POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING

Master of Science in Computer Engineering

Master Degree Thesis

Deep Neural Networks for Speaker Verfication

On the extraction of speaker representations with Deep Learning

Supervisors
Dr. Sandro Cumani

Candidato
Salvatore Sarni

Academic Year 2019-2020

Abstract

Speaker identification and speaker verification are the main tasks in the field of speaker recognition. The former involves inferring the speaker of an utterance from a set of possible identities, whereas the latter aims at assessing whether a claimed identity corresponds to the speaker of a given speech segment.

Thanks to the advances in the field of Deep Learning, Deep Neural Networks (DNN) have recently become the state-of-the-art technique for utterance representation in the speaker recognition field. The DNN approach consists in training a neural network to extract speaker embeddings, i.e. fixed dimensional utterance representations that contain speaker-discriminant information. DNN embeddings significantly outperform previous state-of-the-art methods such as i-vectors in terms of verification accuracy.

One of the most effective architectures for speaker embedding extraction is the Time Delay Neural Network (TDNN), which is able to model long range temporal dependencies. In this work we start with an analysis of the effectiveness of TDNNs for the speaker verification task. We also investigate the combination of TDNNs with other well-known architectures, such as Residual (ResNet) and recurrent neural networks, with the aim of improving the verification accuracy and possibly lowering the computational cost of embedding extraction.

Traditionally, speaker embedding networks are trained on a set of background speakers using a multi-class classification paradigm: acoustic features are propagated through the network and aggregated by a pooling layer, which is often employed as, or followed by, the embedding layer. The network output consists of a softmax layer that computes speaker posterior probabilities. The cross-entropy function is used as objective function during training.

Network training requires the extraction of acoustic features for a large number of speech segments. In this work we analize how the growing size, the dataset diversity dataset and the use of augmentation techniques impact the recognition accuracy, and propose effective methods to train the models in scenarios with limited computational resources.

While the cross-entropy approach works well in classification tasks, it might not be the most effective choice to produce information-rich utterance representations for speakers that have not been seen during training. We therefore also analyze different objective functions that are inspired by solutions adopted in the face recogniton field to increase the robustness of the DNN embeddings.

Finally, the standard TDNN pooling layer consists of a simple temporal average of the DNN-transformed acoustic features. In this work, we consider also alternative pooling approaches.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The idea of linking our voice to a specific function in an automated system is becoming more and more popular. Both speech and speaker recognition have become quickly an important part of our lives not only in personal assistant systems, from Siri to Alexa, but also to authenticate the owner of a bank account. Our voice is used not only to answer a question or perform some task but most importantly, in the optic of this work, to identify or authenticate a person.

1.1 Speaker Recognition

The work presented in the next chapters falls in the field of Speaker Recognition. The main concept and objective are to recognize a speaker by his pronounced utterance. The system in charge of performing this operation has to "understand" both physical and behavioral characteristics of the audio sample, to distinguish each speaker. This task can be further divided into speaker identification and speaker verification. Speaker identification systems try to identify a speaker among a given set of known enrolled ones. If the test speaker may not be part of the enrollment speaker collection, this is called an open-set problem. Otherwise, it's a closed-set task. Speaker identification is a 1 vs N match while speaker verification is a 1 vs 1 match. In speaker verification, a speaker tries to authenticate himself asserting his identity, and the system has to verify the similarity of the produced utterance against the enrolled one, accepting or declining the declaration.

1.2 Outline

In this work, we focused on the use of Deep Learning as embeddings extractor technique, in order to solve the text-independent speaker verification problem. Different neural network architectures have been tested as fronted to extract embeddings while Probabilistic Linear Discriminant Analysis (PLDA) [cit PLDA] and Pairwise Support Vector Machine (PSVM) [cit PSVM] are used as backend. In Chapter 2 we provide an introduction to the techniques related to the feature extraction process and the previous state-of-art approach to the speaker recognition problem. The neural network principle and architectures are presented in Chapter 3. Chapter 4 is about the various test, introducing the data and different approaches followed. In Chapter 5 we expose the main technique used to evaluate our classifiers. Finally, conclusions and future work can be found in Chapter 6.

Chapter 2

Background

2.1 Extraction

Dealing with audio signal means dealing with a continuous dimension, which is infeasible and not directly workable with the digital system. The first transformation of the data is needed to obtain discrete features from the raw signal. This step is based on Frequency Analysis.

2.1.1 Sampling, quantization and filtering

The process of transforming an analog signal into a discrete version is called sampling and quantization. First discretization is done in time, this corresponds to multiplying the signal with a sequence of impulse $\sum\limits_k \delta(t - kt_s)$ where t_s is the sampling time. Given the input signal $y(t)$, the sampled version $y_s(k)$ is:

$$y_s(k) = \sum_k [y(kt_s)\delta(t - kt_s)] \tag{2.1}$$

In the frequency domain, where $Y(w)$ is the Fourier transorm of $y(t)$ and $Y_s(w)$ of $y_s(w)$

$$Y_s(\omega) = \frac{1}{t_s} \sum_k Y\left(\omega + \frac{2\pi k}{t_s}\right) \tag{2.2}$$

Humans are mainly sensitive to frequencies lower than 4 kHz and the Nyquist theorem states that a signal is reconstructable if the sampling frequency is at least twice the highest frequency of the original signal. Then a sufficient sampling frequency is 8kHz. Given a finite set of continuous values, quantization is needed to map them to discrete ones. One way to do this is to uniformly divide the input range and assign to each value the index of the corresponding interval. This is what linear quantization does. However, the acoustic signal is highly non-linear and therefore a logarithmic quantization is usually preferred. This means that the logarithm of the acoustic signal is linearly quantized. In practice different functions are used, to deal with the non-definiteness of the logarithm in zero, as the μ–law (used in American communication nets) or the A–law (used in the European communication nets) [3]. Working with telephone speech, usually, values are represented on 8 bits. In practice, a greater number of bits is used to initially perform a linear quantization. One of the laws in figure 2.1 is then used to map them to 8 bits. A first-order

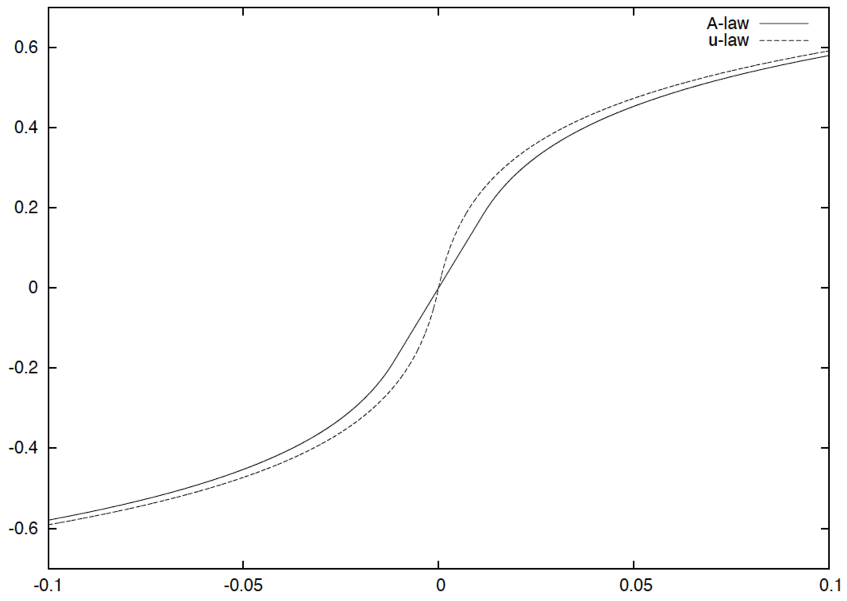


Figure 2.1: The A and μ law

pre-emphasis filter (2.3) is used on the discretized samples to flatten the signal spectrum in the given frequency band

$$H(z) = 1 - az^{-1} \tag{2.3}$$

Equivalent, in time-domain:

$$\hat{Y}(k) = Y(k) - aY(k-1) \tag{2.4}$$

where a is a constat, typically $a = 0.95$ in real applications.

2.1.2 Mel-Frequency Cepstral Coefficients

Mel-Frequency Cepstral Coefficients (MFCC) is the standard representation for the acoustic signal [3] [4]. MFCC provides a short term representation of the signal. The acoustic signal can be considered stationary over periods of the order of milliseconds, therefore it can be split into frames (usually covering about 10ms) which group together a set of samples:

$$X_t(n) = \hat{Y}(M_a t + n) \quad 0 \leq n \leq N_a - 1, 0 \leq t \leq T - 1 \tag{2.5}$$

with N_a as the grouping window size, M_a as the size of the shift, and T is the duration in frames of the signal. Since the grouping in frames distorts the spectrum of the samples of each frame (known as Gibbs phenomenon), we use a Hamming window to reduce the influence of samples near the borders of each frame

$$\hat{X}_t(n) = X_t(n)W(n) \quad 0 \leq n \leq N_a - 1, 0 \leq t \leq T - 1 \tag{2.6}$$

where

$$W(n) = \begin{cases} c + (1 + c)\cos(\frac{\pi n}{N-1} - \frac{\pi}{2}) & if \quad 0 \leq n \leq N - 1 \\ 0 & otherwise \end{cases} \tag{2.7}$$

and $c = 0.54$ in real applications. While a better approximation of the spectrum is available with this approach, the information contained in border samples is penalized, for this reason, the window is shifted only by half its size. This way border samples of a frame can now be found in the middle of either the previous or the next one. At this point, we perform the Fourier Transform over the data of each frame

$$X_f(j) = F(\hat{X}_t(n))(j) \quad 0 \leq n \leq N_a - 1, 0 \leq t \leq T - 1 \tag{2.8}$$

Since humans can not sense phase variations, we are only interested in the amplitude spectrum. We use filters that emulate the human apparatus. The mel scale [3] defines frequency bands to divide the acoustic signal spectrum. For each band, the energy of corresponding samples is evaluated as

$$E_i(f) = \sum_{j=L_i}^{H_i} |X_f(j)|^2 \quad 1 \leq i \leq N_f \tag{2.9}$$

where $E_i(f)$ is the energy of the i–th band, L_i is the lower bound of the corresponding band, H_i is its higher bound and N_f is the number of bands. MFCCs are then obtained computing the Discrete Cosine Transform of the logarithm of the energy parameters $E_i(f)$ so that the i–th MFCC for frame k is given by

$$C_i(k) = \sum_{j=1}^{N_f} \log(E_j(k)) \cos\left[i(j - \frac{1}{2})\frac{\pi}{N_f}\right] \quad 0 \leq i \leq N_f \tag{2.10}$$

The total energy of the frame can be evaluated as

$$E(k) = \sum_{j=1}^{N_f} E_j(k) \tag{2.11}$$

Usually the cepstral parameter $C_0(k)$ is discarded since it carries the same energy information given by $E(k)$. Furthermore, cepstral parameters carry decreasing information as their indices grow, hence high index parameters can be discarded too. The number of cepstral parameters is usually between 12 and 24.

Depstral parameters, MFCC differential counterparts, can be used to better model the acoustic signal. An approximation of the temporal derivative of MFCCs is used to evaluate these parameters. A gain, G, can be used to have similar variance between the set of MFCCs and the set of depstral parameters

$$\Delta\bar{C}_i(k) = G \sum_{j=-N}^{N} j\bar{C}_i(k-j) \quad 1 \leq i \leq p \tag{2.12}$$

and N is half the size of the window used to approximate the derivative. Similarly, differential energy can be evaluated as

$$\Delta E(k) = \sum_{j=-N}^{N} jE(k-j) \tag{2.13}$$

Second-order derivatives of cepstral coefficients can be computed similarly. Finally, combining cepstral parameters and their derivatives we obtain the feature vector

$$\begin{aligned} O_t = \{&\bar{C}_1(t), ..., \bar{C}_p(t), \Delta\bar{C}_1(t), ..., \Delta\bar{C}_p(t), \\ &\Delta\Delta\bar{C}_1(t), ..., \Delta\Delta\bar{C}_p(t), E(t), \Delta E(t), \Delta\Delta E(t)\} \end{aligned} \tag{2.14}$$

2.2 Probabilistic Model

The main problem in speaker recognition (SR) is to extract meaningful characteristics from the feature. One of the first attempts was the Gaussian Mixture Model (GMM), a long time state-of-the-art approach. [5] [6] A more advanced technique is based on the Hidden Markov Model, where the temporal evolution of acoustic features can be modeled. Improvements came with the introduction of Front-End Joint Factor Analysis, which improves the pre-existent GMM model. And finally the pre-Neural Network state of the art, the i-vector.

