cantonese speech were collected in the regions with different telephone systems, the performance gap could be caused by the difference in the quality of the telephone systems. Also, the female subsets seem to be more difficult than the male ones. The performance of the four subsets improves significantly after

applying the three domain adaptation methods.

5.4.2 Robustness to Unseen Domains

In the previous section, we partitioned the training data into gender- and language-homogenous groups. There are always data in the training set that match both the gender and the language of the test set. However, it is not always feasible to obtain training data that match the gender and language of the test data for domain adaptation. Therefore, we conducted a domain robustness experiment. Specifically, for each gender and language (Tagalog or Cantonese) in test sessions, we excluded the speech of the same gender who speaks that language from training. In other words, there is no in-domain data for domain adaptation. Here, the term "domain" refers to genders and languages, and in-domain data are evaluation data with a specific combination of gender and language. For example, for the evaluation of male Tagalog, we exclude male Tagalog data for training the IDVC, DAE and NAE. Note that the gender and language information can be obtained from the key file of the development data provided by NIST.

Table 5.2(b) shows the results of the three DA methods on unseen domains. Fig. 5.6 shows the EERs of the DA methods with and without using in-domain training data. Not surprisingly, without in-domain data for training, the performance of all DA methods degrades. Despite the performance degradation, the performance of these DA methods is still better than the one without domain adaptation. More importantly, DAE and NAE appear to suffer less when compared with IDVC. In particular, for Cantonese speakers, the DAE has 6.8% and 5.2% relative improvement over IDVC for female and male speakers, respectively. We believe that the incorporation of high moments of the data distributions is the reason that MMD-based methods are more robust to unseen domains.

5.4.3 Impact of the Hyperparameters

Comparing with IDVC, DAEs, and NAEs have more hyper-parameters to tune. In this subsection, we present the results of DAEs and NAEs with different values of λ and different choices of kernels. The kernels we experimented with include quadratic kernels with $c = 0$ and $c = 1$, RBF kernels with $\sigma = 1$, and the mixture of four RBF kernels with width equals to 1, 3, 5, and 10, respectively. Tables 5.3(a) and 5.3(b) show the results of DAEs and NAEs with different choices of λ's and kernels. Fig. 5.7 shows the EERs of DAEs and NAEs with different choices of λ and kernels.

Three phenomena can be observed from the results in Table 5.3 and Fig. 5.7. Firstly, quadratic kernels and RBF kernels require different values of λ to obtain good performance. Specifically, both NAEs and DAEs with quadratic kernels perform the best when λ is equal to 0.1 or 1, while NAEs and DAEs with RBF kernels perform the best when λ is equal to 0.01 or 0.1. Secondly, for NAEs, the quadratic kernel with $c = 0$ generally performs poorly in most cases. Recall that c controls the trade-off between the matching in the first and the second moments. It seems that matching the second moment alone is not enough in most cases. Thirdly, there is no noticeable performance gain from using RBF kernels or a mixture of RBF kernels. Theoretically, RBF kernels can match up to infinite moments of data distributions and are therefore better than quadratic kernels for reducing the domain mismatch. However, in our experiments, RBF kernels have no advantage over quadratic kernels. It may be due to the fact that PLDA only uses up to the second moment.

5.4.4 Impact of Data Partition

In the previous sections, we partitioned i-vectors by both genders and languages. In this section, we investigated the influence of different partitioning schemes. According to gender and language, we can partition data into gender-homogenous groups, language-homogenous groups, or gender- and language-homogenous groups. Table 5.4

shows the results of the DA methods using different partitioning schemes. For all of the three DA methods, it is apparent that partitioning by both genders and languages achieves the best result, which clearly demonstrates the advantage of multi-source domain adaptation.

5.4.5 Combined with PLDA Model Interpolation

It is also possible to combine i-vector domain adaptation with PLDA model interpolation [28]. Specifically, the unlabelled i-vectors in the target domain were clustered to obtain some hypothesized labels, which were used to train a PLDA model. As PLDA models obtained in this way may not be very reliable, a better approach is to linearly interpolate the parameters of the source as well as the target PLDA models [28]. Table 5.5 shows the results of i-vector adaptation in combination with model interpolation. The interpolation parameter was set to 0.3. When compared with using the unadapted i-vectors, it is clear that adapting the i-vectors improves the performance of the PLDA model interpolation. Also, the best performance was achieved by the proposed DAE.

5.4.6 Performance Analysis of Supervised DA

Table 5.6 shows the performance of IDVC and SNAN with only unsupervised loss functions. All systems use a PLDA backend. A classical i-vector PLDA system without domain adaptation (No Adapt) is also included for comparison. The SNAN uses a quadratic kernel with $c = 1$ and α and in Eq. 5.12 was set to 1.

For the i-vector systems, we can see from Table 5.6 that both IDVC and SNAN improve the performance in term of EER, although in terms of minimum Cprimary and actual Cprimary, the improvement is minor. We can also observe that SNAN performs better than IDVC by a small margin.

For the x-vector systems, surprisingly IDVC degrades the performance. Although SNAN still outperforms the baseline system, the improvement is marginal. It could

be that the x-vectors are more robust to domain mismatch, which diminishes the benefit of domain adaptation.

5.4.7 Impacts of Supervised Loss Functions

In this section, we investigate the impacts of supervised loss functions on the performance of SNANs. There are three supervised loss functions, namely, cross-entropy loss, center loss as in Eq. 5.16 and triplet loss as in Eq. 5.17. Table 5.7 shows the results of SNANs with these loss functions together with SNANs with only unsupervised loss functions. For i-vector based systems, we can see that in general adding supervised losses indeed improves the performance of SNANs. However, there is no clear winner among the three supervised loss functions. For x-vector based systems, adding supervised loss functions does not give a significant improvement in performance.

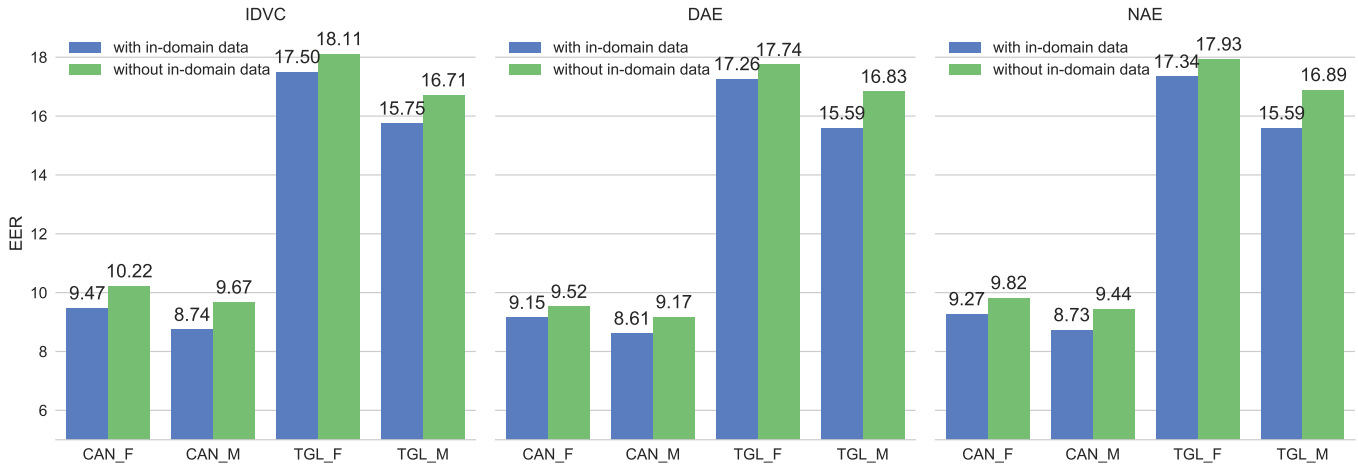

Figure 5.6: Bar charts showing the EERs of three domain adaptation methods with and without using in-domain data.

Table 5.1: The Performance of four domain adaptation methods and the performance of a classical i-vector PLDA system without domain adaptation (*No Adapt*) in the SRE16 evaluation set. "Linear" and "sigm" mean that the hidden nodes in the DAE and NAE use linear and sigmoid activation functions, respectively. *PCA*: The weights of the linear autoencoder were found by PCA. *IDVC*: Inter-dataset variability compensation. "mCprim" and "aCprim" are the minimum detection cost and the actual detection cost as specified in the evaluation plan of SRE16.

	Architecture	EER (%)	mCprim	aCprim	$\mathcal{L}_{\text{mismatch}}$	$\mathcal{L}_{\text{recons}}$	$\mathcal{L}_{\text{total}}$
No Adapt	-	15.84	0.89	0.93	-	-	-
PCA	-	14.77	0.89	0.92	-	-	-
IDVC	-	13.08	0.86	0.93	-	-	-
Linear DAE	300-300(linear)-300	**12.79**	0.85	0.91	0.002	0.012	0.014
Non-linear DAE	300-300(sigm)-300(linear)-300(linear)-300(sigm)-300	24.36	0.98	0.99	0.001	0.220	0.221
Linear NAE	300-10(linear)-300	12.81	0.85	0.91	0.003	0.211	0.214
Non-linear NAE	300-10(sigm)-10(linear)-10(linear)-10(sigm)-300	14.73	0.93	0.93	0.156	2.032	2.188

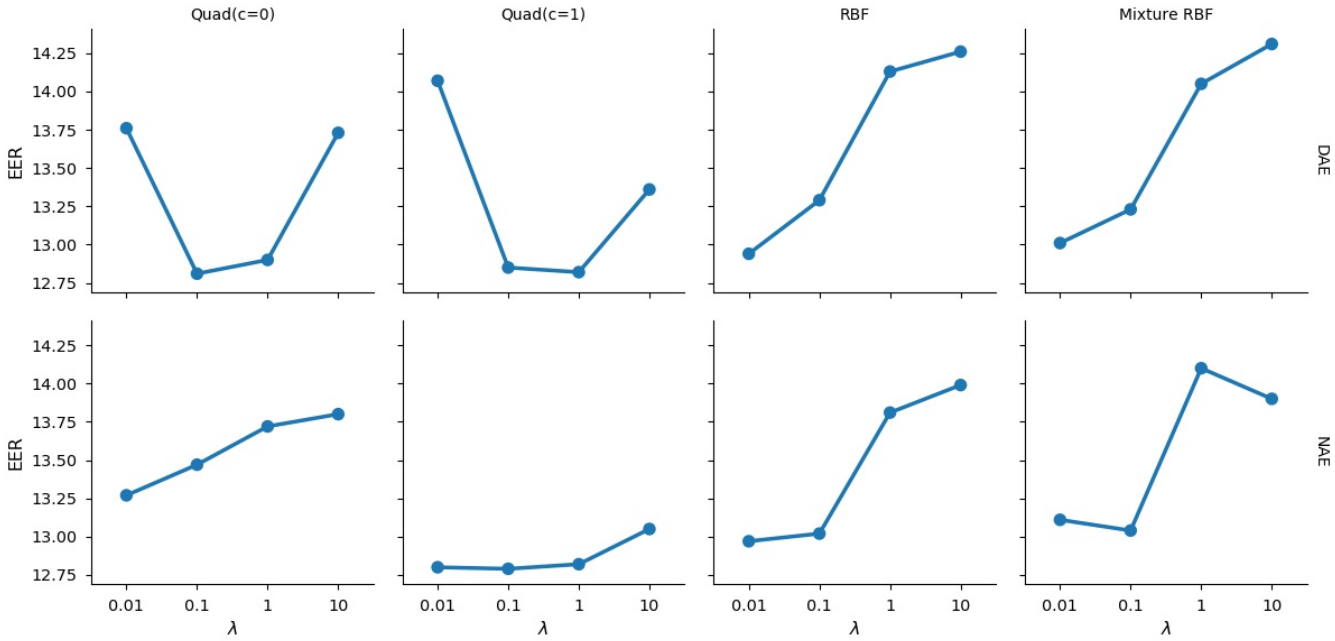

Figure 5.7: Line plots showing the EERs of DAEs (top row) and NAEs (bottom row) with different choices of λ's and kernels.

Table 5.2: The performance of various domain adaptation methods on the subsets of the SRE16 evaluation set. In (a), the IDVC, DAE and NAE were trained using both in-domain data and out-domain data. In (b), The IDVC, DAE and NAE were trained without using in-domain data.

	Cantonese						Tagalog					
	Female			Male			Female			Male		
	EER	mCprim	aCprim	EER	mCprim	aCprim	EER	mCprim	aCprim	EER	mCprim	aCprim
No Adapt	10.92	0.77	0.87	10.87	0.74	0.96	20.55	0.93	0.94	19.89	0.94	0.96
IDVC	9.47	0.74	0.88	8.74	0.68	0.96	17.50	0.91	0.93	15.75	0.90	0.96
DAE	**9.15**	**0.73**	0.84	**8.61**	0.67	0.94	**17.26**	0.90	0.91	15.59	0.89	0.94
NAE	9.27	0.74	**0.83**	8.73	0.67	0.94	17.34	0.90	0.91	15.59	0.89	0.94

(a)

	Cantonese						Tagalog					
	Female			Male			Female			Male		
	EER	mCprim	aCprim	EER	mCprim	aCprim	EER	mCprim	aCprim	EER	mCprim	aCprim
No Adapt	10.92	0.77	0.87	10.87	0.74	0.96	20.55	0.93	0.94	19.89	0.94	0.96
IDVC	10.22	0.75	0.87	9.67	0.70	0.96	18.11	0.91	0.93	**16.71**	0.92	0.95
DAE	**9.52**	**0.73**	0.83	**9.17**	**0.68**	0.95	**17.74**	0.91	**0.91**	16.83	**0.91**	0.94
NAE	9.82	0.74	**0.82**	9.44	0.69	0.95	17.93	**0.91**	0.92	16.89	0.92	0.94

(b)

Table 5.3: The performance of (a) DAEs and (b) NAEs with different choices of kernels and λ's. Quad is the quadratic kernel in Eq. 5.4. Both DAEs and NAEs were trained by partitioning SRE04–10 and SRE16 development data into gender- and language-homogenous groups.

