BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

EXPLORING NEW PATHS IN NEURAL-NETWORK-BASED SPEAKER RECOGNITION

HLEDÁNÍ NOVÝCH CEST V ROZPOZNÁVÁNÍ ŘEČNÍKA ZALOŽENÉHO NA NEURONOVÝCH SÍTÍCH

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **DAMIÁN SOVA**
AUTOR PRÁCE

SUPERVISOR **Ing. ONDŘEJ GLEMBEK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Sova Damián**
Program: Informační technologie
Název: **Hledání nových cest v rozpoznávání řečníka založeného na neuronových sítích**
Exploring New Paths in Neural-Network-Based Speaker Recognition
Kategorie: Zpracování řeči a přirozeného jazyka

Zadání:

1. Study state-of-the-art speaker recognition systems.
2. Re-implement existing system based on ResNet architecture. Evaluate the system on a suitable test set.
3. Implement training of the system on single GPU, extending to multiple GPUs.
4. Propose and implement a modification in the architecture of the NN and/or training scheme.
5. Evaluate your proposals and elaborate on future work.

Literatura:

- D. Snyder et al., "X-vectors: Robust DNN embeddings for speaker recognition", ICASSP, 2018
- H. Zeinali et al., "BUT System Description to VoxCeleb Speaker Recognition Challenge 2019.", arxiv.org, 2019
- P. Matejka et al., "13 years of speaker recognition research at BUT, with longitudinal analysis of NIST SRE", Computer Speech and Language, 2020

Pro udělení zápočtu za první semestr je požadováno:

- First two items of the description

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Glembek Ondřej, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 2. listopadu 2021

Abstract

Since the assignment of this work is very broad, it was necessary to focus only on a certain area. In the end, this work aims to apply the Stochastic Weight Averaging optimization method to the training process of the Deep Neural Network. After presenting the necessary theoretical knowledge in the first part of the work, the second part with the experiments courses follows. In the theoretical part, the main focus is on presenting the complete life-cycle of the training and evaluation process, including a description of each component. The practical part provides a detailed look at each experiment, intended to demonstrate the effectiveness of the overall speaker recognition system's performance enhancement. The overall performance improvement is achieved by gradually applying various training configurations where the experience from previous experiments is taken into account. The key ingredient to the successful Stochastic Weight Averaging in the experiments was a sufficiently high Learning Rate value with the successive transition applied or Cyclic course of the Learning Rate.

Abstrakt

Keďže zadanie tejto práce je veľmi široké, tak sa bolo treba sústrediť len na určitú sféru. Nakoniec, cieľom tejto práce je aplikovať optimalizačnú metódu Stochastického Spriemerovania Váh do tréningového procesu Hlbokej Neurónovej Siete. Po predstavení potrebných teoretických vedomostí v prvej časti práce, nasleduje druhá časť s priebehmi jednotlivých experimentov. V teoretickej časti je dôraz kladený hlavne na objasnenie celého životného cyklu trénovacieho a vyhodnocovacieho procesu, vrátane popisu jednotlivých komponentov. Praktická časť poskytuje podrobný pohľad na každý experiment, ktorých cieľom je demonštrovať dosiahnuteľnosť zvýšenia výkonnosti systému rozpoznávania rečníka. Celkové zlepšenie výkonu sa podarilo dosiahnuť postupným aplikovaním rôznych tréningových konfigurácií, v ktorých sa zohľadňujú skúsenosti z predchádzajúcich experimentov. Kľúčovou zložkou úspešného Stochastického Spriemerovania Váh v experimentoch bola dostatočne vysoká konštantná hodnota Miery Učenia s aplikovaným postupným prechodom alebo Cyklický priebeh Miery Učenia.

Keywords

Speaker Recognition, Residual Network, x-vector, Deep Neural Network Training Optimization Techniques, Stochastic Weight Averaging

Kľúčové slová

Rozpoznávanie Rozprávača, Reziduálna Sieť, x-vektor, Techniky Optimalizácie Tréningu Hlbokej Neurónovej Siete, Stochastické Spriemerovanie Váh

Reference

SOVA, Damián. *Exploring New Paths in Neural-Network-Based Speaker Recognition.* Brno, 2022. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Ondřej Glembek, Ph.D.

Exploring New Paths in Neural-Network-Based Speaker Recognition

Declaration

I declare that I have prepared this bachelor's thesis on my own under the guidance of Ing. Ondřej Glembek Ph.D. I used only the sources listed in the Bibliography and Internet sources.

.....
Damián Sova
May 9, 2022

Acknowledgements

I would like to thank the supervisor Ing. Ondřej Glembek Ph.D. for the time, advice and especially for the model along with the necessary datasets provided to me during the creation of this work. Last but not least, thank you friends and family for your support.

Contents

Chapter 1

Introduction

Speaker recognition (SR) is the task of recognizing the identity of someone based on the speaker's speech. The speech production system's physiological and behavioural characteristics influence the speaker's identity. For example, we can identify our friend by hearing him without the need to see him. We can consider the SR study as the use of employing statistical methods to identify individuals based on their unique acoustic properties, which are encoded in a sequence of successive samples in time [42]. We can tackle this task in two ways: *speaker identification* and *speaker verification*.

Speaker identification determines a speaker's identity from a given utterance about the speaker. The searched identity comes from a closed-set of registered speakers, represented by the model for each speaker, where a given utterance is analyzed and compared to all speech models. As a result, the decision is based on the best utterance–model match.

On the other hand, speaker verification is a process where we accept or reject the identity claimed by the speaker. Occasionally, we can reinterpret this task as deciding whether an utterance matches a specific speaker model. In other words, the main goal is to compare two utterances and determine whether they come from the same speaker or not. We can easily convert a speaker verification system into a speaker identification system by limiting the set of utterances which is the system comparison.

In terms of the content in SR, we can classify it either as *text-dependent* or *text-independent*. An approach based on text-dependent SR looks at speech content. On the contrary, a text-independent approach does not think about what the speaker says. The approach of this thesis is text-independent speaker verification.

There are a variety of applications, and the technology is constantly evolving. As a result, it is possible to verify the identity of speakers by using their voices. Additionally, control the access to services such as voice dialling and voice mail, telebanking, telephone shopping, and information services. Also, manage the security control over confidential information areas and remote access to computers. SR technology is expected to create a host of new services that make our daily lives more convenient.

The SR system's main component and focus of this thesis is a deep neural network (DNN) that encodes the identity of the speaker—mined from speech—into a low-dimensional representation. It is essential that we understand the importance and complexity of DNN and the processes that make it work properly. For this reason, we need to describe in a separate chapter the general principles of development and function of each component of the functional unit. Speaking of DNN, the biggest problem is to train it effectively to ensure its proper functioning. Training that is considered adequate uses minimal computing resources and is as time-efficient as possible. We seek methods to streamline the entire

training process to reduce computing resources. The use of optimization methods would improve recognition accuracy and reduce the "cost". See Section 2.2.5 for a comprehensive introduction to these methods. In the following paragraphs is an overview of the thesis structure and content.

Chapter 2 introduces a complete cycle of SR and brings a detailed view of each processing part. This chapter is more extensive and provides the theoretical background needed for understanding the recognition system's inner and outer connections. Moreover, the chapter explains the thesis's primary purpose and the core of the problem this work addresses.

The implementation details and used techniques in experiments can be found in Chapter 3, which consists of several parts. Each part describes the course of the experiment and the result of a particular process. Additionally, in this chapter, some of the technical problems (which occurred through the development and other issues) are addressed to facilitate future work and to illustrate the system's complexity and the associated problems.

The last Chapter 4 summarizes the results of this thesis and provides ideas for future experiments and other enhancements.

Chapter 2

Theoretical Background

Voice as the sole means of identifying a speaker has some inherent issues. By the nature of the communication channel or by the background noise, the characteristics of a speaker's voice can be altered. Therefore, SR systems should be able to accept a variety of different voice variations. The system could accept other speakers with similar voice characteristics by adding this capability. SR should include a close study of the clues humans use to recognize a speaker for successful recognition. Recognizing speakers requires an understanding of these principles. Therefore, finding stable voice features is an essential task for SR. The features and techniques that have been identified and developed for SR are presented in Section 2.1.

According to the introduction, the speaker verification task is to determine whether a pair of utterances come from the same speaker (hypothesis $\mathcal{H}_1$) or various speakers (hypothesis $\mathcal{H}_2$). This work presumes no presence of multiple voices (present only one speaker) in the utterances themselves.

We make the hypothesis detection by evaluating statistical models based on the data provided. The process involves using a 2-class classifier, whose output is a Log-Likelihood Ratio (LLR) for the two hypotheses. Following the definition of speaker verification from the introduction, the diagram of the speaker verification procedure appears in Figure 2.1. A pair of two utterances d_1 and d_2 is fed into the likelihood function (referred to as a *trial* $x = \langle d_1, d_2 \rangle$) and the system is generally symmetrical, i.e. the order of d_1 and d_2 is irrelevant. Mathematically, the score can be denoted as

$$s = log \frac{p(x|\mathcal{H}_1)}{p(x|\mathcal{H}_2)} \,. \tag{2.1}$$

Thus, the final decision is determined according to the value of the final score and the decision threshold. For more information on the decision-making mechanism, see Section 2.5.

Let us briefly introduce the goal of this work. This thesis examines state-of-the-art SR systems and aims to enhance their accuracy and overall performance. The main interest is better optimization of training methods for neural network training. To better understand optimization training methods, we need to understand the basis of the system we are trying to optimize. The DNN is the core of the SR system, and its main feature is the extraction of the most important and relevant information from a given utterance. As mentioned in the introduction, this thesis aims at optimizing the process of DNN training.

Now let us look at the recognition process and describe each processing step. It is necessary to mention the presence of the two system's modes in the recognition process, *training mode*, *evaluation mode*. The initial step of the training mode is to train DNN on

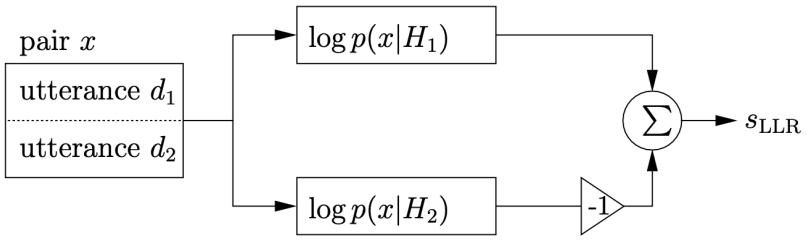


Figure 2.1: General symmetrical speaker verification procedure. As the input is a trial x given as a pair of utterances $\langle d_1, d_2 \rangle$ and the likelihood of utterances computation is conditioned by the hypotheses that the utterances come either from a single speaker ($\mathcal{H}_1$) or two different speakers ($\mathcal{H}_2$)[1].

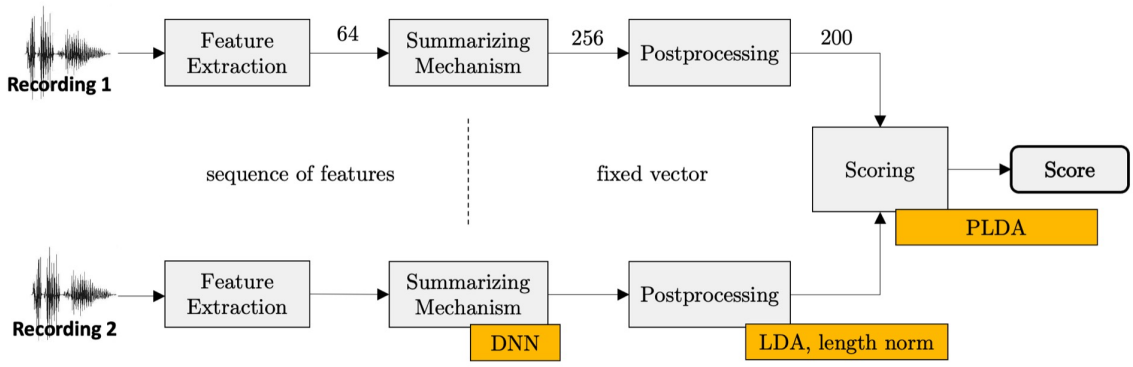


Figure 2.2: The process of the speaker verification scheme displays all the recognition phases, from processing utterances to evaluating the final score. The numbers above elements' relations show dimensionalities of data (vectors) passing from one element to another. The orange elements outline the core functionality of each processing component relevant to the thesis.

a set of training data. Training consists of optimizing the DNN parameters, which ensures the accuracy of the recognition results. We can initiate the evaluation mode after successful model training, allowing SR. The evaluation phase also requires a separate dataset, which is different from the training one. For more details about the datasets and specifications, see Section 2.6.

