UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Self-supervised deep learning approaches to speaker recognition

Umair Khan

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

This thesis is submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Ph.D.)

Self-supervised Deep Learning Approaches to Speaker Recognition

Author:

Umair Khan

Thesis Advisor:

Prof. Francisco Javier Hernando Pericás

TALP Research Center,

Department of Signal Theory and Communications,

Universitat Politecnica de Catalunya (UPC) Barcelona, Spain.

Barcelona, November 2020

Abstract

In speaker recognition, i-vectors have been the state-of-the-art unsupervised technique over the last few years, whereas the recent x-vectors is becoming the state-of-the-art supervised technique, these days. Recent advances in Deep Learning (DL) approaches to speech and speaker recognition have improved the performance but these techniques are constrained to the need of phonetic or/and speaker labels for the background data. In practice, labeled background data is not easily accessible, especially when large training data is required. In i-vector based speaker recognition, cosine and Probabilistic Linear Discriminant Analysis (PLDA) are the two basic scoring techniques. Cosine scoring is an unsupervised technique whereas PLDA parameters are typically trained using speaker-labeled background data. The lack of speaker labeled background data makes a big performance gap between these two scoring techniques. The question is: how to fill this performance gap without using speaker labels for the background data? In this thesis, the above mentioned problem has been addressed using DL approaches without using and/or limiting the use of labeled background data. Three main DL based proposals have been made.

In the first proposal, a Restricted Boltzmann Machine (RBM) based vector representation of speech is proposed for the tasks of speaker clustering and speaker tracking in TV broadcast shows. This representation will be reffered to as RBM vector. A global model will be trained using all the available background data, which will be reffered to as Universal Restricted Boltzmann Machine (URBM). Then, in order to extract the desired RBM vectors for the test utterances, a single RBM will be adapted from the URBM. The experiments, performed on AGORA database, show that in speaker clustering task the proposed RBM vectors outperform the baseline i-vectors, resulting in a relative improvement of 12% in terms of Equal Impurity (EI). In the case of speaker tracking RBM vectors are used only in the speaker identification part, where the relative improvement in terms of Equal Error

Rate (EER) is 11% and 7% using cosine and PLDA scoring, respectively.

In the second proposal, DL approaches are proposed in order to increase the discriminative power of i-vectors in speaker verification. We have proposed the use of autoencoder in several different ways. Firstly, an autoencoder will be used as a pre-training for a Deep Neural Network (DNN) training. This pre-training will be carried out using a large amount of unlabeled background data. After this, a DNN classifier will be trained using relatively small labeled data. The DNN will be initialized with the parameters of the pre-trained autoencoder. Secondly, an autoencoder will be trained in a new framework, to transform i-vectors into a new representation. The purpose is to increase the discriminative power of i-vectors. The training will be carried out based on the nearest neighbor i-vectors rather than the same training i-vector. The nearest neighbor i-vectors will be chosen in an unsupervised manner. The evaluation was performed on the speaker verification trials of VoxCeleb-1 database. The results show that while using autoencoder pre-training for a DNN classifier, we gain a relative improvement of 21% in terms of EER, over i-vector/PLDA system. Whereas, applying the nearest neighbors approach to train the autoencoder, a relative improvement of 42% is gained over i-vectors with cosine scoring. Moreover, if we make use of the background data in the testing part, a relative improvement of 53% is gained over i-vector/PLDA system.

In the third proposal, we will train an end-to-end speaker verification system without using speaker labels. The idea is to utilize impostor samples along with the nearest neighbor samples to make client/impostor pairs in an unsupervised manner. The architecture will be based on a VGG-like Convolutional Neural Network (CNN) encoder which will be trained as a siamese network with two branch networks. Furthermore, another network with three CNN encoder branches will be trained using triplet loss, in order to extract unsupervised speaker embeddings. The experimental results show that both the end-to-end system and the speaker embeddings, despite being unsupervised, show a comparable performance to a similar but fully supervised baseline. Moreover, the score combination of both the systems can further improve the performance.

The above proposed approaches for speaker verification have their pros and cons over each other. From the experiments it is clear that the best result was obtained using the nearest neighbor autoencoder. However, its disadvantage is that it relies on the background i-vectors in the testing part. Compared to this, the use of autoencoder pre-training for DNN is not bound by this factor but the DNN training was carried out using speaker labels. On the other hand, the third proposal is free from both these constraints and still performs pretty reasonably. Firstly, it is a fully unsupervised approach. Secondly, it uses Mel-Spectrogram features in the testing phase and therefore avoids i-vector extraction. And finally, it does not rely on the background i-vectors in the testing phase.

Resumen

En el reconocimiento automático de hablantes, los *i-vectors* han sido la técnica no supervisada preponderante en los últimos años, mientras que los *x-vectors* se están convirtiendo en la técnica supervisada de última generación. Los avances recientes en los enfoques de aprendizaje profundo *Deep Learning* (DL) para el reconocimiento del habla y del hablante han mejorado el rendimiento, pero estas técnicas se limitan a la necesidad de etiquetas fonéticas y/o del hablante para los datos de *background*. En la práctica, los datos de *background* etiquetados no son fácilmente accesibles, especialmente cuando se requieren un volumen grande de datos de entrenamiento. En el reconocimiento de locutor basado en *i-vectors*, la distancia coseno y el análisis discriminante lineal probabilístico (PLDA) son las dos técnicas básicas de puntuación. La puntuación de la distancia coseno es una técnica no supervisada, mientras que los parámetros del PLDA normalmente se entrenan utilizando datos de fondo etiquetados por el hablante. La falta de datos etiquetados del hablante crea una gran brecha de rendimiento entre estas dos técnicas de puntuación. La pregunta es: ¿cómo llenar esta brecha de rendimiento sin usar etiquetas del hablarte en los datos de *background*? En esta tesis, el problema mencionado anteriormente se ha abordado utilizando enfoques de DL sin utilizar y / o limitar el uso de datos etiquetados. Se han realizado tres propuestas basadas en DL.

En la primera, se propone una representación vectorial de voz basada en la máquina de Boltzmann restringida (RBM) para las tareas de agrupación de hablantes y seguimiento de hablantes en programas de televisión.. Se propone entrenar un modelo global utilizando todos los datos disponibles, al que se designa como Máquina de Boltzmann restringida universal (URBM). A continuación, para extraer los vectores de RBM deseados para los experimentso de prueba, se adaptaun solo RBM del URBM. Los experimentos, realizados utilizando la base de datos AGORA, muestran que en la tarea de agrupación de hablantes los vectores RBM propuestos superan a los *i-vectors*, lo que resulta en una mejora relativa del 12% en términos de igualdad de impureza (EI). En el caso del seguimiento del hablante, los vectores RBM se utilizan solo en la etapa de identificación del hablante, donde la mejora relativa en términos de Equal Error Rate (EER) es del 11% y el 7%, utilizando la puntuación de coseno y PLDA, respectivamente.

En la segunda propuesta, se utiliza DL para aumentar el poder discriminativo de los *i-vectors* en la verificación del hablante. Se ha propuesto el uso del codificador automático de varias formas. En primer lugar, se utiliza un codificador automático como preentrenamiento de una red neuronal profunda (DNN). Este entrenamiento previo se lleva a cabo utilizando una gran cantidad de datos de *background* sin etiquetar. Después, se entrena un clasificador DNN utilizando un conjunto reducido de datos etiquetados. El DNN se inicializa con los parámetros del codificador automático preentrenado. En segundo lugar, se entrena un autocodificador en un nuevo marco, para transformar *i-vectors* en una nueva representación. El propósito es aumentar el poder discriminativo de los *i-vectors*. El entrenamiento se lleva a cabo en base a los *i-vectors* vecinos más cercanos, en lugar del mismo i-vector de entrenamiento. Los *i-vectors* vecinos más cercanos se eligen de forma no supervisada. La evaluación se ha realizado con pruebas de verificación del hablante utilzando la base de datos VoxCeleb-1. Los resultados muestran que al utilizar el preentrenamiento con autocodificador para un clasificador DNN, obtenemos una mejora relativa del 21% en términos de EER sobre el sistema i-vector / PLDA. Sin embargo, aplicando el enfoque de vecinos más cercanos para entrenar el autocodificador, se obtiene una mejora relativa del 42% sobre los *i-vectors* usando distancia coseno. Además, si utilizamos los datos de *background* en la etapa de prueba, se obtiene una mejora relativa del 53% sobre el sistema i-vector / PLDA.

En la tercera propuesta, entrenamos un sistema de verificación de locutor de principio a fin sin usar etiquetas de locutor. La idea es utilizar muestras de impostores junto con las muestras del vecino más cercano para formar pares cliente / impostor sin supervisión. La arquitectura se basar en un codificador de red neuronal convolucional (CNN) similar a VGG, que se entrenará como una red siamesa con dos redes de ramas. Además, se entrenará otra red con tres ramas de codificador CNN utilizando función de pérdida de triplete, con el fin de extraer *embeddings* de locutores sin supervisión. Los resultados experimentales muestran que tanto el sistema de principio a fin como las *embeddings* de locutores, a pesar de no estar

supervisadas, muestran un rendimiento comparable a una referencia similar pero completamente supervisada. Además, la combinación de puntuación de ambos sistemas puede mejorar aún más el rendimiento.

Los enfoques propuestos anteriormente para la verificación del hablante tienen sus pros y sus contras. A partir de los experimentos, queda claro que el mejor resultado se obtuvo utilizando el autocodificador con el vecino más cercano. Sin embargo, su desventaja es que se basa en los i-vectores de *background* en la etapa de prueba. El uso del preentrenamiento del codificador automático para DNN no está sujeto a este factor, pero el entrenamiento de DNN se llevó a cabo utilizando etiquetas de locutores. Por otro lado, la tercera propuesta está libre de estas dos limitaciones y funciona de manera bastante razonable. En primer lugar, es un enfoque totalmente no supervisado. En segundo lugar, utiliza características de Mel-Spectrogram en la fase de prueba y, por lo tanto, evita la extracción de i-vector. Y finalmente, no depende de los i-vectores de *background* en la fase de prueba.

Acknowledgments

First of all I would like to thank the Almighty ALLAH, who has always blessed me. Because of His blessings I got the confidence to manage this thesis and all other hardships in my life.

This thesis wouldn't be possible without the supervision and ideas of my advisor, Prof. Dr. Francisco Javier Hernando Pericás. I would like to thank him for his support and cooperation throughout this long journey. Working with him in TALP research center has given me confidence in developing skills and to work in a competitive environment. He provided me opportunities to enhance my research capabilities, specially in the area of Speaker Recognition.

I would like to thank all my colleagues in the TSC department for their support and cooperation. In the TALP research center, I would like to thank Miquel India who has always guided me whenever I needed. I am also very thankful to Abraham Woubie, Omid Ghahabi and Pooyan Safari for their expert advises in the area of Speaker Recognition.

Finally, I am extremely thankful to my parents and siblings. They have always supported me and prayed for my success. My wife, Shaista, was always with me throughout the journey of this PhD. She deserves a very very special thanks. It is only because of her moral support that this thesis has come to a successful completion. Last but not the least, I am grateful to all my friends who are always there whenever I feel down in hard times.

Contents

Acronyms

AHC Agglomerative Hierarchical Clustering.

ASR Automatic Speech Recognition.

BIC Bayesian Information Criterion.

BNF Bottle Neck Features.

CD Cosine Distance.

CD-1 Contrastive Divergence-1.

CI Cluster Impurity.

CNN Convolutional Neural Network.

DBN Deep Belief Network.

DET Detection Error Trade-off.

DL Deep Learning.

DNN Deep Neural Network.

EER Equal Error Rate.

EI Equal Impurity.

EM Expectation Maximization.

FA False Alarm.

FAR False Acceptance Rate.

FC	Fully Connected.
FF	Frequency Filtering.
FRR	False Rejection Rate.
GLR	Generalized Likelihood Ratio.
GMM	Gaussian Mixture Models.
ICR	Information Change Rate.
IT	Impurity Trade-off.
JFA	Joint Factor Analysis.
LDA	Linear Discriminant Analysis.
LFCC	Linear Frequency Cepstral Coefficient.
LLR	Log-Likelihood Ratio.
LPC	Linear Predictive Coefficient.
MAP	Maximum A Posteriori.
MDR	Miss Detection Rate.
MFCC	Mel-Frequency Cepstral Coefficients.
minDCF	minimum of the Detection Cost Function.
MSE	Mean Square Error.
MST	Missed Speaker Time.
MVN	Mean Variance Normalization.
PCA	Principal Component Analysis.
PLDA	Probabilistic Linear Discriminant Analysis.
PLP	Perceptual Linear Predictive.

RBM	Restricted Boltzmann Machine.
ReLU	Rectified Linear Units.
SAD	Speech Activity Detection.
SGD	Stochastic Gradient Descent.
SI	Speaker Impurity.
SVM	Support Vector Machine.
t-SNE	t-Distributed Stochastic Neighbor Embedding.
TDNN	Time Delay Neural Network.
TV	Total Variability.
UBM	Universal Background Model.
URBM	Universal Restricted Boltzmann Machine.
WCCN	Within-Class Covariance Normalization.

List of Figures

List of Tables

Chapter 1

Introduction

The process of automatically retrieving the identity of a person from his speech, i.e., speaker identification, and/or in some cases automatically verifying the identity being claimed by a person, i.e., speaker verification, is known as speaker recognition. The whole process of speaker recognition only relies on the speech signals spoken by the person. Therefore, it is supposed that the person should have unique voice characteristics. These characteristics should be able to carry enough information about the identity of the person which is known as *Voice Biometrics.* Thus, speaker recognition can be used as biometric identity, in addition to or instead of, other common biometric identities like fingerprint, face recognition and iris recognition etc. In certain scenarios, voice biometrics is more convenient than the other biometric identities, for instance, the customer service center of a bank can recognize their clients by using their voice characteristics recorded during an ongoing phone call with the client. Moreover, when the luminance of the environment is not sufficient for using face or iris recognition, one can use voice as a biometric identity which is not affected by the lack of luminance. Apart from speaker identification and verification tasks, where a single speaker is identified or verified, there are several other tasks that use voice as a speaker identity in a multiple speaker applications e.g., speaker segmentation, speaker clustering, speaker diarization and tracking etc. This thesis focuses mainly on speaker verification and partially on speaker clustering and tracking tasks.

1.1 Motivation

The emerging technology of Deep Learning (DL) has been successfully applied to various tasks of speech technologies in recent decades (Mohamed et al., 2010; Dahl et al., 2011; Senior et al., 2015). Their success has influenced the research community to make use of DL in speaker recognition tasks as well (Lei et al., 2014a; Richardson et al., 2015; K. Chen & Salman, 2011; Kenny et al., 2014; Liu et al., 2015). Unsupervised approaches, like autoencoders, Restricted Boltzmann Machine (RBM) and Deep Belief Network (DBN), as well as supervised approaches, like Deep Neural Network (DNN), have been widely used for speaker recognition. The application of DL in a speaker recognition system can be widely classified in three different categories, i.e., at the frontend, at the backend, and as an end-to-end system.

At the frontend, DL can be applied to extract the so called Bottle Neck Features (BNF), with the help of senone or phoneme state labels of speech. The BNF features can be used either to compute Gaussian Mixture Models (GMM) log-likelihood (Deng & Yu, 2014; Yamada et al., 2013) or to train a Gaussian Mixture Models–Universal Background Model (GMM)–(UBM) in a standard i-vector (Dehak et al., 2011) extraction process (Ghalehjegh & Rose, 2015; Lozano-Diez et al., 2016). These features can also be appended to the original acoustic features and can be used in several speaker recognition tasks (Lee et al., 2009; Liu et al., 2015; Jorrín et al., 2017; Jati & Georgiou, 2017; Anna et al., 2017). However, the BNF extraction is costly is terms of computations as compared to the acoustic features. Furthermore, it requires phonetic labels for the speech data. This can be difficult in practice, specially when a large amount of background data is required for the DNN training.

A more popular trend, to use DL at the frontend of a speaker recognition system, is to learn a vector based representation of speech utterances. In most cases, this vector based representation is obtained provided that the speaker labels for the background data are available, such as in (Variani et al., 2014a; Isik et al., 2015; Liu et al., 2015; Bhattacharya et al., 2017; Snyder et al., 2018). Such vector based representation of speech is commonly known as speaker embeddings. Speaker embeddings, like i-vectors, are scored with commonly used scoring techniques, for instance cosine (Dehak et al., 2010) and Probabilistic Linear Discriminant Analysis (PLDA) (Prince & Elder, 2007) backends. Usually, speaker embeddings can be obtained using a supervised approach. However, there exist very few unsupervised DL approaches to speaker embeddings extraction (Vasilakakis et al., 2013). These approaches often compromise on the performance of the system. Therefore, both the BNF features and speaker embeddings DL approaches are typically constrained to labeled background data, either phonetic labels (in the case of BNF features) or speaker labels (in the case of speaker embeddings).

At the backend, one possibility of applying DL in speaker recognition is to post-process the already existing i-vectors. For instance in (Senoussaoui et al., 2012; Stafylakis et al., 2012c) different combinations of RBMs are applied for i-vector classification. On the other hand, in (Stafylakis et al., 2012a; Isik et al., 2015), the authors proposed DL approaches in order to improve the discriminative power of i-vectors. In other words, it transforms i-vectors into new vectors which are scored using cosine or PLDA backends. Similarly, in (Ghahabi & Hernando, 2017) decision scores were directly obtained from DNNs, without using cosine and PLDA scoring backends.

Unlike the above trends, an end-to-end system is typically trained by feeding the feature vectors at the input and obtaining decision scores at the output, without a backend scoring technique. Thus, end-to-end systems are more challenging to train and optimize. Feeding directly the audio samples, at the input, is a huge problem in terms of computational complexity. Therefore, most of the end-to-end systems, nowadays, are still based on the handcrafted features like Mel-Frequency Cepstral Coefficients (MFCC). In the past years, there have been several attempts to DL based end-to-end systems, for instance in (Heigold et al., 2016; S.-X. Zhang et al., 2016; C. Zhang & Koishida, 2017; Dey et al., 2018; Heo et al., 2017). In some cases, for example in (S.-X. Zhang et al., 2016; Heigold et al., 2016), the training is performed in several steps. Firstly, the frontend speaker embeddings part is trained, usually, using multi-class speaker labels. Then, the backend part is added to the network and a joint fine tuning is performed using binary class labels.

As discussed above, most of the different trends of DL approaches to speaker recognition are typically constrained to labeled background data. On the other hand, i-vector extraction is an unsupervised process which does not require phonetic or speaker labels. The two commonly scoring techniques, i.e., cosine and PLDA, are widely used to decide if two i-vectors belong to the same speaker. Unlike cosine scoring, PLDA parameters are trained using speaker labeled background data. Usually, a large number of speakers having several utterances per speaker are required. In practice, access to the speaker labeled background data is costly. This results in a huge performance gap between cosine and PLDA scoring for i-vectors. In (Khoury et al., 2014; Novoselov, Pekhovsky, & Simonchik, 2014), automatic labeling techniques were proposed but they could not appropriately estimate the true labels because the results reported were still far from that of PLDA with actual labels. Similarly, in (Ghahabi & Hernando, 2014a,b, 2018, 2017) several alternative DL based approaches were proposed without using speaker-labeled background data. However, most of their results were based on fusion strategies with i-vector/PLDA, in order to obtain good performance. Moreover, their approach was based on training a separate DL based model for every target speaker in the database, which may not be convenient in some cases.

1.2 Objectives

The main objective of this thesis is to take advantage of the recent advancements in DL approaches for speaker recognition while minimizing/avoiding the requirement of labeled background data. In order to accomplish this objective, we have applied DL in three different ways, i.e, to learn a vector based representation of speech, to increase discriminative power of i-vectors, and to train an end-to-end speaker verification system. We have addressed one major objective in speaker clustering and tracking tasks, and two major objectives in speaker verification task. Following are the specific aims and objectives of this thesis explained in details.

I. To apply unsupervised DL approach in order to learn a vector based representation of speech using RBM for speaker clustering and tracking tasks. Such vector based representation was reffered to as RBM vector, and was first proposed in (Safari et al., 2016) for the task of speaker verification. In this objective, we make use of RBM vectors for the tasks of speaker clustering and speaker tracking in TV broadcast shows. A global model, reffered to as Universal Restricted Boltzmann Machine (URBM), is trained with a large amount of unlabeled background data. Then, an adapted RBM model is trained for every speech utterance. In order to extract the desired RBM vectors, the weight matrices of the adapted RBM models are concatenated and whitened. More details are found in Chapter 3.

II. One possible way of using DL in speaker recognition is to combine it with the existing unsupervised state of the art approaches i.e., i-vectors. In this objective we make use of unsupervised DL techniques with the help of autoencoder pre-training and nearest neighbors approach for i-vector speaker verification. In order to increase the discriminative power of i-vectors, we propose the use of autoencoder training in several ways. In the first case, we train an unsupervised autoencoder as a pre-training for a supervised DNN classifier. This will be reffered to as hybrid autoencoder-DNN. This hybrid autoencoder-DNN training converges faster than the conventional DNN. Moreover, the accuracy of the speaker verification system is also improved. Detailed explanation can be found in Chapter 4. In the second case, we propose to make use of the nearest neighbor i-vectors for speaker verification. During the training, the autoencoder tries to reconstruct a nearest neighbor i-vector instead of the same training i-vector. In this way, the network learns speaker variability in an implicit way. In the third case, the autoencoder is trained by feeding a large set of nearest neighbor i-vectors at the input. Further details are stated in Chapter 5.

III. To make use of impostor i-vectors along with the nearest neighbor i-vectors in a Convolutional Neural Network (CNN) based end-to-end speaker verification system. In this objective, we divide the database into two partitions having equal number of utterances. From one partition we select potential nearest neighbor i-vectors, while from the other partition we select potential impostor i-vectors. In this way we generate the client and impostor pairs in an unsupervised manner. Using these training pairs we train a VGG-like (Simonyan & Zisserman, 2014) CNN encoder based end-to-end speaker verification system. The network is trained as a double-branch siamese network for a binary classification problem. Moreover, we also trained a triple-branch siamese network using triplet loss, which is aimed to extract unsupervised speaker embeddings. In the testing phase, for the double-branch siamese network we directly obtained speaker verification scores, whereas for the triple-branch siamese network we extracted speaker embeddings and then scored them using cosine scoring. In Chapter 6, both the networks and the CNN encoder architectures are explained in detail.

1.3 Outline

The rest of the thesis is organized in the following manner.

Chapter 2 briefly reviews speaker recognition as well as the main state-of-the-art approaches used in the past. It describes the possible uses of DL techniques in i-vector based speaker recognition. After this, some recent developments and trends in speaker recognition using DL approaches, i.e., DL front-ends, like, BNF features and speaker embeddings, DL backends and DL based end-to-end systems, are briefly discussed. Finally, it explains the different evaluation metrics for measuring the performance of speaker verification, clustering and tracking systems.

Chapter 3 describes the impact of RBM vectors for the tasks of speaker clustering and speaker tracking. Different lengths for extracting the RBM vectors are investigated and compared with the state-of-the-art systems. Evaluation experiments are performed on AGORA database which contains audio recordings of Catalan TV broadcast shows.

Chapters 4 and 5 explain our proposed algorithms using DL techniques in the task of i-vector based speaker verification. In these two chapters, all the neural networks are trained using utterance level features i.e., i-vectors. In Chapter 4, we describe our proposed DNN speaker embeddings using autoencoder pre-training. The autoencoder was trained in an unsupervised manner, whereas the DNN training was carried out in a supervised way. In Chapter 5, we avoid the supervised DNN training by using only unsupervised autoencoder. In this chapter, we explain our proposed unsupervised autoencoder speaker embeddings using k nearest neighbor approach. The unsupervised k nearest neighbor selection algorithms are also discussed in this chapter. Experimental results are given on VoxCeleb database.

The previous proposals are only applicable in the i-vector space because they aim at improving the discriminative power of i-vectors. Hence, in Chapter 6 we describe our CNN based siamese networks which are trained using frame level features i.e., spectrograms. Two siamese networks, trained in an unsupervised manner, are aimed to perform an end-to-end speaker verification task and to extract speaker embeddings, respectively. The unsupervised nearest neighbors and impostor samples were used as client/impostor pairs in these approaches. In this chapter, we explain the details of the CNN encoder architecture used as a branch in both the siamese networks. The performance of the proposed CNN based siamese networks is evaluated on VoxCeleb database.

Finally, in Chapter 7 some conclusions are drawn as the findings of this thesis. It also gives directions for possible future research lines in this area.

Chapter 2

State of the Art in Speaker Recognition

This chapter briefly explains the state of the art approaches in the area of Speaker Recognition. It briefly explains the different speaker recognition tasks e.g., speaker identification and verification, speaker segmentation and clustering, and speaker diarization and tracking. This chapter also contains a brief introduction of the convectional pre-processing and speaker modelling techniques used in speaker recognition, like feature extraction and supervectors and i-vectors. Moreover, the different scoring techniques for i-vectors, like cosine and Probabilistic Linear Discriminant Analysis (PLDA), along with session compensation techniques like Within-Class Covariance Normalization (WCCN) and Linear Discriminant Analysis (LDA) are also described. Furthermore, some score normalization and calibration techniques are briefly explained.

Afterwards, this chapter explains the recent advancements using Deep Learning (DL) approaches to speaker recognition. These approaches are broadly categorized in DL frontend, backend and end-to-end systems. The frontend approaches include the popular trends of Bottle Neck Features (BNF) and the so called speaker embeddings extractors like x-vectors (Snyder et al., 2018). The backend approaches include i-vector post-processing and scoring techniques using DL, whereas the end-to-end approaches consist of the popular siamese networks. In this thesis we have contributed in the tasks of speaker verification, speaker clustering and speaker tracking. Therefore, the different evaluation metrics used in these tasks are also explained at the end of this chapter.

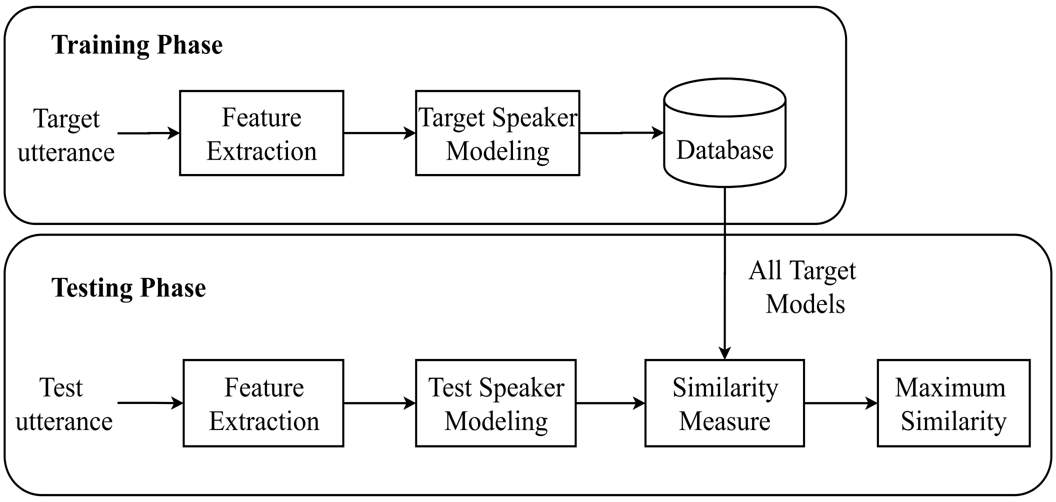


Figure 2.1: Block diagram of a speaker identification system

2.1 Speaker Recognition Tasks

Speaker recognition is a general term used for any task that involves recognizing of a person using his/her voice characteristics. Speaker recognition is mainly categorized in speaker identification and speaker verification. Speaker identification is a task in which a person is being recognized among multiple enrolled speakers. Whereas, speaker verification is a task in which a test person's voice is matched only with the claimed identity, which, based on the score, is either accepted or rejected. Despite of these two main tasks, there are other applications as well, for example, speaker segmentation, speaker clustering, speaker tracking, speaker diarization or speaker indexing etc. These tasks are briefly explained as follows.

Speaker Identification

Speaker identification is a multi-class classification problem where a test speech is identified against one of the multiple pre-enrolled speakers. This is called a closed-set approach. In an open-set approach, the test speech can also be identified as an unknown speaker, if it is not matched to either of the pre-enrolled speakers. In Figure 2.1 a speaker identification system is shown. It has two main phases, i.e., training phase and testing phase. In the training phase, first of all useful features are extracted for all the target speakers. Using these features, speaker models are trained, e.g., Gaussian Mixture Models (GMM) (Reynolds & Rose, 1995) and i-

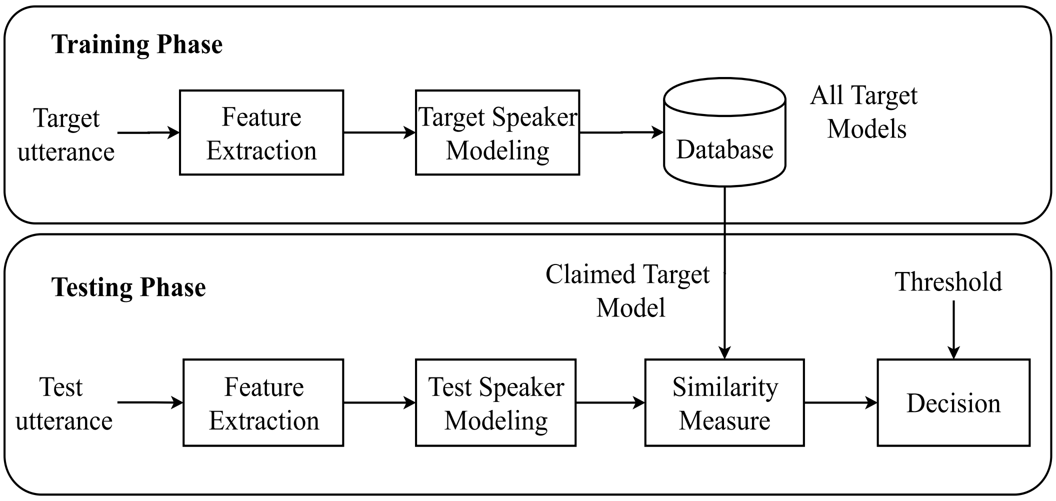


Figure 2.2: Block diagram of a speaker verification system

vectors (Dehak et al., 2011). In this way the target speakers are enrolled and stored in the database. In testing phase speaker model is trained for the unknown speech in a similar way as in the training phase. A similarity is measured between the test and all the target models based on which the test model is assigned to the target model with maximum similarity among all (Schwartz et al., 1982). If the test model does not match with any of the target models and is declared as unknown, it can be enrolled as a new target speaker (open set problem) or it can be discarded (close set problem), depending upon the application.