	Quad(c=0)			Quad(c=1)			RBF			Mixture RBF		
λ	EER	mCprim	aCprim	EER	mCprim	aCprim	EER	mCprim	aCprim	EER	mCprim	aCprim
0.01	13.76	0.86	0.93	14.07	0.87	0.91	**12.94**	0.85	0.92	**13.01**	0.85	0.90
0.1	**12.81**	**0.85**	**0.91**	12.85	**0.84**	0.90	13.29	0.85	0.90	13.23	0.85	0.91
1	12.90	0.86	0.93	**12.79**	0.85	0.91	14.13	0.87	0.89	14.05	0.88	0.91
10	13.73	0.87	0.89	13.36	0.86	0.90	14.26	0.87	0.89	14.31	0.87	0.89

(a)

	Quad(c=0)			Quad(c=1)			RBF			Mixture RBF		
λ	EER	mCprim	aCprim	EER	mCprim	aCprim	EER	mCprim	aCprim	EER	mCprim	aCprim
0.01	**13.27**	0.86	0.91	12.80	0.85	0.91	**12.97**	0.85	0.92	13.11	0.85	0.91
0.1	13.47	**0.85**	0.93	**12.79**	0.85	0.91	13.02	0.85	0.90	**13.04**	0.85	0.91
1	13.72	0.86	**0.90**	12.81	0.85	0.91	13.81	0.85	**0.89**	14.10	0.85	**0.90**
10	13.80	0.86	0.91	13.05	0.85	0.91	13.99	0.85	0.92	13.90	0.88	0.91

(b)

Table 5.4: The performance of IDVC, DAE and NAE using different partition schemes.

Partitioning Scheme	IDVC		
	EER	mCprim	aCprim
Gender and Language	13.08	0.86	0.93
Gender	13.75	0.87	0.9
Language	13.59	0.85	0.93

Partitioning Scheme	DAE		
	EER	mCprim	aCprim
Gender and Language	12.79	0.85	0.91
Gender	13.09	0.85	0.90
Language	13.29	0.85	0.90

Partitioning Scheme	NAE		
	EER	mCprim	aCprim
Gender and Language	12.81	0.85	0.91
Gender	13.03	0.86	0.92
Language	13.29	0.85	0.90

Table 5.5: The performance of unsupervised PLDA model interpolation using unadapted i-vectors and i-vectors adapted by IDVC, DAEs and NAEs. The interpolation parameter was set to 0.3

Adaptation Method	EER	mCprim	aCprim
No Adaptation	13.47	0.86	0.91
IDVC	12.88	0.85	0.93
DAE	12.43	0.84	0.90
NAE	12.51	0.84	0.91

Table 5.6: The performance of PLDA without adaptation, IDVC, and SNAN with supervised loss only ($\beta = 0$ in Eq. 5.18) on the SRE16 evaluation set. "mCprim" and "aCprim" are the minimum detection cost and the actual detection cost as specified in the evaluation plan of SRE16.

Feature	Adaptation	EER(%)	mCprim	aCprim
i-vector	No Adapt	12.78	0.74	0.94
	IDVC	12.17	0.73	0.9
	SNAN	11.95	0.72	0.87
x-vector	No Adapt	10.74	0.65	0.86
	IDVC	11.24	0.65	0.89
	SNAN	10.35	0.61	0.81

Table 5.7: The performance of SNAN using different supervised loss functions on SRE16 using i-vectors and x-vectors from the Kaldi's SRE16 recipe.

Feature	Supervised Loss	EER(%)	mCprim	aCprim
i-vector	None	11.95	0.72	0.87
	Softmax	11.61	0.71	0.87
	Softmax+Center	11.76	0.72	0.86
	Triplet	11.67	0.72	0.85
x-vector	None	10.35	0.61	0.81
	Softmax	10.28	0.61	0.81
	Softmax+Center	10.31	0.61	0.81
	Triplet	10.57	0.62	0.80

Chapter 6

DNN SPEAKER EMBEDDING ADAPTATION USING MMD

In this chapter, we propose a novel DNN adaptation method for speaker recognition across languages. Most of the previous works in DNN adaptation focus on utterances-level adaptation, i.e., the domain divergence is estimated from the speaker embeddings. Although backpropagation could tune the parameters of the whole network to minimize the domain divergence, we argue that it is still better to explicitly adapt frame-level features. The reason is that training data typically only has a fixed duration range. As a result, adapting utterance-level features may only benefit a specific duration range. Adapting frame-level features, on the other hand, is less prone to overfitting a specific duration range only.

Another innovative aspect of our method is making use of unlabeled target-domain data. Data augmentation is an important part of DNN-based SV. However, the augmentation is done only for the labeled data. It was unclear how to apply data augmentation to the unlabeled data. To leverage the unlabeled data, we propose to add a consistency regularization that minimizes the divergence between the augmented and the clean target-domain data. In this way, the consistency regularization makes the target speaker embeddings robust to adverse perturbations. We also propose auxiliary batch normalization in each layer of the neural networks. This is, to the best of our knowledge, the first use of separate batch normalization units in domain adaptation. In summary, we investigates the use of **m**ultilevel domain loss, **s**eparate batch normalization, and **c**onsistency regularization for speaker embeddings. We abbreviate

the proposed method as MSC.

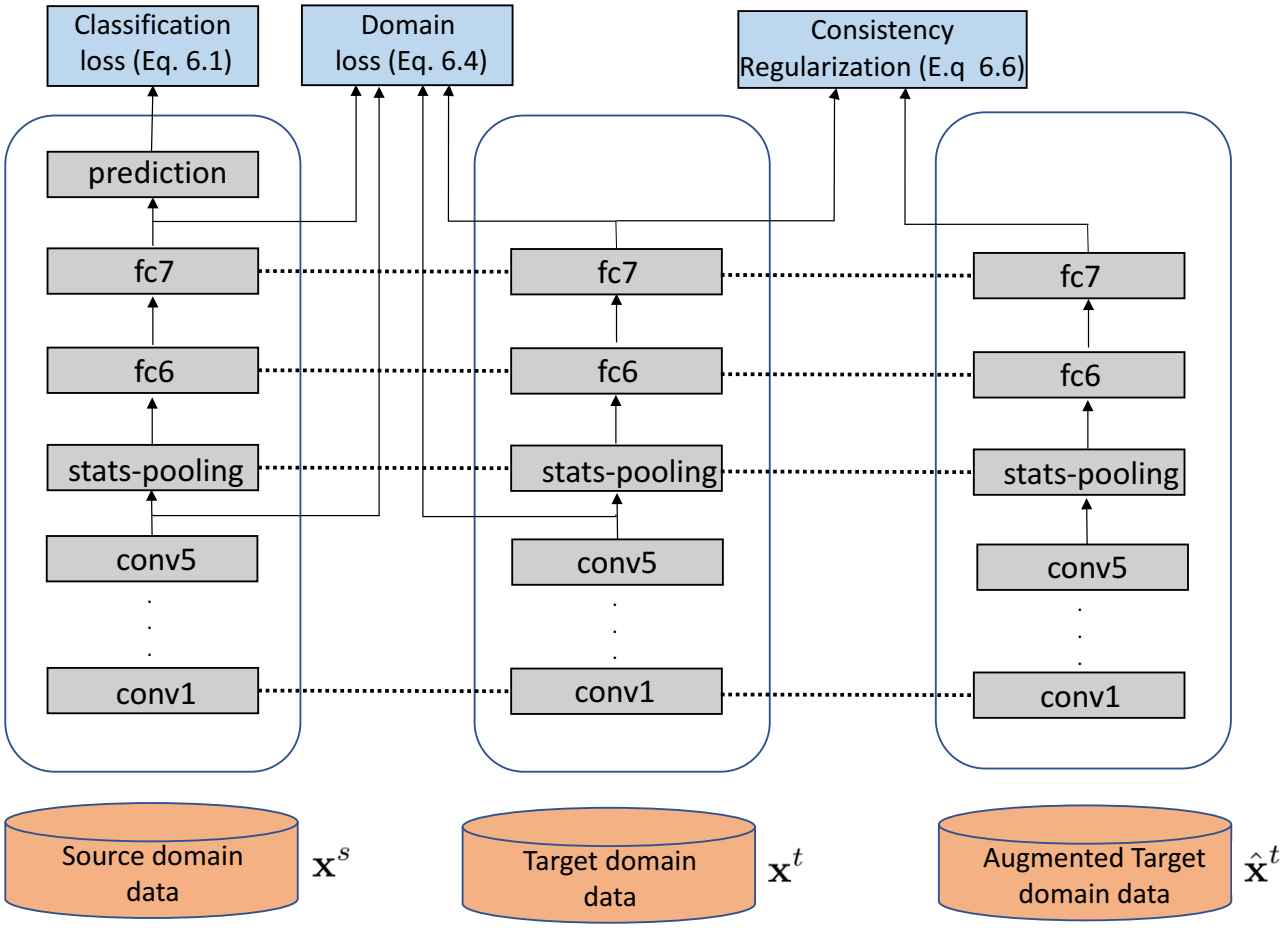

Figure 6.1: The architecture of our proposed framework. The network is trained to minimize the classification loss and the domain loss with consistency regularization (see Eq. 6.6). For target-domain data, no label is required. The dotted lines indicate weight-sharing within individual layers.

6.0.1 Multi-level Adaptation

Assuming that we have a labeled dataset $\{(\mathbf{o}_i^s, y_i^s)\}_{i=1}^I$ from the source-domain and the set of network parameters $\Theta = \{\mathbf{W}^l, \mathbf{b}^l\}_{l=1}^L$, the objective function of the network

can be written as

$$\min_{\Theta} \frac{1}{I} \sum_{i=1}^{I} J(p_{\Theta}(y|\mathbf{o}_i^s), y_i^s), \tag{6.1}$$

where J is the cross-entropy function and $p_{\Theta}(y|\mathbf{o}_i^s)$ is the conditional probability that the network assigns $\mathbf{o}_i^s$ to the class y. Minimizing this objective alone will not guarantee the generalization to the target-domain. To make generalization to the target-domain possible, we need to reduce the divergence between the marginal distribution of the source-domain and the target-domain. In neural networks, we typically reduce the divergence in the hidden activations. Let $\mathcal{H}_*^l = \{\mathbf{h}_i^l\}$ denotes the activations at the l-th layer for source or target data. The cross-entropy loss together with MMD distance is

$$\min_{\Theta} \frac{1}{I} \sum_{i=1}^{I} J(p_{\Theta}(y|\mathbf{o}_i^s), y_i^s) + \lambda \cdot \mathcal{D}(\mathcal{H}_s^l, \mathcal{H}_t^l), \tag{6.2}$$

where λ is a constant controlling the trade-off between the two objectives.

As mentioned in [59], upper layers typically have larger domain discrepancy gaps. Therefore, it is very common to minimize the divergence at the network's last layer. In the x-vector network shown in Table 3.1, it is the 7-th layer (i.e., $l = 7$). The current DNN training scheme typically uses very short speech segments (200 frames to 400 frames) for training and relies on the backend to compensate for the duration discrepancy during the verification phase. However, the embedding distribution shifts with speech duration. The divergence estimated from short utterances is a poor substitution for the divergence estimated from long utterances. Adapting frame-level activations, on the other hand, has no such problem. Therefore, we argue that it is important to adapt frame-level features as well. Here, we choose the last convolutional layer before statistics pooling, i.e., $l = 5$, for adaptation. However, the MMD distance in Eq. 5.1 only works with vectors. Frame-level activations are represented by tensor

with dimension $B \times T \times C$, where B is the batch size, T is the number of frames in the segment, and C is the number of channels (filters). We propose to flatten along the temporal axis to convert the frame-level activations of speech segments into a batch of TC-dimensional vectors and then compute the MMD distance between the two batches of flattened vectors:

$$\mathcal{D}(\text{FLAT}(\mathcal{H}_s^5), \text{FLAT}(\mathcal{H}_t^5))), \tag{6.3}$$

where FLAT(.) denotes flattening a 3D tensor along the temporal axis. Fig. 6.2(b) illustrates the flattening procedure for frame-level activations. Minimizing Eq. 6.3 can reduce domain mismatch even though utterances from the source- and target-domains are of different contexts. This is because the filters (feature detectors) in the lower CNN layers can extract the relevant features irrespective of the location of the features. The total domain loss is:

$$\lambda \cdot \mathcal{D}(\mathcal{H}_s^7, \mathcal{H}_t^7) + \alpha \cdot \mathcal{D}(\mathcal{H}_s^5, \mathcal{H}_t^5)), \tag{6.4}$$

where α is a constant controlling the importance of Eq. 6.3.