2.2.1 Gaussian Mixture Model

GMM is a combination of Gaussian probability density functions (pdf). Given a set of samples $X_s = x_1, ..., x_n$ from random variable X with pdf $f(x)$, a GMM can be used to approximate an estimate of $f(x)$. Assuming that the samples are i.i.d., the pdf can be decomposed as

$$p(X_s) = \prod_{i=1}^{n} p(x_i) \tag{2.15}$$

A weighted sum of a set of m multivariate normal distributions can be used to approximate the pdf $p(x)$

$$p(x|M) = \sum_{i=1}^{m} w_i \mathcal{N}(x|\mu_i, \Sigma_i) \tag{2.16}$$

where $p(x|M)$ is the probability of x given the GMM M and w_i are the mixture weights constrained by

$$\sum_{i=1}^{m} w_i = 1 \quad w_i \geq 0 \tag{2.17}$$

$\mathcal{N}(x|\mu_i, \Sigma_i)$ is a normal pdf with mean μ_i and covariance matrix Σ_i:

$$\mathcal{N}(x|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma_i|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i)\right) \tag{2.18}$$

where the dimensionality of input space is denoted by D. The main advantage in using GMM, given enough components in the mixture, is their ability to accurately approximate any probability density function.

Discrete latent variables [6] can be used to represent a GMM, with $Z = [z_1, ..., z_m]$ as a m-dimensional random variable such that $z_i \in \{0, 1\}$ and $\sum_{i=1}^{m} z_i = 1$. The distribution of z_k can be defined in terms of the mixing coefficients w_i as

$$p(z_k = 1) = w_k \tag{2.19}$$

but, only one term of Z is equal to one so

$$p(Z) = \prod_{i=1}^{m} w_i^{z_i} \tag{2.20}$$

We can interpret the realizations of Z as representations of m mutually exclusive states of which only one is active. Then, given the state $z_k = 1$, the conditional probability of random variable X is

$$p(x|z_k = 1) = \mathcal{N}(x|\mu_k, \Sigma_k) \tag{2.21}$$

and the marignal probability of X can be evaulated as

$$p(x) = \sum_k p(z_k = 1)p(x|z_k = 1) = \sum_{k=1}^{m} w_k \mathcal{N}(x|\mu_k, \Sigma_k) \tag{2.22}$$

which has the same form of (2.16)

To estimate GMMs parameters $\theta = \{w_i, \mu_i, \Sigma_i\}$ describing a model M from a sufficiently large amount of data the Maximum-Likelihood (ML) approach is taken [6] [7] [8]:

$$\theta_{ML} = arg \max_\theta P(X_s|\theta) \tag{2.23}$$

Finally, representing a GMM as a latent variable model allows the Maximum-Likelihood estimate to be computed with the Expectation-Maximization algorithm [6] [7].

2.2.2 Hidden Markov Models

Since a GMM considers all frames as independent the temporal information is lost. Hidden Markov Models (HMM) are a possible solution. [9]

Given a set of nodes or states S and a set of edges E, an HMM is a directed graph $G(E, S)$. Each node is connected to any other node, but since we are interested in modeling the temporal evolution of the acoustic features, a simplified model is used. The simplified model or left-to-right model [10] presents only three kinds of edges: self–loop edges, forward edges, and skip edges. Stationary intervals of the acoustic event are represented by the states. Then in order to remain on the same state for a longer period, self-loops are used, while to move to an adjacent state or a non–adjacent state forward-edge and skip-edge can be used. This structure allows imposing a minimum duration to the acoustic event and allows dynamically aligning acoustic segments to the model.

Transition probabilities describe the stochastic process controlling the evolution through the states of an HMM. Given a state active at time t, q_t, we define the probability of being in state S_i at time $t = 1$ as

$$\pi_i = P(q_i = S_i) \quad 1 \leq i \leq N_i \tag{2.24}$$

and the array of initial probabilities $\pi = (\pi_1, ..., \pi_n)$ constrained by

$$\sum_{i=1}^{N} \pi_i = 1 \tag{2.25}$$

The probability of reaching state S_j from state S_i in a single step is $a_{ij} = P(q_{t+1} = S_J | q_t = S_i)$. These probabilities are organized in a matrix

$$A = a_{ij} \quad 1 \leq i, j \leq N \tag{2.26}$$

constrained by

$$a_{ij} \geq 0 \quad \forall(i, j) \tag{2.27}$$

$$\sum_{j=1}^{N} = 1 \forall i \tag{2.28}$$

Following these definitions, the transition at time t will depends only on the state at time $t-1$, no other factors, like time t and observed sequence of features, are taken into account.

Each state is associated to a stochastic function $f_i(x_t)$, representing the probability that features x_t are generated when the system is in S_i, $P(x_t | q_t = S_i)$. In many real speaker recognition systems f_i are GMMs. Usually, in these models, only the GMM weights are state-dependent, while all the states share the mean vectors and the covariance matrix, allowing a reduction in the number of parameters.

2.3 Speaker verification with Latent Variable Models

Probabilistic models of a speaker can be built with latent variables. These variables can represent both speaker identity or channel information, but they also can be used to perform dimensionality reduction.

In speaker verification, we are interested in systems able to assess wheter if an utterance belongs to a target speaker or not. These systems should produce a verification score or log-likelihood [11].

The next introduced frameworks represented state-of-the-art performance before the advent of deep learning representation.

2.3.1 Universal Background Model

In a GMM-only speaker recognition approach enrollment utterances are used to model the distribution of the acoustic features X of a given speaker s, $P(X|s)$. The probability that utterance $\mathcal{X}$ with associated acoustic observations X is spoken by speaker s can be represented by the likelihood of a test utterance evaluated using this distribution. This likelihood can then be compared with the likelihood of non-target speakers. A model of the non-target speaker is hard to define in a GMM basic approach. A possible solution is to build a Universal Background Model (UBM), which is obtained training a single GMM over a large set of utterances from many different speakers pooled together [12] [13]. The UBM can also represent the common characteristics of the acoustic space, then be used to estimate speaker models with Maximum–a–Posteriori (MAP) adaptation [13] [14]. Parameters can be evaluated by the Maximum-Likelihood estimate as

$$\theta_{ML} = arg\max_{\theta} P(X|\theta) \tag{2.29}$$

where θ is the set of parameters of the model. MAP instead, given the prior probability $g(\theta)$ evaluates the parameters as

$$\theta_{MAP} = arg\max_{\theta} P(X|\theta)g(\theta) \tag{2.30}$$

The speaker-independent UBM is used by classical GMM MAP to estimate the speaker-dependent GMMs. Usually, different speakers will share the GMM parameters (weights and covariance matrices), and only the

UBM means are adapted. From now on a GMM supervector is obtained stacking the GMM means, it has shape $CF \times 1$, where C is the number of Gaussian components and F is the dimension of the acoustic feature vector associated to each component.

2.3.2 Factor Analysis Model

The next techniques are based on Factor Analysis, they provide a probabilistic framework to perform MAP estimate of speaker-dependent GMM. These techniques are the foundation of Joint Factor Analysis [15] and subsequently of i-vectors [16].

Utterance statistics

Before introducing Joint Factor Analysis and i-vectors, it is useful to introduce some definitions which will be used next.

Given a set of observations $X(s) = x_1, ..., x_n$ for speaker s, we can obtain the alignment of X over the components of a GMM. This means associating each observation to a single mixture component. The occupation probability, given GMM model parameters w, μ and Σ, is defined as

$$\gamma_{it} = \frac{w_i \mathcal{N}(x_t|\mu_i, \Sigma_i)}{\sum_j w_j \mathcal{N}(x_t|\mu_j, \Sigma_j)} \tag{2.31}$$

We can then define zero-order, first-order and centered–first order Baum Welch statistics [17]

$$N_i(s) = \sum_{t=1}^{n} \gamma_{it} \tag{2.32}$$

$$F_i(s) = \sum_{t=1}^{n} \gamma_{it} x_t \tag{2.33}$$

$$F_{X,i}(s) = F_i(s) - N_i(s)\mu_i \tag{2.34}$$

and the second–order and centered second–order statistics

$$S_i(s) = \sum_{t=1}^{n} \gamma_{it} x_t x_t^T \tag{2.35}$$

$$S_{X,i} = S_i(s) - 2F_i(s)\mu_i + N_i(s)\mu_i\mu_i^T \tag{2.36}$$

Stacking these statistics allows us to vectorize computation. $N(s)$ is a $CF \times CF$ block diagonal matrix whose blocks are the $F \times F$ matrices given by $N_i I$. $F(s)$ and $F_X(s)$ are obtained by stacking respectively vectors $F_i(s)$ and $F_{X,i}(s)$. $S(s)$ and $S(s)$ are block–diagonal matrices whose blocks are respectively the matrices $S_i(s)$ and $S_{X,i}(s)$. These statistics can then be used to approximate the log-likelihood for a given GMM supervector as

$$\begin{aligned}\log P(X) &= \sum_{c=1}^{C} \left[N_c(s) \log \frac{1}{(2\pi)^{\frac{F}{2}} |\Sigma_c|^{\frac{1}{2}}} - \frac{1}{2} \sum_{t|x_t \in X_i(s)} \gamma_{ct} (x_t - \mu_c)^T \Sigma_c^{-1} (x_t - \mu_c) \right] \\ &= \sum_{c=1}^{C} \left[N_c(s) \log \frac{1}{(2\pi)^{\frac{F}{2}} |\Sigma_c|^{\frac{1}{2}}} - \frac{1}{2} tr \left(\Sigma_c^{-1} S_{X,s}(s) \right) \right] \qquad (2.37)\end{aligned}$$

Map Adaptation

JFA models a speaker's utterance, represented by a GMM supervector, as a combination of multiple factors containing information about different properties: speaker, channel, and residual characteristic.

In [15] an interpretation of MAP adaptation in terms of hidden variables was introduced, in which channel factors are ignored. In this model, each utterance h from speaker s can be represented by a supervector

$$M(s) = \mu + Dz(s) \qquad (2.38)$$

where μ is the UBM supervector, D is a diagonal $CF \times CF$ matrix and $z(s)$ are CF speaker–dependent hidden variables with prior distribution following $\mathcal{N}(z|0,1)$

Eigenchannel

The second approach tries to explicitly model inter-session variability. Inter–session variability is assumed to be mostly due to channel effects. In channel effects, we also model the noise-causing differences between enrollment and test utterance. A GMM supervector, for a given recording h, can be expressed as the contribution of two terms

$$g_h(s) = s(s) + c_h(s) \qquad (2.39)$$

where $s(s)$ is a speaker dependent component, and $c_h(s)$ denotes the channel component. In eigenchannel adaptation [18] the channel component lies in a small subspace spanned by the columns of a rectangular matrix U of size $CF \times R_C$, with $R_C \ll CF$:

$$c_h(s) = Ux_h(s) \tag{2.40}$$

The channel effects are represented by a standard normal distributed hidden variable, $x_h(s)$.

Eigenvoice

The last approach assumes that adaptation parameters are shared among utterances from the same speaker s, then an utterance supervector can be represented as

$$M(s) = \mu + Vy(s) \tag{2.41}$$

where V is a rectangular matrix of size $C \times RF$, with $R \ll CF$ [15] and $y(s)$ is a hidden $R_s \times 1$ hidden vector with standard normal distribution. This approach is called Eigenvoice MAP, which describes the utterance supervectors in terms of the speaker characteristics and is more suitable when a small number of speakers is available.

Joint Factor Analysis

The combination of the three is referred to as Joint Factor Analysis (JFA) [15]

$$g = \mu + Vy + Ux + Dz \tag{2.42}$$

where μ is a speaker-independent supervector from UBM, V, U, and D are defined as the eigenvoice, eigenchannel and residual matrix. And the vectors y, x, and z are the speaker, session, and common factors, and each is assumed to be a random variable with distribution $\mathcal{N}(0,1)$.

JFA provides a framework for speaker modeling and scoring of test utterances. However, classical JFA is very demanding from a computational point of view. For this reason, JFA is usually used as a feature extractor.