Speaker Verification

On the other hand, speaker verification is a binary classification problem where a test speaker claims an identity and the system should verify whether his claim is true or false (Bimbot et al., 2004). In Figure 2.2 a speaker verification system is shown. It has two main phases, i.e., training phase and testing phase. The training phase, is identical as that of the speaker identification system, where all the target speakers are enrolled in the system. Whereas, in the testing phase the unknown speaker claims an identity which is verified by the system. The similarity measure is computed between the test and the claimed target speaker only. A threshold is provided in order to decide whether the claim is true or false. The threshold selection depends on the sensitivity of the application. Speaker verification is further categorized in text-dependent or text-independent scenarios. In the text-dependent speaker verification the test speaker is required to utter exactly the same text or phrase. In the text-independent case the test speaker is free of uttering any specific phrase or text. This freedom of speech makes it a more challenging task as compared to text-dependent case, and is more common in real world applications.

Speaker Segmentation

Speaker segmentation is the task of detecting points in a multi-speaker audio which are more likely to be the change points between different speakers. According to (S. Chen & Gopalakrishnan, 1998) there are three basic approaches to perform speaker segmentation, i.e., silence based segmentation, model based segmentation and metric based segmentation. The silence based methods assume that there occurs a silence between the utterances of two different speakers. These methods are usually dependent on a threshold value of the short term energy but the results obtained are poor (Kemp et al., 2000). There are two types in this method, i.e., energy based and decoder based systems. The energy based systems rely on energy detector to detect the silence frames (Kemp et al., 2000), whereas the decoder based systems rely on a certain recognition system to find the speaker change points from the silence detected locations (Kubala et al., 1997). Since there is no clear relationship between the existence of silence in a recording and a speaker change, because this idea has no direct dependency with the acoustic changes in the speech data. Therefore, these types of approaches to speaker segmentation are not commonly applied in speaker diarization systems.

The model based approach to speaker segmentation rely on training different models for each acoustic class in a closed set, for instance, channel based classes like telephone or broad-band, gender based classes like male or female, and other classes like music, speech or silence. These models are trained using some training database. In the testing phase, the audio signal is classified using the maximum likelihood approaches (Gauvain et al., 1999). The detected boundaries between different models are considered to be the segmentation change points. These methods are not very robust and cannot generalize for unseen data.

The metric based approach to speaker segmentation is the most commonly used techniques (Ajmera & Wooters, 2003). In this approach it is not required to input any prior knowledge about the number of speakers. A distance metric is used between every two adjacent speech segments to determine the possible speaker change points (Sinha et al., 2005). There are two hypothesis tested, i.e., $H1$, assuming the two speech segments belong to the same speaker and described by a single model, and $H2$, assuming the two speech segments belong to different speakers and described by different models. The distance metric is then compared to a threshold in order to go for one of the two hypothesis using Bayesian Information Criterion (BIC), Generalized Likelihood Ratio (GLR) (Willsky & Jones, 1976) or Information Change Rate (ICR) (Vijayasenan et al., 2007).

Speaker Clustering

Speaker Clustering is the task of grouping homogeneous segments that are likely to belong to a single speaker. The two main approaches are online and offline speaker clustering. In online clustering, the speech segments are grouped or split in consecutive iterations which continues until an optimal number of speakers is acquired. Whereas, in offline clustering the entire speech file is available beforehand. Therefore it achieves relatively better performance as compared to the online clustering approach.

The most widely used approach to speaker clustering is the Agglomerative Hierarchical Clustering (AHC). AHC is based on iteratively splitting or merging clusters until an optimum number of clusters is reached. It introduces a hierarchy of clusters showing the relations between the speech segments to be clustered. Then it merges the speech segments based on a similarity or distance criteria. When the optimal stage is reached, the system stops any more iterations and gives an output hypothesis. AHC can be approached in two different ways, i.e., bottom-up and top-down. Two parameters are need to be defined in both bottom-up and top-down clustering. The first parameter is a distance metric between speech segments to determine the acoustic similarity/difference. The distance metric is used to make the decision whether or not two segments must be merged (in bottom-up clustering) or split (in top-down clustering). The second parameter is to define a certain stopping criterion in order to determine when to stop the iterations, e.g, when the optimal number of clusters (in this case speakers) is reached.

In bottom-up AHC the system starts with maximum number of clusters and keeps merging them iteratively, according to some criteria for example the BIC criteria. It is necessary to define the initial number of clusters in order to initiate the algorithm. The bottom-up approach has been applied decades ago in pattern

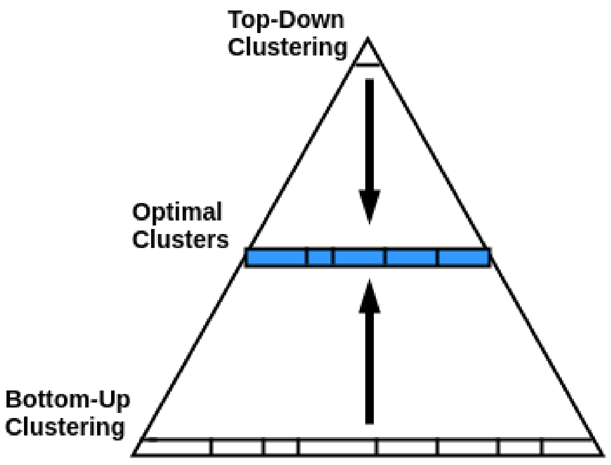


Figure 2.3: Agglomerative Hierarchical Clustering

classification problems in (R. O. Duda & Hart, 1973). In (R. Duda et al., 2001) and (Siegler et al., 1997) it was considered for the first time for the task of speaker clustering. On the other hand, the top-down AHC approach is the opposite to the bottom-up approach. In top-down approach the system starts with small number of clusters, usually just one cluster, and iteratively splits them into new clusters until the optimal stage is reached. An example of bottom-up and top-down AHC approaches is shown in Figure 2.3. The colored bar shoes optimal number of clusters where the system stops.

Speaker Diarization

Speaker diarization refers to the process of automatically segmenting and clustering an audio recording into homogeneous speech regions. It answers the question *who spoke when* without assuming any prior knowledge about the number or characteristics of speakers (Tranter & Reynolds, 2006). There are three basic steps in a typical speaker diarization system. In the first step the system it discriminates speech frames from the silence ones which is called speech activity detection. In the second step the speaker boundaries are detected in order to segment the audio. In the third step it groups the speech segments into homogeneous clusters. The speaker diarization task assumes no prior knowledge about the speakers' identities or how many speakers are participating in the audio. The whole process is carried out in the following sub tasks.

Frontend processing: It usually, includes speech enhancement and noise reduction, speech activity detection and feature extraction. The noise part in an audio is suppressed in order to enhance the output Signal to Noise ratio. This can be achieved by using the Wiener's filtering approach. After this, useful features, like the Mel-Frequency Cepstral Coefficients (MFCC) are extracted from the waveform in order to train appropriate models. Finally, an energy based speech activity detection is usually performed to remove the silence frames.

Speaker segmentation: It segments the whole audio file into speech segments that are only from one speaker. As discussed earlier, speaker change boundaries has to be detected in order to perform segmentation. Finally, the segments generated here are input to the speaker clustering system.

Speaker clustering: The segments generated are grouped together in order to make acoustically homogeneous clusters. Ideally a single cluster is assigned to each speaker that appears in the audio.

Speaker Tracking

Speaker tracking is applied in scenarios where a target speaker, also known as Person Of Interest (POI), is tracked in a multi-speaker audio file. In order to answer the question *when the target speaker speaks?* in the audio, is reffered to as speaker tracking task (Luque, 2012). The main stages of a speaker tracking system are to determine the speaker change points, perform a speaker segmentation, and then identify the target speaker i.e., speaker identification. Usually, in order to detect speaker changes, a fixed length window is slid over the audio and a distance metric is computed between consecutive windows. This distance is set to a threshold to decide whether there exists a speaker change. The speaker change detection can be performed using a Generalized Likelihood Ratio (GLR) (Gish et al., 1991; Gish & Schmidt, 1994), as in (Bonastre et al., 2000), or the Divergence Shape distance (Lu et al., 2001; J. P. Campbell, 1997), as in (Lu & Zhang, 2002). After segmentation, the segments are matched with the target speaker using some modelling strategy like the conventional Gaussian Mixture Models (GMM). In this way a speaker identification is performed in order to decide which segment belongs to the given target speaker or to which target speaker, if there are multiple target speakers.

Speaker Indexing

Speaker indexing is a term which is interchangeably used for speaker diarization systems (Moattar & Homayounpour, 2012; Kwon & Narayanan, 2005; Sturim et al., 2001). However, according to (Sturim et al., 2001), *"speaker indexing is defined as the application of speaker detection to the retrospective search of large speech archives"*. A speaker indexing system follows similar steps as that of a speaker diarization system, i.e., frontend processing (noise removal, feature extraction and speech activity detection), speaker segmentation and speaker clustering (Moattar & Homayounpour, 2012; Kwon & Narayanan, 2005).

2.2 Conventional Techniques

Conventionally, in speaker recognition, the acoustic features extracted from speech are used to discriminate between different individuals. These features possess physical and behavioral characteristics like size of throat, shape of mouth, and fundamental frequency or pitch. In order to model the speakers, the conventional strategies include mainly GMM (Reynolds & Rose, 1995), GMM mean Supervector (GSV) and i-vector (Dehak et al., 2011). The current modeling strategies are mostly based on a Deep Neural Network (DNN) based approaches which model the speakers in a discriminative manner. In the following sections, we will summarize the state of the art approaches used in different stages of a speaker recognition system, starting from features extraction to model training and scoring.

2.2.1 Feature Extraction

Feature extraction is a primary pre-processing step in any speaker recognition system. In this step, meaningful features are extracted from the raw speech data. Usually, the features vectors contain the acoustic characteristics and speaker information. In speaker recognition, the system focus only on the speaker information. A standard feature commonly used in speaker recognition is a set of short-term acoustic features which are obtained from the spectrum of speech. The spectrum of a speech signal convey information about the physiology of vocal tract which can be very helpful in speaker discrimination. Nowadays, the most commonly used features in speech technologies is Mel-Frequency Cepstral Coefficients (MFCC) (Furui, 2004, 1981). However, other features like Linear Predictive Coefficient (LPC), Linear Frequency Cepstral Coefficient (LFCC), Perceptual Linear Predictive (PLP) coefficients and Frequency Filtering (FF) coefficients (Nadeu et al., 2001) are also widely used. Moreover, to improve the performance, the first and second order derivatives are often appended to these basic coefficients. These are called delta and double-delta coefficients. It is recommended to perform features normalization, like Mean Variance Normalization (MVN) (Hansen & Hasan, 2015), in order to compensate possible environmental mismatch.

2.2.2 Supervectors and i-Vectors

The story of supervectors and i-vectors (Dehak et al., 2011) begins with Gaussian Mixture Models (GMM). In the past GMM was considered to be the most common state of the art approach for speaker modeling. A GMM is a parametric probability density function represented as a weighted sum of multiple Gaussian component densities.

$$P(x) = \sum_{i=1}^{k} w_i g(x/\mu_i, \Sigma_i) \tag{2.1}$$

where x is a N-Dimensional feature vector, w_i is the weight of ith Gaussian component and $g(x/\mu_i, \Sigma_i)$ is the ith Gaussian component density. μ_i and Σ_i are the mean vector and covariance matrix of the ith Gaussian component respectively. The mixture weights w_i must satisfy the constraint: $\sum_{i=1}^{k} w_i = 1$. A GMM is represented by the three parameters:

$$\lambda = \{w_i, \mu_i, \Sigma_i\}; \; i = 1 \ldots k \tag{2.2}$$

A speaker GMM is trained by using a few iterations of the Expectation Maximization (EM) algorithm (Dempster et al., 1977). In most cases, it is difficult to train a GMM from scratch with a few data. therefore, a Maximum A Posteriori (MAP) adapted GMM is trained using a global GMM model called Universal Background Model (UBM) (Reynolds et al., 2000). A UBM is trained using the EM algorithm for a large amount of background data. Each speaker's GMM is the result of adapting the UBM model, which represents a large population of speakers, to better represent the characteristics of the specific speaker being modeled.

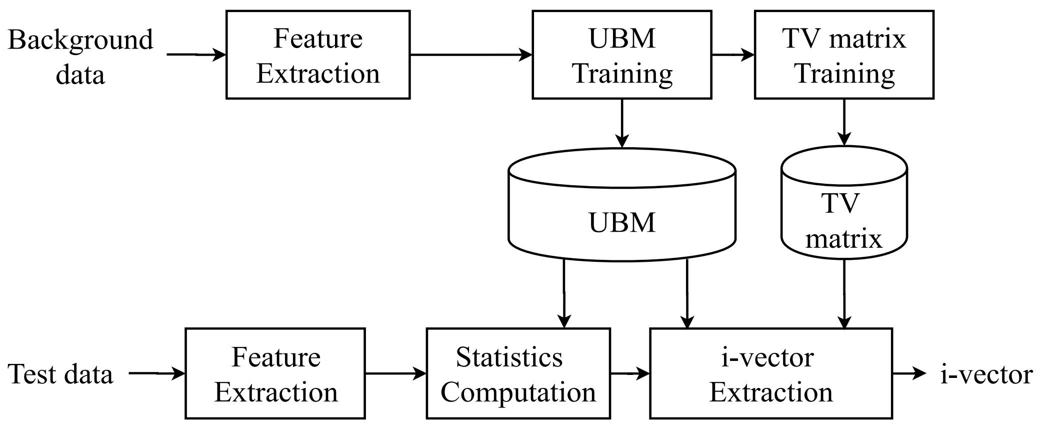


Figure 2.4: Block diagram of i-vector extraction process

The mean vectors of all the components of a MAP adapted GMM are stacked in a high dimensional vector which is called GMM mean supervector or supervector. Supervectors are compared using Support Vector Machine (SVM) classifier (W. M. Campbell et al., 2006). A suervector is defined as:

$$s = (\mu_1, \mu_2, \mu_3, ..., \mu_k)^t \tag{2.3}$$

where s is the desired supervector, μ_k represents the mean vector of the kth GMM component and t represents transpose operation. One of the technique for session variability compensation of supervectors is the Joint Factor Analysis (JFA) (Kenny, 2005; Kenny et al., 2008). According to JFA, a supervector of a speaker, can be split into speaker dependent, speaker independent, channel dependent and residual parts. Each of these components can be represented by a low-dimensional set of factors. In SVM based techniques, these compensation techniques are carried out in the high dimensional supervector space which requires a huge memory space for the training. In (Dehak et al., 2011), it was proposed to first reduce the supervector into a low dimensional space and then apply the session variability compensation in the lower dimensional space. Thus, supervectors can further be transformed to lower dimensional vectors called i-vector. I-vectors are actually a compact representation of speech and have been the state of the art over the last few years. The supervector is represented as:

$$s = m + Tv \tag{2.4}$$

where s is the source side supervector of a speaker, m is a speaker independent supervector obtained from UBM, T is a low rank Total Variability (TV) matrix and v is the vector of latent variables. The mean vector of the posterior distribution of v is conditioned on the first and second order Baum-Welch statistics. This is reffered to as i-vector w and is computed as follows:

$$w = \left(I + T^t \Sigma^{-1} \mathcal{N}(u) T\right)^{-1} T^t \Sigma^{-1} \tilde{\mathcal{F}}(u) \tag{2.5}$$

where $\mathcal{N}(u)$ is the diagonal matrix that contains the zeroth order Baum-Welch statistics, $\tilde{\mathcal{F}}(u)$ is a supervector of the centralized first order statistics, and Σ is a diagonal covariance matrix. The T matrix is trained using the EM algorithm provided the Baum-Welch statistics of the training data. The whole process of i-vector extraction is shown in Figure 2.4. More in depth details are found in (Dehak et al., 2011).

2.2.3 i-Vector Backends

Cosine Distance

Given the i-vectors of two speech utterances, the cosine distance (Dehak et al., 2010) is computed in order to decide if the two i-vectors belong to the same speaker or different speakers. The cosine distance score between i-vector w_i and i-vector w_j is given by:

$$Score(w_i, w_j) = \frac{w_i^t w_j}{||w_i|| \times ||w_j||} = \cos(\theta_{w_i, w_j}) \tag{2.6}$$

If the i-vectors of two utterances point in the same direction, the cosine distance score takes the lowest value of up to -1, which means that the two i-vectors belong to the same speaker. If they point in opposite directions, the cosine distance score takes the highest value of up to 1, which means that the two i-vectors belong to different speakers. Cosine scoring gives a reasonable performance and it does not require speaker labels for the background data. However, if the speaker labels are available, then a session variability compensation technique like Within-Class Covariance Normalization (WCCN), Linear Discriminant Analysis (LDA) or Probabilistic Linear Discriminant Analysis (PLDA) (Prince & Elder, 2007; Kenny, 2010) are usually applied before scoring. These techniques improve the performance of the speaker verification system as compared to cosine scoring. However, in most cases, it is difficult to access speaker labels for the background data, which is the biggest drawback associated with these techniques.

Within-Class Covariance Normalization

In order to compensate for intersession variability, WCCN uses the within-class covariance matrix, denoted by $\mathbf{S}_w$, to normalize the cosine kernel functions (Dehak et al., 2011). WCCN was mainly used as a normalization technique in order to improve the robustness of SVM based speaker recognition systems (Hatch et al., 2006; Hatch & Stolcke, 2006). First, the within-class covariance matrix $\mathbf{S}_w$ is calculated as:

$$\mathbf{S}_w = \frac{1}{S}\sum_{s=1}^{S}\frac{1}{n_s}\sum_{i=1}^{n_s}(w_{s,i}-\widehat{w}_s)(w_{s,i}-\widehat{w}_s)^t \tag{2.7}$$

where $w_{s,i}$ is the ith i-vector of speaker s, n_s is the total number of i-vectors belonging to speaker s, S is the total number of speakers in the background set of i-vectors, and $\widehat{w}_s$ is the speaker-dependent mean i-vector, obtained on each speaker class. Then the projection WCCN matrix is computed using the Cholesky factorization of $\mathbf{S}_w^{-1}$ such that:

$$\mathbf{S}_w^{-1} = \mathbf{A}_{WCCN}\mathbf{A}_{WCCN}^t \tag{2.8}$$

$$\widehat{w} = \mathbf{A}_{WCCN}^t w \tag{2.9}$$

where $\widehat{w}$ is the projection of i-vector w. After WCCN projection, the projected vector $\widehat{w}$ conserves the direction and the dimension of the original feature space (Hansen & Hasan, 2015).

Probabilistic Linear Discriminant Analysis

PLDA performs the scoring of two i-vectors along with their corresponding session variability compensation. In PLDA, it is assumed that every i-vector can be decomposed as follows:

$$w = m + \Phi\zeta + \epsilon \tag{2.10}$$

where w is the decomposing i-vector, m is a global offset, the columns of Φ are the eigenvoices, ζ is a latent vector having a standard normal prior, and ϵ is the residual vector which is normally distributed with zero mean and full covariance matrix denoted by Σ. The PLDA parameters are trained using a large amount of background data with speaker labels using the EM algorithm, as in described in (Prince & Elder, 2007).

There are two hypotheses H_s and H_d, in order to score the two given i-vectors w_1 and w_2 that are involved in an experimental trial for a speaker verification task. H_s indicates that the two i-vectors w_1 and w_2 belong to the same speaker, whereas H_d indicates the other way around, i.e., the two i-vectors w_1 and w_2 belong to different speakers. The speaker verification score can then be calculated as the log-likelihood ratio of the two hypotheses H_s and H_d, as follows:

$$Score(w_1, w_2) = log\frac{p(w_1, w_2|H_s)}{p(w_1|H_d)p(w_2|H_d)} \tag{2.11}$$

The global offset m is removed, by supposing that the i-vectors are generated from a Gaussian distribution. Therefore, the log-likelihood ratio can be computed in a closed-form solution which results in the following equation:

$$Score(w_1, w_2) = w_1^t Q w_1 + w_2^t Q w_2 + 2w_1^t P w_2 \tag{2.12}$$

where P and Q are the square matrices represented in terms of within-class and between-class covariance matrices as follows:

$$P = \Sigma_b^{-1} - (\Sigma_b \Sigma_w \Sigma_b^{-1} \Sigma_w)^{-1} \tag{2.13}$$

$$Q = \Sigma_b^{-1} \Sigma_w (\Sigma_b \Sigma_w \Sigma_b^{-1}) \tag{2.14}$$

where the within-class covariance matrix is expressed as:

$$\Sigma_w = \Phi\Phi^t \tag{2.15}$$

and the between-class covariance matrix is expressed as:

$$\Sigma_b = \Phi\Phi^t + \Sigma \tag{2.16}$$

In order to accelerate the scoring process, typically, the above equations are summarized to be computed in the PLDA lower dimensional space. More in depth details are found in (Garcia-Romero & Espy-Wilson, 2011). It was shown in (Kudashev et al., 2016), that the LDA within-class covariance matrix and between-class covariance matrix can also be used in equations 2.13 and 2.14.

2.2.4 Score Normalization and Calibration

Score normalization aims at augmenting the useful biometric information conveyed in a score. It is applied in order to preserves Log-Likelihood Ratio (LLR) score properties for a single system and for multiple systems when performing score level fusion among them. It is believed that PLDA scores are usually well calibrated in terms of LLRs. However, score normalization has shown to increase the performance (Reynolds et al., 2000; Sturim & Reynolds, 2005; Cumani et al., 2011).

Some of the normalization techniques are cohort normalization (Roseberg et al., 1992), test normalization or shortly T-norm, and zero normalization or shortly Z-norm) (Zheng et al., 2006). Cohort normalization uses cohort speakers that are close to the target speaker. The normalized Z-scores $\hat{s}$ are computed using the mean and standard deviation of the cohort scores' (μ and σ) (Sturim & Reynolds, 2005; Cumani et al., 2011):

$$\hat{s} = \frac{s-\mu}{\sigma} \tag{2.17}$$

Cohort scores describe a score distribution which depends on the reference and the probe to be compared. In Equation 2.17 the μ and σ might only consider one of the reference or probe, or both. It is referred to Z-norm when the μ and σ of Equation 2.17 is considered of the reference samples only. On the other hand, it is reffered to T-norm when μ and σ are considered of the probe samples. In (Navratil & Ramaswamy, 2003), it was shown that Z-norm and T-norm Gaussianize the overall score distributions. Z-norm and T-norm can be applied together on top of each other which is reffered to as Z/T-norm (Sturim & Reynolds, 2005). They are also applied in parallel, which is known as symmetric normalization or shortly S-norm (Cumani et al., 2011). Equal weights are assigned to the Z-score and T-score. The S-norm score is computed by (Cumani et al., 2011):

$$\hat{s} = 0.5(\frac{s-\mu_z}{\sigma_z}) + 0.5(\frac{s-\mu_t}{\sigma_t}) \tag{2.18}$$

Where μ_z and σ_z, and μ_t and σ_t are the corresponding means and standard deviations of Z-norm and T-norm, respectively. It has been shown that normalization applied in a series combination of Z-norm and T-norm, i.e., Z/T-norm, was useful for GMM-UBM systems (Sturim & Reynolds, 2005). However, for PLDA systems the S-norm has shown success, as shown in (Cumani et al., 2011).

Typically, the scores normalized using Z-norm, T-norm and/or Z/T-norm are not LLRs. That is where the score calibration comes into play, in order to preserve LLR properties (Brummer, 2010). Score calibration is also useful for cosine or other non-LLR scoring techniques. Score calibration is done either through logistic regression or through isotonic regression (Brümmer et al., 2014). Calibration through logistic regression, also known as linear calibration, is based on a bias term w_0 and a scaling term w_1. These terms are typically estimated to provide a robust calibrated score $\hat{s}$ for a score s. A linear combination of the bias and scaling weights w_0 and w_1, and score s is optimized in order to obtain the best fit to the predictive values which represents the class labels (Prince, 2012). In particular for binary decisions, the predictive values are 1 and 0 representing two classes, A and B. The logistic sigmoid function $\sigma(x)$, activations and predictions are linked by the calibration function $a(s)$:

$$a(s) = w_0 + w_1 s \tag{2.19}$$

$$\sigma(a(s)) = \frac{1}{1 + \exp^{-a(s)}} \tag{2.20}$$

After optimizing w_0 and w_1 in terms of maximum likelihood, the above equation is expressed as the posterior probability (Prince, 2012):

$$Pr(A/s) = \sigma(a(s)) \tag{2.21}$$

Thus, the LLR calibrated scores $\hat{s}$ are approximated using the logit function, which is the inverse of the sigmoid function $\sigma(\cdot)$:

$$\hat{s} \approx logit(\sigma(a(s))) = a(s) = w_0 + w_1 s \tag{2.22}$$

In score calibration using isotonic regression the original comparison scores are converted to LLRs. The Pool Adjacent Violators or shortly PAV-LLR method is a special case of isotonic regression (Brummer, 2010; Brümmer & De Villiers, 2011; Zadrozny & Elkan, 2002). For empirical scores, the PAV-calibrated LLR scores can be estimated through linear interpolation. Score calibration does not effect the discrimination performance but it sustains the LLR score properties. The PAV method (Brummer, 2010) performs an optimal calibration for a given data. Linear calibration can only achieve good performance on a limited operating regions, whereas the PAV method is capable of achieving good calibrating on a wide range of operating regions, mainly because of its non-linearity (Brümmer et al., 2014).

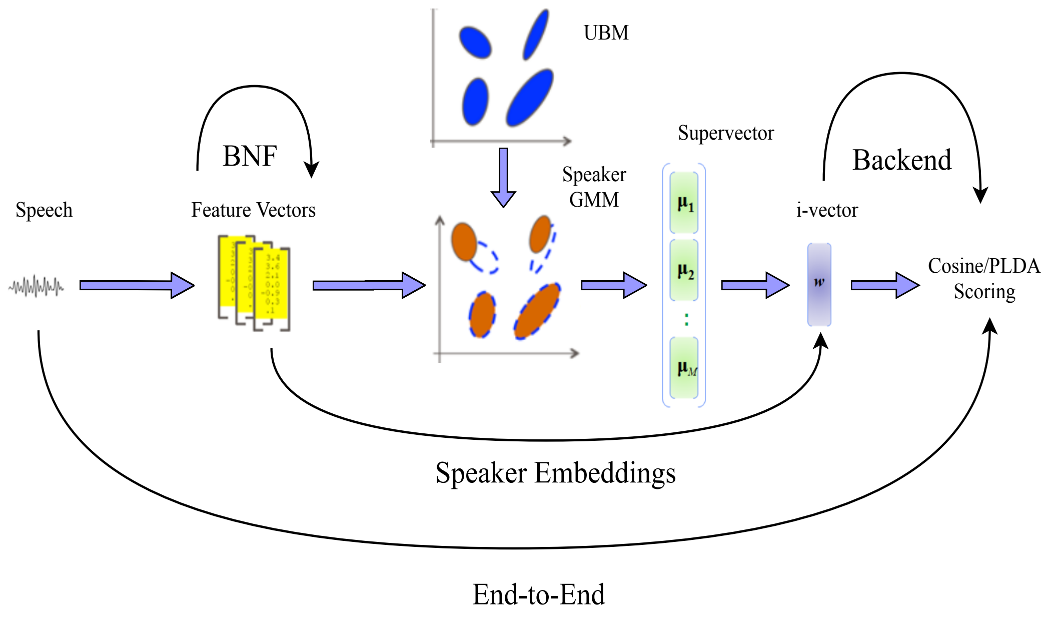


Figure 2.5: Possible Deep Learning Approaches to Speaker Recognition.

2.3 Deep Learning Approaches

Deep Learning approaches have been applied in speaker recognition decades ago (Oglesby & Mason, 1989, 1990; Phan et al., 2000; Pawar et al., 2005). One of the problems, then, was the limitation in the training data and computational resources. However, the recent advancements in computing hardware has inspired the community to apply DL techniques with more complex and deep architecture (Nagrani et al., 2017; Chung et al., 2018; Shon et al., 2018; Snyder et al., 2018; Nagrani et al., 2019). DL approaches can improve the performance of a standard speaker verification system in several ways. For example, at the frontend, DL can be applied to learn the so called Bottle Neck Features (BNF) and to learn the stand-alone speaker embeddings. At the backend, DL can be applied to train alternative scoring backends for i-vectors. Moreover, it can also be applied to train an end-to-end system.

Figure 2.5 shows a standard i-vector extraction process (Dehak et al., 2011). First of all, the speech signal is represented by a set of meaningful features vectors, for instance, the MFCC features in most cases. Then using a pre-trained UBM, a MAP adapted GMM model is trained for these feature vectors. The mean vectors of the GMM model are stacked to generate the higher dimensional supervector.

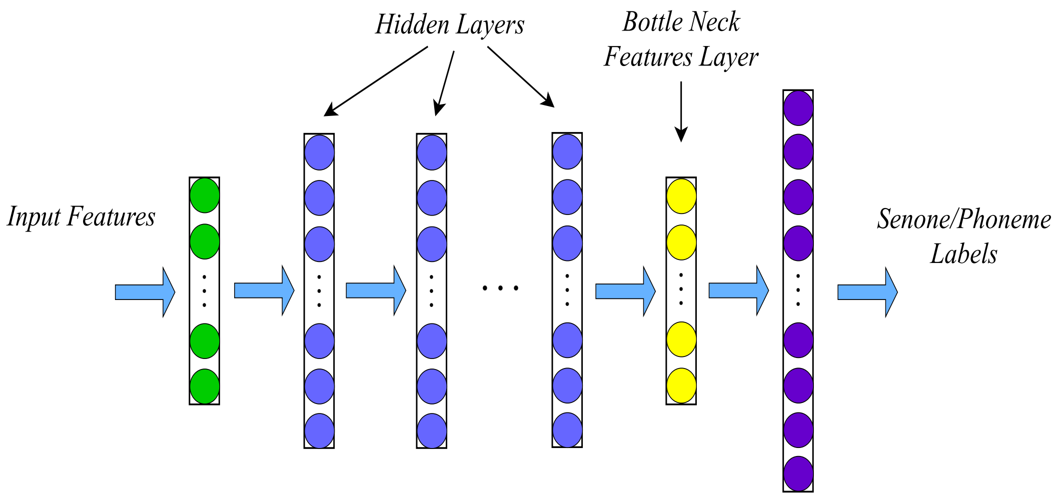


Figure 2.6: A typical architecture of BNF features extractor.

Finally, a Baum-Welch statistics computation is carried out and the supervector is reduced to lower dimensional i-vector which is scored using cosine/PLDA scoring techniques. In Figure 2.5, the curved arrows represent the possible tends of applying DL approaches in the whole i-vector extraction process for speaker recognition.