6.0.2 Consistency Regularization Using MMD

Data augmentation plays an important role in the success of the x-vector framework. However, how to use data augmentation on unlabeled data has not been explored in SV. Consistency training has been successfully explored in semi-supervised learning [72]. The idea is to enforce or regularize a network such that the network predictions are consistent even if the network's input is subjected to noise perturbation for unlabeled data. In [72], the regularization is achieved by minimizing the following KL divergence:

$$\mathop{\mathbb{E}}_{q(\hat{\mathbf{o}}_{\text{unlab}}|\mathbf{o}_{\text{unlab}})}[\text{KL}(p_\Theta(y|\mathbf{o}_{\text{unlab}})||p_\Theta(y|\hat{\mathbf{o}}_{\text{unlab}}))], \tag{6.5}$$

where $\mathbf{o}_{\text{unlab}}$ denotes the clean unlabeled data, $\hat{\mathbf{o}}_{\text{unlab}}$ denotes the augmented unlabeled data, and $q(\hat{\mathbf{o}}_{\text{unlab}}|\mathbf{o}_{\text{unlab}})$ is a data augmentation transformation. For Eq. 6.5 to work, $p_\Theta(y|\mathbf{o}_{\text{unlab}})$ has to be very close to the true class distribution. This is typically achieved by training the network on a large amount of labeled data $\{\mathbf{o}_{\text{label}}, y\}$, where $\mathbf{o}_{\text{label}}$ denotes the labeled data. As a result, the network is discriminative for the class y and $p_\Theta(y|\mathbf{o}_{\text{unlab}})$ is very close to the true class distribution. However, if we do not have labeled data $\{\mathbf{o}_{\text{label}}, y\}$ to train the network, the softmax output $p_\Theta(y|\mathbf{o}_{\text{unlab}})$ is less useful.

We propose another form of consistency penalty that does not require labeled data to work. Instead of minimizing the KL divergence between the softmax output of the clean unlabeled data and the augmented unlabeled data, we propose minimizing the discrepancy between the embeddings produced by the clean data and the embeddings produced by the augmented data. The motivation is that DNN embedding should be robust to input perturbation. After all, the goal of the DNN is to create speaker embedding instead of prediction. We use MMD to measure consistency between the two sets of embeddings. Let $\mathcal{H}_t^7$ and $\hat{\mathcal{H}}_t^7$ denotes the set of Layer-7's hidden activations obtained from the clean and the augmented target-domain data, respectively. The consistency regularization using MMD is written as:[1]

$$\mathcal{D}(\mathcal{H}_t^7, \hat{\mathcal{H}}_t^7) = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{i'=1}^{N} k(\mathbf{h}_i^7, \mathbf{h}_{i'}^7) - \frac{2}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M} k(\mathbf{h}_i^7, \hat{\mathbf{h}}_j^7) + \frac{1}{M^2}\sum_{j=1}^{M}\sum_{j'=1}^{M} k(\hat{\mathbf{h}}_j^7, \hat{\mathbf{h}}_{j'}^7), \tag{6.6}$$

where N and M are the number of speech segments in source-domain and target-domain, respectively. By combining the consistency regularization with Eq. 6.5 we have the total loss function:

$$\min_\Theta \frac{1}{I}\sum_{i=1}^{I} J(p_\Theta(y|\mathbf{o}_i^s), y_i^s) + \lambda \cdot \mathcal{D}(\mathcal{H}_s^7, \mathcal{H}_t^7) + \alpha \cdot \mathcal{D}(\mathcal{H}_s^5, \mathcal{H}_t^5)) + \beta \cdot \mathcal{D}(\mathcal{H}_t^7, \hat{\mathcal{H}}_t^7). \tag{6.7}$$

[1]To simplify notation, the subscript t is omitted in the right side of Eq. 6.6.

Fig 6.1 summarizes the architecture and objective functions.

6.0.3 Auxiliary BN

Batch normalization (BN) is an essential part of modern deep neural networks. The input features are normalized using the mean and the variance computed from the mini-batch. The underlying assumption is that the input data are homogenous. However, if data are heterogeneous, the statistics used by BN is inaccurate. In domain adaptation, it is obvious that the source-domain data and the target-domain data come from two different distributions, as exemplified by the t-SNE plots in Fig. 6.3. Thus we argue that we need two separate BNs to obtain more accurate batch statistics. A similar idea is proposed in [73], where the authors used two different BNs for clean and adversary inputs. The heterogeneity could also come from data augmentation [73]. Thus, we could also use three BNs per layer, i.e., one for the source-domain data, one for the target-domain data, and one for the augmentation data. Fig. 6.4 illustrates the idea of auxiliary BN.

6.1 Experiments

6.1.1 Data Preparation

The training data include NIST SRE 2004–2010 (SRE04–10 in short) and all of the Switchboard data. We used the data augmentation strategy in the Kaldi SRE16 receipt. The training data were augmented by adding noise, music, reverb, and babble to the original speech files in the datasets. After filtering out utterances shorter than 500 frames and speakers with less than 8 utterances, we are left with 4,808 speakers. 23-dimensional Mel-frequency cepstral coefficients (MFCC) were computed from 8kHz speech files. Mean normalization was applied to the MFCC using a 3-second sliding window. Non-speech frames were removed using Kaldi's energy-based voice activity detector.

6.1.2 DNN and Backend Training

The value of λ, β and α in Eq. 6.7 were all set to 1. All DNNs were trained using a batch size of 32 and were optimized by the Adam optimizer [74] with a learning rate of 0.001. The networks were implemented in Pytorch [67]. We used a standard backend comprised of LDA, length-normalization, and PLDA. Both LDA and PLDA were trained using embeddings extracted from full-length utterances. We used correlation alignment [47] for domain adaptation in the PLDA backend.

6.1.3 Data Augmentation

In addition to data augmentation in [5], we also used speech conversion tools in FFmpeg.[2] Every speech file was accompanied by one of the following augmentations:

- Reverberation: Reverberation effect was added to the speech by convolving it with a simulated room impulse response (RIR) filter. [75].
- Music: A randomly selected music file from MUSAN was added to the original speech at an SNR of 5–15dB.
- Noise: Noise from MUSAN was added to the recording intermittently with SNR between 0dB and 10dB.
- Babble: Babble noise was added to the original speech with an SNR of 0dB–10dB.
- Speed: the original speech was speeded up by 1.3 percent using FFMpeg.

Both the target-domain data and the source-domain data were augmented.

[2]https://www.ffmpeg.org

Table 6.1: Comparison with other DNN adaptation methods. Sup. WGAN [1] uses the labels of SRE16 and SRE18 development data. There is no backend adaptation in all of the systems.

	SRE16		SRE18	
Adaption Method	EER (%)	minDCF	EER(%)	minDCF
WGAN [1]	13.25	0.899	9.59	0.652
Sup. WGAN [1]	9.59	0.746	8.88	0.619
LSGAN [76]	11.74	-	-	-
MSC	**8.69**	**0.556**	**7.95**	**0.500**

6.1.4 Evaluation

All systems were evaluated on the evaluation set of SRE 2016 and 2018. The SRE16 evaluation set is composed of Tagalog and Cantonese telephone conversations. For SRE18, we only conducted evaluations on the CMN2 portion, which consists of Tunisian Arabic conversations. We report results in terms of equal error rate (EER) and minimum cost function (DCF) using the scoring tools provided by NIST.

6.2 Results

6.2.1 Comparison with Other DNN Adaptations

In this section, we compared the proposed framework (MSC) with other DNN-based adaptation methods. The latter includes Wasserstein GAN (WGAN) adaptation, supervised WGAN adaptation [1], and least square GAN (LSGAN) [76]. The results are presented in Table 6.1. All the results in Table 6.1 are without additional backend adaptation. It is clear from the table that MSC performs significantly better than the existing methods. It is worth noting that MSC even performs better than the supervised adaptation in [1].

Table 6.2: The Performances of CORAL, PLDA adaptation and the proposed framework MSC.

	SRE16		SRE18	
Adaption Method	EER(%)	minDCF	EER(%)	minDCF
MSC	8.69	0.556	7.95	0.500
CORAL Adapt.	8.49	0.560	8.74	0.553
PLDA Adapt.	8.55	0.556	8.88	0.563
MSC+CORAL Adapt.	**8.13**	**0.530**	**7.70**	**0.486**
MSC+PLDA Adapt.	8.22	0.542	8.12	0.502

6.2.2 Comparison with Backend Adaptation

In this section, we compared the proposed framework with two popular backend adaptation methods, namely, CORAL [47] and Kaldi's PLDA adaptation [5]. The potential of combining the proposed framework with the backend adaptation methods was also investigated. The results are presented in Table 6.2. As can be seen from Table 6.2, in SRE16, CORAL is the most effective adaptation method. However, in SRE18, MSC has a clear advantage over the backend adaptation. This, we believe, is due to the fact that SRE18 has more data for adaptation (over 4,000 utterances compared with only 2,340 in SRE16). Besides, it seems that the proposed framework works well with the backend adaptation, as combining them improves the performance.

6.2.3 Ablation Study of individual components

To investigate the effect of individual components in our framework, we conducted an ablation study. Table 6.5 starts with only utterance-level adaptation (Utt. Adapt.) in the second row and incrementally adds frame-level adaptation (Frame Adapt.), consistency regularization (Consist. Reg.), and auxiliary BN (Aux. BN). We can see that utterance-level adaptation alone already gives a great improvement over no

Table 6.3: Our proposed framework with different numbers of BNs per layer. Refer to Fig. 6.4 for the details of BN types.

BN type	SRE16		SRE18	
	EER	minDCF	EER	minDCF
Standard BN	9.03	0.559	8.21	0.509
One Aux. BN	**8.69**	**0.556**	**7.95**	**0.500**
Two Aux. BNs	9.28	0.566	8.33	0.511

adaptation. Adding frame-level adaptation gives a considerably performance gain for both SRE16 and SRE18. Consistency regularization also further improves the performance in both SRE16 and SRE18. Finally, using auxiliary BN evidently helps.

6.2.4 Auxiliary Batch Normalization

The T-SNE plots in Fig. 6.3 motivate us to use separate batch normalizations for the source-domain data and the target-domain data. In this section, we investigated how the separate batch normalization affects performance. Specifically, we investigated three kind of batch normalization schemes. The first one is the vanilla BN that conflates the source-domain data and the target-domain data in a mini-batch. The second one uses two separate BNs for the source-domain data and the target-domain data, respectively. We refer to this as "One Aux. BN" in Table 6.3. The third one uses three BNs for the clean source-domain data, the clean target-domain data, and the augmented data, respectively. We refer to this as "Two Aux. BN" in Table 6.3 (also see Fig. 6.4). We can see from Table 6.3 that separating the source-domain data and the target-domain data improves the performance in both SRE16 and SRE18, while further separating the augmented data from clean data degrades the performance. One possible explanation is that when using three separate batch normalizations per layer, there are not enough data to compute the mean and the variance reliably. A

similar phenomenon was also reported in [77].

6.2.5 Influence of Network Architectures

Table 6.4: The performance of four DNN speaker embeddings with and without the proposed adaptation framework. The details of x-vector based networks and DenseNets are presented in Table 3.1 and Table 3.2, respectively.

(a) EER(%)

	DNN Embedding			
	X-vector	Wide-x-vector	Densenet61	Densenet121
SRE 16 w/o adapt	12.36	11.90	11.87	11.64
SRE 16 w/ adapt	8.69	8.37	8.31	7.81
SRE 18 w/o adapt	11.83	10.62	10.73	9.47
SRE 18 w/ adapt	7.95	7.63	7.65	7.02

(b) minDCF

	DNN Embedding			
	X-vector	Wide-x-vector	Densenet61	Densenet121
SRE 16 w/o adapt	0.985	0.851	0.854	0.820
SRE 16 w/ adapt	0.556	0.534	0.520	0.515
SRE 18 w/o adapt	0.761	0.692	0.703	0.600
SRE 18 w/ adapt	0.500	0.482	0.482	0.460

Generally speaking, large models and better architectures often improve DNN's performance. However, it is not clear whether domain adaptation will benefit from large models and better architectures. To investigate this, we used two DNN architectures, namely, the x-vector networks and DenseNets, with different number of parameters. The configuration of the two x-vector networks are shown in Table 3.1 and the configuration of the two DenseNets are shown in Table 3.2. The results

Table 6.5: Ablation study of the individual components in the proposed framework.

					SRE16		SRE18	
Utt. Adapt.	Frame Adapt	Consist.	Reg.	Aux. BN	EER(%)	DCF	EER(%)	DCF
×	×	×	×		12.02	0.990	11.59	0.720
✓	×	×	×		9.79	0.621	9.08	0.580
✓	✓	×	×		9.63	0.606	8.77	0.555
✓	✓	✓	×		9.03	0.585	8.33	0.520
✓	✓	✓	✓		8.69	0.556	7.95	0.500

are presented in Table 6.4. It is clear from Table 6.4 that increasing the model size improves the speaker embeddings' performance. We also observed, under the same amount of parameters, the DenseNets outperform the x-vector networks consistently. Besides, DNN domain adaptation does benefit from larger models. By widening the x-vector networks we achieved 3.68% and 4.02% improvement in SRE16 and SRE18, respectively. Using larger DenseNets, we achieved 6.01% and 8.23% improvement on SRE16 and SRE18, respectively. Clearly, DA benefits more from deep architectures than the swallow ones.