2.3.3 i-vector

JFA can estimate and compensate channel effects. But it was shown [19] that in channel factors may lie useful information about a speaker, even if the channel factor was assumed to model only channel and noise. For this reason, an adapted version of JFA was proposed in which a single subspace is estimated, assuming most of the useful information lies within [16]. Channel compensation is then deferred to a subsequent stage (Within–Class Covariance Normalization). This new model can be expressed as

$$g(s) = \mu + Tw(s) \tag{2.43}$$

where T is a $CF \times M$ matrix whose columns span the subspace where GMMs live and $w(s)$ represent a GMM in the subspace associated with T. The matrix T contains both speaker and channel information. The MAP point estimate of the posterior of hidden variable w is referred to as i–vector. It's noticeable that i-vectors and eigenvoice share the same formulation, however, while in eigenvoice MAP it is assumed that recordings from the same speakers share the same values for the hidden variable, i-vectors assume different values for each recording.

An i-vector extractor is trained in a similar way to JFA, through Maximum–Likelihood. Assuming that statistics are always computed from the UBM, the training phase can be restricted to T. As in the previous model, the EM algorithm can be used to compute the ML estimate of T [20]. I–vectors represented for a long time the standard in speaker recognition, in fact, the first applications of Deep Learning included the estimation of T.

2.4 i-Vector Backends

In this section we present the backend systems used with i-vectors, the same concepts and methods can be used with different speaker representations and will be used with embeddings obtained in chapter 4.

One of the original score methods proposed in [16, 21] is the cosine distance

$$S_{\cos}(\omega_1, \omega_2) = \frac{\omega_1^t \omega_2}{\parallel \omega_1 \parallel \times \parallel \omega_2 \parallel} \tag{2.44}$$

where ω_1 and ω_2 are the target and test i-vectors. Cosine scoring doesn't use speaker labels. If they are available, Linear Discriminant Analysis (LDA) and Within-Class Covariance Normalization (WCNN) are used alone or in combination as variability compensation techniques to improve accuracy. However more effective techniques exist, like Probabilistic LDA and Support Vector Machine. In the following these techniques are briefly explained in the i-vector case, but as stated before they are appliable for any feature vectors.

Linear Discriminant Analysis

LDA is used to find directions along which within-class variation is minimized and between-class variation is large. Data are then projected on those directions in order to reduce dimensionality and make classification easier. Given S speakers and n i-vectors, n_s i-vectors for each speaker s then the between and within-class covariance matrix are defined as

$$S_b = \frac{1}{n}\sum_{s=1}^{S} n_s(\overline{\omega}_s - \overline{\omega})(\overline{\omega}_s - \overline{\omega})^t \tag{2.45}$$

$$S_w = \frac{1}{n}\sum_{s=1}^{S}\sum_{i=1}^{n_s}(\omega_{s,i} - \overline{\omega})(\omega_{s,i} - \overline{\omega})^t \tag{2.46}$$

where $\omega_{s,i}$ is the ith i-vector of speaker s, ω_s is the speaker dependent mean, obtained from the class of s, and ω is speaker independent mean, obtained on the whole data.

Within-Class Covariance Normalization

WCCN is a normalization technique and it was mainly used for improving the robustness of cosine distance and SVM based speaker recognition [22]. The within-class covariance matrix S_w obtained as in (2.46) is used to computed the WCCN projection matrix, W

$$S_w^{-1} = WW^t \tag{2.47}$$

where the Cholesky factorization of S_w^{-1} is used. Unlike LDA, WCCN doesn't reduce the dimensionality and preserves the original directions. The new i-vectors are then obtained as

$$\hat{\omega} = W^t\omega \tag{2.48}$$

2.4.1 PLDA

Probabilistic Linear Discriminant Analysis (PLDA) is a probabilistic extension of LDA. PLDA was introduced to combine and use the feature extracted through LDA to perform object recognition [23].

Given J utterances each of I individuals and their relative embeddings, PLDA assumes that each embedding can be decomposed as:

$$x_{ij} = \mu + Uh_i + Vw_{ij} + \epsilon_{ij}$$

Two parts can be distinguished: Uh_i depending only on the identity of the speaker, models between-speaker variation. $Vw_{ij} + \epsilon_{ij}$ depending on the particular audio for that speaker, models within-speaker variation, and μ is the global dataset mean.

U contains the basis for the between-speaker subspace while V the within-speaker one. h_i and w_{ij} represent speaker identity and channel effects. The noise and unexplained information are modeled by the term ϵ_{ij}. In this model, we assume that embeddings from the same speaker share the same hidden variable h_i. In its original formulation [24], PLDA assumes Gaussian priors for the latent variables

$$\begin{aligned} h_i &\sim \mathcal{N}(0, I) \\ w_{ij} &\sim \mathcal{N}(0, I) \\ \epsilon_{ij} &\sim \mathcal{N}(0, \Lambda^{-1}) \end{aligned} \tag{2.49}$$

where Λ is a diagonal positive defined matrix. This model is also known as Gaussian PLDA (GPLDA). In [25] the more complex heavy-taled (HPLDA) model was presented, where Gaussian priors are substitute with Student's t distribution

$$\begin{aligned} h_i &\sim \mathcal{N}(0, u_i^{-1} I), \quad u_i \sim \mathcal{G}(n_i/2, n_i/2) \\ w_{ij} &\sim \mathcal{N}(0, u_{ij}^{-1} I), \quad u_{ij} \sim \mathcal{G}(n_{ij}/2, n_{ij}/2) \\ \epsilon_{ij} &\sim \mathcal{N}(0, v_{ij}^{-1} \Lambda^{-1}), \quad v_{ij} \sim \mathcal{G}(/2, /2) \end{aligned} \tag{2.50}$$

$\mathcal{G}(a, b)$ is the Gamma distribution with corresponfing probability density function

$$\mathcal{G}(u|a, b) = \frac{b^a}{\Gamma(a)} u^{a-1} e^{-bu} \tag{2.51}$$

where $\Gamma(a) = \int_0^{+\infty} x^{a-1} e^{-ax} \mathrm{d}x$.

Speaker Verification Likelihood

The PLDA model can be used to evaluate the speaker's identity. Given some test segments of speaker s_t with corresponding i-vectors, or embeddings, and a set of enrolled ones of a known speaker s_e, we can evaluate the likelihood of the observed embeddings under the *same speaker* and *different speaker* hypothesis [25] [26]. Given one enrollment i–vector ϕ_e and one test i-vector ϕ_t the log-likelihood ratio will correspond to

$$l = log\frac{P(\phi_e, \phi_t|H_s)}{P(\phi_e, \phi_t|H_d)} \tag{2.52}$$

where H_s and H_t are the same speaker and different speaker hypotheses. The result will tell if the test segments and enrollment segments are from the same speaker. In case we have n enrollment i-vectors and m test i-vectors the log-likelihood ratio would be

$$l = log\frac{P(\phi_{e_1}, ..., \phi_{e_n}, \phi_{t_1}, ..., \phi_{t_m}|H_s)}{P(\phi_{e_1}, ..., \phi_{e_n}, \phi_{t_1}, ..., \phi_{t_m}|H_d)} \tag{2.53}$$

However, in practice, the mean of the n enrolled and the mean of the m test i-vectors are used to compute the log-likelihood ration as the binary case showed before (2.52). Maximum–likelihood can be used to train the PLDA model through the Expectation–Maximization algorithm [25] [24]

2.4.2 Support Vector Machine

PLDA and HPLDA try to model the i-vector speaker classes, another possible approach is to address directly the problem of discriminating between same speaker and different speaker pairs of utterance, this is what Pairwise SVM does [27, 28].

Given two classes, support vector machine (SVM) [6, 29–31] looks for the hyperplane that separates the classes with maximum margin. In its original formulation SVM is a linear classifier

$$\mathcal{D}(x) = \mathbf{w}^T x + b \tag{2.54}$$

$\mathcal{D}(x)$ define the decision function and the distance of x from the hyperplane can be evaluated as $\frac{\mathcal{D}(x)}{\|\mathbf{w}\|}$. If the classes are not separable, the hyperplane is obtained optimizing a trade-off between points that fall inside the margin and the margin size. The SVM objective function is given by:

$$\min_{\mathbf{w}} \frac{1}{2} \parallel \mathbf{w} \parallel^2 + C \sum_{i=1}^{n} \max(0, 1 - \zeta_i \mathbf{w}^T x_i) \tag{2.55}$$

where the second term is the Hinge Loss evaluated on training data x_i with associated class label $\zeta_i \in \{-1, +1\}$.

When features are not linearly separable, input features are projected into a higher dimensional space where a linear separating hyperplane hopefully exists. To avoid an explicit feature expansion kernel functions can be used to perform scalar product computations into the new feature space. The kernel used in Pairwise SVM is directly derived from the two-covariance method, a particular case of PLDA. Given two i-vector ϕ_1 and ϕ_2, the likelihood–ratio l between same speaker and different speaker hypotheses can be written as

$$\begin{aligned}\log l =& \phi_1^T \Lambda \phi_2 + \phi_2^T \Lambda \phi_1 + \phi_1^T \Gamma \phi_1 + \phi_2^T \Gamma \phi_2 \\ &+ (\phi_1 + \phi_2)^T c + k \end{aligned} \tag{2.56}$$

where Λ and Γ depend on the PLDA model parameters and B and W are the between and within covariance matrix from the two-matrix model [26]. This expresses the log-likelihood ratio as a quadratic form of the two considered i–vectors. Expression (2.56) is linear in Λ and Γ. Thus, these parameters can be interpreted as a separating hyperplane in the expanded feature space [5]. PSVM estimates the values of Λ and Γ that optimize the SVM objective function.

Chapter 3

Neural Network

In the previous chapter, the old state-of-the-art representations have been introduced. Thanks to the advancements in machine learning brought by the surge of novel deep learning techniques, embeddings extracted from the hidden layer of a neural network started to replace i-vectors. In the next section, there is a brief introduction to the neural networks and their functioning. Next, the architectures used in our tests are presented.

3.1 Neural Network

Neural networks are a well known and not so recent tool of computer science. The ambition to mimic our brain architecture has driven the research in this field since Turing and the early stage of computer programming. NN had different periods of fortune and popularity, due to their intrinsic complexity and limited past resource. But in the last period, Deep Learning has sprinted their development and utilization, thanks to new advancements in both hardware and data availability. In DL neural networks become bigger, deeper, and wider, and their ability to model has increased providing impressive results in solving problems in different fields.

3.1.1 Perceptron

NN consists of a set of artificial neurons, inspired by biological ones, called perceptron (Figure 3.1). A perceptron has n inputs $x_1, ..., x_n$ and one output y. Each input is weigthed by $w_1, ..., w_n$. A bias is used, and it can be represented as an additional input $x_0 = 1$ weighted by w_0. The weighted

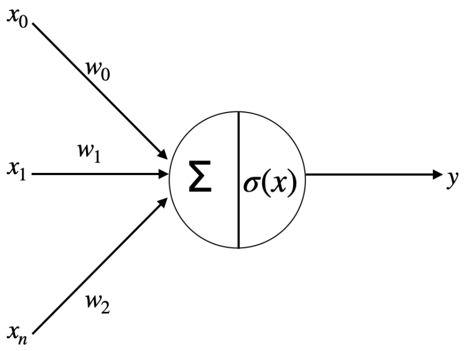


Figure 3.1: A simple perceptron with n inputs and one output.

input is linearly combined and pass through the activation function $\sigma(x)$. Activation function mimics the human neural activity firing a signal when the inputs surpass a certain threshold. A variety of activation functions exists, each with different properties.