2.3.1 Frontends

Bottle Neck Features

Various DL architectures have been proposed to learn the BNF using acoustic features, typically used in Automatic Speech Recognition (ASR) systems. It has been shown that a substantial improvement can be achieved when the BNF features are used or to train a UBM adapted GMM in a standard i-vector extraction process (Lei et al., 2014b; McLaren et al., 2015; Ghalehjegh & Rose, 2015; Lozano-Diez et al., 2016; Lee et al., 2009; Liu et al., 2015; Jorrín et al., 2017; Jati & Georgiou, 2017; Anna et al., 2017). Moreover, the BNF features, in some cases, have also shown improvements if appended to the the standard acoustic features. Figure 2.6 shows a typical architecture of the BNF feature extraction. Usually, the architecture contains several hidden layers. The hidden layer before the last layer is narrowed to few neurons and is typically called the Bottle Neck layer. The hidden unit values of this layer, given the input feature vectors, is taken as the BNF features.

The input layer is fed with a concatenation of consecutive ASR feature vectors. ASR feature vectors are usually the log filter bank energies with no delta and double

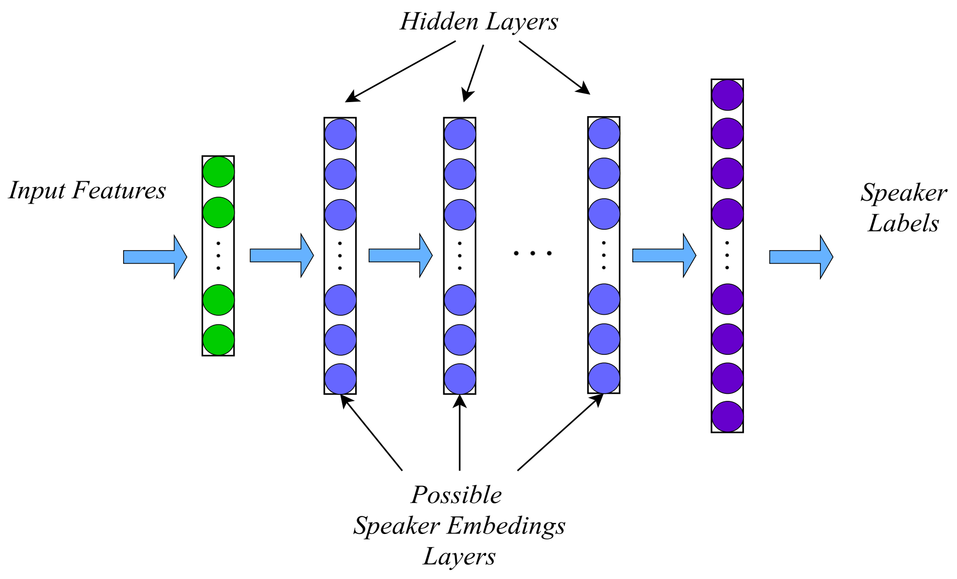


Figure 2.7: A typical architecture of speaker embeddings extractor.

delta coefficients appended. The output layer is usually activated with softmax activation function. For the bottleneck layer usually linear activation is used, and for the rest of the hidden layers Rectified Linear Units (ReLU), sigmoid or other similar activations like tanh is applied.

Although the i-vectors extracted using the BNF features results in a higher accuracy in general, there are a few disadvantages as well. Firstly, the Deep Neural Network (DNN) training itself increases the computational cost of the whole process of i-vector extraction. Secondly, the need of the phonetic labels for training the DNN model is also a drawback of this approach.

Speaker Embeddings

A more popular use of DL in speaker recognition, nowadays, is to learn a compact vector based representation of speaker, as an alternative to the stat of the art i-vectors. This compact representation of speaker is often reffered to as speaker embeddings. Typically, the inputs to the DNN are a concatenated sequence of consecutive feature vectors and the outputs are speaker labels.

A typical DNN architecture used for the speaker embeddings extraction is shown in Figure 2.7. The DNN is trained using the feature vectors of the training data, knowing the speaker labels for each utterance. In the testing phase, the feature vectors of a given test utterance are propagated through the network to get the speaker

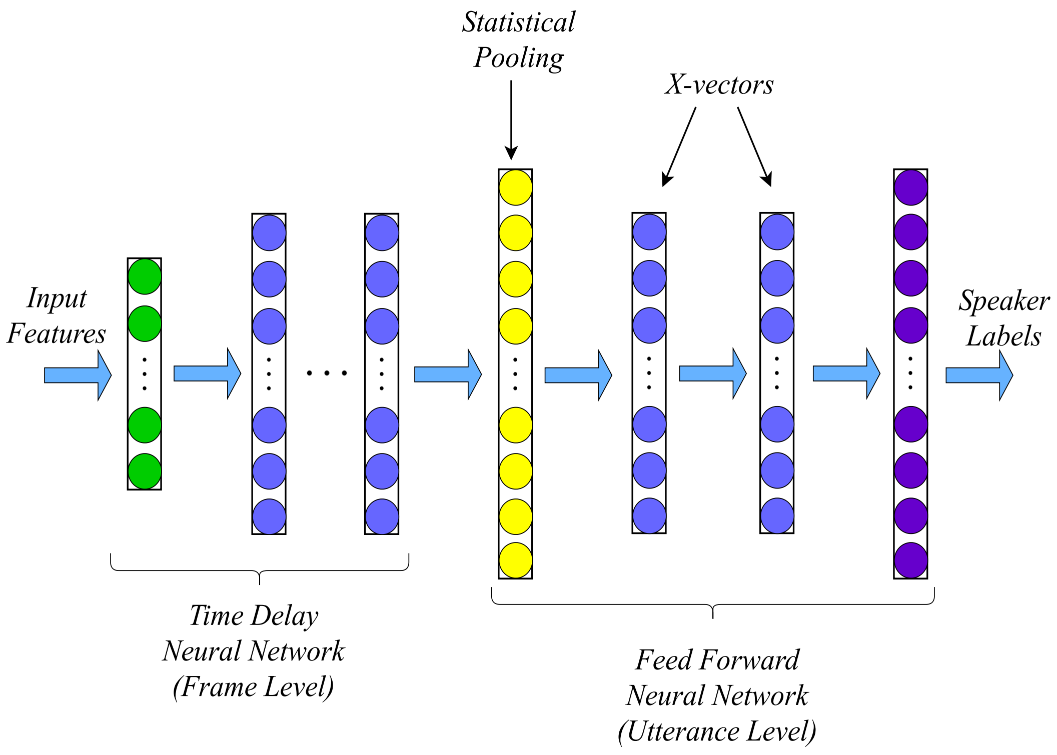

Statistical Pooling
X-vectors
Input Features
Speaker Labels
Time Delay Neural Network (Frame Level)
Feed Forward Neural Network (Utterance Level)

Figure 2.8: A typical architecture of x-vector extractor.

embeddings from a certain hidden layer. The mean of the posterior probabilities of the embeddings layer can be computed as speaker embeddings (Variani et al., 2014a). In (Liu et al., 2015) a Principal Component Analysis (PCA) dimensionality reduction is applied to these probabilities before extracting the speaker embeddings. Different activation functions are applied to the different layers of the network (Isik et al., 2015; Bhattacharya et al., 2017). Typically, a linear activation is applied to the speaker embeddings layer.

Speaker embeddings extracted using a Time Delay Neural Network (TDNN), known as the x-vectors (Snyder et al., 2018), is becoming the state of the art approach in speaker recognition these days. The first part of the TDNN, shown in Figure 2.8, works in the frame level features which takes feature vectors stacked on top of each other as the input. The output of this part is connected to a statistical pooling layer. The second part deploys a feed forward network which propagates the statistically pooled utterance level vectors through the network. The output of this part is connected to the speaker class labels more details can be found in (Snyder et al., 2018).

X-vectors have shown to further improve the performance if data augmentation is applied to the input training data. Data augmentation i.e., addition of noise and reverberation, was shown to be effective in case of x-vectors while it is not very effective in case of i-vectors (Snyder et al., 2018). However, the i-vector extraction has the advantage of being unsupervised, provided that the backend is cosine scoring.

Nowadays, Convolutional Neural Network (CNN) have shown very promising results when used in the speaker embeddings extractors (Nagrani et al., 2017; Chung et al., 2018; Nagrani et al., 2019), based on (Simonyan & Zisserman, 2014; He et al., 2016). In case of CNN based architectures, the spectrograms or Mel-spectrograms of the speech utterances can be used as the input features to the network (India et al., 2019). Moreover, attention based models are also widely used as an alternative of averaging the probabilities at output of the embeddings layer (rahman Chowdhury et al., 2018; Zhu et al., 2018). A recent proposal of using multi-head attention model (India et al., 2019), which is based on (Simonyan & Zisserman, 2014), have shown some improvement over the basic attention models for speaker verification.

Triplet loss, which was used for image recognition in (J. Wang et al., 2014; Schroff et al., 2015), has also inspired the community to employ it in speaker embeddings extraction architecture. A triplet loss is based on a combination of three speech utterances, i.e., Anchor, Positive and Negative. The triplet loss function tries to minimize the distance between Anchor and Positive, while at the same time maximizes the distance between Positive and Negative. In other words, a triplet loss minimizes the intra-speaker variability while maximizing the inter-speaker variability at the same time.

Experimental results have shown that speaker embeddings perform better specially in short utterances speaker recognition as compared to i-vectors (Bhattacharya et al., 2017; Snyder et al., 2017). This implies that DL technology can better model the characteristics of a speaker when only a short duration of speech is available. This is very important in most of the real world problems where the identity of the speaker has to be recognized in a short time.

In most of these works, the input features to the speaker embeddings extractor network is usually the hand crafted features for examples MFCC, spectrograms or Mel-spectrograms. However, there are some attempts to avoid the hand crafted features and use the speech waveform directly (Jung et al., 2018; Ravanelli & Bengio, 2018). However, these approaches are still developing.

The need of speaker labeled background data, for training the DNN, is a major drawback of speaker embeddings architectures. There has been many proposals to tackle this problem and avoid speaker labeled background data, but in most cases the performance has been compromised compared to the supervised speaker embeddings. Another drawback of the speaker embeddings is that they are not very compatible with the traditional PLDA backend. This is because speaker embeddings are usually extracted from hidden layer outputs where the posterior distributions are perfectly Gaussian (Ghahabi & Hernando, 2017).

2.3.2 Backends

One possibility to use DL at the backend of speaker recognition system is to apply it within the i-vector extraction algorithm (Dehak et al., 2011). There are several DL based proposals in i-vector backends. For instance in (Senoussaoui et al., 2012; Stafylakis et al., 2012c) different combinations of RBMs are applied for i-vector classification. In (Stafylakis et al., 2012a; Isik et al., 2015), DL has been proposed in order to improve the discriminative power of i-vectors. They transform i-vectors into new vectors which are scored using cosine or PLDA backends. On the other hand, in (Ghahabi & Hernando, 2014a,b, 2017) several unsupervised DL backends were proposed for i-vectors. In these works, they trained a separate target model for each target speaker using DBN adaptation. During the test, decision scores were directly obtained from the network, without using cosine and PLDA scoring backends.

In (Guzewich & Zahorian, 2017; Tan & Mak, 2017), DL approaches were applied to compensate the effect of noise and reverberations in i-vectors. Moreover, in (Shon et al., 2017) DL is applied for domain adaptation for i-vectors, and in (Bousquet & Rouvier, 2017) for compensation of duration mismatch of speech between train and test i-vectors. On the other hand, DL based alternatives to PLDA were proposed in (Novoselov, Pekhovsky, Simonchik, & Shulipa, 2014; Stafylakis et al., 2012b; Villalba et al., 2017), in order to perform session variability compensation along with scoring of i-vectors.

As it was mentioned earlier, PLDA is the most efficient scoring techniques for i-vectors. PLDA, at the cost of speaker-labeled background data, scores two given i-vectors along with the session variability compensation. In order to compete with PLDA, several DL based approaches were proposed. Of these approaches, many used speaker labeled background data for training, with no significant improvement achieved as compared to PLDA, for example in (Stafylakis et al., 2012b; Villalba et al., 2017). However, a combination of PLDA with RBM and autoencoder, has shown some improvements over PLDA (Novoselov et al., 2015; Pekhovsky et al., 2016).

2.3.3 End-to-end Systems

Most of the DL approaches discussed above, i.e., BNF features to extract i-vectors, and speaker embeddings are subjected to a backend scoring technique like cosine or PLDA in order to make a decision. On the contrary, an end-to-end system is trained in such a way that it provides decision scores without a backend scoring technique. End-to-end systems perform multiple tasks, like learning speaker embeddings and scoring, at the same time as a unified task, trained jointly at once. Such end-to-end systems are more challenging to train and optimize. That is why these systems are still under investigation.

Working directly with the audio samples is a huge problem in terms of computational complexity. Therefore, most of the end-to-end systems, nowadays, are still based on the handcrafted features like MFCC etc, as inputs, as shown in Figure 2.5. There have been several attempts to DL based end-to-end systems, for instance in (Heigold et al., 2016; S.-X. Zhang et al., 2016; C. Zhang & Koishida, 2017; Dey et al., 2018; Heo et al., 2017). Most of these works are limited to text-dependent speaker verification.

In some cases, for example in (S.-X. Zhang et al., 2016; Heigold et al., 2016), the training is performed in several steps. Firstly, the frontend speaker embeddings part is trained, usually, using multi-class speaker labels. Then, the backend part is added to the network and a joint fine tuning is performed using binary class labels.

A typical DL based end-to-end system has been shown in Figure 2.9. This is based on a typical siamese network with two branches. The frontend part of a siamese network usually consists of two branches, while the backend part is a single feed forward neural network. The branch networks consist of input layer and several hidden layers each. The input features of two utterances are fed into the two branch networks each. The two branches share weights with each other, and thus are identical, in most cases. The final layer of each branch is input to a concatenation layer which proceeds into a single feed forward architecture as shown in Figure 2.9.

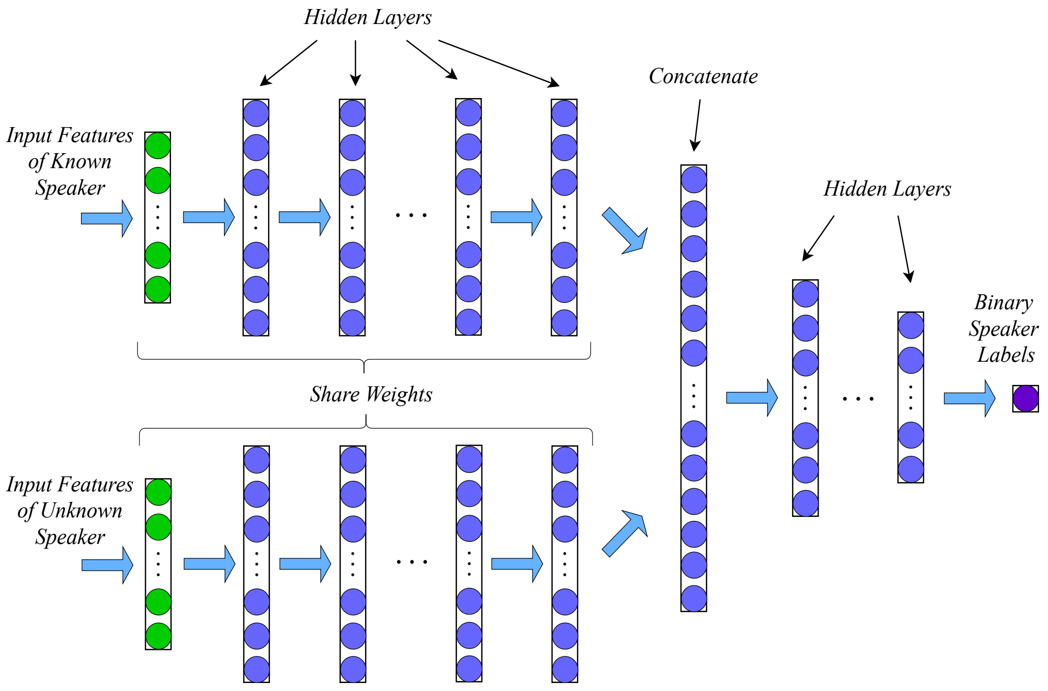


Figure 2.9: A typical siamese architecture of an end-to-end speaker verification system.

The backend part consists of several hidden layers, in which the final layer has only one or two neurons. If the final layer is activated by a sigmoid function, then it has one neuron whereas if softmax activation is used, then it has two neurons. In order to train the network, binary speaker labels are used to compute the cross-entropy between the predicted and actual values. In the testing phase, feature vectors of a trial pair are fed into the network and a decision score is obtained at the end.

2.4 Evaluation Metrics

In the testing phase of a speaker verification system, a trial pair is input to the system for which the system gives a real number score. The score reflects the probability if the trial pair belong to the same speaker or different speakers. Based on this score a final decision is taken using an empirical threshold. For every experimental trial a true key is available which indicates if the trial is a client trial or impostor trial. If the utterances in the trial pair are actually from the same speaker, it is called a client trial, and if they are from different speakers, it is called an impostor trial. Given the true labels for all the trial pairs, the False Acceptance Rate (FAR)

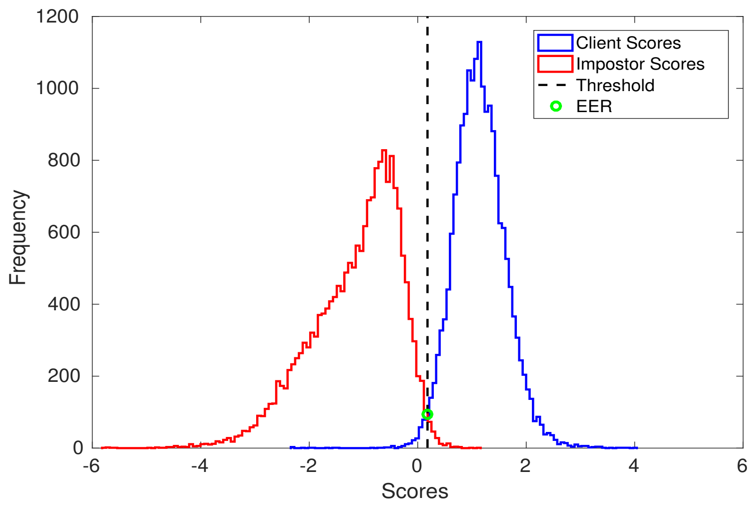


Figure 2.10: False Acceptance Rate (FAR) and False Rejection Rate (FRR)

is computed as the percentage of impostor trials that are incorrectly accepted. Similarly, False Rejection Rate (FRR) is computed as the percentage of client trials that are incorrectly rejected. Figure 2.10 shows the histograms of the scores of client and impostor trials. Usually, the client scores are higher than the impostor scores. A certain part of each histogram is overlapped with the other one. A decision threshold is subjected on the scores. Scores higher than threshold are accepted as client scores. There comes a point where both the FAR and FRR intersect each other for a specific value of threshold. This point is marked as the Equal Error Rate (EER). EER is one of the state of the art metrics for measuring the performance of speaker verification systems. The lower is the value of EER, the better is the system, and vice versa.

Another commonly used metric, in speaker verification, is the minimum of the Detection Cost Function (minDCF). DCF is a weighted sum of FAR and FRR in terms of the threshold t, and is computed as follows:

$$DCF(t) = \alpha_1 FAR(t) + \alpha_2 FRR(t) \tag{2.23}$$

where α_1 and α_2 are the scalar weights that depends on the sensitivity of the application. minDCF is the minimum value of the $DCF(t)$ function. Usually it is recommended to give higher value to α_2 as compared to α_1 because we do not want to incorrectly accept an impostor as a client. In other words, a high value of thresh-

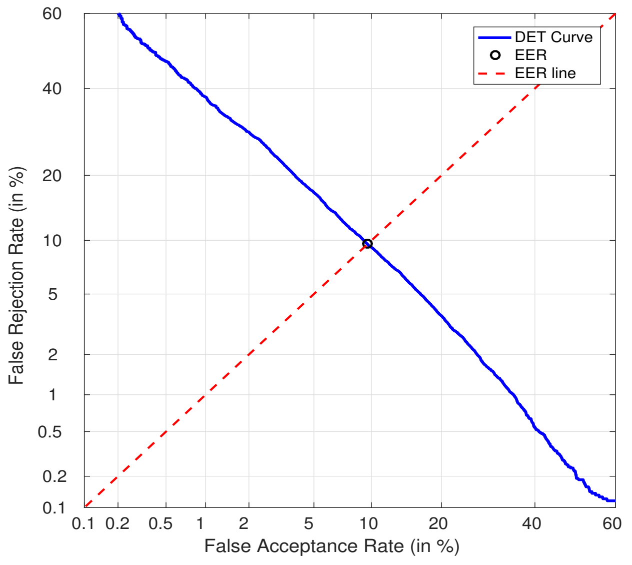


Figure 2.11: Detection Error Trade-off (DET) Curve

old is recommended in sensitive applications where it is dangerous to accept any impostor as client. For instance, in biometric systems for banks, it is very risky to put a low threshold and easily accept a trial as client trial.

FAR and FRR are inversely proportional to each other. Therefore, there is a trade-off between these two metrics when the value of threshold is varied. FAR and FRR are plotted in a Detection Error Trade-off (DET) curve against each other which is the third commonly used metric in speaker verification. Figure 2.11 shows an example of the DET curve. The red line is the EER line. The intersecting point of EER line with the DET curve is the exact value where FAR is equal to FRR. The system is considered to be better if the DET curve is close to the origin, and vice versa.

The results of the speaker clustering system were evaluated in terms of Cluster Impurity (CI). CI measures the quality of a cluster, *to what extent a cluster contains segments from different speakers*. However, this metric has a trivial solution when there is only one segment per cluster. To deal with this, Speaker Impurity (SI) was measured at the same time. SI measures *to what extent a speaker is distributed among clusters*. There is always a trade-off between these two metrics (van Leeuwen, 2010). CI and SI were plotted against each other in an Impurity Trade-off (IT) curve and an Equal Impurity (EI) point was marked as a working point.

We evaluated the results for speaker segmentation in terms of False Acceptance Rate (FAR) and Miss Detection Rate (MDR), as discussed in (Kotti et al., 2008). The overall speaker tracking system was evaluated in terms of False Alarm (FA) and Missed Speaker Time (MST). In this case, FA is the percentage of duration (in seconds) that is falsely accepted for a target speaker while MST is the percentage of duration (in seconds) that is falsely rejected for a target speaker.

Chapter 3

RBM Vectors for Speaker Clustering and Tracking

Unsupervised deep learning architectures like Restricted Boltzmann Machine (RBM), Deep Belief Network (DBN) and Deep Autoencoders have the ability of representational learning power. A first attempt to use RBM at the backend in a speaker verification task was made in (Senoussaoui et al., 2012). Our research group has put efforts into the front end of a speaker verification system, in order to learn a compact and fixed dimensional speaker representation in the form of a speaker vector by means of RBM adaptation (Ghahabi & Hernando, 2018; Safari et al., 2016; Ghahabi & Hernando, 2015). They also make use of DBN adaptation in the i-vector/PLDA (Probabilistic Linear Discriminant Analysis) framework as a the backend for speaker verification (Ghahabi & Hernando, 2017). The vector representation of speakers in (Safari et al., 2016), was referred to as RBM vector. It has been shown that the RBM vectors can extract speaker specific information that can be competitive to i-vector based approach for speaker verification task. This has led us to apply this kind of vector representation of speakers to the tasks of speaker clustering and speaker tracking, as shown in (Khan et al., 2018; Khan, Safari, & Hernando, 2019).

In this chapter, we propose the use of RBM vectors (Safari et al., 2016) for the tasks of speaker clustering and speaker tracking. The RBM vector is extracted in several steps. First of all, a global or Universal RBM—referred to as URBM—is trained with all the available background data. Then, an adapted RBM model is trained for every test utterance. The visible to hidden weight matrices along with the corresponding bias vectors of these adapted RBMs are concatenated to generate RBM supervectors. The RBM supervectors are subjected to a Principal Component Analysis (PCA) whitening and dimensionality reduction to extract the desired RBM vectors.

For the speaker clustering task, we extract RBM vectors using the proposed method. All the utterances that are to be clustered are represented by RBM vectors. Then we cluster these RBM vectors by applying a bottom-up Agglomerative Hierarchical Clustering (AHC) approach using cosine and PLDA scores. For the speaker tracking task, we implement a two step strategy. The first step is based on speaker change detection by using divergence shape distance as in (Lu & Zhang, 2002). The audio is segmented according to these speaker change points. In the second step, the segments generated are matched with all the target speakers, in order to specify which segment belongs to which target speaker. The target speakers are enrolled in the system in prior. We represent all the segments and target speakers by RBM vectors. Then, the RBM vectors of all the segments are scored against the RBM vectors of all the target speakers using cosine and PLDA scoring. We have found that the RBM vector representation of speakers is successful in both these tasks. The experimental results show that the RBM vector outperforms the conventional i-vectors based systems using both the cosine and PLDA scoring methods.

The rest of the chapter is organized as follows: Section 3.1 explains the detailed procedure of the proposed vector representation of speakers by using URBM, RBM adaptation and PCA dimensionality reduction. Section 3.2 contains a brief description of the speaker clustering system. Section 3.3 contains a detailed description of the two steps of our speaker tracking system. Section 3.4 describes the experimental setup, the database used. The results obtained both for speaker clustering and speaker tracking tasks are discussed in Section 3.5. Finally, in Section 3.6, some conclusions are drawn as the findings of this work.

3.1 RBM Vector Representation

In this work, we applied the compact vector based representation of speakers using RBM adaptation, which was reffered to as RBM vectors in (Safari et al., 2016), for speaker tracking and speaker clustering tasks. Figure 3.1 shows a detailed block

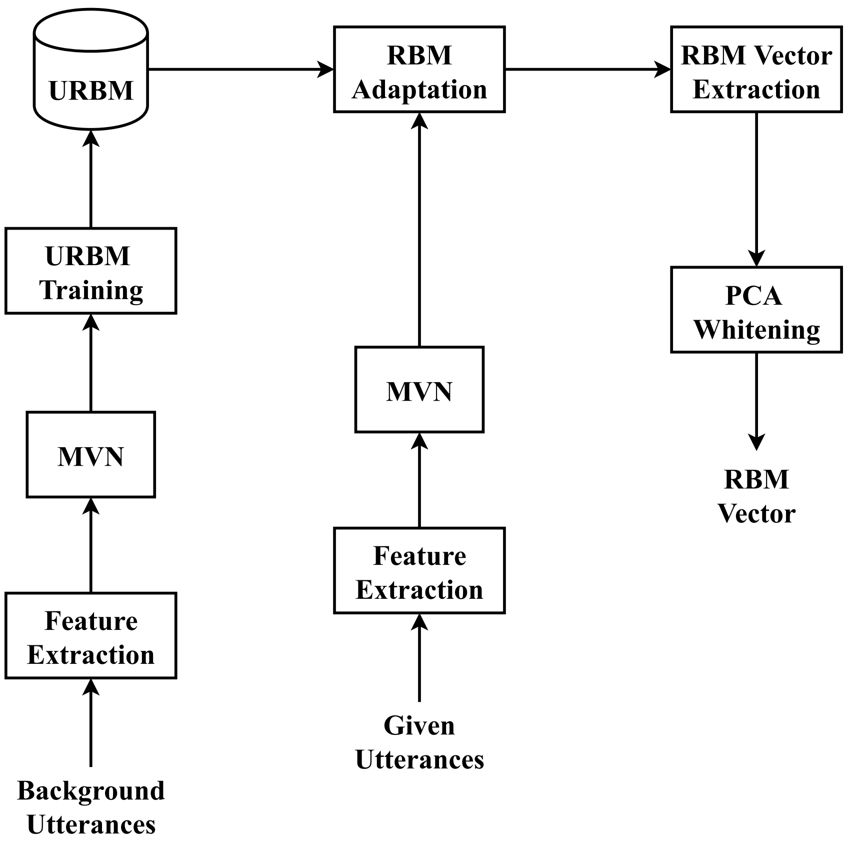


Figure 3.1: Block diagram showing different stages of the RBM vector extraction.

diagram of the proposed RBM vector extraction process. First of all, a global model, which is referred to as URBM, is trained with a large amount of background data. The idea of such a global model is inspired from Universal DBN training and adaptation in (Ghahabi & Hernando, 2014a, 2017), where a DBN model is adapted from the UDBN for every target and test speaker. Similarly, in this work the URBM is adapted to the data of every test speaker and thus an RBM is trained for every test speaker. The visible to hidden weight matrices of these adapted models are used to generate the desired vector representation for the corresponding speaker. In the final step a PCA whitening and dimensionality reduction is applied. These vector representations of speakers are further used in the tasks of speaker clustering and speaker tracking using cosine and PLDA scoring techniques. The whole process of the vector representation of speakers is divided into three main steps, namely URBM training, RBM adaptation and RBM vector extraction using PCA whitening with dimensionality reduction. In the following sections these steps are explained in detail.