As far as neural network training is concerned, correctly setting the values of individual weights in several neural network layers is a question. We need a suitable set of training data to set these values correctly. The weights are then optimized using *backpropagation* (described in Section 2.2.4) in several iterations. The method of finding ideal values is computationally very demanding, and also the accuracy of the found result may not be sufficient. Therefore, we can use different methods of optimizing the training process to increase its efficiency and accuracy. The training process and training optimization methods, such as Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (Adam) and an auxiliary algorithm Stochastic Weight Averaging (SWA), will be described in Section 2.2.5.

[1]Figure used from [23] with permission.

We extract the information from the processed utterance using the earlier trained model, which has a DNN optimized for best accuracy during the SR process. Once we optimize the model, recognition can begin. As was mentioned at the beginning of this chapter, the input to the recognition process is a pair of utterances. The output is a determination of whether they come from the same speaker. According to Figure 2.2, respecting the order of the steps, the recognition process consists of 4 parts:

- **Feature Extraction** - Input signal (utterance) processing.
- **Summarizing Mechanism** - The extraction of the speaker's most relevant information.
- **Postprocessing** - Simplification of the information representation, dimension reduction.
- **Scoring Mechanism** - Generating likelihood score for the given pair of utterances.

In addition, the evaluation mode needs one more component that determines whether two recordings have a common author based on the scores of the scoring mechanism. This component is called *backend*, and its primary purpose is to evaluate the overall recognition success. In other words, we use the previously generated likelihood score to decide the speaker's identity. Afterwards, the accuracy of the made decisions has to be evaluated and expressed by the Equal Error Rate (EER). This rate is often used as an evaluation metric to determine the overall error rate of the system and a more detailed description will be in Section 2.5.2.

To be able to accept or reject the identity claimed by the speaker, a *threshold* needs to be defined. Setting this threshold can affect the way the system behaves and its security. For illustration, if the threshold is too low, the system will falsely accept that the speaker is the same in both recordings. In the case of a too high threshold value, the system will incorrectly reject the speaker's identity to be the same in both recordings. In other words, setting a threshold on a low value enables the system to accept the identity claimed by the speaker even if the speakers are different. In case of a high threshold value, the system will reject the identity claimed by the speaker even if they are the same. This mechanism will be discussed later in Section 2.5.

2.1 Feature Extraction

Before describing the extraction of speech characteristics, it is appropriate to take a closer look at the characteristics themselves, which can be divided into two levels, *higher-level characteristics* and *lower-level characteristics*. Higher-level characteristics are, for example, intelligibility, liveliness, rudeness and voice power. The problem is to quantify and use them for automatic SR. The point of interest for us is lower-level characteristics, which are also divided into two categories, namely *internal characteristics* and *acquired characteristics*.

Internal characteristics are related to the anatomy of the vocal system. The vocal cord source characteristics and the vocal tract size and shape determine the internal characteristics. Differences in the anatomical structure of the articulatory organs of different people affect the acoustic properties of the speech signal. Essentially, they manifest themselves as individual deviations, primarily in the values of the frequency of the fundamental vocal cord tone and in the frequency and bandwidth of individual formants. We can differentiate

between a man and a woman and between an adult and a child by analyzing internal characteristics. Each voice is unique in these characteristics due to the physical dimensions of its vocal system, which is an advantage for speaker verification. The disadvantage, however, is the high sensitivity to the health conditions of the individual, where even with a mild cold, we observe significant changes in the internal characteristics of speech. For example, the distance between the frequencies of neighbour formants averages 1000 Hz for men. This value is related to the median length of the male vocal tract, which is approximately 17 cm. Women have a shorter vocal tract, a greater distance between formant frequencies, and an average of 17 % higher formant frequency values.

The acquired characteristics are a consequence of the vocal tract part's movement dynamics and the environment in which the person grew up. These characteristics include speech rate, prosody, or dialect. As a result, we can easily imitate them, which is disadvantageous for speaker verification [27].

On the one hand, there are sufficient differences in the speech of individual speakers (*variability between speakers*). On the other hand, there are also differences in the speech of a particular speaker (*internal speaker variability*). Various factors might cause internal speaker variability, the factors whose source is the speaker and those whose source is not the speaker. Factors, which source is the speaker include the state of health or the speaker's emotional state. The second group of factors causing internal speaker variability, i.e. non-speaker sources, includes, in particular, the frequency response of the microphone, the transmission paths and the recording device, which affect the frequency spectrum of the speech signal. The frequency spectrum is also affected by the acoustic noise level in the recording environment.

Researchers have done many experiments to determine which lower-level speech characteristics are most effective for SR. That is, which characteristics are highly variable among speakers and at the same time have small internal variability of the speaker. The results of the experiments led to the use of *flags* in speech recognition systems expressing the spectral properties of the speech signal. Some of the experiments are described here [27]. There are no exclusive features in the speech signal that identifies a speaker. As part of the source-filter theory of speech production, it is known that the formants and pitch harmonics of the speech spectrum encode information about the vocal tract shape and glottal source of the speaker. Thus, most SR systems use some form of spectral based feature.

2.1.1 Filter Bank

Consider an input audio signal (utterance, e.g. in .wav format) $x(n)$. The first step is to create a spectrum from this signal by a Discrete Fourier Transformation (DFT). The spectrum of a signal gives the distribution of signal energy as a function of frequency [4]. We need a *bandpass filter* to separate the energy from a signal's spectrum frequency region. Ideal the bandpass filter gives all input signal energy within the desired range as the output. We refer to the range of accepted frequencies as the *band*. The frequency boundaries defining the band, f_{low} and f_{high}, are known as the lower and upper cutoff frequencies. These are the band edges where the difference between them defines the *bandwidth*:

$$BW = f_{high} - f_{low} \ . \tag{2.2}$$

In the middle of the bandwidth lies the centre frequency f_c of the bandpass filter, which is used to describe the ratio known as the *quality factor*:

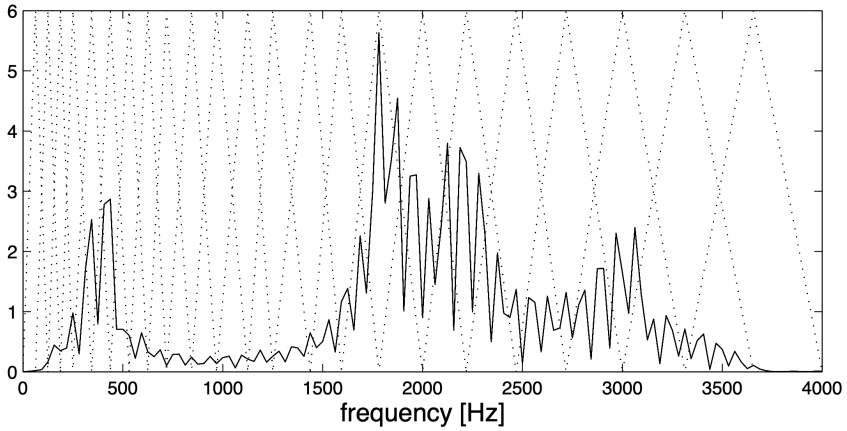


Figure 2.3: Fourier spectrum of speech segment (sampling frequency 8 kHz) with filter bank applied. The horizontal axis indicates the amplitude of the fourier spectrum. The dotted triangles represent particular frequency sub-bands of the filter bank spaced logarithmically over the spectrum[2].

$$Q = \frac{f_c}{BW} \,. \tag{2.3}$$

A filter bank (also called filterbank or f-bank) is an array of bandpass filters that divide the input signal $x(n)$ into multiple components, each one carrying one frequency sub-band (BW) of the original signal [33]. These regions may overlap but cover the entire audible range of human hearing (20 Hz - 20kHz). The decomposition process performed by the filter bank is called analysis (the signal analysis in terms of its components, and sub-bands). At first, we use the absolute value of the short-term discrete Fourier transform to extract the amplitude of the spectrum. Then we divide the signal spectrum into frequency bands using the filter bank and compute energy for each band by taking the square of the amplitude spectrum (see Figure 2.3 to visualize the triangular-shaped bank of filters logarithmically spaced). The analysis result is a sub-band signal (feature vector) with as many subbands as filters in the filter bank.

2.1.2 Extraction Configuration

As Figure 2.2 shows, the initial step involves utterance processing. We can consider the acoustic signal stationary as long as we look at him in milliseconds. As a result, we can divide the signal into short units called *frames*, which overlap somewhat. Features are subjected to short time mean normalization with a sliding window of 3 seconds, and energy-based Voice Activity Detection (VAD) from Kaldi[3] is applied.

Feature vectors are generated from the frames to represent speech information. The feature vector is a low-dimensional representation of a speech frame. In this work is used *filter bank* for signal processing. Concrete specifications used for raw utterance processing are listed below:

- **64-dimensional** - output feature vector

[2]Figure used from [23] with permission.
[3]A speech recognition toolkit, see https://github.com/kaldi-asr/kaldi

- **8kHz** - frequency limits 20-3800Hz
- **25ms** - frame length
- **Filter bank** - 64 filter-bank channels
- **Kaldi** - used toolkit

2.2 Summarizing Mechanism

The following section is crucial for understanding the overall meaning of this work. It brings insights into the meaning and structure of the most relevant component for this research announcing the embeddings, neural network and optimization techniques.

By focusing on the evaluation mode, we assume that we already have a trained DNN. One utterance decomposed to a feature vectors sequence generated in the previous step is the input to this phase. The vertical dashed line in the middle between the „*Summarizing mechanism*“ elements divides process data flow as shown in Figure 2.2. On the left side are data in the form of a variable-length feature sequence. However, on the opposite side is one fixed-sized vector for both branches.

The primary task of summarizing mechanism is to take a sequence of featured vectors (variable-length) previously processed representing the whole utterance and generate output as one fixed-length vector consisting of the most relevant information about the particular speaker from the original utterance. Below is the current state-of-the-art solution for representing the embeddings, DNN for their extraction, and DNN training techniques used in this work later in the experiments, Chapter 3.

2.2.1 Activation Function

Before the introduction of the used neural network and speaker suitable representation format, let us have a look into the crucial mathematical component, *activation function*. Activation functions play a significant role in igniting the hidden nodes to produce a more desirable output. The primary purpose of the activation function is to introduce the property of non-linearity into the model. It was previously more popular to use sigmoid and tanh as activation functions because they were differentiable and monotonous. Nevertheless, these functions suffer from saturation over time, resulting in problems with vanishing gradients. In general, the Rectified Linear Unit (ReLU) is the most popular activation function to overcome this problem.

Rectified Linear Unit

All points except zero are differentiable for ReLU [22] activation function. The maximum value is considered in cases where a value is greater than 0. This can be written as:

$$f(x) = \max(0, x) \ . \tag{2.4}$$

The default value for the negative numbers is 0, and the maximum value for the positive numbers is considered. It is relatively easy to differentiate the ReLU for the backpropagation computation (see Section 2.2.4) of neural networks. Only the derivative at 0 will be assumed, which will also be zero. The slope of a function is its derivative. The slope for negative values is 0, and the slope for positive values is 1. The main ReLU advantages:

- **Convolutional layers and deep learning:** They are the most popular activation functions for training convolutional layers and deep learning models.
- **Computational Simplicity:** The rectifier function is trivial to implement, requiring only a max () function.
- **Representational Sparsity:** An important benefit of the rectifier function is that it is capable of outputting a true zero value.
- **Linear Behavior:** A neural network is easier to optimize when its behavior is linear or close to linear.

Nevertheless, the main problem with the ReLU is that all negative values become zero immediately, decreasing the model's ability to correctly fit or train from the data. That means any negative input given to the ReLU activation function immediately turns the value to zero, affecting the resulting graph by not mapping the negative values appropriately. Fortunately, the problem can be easily solved by using different variants of the ReLU activation function. The other variants of ReLU include Leaky ReLU, ELU, SiLU, etc., which are used for better performance in exceptional cases.