3.1.2 Feed Forward Neural Network

A neural network can be seen as a graph, where perceptrons are nodes and can be conceptually organized as layers, which is a set of perceptrons at the same level of the architecture. A Multi-Layer Perceptron [32] is an example of a famous simple architecture, known as Feed-Forward neural network (FFNN). The main feature of the FFNN is the absence of cycles in their graph, fig. 3.2. In the feed-forward architecture, there are multiple layers and they can be identified by their position or level in the network. The *input* and *ouput* layer can be easily recognized as the first and last layer of the network, respectively. The middle ones are called *hidden* layers. The number of hidden layers determines the *depth* of the network. While the input layer contains only the input values, hidden and output layers are composed of multiple perceptrons called *units*, their number defines the *width* of the network. Usually, in speaker recognition, the sigmoid function is used as the activation function of the hidden units

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

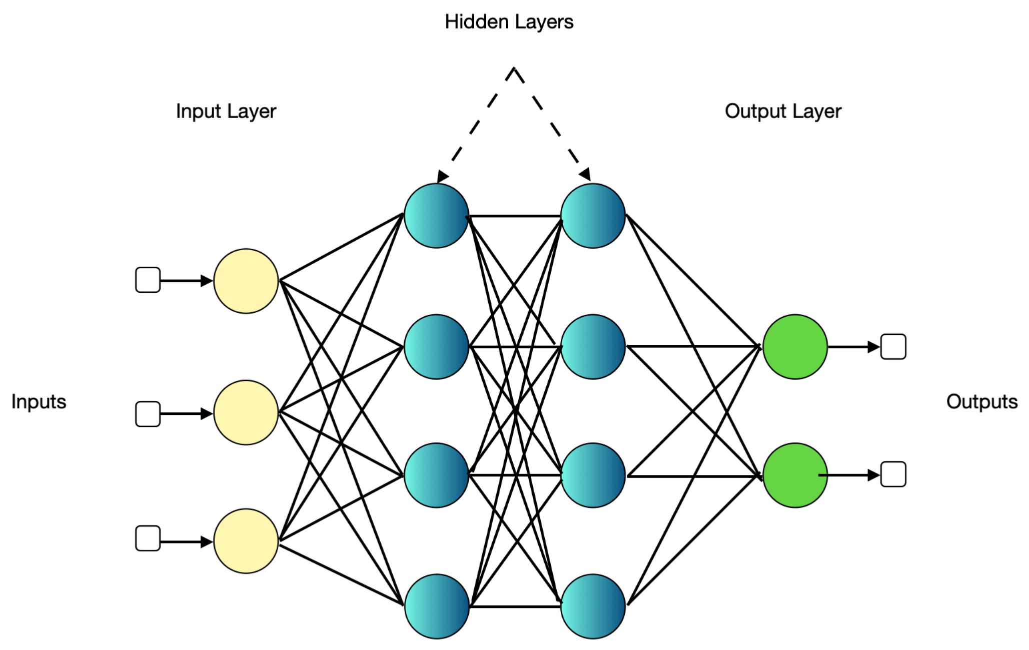


Figure 3.2: A feed forward neural network example

altough in this work we used the ReLU function

$$\sigma(x) = max(0, x) \tag{3.2}$$

The output units have a softmax activation function (3.1.3).

Each unit in a hidden layer is *fully connected* to the units of the next layer. This architecture leads to a fast increase in the number of parameters as the network becomes wider and deeper. However, with enough hidden layers and units with the right activation function, the FFNN can approximate any kind of separation surface.

Given

- the node, or unit, n_i
- the set of input $S(i)$
- the weights connecting a node to a previous one $w_{i,j}$
- the output of previous nodes $o_j(x)$

then each unit evaluates its input value as

$$n_i(t) = \sum_{j \in S(i)} w_{i,j} o_j(x) + b_i \tag{3.3}$$

where b_i is a fixed bias. The ouptut will be evaluated as

$$o_i(x) = \sigma(n_i(x)) \tag{3.4}$$

with $\sigma(x)$ defined in (3.1) or (3.2).

3.1.3 Training

For this work, we focused on supervised training. In supervised training the expected output for each input is available, that is we have a set of training patterns $x_p = (x_{1_p}, ..., x_{N_p})$, with a corresponding set of output vector $t_p = (t_{1_p}, ..., t_{M_p})$, where N and M are the input and output units. Given the network function $f(x_p; \Theta) = o_p$, where Θ are the network parameters, the goal of the training phase is to minimize the error between the predicted output and the given ones. At this point a loss function (3.5) is introduced to achieve this.

$$J(\Theta) = \sum_i loss(f(x_p, \Theta), t_p) \tag{3.5}$$

A loss function maps the discrepancy between the function $f(x_p; \Theta)$ and the target value t_p to a single value of $\mathbb{R}$. The training algorithm used with feedforward networks goes under the name of backward propagation and it minimizes the loss function using gradient descent. The weights are updated, in an iterative way, with the gradient evaluated in the space of the parameters

$$\Theta \leftarrow \Theta - \eta \frac{\partial J(\Theta)}{\partial \Theta} \tag{3.6}$$

where η is the learning rate.

Backpropagation

To compute the gradient with respect to the weights in an efficient way backpropagation is used, this algorithm backpropagates the error from the loss, updating the weights of the hidden layers. The backpropagation algorithm allows obtaining the gradient for every weight through a recursive equation

$$\frac{\partial J(\Theta)}{\partial w^l_{j,k}} = \delta^l_j y^{(l-1)}_k \tag{3.7}$$

with

$$\delta_i^{(l)} = \begin{cases} \frac{\delta J(\Theta)}{\delta y_j^{(l)}} \cdot \sigma_l'(z_j^{(l)}), & \text{if } l \text{ is the output layer} \\ \left(\sum_{i=1}^{q} \delta_j^{l+1} \cdot w_{i,j}^{(l+1)}\right) \cdot \sigma_l'(z_j^{(l)}), & \text{if } l \text{ is a hidden layer} \end{cases}$$

Where $y_j^{(l)}$ describes the output of unit j in layer l and q is the size of units in layer $l+1$. σ_l is the activation function of layer l and $z_j^{(l)} = \sum_{i=0}^{K} w_{i,j}^{(l)} \cdot y_i^{(l-1)}$ is the activation value from unit j in layer l. To obtain these values the forward propagation must be computed. Once the gradient is obtained, gradient descent can be applied

$$w_{j,k}^{(l)} \leftarrow w_{j,k}^{(l)} - \eta \cdot \frac{\delta J(\Theta)}{\delta w_{j,k}^{(l)}} \tag{3.8}$$

Loss Function

The backpropagation algorithm needs a loss function, that gives an error to minimize. Different loss functions exist depending on the problem. Since we want to classify the speaker cross-entropy (CE) is used. CE calculates a loss between two distribution, this means we need to encode the network output in a distribution. *Softmax activation function* is usually adopted in the output layers. This allow to interpret the network outputs as posterior class probabilities $P(c_i|x_t)$, for class c_i given the observed feature vector x_t. Given K classes and output vector $z = [z_1, ..., z_K]^T$ the softmax function is defined as

$$\sigma_j(z) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \quad \text{for } j = 1, ..., K \tag{3.9}$$

The target label is encoded as a vector $\mathbf{p}$, also called a one-hot vector, in which only one entry is 1 and all the other are 0. The classes are enumerated and each one corresponds to an entry in the vector, class i is represented in vector $\mathbf{p}$ at index i. The two vectors, targets, and outputs represent a probability distribution, in fact, the sum of all the entries is 1 and all the entries are ≥ 0. We can then define the CE loss function

$$L(p, q) = -\sum_{i=0}^{n} p_i \cdot ln(q_i) \tag{3.10}$$

Softmax and cross-entropy loss work well in a classification context, but not necessarily producing good encoding, for this reason, we also tested some different loss functions (Section 4.3).

3.2 Deep Learning and i-vector

Deep Learning can be used to solve the entire end-to-end recognition process, and both in the frontend or backend of the speaker recognition system. In this work, we focused on DL as the frontend of our process.

As introduced in Section 2.3.3 in the i-vector approach there are different steps, Baum-Welch statistics collection, extraction, and PLDA backend. However, using DNN in place of GMM in order to compute the Baum-Welch statistics or using bottleneck features in addition to conventional spectral features, a substantial improvement can be achieved [33] [34]. Usually, a bottleneck layer is a much smaller layer, respect to the other hidden layers, before the last hidden layer, and its features are called bottleneck features. In order to extract BW statistics and bottleneck features, a network (Figure 3.3) has to be trained for acoustic modeling in Automatic Speech Recognition (ASR). ASR feature vectors, usually log banks filter energies, are used as input and output layers represent the acoustic classes. Typically these are the states of the Hidden Markov Model (HMM) in ASR.

Then given the DNN acoustic model, the statistics in (2.32) and (2.33)

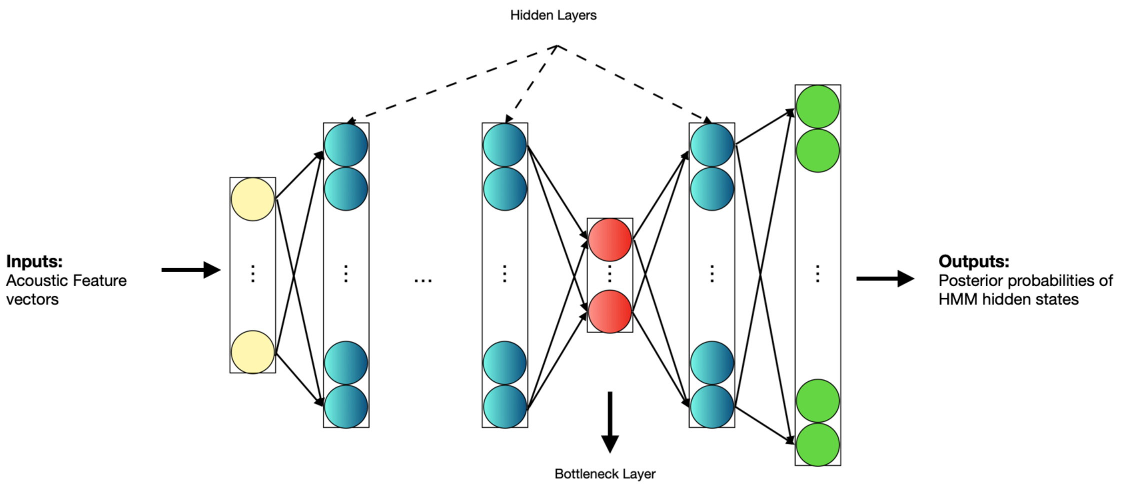


Figure 3.3: A DNN architecture used for acoustic modeling and bottleneck feature extraction

become

$$N_i(s) = \sum_{t=1}^{n} p(o_i|x_t) \tag{3.11}$$

$$F_i(s) = \sum_{t=1}^{n} p(o_i|x_t)\hat{x}_t \tag{3.12}$$

where $p(o_i|x_t)$ is the posterior probability of k-th output unit given the ASR feature vector x_t and $\hat{x}_t$ is the speaker feature vector which can differ from x_t. Despite the improvement in respect to the i-vector approach, this method has some disadvantages. In fact, the computational cost of the statistics increases considerably using DNN. Also, the number of the output units is usually much higher than the number of Gaussian mixtures in the GMM leading to greater complexity in both **T** matrix training and i-vector extraction. For this reason, another use of DNN in frontend has rapidly become the new state-of-art. In this approach, the i-vector is substituted by a low dimensional vector, often known as speaker embeddings.

3.2.1 Embeddings

Embeddings are relatively low dimensional vectors representing an input in a new space. The objective of the embeddings representation is to map similar objects close together in the embeddings space. Embeddings can be used to represent movies in recommendation systems, or to condense information from a text, or as in our case to extract speaker-related information from raw audio data [35, 36]. Embeddings are extracted from one of the hidden layers of the network, where the number of units d defines the dimension of the embedding space. The network is trained as usual and the hidden units learn how to represent the input in the d-dimensional embedding space to optimize the final objective. In speaker recognition, DL networks are trained to classify speakers. [36, 37] The inputs are the acoustic features of a speech segment, which is associated with a label representing the speaker's identity. The objective of training a network to distinguish between different speakers, is to allow it to project similar speakers near to each on the embeddings space. This means that the same network can be used to extract embeddings from a speaker that was not seen during the training phase.