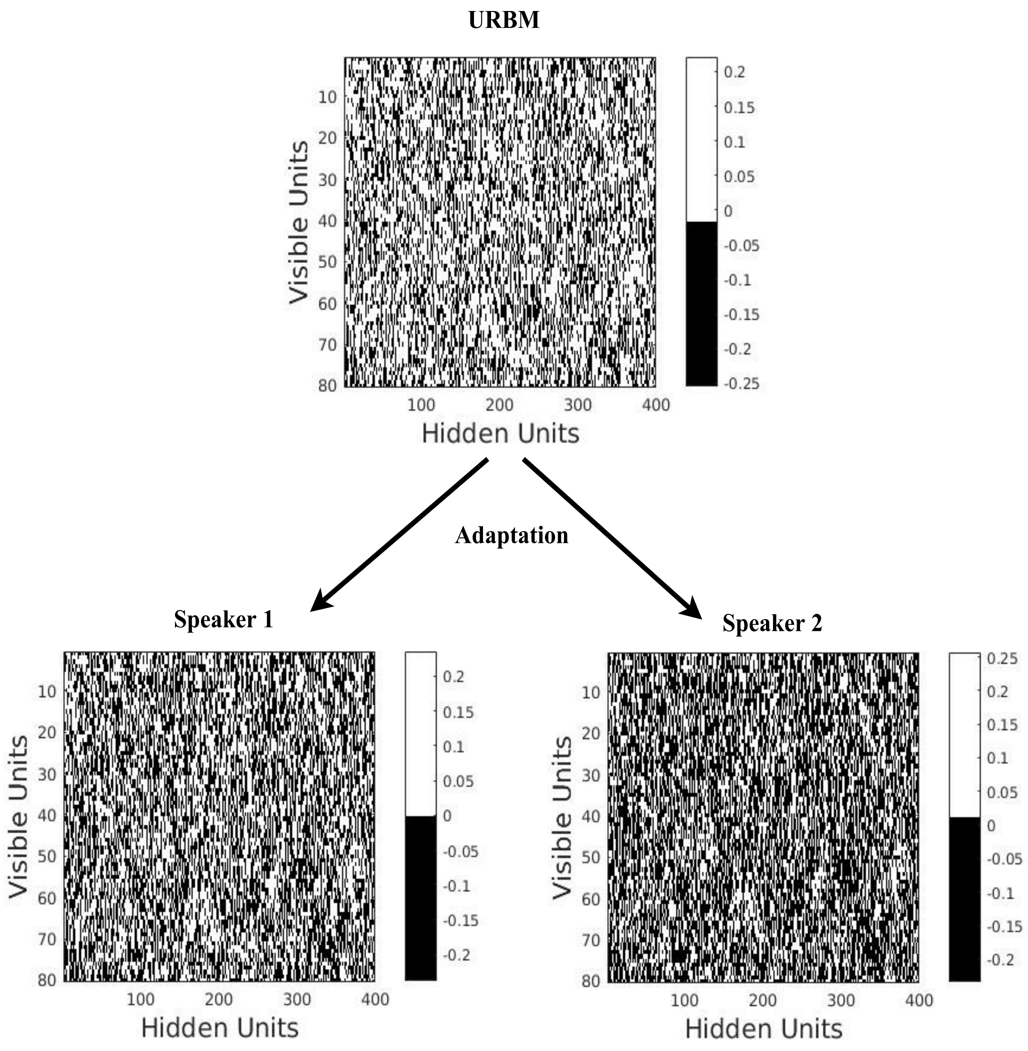


Figure 3.2: Comparison of the weight matrices of URBM and randomly selected adapted RBMs.

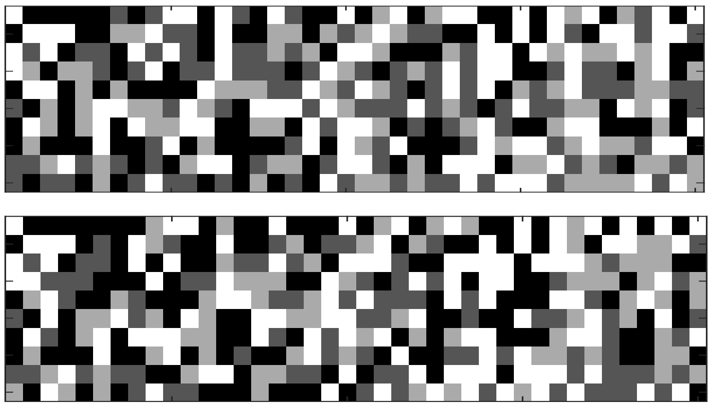
3.1.1 Universal RBM Training

To extract the desired RBM vector, the first step is to train a global or universal model with a large amount of available background speakers' utterances. This global model is referred to as URBM. Figure 3.2 shows a visualization of the connection weights of the URBM (at the top of the figure) and two randomly selected adapted RBMs (at the bottom). The URBM is supposed to convey speaker-independent information from the background data. The URBM is trained as a single RBM model with the feature vectors extracted from all the background speakers' utterances. The features are usually real valued vectors, e.g., the Mel-Frequency Cepstral Coefficients (MFCC) in our case. Thus, for the real valued input feature vectors we have used Gaussian real-valued units for the visible layer of the RBM (Hinton, 2012). The training was performed using the Contrastive Divergence-1 (CD-1) algorithm (Hinton & Salakhutdinov, 2006; Hinton et al., 2006) assuming that the inputs have zero mean and unit variance. Thus, the feature vectors are Mean Variance Normalized (MVN) before the RBM training. Finally, the universal model is trained with a large number of training samples generated from the feature vectors of the background speakers' utterances. This universal model is supposed to learn both speaker and session variabilities from the large amount of background speakers (Safari et al., 2016).

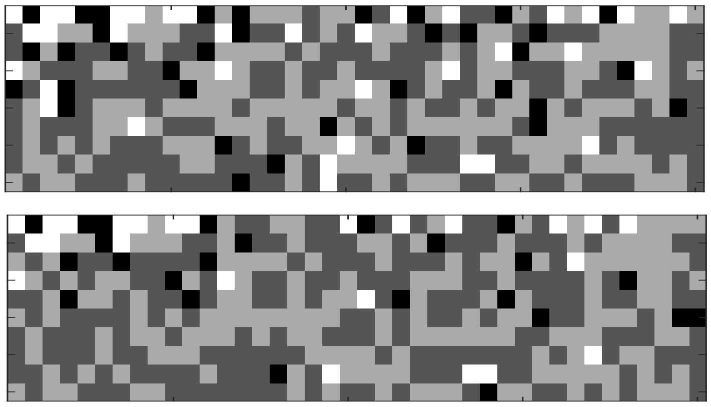
3.1.2 Adaptation and RBM Vector Extraction

After the URBM training, we perform speaker adaptation for every test speaker. The adapted RBM model is trained only with the data of the corresponding speaker, in order to capture speaker-specific information. In this step, the RBM model of the test speaker is initialized with the parameters (weights and biases) of the URBM. In other words, the adaptation step drives the URBM model in a speaker-specific direction. This kind of adaptation technique is successfully applied in (Ghahabi & Hernando, 2017; Safari et al., 2015; Ghahabi & Hernando, 2014a,b). The adaptation is also carried out by the CD-1 algorithm. As we have only one weight matrix in RBM, all the information learned by the adapted RBM is preserved in it's weight matrix and it is supposed to convey speaker-specific information of the corresponding speaker. Figure 3.2 shows the visualization of the connection weights of the URBM (at the top of the figure) and of two randomly selected speakers (Speaker 1 and Speaker 2, at the bottom of the figure). From the figure, it is clear that during the adaptation the URBM weights are driven in speaker-specific direction.

An RBM model is assigned to each test speaker after the adaptation step. The visible to hidden weight matrices along with their corresponding bias vectors of the adapted RBMs are concatenated in order to generate a higher dimensional speaker vector. These are referred to as RBM supervectors. After this, a PCA whitening with dimensionality reduction is applied to the RBM supervectors in order to generate the lower dimensional RBM vectors. The PCA whitening transforms the original data to the principal component space which de-correlates the data components. The PCA is trained with the RBM supervectors extracted from the background



(a) Speaker 1



(b) Speaker 2

Figure 3.3: Examples of 400-dimensional RBM vectors. The figure shows two pairs of RBM vectors from the test audios. Each pair belong to the same speaker. We rearrange the RBM vectors in the form 10×40 for the convenience of visualization. The ordering of the RBM vector is the same for all.

speakers' utterances and is applied to the RBM supervectors of the test speakers. All the RBM supervectors are mean-normalized before subjecting to PCA whitening and dimensionality reduction. The extracted RBM vectors are supposed to convey enough speaker-specific information, which can discriminate different speakers. Figure 3.3 shows a visualization of a pair of RBM vectors (top and bottom) extracted from different utterances of two different speakers randomly selected from the test audios. From the Figure, it is clear that the two RBM vectors extracted for Speaker 1 look similar but are different from those extracted for Speaker 2. Similarly, the two RBM vectors extracted for Speaker 2 look similar but are different from those extracted for Speaker 1.

In (Safari et al., 2016), it has been shown that the RBM vector extracted in this way is successful in learning speaker-specific information in a speaker verification task. In this chapter, we make an effort to make use of the RBM vector in the tasks of speaker clustering and speaker tracking.

3.2 Speaker Clustering

Speaker clustering refers to the task of grouping speech segments in order to have segments from the same speaker in the same group. Ideally each group or cluster must contain speech segments that belong to the same speaker. On the other hand, utterances from the same speakers must not be distributed among multiple clusters. Several approaches to speaker clustering tasks exist, for example, cost optimization, sequential and AHC (Sayoud & Ouamour, 2010; Siegler et al., 1997; Ghaemmaghami et al., 2016; Tranter & Reynolds, 2006). Some approaches rely on commonly used statistical speaker modeling like GMM, while others use features extracted using DNN. For example, in (Jorrín et al., 2017), BNF extracted from different DNNs are used for speaker clustering using an AHC approach.

In order to evaluate the effect of RBM vectors in a speaker clustering task, we considered the conventional bottom-up AHC clustering system with the options of single and average linkages. We did not consider the model retraining approach because it is costly in terms of computations as compared to the linkage approaches to clustering (Ghaemmaghami et al., 2016). The system starts with an initial number of clusters equal to the total number of speech segments. Iteratively, the segments that are more likely to be from the same speaker are clustered together until a stopping criterion has reached. The stopping criterion can be thresholding the score in order to decide to merge clusters or it can be a desired (known) number of clusters achieved. Our clustering algorithm is based on computing a similarity matrix $M(X)$ between all the speakers' segments where X is the set of segments to be clustered. Hence, the RBM vectors of all the segments are extracted, the matrix $M(X)$ is computed by scoring all the RBM vectors against all. Thus, for N RBM vectors, the matrix $M(X)$ has dimensions $N \times N$. In every iteration, the segments with maximum similarity scores are clustered together and the matrix $M(X)$ is updated. The corresponding rows and columns of the clustered segments are removed from $M(X)$ and a new row and column are added. The new row and column contain the distance scores between the new and old clusters. The new scores are computed according to the linkage algorithm used. For example, segments S_a and S_b are clustered in S_{ab}. Then the scores between new cluster (S_{ab}) and old segment (S_n) are computed as follows:

(a) Average Linkage:

$$s(S_{ab}, S_n) = \frac{1}{2}\{s(S_a, S_n) + s(S_b, S_n)\} \tag{3.1}$$

(b) Single Linkage:

$$s(S_{ab}, S_n) = max\{s(S_a, S_n), s(S_b, S_n)\} \tag{3.2}$$

where $s(S_{ab}, S_n)$ is the score between new cluster S_{ab} and old segment S_n while $s(S_a, S_n)$ is the score between old segments S_a and S_n.

In this way, the process is iterated until a stopping criterion is met. There are two methods to control the iterations: (1) to fix a threshold and (2) to add an additional information to the system about the desired (known) number of clusters. The system stops when this number is reached. In this work, we did not let the system know any desired number of clusters and we have used the thresholding method. We have tuned a threshold in order to see the performance of the system at different possible working points. The system performance is measured with respect to a ground truth cluster label.

3.3 Speaker Tracking

In certain applications, a person of interest (target) is tracked in an audio file, by using his voice characteristics. To identify *when the target speaker speaks* in the audio, is a speaker tracking task (Luque, 2012). The main stages in speaker tracking are to determine the positions in the audio where the speaker changes occur, that is, speaker segmentation, and to then identify the speaker, that is, speaker identification. Based on these two stages, there can be a joint or a separate approach to a speaker tracking task. In the past, several approaches to speaker tracking were proposed based on different segmentation and speaker modeling strategies. In order to detect speaker changes, a fixed length window is slid over the audio and a distance metric is computed between consecutive windows. This distance is set to a threshold to decide whether there exists a speaker change. In (Bonastre et al., 2000),

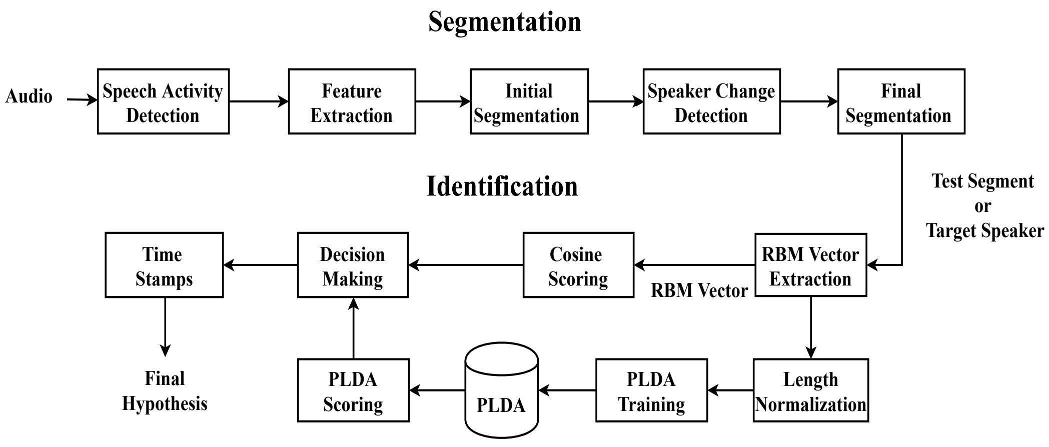


Figure 3.4: Architecture of Our Two Step Speaker Tracking System using RBM vectors.

speaker change detection is performed using a Generalized Likelihood Ratio (GLR) (Gish et al., 1991; Gish & Schmidt, 1994). In (Lu & Zhang, 2002) the Divergence Shape distance, described in (Lu et al., 2001; J. P. Campbell, 1997), is computed for speaker change detection. After this, the audio is segmented using the speaker change detection. The conventional GMMs are trained to represent the speakers and a speaker identification is performed in order to decide which segment belongs to which speaker among the set of given target speakers.

In this thesis, we implemented a two stage speaker tracking system, that is, speaker segmentation and speaker identification as in (Khan, 2016). Figure 3.4 shows the basic steps of speaker segmentation, RBM vector extraction and identification. First of all, the audio is segmented according to the speaker change points. The speaker change points are detected using the sliding window and searching for speaker change approach. A fixed length window is slid over the audio with a very small shift and speaker change is detected using some distance metric. We have used the Divergence Shape distance as a distance metric in this work. The distance is thresholded in order to decide if the neighboring windows are spoken by the same speaker or whether there exists a speaker change. As a result of these speaker change points, the audio is segmented. In the next stage, a speaker identification of the target speakers against the segments is performed in order to know *'to which target speaker the corresponding segment belongs?'* All the target speakers and segments are transformed into a vector based representation by means of RBMs, i.e., RBM vectors. These RBM vectors are scored using cosine and PLDA scoring methods. In the following sections, the two stages of our speaker tracking system are discussed in detail.

Segmentation

As shown in Figure 3.4, first an energy-based Speech Activity Detection (SAD) is performed on the audio. Then, the speech parts are segmented into small segments of d seconds with an overlap of $(d - \Delta)$ seconds, where Δ is the shift. This is referred to as *initial segmentation* in the segmentation part of Figure 3.4. The segments generated in this step are reffered to as *small segments*. The shift Δ defines the resolution of speaker change detection. Then, Mel-Frequency Cepstral Coefficients (MFCC) features are extracted for every *small segment*. In order to detect speaker change points, the Divergence Shape distance is computed between every adjacent *small segments* and is thresholded. We compute the Divergence Shape distance as in (Lu & Zhang, 2002; J. P. Campbell, 1997), using the following simplified expression:

$$D = \frac{1}{2} tr[(C_i - C_j)(C_j^{-1} - C_i^{-1})] \tag{3.3}$$

where tr is the trace function that sums the diagonal elements of a matrix, C_i is the covariance of the features from *small segment* S_i and C_j is the covariance of the features from *small segment* S_j. A speaker change point is marked if the distance at that point is greater than the distances at the two neighboring points (one before and one after) and a threshold at that point. For example, a speaker change point at *small segment* S_i occurs if:

$$D(i, i+1) > D(i, i+2) \tag{3.4}$$

and

$$D(i, i+1) > D(i-1, i) \tag{3.5}$$

and

$$D(i, i+1) > Threshold_i \tag{3.6}$$

where $D(i, i+1)$, $D(i, i+2)$ and $D(i-1, i)$ are the Divergence Shape distances of *small segment* S_i to S_{i+1}, S_i to S_{i+2} and S_{i-1} to S_i, respectively. $Threshold_i$ is an adaptive threshold which is computed for every *small segment* and is defined in (Lu & Zhang, 2002) as:

$$Threshold_i = \frac{\alpha}{N} \sum_{n=0}^{N} D(i-n-1, i-n) \tag{3.7}$$

where α is a scaling factor and needs to be tuned experimentally. We have evaluated the segmentation with different values of α which we will discuss in Section 3.4. In Equation (3.7), N is the number of previous distances used for predicting the threshold. Once we detect the speaker change points by using this method, we segment the audio on these points. The segments generated will be used in the next step, that is, speaker identification. It is worth noting that we did not perform any refining algorithm for the speaker change points. Rather, we fixed the value of α so as to minimize the Miss Detection error in order not to miss a speaker change. This is because a False Alarm error can possibly be corrected in the speaker identification stage but a Miss Detection error cannot be corrected.

Identification

The second stage of our speaker tracking system performs a conventional speaker identification test on the segments and target speakers as shown in Figure 3.4. The goal is to answer *to which target speaker, the segments belong?* We propose the use of RBM vector representation for both the target speakers and segments generated in the segmentation stage. The MFCC features are extracted both for target speakers and segments. Then, RBM vectors are extracted and all the segments are tested against target speakers using cosine and PLDA scoring. Assume that $S_{Tm,Sn}$ represents the cosine/PLDA score for testing the target speaker T_m against the segment S_n. For a segment under test, first we select a potential candidate among all the target speakers. The target speaker with the maximum score is a potential candidate for the segment. Then, if the maximum score is greater than a threshold, the identity of that target speaker is assigned to that segment. Generally, the identity of the target T_m is assigned to the segment S_n according to:

$$Id_{S_n} = \arg\max_m (S_{Tm,Sn}) \quad \text{if} \quad max(S_{Tm,Sn}) > \lambda \tag{3.8}$$

where λ is a threshold to decide whether the segment under test does not belong to any of the target speakers. If the score is less than λ, the segment is not assigned to any of the target speakers. This is reflected as a Missed Speaker Time (MST) error for the target speaker which the segment actually belongs to. There are no speakers that should be rejected by the system because we consider all the speakers as possible target speakers.

3.4 Experimental Setup and Database

The experiments are performed on the AGORA database, which contains audio recordings of 34 TV shows from Catalan broadcast TV3 (Schulz & Fonollosa, 2009) (in total 68 audios of approximately 38 minutes each). These audios contain segments from 871 adult Catalan and 157 adult Spanish speakers. For all the experiments in this work, we selected 38 audio files for testing and 30 audios are used as background data. The background data were used to train the Universal Background Model (UBM) and Total Variability (TV) matrix for the baseline i-vector system. For the proposed system, the background data were used to train the URBM and PCA.

We manually extracted 2631 speaker segments from the test audios, according to ground truth rich transcription. These segments were used in the speaker clustering experiments. In the testing audios, 414 different speakers appear which were used as target speakers for the tracking experiments. For an audio file, all the speakers are considered as possible target speakers. A priori knowledge is required to enroll the target speakers in the system. Thus, the target speakers are enrolled using i-vectors and RBM vector approaches for the baseline and proposed systems, respectively. The target speakers are enrolled with 30 s of utterances. These enrollment utterances of target speakers are manually selected from the corresponding audio file (in which they appear) according to the ground truth rich transcription. It is worth noting that each target speaker appears in at least one of the test segments.

For all the experiments, 20 dimensional MFCC features were extracted, for both the baseline and proposed systems, using a Hamming window of 25 ms with 10 ms shift. A 512 component UBM was trained to extract i-vectors for the baseline system and the PLDA was trained with the background i-vectors. A more recent and competitive features could have been used, for example the BNF. These features (either in the baseline or in the proposed approach) would require a huge amount of labeled background data (for example phonetic labels). On the other hand, MFCC features do not require labeled data for training our models. This is the strength of our proposed RBM vectors, which were trained in a completely unsupervised manner. The UBM training, TV matrix and i-vector extraction were carried out using Alize, a free open source toolkit (Larcher et al., 2013).

For the proposed system, more than 3000 speaker segments were extracted from the background audios according to the ground truth rich transcription. For each segment, the features of 4 neighboring frames were concatenated in order to generate 80-dimensional feature inputs to the RBMs. With a shift of one frame, we generated almost 10 million samples for the URBM training. All the RBMs used in this paper consisted of 80 visible and 400 hidden units. The URBM was trained for 200 epochs with a learning rate of 0.0005, weight decay of 0.0002 and a batch size of 100. All the adapted RBM models for the segments and target speakers were trained with 200 epochs with a learning rate of 0.005, weight decay of 0.000002 and a batch size of 64.

The PCA was trained with the background RBM supervectors and was applied to the background RBM supervectors and test RBM supervectors, as discussed in Section 3.1.2. Finally, fixed dimensional RBM vectors were extracted for the test speakers that were used in the speaker tracking and clustering experiments. Different dimensions for the RBM vectors were evaluated in the experiments which is discussed in Section 3.5.

3.5 Results

3.5.1 Speaker Clustering

Different lengths for RBM vectors, as well as for i-vectors, were evaluated using cosine scoring and the average linkage clustering algorithm. The results are shown in the second column of Table 3.1. From the Table, it can be observed that if the dimension is increased, the performance is improved, both in case of i-vectors and RBM vectors, in terms of EI. However, in the case of i-vectors, the best choice is 800 dimension. In case of RBM vectors, the 2000 dimensional RBM vectors perform better than the others. In this case, a relative improvement of 11% is achieved compared to 800 dimensional i-vectors. A further increase in the length of RBM vectors beyond 2000 degrades the performance in terms of EI.

The third column of Table 3.1 compares the performance of the RBM vector with the baseline i-vectors in the case of the single linkage algorithm for clustering

Table 3.1: Comparison of speaker clustering results for the proposed RBM vectors with i-vectors, in terms of EI in %. The dimensions of vectors are given in parenthesis. Each column shows EI in % for different scoring and linkage combinations.

Approach	**Cosine (Average)**	**Cosine (Single)**	**PLDA (Single)**
i-vector(400)	49.19	46.26	36.16
i-vector(800)	46.66	42.19	35.91
i-vector(2000)	46.79	42.83	35.89
RBM vector(400)	51.36	39.66	37.36
RBM vector(800)	47.20	40.02	32.36
RBM vector(2000)	41.53	37.14	31.68

using cosine scoring. From the table it is seen that single linkage was a better choice for our experiments. In this case, a minimum EI of 37.14% is obtained with 2000 dimensional RBM vectors which has a relative improvement of 12% over 800 dimensional i-vectors.

Finally, we evaluated the proposed system using PLDA scoring as well. The PLDA was trained using background RBM vectors for 15 iterations. The number of eigenvoices were set to 250, 450 and 500 for RBM vectors of dimensions 400, 800 and 2000, respectively. All the RBM vectors were subjected to length normalization prior to PLDA training. As per the previous results, we performed this experiment with the single linkage algorithm only. The results were compared with i-vectors in the fourth column of Table 3.1. It was observed that 800 and 2000 dimensional RBM vectors have a better EI compared to the respective similar dimensional i-vectors. In this case, the RBM vectors of dimension 2000 have a minimum EI of 31.68% which results in a relative improvement of 11% over the 800 dimensional i-vectors. However, in the case of 400 dimensions, the i-vectors outperform RBM vectors.

The Impurity Trade-off (IT) curves for the baseline, as well as the proposed system, are shown in Figure 3.5. The figure shows the evaluation of different dimensions of i-vectors and RBM vectors in the average linkage clustering using cosine scoring. It can be seen that RBM vectors of length 2000 gives a better performance than 800 dimensional i-vectors at all working points. On the other hand, RBM vectors of dimensions 400, 800, 2400 and 3000 perform worse than i-vectors. It is

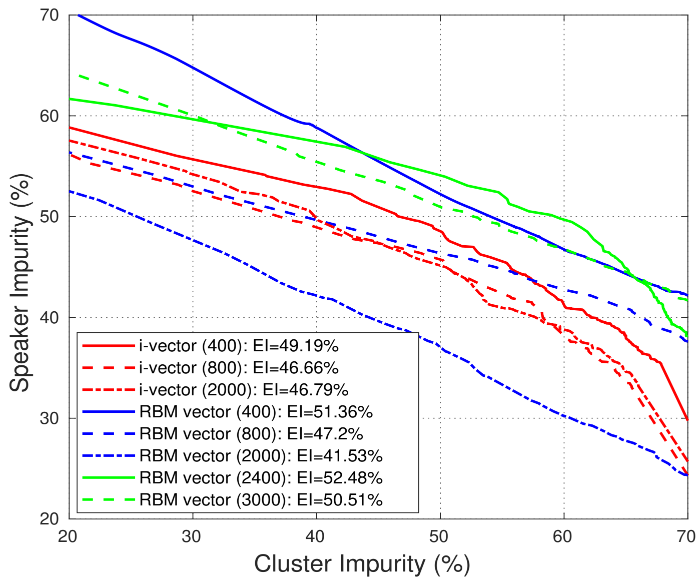


Figure 3.5: IT curves for the proposed RBM vectors with i-vectors using cosine scoring and average linkage algorithms for clustering.

observed that 400 and 800 dimensional RBM vectors could not capture enough information about the speaker while 2400 and 3000 dimensional RBM vectors include unnecessary information which degrades the performance.

In Figure 3.6 we show a comparison of 2000 dimensional RBM vectors with 800 dimensional i-vectors using both cosine and PLDA scoring with the single linkage algorithm for clustering. The choices of dimensions are based on the previous experiments as 2000 dimensional RBM vectors and 800 dimensional i-vectors give the best results with cosine scoring and average linkage. From Figure 3.6, it can be seen that the RBM vectors perform better at all working points as compared to i-vectors using their respective cosine and PLDA scoring. However, at low Speaker Impurity regions, the RBM vector with cosine scoring outperforms the baseline i-vector with PLDA scoring. Overall, the 2000 dimensional RBM vector has a consistent improved performance compared to i-vectors.

3.5.2 Speaker Tracking

For speaker change detection and segmentation, 20 MFCC features were extracted for all the *small segments* using a Hamming window of 25 ms with 10 ms shift.

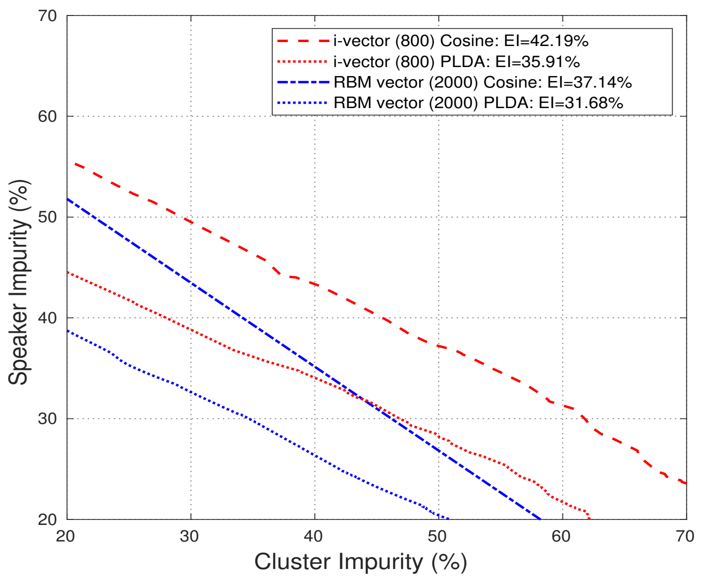


Figure 3.6: IT curves for the proposed RBM vectors with i-vectors using different scoring and linkage algorithms for clustering.

We performed segmentation using different sizes of small segments, that is, the d parameter discussed in Section 3.3 was equal to 2, 2.5 and 3 s. The value of Δ was set to 0.25 s. The speech parts smaller than d were not considered in these experiments and were simply discarded. Figure 3.7 shows the graph of FAR against MDR for different values of d and α. The results were computed, accepting a tolerance (collar) of ± 0.25 s in the position of detected speaker change points. We experimented with different values of d in order to see the behaviour at different working points, that is, $d = 2$, 2.5 and 3 s. Then, we experimented with different values of α and the results are plotted in Figure 3.7. From the Figure it is clear that the best choice for d is a 3 s window.

Our actual working point is marked as a black circle which is obtained for $\alpha = 2$ (in Equation (3.7)). We performed the *final segmentation* at this point which has less MDR as compared to FAR. At this point, a FAR of 10% and MDR of 7.8% are achieved. There is a trade-off between the two metrics (FAR and MDR). One can decrease one of the metrics at the cost of increasing the other.

The segments generated in the speaker segmentation were then tested against the target speakers for the tracking task. Table 3.2 shows the results of speaker

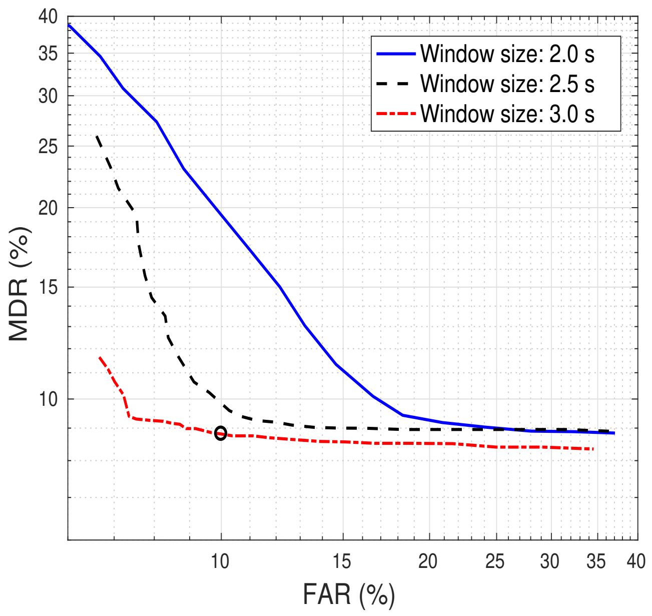


Figure 3.7: Speaker segmentation results in terms of FAR and MDR in %. Results are obtained using different Window sizes (d) and a constant shift i.e., $\Delta = 0.25$ s. A collar of ± 0.25 s is accepted around a speaker change point.

Table 3.2: Comparison of speaker tracking results for the proposed RBM vectors with 800 dimensional i-vectors, in terms of EER in %. The lengths of RBM vectors and i-vectors are given in parenthesis. Column 2 and 3 represents EER in % for cosine and PLDA Scoring respectively.