Even though the ReLU was one of the best activation functions, it was not frequently used before. It caused a not differentiable function at point 0. Researchers tended to use differentiable functions like sigmoid and tanh. However, ReLU is the best activation function for deep learning currently—also used in this work.

2.2.2 x-vectors

This subsection provides essential insights into representing the speaker's relevant information from the utterance. There will also be some historical context of the x-vectors evolution. The state-of-the-art systems for text-independent speaker verification are based on DNN embeddings. These embeddings called x-vectors are low dimensional fix-length representations of utterances.

As for the predecessors of the current embeddings, the i-vector [5] or Gaussian Mixture Model (GMM) mean supervector in Support Vector Machines [40] can be taken into account. After these embeddings another is considered as the predecessor, the d-vector [2]. The core idea of the d-vector is to assign the ground-truth speaker identity of a training utterance as the labels of the training frames belonging to the utterance in the training stage, which transforms the model training into a classification problem [39].

X-vector is an essential evolution of d-vector that evolves SR from frame-by-frame speaker labels to utterance-level speaker labels with an aggregation process. X-vector is an output of a standard feedforward network where the main component is the network described below in Section 2.2.3. It first extracts frame-level embeddings of speech frames by convolutional layers, then concatenates the mean and standard deviation of an utterance as an utterance-level feature by a statistical pooling layer that classifies the segment-level feature to its speaker by a standard feedforward network. Together, the time-delay layer, statistical pooling layer, and feedforward network are trained. The x-vector is the segment-level embedding of the feedforward network produced from the second to last hidden layer. Data augmentation is vital in improving the performance of the x-vector, according to research [37]. This research describes one of the first uses of this neural network for speaker verification and provides details on robustly training the x-vector architecture.

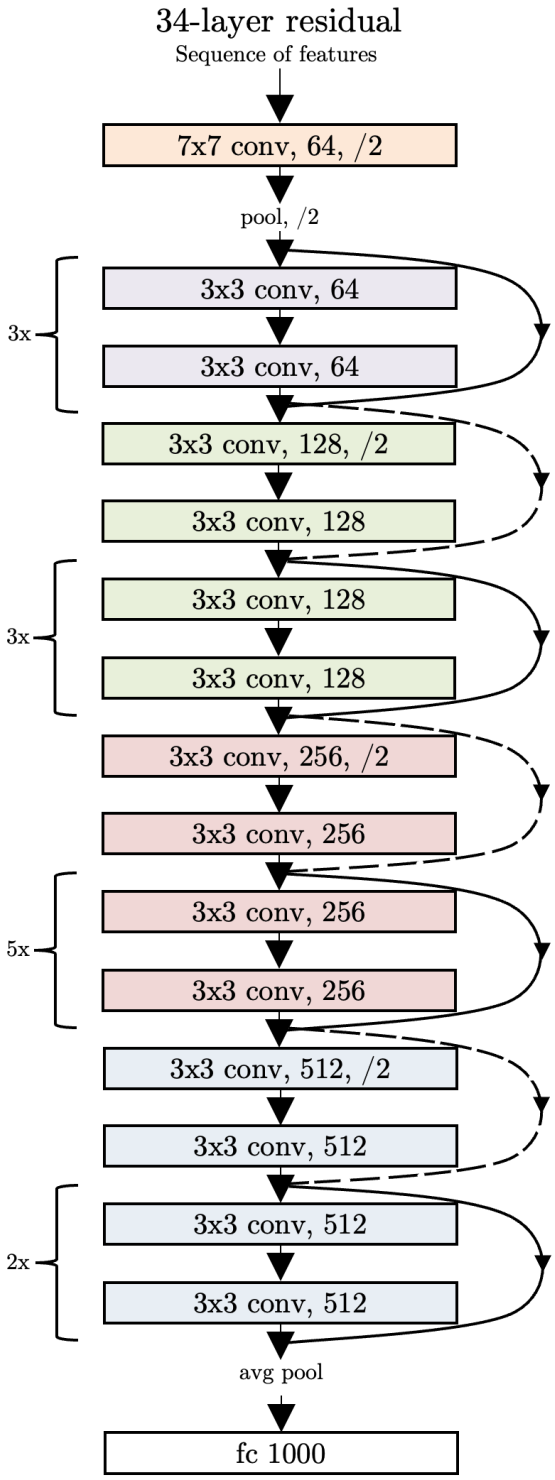


Figure 2.4: Simplified ResNet structure: the main component is a residual block from Figure 2.5 which represents two convolutional layers and direct propagation of input to the block output. This relation is described with a right-side connection (This feature is described later in Figure 2.5. Note that dashed relation symbolizes the change in input format between different layers dimensions.). For simplification, several residual blocks are omitted, and their number is shown on the left side of the corresponding block.

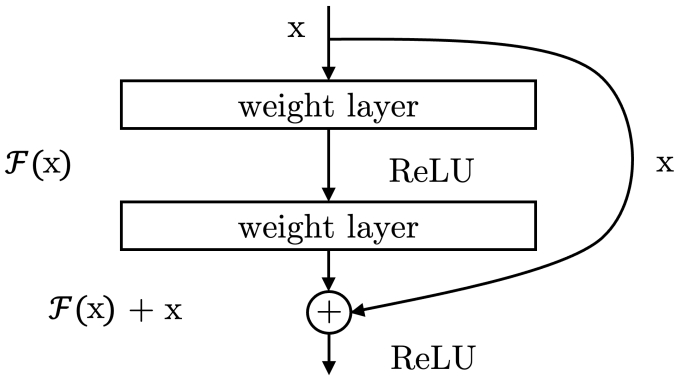


Figure 2.5: Residual building block: a key component in a residual network where the main feature is to propagate input to the output computation directly. According to the inventors of the residual block, the input propagation is called the "shortcut connection" [14]. The shortcut connections simply perform identity mapping, and their outputs are added to the outputs of the stacked layers. Identity shortcut connections add neither extra parameters nor computational complexity. The entire network can still be trained end-to-end by optimization methods with backpropagation, enabling us to train a much deeper network easily.

2.2.3 Residual Network

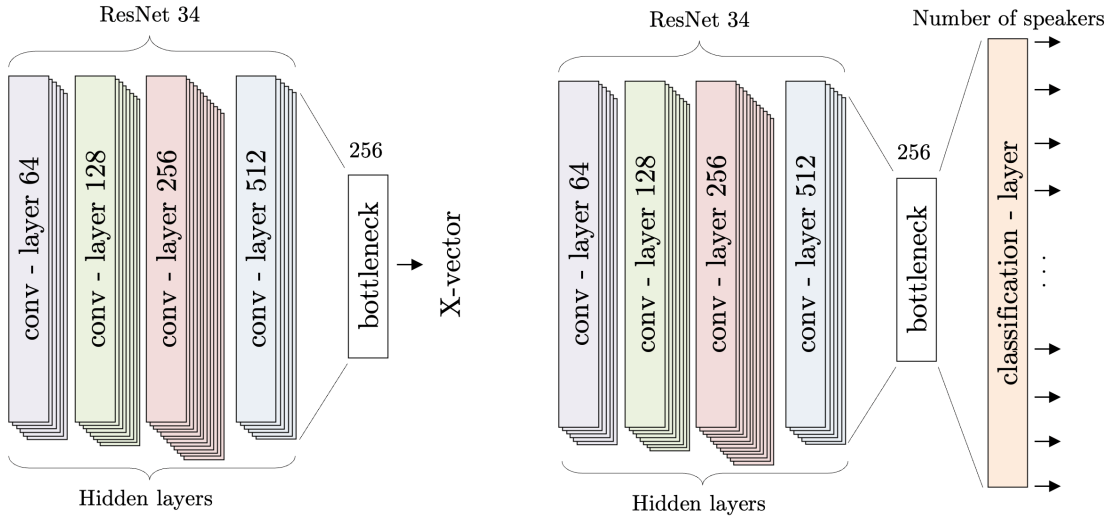
Residual networks (ResNets) are a famous structure for speaker embedding. Its trunk architecture is a two-dimensional CNN with convolutions in time and frequency domains. ResNet can be used for a variety of computer vision tasks. This model's first announcement was in 2015 in [14]. In the same year, ResNet was the winner of the ImageNet challenge. ResNet allowed us to train deep neural networks with 150 or more layers successfully and still achieve outstanding performance, a fundamental breakthrough. Our model works as a backbone, and its specific purpose is to extract the x-vectors.

Let us now look at the ResNet structure, which simplified form is shown in Figure 2.4. The ResNet comprises several (34) convolutional layers (CLs) that operate in a frame-by-frame manner. A global pooling layer follows the last one. This layer estimates the mean and the standard deviations of the output CL over time to obtain the fixed-length utterance representation, in our case, the x-vector. One more softmax layer is added when the ResNet is trained, serving as the classifier of training speaker identities. The same architecture used in this work is shown in Appendices B.

The ResNet's core building block *residual block* is shown in Figure 2.5. Formally, this building block can be defined as:

$$y = \mathcal{F}(x, \{W_i\}) + x \; . \tag{2.5}$$

Here x and y are the input and output vectors of the layers considered respectively. The function $\mathcal{F}(x, \{W_i\})$ represents the residual mapping to be learned. For the example in Figure 2.5 that has two layers, $\mathcal{F} = W_2\sigma(W_1 x)$ in which σ denotes ReLU and the biases are omitted for simplifying notations. The operation $\mathcal{F} + x$ is performed by a *shortcut connection* and element-wise addition. Which is followed by the second nonlinearity.



(a) DNN struncture for evaluation

(b) DNN struncture for training

Figure 2.6: The illustration of DNN structure both for evaluation and training. In the Figure 2.6a, there is visualization of the evaluation network structure where the key component is ResNet followed by the *bottleneck layer*. Function of this layer is to condensate the most relevant speaker information into 256 dimension embedding (x-vector). Figure 2.6b shows, that training network structure consist of the same elements as the evaluation one, plus one extra layer called *classification layer*. This layer is a key factor for weights optimization, where the output has the same number of dimensions as the number of speakers in training set. Coresponding value respresents concrete speaker, where the target speaker value of the output vector is set to clearly different value than values of the other speakers (depending on classification output format). This technique determines recognition result and enables loss computation which is crucial for backpropagation and so weights adjustement. More specific information about DNN training procedure contain subsection 2.2.4. Classification format can be linear (target speaker value 1, others 0) or position on unit globe (values are teoretically in range $(-\infty, \infty)$), etc.

The shortcut connections in Equation (2.5) introduce neither extra parameter nor computation complexity. It is attractive in practice and easy to compare plain and residual networks. We can fairly compare plain/residual networks with the same number of parameters, depth, width, and computational cost (except for the negligible element-wise addition).

2.2.4 DNN Training

This subsection describes the DNN training method and brings insights into used technologies. Training the complex neural network is a demanding task for time and computational resources. Train a feedforward neural network (acyclical connections between layers) involves adjusting biases and weights in each network neuron iteratively. For the weights and bias adjustment is a widely used *backpropagation algorithm* [11]. Before the backpropagation algorithm description, it is desirable to introduce the cost function, which is the key to the whole optimization problem.

A *cost function* [29] (referred as a *loss* function or an error function) translates a values of one or more variables onto an actual number intuitively, representing some "cost" associated with the event. The loss function is directly proportional to the predictions in the recognition model. The smaller the value of the loss function, the better will be the recognition results. The cost function evaluates the model performance and needs to be minimized to achieve improvements. The subject of minimization is called an optimization problem. An objective function is either a loss function or its opposite, in which case it is to be maximized. After a training example propagates through a network, the loss function calculates the difference between the network output and its expected output.

Let y, y' be vectors in $\mathbb{R}^n$. Calculate the difference between two outputs using the error function $E(y, y')$. It is standard to measure the distance between the vectors y and y' as the square of their Euclidean distance:

$$E(y, y') = \frac{1}{2}\|y - y'\|^2 \, . \tag{2.6}$$

The error function over n training examples can then be written as an average of losses over individual examples:

$$E = \frac{1}{2n} \sum_x \|y(x) - y'(x)\|^2 \, . \tag{2.7}$$

The supervised learning algorithms aim to find the best function that maps a set of inputs to their correct outputs. A multi-layer neural network is trained with backpropagation in order to learn its internal representation and thus to have the ability to learn any input to output mapping.