6.2.6 Influence of MMD Kernels

The most important component for MMD based DA is the kernel. The Gaussian kernel is theoretically more powerful than the quadratic kernel in that it can match up to infinite moments of two distributions. However, it is much harder to find appropriate bandwidth parameters for the Gaussian kernels, which is still an on-going research area [78]. A heuristic is to use the median pairwise distance computed from data [79]. Another way is to use multiple Gaussian kernels and hope that some of the kernels are close to the ideal ones. It is also possible to combine the median heuristic and the multi-kernel approach. Table 6.6 shows the effectiveness of different

Gaussian kernels for DA in SRE16 and SRE18.

For the median heuristic, we randomly sampled 10,000 frames of MFCC vectors from the unlabeled part of SRE16 and SRE18 and used the sampled data to compute the pairwise median distance. For the multi-kernel approach, we chose $\sigma_q = 1$ in Eq. 5.6 and used a multiplicative step-size of 0.1. Specifically, with 5 Gaussian kernels, the bandwidth parameters range from $10^{-2}\sigma_q$ to $10^{2}\sigma_q$, with a multiplicative step-size of 0.1. With 19 Gaussian kernels, the bandwidth parameters range from $10^{-9}\sigma_q$ to $10^{9}\sigma_q$, with a multiplicative step-size of 0.1. For combining the median heuristic with the multi-kernel approach, the bandwidth parameter σ_q was set to the median computed from data and a multiplicative step-size of 0.1 was used. We can see from Table 6.6 that the single kernel with arbitrarily chosen $\sigma_q = 1$ performs even worse than no adaptation, which shows that good kernel parameters are essential for MMD domain adaptation. On the other hand, the median heuristic performs much better than the single kernel with $\sigma_q = 1$. With the multi-kernel approach, the performance of the adaptation improves significantly, even though σ_q was arbitrarily chosen. The multiple kernel approach performs significantly better than the median heuristic. Finally, combining the multi-kernel approach with the median heuristic achieves the best results.

Table 6.6: The influence of the Gaussian kernel configuration on speaker embedding adaptation

Kernel Configuration	SRE16		SRE18	
	EER	minDCF	EER	minDCF
No Adaptation	12.02	0.990	11.59	0.720
Single Gaussian Kernel with $\sigma_q = 1$	14.93	0.841	13.33	0.776
Median Heuristic [79]	10.85	0.660	10.33	0.636
5 Gaussian Kernels	9.69	0.569	8.77	0.529
9 Gaussian Kernels	9.17	0.559	8.01	0.510
19 Gaussian Kernels	8.99	**0.547**	8.31	0.510
19 Gaussian Kernels with Median Heuristic	**8.69**	0.556	**7.95**	**0.50**

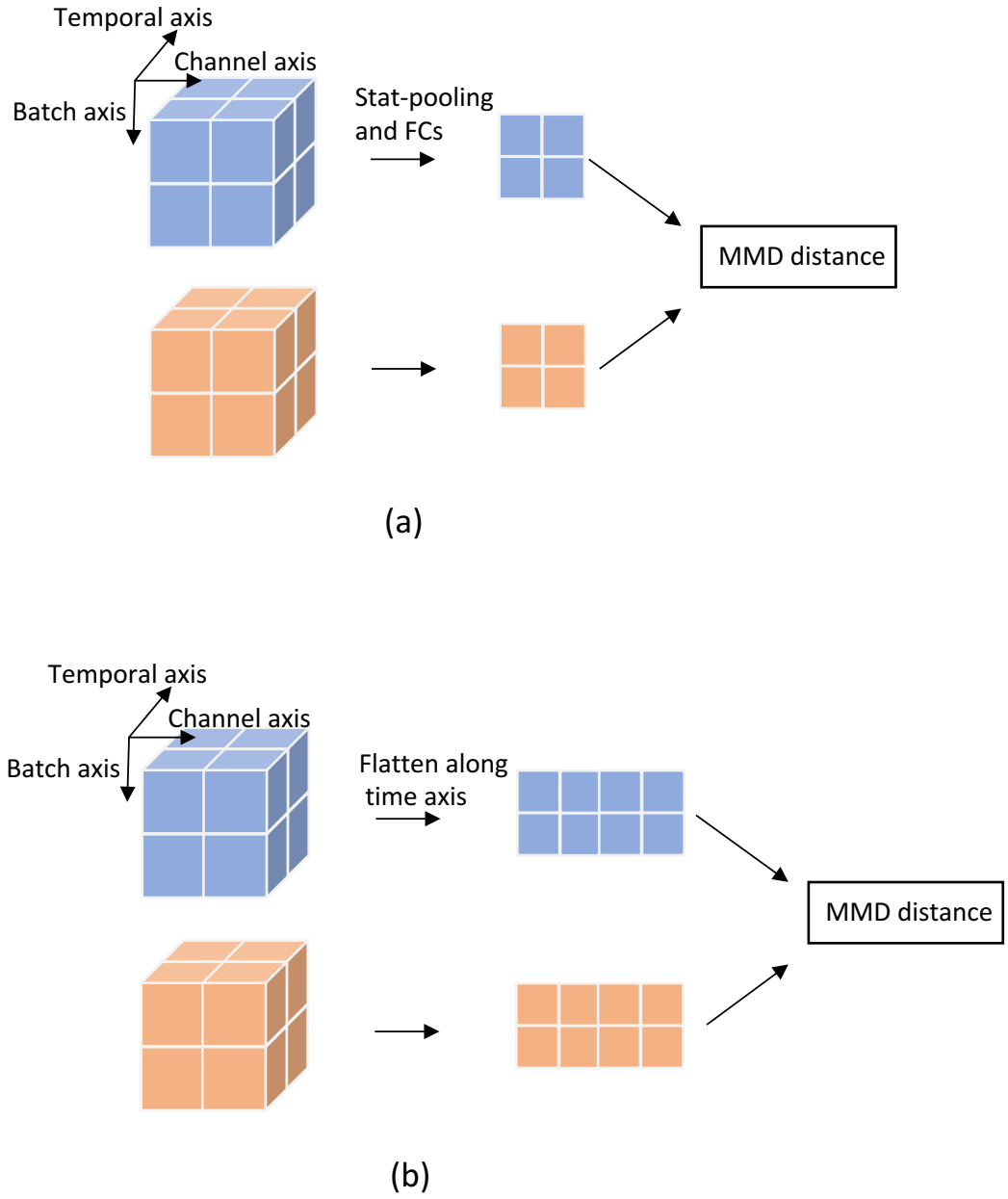

Figure 6.2: Diagrams demonstrate the difference between (a) utterance-level MMD distance $\mathcal{D}(\mathcal{H}_s^7, \mathcal{H}_t^7))$, and (b) frame-level MMD distance $\mathcal{D}(\text{FLAT}(\mathcal{H}_s^5), \text{FLAT}(\mathcal{H}_t^5)))$. Blue and orange cubes represent a batch of 3-dimensional data from two domains. In the case of utterance-level MMD distance, it is computed after the aggregation along the temporal axis in the statistics pooling layer. In the case of frame-level MMD distance, the temporal axis is flattened to transform 3D array into 2D array for computing MMD distance.

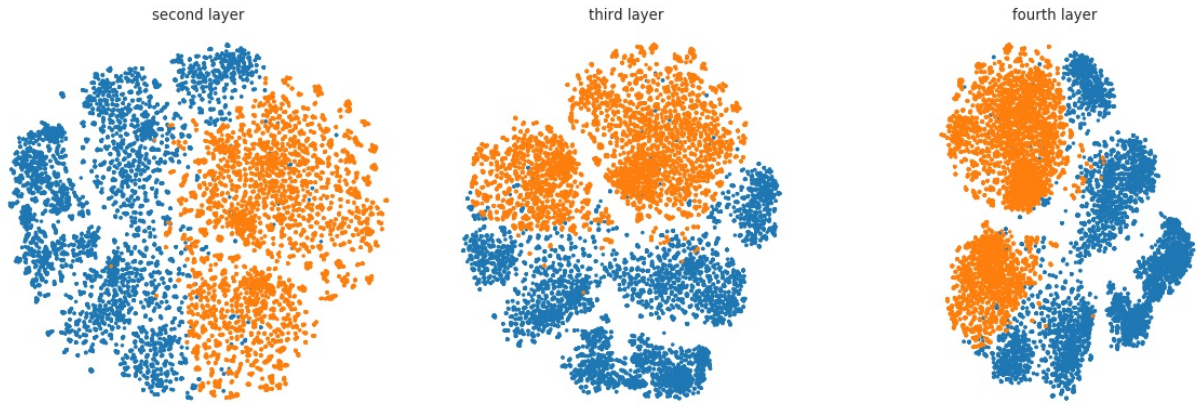

Figure 6.3: T-SNE plots of the hidden activations at the convolutional layers of the x-vector network in Table 3.1. The orange dots correspond to the data from the source-domain (SRE04–SRE10). The blue dots correspond to the data from the target-dominance (SRE18 evaluation set.).

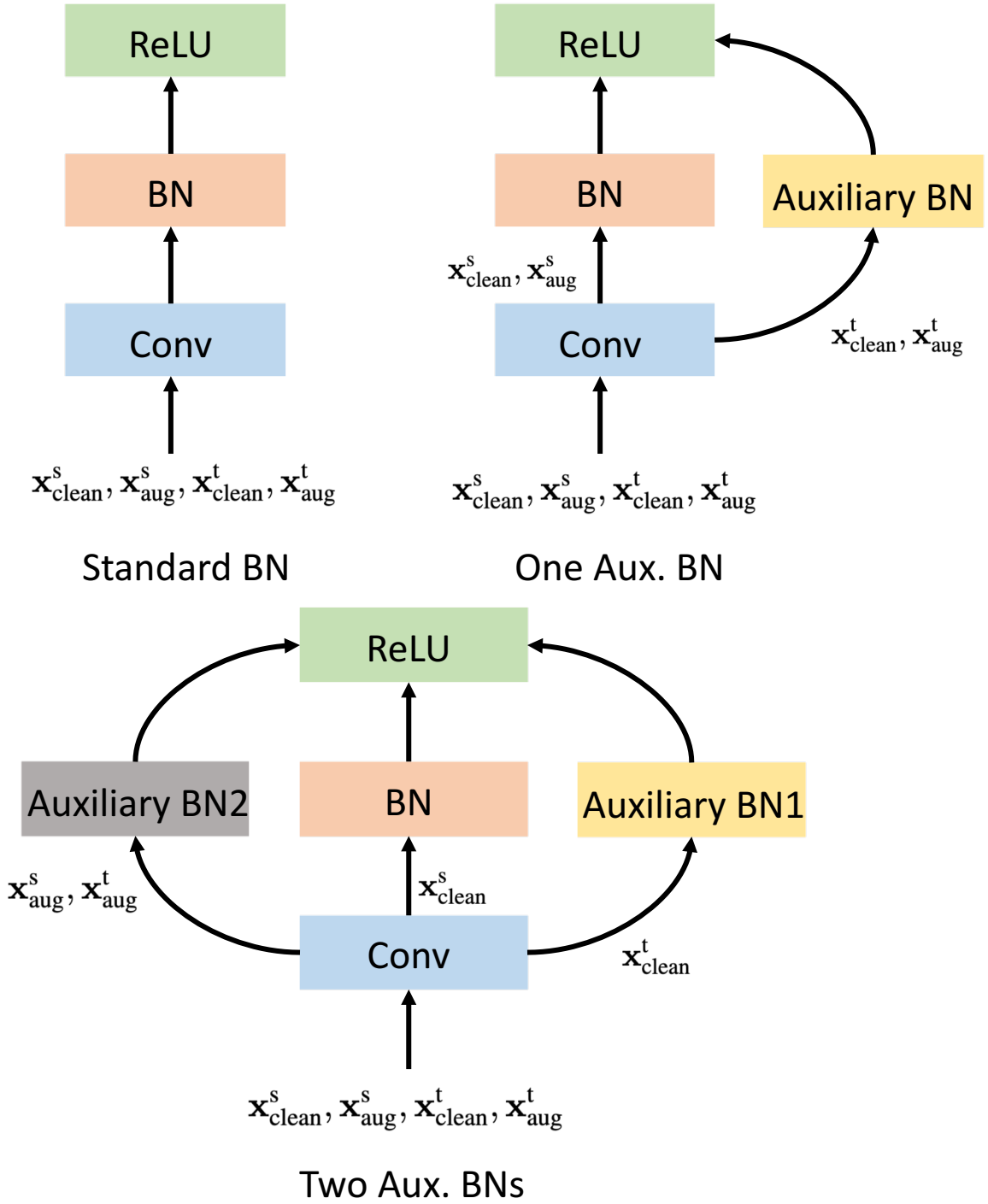

Figure 6.4: "One Aux. BN" means that the batch statistics of the source- and target-domains are computed separately. In the case of "Two Aux. BNs", we further divided the mini-batch into clean source-domain data, clean target-domain data, and augmented data when computing the mini-batch statistics. The superscripts "s" and "t" stand for source- and target-domains, respectively.