Approach	**EER % (Cosine)**	**EER % (PLDA)**
i-vector (800)	3.74	2.97
RBM vector (600)	4.33	3.57
RBM vector (800)	4.03	3.47
RBM vector (2000)	3.30	2.74

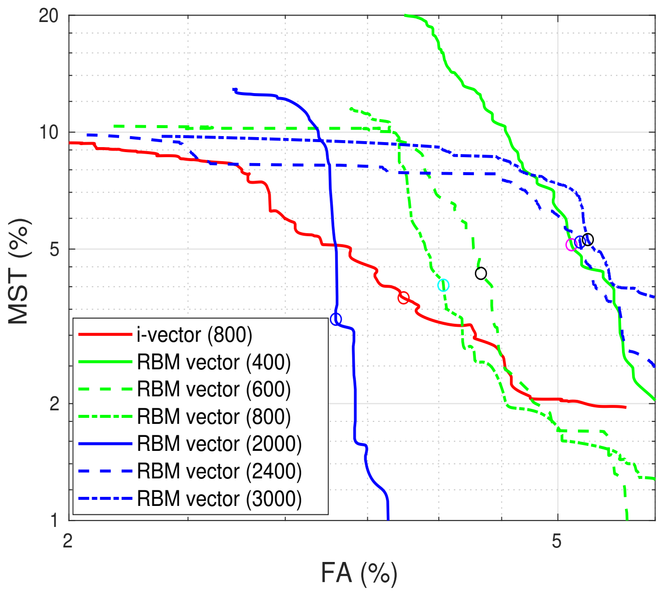
tracking for different lengths of RBM vector in terms of Equal Error Rate (EER). In this case EER was the coinciding point between FA and MST. The second column of Table 3.2 shows the comparison of RBM vector with the baseline i-vectors using cosine scoring. We fixed the length of i-vectors to 800 as a conclusion of the speaker clustering experiments. It is observed that, as the length of the RBM vector is increased, the performance is improved. The best EER of 3.30% was obtained using

2000 dimensional RBM vector, which gained a relative improvement of 11.76% as compared to the baseline 800 dimensional i-vectors. Increasing the dimensions of the RBM vectors does not affect the computational costs of training the models. The dimensions of RBM vectors are only controlled by the number of components while applying PCA to the RBM supervectors, as discussed in Section 3.1.2

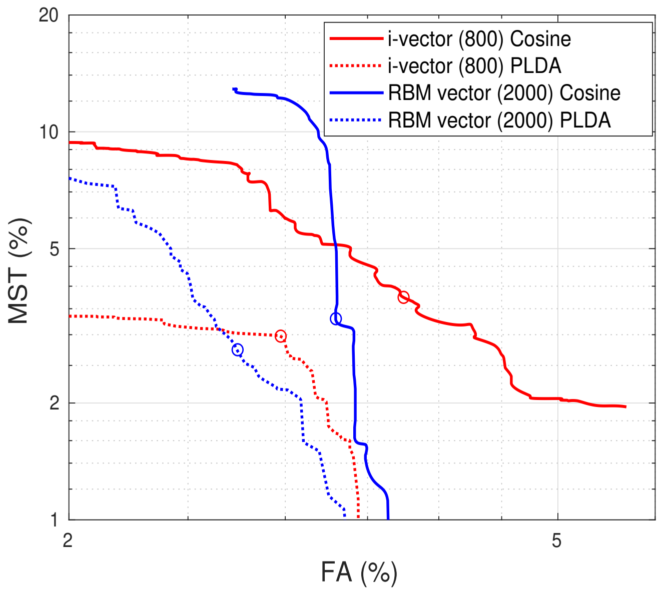
The third column of Table 3.2 shows the comparison of the RBM vector with the baseline i-vectors using the PLDA scoring method. For the RBM vector/PLDA framework, the PLDA is trained using the background RBM vectors for 15 iterations. The number of eigenvoices are set to 350, 450 and 500 for RBM vectors of lengths 600, 800 and 2000 respectively. All the RBM vectors are subjected to length normalization prior to PLDA training. From the table, it is clear that the 2000 dimensional RBM vector/PLDA system outperforms the 800 dimensional i-vector/PLDA system by a relative improvement of 7.74%. In the case of PLDA post processing, increasing the dimensions of the RBM vectors will increase the computational costs of PLDA training. This is because the PLDA model is trained on higher dimensional background RBM vectors.

Figure 3.8 shows the comparison of Detection Error Trade-off (DET) curves for the baseline as well as the proposed system. These graphs are obtained by tuning the λ parameter in Equation (3.8). In Figure 3.8a we have evaluated different lengths of RBM vectors by comparing with i-vectors using cosine scoring. It can be observed that RBM vector of lengths 800 and 2000 give a better performance than the baseline i-vectors at low MST points only. An RBM vector of length 600 can be comparable with baseline i-vectors in this region. On the other hand, at low FA points the baseline i-vectors outperform RBM vectors of either length. However, at very few working points in low FA region, the RBM vector of length 2400 can be comparable with the baseline i-vectors.

In Figure 3.8b we have shown a comparison of 2000 dimensional RBM vector (which gives the best results with the cosine scoring method) with baseline i-vectors using both cosine and PLDA scoring. From the figure, a similar kind of behavior is observed for RBM vectors using PLDA scoring as well. It can be seen that the 2000 dimensional RBM vector outperforms the baseline i-vectors in low MST regions using both cosine and PLDA scoring. However, in the low FA regions, the i-vector/PLDA framework still performs better which was also the case using the cosine scoring method.



(a) Length selection using cosine scoring



(b) Best length using cosine/PLDA scoring

Figure 3.8: Comparison of DET curves for the proposed RBM vectors with 800 dimensional i-vectors. Different lengths of RBM vectors are evaluated using cosine and PLDA scoring. The lengths of RBM vectors and i-vectors are given in parenthesis.

3.6 Conclusion

RBM vectors were first proposed for speaker verification task in (Safari et al., 2016). In this chapter, we have proposed the use of RBM vectors for the tasks of speaker tracking and speaker clustering in TV broadcast shows. RBM is applied for learning a fixed dimensional vector representation of a speaker which is referred to as RBM vector. The visible to hidden weight matrices along with the bias vectors of the URBM-adapted models are concatenated to generate RBM supervectors. The RBM supervectors are further subjected to a PCA whitening with dimensionality reduction to extract the desired RBM vectors. These RBM vectors are used in the tasks of speaker clustering and speaker tracking.

For speaker clustering experiments, two linkage algorithms for an AHC approach were explored with RBM vectors scored using cosine and PLDA. Using cosine scoring, the performance of the proposed system was better for both the linkage algorithms as compared to i-vector based clustering. Overall, the single linkage algorithm with 2000 dimensional RBM vectors was the best choice for our experiments, using both cosine and PLDA scoring. For speaker tracking experiments, we performed speaker segmentation followed by a speaker identification. The RBM vectors were applied in the speaker identification stage. In general, the proposed system was more effective in low MST regions. The experimental results have shown that, in terms of EER, the proposed system outperformed the baseline i-vectors system using both cosine and PLDA scoring methods. We conclude that the RBM vectors can be successfully used as a speaker representation in speaker clustering and speaker tracking tasks.

Chapter 4

DNN speaker embeddings by means of autoencoder pre-training

Over the last years, the compact representation of speech utterances known as i-vector (Dehak et al., 2011) has been the state of the art approach in speaker recognition. Cosine scoring and Probabilistic Linear Discriminant Analysis (PLDA) (Prince & Elder, 2007; Kenny, 2010) backends are the two commonly used scoring techniques to decide if two i-vectors belong to the same speaker. Cosine scoring is a straight forward technique while the PLDA parameters are always trained using speaker-labeled background data. A PLDA backend for i-vectors leads to a superior performance as compared to cosine scoring technique. But in practice, it is difficult to access large amount of labeled background data for training. Therefore, in i-vector based speaker verification, the lack of speaker-labeled background data results in a huge performance gap between the unsupervised (cosine) and the supervised (PLDA) scoring techniques.

In this chapter, we propose to reduce the demand of labeled background data for i-vector based speaker verification. We propose to transform i-vectors into a new speaker vector representation, in order to increase their discriminative power. The goal is to reduce the performance gap between cosine and PLDA scoring techniques while limiting the use of speaker labels. The rest of the chapter is organized as follows. In section 4.1 the proposed autoencoder initialized DNN architecture is explained. In Section 4.2 the experimental setup and details of the database used are described. In 4.3 the experiments and results are discussed. Finally, in 4.4 some conclusions are drawn.

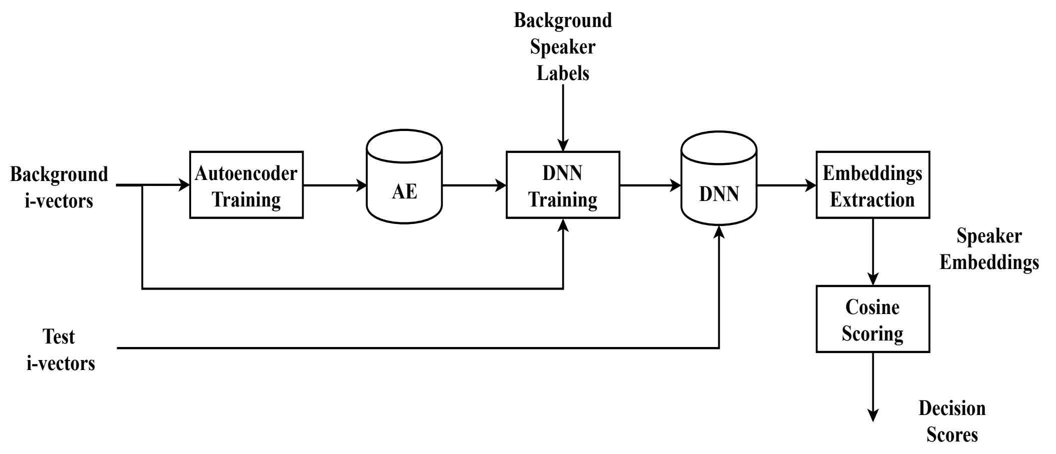


Figure 4.1: Block diagram of the proposed speaker embeddings extraction from i-vectors using autoencoder pre-training.

4.1 Autoencoder Pre-Training for DNN

We propose a new framework using autoencoder pre-training, to produce an alternative vector based representation of speakers (Khan & Hernando, 2019). Figure 4.1 shows the block diagram of our proposed system. In order to limit the need of large amount of labeled data, we train the autoencoder using a large amount of unlabeled data. Then, we train a DNN classifier using a relatively small amount of labeled data. We propose to initialize the DNN training with the weight matrices and bias vectors of the pre-trained autoencoder and a fine tuning is carried out using small labeled data. In this way, we train a hybrid autoencoder-DNN classifier. After the training, we extract speaker embeddings from i-vectors as the output form the second last layer of the supervised network. The goal is to improve the performance using fewer background speaker labels and take advantage of large amount of unlabeled data. We evaluated our system using the speaker verification trials of VoxCeleb-1 (Nagrani et al., 2017) database. The experimental results have shown that the proposed approach has improved the baseline system in two aspects. Firstly, the proposed speaker embeddings, with cosine scoring technique, has gained a relative improvement of 21%, in terms of Equal Error Rate (EER), over the baseline i-vector/PLDA system. Secondly, we have observed that the proposed hybrid autoencoder-DNN training converges faster as compared to the one without autoencoder pre-training.

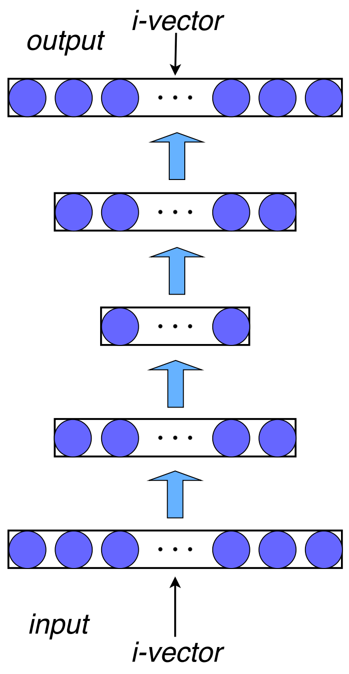


Figure 4.2: Autoencoder pre-training for the DNN.

Unsupervised Training

Figure 4.2 shows the architecture of the proposed autoencoder. The conventional architecture of an autoencoder consists of an *encoder* and a *decoder* as shown in the figure. The *encoder* is a function that encodes the input i-vector w into a shorter dimensional space, and the *decoder* is a function that decodes it back in order to reconstruct w. The conventional training is carried out by minimizing the Mean Square Error (MSE) between the input w and the reconstructed $\hat{w}$. Thus the loss function is :

$$Loss = MSE(\hat{w}, w) \tag{4.1}$$

Where $\hat{w} = decoder(encoder(w))$, and w is the input i-vector. We perform unsupervised autoencoder training, in order to make use of the large amount of unlabeled background i-vectors. The training is carried out in a conventional way i.e., minimizing the MSE loss between input and reconstructed i-vectors using the Stochastic Gradient Descent (SGD) optimizer. The autoencoder is supposed to learn speaker independent information from the background i-vectors as it has the ability to learn compact representation.

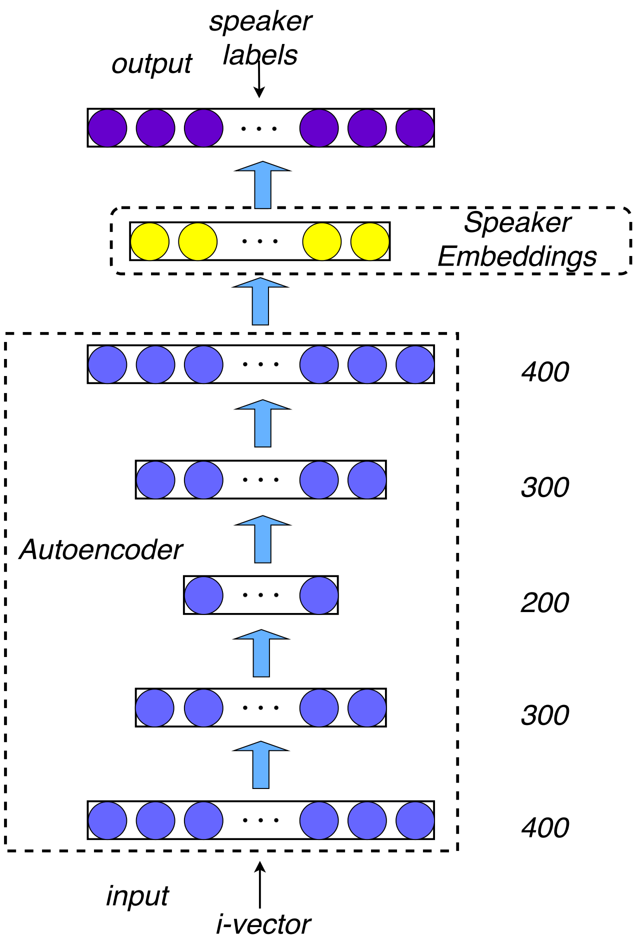


Figure 4.3: DNN training as a classifier.

Fine Tuning and Testing

Once the autoencoder is trained, we train a DNN classifier in a supervised way, in order to learn information about speaker classes. Figure 4.3 shows the architecture of the supervised DNN. The encoder part and decoder part are symmetric as of the autoencoder. The DNN has a similar structure except the fully connected and the output layers. We add a fully connected layer and a classification layer after the last layer of the autoencoder. The number of neurons in the fully connected layer is 600 while that of the classification layer is equal to the number of speaker classes. The fully connected layer of 600 neurons is used to extract speaker embeddings and is therefore named as speaker embedding layer in Figure 4.3. We feed speaker labels at the output of the classification layer and train the network in a supervised manner. Since it is a multi-class classification problem, we minimize categorical cross-entropy loss function. This network is referred to as hybrid autoencoder-DNN classifier and is trained using relatively smaller labeled i-vectors. We initialize the hybrid autoencoder-DNN with the weight matrices and bias vectors of the pre-trained autoencoder. This type of initialization has been applied for adapting the unsupervised model to learn speaker specific information in (Ghahabi & Hernando, 2014a; Safari et al., 2016). The autoencoder pre-training helps in the supervised learning, and the network converges relatively faster than without pre-training. This can result in a reduction in training time for a certain accuracy as compared to the conventional training.

There are two different scenarios in order to use the pre-trained autoencoder for the DNN initialization. One possibility is to add the fully connected layer and classification layer directly after the encoder part. The encoder part compresses the data into a shorter dimensional space which preserves enough information to reconstruct an approximation of the original data. However, this was not recommended in our experiments because in the hybrid autoencoder-DNN training the network learns additional information from the speaker labels at the output. The shorter dimensional space of the encoder part is not enough to learn efficient information from the higher dimensional classification layer.

Another scenario is to use the full autoencoder by adding the fully connected layer and classification layer at the end of the autoencoder. We prefer to expand the input data to its original dimensional space and then train the hybrid autoencoder-DNN classifier. In this way, firstly, we remove the unnecessary information from the data by encoding it into shorter dimensional space. Secondly, we expand the data back to its original dimensional space in order to ease the learning of additional information from the speaker labels. It is shown in the experiments that using only the encoder part could not show better performance as compared to the full autoencoder case.

Finally, after the supervised fine tuning, we extract the desired speaker embeddings from the output of the fully connected (embeddings) layer as shown in Figure 4.3. The test i-vectors, that are involved in the experimental trials, are propagated through the hybrid network in order to extract speaker embeddings from i-vectors. These speaker embeddings have shown to preserve speaker specific information and are more discriminative than the original i-vectors. Using these embeddings, we perform the trials of the experiments using cosine scoring technique.

4.2 Experimental Setup and Database

The experiments were performed on VoxCeleb-1 and VoxCeleb-2 databases (Nagrani et al., 2017; Chung et al., 2018) which contains 153,516 and 1,128,246 number of utterances, respectively. Both these databases are further partitioned into development and test sets. In this work, we have used the whole VoxCeleb-2 database (development and test) as unlabeled background data to train the autoencoder. The supervised training was carried out using the development partition of VoxCeleb-1 (the smaller database). VoxCeleb-1 is partitioned into 148,642 development and 4,874 test utterances, that belong to 1211 and 40 speakers, respectively. Thus, the classification layer in our hybrid autoencoder-DNN consists of 1211 number of neurons. From the test set of VoxCeleb-1, 37,720 experimental trials were scored. Half of them are client while the other half are impostor trials.

The development set of VoxCeleb-1 was used to train the Universal Background Model (UBM), the Total Variability (TV) matrix and the PLDA for the baseline i-vector/PLDA system. 20 dimensional Mel-Frequency Cepstral Coefficients (MFCC) features, appended by delta coefficients, were extracted for all the utterances. A 1024 components UBM is trained to extract 400 dimensional i-vectors. The PLDA for baseline was trained for 20 iterations and the number of eigenvoices was empirically set to 200. The UBM & TV matrix training and i-vector extraction process were carried out using Alize toolkit (Larcher et al., 2013).

The autoencoder used in this work consist of 3 hidden layers. The encoder and decoder parts are symmetrical. The hidden layer 1 and 3 have 300 neurons each, while hidden layer 2 consists of 200 neurons as shown in Figure 4.3. The input and output layers consist of 400 neurons each. In the DNN, the dimension of speaker embeddings layer was fixed to 600 while the classification layer consists of 1211 neurons. The autoencoder pre-training was carried out for 400 epochs. All the layers of the autoencoder used Rectified Linear Units (ReLU) activation function except the last layer which used linear activation. The learning rate was set to 0.03 with a decay of 0.0002 and the batch size was set to 100. The supervised DNN training was carried out for 200 epochs using Adagrad optimizer with an initial learning rate of 0.03 and a batch size of 100. Sigmoid activations were used for all the layers. Both the autoencoder and DNN were trained using Keras deep learning library (Chollet, 2015).

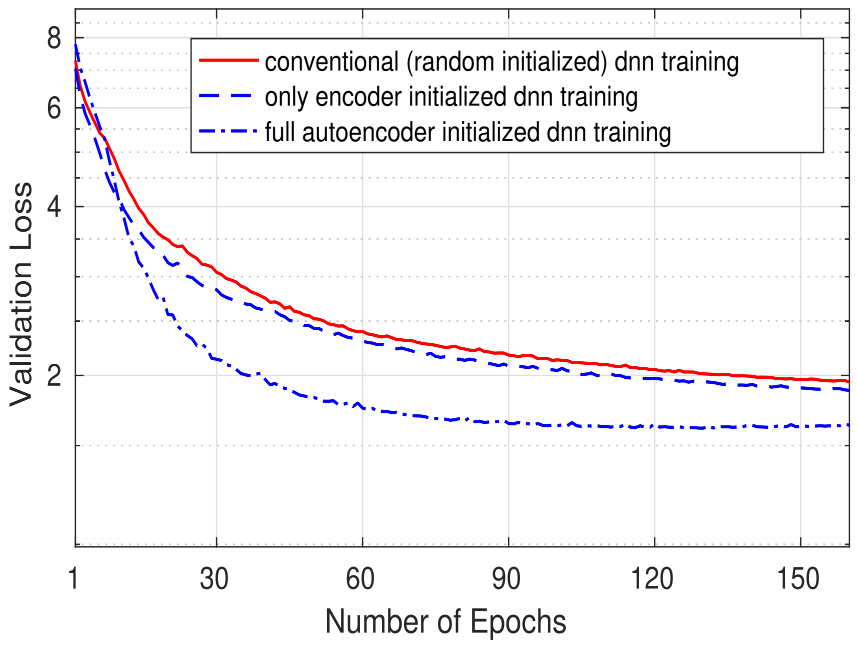


Figure 4.4: Comparison of the training convergence, in terms of validation loss against number of epochs, between the conventional and the proposed training of the DNN classifier.

4.3 Results

In the experiments it was observed that the autoencoder pre-training has learned speaker independent information from the large amount of unlabeled i-vectors. This information was utilized by the DNN classifier as it was initialized with the weights and biases of the autoencoder. This resulted in a significant improvement in the convergence of the DNN training. Figure 4.4 shows the comparison of the DNN training i.e., conventional DNN and both the cases of autoencoder pre-trained DNN. In this paper, conventional DNN refers to randomly initialized DNN. The plots were obtained using the validation loss only. It can be seen that if the DNN was trained using only the encoder, we have a slight improvement over randomly initialized DNN training. However the full autoencoder initialization is the best choice which helps in fast convergence of the training. For epochs greater than 10, the absolute value of the validation loss is reasonably lower as compared to the randomly initialized training. Furthermore, using the our proposed approach a validation loss achieved after only 40 epochs is achieved by the baseline after 100 epochs.

Table 4.1: Performance comparison, in terms of EER (%), between the baseline and the proposed speaker embeddings.

Approach	Scoring	EER(%)
i-vector	Cosine	17.61
i-vector	PLDA	9.54
only-encoder-dnn	Cosine	12.73
conventional-dnn	Cosine	8.58
full-autoencoder-dnn	Cosine	**7.51**

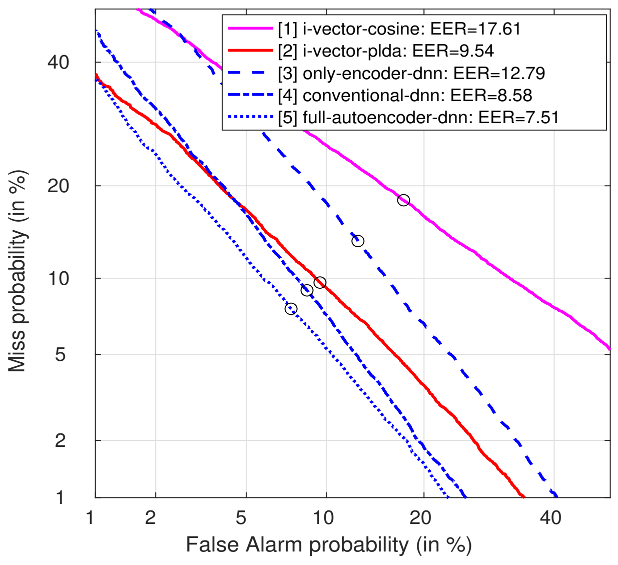


Figure 4.5: DET plots of the baseline and the proposed speaker embeddings, using only encoder and full autoencoder initialization for the DNN.

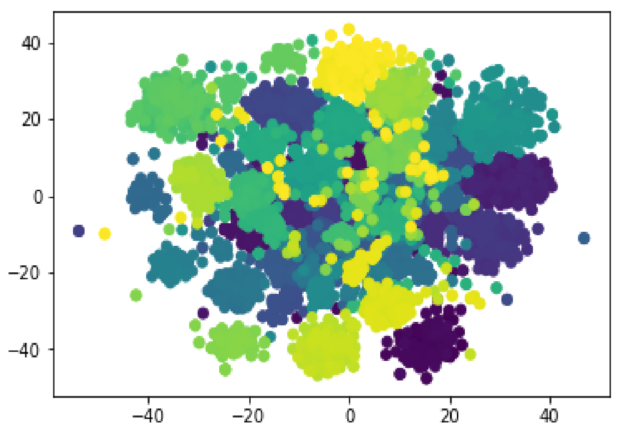
After extracting the desired speaker embeddings from i-vectors using the proposed approach, we score the experimental trials using cosine scoring technique. However the baseline i-vectors were scored using PLDA as well. Table 4.1 compares the performance of our proposed speaker embeddings with the conventional DNN speaker embeddings and i-vectors. Using only the encoder part to train the DNN, is not the preferred choice for our experiments. The full autoencoder initialization has an advantage of learning efficient information from the speaker labels as compared to the the only encoder case. From the table it is clear that our proposed speaker embeddings has outperformed both the other systems. The relative improvement between the proposed speaker embeddings and i-vector/PLDA is 21.28%, in terms of EER. If we compare the proposed speaker embeddings with the conventional DNN speaker embeddings, the relative improvement is 12.47%.

Figure 4.5 shows the Detection Error Trade-off (DET) curves of our proposed approach and the other two approaches. We can see that the conventional DNN speaker embeddings perform worse than i-vector/PLDA, at low False Acceptance Rate (FAR) regions. However, the DET plot for the proposed speaker embeddings shows better performance at all working regions.

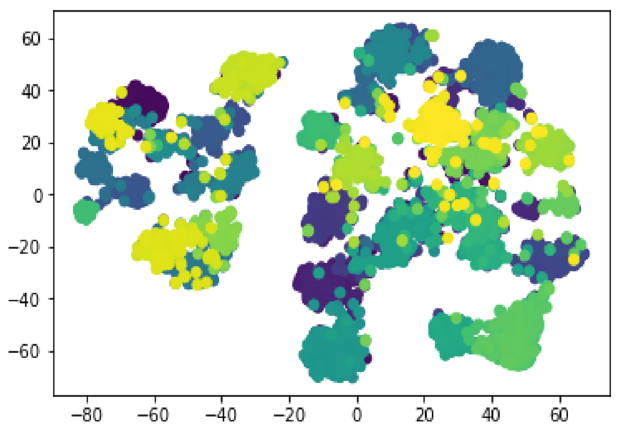
Finally, in Figure 4.6, we have shown the t-Distributed Stochastic Neighbor Embedding (t-SNE) plots for three different vector representation of speakers i.e., i-vectors, conventional DNN speaker embeddings and the proposed speaker embeddings. t-SNE is a dimensionality reduction technique to graphically visualize (Maaten & Hinton, 2008) higher dimensional vectors. In order to see the discriminative power of our proposed speaker embeddings, we have compared the t-SNE plots with the other two approaches. The plots were obtained using the test partition of the VoxCeleb-1 database. The dimensions of all the vectors was reduced to 2 for plotting the t-SNE. It is clear from the figure that our proposed speaker embeddings has the highest discrimination power as compared to the other two. The clusters generated are mostly pure and distinct. However, for the conventional DNN speaker embeddings, some of the clusters are overlapping with the others. For the baseline system, raw i-vectors were used to obtain the plots. The clusters formed for i-vectors are not very clear as compared to the former two DNN based speaker embeddings.

4.4 Conclusion

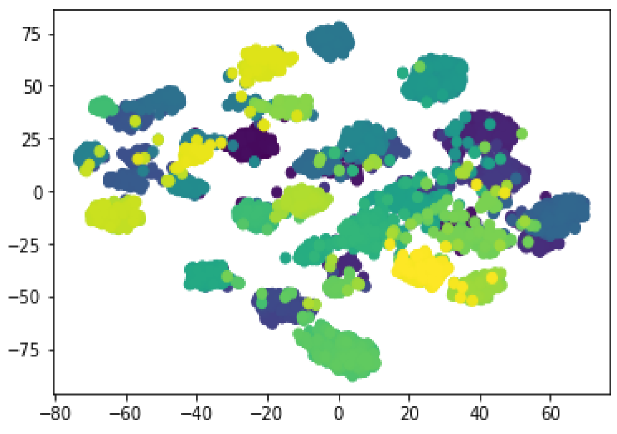
The requirements of large amount of labeled background data has put a constraint on deep learning approaches to speaker recognition. In practical scenarios large amount of labeled background data is not easily available. Thus we make use of unlabeled data to minimize the impact of lack of labeled data. In this chapter we proposed autoencoder pre-training for DNN training (Khan & Hernando, 2019). An autoencoder was pre-trained on a large amount of unlabeled background data which



(a) Raw i-vectors



(b) Conventional DNN speaker embeddings



(c) Proposed speaker embeddings

Figure 4.6: Comparison of the t-SNE Plots, between raw i-vectors, conventional and proposed speaker embeddings. All the vectors were reduced to 2 dimensional space.

learns speaker independent information. Then a DNN classifier was trained using a relatively small labeled data, initialized with the parameters of the pre-trained autoencoder. The objective was to fill the performance gap between the cosine and the PLDA scoring techniques when limited labeled background data is available.

The evaluation was performed on the speaker verification trials of VoxCeleb-1 database. The results have shown that by using autoencoder pre-training for DNN, we gain a relative improvement of 21% in terms of EER, over the baseline i-vectors/PLDA system. Furthermore, we have observed that the DNN training converged faster, compared to the conventional (randomly initialized) dnn case (Khan & Hernando, 2019).

Chapter 5

Unsupervised training of autoencoder speaker embeddings using k-nearest neighbors

Deep Learning (DL) approaches are applicable to learn a compact representation of speech utterance, which is commonly referred to as speaker embedding, such as in (Variani et al., 2014b; Rouvier et al., 2015; Snyder et al., 2018; Bhattacharya et al., 2017; Okabe et al., 2018; Novoselov et al., 2018). Speaker embeddings are the most effective speaker vector representation which is commonly used these days. They are usually extracted from an intermediate layer of a Deep Neural Network (DNN) which is trained as a classifier. Therefore, these DL approaches are typically constrained to speaker labeled background data. In the previous chapter we proposed DNN speaker embeddings which were extracted using a supervised DNN training. The DNN was initialized using the parameters of a pre-trained autoencoder. Although, the autoencoder pre-training was carried out in an unsupervised manner, the DNN training still required speaker labels for the background i-vectors.