As part of fitting a neural network, backpropagation computes the gradient of the cost function concerning the network weights for a single input-output example efficiently, instead of calculating it naively for each weight separately. Gradient methods can train multi-layer networks and update weights to minimize losses (cost). The loss function gradient is computed by the backpropagation algorithm for each weight by the *chain rule*, computing the gradient of the one layer at a time. The backpropagation algorithm iterates backwards for each successive layer to prevent redundant calculations of intermediate terms in the chain rule. The chain rule expresses the derivative of two differentiable functions composition f and g in terms of those two derivatives. If $h = f \circ g$ is the function such that $h(x) = f(g(x))$ for every x, then the chain rule is (using Lagrange's notation)

$$h'(x) = f'(g(x))g'(x) \, . \tag{2.8}$$

The weights are fixed during model evaluation, while the inputs vary (and the target output may be unknown). The network ends with the output layer, not including the loss function. In contrast, the input-output pair is fixed during the model training, while the weights vary, and the network ends with the loss function.

- x: input (feature vector)
- y: target output - for classification, output will be a vector of class probabilities (0.1,0.6,0.3), and target output is a specific class, encoded by the one-hot/dummy variable (0,1,0)
- C: cost function - for classification, this is cross entropy

- L: number of layers
- $W^l = (w^l_{jk})$: the weights between layer $l-1$ and l, where w^l_{jk} is the weight between the k-th node in layer $l-1$ and the j-th node in layer l
- f^l: activation functions at layer l - for the multi-class classification, the last layer is a softmax (softargmax), while for the hidden layers the rectifier (ReLU) is being common.

In the derivation of backpropagation, other intermediate quantities are used. For example, the term *bias* is not treated specially, as it corresponds to a weight with a fixed input of 1. The specific cost function and activation functions do not matter for backpropagation while they and their derivatives can be evaluated efficiently. There are many traditional activation functions as described in Section 2.2.1, including but not limited to sigmoid, tanh, and ReLU. Overall, the network is a combination of matrix multiplication and function composition:

$$g(x) = f^L(W^L f^{L-1}(W^{L-1} \cdots f^1(W^1 x) \cdots)) \ . \tag{2.9}$$

There will be a set of input-output pairs for a training set, (x_i, y_i). For each input-output training set pair (x_i, y_i), the model loss on corresponding pair is the difference between the predicted output $g(x_i)$ and the target output y_i:

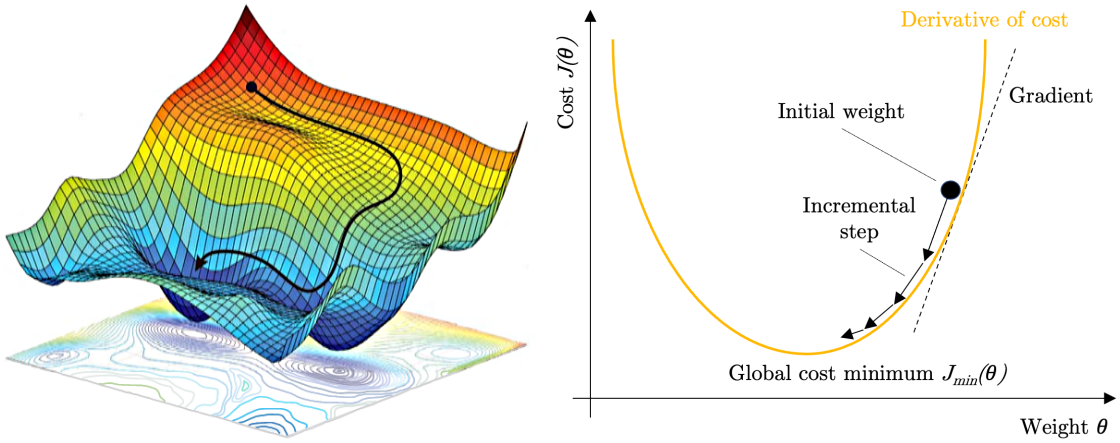
$$C(y_i, g(x_i)) \ . \tag{2.10}$$

Backpropagation computes the gradient for a fixed input-output pair (x_i, y_i), where the weights w^l_{jk} can vary. Each component of the gradient, $\partial C/\partial w^l_{jk}$, can be computed by the chain rule. Doing this separately for each weight is very inefficient. The backpropagation algorithm can efficiently compute the gradient by avoiding duplicate calculations. Additionally, by not computing extreme intermediate values (the gradient of each layer) – specifically, the weighted input gradient of each layer, denoted by δ^l – from back to front. Commonly used algorithms will be described later in the following subsection, along with the process explained in the context of this work.

2.2.5 DNN Training Optimization Techniques

The primary goal of this thesis is to implement these techniques into the functional model, thereby achieving better training performance and overall system improvement. Training DNNs usually requires a large amount of computational time and resources. As model sizes grow more prominent, training DNNs more efficiently and using less memory becomes increasingly important. It is desirable to have an optimized training process, ideally consuming as little computational power and time as possible [41].

Many fields of science and engineering benefit greatly from stochastic gradient-based optimization. Many problems can be understood in these fields as optimizing some scalar parameterized objective function that requires maximization or minimization of its parameters. Gradient Descent (GD) is an efficient optimization method when the function is differentiable because computing first-order partial derivatives for all parameters have the same computational complexity as evaluating the function. Objective functions are often stochastic. When many objective functions consist of a sum of subfunctions evaluated at different subsamples of data, optimization may be improved by applying SGD or ascent to



(a) 3D SGD visualization

(b) SGD overview

Figure 2.7: Demonstration of SGD algorithm. On the Figure 2.7a is a 3D SGD algorithm visualization of finding a global minima. The course of individual iterations is shown by an oriented curve starting at the top in a black circle and ending in the graph valley. The Figure 2.7b illustrate and explain the SGD process where the goal is to find the global minima of the cost function. SGD starts by initial weights value and in iterations slides down the slope, in the opposite direction of the cost function gradient where the step length is decreasing.

each subfunction. Several machine learning successes have been attributed to SGD, including recent advances in deep learning [19, 7, 12]. In addition to data subsampling, there may also be other sources of noise, such as dropout [15] regularisation. This calls for stochastic optimization techniques with high efficiency.

There will be an explanation of the algorithms used for DNN training optimization and a discussion of the newly used algorithms, hopefully improving performance and recognition accuracy expressed by ERR. As for the current solution used in the model, Mini-Batch SGD is applied to optimize DNN training.

Mini-Batch Stochastic Gradient Descent

GD is a popular algorithm used for optimization and is probably the most common method of optimizing neural networks. GD minimizes an objective function $J(\theta)$ parameterized by a model's parameters $\theta \in R^d$. The parameters are updated in the objective function $\nabla_\theta J(\theta)$ gradient opposite direction. We used the learning rate (LR) η to determine the step size taken to reach a (local) minimum. We will eventually reach a valley by following the slope of the surface created by the objective function downhill. A simple variant is a batch gradient descent, which computes the gradient of the cost function based on the parameters θ for the entire training dataset:

$$\theta = \theta - \eta.\nabla_\theta J(\theta) \tag{2.11}$$

Batch GD used to calculate the gradients for the whole dataset for just one update is very slow and even impossible for datasets not fitting into memory. Batch GD does not allow online model updates with new examples on-the-fly. In contrary SGD performs a parameter update for each training example $x^{(i)}$ and label $y^{(i)}$:

$$\theta = \theta - \eta.\nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.12}$$

Finally, Mini-Batch GD takes the best of the previous algorithms and performs an update for every mini-batch of n training examples:

$$\theta = \theta - \eta.\nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{2.13}$$

An increase in the update frequency reduces the variance in the adjusted parameters and thus to more stable convergence and using the matrix optimizations common to state-of-the-art deep learning libraries that enable computing the gradient in a mini-batch to be very efficient. The mini-batch size used in this work is 32, but this can vary for different applications. Standard mini-batch sizes range from 50 to 256. Mini-batch GD is generally the preferred algorithm when training a neural network. The term SGD is usually employed when mini-batches are used, as in this work.

SGD is an iterative method for optimizing an objective function with smooth properties. By the word "stochastic", we understand a system or process linked with a random probability. It is a stochastic approximation of gradient descent optimization because the actual gradient (calculated from the entire data set) is replaced with an estimate (calculated from a randomly selected subset of the data). In other words, SGD randomly picks one data point from the whole data set at each iteration to reduce the computations enormously. Especially in high-dimensional optimization problems, this reduces the computational burden, achieving faster iterations in trade for a lower convergence rate [30].

Challenges of the SGD

Mini-batch gradient descent does not guarantee good convergence but offers a few challenges that need to be addressed:

- It is not easy to choose a reasonable LR. A too-small LR leads to really slow convergence, while a too-large LR hinders convergence and leads to the loss function fluctuating around the minimum or diverging entirely.
- LR schedules try to adjust the LR during training by, e.g. annealing, i.e. reducing the LR according to a pre-defined schedule or when the change in objective between epochs falls below a threshold. However, these schedules and thresholds must be defined in advance and can not adapt to a dataset's characteristics.
- Similarly, all parameter updates are subject to the same LR. Data with sparse features and features with very different frequencies may not require updating all of them to the same extent, but performing an extensive update for rare features could be beneficial.
- Another critical challenge of minimizing highly non-convex error functions standard for neural networks is avoiding getting trapped in their numerous suboptimal local minima. Even more significant problems occur at the saddle points (one dimension slopes up, and the other slopes down). The saddle points are usually surrounded by the same error values, which causes a challenging escape for SGD, as the gradient is close to zero in all dimensions.

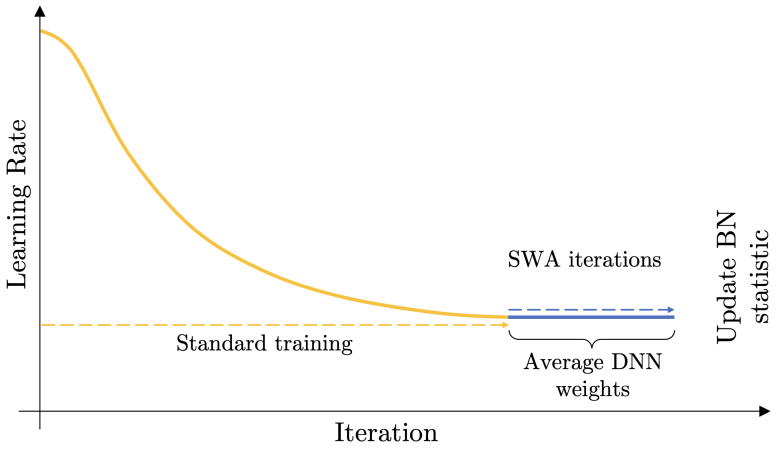


Figure 2.8: Demonstration of SWA algorithm. Displaying one possible scenario of applying the SWA algorithm to the training process used in this work. The graph shows the course of the LR value through the training process and the differences between the standard training and SWA phase. After the model is fully trained, SWA is used in the form of additional iterations.

Adaptive Moment Estimation

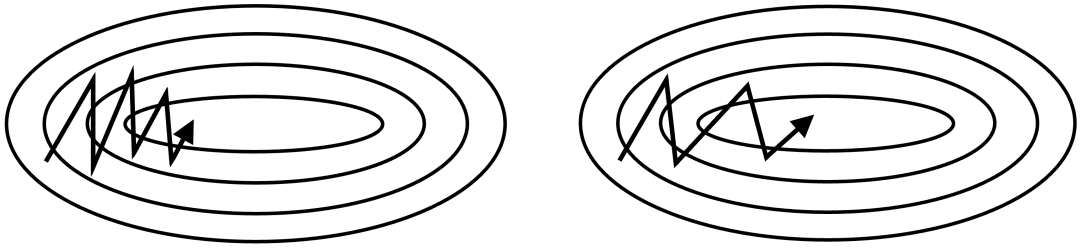
Adam is a popular method for finding the local minima of a function. With this method, there is no need for higher-order gradients for efficient stochastic optimization. Adaptive LRs are computed based on estimates of each parameter's first and second moments of gradients. Two popular methods are combined: Adaptive Gradients (referred to as AdaGrad [8]), which provides good results for sparse gradients, and Root Mean Squared Propagation (referred to as RMSProp [38]), which works well in online and non-stationary settings. One of Adam's advantages is that the magnitudes of parameter updates are invariant to rescaling of the gradient, the stepsize hyperparameter approximately bounds its stepsizes, it does not require a stationary objective, it works with sparse gradients, and it naturally performs a form of step size annealing [18].