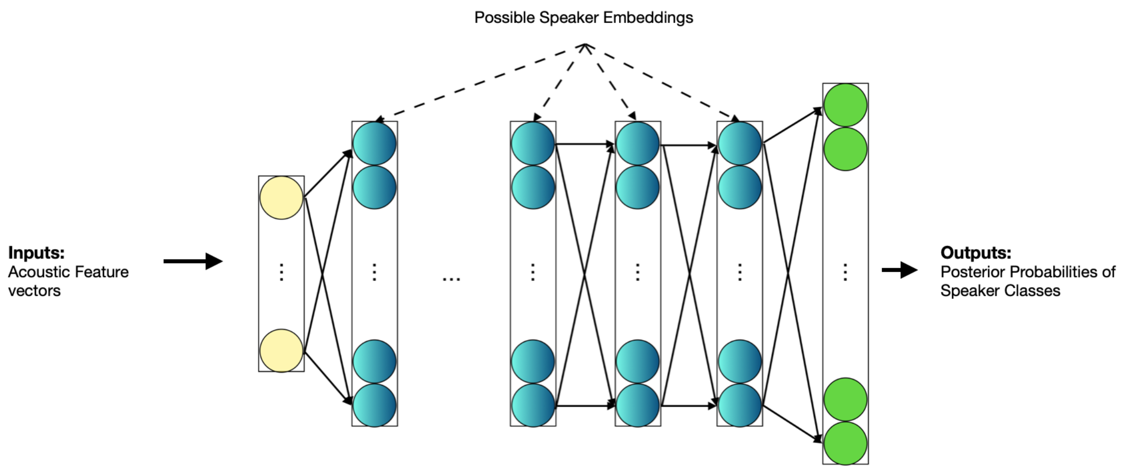

Possible Speaker Embeddings
Inputs:
Acoustic Feature vectors
Outputs:
Posterior Probabilities of Speaker Classes

Figure 3.4: A DNN architecture used for embedding extraction

3.3 Architectures

The set of activation functions, the number of neurons in each layer as well the number of layers define the network architecture. A variety of neural network architectures exist, but given the temporal related nature of the data in this work, only a subset can be exploited. The goal of an automatic system is to build a speaker representation from speech. This means the system has to deal with both short term related characteristics and long term acoustic dependencies. For this reason, we focused on Time Delay and Recurrent architecture, showed in this section, and some combinations involving Residual networks [cit res net] , which will be presented in Chapter Chapter 4.

3.3.1 Time Delay Neural Network

To be effective a feed-forward network has to be complex enough to capture the nonlinearity of the data. Given the nature of speech data, it should be able to represent and capture the time-dependant relationship of spectral coefficients and higher-level features. The network should also learn time-invariant abstractions or features from the data. Finally, to let the network be able to model the input in some encoded form, the number of parameters should be small compared to the training set dimension. Time Delay Neural Network (TDNN) was introduced explicitly satisfying these constraints. [38]

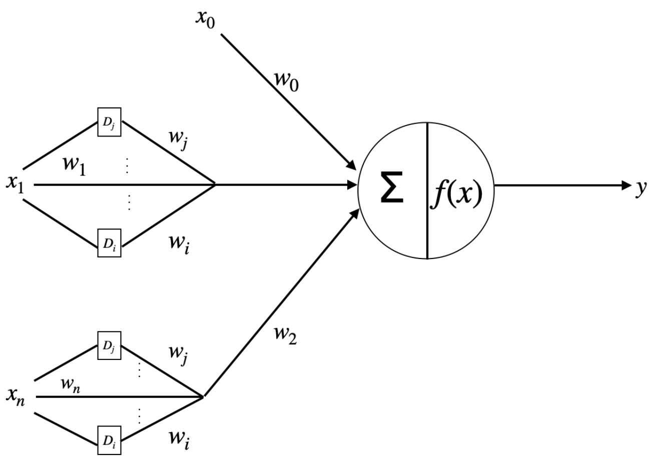


Figure 3.5: The TDNN unit cell.

Differently from Figure 3.1, in the TDNN a delay D_i is introduced on each input. The units at each layer will not receive the outputs of the activations, or features, of the layer below but from a subset of units and its context. Given the feature x_t, a *moving window* $[x_{t-m}, ..., x_t, ..., x_{t+n}]$ is input into the network. Each neuron will process the same features, in different windows, with different sets of weights, as shown in 3.5.

3.3.2 Recurrant Neural Network and LSTM

Unlike a feedforward neural network, like a deep one, recurrant neural network (RNN) has cycles feeding the current layer with inputs from the previous time step. This architecture introduces a state variable, storing old information and enabling the network to learn from a long temporal context in contrast to the static window presented to the units of a feed-forward NN. Long Short-Term Memory (LSTM) has been introduced to resolve some problems related to the training phase of the classical RNN architecture [39]. While RNN uses a single state variable to store memory-related information, LSTM uses two state terms to address both long and short memory.

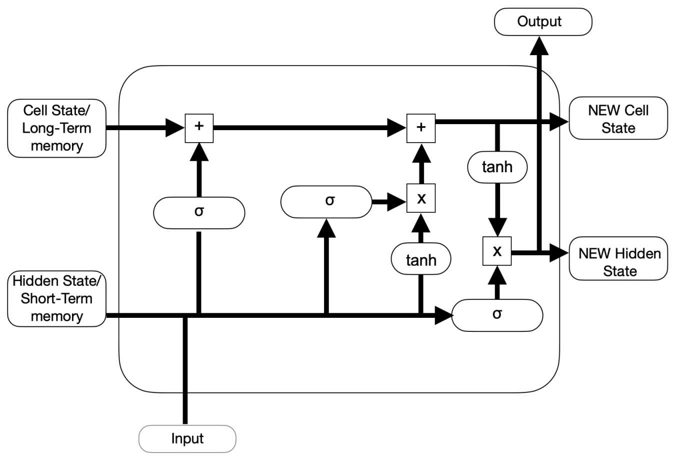


Figure 3.6: The inside architecture of an LSTM cell

LSTM architecture

In Figure 3.6 the internal structure of an LSTM cell is shown. Three zones can be distinguished, called gates. In the middle of the figure, there is the so-called input gate. This filter decides the new information to be stored in the long term memory or cell state

$$i_1 = \sigma(W_{i_1} * (H_{(t-1)}, x_t) + bias_{i_1})$$
$$i_2 = \tanh(W_{i_2} * (H_{(t-1)}, x_t) + bias_{i_2})$$
$$i_{input} = i_1 * i_2$$

The first gate in the picture is called forget gate, which filters information to be discarded from the cell state using the current short memory, or hidden state, and input

$$C_t = C_{t-1} * \sigma(W_{forget} * (H_{t-1}, x_t) + bias_{forget}) + i_{input}$$

The last one is called the output gate, all the fresh updated states are used with the input to produce the new short memory state

$$O_1 = \sigma(W_{o_1} * (H_{t-1}, x_t) + bias_{o_1})$$
$$O_2 = \tanh(W_{o_2} * C_t + bias_{o_2})$$
$$H_t, O_t = O_1 * O_2$$

3.3.3 Residual Neural Network

In the definition of deep learning, the number of layers of the architecture plays a fundamental role. However, at some point stacking more hidden layers will not increase accuracy. In fact, various problems arise in deep architectures, like vanishing or exploding gradient. But even if some solutions have been adopted, when deep architectures are able to converge accuracy gets saturated and also starts to degrade. This is known as the degradation problem, referred also as overfitting.

This problem has been addressed introducing a deep residual learning framework [40]. Traditionally given the input x, a neural network block tries to learn the true distribution $H(x)$. The difference between them is also called the residual

$$R(x) = H(x) - x \tag{3.13}$$

That can be rearranged as

$$H(x) = R(x) + x \tag{3.14}$$

The block is trying to learn $H(x)$ but since the identity connection x it will try to learn $R(x)$, hence *residual network*. The formulation in 3.14 is obtained in a feed-forward neural network with "shortcut connections". As shown in 3.7, shortcut simply skip one or more layers, performing

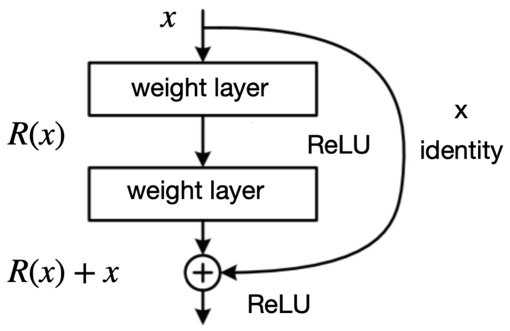


Figure 3.7: Single residual block

an identity mapping that is added to the output of the stacked layers. Residual Neural Networks were implemented to solve image recognition problems, but this architecture became a general approach in different fields. In our test, we used the standard resnet34 architecture, but we also tried some adaptation to work with audio data (Section 4.2.2)

Chapter 4

Experiments

In this chapter, we present the methodologies used to implement a speaker verification pipeline. First, we introduce the training data used in our tests. Then we present the actual network implementation, and some training strategies directly inspired by image recognition.

4.1 Data

In this work, we used different datasets to train the classifiers from which embeddings are extracted. Each dataset consists of a collection of different speech segments, of different duration, from different speakers. In the next subsection, a brief description of each dataset is provided.

VoxCeleb

The principal and bigger dataset we use is VoxCeleb (VC), consisting of two dataset VoxCeleb1 and VoxCeleb2. VoxCeleb is a large set of text-independent speech segments collected from youtube videos, with a system based on speech detection and face recognition [41].

Mixer

Mixer is a collection of data consisting of different datasets, Mixer6 and MixerPhone, where the last one contains different versions like Mixer5, Mixer10, and so on. Mixer 6[1] contains native speaker talking for a longer time than VoxCeleb speaker's as noticeable from tab Table 4.1.

Different from VC, this dataset contains interviews, transcript readings, and conversation over telephone speeches.

Table 4.1: Statistics of the datasets used.

	Number of speaker	**Number of file**	**Avg duration**
VoxCeleb1	1,251	22,496	43.59 s
VoxCeleb2	5,994	145,569	46.49 s
Mixer6	594	13,088	388.65 s
MixerPhone	3,269	37,909	106.04 s
SwitchBoard	1,676	24,556	116.68 s

4.1.1 Augmentation

Having a large dataset, and numerous speakers improve the system performance as discussed in Chapter 5. The evaluation data are not seen by the network during the training phase. The network has to learn how to extract characteristics from the available data and be able to do the same for new and unseen data. To make the system more robust augmentation is often used. In augmentation, data are preprocessed and transformed to replicate the original audio in different "scenarios". The replicated data provide additional distorted data and hopefully training the network with them will make it invariant to these distortions and generalize better to unseen data.

There are different ways to perform speech augmentation, in this work we used four different methods

- Music augmentation: a segment of random music is added as a background to the original utterances.

[1]More details on this dataset is available here: https://catalog.ldc.upenn.edu/LDC2013S03

- Noise augmentation: differents noises are added to the original utterances.
- Babble augmentation: a random number of speech segments belonging to different people is added to the background of the original utterance.
- Reverb augmentation: reverberation in different kind of room, from small to large, is simulated for each original utterance.

Music, speech and noises are taken from the corpus MUSAN [42]. Reverberation is done using Room Impulse Response and Noise Database[2].

The effect of the augmentation process on the performance is presented in the results chapter.

4.1.2 Extraction

As in Section 2.1 the raw audio file has to be processed to be used in an automatic system, in this case as an input for our network. We extract a 45-dimensional feature vector from each utterance, by stacking together 18 cepstral, 19 delta, and 8 double-delta parameters. These parameters are obtained by extracting 19 Mel frequency cepstral coefficients every 10 ms and the frame log–energy on a 25 ms Hamming window, and processing them with short time mean and variance normalization using a 3 s sliding window.

Moreover, each utterance is divided into 3 seconds segments that are then used as input. In our first experiment, the extraction of these segments was done on-the-fly. Thanks to the characteristic of the DataSet and DataLoader[3] framework offered by PyTorch this solution was very easy to implement and zeroed the space overhead since no data was saved. But the multiple extractions from a single file, for each segment, was extremely time-consuming. For this reason, we decided to save all the extracted data and use the stored features to form the segments in the training phase. This lets us speed up the training process. Table 4.2 shows the duration of a training epoch using 800 speakers from VoxCeleb1 and a smaller network, compared to the one used in our test.

[2]https://www.openslr.org/28/

[3]https://pytorch.org/docs/stable/data.html

Table 4.2: Training time of one epoch.

On-The-Fly	All Saved	Biggest Saved
15 hours	12 minutes	1 hour

As shown in table 4.2 we also tried a third approach. Since the available space is limited we decided to save only the bigger and longest files, and extract on-the-fly the smaller ones. Analysis of VoxCeleb1, in fact, showed that more than 80% of the time was spent on multiple extractions of the largest files that correspond to 20% of space.

4.2 TDNN

The general architecture of our networks is showed in 4.1. In the next subsection, each component is explained in greater detail.