Chapter 7

LEARNING MIXTURE REPRESENTATION FOR DEEP SPEAKER EMBEDDING

How to effectively convert a sequence of variable length frame-level features to a fixed-dimension representation has always been a research focus in speaker recognition. In the x-vector network, the conversion is implemented by concatenating the mean and standard deviation of a sequence of frame-level features. However, mean and standard deviation are limited statistics for describing an acoustic sequence even with powerful feature extractors such as convolutional neural networks. In this chapter, we propose a novel statistics pooling method that can produce more descriptive statistics through a mixture representation. Applying the proposed mixture representation pooling (MRP) to the VOiCES19 dataset, we found that the MRP improves the robustness of speaker embeddings against distracting noise and room reverberation.

7.1 Statistics Pooling in Deep Speaker Embedding Systems

The process that converts an acoustic sequence to a fixed-dimensional representation is referred to as statistics pooling in the literature [5]. Given a sequence of frame-level features $\{\mathbf{h}_t\}_{t=1}^{T}$, a statistics pooling layer computes the mean $\boldsymbol{\mu}$ and standard

deviation $\boldsymbol{\sigma}$ over $\{\mathbf{h}_t\}_{t=1}^{T}$:

$$\boldsymbol{\mu} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{h}_t \tag{7.1}$$

$$\boldsymbol{\sigma} = \sqrt{\frac{1}{T}\text{Diag}\left(\sum_{t=1}^{T} \mathbf{h}_t\mathbf{h}_t^{\mathsf{T}} - \boldsymbol{\mu}\boldsymbol{\mu}^{\mathsf{T}}\right)}, \tag{7.2}$$

where Diag($\mathbf{A}$) means the diagonal of $\mathbf{A}$ and the square root is applied element-wise. The utterance-level representation $\mathbf{x}$ is obtained by concatenating the mean and standard deviation:

$$\mathbf{x} = \text{CONCAT}(\boldsymbol{\mu}, \boldsymbol{\sigma}) \tag{7.3}$$

In [6, 80], the authors argued that each frame should not be treated equally when computing the statistics for an input sequence. They proposed to use an attention mechanism to calculating the score($\mathbf{h}_t, \mathbf{v}$) for each frame, where $\mathbf{v}$ is an attention vector and score(., .) is a score function. A score function parameterized by a single-layer neural network was used:

$$\text{score}(\mathbf{h}_t, \mathbf{v}) = \mathbf{v}^{\mathsf{T}} f(\mathbf{W}\mathbf{h}_t + \mathbf{b}), \tag{7.4}$$

where $f(.)$ is a non-linear activation function. The score is normalized across frames using a softmax function:

$$\alpha_t = \frac{\exp\left(\text{score}(\mathbf{h}_t, \mathbf{v})\right)}{\sum_{t=1}^{T}\exp\left(\text{score}(\mathbf{h}_t, \mathbf{v})\right)}. \tag{7.5}$$

The normalized scores are used to weight each frame:

$$\boldsymbol{\mu} = \sum_{t=1}^{T} \alpha_t \boldsymbol{h}_t \tag{7.6}$$

$$\boldsymbol{\sigma} = \sqrt{\text{Diag}\left(\sum_{t=1}^{T} \alpha_t \mathbf{h}_t \mathbf{h}_t^{\mathsf{T}} - \boldsymbol{\mu}\boldsymbol{\mu}^{\mathsf{T}}\right)}. \tag{7.7}$$

Some researchers conjecture that vowels may have higher α_t because vowels are generally more speaker discriminative [81]. Other researchers suggest that α_t merely differentiates speech and non-speech frames [6]. We refer to this statistics pooling as attentive statistics pooling (ASP) in the rest of the chapter. The attentive statistics pooling is further extended to multi-head attention in [6], where multiple attention heads are used to weight each frame to produce multiple means and standard deviations for an input sequence.

Suppose each attention head k is parameterized by $\mathbf{v}_k$, where $k = 1, \ldots, K$. Then the attention scores for each head can be written as:

$$\text{score}(\mathbf{h}_t, \mathbf{v}_k) = \mathbf{v}_k^{\mathsf{T}} f(\mathbf{W}\mathbf{h}_t + \mathbf{b}), \tag{7.8}$$

where $\mathbf{v}_k$ is the attention vector for head k. The normalized scores are obtained by:

$$\alpha_{t,k} = \frac{\exp\left(\text{score}(\mathbf{h}_t, \mathbf{v}_k)\right)}{\sum_{t=1}^{T} \exp\left(\text{score}(\mathbf{h}_t, \mathbf{v}_k)\right)}. \tag{7.9}$$

The normalized scores are used to weight the frame-level features:

$$\boldsymbol{\mu}_k = \sum_{t=1}^{T} \alpha_{t,k} \boldsymbol{h}_t \tag{7.10}$$

$$\boldsymbol{\sigma}_k = \sqrt{\text{Diag}\left(\sum_{t=1}^{T} \alpha_{t,k} \mathbf{h}_t \mathbf{h}_t^{\mathsf{T}} - \boldsymbol{\mu}\boldsymbol{\mu}^{\mathsf{T}}\right)}. \tag{7.11}$$

The utterance-level representation $\mathbf{x}$ is obtained by concatenating the means and standard deviations from all attention heads:

$$\mathbf{x} = \text{CONCAT}\left(\boldsymbol{\mu}_1, \boldsymbol{\sigma}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\sigma}_K\right). \tag{7.12}$$

A regularization term was also introduced to prevent the attention heads from learning the same information [6]. In [82], the idea is further extended to the frequency axis. Instead of weighting frames alone, the authors proposed to weight both the frames and the frequency bins simultaneously.

Although attentive statistics pooling shows promising results, we believe that weighting frame-level features is not enough to produce ideal speaker representation. We believe that the statistics pooling layer needs to produce a good statistical summary of the frame-level features. However, means and standard deviations are limited statistics for summarizing a distribution. Because mixture models are more powerful models to capture the underlying distribution, we propose to incorporate mixture representation into the statistics pooling layer. Although the attention mechanism in [6] can produce multiple means and standard deviations for an input sequence, it does not produce a valid mixture representation. A valid mixture model requires the probability mass sums to 1 across all mixture assignments for the same input. There is no mechanism in [6] to enforce this requirement. Therefore, the multiple attention heads merely increase the number of parameters in the network without producing a richer representation from a statistical point of view. Instead, our method explicitly uses an attention mechanism to produce valid mixture assignments, from which the means and standard deviations are computed just like the M-step in Gaussian mixture models.

Work similar to ours was proposed in [83], where a dictionary of centers are learned from data and the center assignments are computed based on the Euclidean distances between frame-level features and the centers. There are two key differences between

our method and that of [83]. First, unlike [83], our method does not explicitly use the *centers* to compute the assignments; instead, we use an attention mechanism to produce mixture assignments. Because the score function in the attention mechanism is more general than Euclidean distances, our method can have more desirable mixture assignments. Second, the means and standard deviations are computed based on the sufficient statistics as in the EM for GMMs, which makes our multi-head attention to have a probabilistic interpretation.

7.2 Statistics Pooling with Mixture Representation

Gaussian mixture models (GMMs), which are exemplified by the GMM-HMM in ASR and the i-vectors in speak recognition [8], are very popular in the speech community to model complex distributions. A GMM is typically trained by the EM algorithm that alternates between the E-step and the M-step [12]. Assume that the mixture assignments of frame $\mathbf{h}_t$ are $\alpha_{t,k}$, where $k = 1, \ldots, K$; then the parameters of each mixture component can be computed by:

$$N_k = \sum_{t=1}^{T} \alpha_{t,k} \tag{7.13}$$

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{t=1}^{T} \alpha_{t,k} \mathbf{h}_t \tag{7.14}$$

$$\boldsymbol{\sigma}_k = \sqrt{\text{Diag}\left(\frac{1}{N_k} \sum_{t=1}^{T} \alpha_{t,k} \left(\mathbf{h}_t - \boldsymbol{\mu}_k\right) \left(\mathbf{h}_t - \boldsymbol{\mu}_k\right)^{\mathsf{T}} \right)}. \tag{7.15}$$

The attention scores are still produced by the score function in Eq. 7.8. However, the scores are normalized across the attention heads:

$$\alpha_{t,k} = \frac{\exp\left(\text{score}(\mathbf{h}_t, \mathbf{v}_k)\right)}{\sum_{k=1}^{K} \exp\left(\text{score}(\mathbf{h}_t, \mathbf{v}_k)\right)}. \tag{7.16}$$

The normalized scores are used to compute the means and standard deviations as in Eqs. 7.13–7.15. The interpretation of Eq. 7.16 is totally different from that of Eq. 7.9. The $\alpha_{t,k}$ in Eq. 7.16 is similar to the mixture assignments in Gaussian mixture models, while $\alpha_{t,k}$ in Eq. 7.9 acts as a frames selector without probabilistic interpretation. We refer to this statistics pooling method as mixture representation pooling (MRP) in the rest of the chapter.

Table 7.1: Systems performance on VoxCeleb1, VOiCES19-dev, and VOiCES19-eval. For VoxCeleb1, we used cosine scoring. For VOiCES19-dev and VOiCES19-eval, we used a PLDA backend.

		VoxCeleb1	
Model	Pooling Method	EER(%)	minDCF
X-vector network	Mean & STD	2.14	0.197
Wide x-vector network	Mean & STD	2.03	0.219
Densenet121	Mean & STD	1.37	0.156
Densenet121	ASP	1.22	0.150
Densenet121	MRP	**1.10**	**0.131**

		VOiCES19-dev		VOiCES19-eval	
Model	Pooling Method	EER(%)	minDCF	EER(%)	minDCF
X-vector network	Mean & STD	2.66	0.300	6.98	0.520
Wide x-vector network	Mean & STD	2.65	0.294	6.62	0.503
Densenet121	Mean & STD	**1.53**	0.222	5.53	0.415
Densenet121	ASP	1.84	0.197	5.20	0.402
Densenet121	MRP	1.65	**0.184**	**4.77**	**0.390**

Table 7.2: Performance of attentive statistics pooling (ASP) and our proposed method (MRP) with different numbers of attention heads using Densenet121 on VOiCES19-eval.

	EER(%)		minDCF	
# of Heads	ASP	MRP	ASP	MRP
1	**5.20**	5.53	**0.402**	0.415
2	5.50	4.86	0.428	0.397
3	5.69	**4.77**	0.442	**0.390**
4	5.77	5.20	0.453	0.404
5	6.00	5.73	0.430	0.413

7.3 Experiments

7.3.1 Data Preparation

The training data includes the VoxCeleb1 development set and the VoxCeleb2 development set [84, 85]. We followed the data augmentation strategy in the Kaldi SRE16 receipt. The training data were augmented by adding noise, music, reverb, and babble to the original speech files in the datasets. After filtering out the utterances shorter than 400-ms and the speakers with less than 8 utterances, we are left with 7,302 speakers. We used the filter bank feature implemented in Kaldi. Mean normalization was applied to the filter-bank features using a 3-second sliding window. Non-speech frames were removed using Kaldi's energy-based voice activity detector.

7.3.2 DNN Architecture and Training

We experimented with three different networks, namely, the standard x-vector network as in [5], the wide x-vector network, and Densenet121 as mentioned in Section 3.3. The wide x-vector network has the same structure as the x-vector network except that the channel size in each convolutional layer is doubled.

The networks were trained using additive margin softmax with a margin of 0.35. The networks were optimized using stochastic gradient descent (SGD). For each mini-batch, we randomly selected 64 utterances from the training set and then randomly cropped 400-ms speech segments from these utterances. We define one epoch as looping through 120,000 such segments. We trained the networks for 320 epochs. The learning rate was set to 0.005 and was divided by 10 at epoch 80, epoch 120, and epoch 160. All networks were implemented in PyTorch [67].

7.3.3 Backend Training

We used a standard backend comprised of linear discriminant analysis (LDA), length-normalization, and probabilistic linear discriminant analysis (PLDA). We concatenated speech from the same video session in VoxCeleb1 and VoxCeleb2. These concatenated speech segments were used to train LDA and PLDA models. We used adaptive score normalization [65] in all systems.

7.3.4 Evaluation

We evaluated the performance of various statistics pooling methods on the VoxCeleb1 test set, VOiCES19 development set, and VOiCES19 evaluation set [86]. We report results in terms of equal error rate (EER) and minimum cost function (DCF) with $P_{target} = 0.01$.