Stochastic Weight Averaging

SWA is recognized as a simple but effective way to improve the generalization of SGDs for training DNNs. Its success can be explained by averaging weights through an SGD process equipped with cyclical or high constant training rates that can help uncover wider optima, leading to better generalization. It means to find more accurate results [13].

Averaged SGD is often used with a decaying LR and an Exponential Moving Average (EMA), typically convex optimization. Improvements in convergence rates have been the focus of convex optimization. This form of averaged SGD smoothes the trajectory of SGD iterates but does not perform much differently. In contrast, SWA uses an equal average of SGD iterates with a modified cyclical or high constant LR and exploits the flatness of training objectives [24] specific to deep learning for improved generalization.

Two essential ingredients make SWA work. First, SWA uses a modified LR schedule. SGD (or Adam) continues to bounce around the optimum and explore diverse models instead of single solution convergence. For example, in the first 80% of training time, the standard decaying LR strategy can be used and then the LR can be set to a reasonably high



(a) Plain SGD (b) SGD with momentum

Figure 2.9: Demonstration of SGD with momentum involved.

constant value for the remaining 20% of the time. The second improvement is to take an average of the weights (an equal average) of the networks traversed by SGD. For example, we can calculate the running average of weights obtained at the end of each iteration within the last 20% of training time. After completing the training process, the network's weights are set to the computed SWA averages.

One of the most unanswered questions in deep learning is why SGD finds reasonable solutions. Multiple settings of parameters can accomplish no training loss but result in poor generalization. Understanding geometric properties like flatness, which relate to generalization, can help resolve these questions and build optimizers that provide much better generalization and other valuable properties, like uncertainty representation.

Momentum

According to Section 2.2.5, SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in a different dimension, which are common around local optima. In these scenarios, SGD oscillates across the ravine's slopes while only making hesitant progress along the bottom toward the local optimum, as in Figure 2.9a.

Momentum [28] is a method that helps accelerate SGD in the relevant direction and dampens oscillations as can be seen in Figure 2.9b. It does this by adding a fraction γ of the past time update vector step to the current update vector:

$$\begin{aligned} v_t &= \gamma_{vt-1} + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{2.14}$$

The usual value of the momentum γ is 0.9 as in my experiments. Momentum tends to pull a ball downward. With each successive step, the ball gains momentum (until it reaches its terminal velocity if there is air resistance, i.e. $\gamma < 1$). Our parameter updates are impacted by the momentum term as well. Updates increase for dimensions whose gradients point in the exact direction and decrease for dimensions whose gradients reverse directions. Consequently, we achieve faster convergence and a lower oscillation.

Batch Normalization

An additional strategy can be used alongside the SGD algorithm to improve the performance further.

Batch normalization [16] (BN) is a layer that dynamically adjusts (normalizes) the output range of the previous layer and thus helps to optimize the parameter. BN restores normalization for each mini-batch, and changes are backpropagated as well. Thanks to this normalization as a part of the model architecture, we can use higher values of the LRs. Therefore, the initialization of the parameters does not play a significant role. BN can reduce (and sometimes even eliminate) the dropout need. The relationship describing the normalization between every layer is as follows:

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} * \gamma + \beta \ . \tag{2.15}$$

We calculate each dimension's mean and standard deviation over the mini-batches. Our parameters γ and β are learnable parameter vectors of C size (where C is the input size). The elements of γ are set to 1 by default, while the elements of β are set to 0. The standard deviation is calculated via the biased estimator. Also, by default, this layer keeps running estimates of its computed mean and variance during training, then used for normalization during evaluation.

2.3 Postprocessing

Since the first time fixed vectors have been used, it has been shown that it is typically necessary to post-process them before using them for the final classification in order to impose some useful properties on them.

The first step is mean subtraction, which centres the x-vector around the zero coordinate. Then Linear Discriminant Analysis (LDA) lowers the x-vector's dimensionality; in our case from 256 to 200, but more drastic reductions are not uncommon. The following step is length normalization. Lastly, length normalization is applied so that the Euclidian length of the vector is constant. Let us now take a look at each step in detail.

2.3.1 LDA

LDA was developed in 1936 by Ronald A. Fisher, named Linear Discriminant and described as a two-class technique. The multi-class version came later by C. R. Rao as a Multiple Discriminant Analysis. It is commonly used in pattern recognition to find new orthogonal axes to better discriminate between classes. The primary purpose of LDA is to maximize the between-class variance (S_b) and simultaneously minimize the within-class variance (S_w) of a speaker population (a crucial aspect of speaker verification):

$$S_b = \sum_{l=1}^{L} (\phi_l - \overline{\phi})(\phi_l - \overline{\phi})^t, \tag{2.16}$$

where L is the number of speakers, ϕ_l is the mean of the x-vectors of each speaker and $\overline{\phi}$ is the global mean vector of the speaker population, and

$$S_w = \sum_{l=1}^{L} \frac{1}{n_l} \sum_{i=1}^{n_l} (\phi_i^l - \overline{\phi})(\phi_i^l - \overline{\phi})^t, \tag{2.17}$$

where n_l is the number of utterances for speaker l and ϕ_i^l is the vector of the i-th utterance of speaker l.

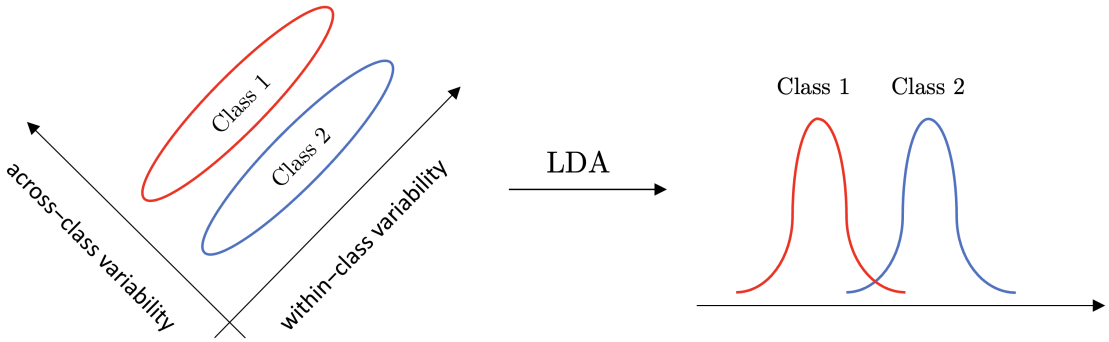


Figure 2.10: Demonstration of LDA algorithm.

Then, the ratio (referred to as the Rayleigh coefficient) representing the amount of information between S_b and S_w is defined as:

$$J(v) = \frac{v^t S_b v}{v^t S_w v} \,. \tag{2.18}$$

where v is a given space direction. The LDA approach uses discriminative criteria to eliminate unwanted directions and minimize the amount of information removed about the variance between speakers. For more information, see [6].

2.3.2 Length Normalization

The Probabilistic Linear Discriminant Analysis (PLDA)—mechanism explained later—assumes that the input x-vectors are normally distributed. However, this assumption is almost always violated and various modifications to the PLDA algorithm have been proposed to tackle this issue (e.g. [17]). In our work, we use length normalization.

This procedure simply scales the lengths of each x-vector to unit length. The transformation is given as

$$\bar{\phi} = \frac{\phi}{\|\phi\|} = \frac{\phi}{\sqrt{\phi' \phi}} \,. \tag{2.19}$$

Length normalization [9] forces the x-vectors to lie on a unity sphere. They become closer to the Gaussian distribution shell, where most of the probability mass is concentrated [10]. This normalization ensures a more accurate score computation results in the next step.

2.4 Scoring Mechanism

This section aims to discuss how to process the output from the previous component so that the recognition task can continue. The section focuses on the verification score generating algorithm that compares two embeddings. The input to this stage is a pair of (post-processed) x-vectors. A speaker verification (likelihood) score is obtained using the PLDA.

2.4.1 PLDA

Given a pair of x-vectors, PLDA computes the log of the ratio of the likelihoods for the same-speaker hypothesis and the different-speaker hypothesis [25, 17]. We can consider a special kind of PLDA (the two-covariance model), in which both speaker and channel variability is assumed to be Gaussian, with their variability described by across-class and within-class matrices Σ_{ac} and Σ_{wc}, respectively. Speaker identity is represented mathematically by a hidden variable y whose prior distribution is assumed to be

$$p(y) = N(y; \mu, \Sigma_{ac}) \ . \tag{2.20}$$

For a known speaker, represented by vector $\hat{y}$, the distribution of x-vectors is given as

$$p(\phi|\hat{y}) = N(\phi; \hat{y}, \Sigma_{wc}) \ . \tag{2.21}$$

In general PLDA, the covariance matrices do not necessarily have to be full-rank. The x-vectors can be decomposed as

$$\phi = \mu + Vy + Ux + \epsilon \ , \tag{2.22}$$

where μ describes the global mean of observed data, V the speaker subspace, y is a hidden variable representing the speaker, U describes the channel subspace, x is a hidden variable representing the channel, and ϵ is a variable representing the residual data noise. Once x and y are known, Equation (2.22) can be computed. In its simplest form, PLDA imposes Gaussian priors on the variables:

$$\begin{aligned} p(y) &= N(y; 0, I) \\ p(x) &= N(x; 0, I) \\ p(\epsilon) &= N(\epsilon; 0, D-1) \ , \end{aligned} \tag{2.23}$$

where D is a diagonal precision matrix of the residual data variability. The PLDA within-class covariance would be $\Sigma_{wc} = UU'$, and the across-class covariance would then be given as $\Sigma_{ac} = VV'$. PLDA was jointly optimized with the DNN embeddings and used directly as the metric for the pair-wise speaker comparisons of the evaluation.

Now let us review that the speaker verification score is a function of the trial (the x-vectors pair ϕ_1, ϕ_2), and it is computed in a balanced way. For illustration, the x-vector pair is tested to determine whether the same speaker produced the pair ($\mathcal{H}_1$) or not ($\mathcal{H}_2$). Afterwards, the trial score is computed as the log-likelihood ratio between the two hypotheses, as defined by Equation (2.1):

$$s(\phi_1, \phi_2) = log \frac{p(\phi_1, \phi_2|\mathcal{H}_1)}{p(\phi_1, \phi_2|\mathcal{H}_2)} \ . \tag{2.24}$$

2.5 Evaluation Metrics

As a next step, let us look at how to evaluate the performance of speaker verification systems. As stated at the beginning of Chapter 2, a speaker verification trial x is defined as an utterance pair $x = \langle d_1, d_2 \rangle$. Evaluation requires a *test set* of *supervised trials* which are characterized by a label $h_x \in \mathcal{H}_1, \mathcal{H}_2$. Label depends on whether the two utterances come from the same or different speakers. A test set X consists of same-speaker and different-speaker supervised trials X_1 and X_2, also called *target* and *non-target* trials, respectively.

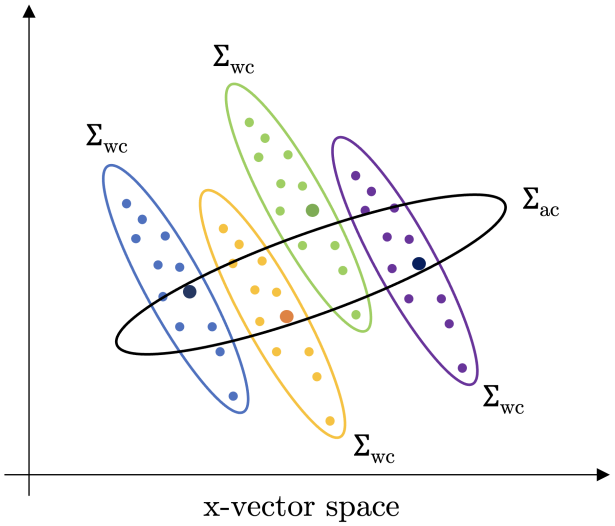


Figure 2.11: Displaying the PLDA algorithm in the x-vector space: the bold points represent the speaker identifiers. By assuming that the speaker identity y is known, we can determine the conditional distribution of the x-vectors by looking at the within-class covariances, represented by ellipses around the speaker identities.