In this chapter, we put an effort to completely avoid the use of labeled background data for i-vector based speaker verification. Unlike the conventional PLDA backend for i-vectors, and DNN based classifiers, autoencoder training is an unsupervised process which does not require labeled data. We propose to train an autoencoder in a new framework, in order to compensate session variability among

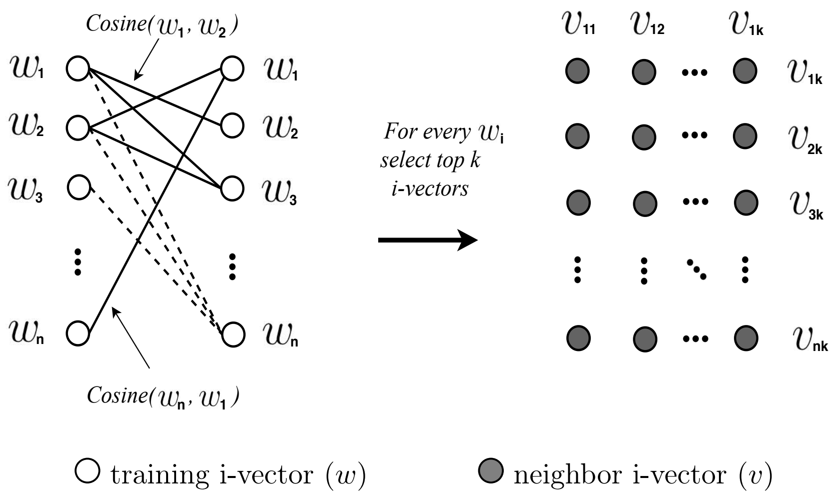


Figure 5.1: Visualization of selection of neighbor i-vectors.

i-vectors when no labeled background data is available. We propose to take advantage of nearest neighbor i-vectors in order to train the network. For all the training data we select neighbor i-vectors which are used to train the autoencoder. After training, we transform i-vectors into a new speaker vector representation, in order to increase their discriminative power. The goal is to reduce the performance gap between cosine and PLDA scoring techniques without using speaker labels. The rest of the chapter is organized as follows. In section 5.1 the proposed unsupervised selection of nearest neighbor i-vectors is explained. In Section 5.2 and 5.3 we explain the two proposed nearest neighbor autoencoders and the corresponding i-vector transformation. In Section 5.4 the experimental setup and details of the database used are described. In 5.5 the experiments and results are discussed. Finally, in 5.6 some conclusions are drawn.

5.1 Selection of Nearest Neighbor i-Vectors

All the training i-vectors are scored among each other using cosine scoring technique. For every i-vector, we select a set of similar i-vectors as neighbors. A straightforward approach to select the neighbor i-vectors is to apply a *threshold* to the cosine scores. The i-vectors with scores higher than the *threshold* are selected as neighbor i-vectors. Another approach is to select a constant k number of neighbor i-vectors for every training i-vector. The values of *threshold* and k are determined experimentally and will be discussed in section 5.5.

Algorithm 5.1: Proposed neighbor i-vectors selection algorithm for a constant k, regardless of the scores.

```
Input  : Training i-vectors $w_i$, $1 \leq i \leq n$
Output: Neighbor i-vectors $v_{ij}$, $1 \leq i \leq n$ & $1 \leq j \leq k$
1 for each training i-vector $w_i$ do
2     for each training i-vector $w_t$, $1 \leq t \leq n$ do
3         if $i \neq t$ then
4             Compute $score_{i,t} = cosine(w_i, w_t)$
5         end
6     end
7     From $score_{i,t}$, select $k$ highest scores as $S_k$.
8 end
```

Figure 5.1 shows a visualization of the selection process of the neighbor i-vectors for a constant k number of neighbors. Suppose w_i is a training i-vector, where $i = (1, \ldots, n)$ and n is the total number of training i-vectors. First, we score all the training i-vectors among each other using cosine scoring technique. Then, we select the top k i-vectors with highest scores as neighbor i-vectors for every w_i. The selected neighbor i-vectors are denoted by v_{ij}, where v_{ij} is the j^{th} neighbor of i^{th} training i-vector. Algorithm 5.1 summarizes how the selection of the neighbor i-vectors is carried out for a constant k number of neighbors, regardless of scores.

5.2 Nearest Neighbor Autoencoder

In the first proposal, we train an autoencoder to reconstruct neighbor i-vectors, rather than to reconstruct the same training i-vectors. In this way, the autoencoder is capable of compensating session variability among i-vectors without using speaker labels, in an implicit way. After training, we extract speaker vectors for the testing i-vectors, which are referred to as autoencoder vectors or shortly ae-vectors. For the experimental trials, we score the ae-vectors using cosine scoring. The ae-vectors have shown high discriminative power compared to i-vectors. The experimental results show that while training the autoencoder in the proposed manner, a relative improvement of 42% is gained, over the baseline system using cosine scoring technique, which reduced the performance gap between cosine and PLDA, by 92%.

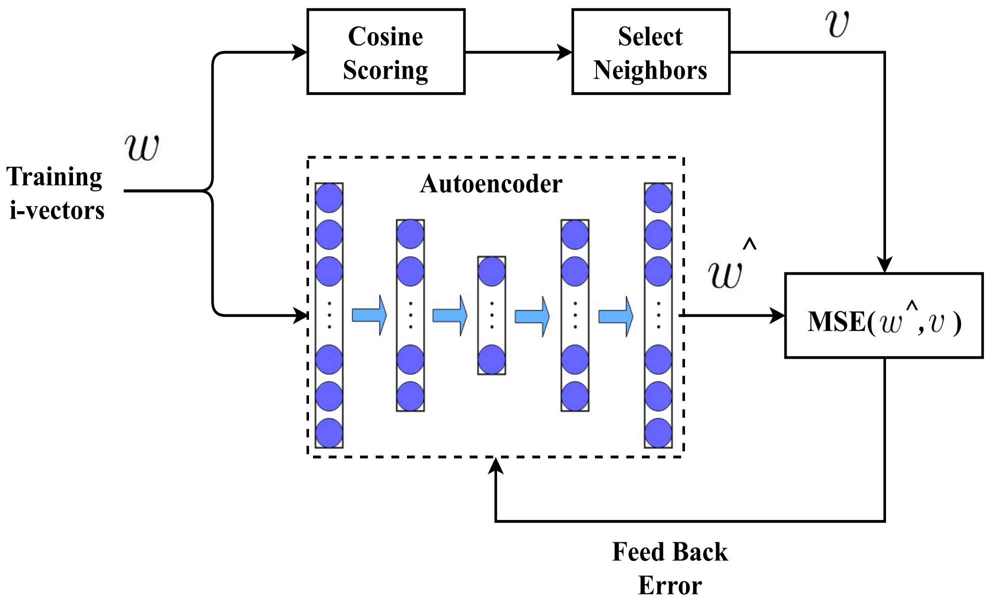


Figure 5.2: Block diagram of the proposed training of the autoencoder. The loss is computed between input i-vector and it's nearest neighbor.

Figure 5.2 shows the block diagram of the training phase of our proposed system (Khan, India, & Hernando, 2019). For an input i-vector w a similar i-vector v is set at the output and the training is carried out in this manner. The similar i-vectors are selected in an unsupervised manner according to the cosine scores to the training i-vector. After training, we extract speaker vectors for the testing i-vectors, which are used in the experiments as shown in Figure 5.3. The main steps involved in extracting ae-vectors are explained as follows.

Training

The conventional architecture of an autoencoder consists of an *encoder* and a *decoder* as shown in the enclosed block of Figure 5.2. The *encoder* is a function that encodes the input i-vector w into a shorter dimensional space, and the *decoder* is a function that decodes it back in order to reconstruct w. The conventional training is carried out by minimizing the MSE loss between the original input i-vector w and the reconstructed $w\hat{}$. Thus the loss function is $MSE(w\hat{}, w)$, where $w\hat{} = decoder(encoder(w))$.

In this work, we propose to train the autoencoder by minimizing the loss function $MSE(w\hat{}, v)$, as shown in Figure 5.2, where v is a similar i-vector to w and

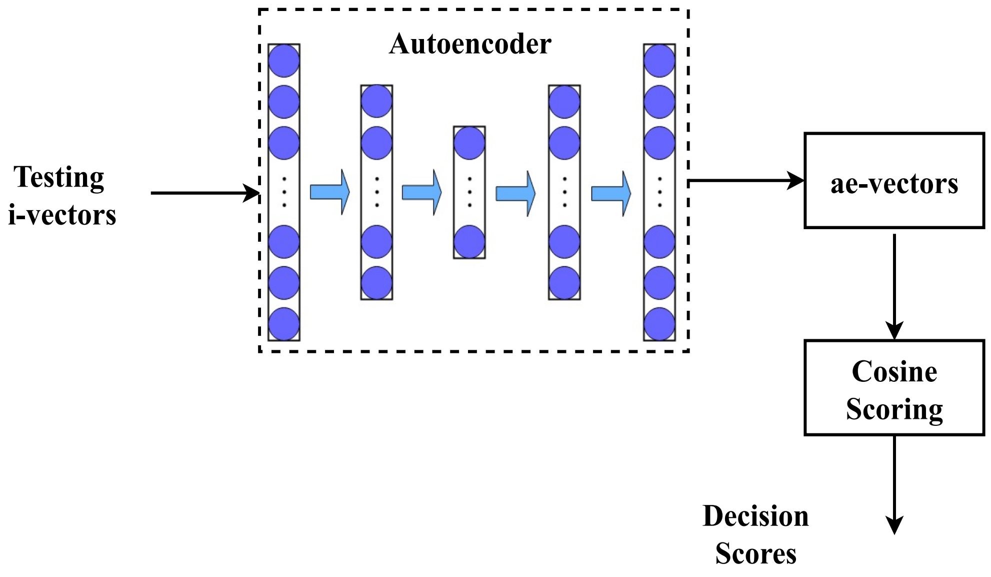


Figure 5.3: Block diagram of the proposed Autoencoder vector extraction for the test i-vectors.

$w\hat{} = decoder(encoder(w))$. We propose an automatic selection of similar i-vectors. Multiple similar i-vectors can be considered for every training i-vector w. We will compare the results of our proposed training with that of a conventional training i.e., reconstructing the same training i-vectors.

Autoencoder Vector Extraction

Once the autoencoder is trained with the selected neighbor i-vectors, we transform the testing i-vectors into a new speaker vector representation, using the autoencoder as shown in Figure 5.3. We extract the desired speaker vectors at the output of the autoencoder. These are referred to as autoencoder vectors or shortly ae-vectors (Khan, India, & Hernando, 2019). In the experiments, ae-vectors have shown to increase the discriminative quality of i-vectors without using speaker labels. Using the proposed ae-vectors, we perform the trials of the experiments with cosine scoring technique.

5.3 Average Pooled Nearest Neighbor Autoencoder

Figure 5.4 shows the block diagram of the training and testing phases of our proposed system (Khan et al., 2020). For every input i-vector w a set of nearest neighbor i-

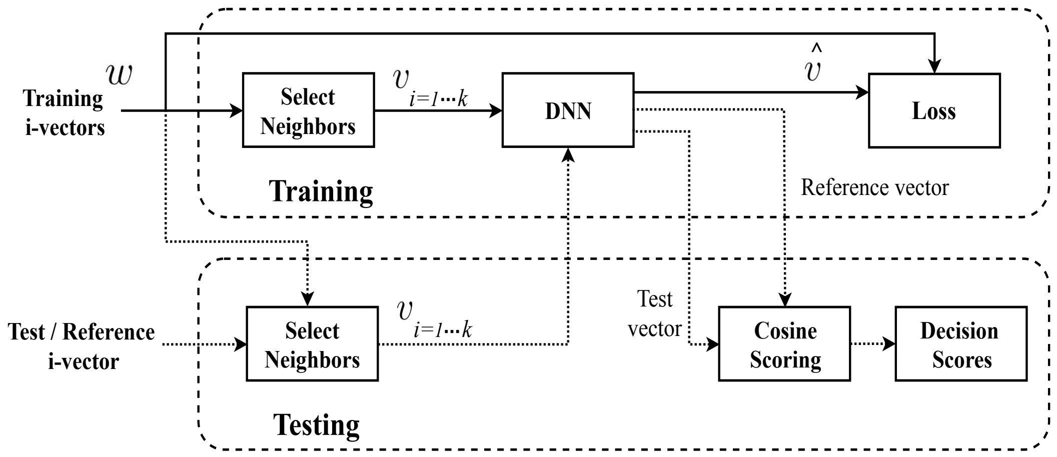


Figure 5.4: Block diagram of the proposed system. Solid arrows corresponds to the training phase, while dotted arrows corresponds to the proposed vector extraction and testing phase.

vectors v_k is input to the DNN. During the training we minimize the loss between reconstructed $\hat{v}$ and actual training i-vector w. The nearest neighbor i-vectors are selected in a similar manner as discussed in Section 5.1. In this work, we propose to set a *threshold* after selecting the constant k number of neighbors. Algorithm 5.2 summarizes how the selection of the neighbor i-vectors is carried out for a constant k number of neighbors constrained to a threshold. After we select the top k neighbors, their corresponding scores are constrained to a *threshold*. After training, we extract speaker vectors for the testing i-vectors, which are used in the experiments as shown in Figure 5.4. The main steps involved in the proposed system are explained as follows.

Training

Once we select the k nearest neighbors for every training i-vector, we train the DNN using these nearest neighbors. Figure 5.5 shows the architecture of our proposed DNN. The first layer performs an average pooling over the input samples which is followed by several Fully Connected (FC) layers. The input layer is fed by the set of neighbor i-vectors denoted by v_k. We train the DNN by minimizing the loss function $L(\hat{v}, w)$, as shown in Figure 5.4, where $L(\cdot)$ can be Cosine Distance (CD) or MSE, w is the training i-vector and $\hat{v} = f(v_k)$, where $f(\cdot)$ is the non-linearity deployed by the DNN. During the training, the loss $L(\cdot)$ is back-propagated to the network

Algorithm 5.2: Proposed neighbor i-vectors selection algorithm for a constant k, constrained by a threshold.

```
   Input  : Training i-vectors $w_i$, $1 \leq i \leq n$
   Output: Neighbor i-vectors $v_{ij}$, $1 \leq i \leq n$ & $1 \leq j \leq k$
1  for each training i-vector $w_i$ do
2      for each training i-vector $w_t$, $1 \leq t \leq n$ do
3          if $i \neq t$ then
4              Compute $score_{i,t} = cosine(w_i, w_t)$
5          end
6      end
7      From $score_{i,t}$, select $k$ highest scores as $S_k$.
8      if $S_k \geq threshold$ then
9          $v_{i,j} = w_t$
10     end
11 end
```

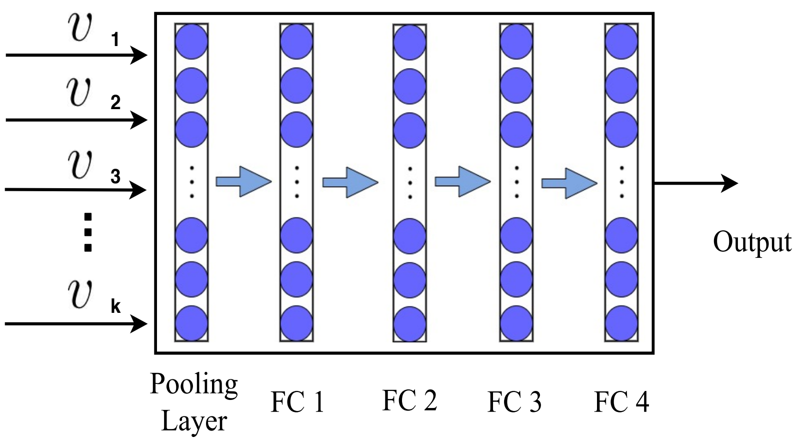


Figure 5.5: Proposed DNN architecture.

in every iteration. In this way, the DNN is able to learn from the nearest neighbor i-vectors and avoids using actual speaker labels.

Speaker Vector Extraction

After the DNN is trained with the selected k nearest neighbor i-vectors, we transform the test i-vectors into a new speaker vector representation, using the DNN as shown in Figure 5.4. For every test i-vector, we select nearest neighbors from the training set in a similar manner as discussed in Section 5.1. The DNN is fed with these nearest neighbors and the desired speaker vectors are extracted at the output of the DNN. Using these speaker vectors, we perform the trials of the experiments with cosine scoring technique.

The proposed vectors are extracted using nearest neighbor i-vectors i.e., v_k. The role of v_k is very important in the whole process because the proposed vectors are highly dependent on v_k. Furthermore, in order to extract the proposed vectors for the test set, it is necessary to store the training i-vectors which may not be convenient in some cases. Therefore, our proposed vectors avoid speaker labels for the background data but at the cost of keeping the background data even in the testing part (Khan et al., 2020).

5.4 Experimental Setup and Database

The experiments were performed on VoxCeleb-1 database (Nagrani et al., 2017; Chung et al., 2018). It contains 148,642 development and 4,874 test utterances which belong to 1,211 and 40 speakers, respectively. For i-vector baseline, the development set was used to train the UBM, the TV matrix and the PLDA parameters. MFCC features of 20 dimensions along with delta coefficients and a 1024 component UBM were used to extract 400 dimensional i-vectors. The PLDA was trained for 20 iterations with 200 eigenvoices. The whole i-vector extraction process was carried out using Alize toolkit (Larcher et al., 2013). The performance was evaluated using the EER(%) and the minimum of the Detection Cost Function (minDCF) calculated using $C_M = C_{FA} = 1$, and $P_T = 0.01$, as in (Nagrani et al., 2017). From the test partition of the database 37,720 experimental trials were scored. Half of them are client trials while the other half are impostor trials.

The Nearest Neighbor Autoencoder, explained in 5.2, was a fully connected feed forward network which consists of 3 hidden layers. The encoder and decoder parts are symmetrical as shown in Figure 5.2. The hidden layer 1 and 3 have 300 neurons each, while hidden layer 2 consists of 200 neurons. The input and output layers consist of 400 neurons each. The training was carried out with 100 epochs using SGD optimizer in Keras deep learning library (Chollet, 2015). All the layers of the autoencoder used ReLU activation except the last layer which used linear activation. The learning rate was set to 0.01 with a decay of 0.0002 and the batch size was 100.

The Average Pooled Nearest Neighbor Autoencoder, explained in 5.3, was trained with the development set using Keras deep learning library. With k number of nearest neighbors there are a total of $(N \times k)$ training samples, where N is the number of i-vectors in the development set. The architecture of the network consists of a pooling layer followed by 4 FC layers as shown in Figure 5.5. The first layer deploys an average pooling while the FC layers performs a ReLU activation except FC4 which uses linear activation. The structure of the pooling layer varies with the number k while that of the FC layers is fixed with 400 neurons each. The DNN was trained for a maximum of 500 epochs using SGD optimizer. The learning rate and batch size were set to 0.01 and 100, respectively.

5.5 Results

5.5.1 Nearest Neighbor Autoencoder

We have compared our proposed ae-vectors with baseline i-vectors and ae-vectors extracted using a conventionally trained autoencoder i.e., same output as input. In Tables 5.1 & 5.2, and in Figures 5.6 & 5.7, approach [2] corresponds to ae-vectors extracted using a conventionally trained autoencoder. Table 5.1 compares the performance of our proposed ae-vectors with the baseline i-vectors by setting a threshold to select number of neighbor i-vectors. All the vectors are scored using cosine scoring technique. Different values of *threshold* were evaluated in order to tune the system for obtaining best results. From the table it is clear that our proposed ae-vectors has outperformed the baseline system. As we decrease the value of *threshold*, the performance of the system improves. The best EER of 11.19% was obtained for a *threshold* equal to 0.1 which gains a relative improvement of 36% over the baseline i-vectors. This leads to fill the performance gap between cosine and PLDA scoring techniques by 80%.

The best value for the minDCF was obtained for a *threshold* equal to 0.4. A score level fusion of i-vectors and the ae-vectors with *threshold* equal to 0.1, has further improved the performance, both in terms of EER and minDCF. An EER of 10.17% was obtained for the fusion strategy. The optimum weights for the fusion were obtained empirically, and were set to 0.55 and 0.45 for i-vectors and ae-vectors, respectively.

Table 5.1: EER and minDCF for the proposed ae-vectors and i-vectors evaluated for different values of threshold using cosine scoring.

Approach	**threshold**	**EER(%)**	**minDCF**
[1] i-vector	-	17.61	0.8390
[2] ae-vector	-	16.84	0.8569
[3] ae-vector	0.4	14.92	**0.8376**
[4] ae-vector	0.3	12.91	0.8594
[5] ae-vector	0.2	11.65	0.8420
[6] ae-vector	0.1	**11.19**	0.8527
Fusion of [1] & [6]	-	**10.17**	**0.7568**

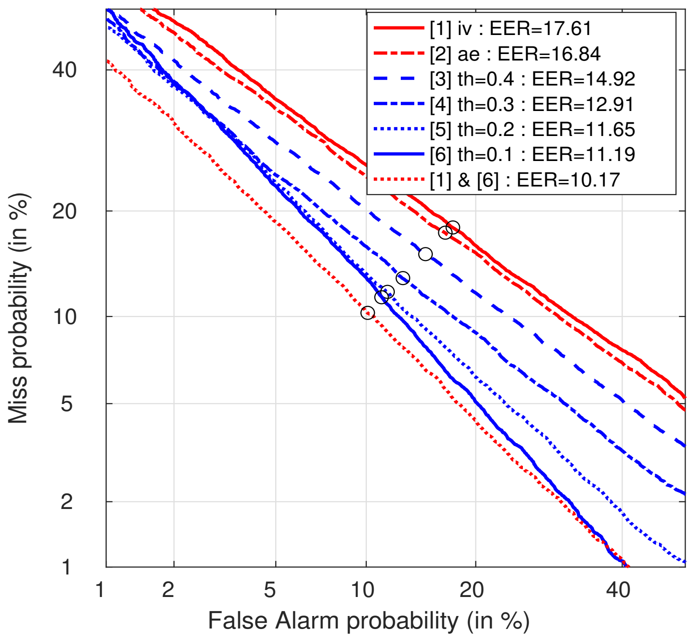


Figure 5.6: DET curves for the proposed ae-vectors and i-vectors evaluated for different values of threshold using cosine scoring.

Figure 5.6 shows a comparison of the DET curves for the baseline and the proposed system. Different plots are shown for different ae-vectors obtained with different values of the *threshold* parameter. It can be observed that all the ae-vectors, show better performance in all working regions, compared to i-vectors.

Table 5.2: EER and minDCF for the proposed ae-vectors and i-vectors for different values of k using cosine scoring.

Approach	**k**	**EER(%)**	**minDCF**
[1] i-vector	-	17.61	0.8390
[2] ae-vector	-	16.84	0.8569
[3] ae-vector	1	15.32	0.9218
[4] ae-vector	2	12.36	0.8382
[5] ae-vector	5	10.62	**0.8053**
[6] ae-vector	15	**10.20**	0.8066
Fusion of [1] & [6]	-	**9.82**	**0.7625**

In Table 5.2, we have shown a performance comparison of our proposed ae-vectors with the baseline i-vectors using a constant k for the selection of neighbor i-vectors. We have experimented with different values of k in order to tune the system for obtaining best results. From the table it is clear that a fix value of k has shown improvement compared to the *threshold* approach as well as to the baseline. Starting with k equal to 1, an EER of 15.32% was obtained. Thus, by considering only one nearest neighbor for every i-vector, a relative improvement of 13% is gained, compared to the baseline system. As we increase the value of k, the performance of the system improves. The best EER of 10.20% was obtained for k equal to 15 which gains a relative improvement of 42% over the baseline i-vectors. This has filled the performance gap between cosine and PLDA scoring techniques by 92%. The best result in terms of minDCF, was obtained with k equal to 5. This approach allows a balanced training which improved the performance of the system. A further increase in the value of k may include very far neighbors, which degraded the performance. A score level fusion of i-vectors and ae-vectors with k equal to 15, has further improved the performance, both in terms of EER and minDCF. An EER of 9.82% was obtained for the fusion, which is very close to the results for i-vector/PLDA using actual speaker labels. The optimum weights for the fusion were tuned experimentally and were set to 0.49 and 0.51 for i-vectors and ae-vectors, respectively.

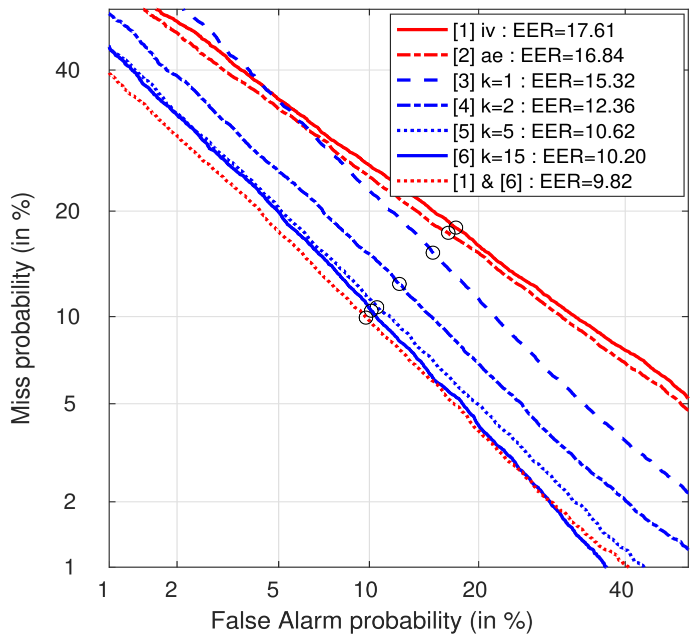


Figure 5.7: DET curves with different values of k using cosine scoring.

The DET curves for the baseline and the proposed system, using the second method, are shown in Figure 5.7. Different plots are shown for different ae-vectors obtained with different values of k. It can be observed that all the ae-vectors, have shown better performance than the i-vectors in all working regions.

If we perform a score level fusion between i-vector/PLDA and the ae-vectors with k equal to 15, which gives the best results, an EER of 9.0% is obtained. This gains a relative improvement of almost 6% over i-vector/PLDA. The EER comparison and DET curves for the fusion are shown in Table 5.3 and Figure 5.8, respectively. The fusion has improved the system in terms of minDCF as well. The optimum weights for the fusion were set to 0.04 and 0.96 for i-vectors and ae-vectors, respectively.

Table 5.3: Fusion of i-vector/PLDA and the proposed ae-vectors with k equal to 15.

Approach	**Scoring**	**EER(%)**	**minDCF**
[1] i-vector	PLDA	9.54	0.7768
[2] ae-vector ($k = 15$)	Cosine	10.20	0.8066
Fusion of [1] & [2]	-	**9.00**	**0.7338**

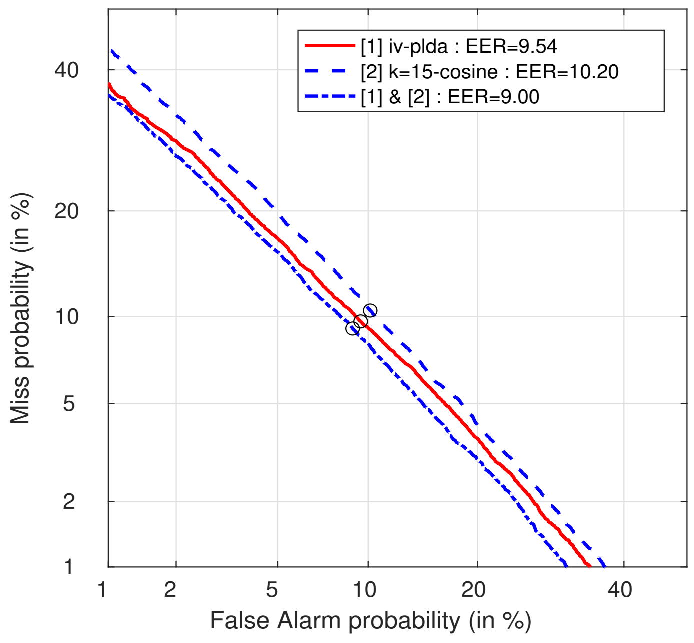


Figure 5.8: DET curves for the fusion of i-vector/PLDA and the proposed ae-vectors with k equal to 15.

5.5.2 Average Pooled Nearest Neighbor Autoencoder

Table 5.4 shows the results obtained for our proposed vectors obtained using the average pooled nearest neighbor autoencoder. This experiment was performed in order to choose the optimal value of k without setting the *threshold* on it. We have shown the EER for both CD and MSE losses as discussed in Section 11. From Table 5.4 we can observe that, an increase in the value of k results in a better performance in terms of EER. This relation is seen up-to a certain value of k. The best EER of 4.45% was obtained for k equal to 100 in the case of MSE loss, which gained a relative improvement of 25% and 53% over the baselines x-vector/PLDA and i-vector/PLDA, respectively. However, for k higher than 100 the EER started increasing. The EER is worse in the case of CD loss for k higher than 100. In overall, we can observe that the proposed vectors obtained using MSE loss performs better compared to CD loss for all values of k.

Figure 5.9 shows a comparison of the DET curves for the baseline i-vectors and our proposed vectors. Different DET plots are shown for different vectors obtained with different values of k. The proposed vectors were extracted using DNN trained

Table 5.4: EER(%) for the proposed vectors using both CD and MSE losses with different values of k. Using i-vectors, the EER(%) is equal to 17.61 and 9.54 for cosine and PLDA, respectively (Khan, India, & Hernando, 2019).

k	EER using CD	EER using MSE
10	8.81	8.70
20	6.60	6.56
30	5.68	5.64
50	4.97	4.98
100	4.84	**4.45**
150	6.53	4.48

with MSE loss as discussed in Section 11. From the DET curves, it can be observed that all the proposed vectors show better performance in the middle regions (i.e., the EER), compared to the baseline i-vectors. However, in the regions where False Acceptance Rate (FAR) and False Rejection Rate (FRR) are very high, the i-vectors tend to show better performance than our proposed speaker vectors.

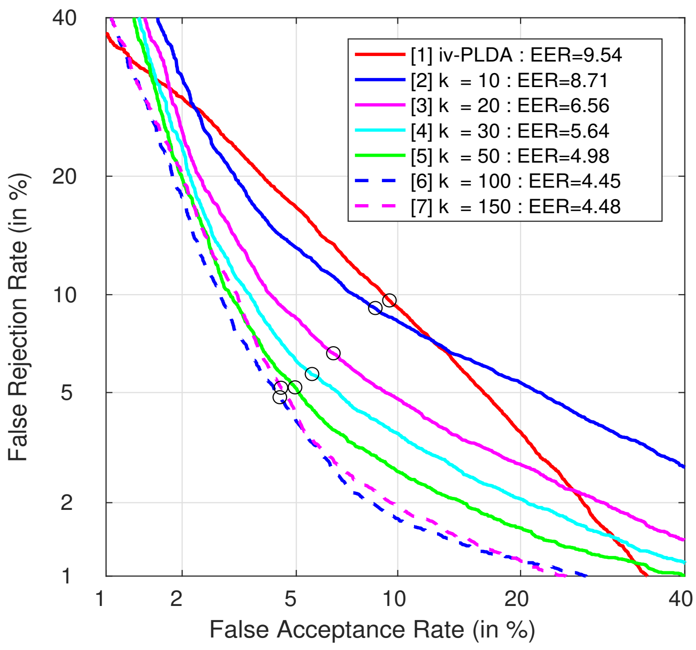


Figure 5.9: DET curves using MSE loss with different values of k.