7.4 Results

7.4.1 Performance Comparison

We compared the proposed mixture representation pooling (MRP) with the attentive statistics pooling (ASP) and the vanilla single mean and standard deviation pooling. To make a fair comparison with vanilla single mean and standard deviation pooling, we fixed the dimension of the concatenated means and standard deviations in Eq. 7.12. In

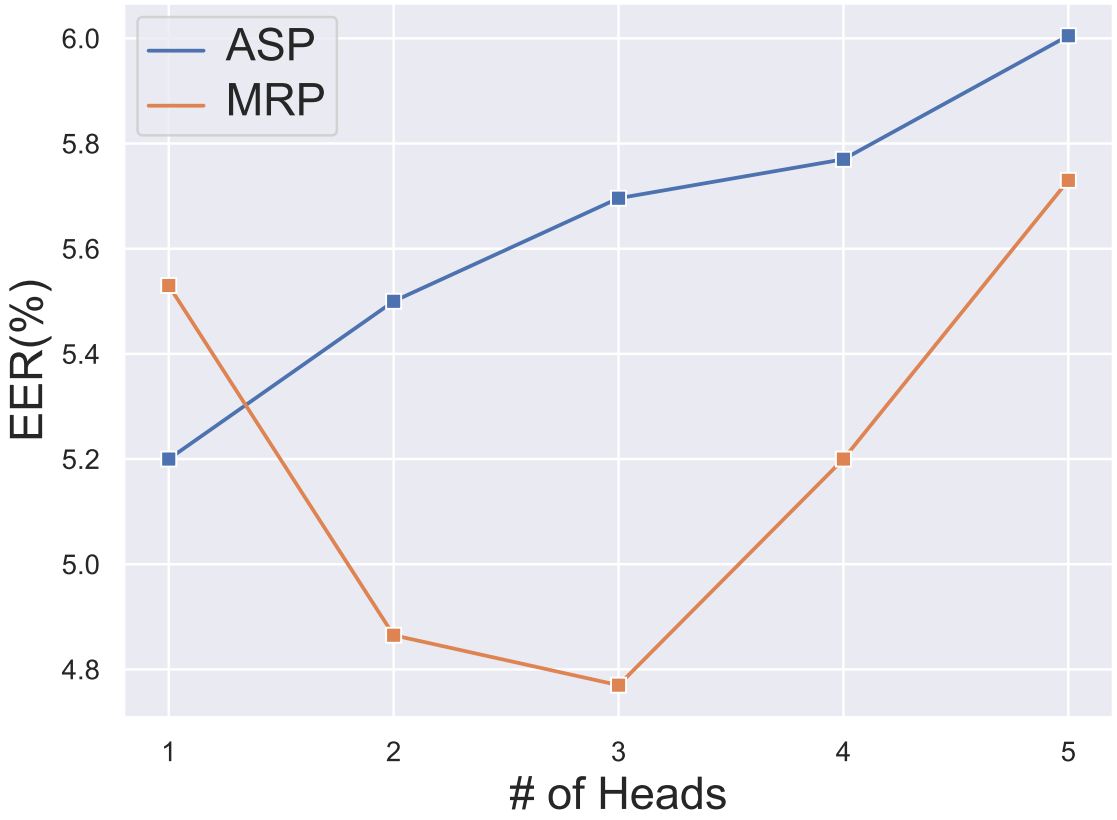

Figure 7.1: Line plots showing the EER of attentive statistics pooling (ASP) and the proposed mixture representation pooling (MRP) with different numbers of attention heads on VOiCES19-eval.

other words, increasing the number of attention heads would reduce the dimensionality of the individual mean and standard deviation vectors.

The first three rows of Table 7.1 show the results of the x-vector network, the wide x-vector network, and Densenet121. Our Densenet121 outperforms the x-vector network and the wide x-vector network significantly. For VoxCeleb1, Densenet121 almost reduces the EER by half when compared with the x-vector network. What's more, our Densenet121 requires even lower flops than the x-vector network, as shown in Table 3.3. Although the wide x-vector network has more parameters than Densenet121 and the x-vector network, the former only has very small performance gain over the

x-vector network, which suggests that increasing the depth is more beneficial than increasing the number of channels.

The last two rows of Table 7.1 show the results of our Densenet121 with attentive statistics pooling and the proposed method, respectively. The numbers of heads were set to 1 and 3, respectively, as these hyper-parameters lead to the best performance in the two systems. The proposed method outperforms the attentive statistics pooling in all of the experiments.

7.4.2 Effect of the Number of Heads

We investigated the effect of the number of attention heads on speaker verification performance. We fixed the dimension of the concatenated means and standard deviations in Eq. 7.12 to prevent the model from gaining the benefit of having more parameters. The results are shown in Table 7.2. MRP with the number of heads equal to 1 is just the vanilla mean and standard deviation pooling. As can be observed from Table 7.2, the number of heads influences the performance quite significantly. Actually, attentive statistics pooling with 5 heads performs even worse than the model without any attention. The performance of the proposed method degrades when the number of heads increases beyond 3, but it is still better than the vanilla single mean and standard deviation pooling.

To gain some insights into how the attention mechanism assigns mixture components, we plotted the mixture assignments for a speech segment from the VoxCeleb1 test set in Fig. 7.2. For ease of visualization, we assume that each frame is assigned to the attention head (mixture) with the highest probability in Eq. 7.16. The first row and the second row of Fig. 7.2 show the MRP with three heads and five heads, respectively. There are more assignment transitions in the five-head model. Compared with the model with three heads, the five heads model has more assignment transitions occurred in short periods. This is undesirable, as the adjacent frames are typically similar to each other and should be assigned to the same mixture, the assignment

transitions should not occur frequently.

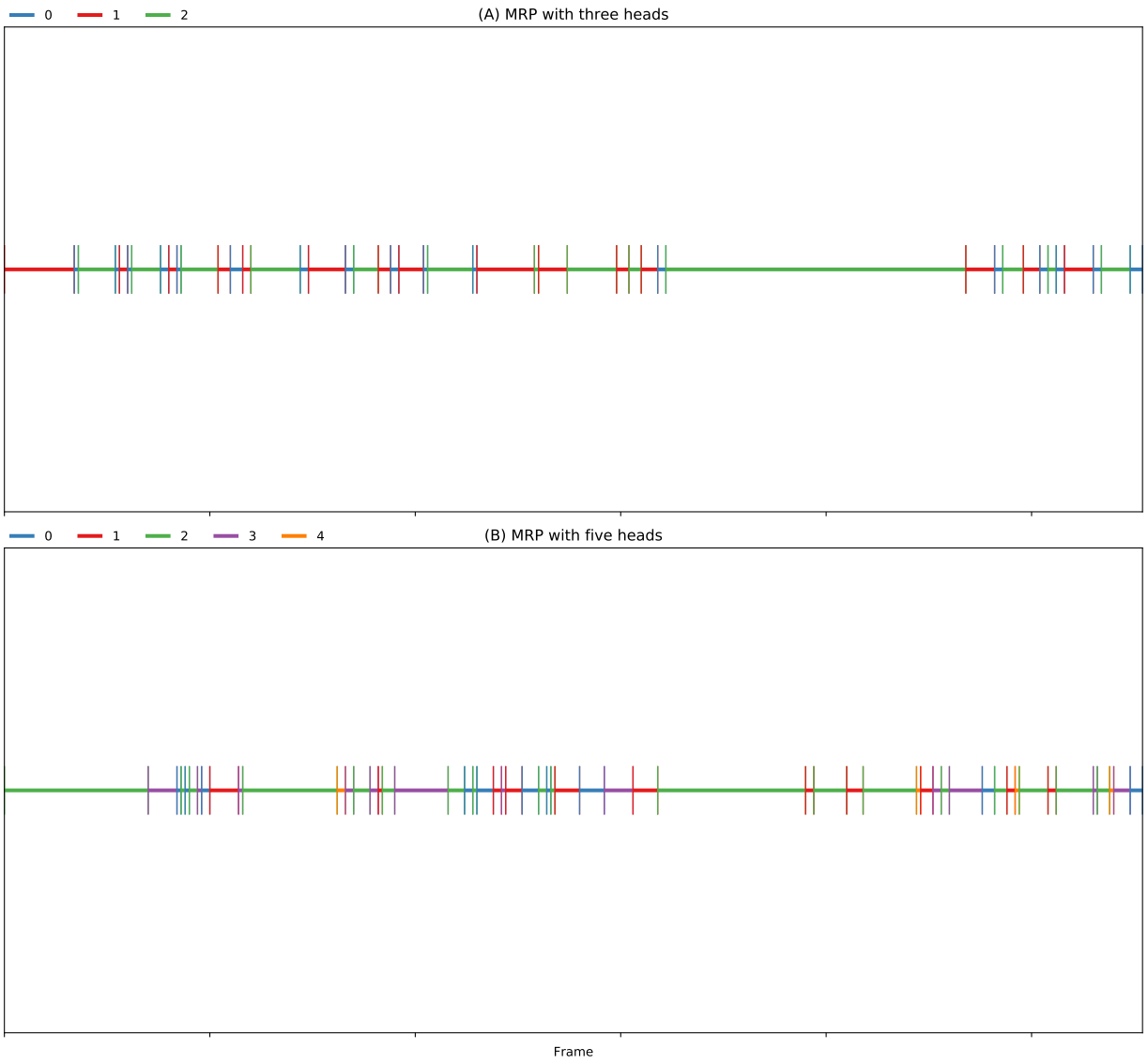

Figure 7.2: The mixture assignments across frames on a speech segment chosen from the VoxCeleb test set. Different colors represent different mixture components in the mixture representation pooling (MRP). For ease of interpretation, the assignments are converted to one-hot vectors by setting the maximum assignment probability in Eq. 7.16 to 1 and the rest to 0.

Chapter 8

TOWARDS END-TO-END SPEAKER VERIFICATION

8.1 The Role of Backend PLDA Model

It is worth noting that the x-vector system is not an end-to-end system. After the network is trained, the embeddings are extracted from an utterance-level layer using full-length utterances as inputs. Because embeddings are not good enough for verification, a backend model is trained on these embeddings and used for verification. The reason for the extra backend model is that the x-vector network is trained on short speech segments only, which results in poor performance on long speech segments. A backend model trained on speaker embeddings extracted from long speech segments can mitigate this problem. In other words, the backend model makes the SV system robust to the duration mismatch. In this section, we propose an end-to-end SV system without a backend model, which is achieved by using the three strategies that are designed to make speaker embeddings robust to duration mismatch.

8.2 Two-Stage Approach in State-of-the-art SV Systems

An end-to-end system using only an integrated neural network is attractive in several aspects. Firstly, hyperparameter optimization is easier in an end-to-end system. In an x-vector system, the DNN and the backend are optimized separately, which complicates the hyperparameter search as validation has to be done for the network and backend separately. Secondly, although training the backend itself is fairly fast when compared with training the network, the extraction process can be time-consuming. Besides, it is not clear which part of the dataset should be used for backend training.

Thirdly, an end-to-end system is easier to deploy and debug.

In this chapter, we propose three modifications to improve DNN embeddings' discriminative ability without relying on a backend. Firstly, instead of randomly sampling a single segment out of an utterance, multiple segments are sampled from an utterance and spliced together to form a long training sample. Secondly, we introduce additional convolutional layers as a learnable pooling layer to save computation cost by reducing temporal resolution, which is especially desirable for working with long speech segments. Thirdly, we introduce a mask-pooling layer as a special form of data augmentation inside the network. The mask-pooling layer produces multiple utterance-level representations out of a single speech segment by randomly masking out frame-level activations and then computing the temporal statistics of the remaining activations across the temporal axis.

8.3 Proposed End-to-end Approach

In x-vector systems, the segments for training the network are densely sampled from the original utterances. The sampled segments typically range from 200 ms to 400 ms, which are much shorter than the original utterances. As a result, the embeddings may not be good representations of long utterances. A simple solution is to use longer segments or directly use the original utterances for training. This strategy, however, also has problems. Because the training segments are sampled from the original utterances, long segments have less sample diversity than short segments. Training with long segments alone may lead to overfitting. Another disadvantage is that long segments require more computation and GPU memory. Ideally, we want to have both short and long segments for training. This approach, however, will lead to a substantial computational burden. We propose three techniques that reduce the duration discrepancy between the training and test data while maintaining the sample diversity and computation cost at a reasonable level.

8.3.1 Splice Sampling

When training an x-vector network, one segment is sampled from one utterance. A drawback of this sampling approach is that each training segment only comprises a number of consecutive frames. A good speaker embedding should be able to exploit speaker information across time. Therefore, we propose to sample several chunks of non-consecutive segments from each training utterance and splice them together to form a single training segment as shown in Fig. 8.1. This approach can diversify training data when using long segments for training.

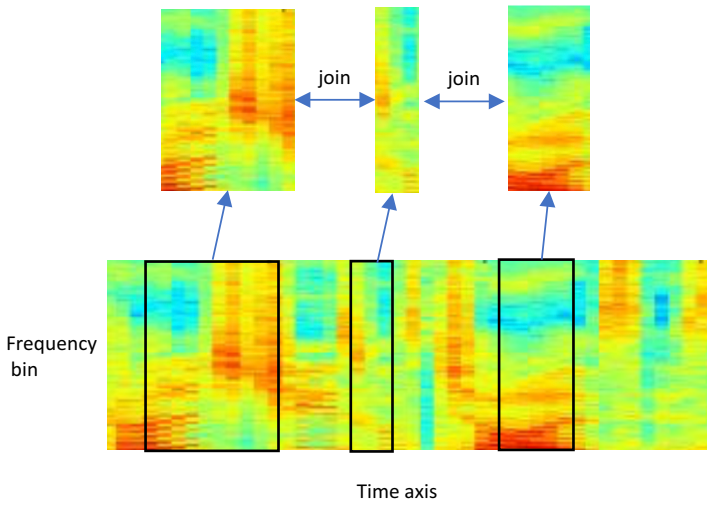

Figure 8.1: Splice sampling on a spectrogram. Three chunks are taken out of the spectrogram and spliced together to form a training segment.