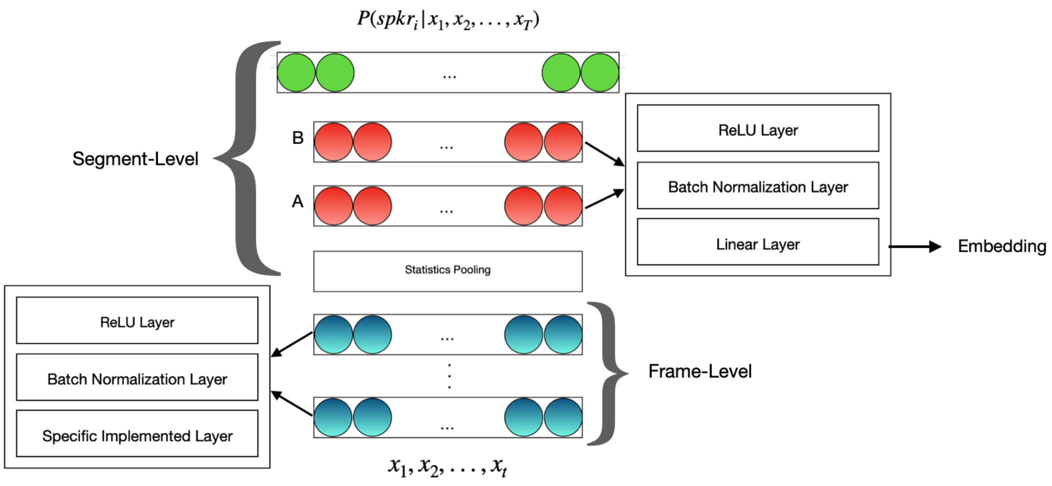

$P(spkr_i|x_1, x_2, \ldots, x_T)$
Segment-Level
B
A
Statistics Pooling
ReLU Layer
Batch Normalization Layer
Linear Layer
Embedding
ReLU Layer
Batch Normalization Layer
Specific Implemented Layer
Frame-Level
$x_1, x_2, \ldots, x_t$

Figure 4.1: The general architecture of the classifiers.

4.2.1 Training

In gradient descent all the training data are used to compute one back-propagation step, this means that from millions of input one update will be made after feed-forwarding the entire training set. This approach is slow both in computation time and update speed. For this reason, some variations have been introduced. The one used in our test is the *Adam*. [43]

The training data is divided into small subsets, called batches, and one backpropagation step is taken at the end of the forwarding of each of them. The weights are updated multiple times based on a small fixed quantity of data. Even if some update will go in the wrong direction, the huge number of updates will go in the right direction and minimize the loss. In our case, the batch is a subset of 128 segments. The shape of a segment is $T \times F$, where T is the number of frames and F is the number of features for each frame. As said before, usually 3 seconds segments are used so $T = 300$, while $F = 45$. The batches will be $128 \times 300 \times 45$. The learning rate, defined in (3.6), is updated in a cyclic way[4] [44]. This approach set two boundaries for the learning rate and change it with a fixed frequency. In our case, 0.0001 is the lower boundary and 0.001 the upper one. The value is increased by half each 1500 iteration. Batch normalization is used between each layer to increase the stability and speed of the network [45].

4.2.2 Frame-Level

At this level, the network has to extract patterns and characteristics working at the frame-level. In this stage, the various architectures introduced in Section 3.3 have been tested. The basic unit of the network consists of three components: a layer, implementing the architecture principles, a Batch Normalization layer and a ReLU activations layer. PyTorch comes with the implementation of some of the most used networks including LSTM and classic ResNet. In the next Section we present our implementation of the TDNN architecture.

[4]Pytorch CyclicLR

TDNN implementation

TDNN, as explained in [38], can be easily implemented with a 1-D convolutional layer, exploiting the dilation parameter. As shown in Figure 4.2 the inputs are convoluted with a matrix of weights taking into account a window of frames, in this specific case frames at time $[t-2, t, t+2]$. As in [46] in our implementation, the single layer of the network is a sequence of three elements: the TDNN layer, or 1-D convolutional layer, a Batch Normalization layer and finally a ReLU layer. This structure is replicated with different time windows as showed in table 4.3.

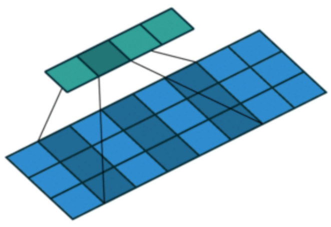


Figure 4.2: A 1-D convolution with dilation representation. Image taken from [1]

Table 4.3: Frame-Level layers of the full TDNN architecture

Layer	Layer Type	Context	Size
1	TDNN+BN+RL	[t-2, t+2]	512
2	TDNN+BN+RL	t	512
3	TDNN+BN+RL	[t-2, t, t+2]	512
4	TDNN+BN+RL	t	512
5	TDNN+BN+RL	[t-3, t, t+3]	512
6	TDNN+BN+RL	t	512
7	TDNN+BN+RL	[t-4, t, t+4]	512
8	TDNN+BN+RL	t	512
9	TDNN+BN+RL	t	1500

ResNet and TDNN combination

The resnet implemented with PyTorch, resnet34, takes 2×2 matrix data. However, our data must be interpreted as a 1-D sequence of vectors. In our experiments resnet34 performance was significantly lower than the simple TDNN, due to this. For this reason, we decided to change the original implementation that uses 2-D convolutional layers, substituting them with the 1-D convolutional layer used in the TDNN. This resulted in a hybrid structure, with better results.

4.2.3 Pooling

The pooling layer takes as input the patterns and features extracted focusing on single frames and aggregates them obtaining a segment representation. Usually, the pooling layer is applied after some convolution layers and its purpose is to reduce the spatial size of the feature maps. The pooling layer has no activation function or weights to learn. This layer helps the network to learn characteristic invariant to the small translations of the input. The first kind of pooling applied in our test is the one used in [46]. The output of the frame-level layers is collapsed over the time dimension. This is done computing the mean and standard deviation of each features over time and stacking them together. In terms of batch dimensionality, given an input of $B \times F \times T$, where B is the batch dimension, F feature dimension and T is the sequence length, the pooling layer transforms it in $B \times 2 * F$.

Learnable Dictionary Encoding

In [47] a new learnable dictionary encoding (LDE) layer has been proposed. The new layer is directly inspired by the GMM supervector encoding. Given a $F \times T$ temporal ordered feature sequence as input, the LDE aggregates them over time, obtaining an utterance level of shape $F \times C$. This new method combines the encoding of a vector with the dictionary learning in a single layer. We recall that for a C component GMM UBM model λ wih $\lambda_c = \{p_c, \mu_c, \Sigma_c\}$, given an utterance with L frames $x = \{x_1, ..., x_L\}$ the 0^{th} order Baum-Welch statistics on the UBM is calculated as

$$N_c = \sum_{t=1}^{L} P(c|x_t, \lambda) \tag{4.1}$$

where c is the GMM component and $P(c|x_t, \lambda)$ is the occupancy probability for x_t on λ_c. This layer introduces two groups of parameters to be learned. The center of the dictionary component, $\mu = [\mu_1, ..., \mu_C]$, and the weights, w, used to assign the features to the components, imitating $P(c|x_t, \lambda)$. Generally, the assignment can be done in two ways:

- Hard-assignment: each feature x_t is assigned with the nearest dictionary component using a binary weight.
- Soft-assignment: the feature is assigned to each component, using non-negative weights.

However, the hard assignment is not differentiable, this means that it is not learnable with gradient descent. Soft-assignment is used in LDE and weights are given by a softmax function

$$w_{tc} = \frac{exp(-s_c \parallel r_{tc} \parallel^2)}{\sum_{m=1}^{C} exp(-s_c \parallel r_{tm} \parallel^2)} \tag{4.2}$$

where r_{tc} is the residual vector defined as $r_{tc} = x_t - \mu_c$ and s_c is the c-th learnable smoothing factor. The encoding can then be obtained as $E = [e_1, ..., e_C]$ where

$$e_c = \sum_{t=1}^{L} e_{tc} = \frac{\sum_{t=1}^{L}(w_{tc} \cdot r_{tc})}{\sum_{t=1}^{L} w_{tc}} \tag{4.3}$$

4.2.4 Embedding layer

The last two layers, before the classification one, are the layers from which embeddings are extracted. These layers follow the structure presented at the frame-level. Three components are stacked together: a linear layer, a Batch Normalization, and a ReLU. Embeddings correspond to the output of the two linear layers Figure 4.1. In the test presented in the next chapter both embedding from layer A and B are taken. However, this structure is not the only one tested, we also tried to have only one layer, or to extract embeddings from the output of the ReLU layer.

4.3 Loss Function

As introduced in Section 3.1.3 we used the standard *softmax loss function*. Softmax function performs well in the classification task, however, it's not assured to provide separable embeddings, that are discriminative for unseen speakers. For this reason, we used some adaptation and reformulation of the loss function introduced to improve the face recognition system. First, we recall the original softmax function definition for a single point, given the input feature x_i and its label y_i

$$L = \frac{1}{N} \sum_{i}^{N} -log(\frac{e^{f_{i,y_i}}}{\sum_{j} e^{f_{i,j}}}) \tag{4.4}$$

where f_j denotes the j-th element of the class score vector, usually the output of the last layer. And N is the number of training samples.

4.3.1 Angular Softmax

To introduce the Angular Softmax [48], also SpeherFace, we have to reformulate the original softmax function. Given the weights of the last layer W then $f_{i,j} = W_j^T x_i + b_j$ and $f_{i,y_i} = W_{y_i}^T x_i + b_{y_i}$ where x_i, W_j and W_{y_i} are the i-training sample, the j-th and y_i-th column of W then 4.4 can be reformulate as

$$L = -\frac{1}{N} \sum_{i}^{N} \log \left(\frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_j e^{W_j^T x_i + b_j}} \right) \tag{4.5}$$

$$= -\frac{1}{N} \sum_{i}^{N} \log \left(\frac{e^{\left\| W_{y_i} \right\| \|x_i\| \cos(\theta_{y_i,i}) + b_{y_i}}}{\sum_j e^{\|W_j\| \|x_i\| \cos(\theta_{j,i}) + b_j}} \right) \tag{4.6}$$

in which $0 \leq \theta_{j,i} \leq \pi$ is the angle between W_j and x_i. In the binary class case this equations define a decision boundary $(W_1 - W_2)x + b_1 - b_2 = 0$ and constraining $\|W_i\| = 1, b_i = 0$ the decision boundary becomes $\cos(\theta_1) - \cos(\theta_2) = 0$. This is knonw as *modified softmax*. In this case, given x belonging to class 1, classifing it correctly requires $\cos(\theta_1) > \cos(\theta_2)$. In [48] it is suggested to require a more stringent boundary: $\cos(m\theta_1) > \cos(\theta_2)$, with $m > 2$. By rewriting 4.6 including the new constrain we finally obtain the Angular softmax

$$L_{ang} = \frac{1}{N} \sum -log \left(\frac{e^{\|x_i\| \phi(\theta_{y_i,i})}}{e^{\|x_i\| \phi(\theta_{y_i,i}) + \sum_{j \neq y_i} \|x_i\| cos(m\theta_{j,i})}} \right) \tag{4.7}$$

where $\phi(\theta_{y_i,i})$ is a function introduced to remove the constrain on $cos(\theta(m\theta_{y_i,i}))$. In this form, in fact, $\theta_{y_i,i}$ has to be in the range of $[0, \frac{\pi}{m}]$.

$$\phi(\theta_{y_i,i}) = (-1)^k cos(m\theta_{y_i,i}) - 2k \tag{4.8}$$

now $\theta_{y_i,i} \in [\frac{k\pi}{m}, \frac{(k+1)\pi}{m}]$ and $k \in [0, m-1]$. The parameter m controls the size of the angular margin, and it can be shown that when $m = 1$ equation (4.7) becomes the *modified softmax*.