Table 5.5: EER(%) with & without a *threshold* of 0.0 applied on high values of k.

k	**No threshold applied**	**threshold applied**	**Relative Improvement**
100	4.45	4.45	0.00
150	4.48	4.40	1.78
200	5.03	4.45	**11.53**

After choosing an optimal value of k, we have applied a *threshold* on k as discussed in Section 5.1. In Table 5.5, we have shown the effect of the *threshold* applied to the cosine scores of k nearest neighbors. The value of *threshold* was empirically set to 0.0. As per our first experiment the optimal value of k is equal to 100, which is not effected by the *threshold*, as shown in Table 5.5. When we increase the value of k, the impact of the *threshold* becomes more effective. This is because a high value of k includes a high number of nearest neighbors which may not be optimal for all the i-vectors in the whole database. Thus the *threshold* eliminates some neighbors with cosine scores lower than the *threshold*. This resulted in some improvement compared to the case where no *threshold* was applied. The relative improvement in terms of EER(%) is 11.53% for the case of k equal to 200.

In overall, this approach has obtained 25% and 53% relative improvement over the supervised baselines respectively. The main advantage of our approach is that it avoids speaker-labeled background data. On the other hand, the necessity of the background data in the testing part can be considered as the cost of this improvement. Due to the usage of nearest neighbors for the testing i-vectors we have obtained excellent results.

5.6 Conclusion

In this chapter we proposed the use of autoencoder for i-vector based speaker verification, without using speaker labels for the background data. An autoencoder was trained in a new framework in order to increase the discriminative power of i-vectors. The training takes advantage of the nearest neighbor i-vectors for all the database. The nearest neighbor i-vectors were selected in an unsupervised manner.

During testing, the test i-vectors were transformed into new speaker vectors that were more discriminative as compared to the original i-vectors. The objective was to fill the performance gap between the cosine and the PLDA scoring techniques when no labeled background data is available.

The evaluation was performed on the speaker verification trials of VoxCeleb-1 database. The experimental results have shown that our proposed ae-vectors gain a relative improvement of 42% in terms of EER over the unsupervised baseline system (Khan, India, & Hernando, 2019). While the second nearest neighbor approach have shown that our proposed speaker vectors gain a reasonable improvement in terms of EER over the supervised baseline systems (Khan et al., 2020). This approach avoids speaker-labeled background data at the cost of using the background data even in the testing part.

Chapter 6

Unsupervised training of siamese networks for speaker verification

In the previous chapters, we have explained several autoencoder based unsupervised approaches for post-processing of i-vectors, which has shown to improve the performance of speaker verification system (Khan, India, & Hernando, 2019; Khan & Hernando, 2019; Khan et al., 2020). Since, these approaches were aimed to increase the discriminative power of i-vectors. They are applicable as a back-end in the utterance level features i.e., in i-vector space only. In this chapter we avoid the i-vector extraction part, in testing phase, by training the networks using the frame level features, i.e., spectrograms. The goal is: (1) to obtain end-to-end speaker verification scores, and (2) to extract speaker embeddings, from spectrograms without using speaker labels. We propose two siamese networks (Bromley et al., 1994), i.e., double-branch and triple-branch, which consist of two and three branches, respectively. Each branch is composed of a Convolutional Neural Network (CNN) encoder, inspired by the VGG architecture (Simonyan & Zisserman, 2014), which was recently adapted for speaker verification in (Nagrani et al., 2017; Chung et al., 2018; India et al., 2019). Typically, siamese networks are trained by feeding the training samples in pairs, e.g., [anchor, client] and [anchor, impostor]. Since the goal is to avoid speaker labels, we propose to generate the pairs of training samples in an unsupervised manner (Khan & Hernando, 2020a). The client samples are selected within one database according to the highest similarity with the anchor. The impostor samples are selected in the same way but from another database, provided that the two databases does not contain utterances from same speakers.

We propose the use of double-branch siamese network as a binary classifier by minimizing binary cross entropy loss. The selected client and impostor samples, paired one by one with the anchor samples, are fed into the network, and their respective binary labels of 1/0 are provided at the output in order to compute the loss. After training, we obtain decision scores for the speaker verification trials from the output of the network. Thus, our double-branch siamese deploys an end-to-end speaker verification system. On the other hand, the proposed triple-branch network is trained by minimizing triplet loss (J. Wang et al., 2014; Schroff et al., 2015). The selected client and impostor samples, paired both at the same time with the anchor samples, are fed into the network in pairs of three. Each branch encodes the input sample into a vector based representation which is used to compute the triplet loss. In the testing phase, we extract speaker embeddings for the test data using a branch of the network. These embeddings are scored for the experimental trials using cosine scoring. The evaluation was performed on VoxCeleb-1 database (Nagrani et al., 2017). The results show that using our proposed systems, despite of being unsupervised, the result get closer to a similar but fully supervised baseline. Moreover, fusion of the two proposed systems can further improve the performance.

The rest of the chapter is organized as follows. Section 6.1 explains the proposed method for the selection process of clients and impostor samples. Section 6.2 depicts the proposed architectures of our double-branch and triple-branch siamese networks. Section 6.3 describes the experimental setup and the database. The results obtained are discussed in Section 6.4. Finally in section 6.5, some conclusions are drawn as the findings of this work.

6.1 Clients and Impostor Selection

The selection process of client and impostor samples is carried out in the i-vector space using two databases, i.e., A and B. Suppose Spk_A and Spk_B denote the speakers appearing in database A and B, respectively. We assume that the speakers in database A do not appear in database B, i.e., $Spk_A \cap Spk_B = \phi$. Algorithm 6.1 summarizes how the selection of the clients and impostor i-vectors is carried out in an unsupervised manner. First of all we extract i-vectors for all the utterances in both the databases A and B. Then, all the i-vectors in A are scored among each other using cosine scoring technique. For every i-vector in A we select a fix k number

Algorithm 6.1: Proposed algorithm for selection of clients and impostor i-vectors

```
Input  : Training i-vectors $a_i \in A$ & $b_i \in B$, $1 \leq i \leq n$
Output: Client and impostor i-vectors $C_{ij}$ and $I_{ij}$, $1 \leq i \leq n$ & $1 \leq j \leq k$
1  for each training i-vector $a_i$ do
2      for each training i-vector $a_t$, $1 \leq t \leq n$ do
3          Compute $ClientScores_{i,t} = cosine(a_i, a_t)$
4      end
5      From $ClientScores_{i,t}$, select $k$ highest ones.
6      if $ClientScores_{i,t} \geq threshold$ then
7          $C_{i,j} = a_t$
8      end
9      for each training i-vector $b_t$, $1 \leq t \leq n$ do
10         Compute $ImpostorScores_{i,t} = cosine(a_i, b_t)$
11     end
12     From $ImpostorScores_{i,t}$, select $k$ highest ones.
13     if $ImpostorScores_{i,t} \geq threshold$ then
14         $I_{i,j} = b_t$
15     end
16 end
```

of similar i-vectors as potential client i-vectors. After this we apply a *threshold* to the cosine scores of these k selected potential clients. The potential clients with scores higher than the *threshold* are selected as the final client i-vectors.

In order to select the impostor i-vectors, we score all the i-vectors in A with those in B, using cosine scoring. For every i-vectors in A, we select k number of i-vectors from B that are closest according to the cosine scores. Since, the speakers in A does not appear in B, these k selected i-vectors are the potential impostors. After this, we apply a threshold to their corresponding cosine scores in order to select the hardest impostors among them. In this way, every i-vector in A has been assigned k client and k impostor i-vectors. Suppose we have n number of i-vectors in each of the databases A and B. Then, we have a total of $(2n \times k)$ samples for training our networks. The value of k is determined experimentally and will be discussed in section 6.4. For the double-branch siamese network we make training pairs of two, i.e., [anchor, client] and [anchor, impostor], for which the binary labels are 1 and 0 respectively. Whereas for the triple-branch siamese network we make pairs of three samples, i.e., [anchor, client, impostor], in order to compute the triplet loss according to Equation 6.1. It is worth noting the client and impostor selection is carried out in i-vector space while the actual inputs to the networks are Mel-spectrogram features of the utterances.

6.2 Proposed Siamese Architectures

Training a Deep Neural Network (DNN), either in end-to-end fashion or to extract speaker embeddings, usually requires speaker labels for the background data, which is difficult to access in reality. In order to do so, when no labels are available for the background data, we propose the use of two siamese networks which are fully unsupervised unlike the conventional DNN classifiers. Typically, a siamese network is trained using pairwise training samples, i.e., anchor, client and impostor. Since we do not use speaker labels, we propose to generate the training pairs in an unsupervised manner. These training pairs are fed at the input of the network. We propose two different networks, i.e., a double-branch siamese network using binary cross-entropy loss, and a triple-branch siamese network using triplet loss.

After training, we obtain decision scores for the experimental trials at the output of the double-branch network. Whereas the triple-branch network is used to extract speaker embeddings for the test data. These speaker embeddings are scored using cosine scoring in order to perform the experimental trials.

6.2.1 Double-Branch for End-to-End Speaker Verification

Figure 6.1 shows the block diagram of our proposed double-branch siamese network. There are two identical branches, i.e., the CNN encoder. Mel-spectrogrm features of a training pair of an anchor along with a client or an impostor sample is fed as input to the network. The two branches share weights and biases with each other. After the CNN encoder, the outputs of the two branches are concatenated which is followed by five Fully Connected (FC) layers. The last layer is connected to the binary class labels, i.e., 1/0, indicating if the anchor sample is paired by a client/impostor sample, respectively. During training, binary cross-entropy loss is minimized to update the network weights. Once the network is trained with

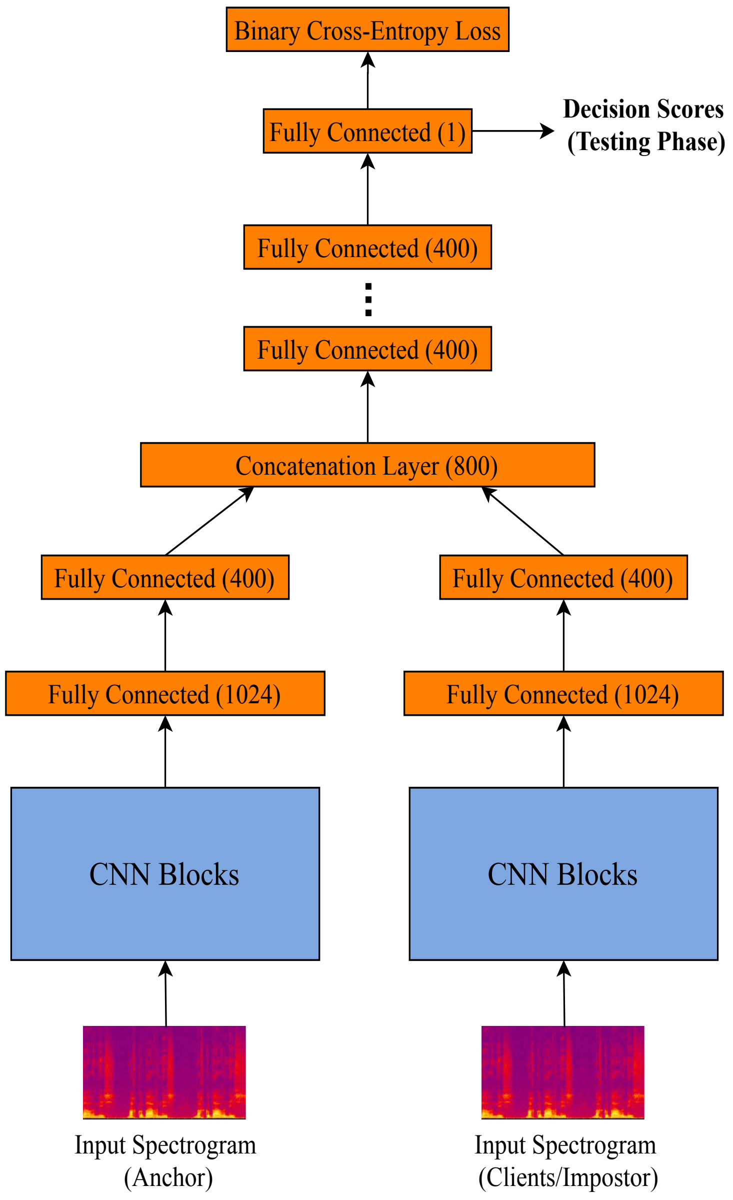


Figure 6.1: Block diagram of our double-branch siamese network. In testing phase decision scores for speaker verification are obtained form last FC layer.

the selected client and impostor samples in unsupervised manner, we perform the evaluation in an end-to-end fashion. A pair of reference and test utterances, which is involved in an experimental trial, is fed into the network. The decision scores are obtained directly from the output of the last fully connected layer of the network. In this way, our double-branch siamese network deploys an unsupervised end-to-end speaker verification system which does not require any scoring backend.

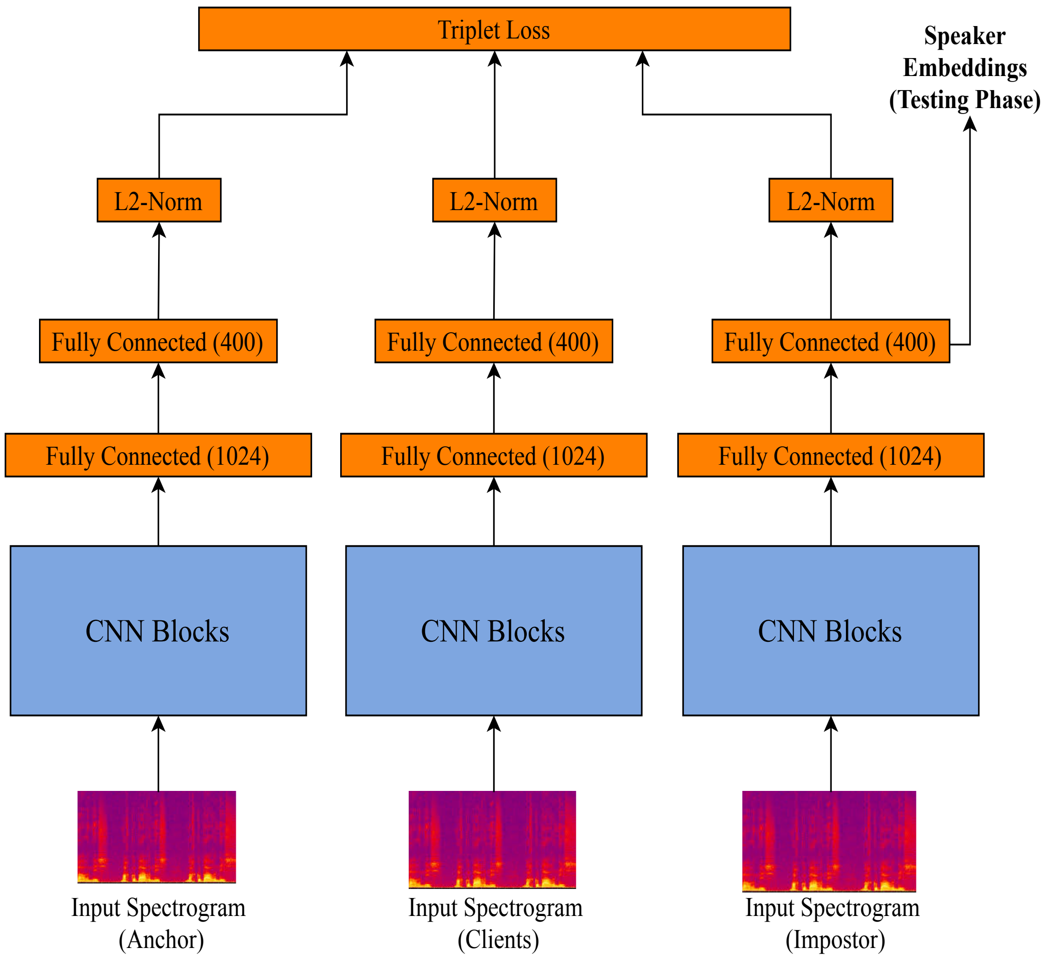


Figure 6.2: Block diagram of our triple-branch siamese network. Speaker embeddings are extracted from the last FC layer of any CNN encoder branch.

6.2.2 Triple-Branch for Speaker Embeddings

A block diagram of our proposed triple-branch siamese network is shown in Figure 6.2. As indicated by the name, there are three branches, each of which is the CNN encoder followed by l2-normalization layer. Each branch is fed by the Mel-spectrogrm features of a training pair of an anchor along with a client and an impostor sample. The CNN encoder encodes the Mel-spectrogram inputs into a vector based representation i.e., speaker embeddings. After the l2-normalization layer, triplet loss (J. Wang et al., 2014) is computed between the embeddings of anchor, client and impostor samples. It is worth noting that all the three branches share weights with each other, like in the double-branch network. The triplet loss is computed as follows:

$$L_{triplet} = max(d(a, c) - d(a, i) + m, 0) \tag{6.1}$$

where $d(.)$ is the distance between two samples, and a, c and i denote the anchor, client and impostor samples, respectively. m is the margin value which defines how far away the dissimilarities should be. Once the network is trained, we extract speaker embeddings using any CNN encoder branch of the network. These speaker embeddings are scored using cosine scoring technique for the experimental trials.

6.2.3 CNN Encoder

The CNN encoder block is inspired by the VGG architecture (Simonyan & Zisserman, 2014), which was recently adapted for speaker verification task in (Nagrani et al., 2017; Chung et al., 2018; India et al., 2019). It consists of three main blocks, where each block contains two convolutional and one maxpooling layer. The three blocks are followed by a self attention pooling (SAP) layer (Cai et al., 2018), and two FC layers of 1024 and 400 neurons, respectively. The CNN encoder encodes the

Table 6.1: Architecture of the VGG based CNN Encoder. In and Out Dim. refers to the input and output feature maps of the layer. Feat Size is the dimension of every output feature map.

Layer	Size	In dim	Out dim	Stride	Feat size
conv1-1	3x3	1	128	1x1	80xN
conv1-2	3x3	128	128	1x1	80xN
mpool-1	2x2	-	-	2x2	40xN/2
conv2-1	3x3	128	256	1x1	40xN/2
conv2-2	3x3	256	256	1x1	40xN/2
mpool-2	2x2	-	-	2x2	20xN/4
conv3-1	3x3	256	512	1x1	20xN/4
conv3-2	3x3	512	512	1x1	20xN/4
mpool-3	2x2	-	-	2x2	10xN/8
SAP	-	N/8	1	-	512x10
fc-1	-	1	1	-	1024
fc-2	-	1	1	-	400

input Mel-spectrograms of shape $(80 \times N)$ into 400 dimensional vectors, i.e., speaker embeddings, where N is the number of frames. More details of the CNN encoder architecture are given in Table 6.1.

6.3 Experimental Setup and Database

The training was performed on the development partition of VoxCeleb-2 database (Chung et al., 2018) which contains 5994 speakers having 1,092,009 utterances in total. The evaluation was performed on the test partition of VoxCeleb-1 (Nagrani et al., 2017), which contains 40 speakers having 4,874 utterances in total. The development partition of VoxCeleb-2 was used to train the two siamese networks. For the i-vector extraction process, the same partition was used to train the Universal Background Model (UBM) and the Total Variability (TV) matrix. MFCC features of 20 dimensions, appended by delta coefficients, were extracted for all the utterances, and a 1024 component UBM was trained to extract i-vectors of length 400. From the test partition of VoxCeleb-1, 37,720 speaker verification trials were scored. Half of them are client trials while the other half are impostor trials. The performance was evaluated using the Equal Error Rate (EER).

For the clients and impostor selection we split the development partition of VoxCeleb-2 into two equal parts, in order to generate databases A and B as discussed in Section 6.1. Mel-spectrograms of 80 dimensions were computed, of which a randomly selected window of length N was input to the networks. The CNN encoder, depicted in Table 6.1, was identical for both the networks. The value of N, in Table 6.1, was set to 350 frames. The margin value m while computing the triplet loss defined in Equation 6.1 was set to 0.8. The value of *threshold* in the selection of client and impostor samples was set to 0.2 and 0.0, respectively. All the CNN and fully connected layers were activated using ReLU function, whereas the last layer of the double-branch network used sigmoid activation. The training was carried out using Adam optimizer with the initial learning rate and batch size of 0.0001 and 35, respectively. The training continued for a maximum of 500 epochs using an early stopping criteria with a patience equal to five epochs.

The baseline system was trained using the whole development partition of VoxCeleb-2. The architecture of the baseline is composed of a CNN encoder followed by a classification layer at the end. In order to have a fair comparison, we used exactly

Table 6.2: EER in % for the proposed double- and triple-branch siamese networks, compared with the supervised baseline. Different results are shown for different values of k.

Approach	**k**	**EER(%)**
[1] Baseline (Softmax)	-	6.81
[2] Baseline (AMSoftmax)	-	5.71
[3] Double-branch	2	7.81
[4] Double-branch	5	7.73
[5] Double-branch	10	6.90
[6] Triple-branch	10	6.95
Fusion of [5] & [6]	10	6.07

the same architecture for the CNN encoder as that of the siamese networks shown in Table 6.1. The only difference is the last layer which has 5994 neurons (number of speakers) activated by softmax and AMsoftmax (F. Wang et al., 2018) functions. We minimized the categorical cross-entropy loss using Adam optimizer with the same parameters as of the siamese networks. In the testing phase, we extract speaker embeddings from the CNN encoder of this network.

6.4 Results

We have compared our proposed unsupervised systems with the supervised baseline in terms of Equal Error Rate (EER). Table 6.2 shows the EER in % for different values of k. The fist three rows of Table 6.2 depict results for the selection of the value of k using our double-branch siamese network. For all the experimental trials we obtain speaker verification scores directly from the output of the double-branch network. From the Table we can see that as we increase the value of k, the performance of the system improves. The best EER of 6.90% was achieved using k equal to 10. We have tried further higher values of k but no substantial improvement was seen, despite increasing the computational cost.

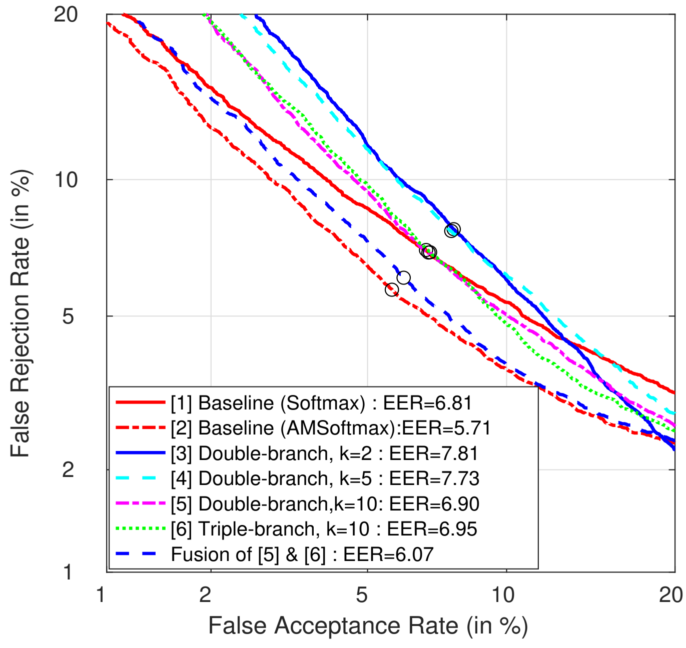


Figure 6.3: DET curves for the proposed double- and triple-branch siamese network, compared with the supervised baseline. Different plots are shown for different values of k.

Setting the value of k equal to 10, we have trained out triple-branch siamese network using triplet loss, as discussed in Section 6.2.2. Using our triple-branch network, first we extract speaker embeddings for all the test data which requires a scoring backend. Therefore, we score the obtained speaker embeddings using cosine scoring technique. From the Table we can see that the triple-branch siamese network has achieved an EER of 6.95% which is almost similar to that of the double-branch network. The EER of 6.90% and 6.95%, obtained by the two systems respectively, are very close to that of the supervised Softmax baseline.

Moreover, if we perform a score level fusion of our two proposed systems, i.e, double- and triple-branch networks, we obtain an EER of 6.07%. This has outperformed the softmax baseline by a relative improvement of 10.86%. However, the fusion strategy could not overcome the second baseline with AMSoftmax activation.

Figure 6.3 shows a comparison of the Detection Error Trade-off (DET) curves for the supervised baselines and our proposed systems. Different plots are shown for scores obtained with different values of the k using the double-branch siamese network. As discussed earlier, we observed that k equal to 10 is the best choice for

Table 6.3: Comparison of the different proposed approaches. Evaluation was performed on VoxCeleb-1 test set.

Chapter	**Approach**	**Input features During Testing**	**EER(%)**
4	Supervised	i-vectors of test data	7.51
5 (1)	Unsupervised	i-vectors of test data	10.20
5 (2)	Unsupervised	i-vectors of test & background data	4.40
6	Unsupervised	Mel-Spectrograms of test data	6.07

our experiments. After this we plotted the DET curve for triple-branch for k equal to 10 only. It can be observed that both the proposed systems show comparable performance with the softmax baseline in the EER region. Furthermore, in the low FRR regions the DET plot of the proposed systems become closer to the AMSoftmax baseline. However, this relation seems to be reversed in the low FAR regions. Similarly, the DET curve for the fusion of the two proposed systems is even closer to the AMSoftmax baseline in low FRR regions.

A brief comparison of the different proposed approaches is shown in Table 6.3. In Chapter 4 we proposed an autoencoder as a pre-training for DNN training. The autoencoder pre-training was carried out in an unsupervised manner, whereas the DNN training was carried out in a supervised manner. Using this approach, we gain a relative improvement of 21% in terms of EER, over the baseline i-vectors/PLDA system. On the other hand, in Chapter 5 we proposed two completely unsupervised approaches in which relative improvement of 42% and 53% were obtained, respectively. These approaches have the advantage of being unsupervised which was the main goal of this thesis. However, their main disadvantage is that they are only applicable in the utterance level features i.e., i-vectors, because they were aimed at improving the discriminative power of i-vectors. Thus it is necessary to extract i-vectors for the all the background and test data beforehand. Moreover, in the second approach of Chapter 5, it is necessary to have the background i-vectors in the testing phase, as discussed in Section 5.3. In order to avoid the i-vector extraction in the testing phase, in Chapter 6 we proposed to work directly in the frame level features. We used two unsupervised siamese networks, (1) an end-to-end system, and (2) speaker embeddings extractor, which used Mel-Spectrograms as input features. The experimental results have shown that both the proposed networks have obtained comparable performance to a similar but fully supervised baseline, despite of being unsupervised and avoiding i-vector extraction in the testing phase.

The different proposed approaches have their pros and cons over each other. From Table 6.3 it is clear that in terms of EER the best result was obtained with the second approach of Chapter 5. However, it has the disadvantage of relying on the background i-vectors in the testing part, which may not be convenient in most scenarios. If we avoid this factor, we have to compromise on the results which is shown in the first approach of Chapter 5. Compared to this, the result was improved by the approach in Chapter 4 but its DNN training was carried out using speaker labels. Moreover, all these three approaches are only applicable in the i-vector space, which means one need to extract i-vectors of the dataset beforehand. On the contrary, the approach in Chapter 6 is free from the above mentioned constraints and still performs pretty reasonable in terms of EER, as shown in (Khan & Hernando, 2020a,b). Firstly, it is a fully unsupervised approach. Secondly, it uses Mel-Spectrogram features in the testing phase and therefore avoids i-vectors. And finally, it does not rely on the background i-vectors in the testing phase.

6.5 Conclusion

In this work, we proposed two different siamese networks for speaker verification without using speaker labels. The first network has two branches and was trained as a binary classifier, whereas the second network has three branches and was trained to learn speaker embeddings, where each branch was a CNN encoder. Since the goal was to avoid speaker labels, the pairs of training samples were generated in an unsupervised manner. The client samples were selected within one database according to the highest cosine scores with the anchor, in i-vector space. Whereas, the impostor samples were selected in the same way but from another database. After training, we obtain decision scores using the double-branch network, whereas from the triple-branch network we extract speaker embeddings for the test data. The evaluation was performed on VoxCeleb-1 database. The experiments have shown that using our proposed systems, despite of being unsupervised, the result was closer to the fully supervised baselines.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis three major contributions are presented. In the first contribution, a Restricted Boltzmann Machine (RBM) based vector representation of speech was proposed for the tasks of speaker clustering and speaker tracking in TV broadcast shows. This representation was first proposed in (Safari et al., 2016), and was reffered to as RBM vector. First of all a global model was trained using all the available background data, which was reffered to as Universal Restricted Boltzmann Machine (URBM). Then, for the test utterances, a single RBM was trained which was adapted from the URBM. The weight matrices along with the bias vectors of these adapted RBM models were concatenated to generate RBM supervectors. The RBM supervectors were further subjected to a PCA whitening with dimensionality reduction to extract the desired RBM vectors. These RBM vectors were used in the tasks of speaker clustering and speaker tracking. Speaker clustering was explored using two linkage algorithms for an Agglomerative Hierarchical Clustering (AHC) approach. The experiments, performed on AGORA database, have shown that the proposed RBM vectors have outperformed the baseline i-vectors, resulting in a relative improvement of 12% in terms of Equal Impurity (EI). In the case of speaker tracking RBM vectors were used only in the speaker identification part, where the relative improvement in terms of Equal Error Rate (EER) was 11% and 7% using cosine and Probabilistic Linear Discriminant Analysis (PLDA) scoring, respectively.

In the second contribution, we have addressed to the problem of the lack of labeled background data in i-vector based speaker verification. In practice large amount of labeled background data is not easily available, which has a big impact on the performance of i-vectors scored with cosine technique. We make use of Deep Learning (DL) techniques to tackle this problem. For this purpose we proposed the use of autoencoder for speaker verification in three different ways. In the first proposal we used an autoencoder as a pre-training for Deep Neural Network (DNN) training. This unsupervised training was carried out using a large amount of unlabeled background data. Then a DNN classifier was trained using a relatively small labeled data, initialized with the parameters of the pre-trained autoencoder. Although, the autoencoder pre-training was carried out in unsupervised manner, the DNN training was still a supervised approach. Therefore, in the other two proposals we used a completely unsupervised approach. For this purpose, only the unsupervised autoencoder was used in order to increase the discriminative power of i-vectors. This time the training was based on reconstruction of the nearest neighbor i-vectors, instead of the same input i-vectors. The main objective was to fill the performance gap between cosine and PLDA scoring techniques when no labeled background data is available.