8.3.2 CNN Local Pooling

Subsampling operation, such as max-pooling or mean pooling, can significantly reduce the computation cost of a CNN by decreasing the resolution of the input. In addition, max-pooling or mean pooling introduces invariance to translation. Although translation invariance is not very important to speaker embedding as the statistics pooling layer computes the mean and variance across the time axis, the computational reduc-

tion is still very attractive. In this chapter, we consider the pooling operation over the time axis.

Denote $\mathbf{I}$ as the feature map to be input to a pooling layer. The output $\mathbf{O}$ of max pooling takes the maximum value inside the kernel and move the kernel with stride S:

$$\mathbf{O}[i,t] = \max_{k=0,\ldots,K-1} \mathbf{I}[i, S \times t + k], \tag{8.1}$$

where k indexes the elements inside the kernel, K is the length of the kernel, i is the channel index, and t is the time index of the output. The mean pooling can be described in a similar manner:

$$\mathbf{O}[i,t] = \frac{1}{K} \sum_{k=0,\ldots,K-1} \mathbf{I}[i, S \times t + k]. \tag{8.2}$$

It was demonstrated in [87] that the pooling operations in Eq. 8.1 and Eq. 8.2 can be replaced by a convolutional layer with increased stride without loss in accuracy. Given the input feature map $\mathbf{I}$ and the convolutional filter matrix $\mathbf{W}$, the output of the convolutional layer at channel j and frame t is

$$\mathbf{O}[j,t] = \sum_{k=0,\ldots,K-1} \sum_{i} \mathbf{W}[j,i] \cdot \mathbf{I}[i, S \times t + k], \tag{8.3}$$

where j indexes the output channel and i indexes the input channel. Here we omit the bias term for simplicity. We can see the similarity between the convolutional layer and the pooling layer from the above equations. In fact, if we use a kernel parameterized by $1/K$ with kernel size K and stride S, the convolutional layer would act like mean pooling [87]. We adopt two 1D convolutional layers with kernel size 2 and stride 2 in our DNN as a way to reduce subsequent computation. The architecture of our network is summarized in Table 3.1. Because the two convolutional layers with stride 2 reduce the temporal resolution by half for the following convolutional layers,

for an input sequence of size 23×3000, our network requires only 4.3 GMac (Giga multiply–accumulate operations) to produce an embedding as compared to 8.0 GMac in the original x-vector network.

8.3.3 Mask-pooling Layer

Besides high computational cost, another disadvantage of using long training segments is the lack of sample diversity. We propose a novel mask-pooling layer that augments training data without significantly increasing computational cost. Assume that the activation at the last convolutional layer (after flattening) is $\mathbf{z}_t$, where t is a time index. The mask-pooling layer involves the following operations. First a mask r_t is sampled from a Bernoulli distribution parameterized by p. Then we multiply $\mathbf{z}_t$ with r_t to decide whether we keep this frame or not. The resulting frames are denoted as $\{\hat{\mathbf{z}}_t\}$. Finally, the utterance-level representation $\mathbf{x}$ is obtained by concatenating the mean and the standard deviation of $\{\hat{\mathbf{z}}_t\}$:

$$r_t \sim \text{Bernoulli}(p), \tag{8.4}$$

$$\hat{\mathbf{z}}_t = r_t \cdot \mathbf{z}_t, \tag{8.5}$$

$$\mathbf{x} = \text{Concat}(\text{MEAN}(\{\hat{\mathbf{z}}_t\}), \text{STD}(\{\hat{\mathbf{z}}_t\}). \tag{8.6}$$

Here MEAN$(\cdot)$ and STD$(\cdot)$ are operated on non-zero elements in $\{\hat{\mathbf{z}}_t\}$. We denote Eq. 8.4–8.6 in whole as:

$$\mathbf{x} = \text{MaskPooling}(\{\mathbf{z}_t\}, p). \tag{8.7}$$

The above operations are very similar to Dropout [88]. However, unlike Dropout, we can applied the operations I times with a different p_i sampled from a uniform distribution over 0 to 1:

$$\mathbf{x}_i = \text{MaskPooling}(\{\mathbf{z}_t\}, p_i), \quad i = 1, \ldots, I. \tag{8.8}$$

Suppose $f(\cdot)$ represents the operations of the fully-connected layers and the softmax layer. Assume that the loss function is $\mathcal{L}(\cdot)$. Then the loss for these I copies of the augmented data is:

$$\sum_{i=1}^{I} \mathcal{L}(f(\mathbf{x}_i), y), \tag{8.9}$$

where y is the label of all $\mathbf{x}_i$. Different from the data augmentation methods in [5], where noise and reverberation were added to the waveforms, our mask-pooling layer operates on the internal representation of the network. In terms of the augmentation effect, mask-pooling produces utterance-level representations with different durations. It is similar to the cutout and spectrum augmentation [89, 90] in that the augmented sampled are produced by withholding information. An important advantage of the proposed method over cutout and spectrum augmentation is that for I augmented samples, there is only one forward propagation through the convolutional layers and I forward propagations through the fully-connected layers.

8.4 Experiments

8.4.1 Data Preparation

The training data include NIST SRE 2004–2010 (SRE04–10 in short) and all of Switchboard data. We followed the data augmentation strategy in the Kaldi SRE16 receipt [5, 91]. The training data were augmented by adding noise, music, reverb, and babble to the original speech files in the datasets. After filtering out utterances shorter than 500 ms and speakers with less than 8 utterances, we are left with 4,808 speakers. 23-dimensional Mel-frequency cepstral coefficients (MFCC) were computed from 8kHz speech files. Mean normalization was applied to the MFCC using a 3-second sliding window. Non-speech frames were removed using Kaldi's energy-based voice activity detector.

8.4.2 Training of DNNs and PLDA

For fair comparisons, all systems under evaluation were trained to minimize the additive margin loss using an Adam optimizer [74] with a learning rate set to 0.001. In the x-vector systems, embeddings were extracted from the affine transform layer after statistics pooling. For the proposed system, the embeddings were extracted from the last fully-connected layer before computing log-softmax. We used a standard backend comprised of LDA, length-normalization, and PLDA. Both LDA and PLDA were trained using embeddings extracted from full-length utterances. We used correlation alignment [47] for domain adaptation before presenting the embeddings to the PLDA backend. For the end-to-end systems, we applied a whitening transformation to the embeddings of enrollment data and the test data before cosine-distance scoring. The whitening matrix was estimated using target-domain data.

8.4.3 Evaluation

All systems were evaluated on the evaluation sets of SRE 2016 and 2018. The SRE16 evaluation set is composed of Tagalog and Cantonese telephone conversations. For SRE18, we only conducted evaluations on the CMN2 portion, which consists of Tunisian Arabic conversations. Both evaluations aim to evaluate the robustness of systems against noise, channel, and language mismatches. We report results in equal error rate (EER) and minimum cost function (DCF). Both metrics are obtained using the scoring tools provided by NIST.

8.5 Results

We present the performance of the proposed method and x-vector systems on SRE16 and SRE18. We conducted experiments for three different DNNs, namely Xvec_short, Xvec_long, and Our_Xvec. Xvec_short refers to the x-vector network trained on 200ms-400ms chunks. Xvec_long refers to the x-vector network trained on 1200ms

chunks. Our_Xvec refers to the x-vector network with the proposed three modifications. We also compared systems using the PLDA backend with systems directly using cosine similarity.

As can be seen from Table 8.5, when directly using cosine similarity for scoring, Xvec_long outperforms Xvec_short. However, with the PLDA backend, it is the other way around. Still, the best performance for x-vector systems is obtained by using short training segments with a PLDA backend. The proposed approach outperforms the best x-vector system with simply cosine-distance scoring. Another interesting finding is that the proposed approach does not benefit from the PLDA backend as the x-vector system does, which suggests that a backend is no longer necessary.

Table 8.1: Performances of x-vector systems and the proposed approach with different scoring methods.

		SRE16		SRE18	
Front-end	Scoring	EER	DCF	EER	DCF
Xvec_short	PLDA	8.34	0.593	8.73	0.556
Xvec_long	PLDA	8.96	0.593	8.83	0.570
Our_Xvec	PLDA	8.90	0.600	8.91	0.598
Xvec_short	Cosine	10.57	0.674	11.98	0.681
Xvec_long	Cosine	9.88	0.653	11.12	0.643
Our_Xvec	Cosine	**8.26**	**0.583**	**8.65**	**0.551**

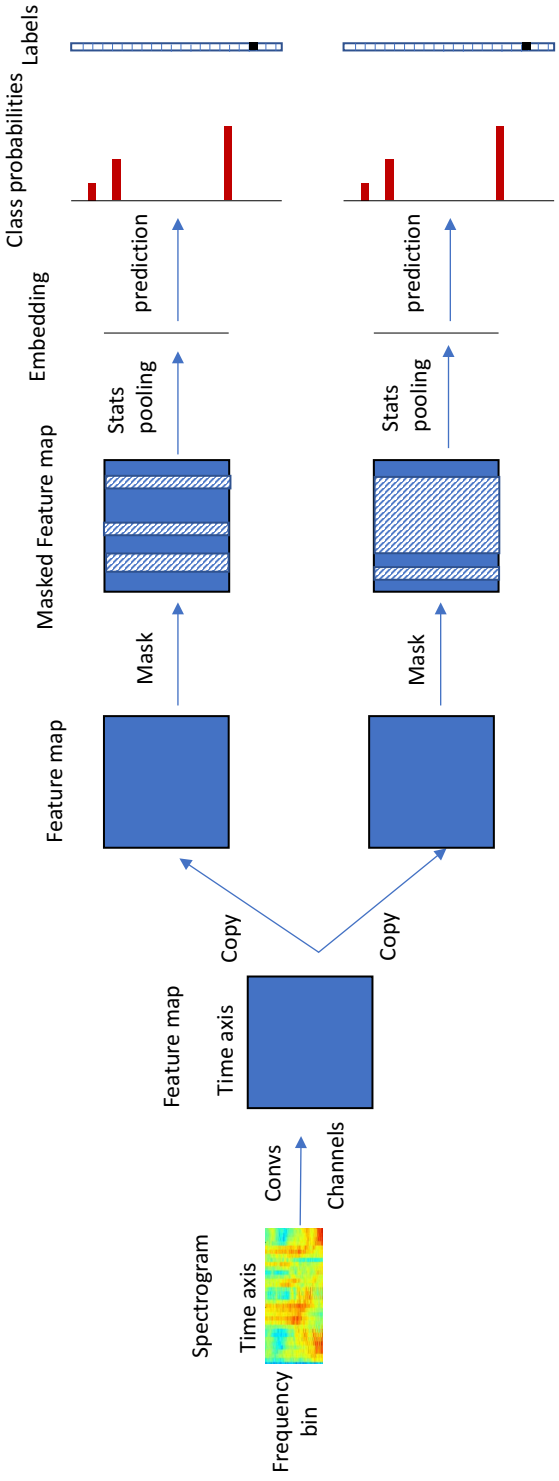

Figure 8.2: The x-vector network with the proposed mask-pooling layer operates on a spectrogram.

Chapter 9

CONCLUSIONS AND FUTURE WORKS

This thesis proposed several solutions for domain adaptation, end-to-end speaker verification, and efficient statistics pooling. For domain adaptation, Chapter 6 presents a framework for adapting DNN speaker embeddings across languages. We studied all three individual components of our framework (multi-level adaptation, separate batch normalization, and consistency regularization) in detail and found that combining them achieves the best result. This thesis also studied the effect of the kernel parameters in MMD and found that the multi-kernel approach, together with the median heuristic, gives the best result. As the proposed framework does not make specific assumptions about the characteristics of domain mismatch, it should be applicable to domain mismatch beyond the language mismatch studied in this thesis. It would be interesting to investigate other factors such as noise- and channel-induced domain differences in the future.

For end-to-end speaker verification, Chapter 8 shows that with three modifications to the x-vector system, it is possible to train a state-of-the-art speaker embedding system without a backend. The proposed methods not only eliminate the need for the complicated backend but also reduce the x-vector extraction time to about half. However, it is still not very clear how the duration mismatch of training and test speech will affect the speaker embeddings. In the future, we will investigate this problem in depth.

For efficient statistics pooling, Chapter 7 presents a new way of performing statistics pooling for deep speaker embeddings. The method is inspired by Gaussian mixture models and attention mechanisms. Chapter 7 introduces the concept of mixture

representations into statistics pooling by using multi-head attention in a novel way. The experimental results show that the mixture representation pooling is more effective than the previously proposed attentive statistics pooling. More importantly, the proposed method is less prone to overfitting when the number of heads increases. In future work, we will explore how to regularize the attention mechanism when the number of attention heads increases further.

BIBLIOGRAPHY

[1] J. Rohdin, T. Stafylakis, A. Silnova, H. Zeinali, L. Burget, and O. Plchot, "Speaker verification using end-to-end adversarial language adaptation," in *Proc. ICASSP*, pp. 6006–6010, 2019.