Then the speaker verification system assigns a correct label for each trial. By other words classify the trial as $\mathcal{H}_1$ or $\mathcal{H}_2$. Two possible error scenarios can arise:

- **FA** - False alarms error arises when the system classifies the different-speaker trial as same-speaker.
- **Miss** - Missed detections error arises when the system classifies the same-speaker trial as a different speaker.

Detection errors probabilities for given test sets can be estimated as follows:

$$\begin{aligned} p(miss|X) &= \frac{|X_1|}{N_{miss}} \\ p(fa|X) &= \frac{N_{fa}}{|X_2|} \ , \end{aligned} \tag{2.25}$$

where $|X_1|$ and $|X_2|$ are the numbers of same and different-speaker trials, respectively. N_{fa} and N_{miss} are the numbers of both the false alarms and missed detections that occurred during the system evaluation, respectively. The output of the system evaluation is a likelihood score. The value of the score is directly proportional to the similarity of the speakers in the trial. An increased value represents a similar-speaker hypothesis, whereas a decreasing value represents a different-speaker hypothesis. Through thresholding, the score becomes a hard decision. Shifting the threshold t enables the user to configure the system's operating point, balancing the two error types. This phenomenon can be used for securing the system by intentionally increasing the threshold value to ensure virtually no false acceptance. For instance, this approach is used by bank verification systems to identify customers through phone calls. On the other hand, eliminating false rejection by decreasing the threshold value can help spot the terrorist in many phone calls or other available recordings by security services.

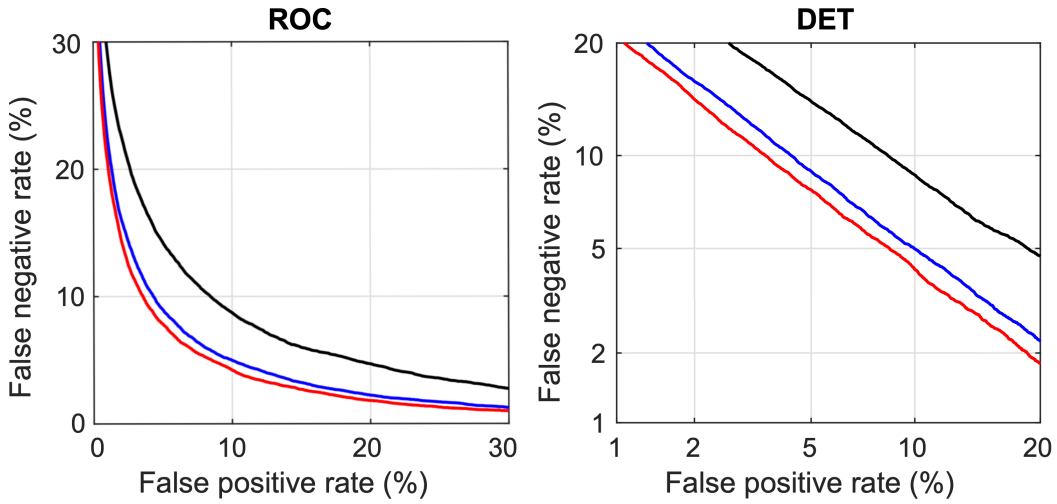


Figure 2.12: Comparison of ROC and DET curves for three different systems. Shows differences in the readability of a graph displaying the same information. Figure serves just as an example, it does not describe any of the results of this work.

$$\begin{aligned} p(miss|X) &= p(miss|X,t) \\ p(fa|X) &= p(fa|X,t) \quad . \end{aligned} \tag{2.26}$$

The SR system operation is evaluated in the evaluation phase when the system operates as in the recognition phase. However, in addition, it has information about the correct recognition result. When evaluating the system's operation, it is also necessary to state the conditions under which the evaluation of the system was achieved. The system evaluation outcome depends on several factors, e.g. the amount of data used for training and testing, the microphone and recording equipment used to record the reference and test data, ambient noise level, etc. Details about specific conditions throughout individual experiments are specified in Section 2.6.

2.5.1 System Performance Plotting

When we need to evaluate the system on a given dataset, it is desirable to visualize the errors for different thresholds. In the SR community, the detection error tradeoff (DET) graph is commonly used. It is an alternative to a commonly used receiver operating characteristic (ROC), and it plots the two types of errors on non-linearity transformed x and y axes. An example of such a plot is in Figure 2.12 on the left.

ROC curves can be used to evaluate the effectiveness of speaker verification systems. In ROC analysis, two probabilities are assigned non-linearly to the vertical and horizontal axes, the probability of incorrect rejection and incorrect acceptance. A DET curve can also be used, in which false rejection and false acceptance rates are assigned to vertical and horizontal axes, respectively. The error curve is usually plotted on the normal deviation scale. When the true speaker and impostor scores are Gaussian with the same variance, the SR system produces a linear curve with a slope of 1. Because DET curves are more easily readable than ROC curves, one can easily compare the system's performance over various

operating conditions. For illustration see Figure 2.12 and for more details about plotting metrics see [3]. Note that, although the DET and ROC curves are a nice way to display the result, for our purposes we will use the EER, which is described in the following paragraph.

2.5.2 Equal Error Rate

It is often required to report a system performance using a single number instead of a complex figure like the DET plot. In the SR community, it is common to report the EER—an operating point where the two errors are equal. The point is the intersection of the curve and the x = y line on a DET plot. In other words, it corresponds to the threshold at which the false acceptance rate is equal to the false rejection rate. This rate is a commonly accepted overall measure of system performance. It is important to emphasize that this is our primary metric. And with the help of which we will present the achieved results.

2.6 Datasets

This section aims to familiarise the reader with the terminology and the data qualities. It would be beneficial to introduce and explain this issue. Refer to them later in Chapter 3. The SR system is built in two steps: first, the system is trained and evaluated. Separate data sets are needed for both steps, which must be carefully selected to ensure the system generalizes when faced with unknown data. The data sets are as follows, in the order of the steps:

- **Training set** (referred to as background set) is a large corpus for estimating the robust parameters. There are usually several thousand hours in this set, which can be used to train.
- **Development set** (referred to as heldout set) is usually used to tune the system parameters. In other words, to train the backend component, in order to evaluate the system's performance.
- **Evaluation set** (referred to as test set) is used to report the final system performance. Ideally, the performance of the developed system will correlate positively with that of the development set.

2.7 Databases

This section contains the description of the data corpora that were used to build the datasets as described in Section 2.6.

2.7.1 NIST SRE

Many Speaker Recognition Evaluations [26, 20, 36, 32, 35, 34, 31](SRE) have been completed by the National Institute of Standards and Technology (NIST) of America in the past years. SRE is by far the most popular challenge in SR. Further information is available in [1]. Presented are experiments using the BUT SRE 2020 Dev as described later.

Note that there are numerous tests involved in each of these evaluations, commonly referred to as *conditions*, but it is beyond the scope of this thesis to report these. See the following paragraphs for a more detailed description of the individual databases.

2.7.2 Voxceleb 2 - Training Data and Augmentations

For all fixed systems, we used the development part of the VOXCELEB-2 dataset [21] for training. This set has 5994 speakers spread over 145 thousand sessions (distributed in approx. 1.2 million speech segments). We used the original speech segments and their augmentations to train DNN-based embeddings. The Kaldi recipe[4] was used in the augmentation process, and it resulted in additional 5 million segments belonging to the following categories [43]:

- Reverberated using RIRs[5]
- Augmented with Musan[6] noise
- Augmented with Musan music
- Augmented with Musan babel

2.7.3 NIST SRE - Development data

4368 speakers in 7612.09 hours of speech compiled from the NIST 2004-2012 SRE Evaluations[7]. All data was further augmented in the same fashion as the Voxceleb set.

2.7.4 BUT SRE 2020 DEV - Evaluation data

This set is a compilation of data gathered from the NIST 2016-2019 evaluations as an auxiliary development set for the 2020/2021 evaluations[8]. There are 1355 enrollment speakers in the set, each having 71 seconds of speech on average. There are also more complex speeches in this set, which is why the results are around 10% EER.

[4] https://github.com/kaldi-asr/kaldi/tree/master/egs/sre16/v2
[5] http://www.openslr.org/resources/28/rirs_noises.zip
[6] http://www.openslr.org/17/
[7] See [1] and references therein for more information
[8] [34, 31]

Chapter 3

Experiments

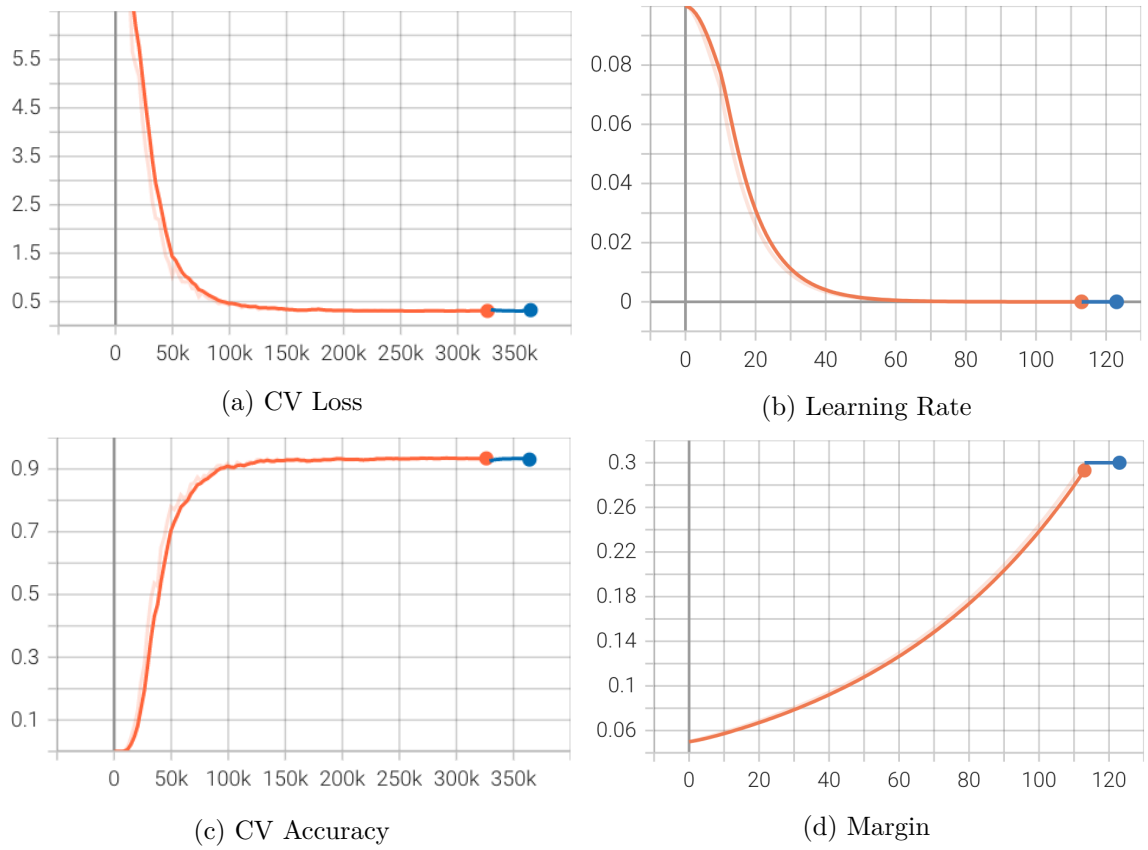
This chapter introduces the techniques used in this work and describes my implementation, especially experiments I performed and then evaluated. Additionally, there will be a section on the main technical problems which occurred during the implementation and system manipulation.

In the beginning, I obtained source codes along with the previously trained model from my supervisor. This model is commonly used by researchers from the Department of Computer Graphics and Multimedia (DCGM) in the Faculty of Information Technologies (FIT) in SR projects. The task was to replicate the evaluation and training baselines described earlier in Chapter 2. The first step was to extract the x-vectors on the NIST dataset for later evaluation using the PLDA backend. After I trained my model a successful system evaluation followed. Evaluation brought the value of EER equal to **11.17%** achieved on the NIST SRE20 DEV (referred to earlier in Section 2.7.4) dataset with the help of 2 GPUs for training. This value will be our baseline for further achievements.