The evaluation was performed on the speaker verification trials of VoxCeleb-1 database. The results have shown that in the first proposal, we gain a relative improvement of 21% in terms of EER, over the baseline i-vectors/PLDA system. Furthermore, it was observed that the DNN training was faster, compared to the conventional (randomly initialized) DNN case. In the second proposed approach a relative improvement of 42% was gained, in terms of EER over the unsupervised baseline i-vector system. Whereas in the third approach, using the background data in the testing part, a relative improvement of 53% was gained over the supervised baseline systems.

Since the previous contribution was aimed to increase the discriminative power of i-vectors, it is only applicable as a backend for i-vectors i.e., in the i-vector space. Therefore, in the third contribution we have worked directly in the frame level features, i.e., spectrograms, using Convolutional Neural Network (CNN). We proposed two siamese networks with two and three branched, respectively. The double-branch siamese network was trained as an end-to-end classifier for speaker verification, whereas the triple-branch siamese was aimed to extract unsupervised speaker embeddings trained using triplet loss. The idea was to utilize impostor samples along with the neighbor samples. The selection of training pairs of client and impostor samples was carried out in an unsupervised manner. The siamese architectures were inspired by the VGG Convolutional Neural Network (CNN) encoder. The experimental results have shown that both the proposed networks, despite of being unsupervised, has shown comparable performance to a similar but fully supervised baseline. Moreover, their score level fusion have shown to further improve the performance.

7.2 Future Research Lines

In this thesis, the major objective was to apply DL approaches in speaker recognition tasks avoiding/limiting the use of both phonetic and speaker labels. Labeled training data is expensive and not easily available in practice. On the other hand, the availability of large labeled training data can open numerous research lines in this area. Therefore, the future research lines can be unsupervised and supervised DL approaches. Various supervised approaches has already been published due to the availability of very large labeled databases for speaker recognition e.g: the recent VoxCeleb database (Nagrani et al., 2017; Chung et al., 2018).

A possible research line can be the usage of variational autoencoder and adversarial autoencoder in a similar manner as in Chapter 5. Instead of using a simple vanilla autoencoder, it will be interesting to see the performance of variational and adversarial autoencoders in this task. Furthermore, the selection of neighbor samples could be done in a more sophisticated way unlike discussed in Section 5.1. Instead of just relying on cosine scoring, in order to improve the quality of the neighbor selection one can apply other constraints on the scores. On the contrary, if the actual speaker labels are available for the background data, then the neighbor selection can be fully supervised according to the labels, which could substantially improve the system performance.

Moreover, the usage of training data in the testing part, as shown in Section 11, could be avoided by some mechanism. For instance, it could be tested how the system responds if the same testing sample is replicated several times, instead of using the neighbor samples from the training data. Another possibility is to use a very small training set for selecting the neighbor samples for the testing part. In this way, it will be more feasible to deal with a small portion instead of the whole training set.

Another possibility of future research can be seen in our end-to-end systems. Since, our proposed end-to-end system is based on Mel-spectrograms as the input features for the network. Hence, it is a good idea to use speech waveform directly without the computation of the Mel-spectrograms or any other hand crafted features.

Publications

Journal Articles

Khan, U., ., Safari, P., & Hernando, J. (2019). Restricted boltzmann machine vectors for speaker clustering and tracking tasks in tv broadcast shows. *Applied Sciences*, *9*(13), 2761.

Conference Papers

Khan, U., ., Safari, P., & Hernando, J. (2018). Restricted Boltzmann Machine Vectors for Speaker Clustering. In *Proc. iberspeech* (pp. 10–14).

Khan, U., ., & Hernando, J. (2019). Dnn speaker embeddings using autoencoder pre-training. In *2019 27th european signal processing conference (eusipco)* (pp. 1–5).

Khan, U., ., India, M., & Hernando, J. (2019). Auto-encoding nearest neighbor i-vectors for speaker verification. *Proc. Interspeech 2019*, 4060–4064.

Khan, U., ., India, M., & Hernando, J. (2020). i-vector transformation using k-nearest neighbors for speaker verification. In *Ieee international conference on acoustics, speech, and signal processing (icassp)* (pp. 7574–7578).

Khan, U., ., & Hernando, J. (2020). Unsupervised training of siamese networks for speaker verification. *Proc. Interspeech 2020*, 3002–3006.

Bibliography

Ajmera, J., & Wooters, C. (2003). A robust speaker clustering algorithm. In *2003 ieee workshop on automatic speech recognition and understanding (ieee cat. no. 03ex721)* (pp. 411–416).

Anna, S., Lukáš, B., & Jan, C. (2017). Alternative approaches to neural network based speaker verification. In *Proc. interspeech 2017* (pp. 1572–1575).

Bhattacharya, G., Alam, J., & Kenny, P. (2017). Deep speaker embeddings for short-duration speaker verification. *Proc. Interspeech 2017*, 1517–1521.

Bimbot, F., Bonastre, J.-F., Fredouille, C., Gravier, G., Magrin-Chagnolleau, I., Meignier, S., ... Reynolds, D. A. (2004). A tutorial on text-independent speaker verification. *EURASIP Journal on Advances in Signal Processing*, *2004*(4), 101962.

Bonastre, J. F., Delacourt, P., Fredouille, C., Merlin, T., & Wellekens, C. (2000). A speaker tracking system based on speaker turn detection for nist evaluation. In *Ieee international conference on acoustics, speech, and signal processing (icassp)* (Vol. 2, pp. II1177–II1180).

Bousquet, P.-M., & Rouvier, M. (2017). Duration mismatch compensation using four-covariance model and deep neural network for speaker verification..

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., & Shah, R. (1994). Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems* (pp. 737–744).

Brummer, N. (2010). *Measuring, refining and calibrating speaker and language information extracted from speech* (Unpublished doctoral dissertation). Stellenbosch: University of Stellenbosch.

Brümmer, N., & De Villiers, E. (2011). The bosaris toolkit user guide: Theory, algorithms and code for binary classifier score processing. *Documentation of BOSARIS toolkit*, 24.

Brümmer, N., Swart, A., & van Leeuwen, D. (2014). A comparison of linear and non-linear calibrations for speaker recognition. In *Proceedings of odyssey speaker and language recognition workshop* (pp. 14–18).

Cai, W., Chen, J., & Li, M. (2018). Exploring the encoding layer and loss function in end-to-end speaker and language recognition system. In *Proc. odyssey 2018 the speaker and language recognition workshop* (pp. 74–81).

Campbell, J. P. (1997). Speaker recognition: A tutorial. *Proceedings of the IEEE*, *85*(9), 1437–1462.

Campbell, W. M., Sturim, D. E., & Reynolds, D. A. (2006). Support vector machines using gmm supervectors for speaker verification. *IEEE signal processing letters*, *13*(5), 308–311.

Chen, K., & Salman, A. (2011, 11). Learning speaker-specific characteristics with a deep neural architecture. *IEEE Transactions on Neural Networks*, *22*(11), 1744–1756.

Chen, S., & Gopalakrishnan, P. (1998). Speaker, environment and channel change detection and clustering via the bayesian information criterion. In *Proc. darpa broadcast news transcription and understanding workshop* (Vol. 8, pp. 127–132).

Chollet, F. (2015). *Keras.* https://keras.io. Retrieved from https://keras.io

Chung, J. S., Nagrani, A., & Zisserman, A. (2018). Voxceleb2: Deep speaker recognition. In *Interspeech.*

Cumani, S., Batzu, P. D., Colibro, D., Vair, C., Laface, P., & Vasilakakis, V. (2011). Comparison of speaker recognition approaches for real applications. In *Twelfth annual conference of the international speech communication association.*

Dahl, G. E., Yu, D., Deng, L., & Acero, A. (2011). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, *20*(1), 30–42.

Dehak, N., Dehak, R., Glass, J. R., Reynolds, D. A., & Kenny, P. (2010). Cosine similarity scoring without score normalization techniques. In *Odyssey* (p. 15).

Dehak, N., Kenny, P. J., Dehak, R., Dumouchel, P., & Ouellet, P. (2011). Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, *19*(4), 788–798.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1–38.

Deng, L., & Yu, D. (2014). Deep learning: methods and applications. *Foundations and Trends in Signal Processing*, *7*(3–4), 197–387.

Dey, S., Madikeri, S. R., & Motlicek, P. (2018). End-to-end text-dependent speaker verification using novel distance measures. In *Interspeech* (pp. 3598–3602).

Duda, R., Hart, P., Stork, D., & Ionescu, A. (2001). *Pattern classification, chapter nonparametric techniques.* Wiley-Interscience.

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis* (Vol. 3). Wiley New York.

Furui, S. (1981). Cepstral analysis technique for automatic speaker verification. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *29*(2), 254–272.

Furui, S. (2004). Fifty years of progress in speech and speaker recognition. *The Journal of the Acoustical Society of America*, *116*(4), 2497–2498.

Garcia-Romero, D., & Espy-Wilson, C. Y. (2011). Analysis of i-vector length normalization in speaker recognition systems. In *Twelfth annual conference of the international speech communication association.*

Gauvain, J.-L., Lamel, L., Adda, G., & Jardino, M. (1999). The limsi 1998 hub-4e transcription system. In *Proc. darpa broadcast news workshop* (pp. 99–104).

Ghaemmaghami, H., Dean, D., Sridharan, S., & van Leeuwen, D. A. (2016, 11). A study of speaker clustering for speaker attribution in large telephone conversation datasets. *Computer Speech & Language*, *40*, 23–45.

Ghahabi, O., & Hernando, J. (2014a). Deep belief networks for i-vector based speaker recognition. In *Ieee international conference on acoustics, speech, and signal processing (icassp)* (pp. 1700–1704).

Ghahabi, O., & Hernando, J. (2014b). I-vector modeling with deep belief networks for multi-session speaker recognition. *network*, *20*, 13.

Ghahabi, O., & Hernando, J. (2015). Restricted boltzmann machine supervectors for speaker recognition. In *Ieee international conference on acoustics, speech, and signal processing (icassp)* (pp. 4804–4808).

Ghahabi, O., & Hernando, J. (2017, 4). Deep learning backend for single and multisession i-vector speaker recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *25*(4), 807–817.

Ghahabi, O., & Hernando, J. (2018, 1). Restricted boltzmann machines for vector representation of speech in speaker recognition. *Computer Speech & Language*, *47*, 16–29.

Ghalehjegh, S. H., & Rose, R. C. (2015). Deep bottleneck features for i-vector based text-independent speaker verification. In *2015 ieee workshop on automatic speech recognition and understanding (asru)* (pp. 555–560).

Gish, H., & Schmidt, M. (1994). Text-independent speaker identification. *IEEE signal processing magazine*, *11*(4), 18–32.

Gish, H., Siu, M. H., & Rohlicek, R. (1991). Segregation of speakers for speech recognition and speaker identification. In *Ieee international conference on acoustics, speech, and signal processing (icassp)* (pp. 873–876).

Guzewich, P., & Zahorian, S. A. (2017). Improving speaker verification for reverberant conditions with deep neural network dereverberation processing. In *Interspeech* (pp. 171–175).

Hansen, J. H., & Hasan, T. (2015). Speaker recognition by machines and humans: A tutorial review. *IEEE Signal processing magazine*, *32*(6), 74–99.

Hatch, A. O., Kajarekar, S., & Stolcke, A. (2006). Within-class covariance normalization for svm-based speaker recognition. In *Ninth international conference on spoken language processing.*

Hatch, A. O., & Stolcke, A. (2006). Generalized linear kernels for one-versus-all classification: application to speaker recognition. In *2006 ieee international conference on acoustics speech and signal processing proceedings* (Vol. 5, pp. V–V).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

Heigold, G., Moreno, I., Bengio, S., & Shazeer, N. (2016). End-to-end text-dependent speaker verification. In *2016 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 5115–5119).

Heo, H.-s., Jung, J.-w., Yang, I.-h., Yoon, S.-h., & Yu, H.-j. (2017). Joint training of expanded end-to-end dnn for text-dependent speaker verification. In *Interspeech* (pp. 1532–1536).

Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade* (pp. 599–619). Springer.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006, 6). A fast learning algorithm for deep belief nets. *Neural computation*, *18*(7), 1527–1554.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, *313*(5786), 504–507.

India, M., Safari, P., & Hernando, J. (2019). Self Multi-Head Attention for Speaker Recognition. In *Proc. interspeech 2019* (pp. 4305–4309). Retrieved from http://dx.doi.org/10.21437/Interspeech.2019-2616 doi: 10.21437/Interspeech.2019-2616

Isik, Y. Z., Erdogan, H., & Sarikaya, R. (2015). S-vector: A discriminative representation derived from i-vector for speaker verification. In *23rd european signal processing conference (eusipco)* (pp. 2097–2101).

Jati, A., & Georgiou, P. (2017). Speaker2vec: Unsupervised learning and adaptation of a speaker manifold using deep neural networks with an evaluation on speaker segmentation. *Proc. Interspeech 2017*, 3567–3571.

Jorrín, J., García, P., & Buera, L. (2017). Dnn bottleneck features for speaker clustering. *Proc. Interspeech 2017*, 1024–1028.

Jung, J.-W., Heo, H.-S., Yang, I.-H., Shim, H.-J., & Yu, H.-J. (2018). Avoiding speaker overfitting in end-to-end dnns using raw waveform for text-independent speaker verification. *extraction*, *8*(12), 23–24.

Kemp, T., Schmidt, M., Westphal, M., & Waibel, A. (2000). Strategies for automatic segmentation of audio data. In *2000 ieee international conference on acoustics, speech, and signal processing. proceedings (cat. no. 00ch37100)* (Vol. 3, pp. 1423–1426).

Kenny, P. (2005). Joint factor analysis of speaker and session variability: Theory and algorithms. *CRIM, Montreal,(Report) CRIM-06/08-13*, *14*, 28–29.

Kenny, P. (2010). Bayesian speaker verification with heavy-tailed priors. In *Odyssey* (Vol. 14).

Kenny, P., Gupta, V., Stafylakis, T., Ouellet, P., & Alam, J. (2014). Deep neural networks for extracting baum-welch statistics for speaker recognition. In *Proc. odyssey* (pp. 293–298).

Kenny, P., Ouellet, P., Dehak, N., Gupta, V., & Dumouchel, P. (2008). A study of interspeaker variability in speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, *16*(5), 980–988.

Khan, U. (2016). *Speaker tracking system using speaker boundary detection* (Unpublished master's thesis). Universitat Politècnica de Catalunya.

Khan, U., & Hernando, J. (2019). Dnn speaker embeddings using autoencoder pre-training. In *2019 27th european signal processing conference (eusipco)* (pp. 1–5).

Khan, U., & Hernando, J. (2020a). Unsupervised training of siamese networks for speaker verification. *Proc. Interspeech 2020*, 3002–3006.

Khan, U., & Hernando, J. (2020b). The upc speaker verification system submitted to voxceleb speaker recognition challenge 2020 (voxsrc-20). *arXiv preprint arXiv:2010.10937*.

Khan, U., India, M., & Hernando, J. (2019). Auto-encoding nearest neighbor i-vectors for speaker verification. *Proc. Interspeech 2019*, 4060–4064.

Khan, U., India, M., & Hernando, J. (2020). I-vector transformation using k-nearest neighbors for speaker verification. In *Icassp 2020 - 2020 ieee international conference on acoustics, speech and signal processing (icassp)* (p. 7574-7578). doi: 10.1109/ICASSP40776.2020.9053504

Khan, U., Safari, P., & Hernando, J. (2018). Restricted Boltzmann Machine Vectors for Speaker Clustering. In *Proc. iberspeech* (pp. 10–14).

Khan, U., Safari, P., & Hernando, J. (2019). Restricted boltzmann machine vectors for speaker clustering and tracking tasks in tv broadcast shows. *Applied Sciences*, *9*(13), 2761.

Khoury, E., El Shafey, L., Ferras, M., & Marcel, S. (2014). Hierarchical speaker clustering methods for the nist i-vector challenge. In *Odyssey: The speaker and language recognition workshop* (pp. 254–259).

Kotti, M., Moschou, V., & Kotropoulos, C. (2008). Speaker segmentation and clustering. *Signal processing*, *88*(5), 1091–1124.

Kubala, F., Jin, H., Matsoukas, S., Nguyen, L., Schwartz, R., & Makhoul, J. (1997). The 1996 bbn byblos hub-4 transcription system. In *Proceedings of the 1997 darpa speech recognition workshop* (pp. 90–93).

Kudashev, O., Novoselov, S., Pekhovsky, T., Simonchik, K., & Lavrentyeva, G. (2016). Usage of dnn in speaker recognition: advantages and problems. In *International symposium on neural networks* (pp. 82–91).

Kwon, S., & Narayanan, S. (2005). Unsupervised speaker indexing using generic models. *IEEE transactions on speech and audio processing*, *13*(5), 1004–1013.

Larcher, A., Bonastre, J. F., Fauve, B. G. B., Lee, K. A., Lévy, C., Li, H., ... Parfait, J. Y. (2013). Alize 3.0-open source toolkit for state-of-the-art speaker recognition. In *Interspeech* (pp. 2768–2772).

Lee, H., Pham, P., Largman, Y., & Ng, A. Y. (2009). Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems* (pp. 1096–1104).

Lei, Y., Scheffer, N., Ferrer, L., & McLaren, M. (2014a). A novel scheme for speaker recognition using a phonetically-aware deep neural network. In *Ieee international conference on acoustics, speech, and signal processing (icassp)* (pp. 1695–1699).

Lei, Y., Scheffer, N., Ferrer, L., & McLaren, M. (2014b). A novel scheme for speaker recognition using a phonetically-aware deep neural network. In *2014 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 1695–1699).

Liu, Y., Qian, Y., Chen, N., Fu, T., Zhang, Y., & Yu, K. (2015, 10). Deep feature for text-dependent speaker verification. *Speech Communication*, *73*, 1–13.

Lozano-Diez, A., Silnova, A., Matejka, P., Glembek, O., Plchot, O., Pesan, J., ... Gonzalez-Rodriguez, J. (2016). Analysis and optimization of bottleneck features for speaker recognition. In *Odyssey* (pp. 352–357).

Lu, L., Jiang, H., & Zhang, H. (2001). A robust audio classification and segmentation method. In *Proceedings of the ninth acm international conference on multimedia* (pp. 203–211).

Lu, L., & Zhang, H. J. (2002). Speaker change detection and tracking in real-time news broadcasting analysis. In *Proceedings of the tenth acm international conference on multimedia* (pp. 602–610).

Luque, J. (2012). *Speaker diarization and tracking in multiple-sensor environments* (Unpublished doctoral dissertation). Department of Signal Theory and Communications, Universitat Politècnica de Catalunya, Spain.

Maaten, L. V. d., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, *9*(Nov), 2579–2605.

McLaren, M., Lei, Y., & Ferrer, L. (2015). Advances in deep neural network approaches to speaker recognition. In *2015 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 4814–4818).

Moattar, M. H., & Homayounpour, M. M. (2012). A review on speaker diarization systems and approaches. *Speech Communication*, *54*(10), 1065–1103.

Mohamed, A.-r., Yu, D., & Deng, L. (2010). Investigation of full-sequence training of deep belief networks for speech recognition. In *Eleventh annual conference of the international speech communication association.*

Nadeu, C., Macho, D., & Hernando, J. (2001). Time and frequency filtering of filter-bank energies for robust hmm speech recognition. *Speech Communication*, *34*(1-2), 93–114.

Nagrani, A., Chung, J. S., Xie, W., & Zisserman, A. (2019). Voxceleb: Large-scale speaker verification in the wild. *Computer Science and Language*.

Nagrani, A., Chung, J. S., & Zisserman, A. (2017). Voxceleb: a large-scale speaker identification dataset. In *Interspeech.*

Navratil, J., & Ramaswamy, G. N. (2003). The awe and mystery of t-norm. In *Eighth european conference on speech communication and technology.*

Novoselov, S., Pekhovsky, T., Kudashev, O., Mendelev, V. S., & Prudnikov, A. (2015). Non-linear plda for i-vector speaker verification. In *Sixteenth annual conference of the international speech communication association.*

Novoselov, S., Pekhovsky, T., & Simonchik, K. (2014). Stc speaker recognition system for the nist i-vector challenge. In *Odyssey: The speaker and language recognition workshop* (pp. 231–240).

Novoselov, S., Pekhovsky, T., Simonchik, K., & Shulipa, A. (2014). Rbm-plda subsystem for the nist i-vector challenge. In *Fifteenth annual conference of the international speech communication association.*

Novoselov, S., Shulipa, A., Kremnev, I., Kozlov, A., & Shchemelinin, V. (2018). On deep speaker embeddings for text-independent speaker recognition. In *Proc. odyssey 2018 the speaker and language recognition workshop* (pp. 378–385).

Oglesby, J., & Mason, J. (1989). Speaker recognition with a neural classifier. In *1989 first iee international conference on artificial neural networks,(conf. publ. no. 313)* (pp. 306–309).

Oglesby, J., & Mason, J. (1990). Optimisation of neural models for speaker identification. In *International conference on acoustics, speech, and signal processing* (pp. 261–264).

Okabe, K., Koshinaka, T., & Shinoda, K. (2018). Attentive statistics pooling for deep speaker embedding. *Proc. Interspeech 2018*, 2252–2256.

Pawar, R., Kajave, P., & Mali, S. (2005). Speaker identification using neural networks. In *Iec (prague)* (pp. 429–433).

Pekhovsky, T., Novoselov, S., Sholohov, A., & Kudashev, O. (2016). On autoencoders in the i-vector space for speaker recognition. In *Odyssey* (pp. 217–224).

Phan, F., Micheli-Tzanakou, E., & Sideman, S. (2000). Speaker identification using neural networks and wavelets. *IEEE Engineering in Medicine and Biology Magazine*, *19*(1), 92–101.

Prince, S. J. (2012). *Computer vision: models, learning, and inference.* Cambridge University Press.

Prince, S. J., & Elder, J. H. (2007). Probabilistic linear discriminant analysis for inferences about identity. In *2007 ieee 11th international conference on computer vision* (pp. 1–8).

rahman Chowdhury, F. R., Wang, Q., Moreno, I. L., & Wan, L. (2018). Attention-based models for text-dependent speaker verification. In *2018 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 5359–5363).

Ravanelli, M., & Bengio, Y. (2018). Speaker recognition from raw waveform with sincnet. In *2018 ieee spoken language technology workshop (slt)* (pp. 1021–1028).

Reynolds, D. A., Quatieri, T. F., & Dunn, R. B. (2000). Speaker verification using adapted gaussian mixture models. *Digital signal processing*, *10*(1-3), 19–41.

Reynolds, D. A., & Rose, R. C. (1995). Robust text-independent speaker identification using gaussian mixture speaker models. *IEEE transactions on speech and audio processing*, *3*(1), 72–83.

Richardson, F., Reynolds, D., & Dehak, N. (2015, 10). Deep neural network approaches to speaker and language recognition. *IEEE Signal Processing Letters*, *22*(10), 1671–1675.

Roseberg, E., Delong, J., Lee, C., Juang, B., & Soong, F. (1992). The use of cohort normalized scores for speaker recognition. *Proceedings of the InternationalCon-*

ference on SpokenLanguage Processing. Banff, A lberta: U niversity ofA lberta, 599–602.

Rouvier, M., Bousquet, P.-M., & Favre, B. (2015). Speaker diarization through speaker embeddings. In *2015 23rd european signal processing conference (eusipco)* (pp. 2082–2086).

Safari, P., Ghahabi, O., & Hernando, J. (2015). Feature classification by means of deep belief networks for speaker recognition. In *23rd european signal processing conference (eusipco)* (pp. 2117–2121).

Safari, P., Ghahabi, O., & Hernando, J. (2016). From features to speaker vectors by means of restricted boltzmann machine adaptation. In *Odyssey 2016-the speaker and language recognition workshop* (pp. 366–371).

Sayoud, H., & Ouamour, S. (2010, 10). Speaker clustering of stereo audio documents based on sequential gathering process. *Journal of Information Hiding and Multimedia Signal Processing*, *4*, 344–360.

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 815–823).

Schulz, H., & Fonollosa, J. A. R. (2009). A catalan broadcast conversational speech database. In *Joint sig-il/microsoft workshop on speech and language technologies for iberian languages* (pp. 27–30).

Schwartz, R., Roucos, S., & Berouti, M. (1982). The application of probability density estimation to text-independent speaker identification. In *Icassp'82. ieee international conference on acoustics, speech, and signal processing* (Vol. 7, pp. 1649–1652).

Senior, A., Sak, H., & Shafran, I. (2015). Context dependent phone models for lstm rnn acoustic modelling. In *2015 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 4585–4589).

Senoussaoui, M., Dehak, N., Kenny, P., Dehak, R., & Dumouchel, P. (2012). First attempt of boltzmann machines for speaker verification. In *Odyssey 2012-the speaker and language recognition workshop.*

Shon, S., Mun, S., Kim, W., & Ko, H. (2017). Autoencoder based domain adaptation for speaker recognition under insufficient channel information. *arXiv preprint arXiv:1708.01227*.

Shon, S., Tang, H., & Glass, J. (2018). Frame-level speaker embeddings for text-independent speaker recognition and analysis of end-to-end model. In *2018 ieee spoken language technology workshop (slt)* (pp. 1007–1013).

Siegler, M. A., Jain, U., Raj, B., & Stern, R. M. (1997). Automatic segmentation, classification and clustering of broadcast news audio. In *Proc. darpa speech recognition workshop* (pp. 97–99).

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sinha, R., Tranter, S. E., Gales, M. J., & Woodland, P. C. (2005). The cambridge university march 2005 speaker diarisation system. In *Ninth european conference on speech communication and technology.*

Snyder, D., Garcia-Romero, D., Povey, D., & Khudanpur, S. (2017). Deep neural network embeddings for text-independent speaker verification. *Proc. Interspeech 2017*, 999–1003.

Snyder, D., Garcia-Romero, D., Sell, G., Povey, D., & Khudanpur, S. (2018). X-vectors: Robust dnn embeddings for speaker recognition. In *Ieee international conference on acoustics, speech, and signal processing (icassp)* (pp. 5329–5333).

Stafylakis, T., Kenny, P., Senoussaoui, M., & Dumouchel, P. (2012a). Plda using gaussian restricted boltzmann machines with application to speaker verification. In *Thirteenth annual conference of the international speech communication association.*

Stafylakis, T., Kenny, P., Senoussaoui, M., & Dumouchel, P. (2012b). Plda using gaussian restricted boltzmann machines with application to speaker verification. In *Thirteenth annual conference of the international speech communication association.*

Stafylakis, T., Kenny, P., Senoussaoui, M., & Dumouchel, P. (2012c). Preliminary investigation of boltzmann machine classifiers for speaker recognition. In *Odyssey 2012-the speaker and language recognition workshop.*

Sturim, D. E., & Reynolds, D. A. (2005). Speaker adaptive cohort selection for tnorm in text-independent speaker verification. In *Proceedings.(icassp'05). ieee international conference on acoustics, speech, and signal processing, 2005.* (Vol. 1, pp. I–741).

Sturim, D. E., Reynolds, D. A., Singer, E., & Campbell, J. P. (2001). Speaker indexing in large audio databases using anchor models. In *2001 ieee international conference on acoustics, speech, and signal processing. proceedings (cat. no. 01ch37221)* (Vol. 1, pp. 429–432).

Tan, Z., & Mak, M.-W. (2017). i-vector dnn scoring and calibration for noise robust speaker verification. In *Interspeech* (pp. 1562–1566).

Tranter, S. E., & Reynolds, D. A. (2006, 9). An overview of automatic speaker diarization systems. *IEEE Transactions on audio, speech, and language processing*, *14*(5), 1557–1565.

van Leeuwen, D. A. (2010, 6). Speaker linking in large data sets. *Proceedings of the Speaker and Language Recognition Odyssey*, 202–208.

Variani, E., Lei, X., McDermott, E., Moreno, I. L., & Gonzalez-Dominguez, J. (2014a). Deep neural networks for small footprint text-dependent speaker verification. In *Ieee international conference on acoustics, speech, and signal processing (icassp)* (pp. 4052–4056).

Variani, E., Lei, X., McDermott, E., Moreno, I. L., & Gonzalez-Dominguez, J. (2014b). Deep neural networks for small footprint text-dependent speaker verification. In *Ieee international conference on acoustics, speech, and signal processing (icassp)* (pp. 4052–4056).

Vasilakakis, V., Cumani, S., Laface, P., & Torino, P. (2013). Speaker recognition by means of deep belief networks. *Proc. Biometric Technologies in Forensic Science*, 52–57.

Vijayasenan, D., Valente, F., & Bourlard, H. (2007). Agglomerative information bottleneck for speaker diarization of meetings data. In *2007 ieee workshop on automatic speech recognition & understanding (asru)* (pp. 250–255).

Villalba, J., Brümmer, N., & Dehak, N. (2017). Tied variational autoencoder backends for i-vector speaker recognition. In *Interspeech* (pp. 1004–1008).

Wang, F., Cheng, J., Liu, W., & Liu, H. (2018). Additive margin softmax for face verification. *IEEE Signal Processing Letters*, *25*(7), 926–930.

Wang, J., Song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., ... Wu, Y. (2014). Learning fine-grained image similarity with deep ranking. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1386–1393).

Willsky, A., & Jones, H. (1976). A generalized likelihood ratio approach to the detection and estimation of jumps in linear systems. *IEEE Transactions on Automatic control*, *21*(1), 108–112.

Yamada, T., Wang, L., & Kai, A. (2013). Improvement of distant-talking speaker identification using bottleneck features of dnn. In *Interspeech* (pp. 3661–3664).

Zadrozny, B., & Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth acm sigkdd international conference on knowledge discovery and data mining* (pp. 694–699).

Zhang, C., & Koishida, K. (2017). End-to-end text-independent speaker verification with triplet loss on short utterances. In *Interspeech* (pp. 1487–1491).

Zhang, S.-X., Chen, Z., Zhao, Y., Li, J., & Gong, Y. (2016). End-to-end attention based text-dependent speaker verification. In *2016 ieee spoken language technology workshop (slt)* (pp. 171–178).

Zheng, R., Zhang, S., & Xu, B. (2006). A comparative study of feature and score normalization for speaker verification. In *International conference on biometrics* (pp. 531–538).

Zhu, Y., Ko, T., Snyder, D., Mak, B., & Povey, D. (2018). Self-attentive speaker embeddings for text-independent speaker verification. In *Interspeech* (pp. 3573–3577).