[2] O. Plchot, P. Matějka, A. Silnova, O. Novotný, M. D. Sánchez, J. Rohdin, O. Glembek, N. Brümmer, A. Swart, J. Jorrín-Prieto, P. García, L. Buera, P. Kenny, J. Alam, and G. Bhattacharya, "Analysis and description of ABC submission to NIST SRE 2016," in *Proc. Interspeech*, pp. 1348–1352, 2017.

[3] S. B. David, T. Lu, T. Luu, and D. Pál, "Impossibility theorems for domain adaptation," in *Proc. the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 129–136, 2010.

[4] Y. Mansour, M. Mohri, and A. Rostamizadeh, "Domain adaptation: Learning bounds and algorithms," *arXiv preprint arXiv:0902.3430*, 2009.

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

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

[7] A. Gretton, K. M. Borgwardt, M. Rasch, B. Schölkopf, and A. J. Smola, "A kernel

method for the two-sample-problem," in *Proc. Advances in Neural Information Processing Systems*, pp. 513–520, 2007.

[8] N. Dehak, P. 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, 2011.

[9] Y. Jiang, K. A. Lee, and L. Wang, "PLDA in the i-supervector space for text-independent speaker verification," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2014, no. 1, p. 29, 2014.

[10] D. Garcia-Romero and C. Y. Espy-Wilson, "Analysis of i-vector length normalization in speaker recognition systems," in *Proc. Interspeech*, pp. 249–252, 2011.

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

[12] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, New York, 2007.

[13] O. Glembek, J. Ma, P. Matejka, B. Zhang, O. Plchot, L. Burget, and S. Matsoukas, "Domain adaptation via within-class covariance correction in i-vector based speaker recognition systems," in *Proc. ICASSP*, pp. 4032–4036, 2014.

[14] P. Li, Y. Fu, U. Mohammed, J. Elder, and S. Prince, "Probabilistic models for inference about identity," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 1, pp. 144–157, 2012.

[15] P. Kenny, T. Stafylakis, P. Ouellet, M. J. Alam, and P. Dumouchel, "PLDA for speaker verification with utterances of arbitrary duration," in *Proc. ICASSP*, pp. 7649–7653, 2013.

[16] M.-W. Mak, X. Pang, and J.-T. Chien, "Mixture of PLDA for noise robust i-vector speaker verification," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 1, pp. 130–142, 2015.

[17] M.-W. Mak and J.-T. Chien, "Machine learning for speaker recognition," *Cambridge University Press*, 2020.

[18] 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. ICASSP*, pp. 4052–4056, 2014.

[19] C. Zhang and K. Koishida, "End-to-end text-independent speaker verification with triplet loss on short utterances.," in *Proc. Interspeech*, pp. 1487–1491, 2017.

[20] H. Zeinali, S. Wang, A. Silnova, P. Matějka, and O. Plchot, "BUT system description to voxceleb speaker recognition challenge 2019," *arXiv preprint arXiv:1910.12592*, 2019.

[21] W. Lin, M. W. Mak, and L. Yi, "Learning mixture representation for deep speaker embedding using attention," in *Proc. Odyssey 2020 The Speaker and Language Recognition Workshop*, pp. 210–214, 2020.

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

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[24] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. Computer Vision and Pattern Recognition (CVPR)*, pp. 4700–4708, 2017.

[25] F. Wang, J. Cheng, W. Liu, and H. Liu, "Additive margin softmax for face verification," *IEEE Signal Processing Letters*, vol. 25, no. 7, pp. 926–930, 2018.

[26] Y. Liu, L. He, and J. Liu, "Large margin softmax loss for speaker verification," *arXiv preprint arXiv:1904.03479*, 2019.

[27] D. Garcia-Romero and A. McCree, "Supervised domain adaptation for i-vector based speaker recognition," in *Proc. ICASSP*, pp. 4047–4051, 2014.

[28] D. Garcia-Romero, A. McCree, S. Shum, N. Brummer, and C. Vaquero, "Unsupervised domain adaptation for i-vector speaker recognition," in *Proc. Odyssey*, pp. 260–264, 2014.

[29] J. Villalba and E. Lleida, "Bayesian adaptation of PLDA based speaker recognition to domains with scarce development data," in *Proc. Odyssey*, pp. 47–54, 2012.

[30] H. Aronowitz, "Inter dataset variability compensation for speaker recognition," in *Proc. ICASSP*, pp. 4002–4006, 2014.

[31] M. McLaren and D. Van Leeuwen, "Source-normalized LDA for robust speaker recognition using i-vectors from multiple speech sources," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 3, pp. 755–766, 2012.

[32] A. Sholokhov, T. Kinnunen, and S. Cumani, "Discriminative multi-domain PLDA for speaker verification," in *Proc. ICASSP*, pp. 5030–5034, 2016.

[33] S. O. Sadjadi, T. Kheyrkhah, A. Tong, C. Greenberg, D. Reynolds, E. Singer, L. Mason, and J. Hernandez-Cordero, "The 2016 NIST speaker recognition evaluation," in *Proc. Interspeech*, pp. 1353–1357, 2017.

[34] K. Jones, S. Strassel, K. Walker, D. Graff, and J. Wright, "Call My Net corpus: A multilingual corpus for evaluation of speaker recognition technology," in *Proc. Interspeech*, pp. 2621–2624, 2017.

[35] L. v. d. Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[36] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3. JHU Press, 2012.

[37] J. Villalba and E. Lleida, "Unsupervised adaptation of PLDA by using variational bayes methods," in *Proc. ICASSP*, pp. 744–748, 2014.

[38] S. H. Shum, D. A. Reynolds, D. Garcia-Romero, and A. McCree, "Unsupervised clustering approaches for domain adaptation in speaker recognition systems," in *Proc. Odyssey*, pp. 265–272, 2014.

[39] L. Li and M. Mak, "Unsupervised domain adaptation for gender-aware PLDA mixture models," in *Proc. ICASSP*, 2018.

[40] G. Csurka, "Domain adaptation for visual applications: A comprehensive survey," *arXiv preprint arXiv:1702.05374*, 2017.

[41] H. Aronowitz *et al.*, "Compensating inter-dataset variability in PLDA hyper-parameters for robust speaker recognition," in *Proc. Odyssey*, pp. 282–286, 2014.

[42] H. Aronowitz, "Inter dataset variability modeling for speaker recognition," in *Proc. ICASSP*, pp. 5400–5404, 2017.

[43] K. A. Lee and et al, "The I4U mega fusion and collaboration for NIST speaker recognition evaluation 2016," in *Proc. Interspeech*, pp. 1328–1332, 2017.

[44] Q. Wang, W. Rao, S. Sun, L. Xie, E. S. Chng, and H. Li, "Unsupervised domain adaptation via domain adversarial training for speaker recognition," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 4889–4893, 2018.

[45] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[46] A. Solomonoff, C. Quillen, and W. M. Campbell, "Channel compensation for SVM speaker recognition.," in *Proc. Odyssey*, vol. 4, pp. 219–226, 2004.

[47] B. Sun, J. Feng, and K. Saenko, "Correlation alignment for unsupervised domain adaptation," in *Domain Adaptation in Computer Vision Applications*, pp. 153–171, Springer, 2017.

[48] K. A. Lee, Q. Wang, and T. Koshinaka, "The CORAL+ algorithm for unsupervised domain adaptation of PLDA," in *Proc. ICASSP*, pp. 5821–5825, 2019.

[49] Y. Tu, M. W. Mak, and J. T. Chien, "Variational domain adversarial learning for speaker verification," *Proc. Interspeech 2019*, pp. 4315–4319, 2019.

[50] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[51] Y. Z. Tu, M. W. Mak, and J. T. Chien, "Information maximized variational

domain adversarial learning for speaker verification," *Proc. ICASSP*, pp. 6444–6448, 2020.

[52] S. Gao, R. Brekelmans, G. V. Steeg, and A. Galstyan, "Auto-encoding total correlation explanation," *arXiv preprint arXiv:1802.05822*, 2018.

[53] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv preprint arXiv:1701.07875*, 2017.

[54] G. Bhattacharya, J. Monteiro, J. Alam, and P. Kenny, "Generative adversarial speaker embedding networks for domain robust end-to-end speaker verification," in *Proc. ICASSP*, pp. 6226–6230, 2019.

[55] P. Germain, A. Habrard, F. Laviolette, and E. Morvant, "A PAC-Bayesian approach for domain adaptation with specialization to linear classifiers," in *Proc. International Conference on Machine Learning*, pp. 738–746, 2013.

[56] H.-Y. Chen and J.-T. Chien, "Deep semi-supervised learning for domain adaptation," in *IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, 2015.

[57] W. W. Lin, M. W. Mak, L. X. Li, and J. T. Chien, "Reducing domain mismatch by maximum mean discrepancy based autoencoders.," in *Odyssey*, pp. 162–167, 2018.

[58] Y. Li, K. Swersky, and R. Zemel, "Generative moment matching networks," in *Proc. International Conference on Machine Learning*, pp. 1718–1727, 2015.

[59] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," in *Proc. International Conference on Machine Learning*, pp. 97–105, 2015.

[60] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, ACM, 2008.

[61] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A discriminative feature learning approach for deep face recognition," in *Proc. European Conference on Computer Vision*, pp. 499–515, Springer, 2016.

[62] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proc. the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.

[63] M.-W. Mak and H.-B. Yu, "A study of voice activity detection techniques for NIST speaker recognition evaluations," *Computer Speech & Language*, vol. 28, no. 1, pp. 295–313, 2014.

[64] J. Pelecanos and S. Sridharan, "Feature warping for robust speaker verification," in *Proc. Odyssey*, 2001.

[65] P. Matejka, O. Novotnỳ, O. Plchot, L. Burget, and J. Cernockỳ, "Analysis of score normalization in multilingual speaker recognition," in *Proc. Interspeech*, 2017.

[66] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

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

[68] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical programming*, vol. 45, no. 1, pp. 503–528, 1989.

[69] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*. MIT press Cambridge, 2016.

[70] Y. Bengio *et al.*, "Learning deep architectures for AI," *Foundations and Trends® in Machine Learning*, 2009.

[71] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological cybernetics*, vol. 59, no. 4-5, pp. 291–294, 1988.

[72] Q. Xie, Z. Dai, E. Hovy, M. T. Luong, and Q. V. Le, "Unsupervised data augmentation," *arXiv preprint arXiv:1904.12848*, 2019.

[73] C. Xie, M. Tan, B. Gong, J. Wang, A. Yuille, and Q. V. Le, "Adversarial examples improve image recognition," *arXiv preprint arXiv:1911.09665*, 2019.

[74] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[75] T. Ko, V. Peddinti, D. Povey, M. L. Seltzer, and S. Khudanpur, "A study on data augmentation of reverberant speech for robust speech recognition," in *Proc. ICASSP*, pp. 5220–5224, 2017.

[76] G. Bhattacharya, J. Alam, and P. Kenny, "Adapting end-to-end neural speaker verification to new languages and recording conditions with adversarial training," in *Proc. ICASSP*, pp. 6041–6045, 2019.

[77] Y. Wu and K. He, "Group normalization," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.

[78] C. L. Li, W. C. Chang, Y. Cheng, Y. Yang, and B. Póczos, "MMD GAN: Towards deeper understanding of moment matching network," in *Proc. Advances in Neural Information Processing Systems*, pp. 2200–2210, 2017.

[79] A. Gretton, K. Fukumizu, C. H. Teo, L. Song, B. Schölkopf, and A. J. Smola, "A kernel statistical test of independence," in *Advances in Neural Information Processing Systems*, pp. 585–592, 2008.

[80] K. Okabe, T. Koshinaka, and K. Shinoda, "Attentive statistics pooling for deep speaker embedding," pp. 2252–2256, 2018.

[81] Q. Wang, K. Okabe, K. A. Lee, H. Yamamoto, and T. Koshinaka, "Attention mechanism in speaker recognition: What does it learn in deep speaker embedding?," in *IEEE Spoken Language Technology Workshop (SLT)*, pp. 1052–1059, IEEE, 2018.

[82] X. Miao, I. McLoughlin, and Y. Yan, "A new time-frequency attention mechanism for TDNN and CNN-LSTM-TDNN, with application to language identification," *Proc. Interspeech*, pp. 4080–4084, 2019.

[83] 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*, pp. 74–81, 2018.

[84] A. Nagrani, J. S. Chung, and A. Zisserman, "Voxceleb: a large-scale speaker identification dataset," in *Interspeech*, 2017.

[85] J. S. Chung, A. Nagrani, and A. Zisserman, "Voxceleb2: Deep speaker recognition," in *Interspeech*, 2018.

[86] C. Richey, M. A. Barrios, Z. Armstrong, C. Bartels, H. Franco, M. Graciarena, A. Lawson, M. K. Nandwana, A. Stauffer, J. van Hout, *et al.*, "Voices obscured in complex environmental settings (VOiCES) corpus," *arXiv preprint arXiv:1804.05053*, 2018.

[87] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *ICLR (workshop track)*, 2015.

[88] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[89] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1708.04552*, 2017.

[90] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition," *arXiv preprint arXiv:1904.08779*, 2019.

[91] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, *et al.*, "The Kaldi speech recognition toolkit," in *Proc. Workshop on Automatic Speech Recognition and Understanding*, 2011.