Now let us approach parallel processing and the associated final number of training iterations. The number of iterations depends on the number of used GPUs. Additional GPUs benefit from parallel processing, where each GPU takes input data intended for one original (when used only one GPU) iteration, and after the iteration is done, parallel processes synchronize with each other and continue to the next iteration. In other words, when the training process runs on 2 GPUs, the first GPU takes data initially intended for the first iteration, and the second GPU takes data initially intended for the second iteration. Therefore in one iteration, two originals are processed. The initial number of iterations (for the used dataset) is 226; thus, using 2 GPUs results in 113 iterations, as can be seen in Table 3.1.

Regarding the graphs showing the course of individual values interesting for the evaluation of the system behaviour during the training process, the true values are always displayed in a paler colour of the given value course. The darker colour symbolizes a waveform that has been smoothed by a coefficient (0.6) because the real values are limited by the number of iterations and do not completely show the exact waveform bound to the LR function. Another reason is for better visualization. Due to the need to fit the most important parts of the graphs into the figures, it is necessary to fine-tune the waveforms, especially when displaying Loss and Accuracy waveforms using cyclic LR values. In the absence of this coefficient, the graph was unreadable.

It should also be noted that the graphs used to display the LR values differ from the values in the configuration tables. This is because when using parallel training, a coefficient is used to multiply the LR based on the number of used GPUs. In other words, if the training



(a) CV Loss

(b) Learning Rate

(c) CV Accuracy

(d) Margin

Figure 3.1: Experiment number 1: there are 4 elements or rather say indicators which sufficiently describe the course of the training process. From the Figure 3.1a can be seen sharp loss reduction in the first quarter of training caused by the combination of high LR value and small margin value. This mean that the fast convergence is provide by the big steps (high LR) toward the optimal state and small penalty (small margin)—margin only affects Loss and Accuracy on training data. By the decreasing the LR, the loss slope starts to behave more like a constant. Analogicaly, the Figure 3.1c displays systems accuracy. At the point when the SWA is involved, the Loss and Accuracy value made little step away from optimal state (difference in LR value, in Figure 3.1b, the LR in second half is shown to be a constant, but in the orange part it is the $\lim_{n\to\infty} 0$, until the SWA part is constant, namely $1e^{-6}$) and gradualy made progress toward the optimum. Unfortunately, the number of iterations with SWA involved is too small to have a positive effect on the model optimization. Moreover, the value of LR is too small to achieve performance improvement. Figures 3.1a and 3.1c shows Loss and Accuracy achieved on the Cross Validation (CV) data—unseen data through the training process. CV data are processed after every training iteration to evaluate the current model Loss and Accuracy. Therefore, the horizontal axis represents data positon from the evaluation dataset, not iterations like other two graphs. The same situation will be used in all following graphs.

is running by 1 GPU, the values in the relevant table would correspond to the LR graphs. In our case, the coefficient is equal to 2 (2 GPUs), so in the LR graph, it is twice as large as in the table.

Table 3.1: Training configuration used in the first experiment.

	Baseline	SWA model
Optimizer	mini-batch SGD	mini-batch SGD + SWA
Start iteration	0	113
Number of iterations	113	11
Momentum	0.9	0.9
Initial LR	0.1	$1e^{-6}$
Final LR	$1e^{-6}$	$1e^{-6}$
Initial margin	0.05	0.3
Final margin	0.3	0.3
Mini-batch size	32	32

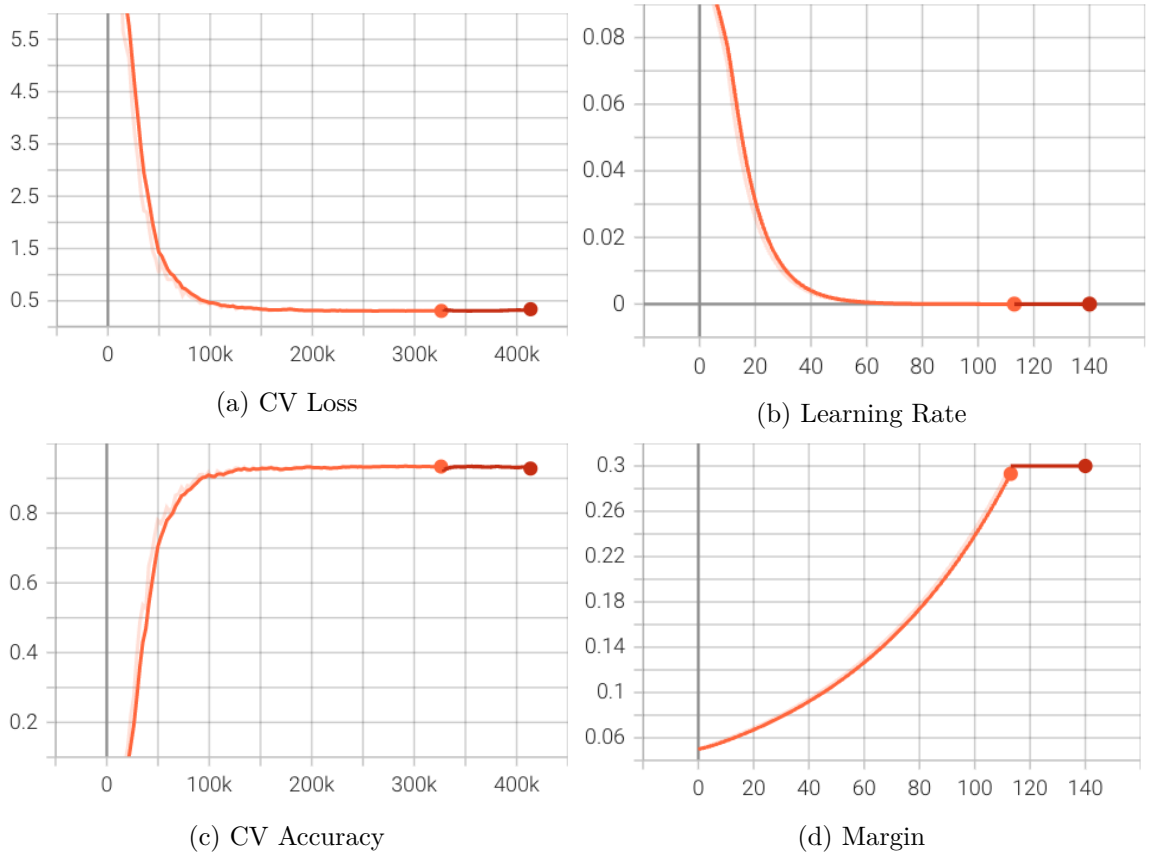
3.1 Experiment 1

After successfully executing the training and evaluation processes came time for the first experiment. This experiment is based on applying the SWA optimization algorithm to the already trained model in the form of additional training iterations.

The experiment used a model based on ResNet with 34 layers with a 0.9 momentum SGD optimizer, mini-batch size 32. After involving the SWA algorithm, training runs for an additional 10% of the overall iteration number (for 2 GPUs, it corresponds to 11 iterations).

3.1.1 Result

The first experiment showed that incorporating the SWA algorithm with the given configuration in the form of extra iterations after baseline training did not bring any accuracy improvements. It even achieved 0.1% worse accuracy (baseline 11.17%) with an EER of 11.27%. Several factors may have influenced this result, such as low LR levels. After averaging the models, we came to a worse setting of DNN parameters. The averaging time (SWA training) also had a lesser influence, when 10% of the added iterations were insufficient to improve the model accuracy. Based on the experience from this experiment, I performed different configurations of the training process, which are described in the following experiments.



(a) CV Loss

(b) Learning Rate

(c) CV Accuracy

(d) Margin

Figure 3.2: Experiment number 2: except a few details, the course of all values shown in the graphs are identical to the courses from experiment number 1. One difference is the number of SWA iterations and the other one is the behavior of the SWA curve in Figures 3.2a and 3.2c. In the first three quarters of SWA training, the model converges to a better solution. Nevertheless, in the last quarter, the direction reverses and begins to deteriorate. This may indicate that the model is overtrained or the LR is too small and cannot get over the saddle point in the Loss function space.

3.2 Experiment 2

In this experiment, the same training configuration is used as in the previous one, but the number of SWA iterations changed. For detailed configuration see following Table 3.2.

According to the result from the first experiment, where the additional 10% iterations were insufficient to improve the model performance, I conducted another experiment and increased the number of iterations to 25% (for 2 GPUs, it corresponds to 28 iterations). Hopefully, the increased averaging time of the model parameters is sufficient to converge to a more optimal point.

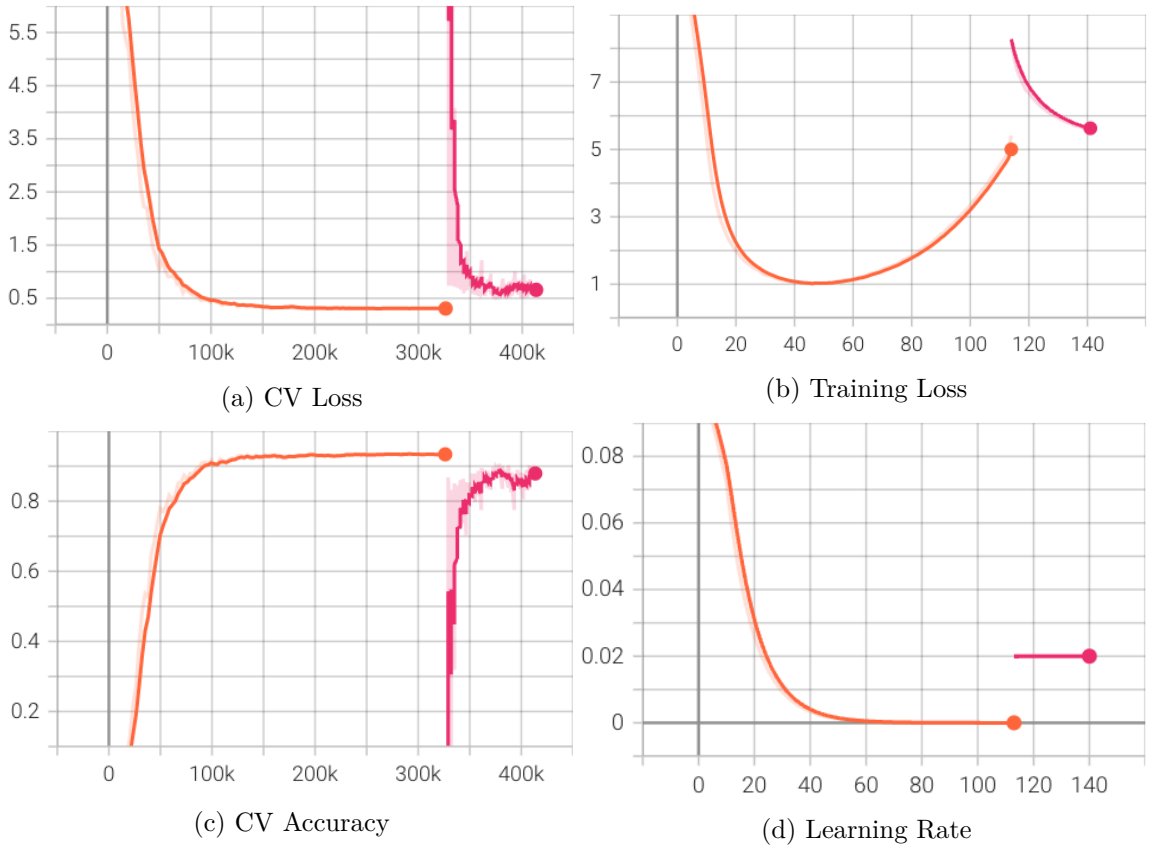
3.2.1 Result

The second experiment proves that only prolonging the SWA training with keeping the same small LR does not enhance the system's performance. After involving the SWA optimization algorithm with constant LR equal to $1e^{-6}$ and 25% additional iterations, a little worse

Table 3.2: Training configuration used in the second experiment. In order to reduce duplicity and less significant information for course of this work, the following training attributes are excluded from all future tables: *Start iteration*, *Momentum*, *Initial margin*, *Final margin* and *Mini-batch size*. For the same reasons, a simplified notation will be used to symbolize SWA training. Since it is only an auxiliary algorithm that cannot be used alone, we implicitly assume that it is used with the mini-batch SGD algorithm (unless stated otherwise). Therefore, from now on, only "SWA" will be listed in the following tables.