4.3.2 Additive Margin Softmax

The second softmax derived function used in our test is the one presented in [49], called Additive Margin or CosFace. In this paper, the authors propose an analogous approach to the Angular Softmax but simplifying the implementation of an angular margin. In fact they proposed to substitute $\phi(\theta)$ defined at 4.8 with $\phi = \cos(\theta) - m$, from here the name. Assuming, as for the modified and angular function, that both W_i and x norms are normalized to 1.

$$L_{ams} = -\frac{1}{N}\sum_{i=1}^{N} \log\left(\frac{e^{s\cdot(\cos\theta_{y_i,i}-m)}}{e^{s\cdot(\cos\theta_{y_i,i}-m)} + \sum_{j\neq y_i} e^{s\cdot\cos\theta_{i,j}}}\right) \tag{4.9}$$

$$= -\frac{1}{N}\sum_{i=1}^{N} \log\left(\frac{e^{s\cdot(W_{y_i}^T x_i - m)}}{e^{s\cdot(W_{y_i}^T x_i - m)} + \sum_{j\neq y_i} e^{s\cdot W_j^T x_i}}\right) \tag{4.10}$$

where s is a scaling factor. It has been noted that the scaling factor s if learnable, converges slowly, and the better results are obtained when fixed at the beginning.

4.3.3 Additive Angular Margin Softmax

The last approach tested enforcing angular margin is the one proposed in [2], ArcFace. The author proposed a new ϕ function in which the margin m is now added directly to the angle between W and f

$$\phi(\theta_{y_i,i}) = \cos(\theta_{y_i,i} + m) \tag{4.11}$$

They also presented an unified framework combining SphereFace, CosFace and ArcFace

$$L = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{s(\cos(m_1\theta_{y_i,i}+m_2)-m_3)}}{e^{s(\cos(m_1\theta_{y_i,i}+m_2)-m_3)} + \sum_{j\neq y_i} e^{s\cdot\cos\theta_{i,j}}} \tag{4.12}$$

Comparison

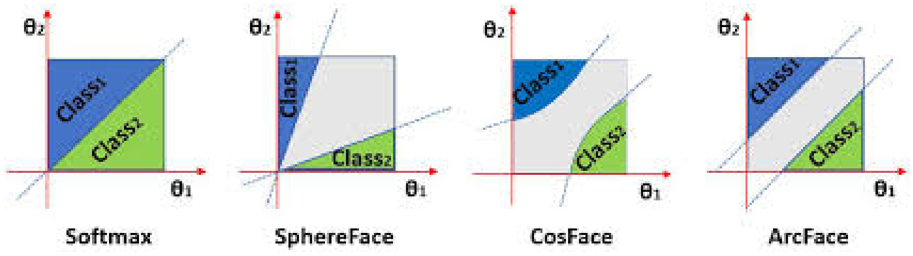


Figure 4.3: Decision boundary in a binary case with different loss functions. Image taken from [2]

4.4 Setup

Given the size of the dataset and the complexity of our training, we use the HPC cluster[5]. Network training was done using a two GPUs available on a cluster node.

Table 4.4: Hardware and software specifications of the workstation used to run the Model Trainer and the Link Evaluator.

Workstation hardware and software specifications	
CPU model	2x Intel Xeon E5-2680 v3 2.50 GHz 12 cores.
GPU model	2x nVidia Tesla K40 - 12 GB - 2880 cuda cores.
OS	Centos 7 - OpenHPC 1.3
SW packages	CUDA toolkit 9.2, Python 3.7.6, PyTorch 1.2

[5]Computational resources provided by hpc@polito (http://hpc.polito.it)

Chapter 5

Evaluation

In Section 2.4 we presented some backend systems used to obtain scores from i-vectors or in general using embeddings. Labels can then be assigned using those scores. The case in which positive samples are classified as negative, or the reverse one, is called misclassification. A threshold divide scores that yield a positive classification from scores that yield a negative one, leading to different kinds of classification mistakes. Calibration is the process through which this threshold is chosen and it depends on which mistake is more costly for a given target application.

5.1 Errors Evaluation

The output of a binary classifier, producing continuous scores in the range $(0, 1)$, can be interpreted as an estimate of the probability of belonging to the positive class p or the negative one n. Through calibration, a threshold $0 < \theta < 1$, is chosen so that scores $s > \theta$ will be interpreted as positive and negative otherwise. Classifying a sample with different threshold and score will produce four possible outcomes:

- *True Positive*: a positive sample classified as positive
- *True Negative*: a negative sample classified as negative
- *False Positive*: a negative sample classified as positive
- *False Negative*: a positive sample classified as negative

A classifier for speaker recognition can be evaluated in terms of the probability that a target speaker (belonging to the positive class) is misclassified as an impostor (belonging to the negative class) and the probability that an impostor is misclassified as the enrolled speaker. These two probabilities are called False Reject Rate (FRR) and False Accept Rate (FAR), given by

$$FRR = \frac{\#\text{false negatives}}{\#\text{total positive samples}} = \frac{\#\text{target speaker scores} < \theta}{\#\text{total target speaker samples}} \tag{5.1}$$

$$FAR = \frac{\#\text{false positives}}{\#\text{total negative samples}} = \frac{\#\text{impostor scores} > \theta}{\#\text{total impostor samples}} \tag{5.2}$$

The threshold θ controls the tradeoff between FRR and FAR. For example, to decrease the probability of the model to accept an impostor, or increase the FRR, θ has to be increased but FAR will decrease and the system will reject target speakers more often. The value of θ at which $FAR = FRR$ is called Equal Error Rate (EER). Being EER an upper bound for error measures, reducing it will mean improving the maximum error of the other operating points [11]. The EER can be used to evaluate the performance of a classifier or comparing it with other classifiers.

5.1.1 ROC curve

The ROC curve can be obtained plotting FAR and FRR with respect to different threshold θ values. A set of points $p_\theta = (FAR_\theta, FRR_\theta)$ is obtained by evaluating FAR and FRR at different values of θ. The ROC curve lets visualize the tradeoff between FAR and FRR changing with different thresholds while giving an easy way to obtain the EER point, it can be evaluated as the intersection between the curve and the quadrant bisector $y = x$. The better a classifier performs, the closer the curve will be to the axes.
An useful variant of the ROC curve is the DET curve, where the axes representing FAR and FRR are scaled so that the plot is more spread and the curve is closer to a linear line [50]. The original interval of the axes [0,1] is transformed to $[-\inf, +\inf]$ through the probit transformation [11]:

$$probit(p) = \sqrt{2}erf^{-1}(2p - 1) \tag{5.3}$$

where erf is the error function

$$erf = \frac{2}{\sqrt{\pi}} \int_0^p e^{-t^2} dt \tag{5.4}$$

The intersection between the new curve and the bisector $y = x$ represents the EER. Due to the linear nature of the DET curve it's easier to compare classifiers performance, using their distance from each other, the distance from the origin and differences in regions with low error–rates [11].

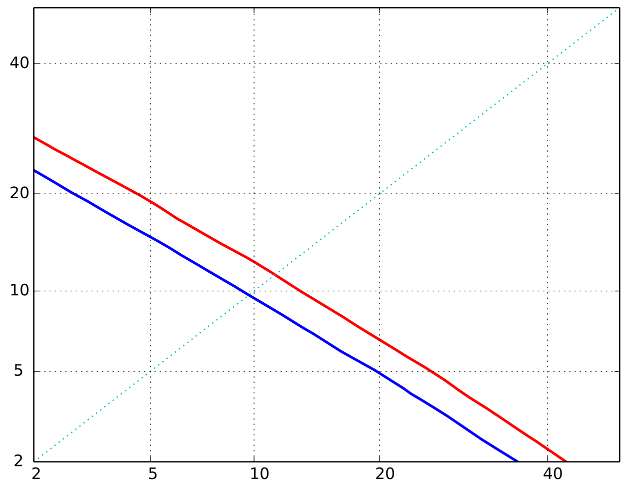


Figure 5.1: An example of DET curves

5.1.2 Detection Cost Function

Given two weights α_1 and α_2, depending on the application and the type of evaluation, the class prior probability π_T and $\pi_F = 1 - \pi_T$, and the decision threshold t then the Detection Cost Function (DCF) is defined as the weighted sum of FAR and FRR

$$DCF(t) = \pi_T \alpha_1 FRR(t) + \pi_F \alpha_2 FAR(t) \tag{5.5}$$

Usually α_2 is higher in order to penalize the acceptance of a non target speaker. One of the main metrics in speaker recognition is the minimum value of $DCF(t)$ referred as *minDCF*.

5.2 Experiental Results

In this chapter we present some experimental results obtained with the architectures shown in Section 3.3 and the different optimization techniques presented in Chapter 4. The test set used to evaluate the performance is the NIST Speaker Recognition Evaluation 2019 (SRE19)[1] dataset. In the following tables we present the results obtained on the Eval set. Table 5.1 shows the results achieved by our baseline system, which is an Hybrid DNN - GMM trained i-vector system [51].

Table 5.1: Baseline i-vector result.

	EER	minDCF
i-vector	11.3%	0.685

5.2.1 Clean Training

In the first stage of this work, we focused on the contribution of the network architectures, training different models on the clean version of the dataset. The first experiment used VoxCeleb1 (VC1) only. We recall that the network has two hidden layers before the output one, A and B in Figure 4.1. The output from the linear layer is the embedding, respectively embedding A and embedding B. As shown in Table 5.2 embedding A allows better performance and there isn't great improvement using a TDNN with LDE as the pooling layer against a TDNN with standard deviation and mean. The bad performance of resnet34 is expected as explained previously.

In Table 5.3 we added Mixer6 (M6) to the training data, obtaining a relative improvement of about 20% for all the networks. In this case, we also tested the ResNet implemented with TDNN layers obtaining a further 10% improvement in the EER with respect to the classical resnet34.

[1] https://www.nist.gov/itl/iad/mig/nist-2019-speaker-recognition-evaluation

Table 5.2: Results for different architectures trained with VoxCeleb 1 clean data only.

	Emb. A		Emb. B	
	EER	minDCF	EER	minDCT
TDNN	13.9%	0.779	18.6%	0.873
TDNN + LDE	13.8%	0.802	18.7%	0.872
ResNet34*	16.2%	0.870	—	—

We also tried to train the network with only one layer, obtaining only embedding A, noticing no worsening in performance overall.

Table 5.3: Results for different architectures trained with VoxCeleb 1 and Mixer 6 clean.

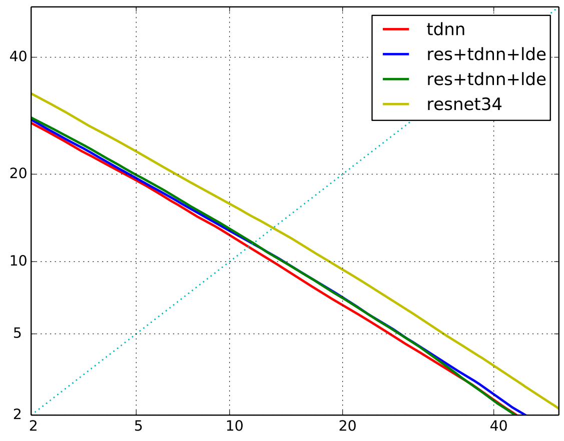
	Emb. A		Emb. B	
	EER	minDCF	EER	minDCT
TDNN	11.0%	0.676	14.7%	0.729
ResNet34*	12.5%	0.741	—	—
ResNet34+TDNN	11.4%	0.684	14.4%	0.787
ResNet34+TDNN+LDE*	11.2%	0.697	—	—
ResNet34+TDNN+LDE	11.2%	0.705	13.4%	0.783

Table 5.4 shows the result obtained using all the dataset available to train a TDNN network. Training the network with all the speakers allowed a further 20% improvement with respect to using only VC1 and M6, and almost 40% against VC1 alone. Despite the promising result, the huge amount of data and the consequent long training time for a single model didn't allow us to train other architectures. In Figure 5.2 we plotted the DET curves obtained from the different architectures trained with VoxCeleb1 and Mixer6 without augmentation.

Table 5.4: Results for different architectures trained with ALL clean dataset.

	Emb. A		Emb. B	
	EER	minDCF	EER	minDCT
TDNN	8.7%	0.573	11.2%	0.682

Figure 5.2: DET plot of different architectures.



5.2.2 Augmentation

In this section, the training data are augmented as introduced in Section 4.1.1. Table 5.5 shows the results obtained using VoxCeleb1 augmented in all the 4 ways. This leads to a 10% reduction of the EER on all the networks, once again the combination of TDNN and LDE achieves slightly better results, while resnet34 performs the worst.

Table 5.5: Results for different architectures trained with VoxCeleb1 augmented 4 times.

	Emb. A	
	EER	minDCF
TDNN	12.0%	0.722
TDNN+LDE	11.9%	0.727
ResNet34*	13.4%	0.775

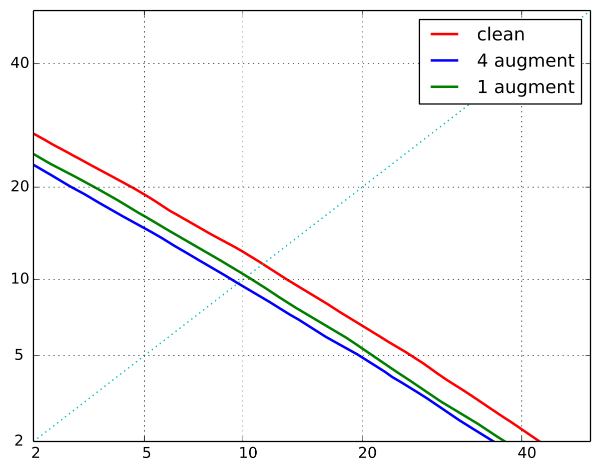
The second set of experiments with augmented data used VoxCeleb1 and Mixer6, both augmented either 4 times or once for each epoch. In Table 5.6 it is possible to notice the similar performance obtained, this is important due to the long training time in case of full augmentation and space required to save the data. Due to this, and given the overall better performance of TDNN, we only tested this architecture.

Table 5.6: Results for different architectures trained with VoxCeleb1 and Mixer6 augmented.

	Emb. A		Emb. B	
	EER	minDCF	EER	minDCT
TDNN 4×epoch	9.6%	0.620	11.7%	0.681
TDNN 1×epoch	9.9%	0.622	13.0%	0.732

Figure 5.3 shows the curves of a TDNN trained with VoxCeleb1 and Mixer6 at different level of augmentation.

Figure 5.3: DET plot of TDNN trained with VC1 and M6



5.2.3 Optimization

In the last section, we present the results obtained changing the loss function. Given the good performance of embedding A, we decide to bypass the layer B and use the first one as input for the new loss functions. Table 5.7 shows the results achieved with the resnet34 using VoxCeleb1 and Mixer6 as training data. The focus was to let the network train for some extra epochs with different loss functions.

Table 5.7: Results for ResNet34 and optimization with VoxCeleb1 and Mixer6.

	Softmax			SphereFace		
	Ep5	Ep6	Ep7	Ep6	Ep7	Ep9
EER	12.5%	12.5%	12.6%	16.3%	16.6%	16.5%
minDCF	0.741	0.740	0.733	0.900	0.900	0.907

In Table 5.8 the results obtained by the different loss functions on the TDNN are shown. The additional epoch without changing the loss function achieved 20% better while using the Angular Softmax (SphereFace) allowed to reduce the EER of 25%. On the other hand, our implemen-

Table 5.8: Results for TDNN and optimization with VoxCeleb1 and Mixer6.

	Spftmax		SphereFace		CosFace[a]		ArcFace[b]
	Ep5	Ep6	Ep6	Ep7	Ep6	Ep7	Ep6
EER	11.0%	11.0%	10.6%	10.3%	11.2%	12.0%	11.1%
minDCF	0.676	0.669	0.649	0.653	0.727	0.741	0.707

[a]Our implementation

[b]No softmax pretraining.

tation of the Additive Margin (CosFace) achieved worst results in the additional epochs. We also tried to apply the Additive Angular Margin Loss (Arcface) from scratch. The result of this training is similar to the one trained with classical softmax.

The model trained with VoxCeleb1 and Mixer6 augmented once each epoch, was further trained with SphereFace loss. The results of the additional training are shown in Table 5.9. The classic Softmax decreased the EER of about 1%, while SphereFace achieved a reduction of 4% and 5% in the two extra epochs.

Table 5.9: Results for TDNN and optimization with VoxCeleb1 and Mixer6 augmentd Once.

	Softmax		SphereFace	
	Ep5	Ep6	Ep6	Ep7
EER	9.9%	9.8%	9.6%	9.5%
minDCF	0.622	0.623	0.630	0.646

In Table 5.10 we present the results obtained with a different backend, Pairwise SVM instead of PLDA. The model evaluated is the same one used to obtain the results shown in Table 5.8. The EER calculated using PSVM is around 10% less than the one obtained using PLDA.

Table 5.10: Results for TDNN with VoxCeleb1 and Mixer6 clean using PSVM.

		Emb. A	
		EER	minDCF
SphereFace Ep6	PSVM	9.45%	0.623
	PLDA	10.6%	0.649

Overall our embeddings achieved better results than the one obtained with the i-vector, Table 5.1. The best result was achieved using all the datasets to train a TDNN, Table 5.4. The results obtained using augmentation,Table 5.6, suggests that augmenting all the datasets will produce better results. Also, SphereFace showed to be able to further improve the performace of a pretrained network, Table 5.8.

Chapter 6

Conclusions and Future Work

In this work we have shown how the embeddings extracted from a deep architecture, like TDNN, allows us to obtain comparable if not better performance than the i-vector. In particular, it has been shown how increasing the number of speakers in the training dataset allows the network to learn to better generalize the characteristics of the speakers not yet seen. The contribution of the augmentation is also important, allowing us to achieve performance similar to that obtained using datasets with many more speakers. It has also been shown that having 4 augmentations, music, noise, babble, and reverberation, does not have many benefits compared to having a different augmentation per epoch, allowing to save space and time for training. The results obtained with the loss functions originally developped for image recognition were promising. Angular softmax and additive angular margin allowed to further improve the results obtained by a pre-trained TDNN network. On the other hand, our implementation of additive margin proved to be slightly less performing than the two. Future work will focus on extending the results obtained here to further improve the results, trying to obtain state-of-the-art performance. For this, we will study new combinations of architectures, and new layers. Other approaches to optimizing embeddings, such as end-to-end training, or other loss functions, will be considered. Furthermore, we will try to combine speaker recognition and face recognition to obtain more robust and multimodal recognition systems.

Bibliography

[1] "Time delay neural network," https://kaleidoescape.github.io/tdnn/, accessed: 05/07/2020.

[2] J. Deng, J. Guo, and S. Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4685–4694, 2019.

[3] L. R. Rabiner and B.-H. Juang, "Fundamentals of speech recognition," in *Prentice Hall signal processing series*, 1993.

[4] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 28, pp. 357–366, 1980.

[5] S. Cumani, "Speaker and language recognition techniques," Ph.D. dissertation, Politecnico di Torino, 2012.

[6] C. M. Bisho, *Pattern Recognition and Machine Learning.* Springer, 2006.

[7] D. A. Reynolds and R. C. Rose, "Robust text-independent speaker identification using gaussian mixture speaker models," *IEEE Trans. Speech Audio Process.*, vol. 3, pp. 72–83, 1995.

[8] N. M. L. A. P. Dempster and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society*, pp. 1–39, 1977.

[9] M. A. Mohamed and P. D. Gader, "Generalized hidden markov models — part i : Theoretical frameworks," 2008.

[10] A. H. Waibel and K.-F. Lee, "Readings in speech recognition," 1990.

[11] N. Brümmer, "Measuring, refining and calibrating speaker and language information extracted from speech," Ph.D. dissertation, University of Stellenbosh, 2010.

[12] D. A. Reynolds, "Comparison of background normalization methods for text-independent speaker verification," in *EUROSPEECH*, 1997.

[13] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker verification using adapted gaussian mixture models," *Digit. Signal Process.*, vol. 10, pp. 19–41, 2000.

[14] J.-L. Gauvain and C.-H. Lee, "Maximum a posteriori estimation for multivariate gaussian mixture observations of markov chains," *IEEE Trans. Speech Audio Process.*, vol. 2, pp. 291–298, 1994.

[15] P. Kenny, "Joint factor analysis of speaker and session variability: Theory and algorithms," *Technical report CRIM-06/08-13*, 2005.

[16] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, pp. 788–798, 2011.

[17] P. Kenny, G. Boulianne, P. Ouellet, and P. Dumouchel, "Joint factor analysis versus eigenchannels in speaker recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, pp. 1435–1447, 2007.

[18] P. Kenny, M. Mihoubi, and P. Dumouchel, "New map estimators for speaker recognition," in *INTERSPEECH*, 2003.

[19] N. Dehak, "Discriminative and generative approaches for long- and short-term speaker characteristics modeling: application to speaker verification," 2009.

[20] P. Kenny, G. Boulianne, and P. Dumouchel, "Eigenvoice modeling with sparse training data," *IEEE Transactions on Speech and Audio Processing*, vol. 13, pp. 345–354, 2005.

[21] N. Dehak, R. Dehak, J. R. Glass, D. A. Reynolds, and P. Kenny, "Cosine similarity scoring without score normalization techniques," in *Odyssey*, 2010.

[22] A. O. Hatch, S. S. Kajarekar, and A. Stolcke, "Within-class covariance normalization for svm-based speaker recognition," in *INTERSPEECH*, 2006.

[23] S. Ioffe, "Probabilistic linear discriminant analysis," in *ECCV*, 2006.

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

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

[26] N. Brü"mmer and E. de Villiers, “The speaker partitioning problem,” in *Odyssey*, 2010.

[27] S. Cumani, N. Brümmer, L. Burget, and P. Laface, “Fast discriminative speaker verification in the i-vector space,” *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4852–4855, 2011.

[28] L. Burget, O. Plchot, S. Cumani, O. Glembek, P. Matejka, and N. Brümmer, “Discriminatively trained probabilistic linear discriminant analysis for speaker verification,” *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4832–4835, 2011.

[29] C. Hc, “Burges: A tutorial on support vector machines for pattern recognition,” 1998.

[30] J. C. BurgesChristopher, “A tutorial on support vector machines for pattern recognition,” *Data Mining and Knowledge Discovery*, 1998.

[31] V. N. Vapni, “The nature of statistical learning theory,” 1995.

[32] C. M. B. G. Hinton, “Neural networks for pattern recognition,” 2005.

[33] Y. Lei, N. Scheffer, L. Ferrer, and M. McLaren, “A novel scheme for speaker recognition using a phonetically-aware deep neural network,” *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1695–1699, 2014.

[34] F. Richardson, D. A. Reynolds, and N. Dehak, “Deep neural network approaches to speaker and language recognition,” *IEEE Signal Processing Letters*, vol. 22, pp. 1671–1675, 2015.

[35] C. Guo and F. Berkhahn, “Entity embeddings of categorical variables,” *ArXiv*, vol. abs/1604.06737, 2016.

[36] Y. Konig, L. Heck, M. Weintraub, and K. S. Sonmez, “Nonlinear discriminant feature extraction for robust text-independent speaker recognition,” 1997.

[37] L. Heck, Y. Konig, M. K. Sönmez, and M. Weintraub, “Robustness to telephone handset distortion in speaker recognition by discriminative feature design,” *Speech Commun.*, vol. 31, pp. 181–192, 2000.

[38] A. H. Waibel, T. Hanazawa, G. E. Hinton, K. Shikano, and K. J. Lang, “Phoneme recognition using time-delay neural networks,” *IEEE Trans. Acoust. Speech Signal Process.*, vol. 37, pp. 328–339, 1989.

[39] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, pp. 1735–1780, 1997.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

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

[42] D. Snyder, G. Chen, and D. Povey, "MUSAN: A Music, Speech, and Noise Corpus," 2015, arXiv:1510.08484v1.

[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[44] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[45] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *ArXiv*, vol. abs/1502.03167, 2015.

[46] D. A. Snyder, J. Villalba, N. Chen, D. Povey, G. Sell, N. Dehak, and S. Khudanpur, "The jhu speaker recognition system for the voices 2019 challenge," in *INTERSPEECH*, 2019.

[47] W. Cai, Z. Cai, X. Zhang, X. Wang, and M. Li, "A novel learnable dictionary encoding layer for end-to-end language identification," *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5189–5193, 2018.

[48] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "Sphereface: Deep hypersphere embedding for face recognition," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6738–6746, 2017.

[49] F. Wang, J. Cheng, W. Liu, and H. Liu, "Additive margin softmax for face verification," *IEEE Signal Processing Letters*, vol. 25, pp. 926–930, 2018.

[50] A. F. Martin, G. R. Doddington, T. Kamm, M. Ordowski, and M. A.

Przybocki, “The det curve in assessment of detection task performance,” in *EUROSPEECH*, 1997.

[51] S. Cumani, P. Laface, and F. Kulsoom, “Speaker recognition by means of acoustic and phonetically informed gmms,” in *INTERSPEECH*, 2015.