	Baseline	SWA model
Optimizer	mini-batch SGD	SWA
Start iteration	0	113
Number of iterations	113	28
Momentum	0.9	0.9
Initial LR	0.1	$1e^{-6}$
Final LR	$1e^{-6}$	$1e^{-6}$
Initial margin	0.05	0.3
Final margin	0.3	0.3
Mini-batch size	32	32

accuracy was achieved, EER 11.26%. Now let us look at the following experiments, where finally different LR is applied and if it will lead to some performance enhancements.



(a) CV Loss

(b) Training Loss

(c) CV Accuracy

(d) Learning Rate

Figure 3.3: Experiment number 3: In order to display more important information and to reduce the duplicity (margin has the same course in all experiments), the margin must have been excluded from training graphs in next few experiments. The Figures 3.3a and 3.3c expediently describe the problem that occurred with a step change in the LR value. There was a very sharp deterioration in performance, so great that the visualization did not fit into the graph. According to the wavy course of SWA training, I conclude that the value of LR is too large and causes sudden deterioration in performance—bouncing away from the optimum. From the Figure 3.3b can be seen sharp loss reduction in the first quarter of training caused by the combination of high LR value and small margin value. This mean that the fast convergence is provide by the big steps (high LR) toward the optimal state and small penalty (small margin). By the decreasing the LR and incerasing the margin value, the loss slope starts to steep. Analogicaly, the steps toward the optimal state are very small and a penalty grows continuously. This continues only to the point when the SWA is involved (LR and margin stays fixed), then gradually—after the big performance drop—the loss value starts to decrease.

3.3 Experiment 3

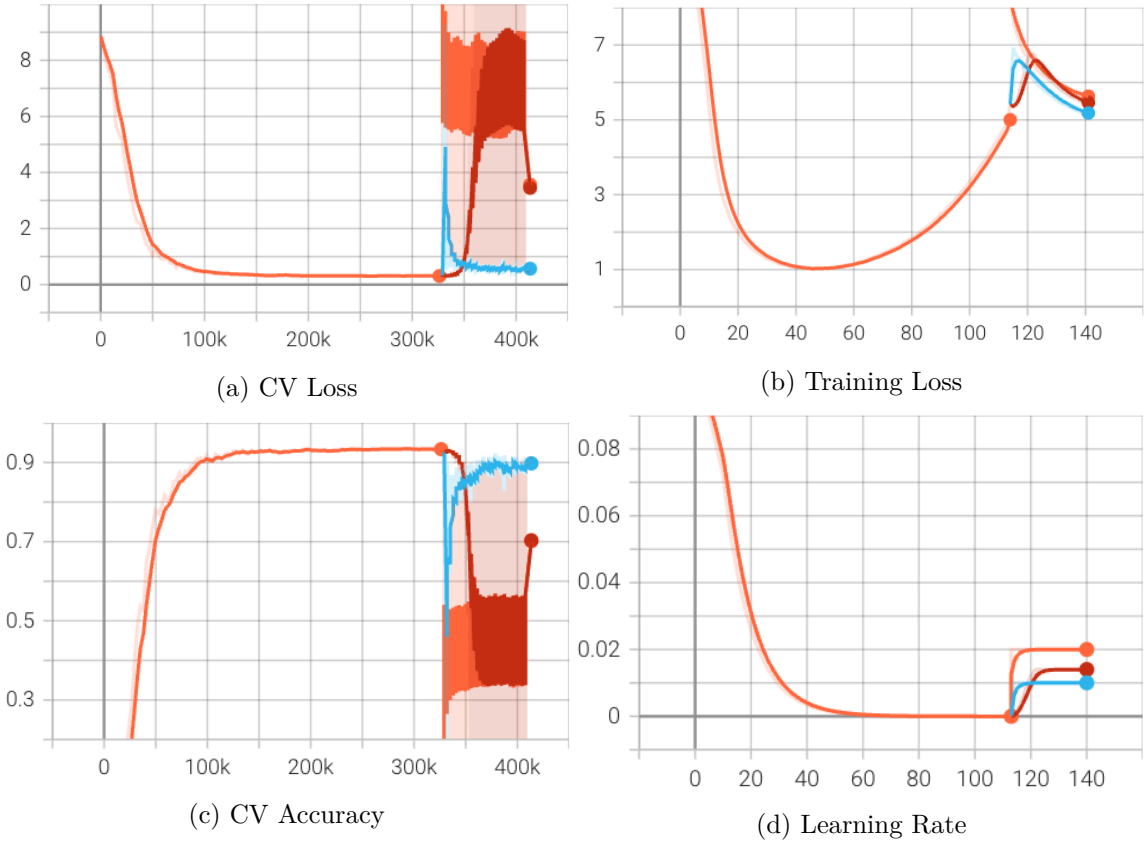
After I gained experience from previous experiments, the time has come for a change in LR value. As can be seen in Table 3.3, the same training configuration is used, while the LR is kept constant but set at 0.01. The number of additional iterations is still kept at a value of 25%. However, the LR value is set to a much higher value to verify that a higher value will lead to faster convergence than in the previous experiments.

Table 3.3: Training configuration used in the third experiment.

	Baseline	SWA model
Optimizer	mini-batch SGD	SWA
Number of iterations	113	28
Initial LR	0.1	0.01
Final LR	$1e^{-6}$	0.01

3.3.1 Result

As shown in Figures 3.3a and 3.3c, even an increase in the LR value was not enough to improve system performance. On the contrary, the jump in LR value at the start of SWA training caused a huge drop in system accuracy, which could no longer be caught up. The resulting accuracy is equal to 11.54 % EER. Therefore, it is necessary to smooth the difference between LR from the end of baseline training and the beginning of the SWA training, which is described in the next experiment.



(a) CV Loss

(b) Training Loss

(c) CV Accuracy

(d) Learning Rate

Figure 3.4: Experiment number 4: when showing the SWA training courses in Figures 3.4a and 3.4c, we come to the problem described in the introduction to this chapter. In the case without the smoothing coefficient used, we would not be able to read anything from the Loss and Accuracy waveforms. In this way we can see a sharp drop in performance with a strong onset of oscillation. This is mainly due to a step transition to a high LR value. We can say that any significant change in LR (especially with increasing value) can adversely affect the results on the CV data. Figure 3.4b describes these facts in a more readable way, but shows the Loss only on the training data, which is not so truthful. Here can be seen a clear delay in the behaviour while use of cosine annealing (the red curve).

3.4 Experiment 4

Previously, we discussed the performance drop caused by the jump change in LR value. Three variants of continuous transition are being announced to resolve this issue. Still considering the same training configuration, but using a vertical step, cosine annealing and sloping step before setting constant LR value as can be seen in Table 3.4. In order to unify the related experiments, which differ only in small details, three configurations meet in this experiment. The true course of LR values without the use of the smoothing coefficient is shown in Figure 3.5. The number of additional iterations does not change.

3.4.1 Result

By gradually adjusting the configuration of the training process using the SWA algorithm, it was finally possible to achieve an improvement in recognition accuracy. The results

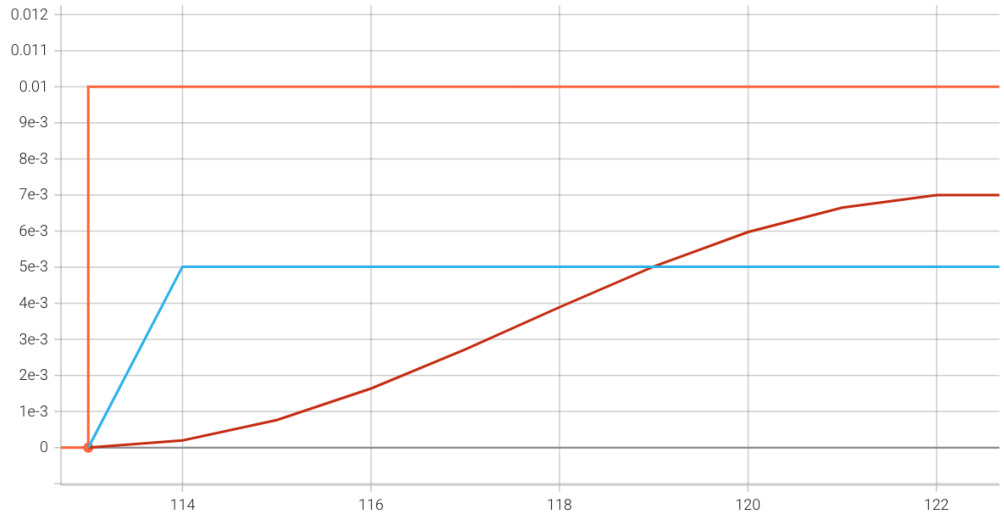


Figure 3.5: Experiment number 4, Effective Learning Rate detail: the detailed Effective LR (without a factor taking into account the number of GPUs) values without smoothing, displaying LR courses between iterations no. 113 and 123. Here you can see the difference in the individual LR waveforms, when the orange curve starts with a steep vertical jump, the blue curve jumps in the next iteration and the red curve reaches a constant value in the 9 iterations of SWA training.

Table 3.4: Training configuration used in the fourth experiment. In order for the table to fit in width, the "model" from the training process name had to be omitted. The same notation is applied in the following experiment no. 5.

	SWA 1 (blue)	SWA 2 (red)	SWA 3 (orange)
Optimizer	SWA	SWA	SWA
Number of iterations	28	28	28
Initial LR	$1e^{-6}$	$1e^{-6}$	$1e^{-6}$
Final LR	0.005	0.007	0.01
LR course	sloping step 0.005	cosine annealing 0.007	vertical step 0.01

obtained are as follows: 11.21% (blue), 10.55% (red) and finals 10.38% (orange). From these results, we can conclude that a higher LR value is a key factor in improving the overall system accuracy (with the use of the SWA algorithm). The result of the orange curve could be further improved by the steep use of the cosine annealing to reduce the loss caused by the steep jump to the final constant LR value.

Table 3.5: Training configuration used in the fifth experiment.

	SWA 1 (green)	SWA 2 (blue)	SWA 3 (gray)
Optimizer	SWA	SWA	SWA
Number of iterations	28	28	28
Initial LR	$1e^{-6}$	$1e^{-6}$	$1e^{-6}$
Final LR	$1e^{-6}$	$1e^{-6}$	0.02
LR course	cosine	cosine	cosine
LR cycle	2	1	0.5
LR cycle amplitude	0.01	0.01	0.01

3.5 Experiment 5

This experiment represents a completely different approach to the course of LR value during SWA training, namely the cyclic course of LR value. The cosine function is used with the shift equal to π on the x-axis (starting at the minimum) and on the y-axis by the amplitude of the function and the LR value from the last iteration (continuously connected to the previous course). The formula for using cyclic Effective LR is as follows

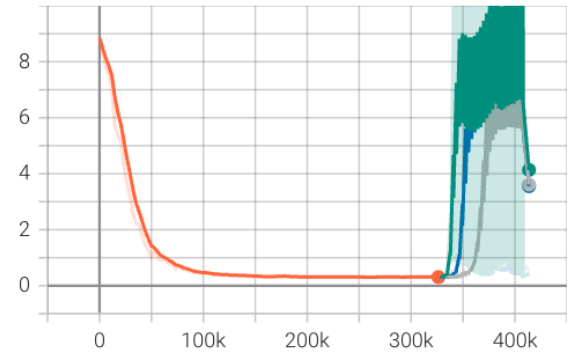
$$lr = \alpha \cos(i + \pi) + (\alpha + \delta), \tag{3.1}$$

where α refers to the function amplitude, i to the value from premade list[1] according to the number of cosine cycles and SWA iteration number. δ refers to the last LR from previous (standard) training. Training specifications can be seen in Table 3.5. The number of additional iterations stays the same. For a better comparison between similar experiments, three of them are described in this one. They differ only in the coefficient determining the cycle frequency. Following Figure 3.6d, we can see the difference over the LR value, where the blue curve performs 1 full cycle, the grey curve only half cycle and the green curve 2 cycles. This results in a different LR value at the end of the training. The green and blue one end, where they started and the grey ends at the maximum cycle value.

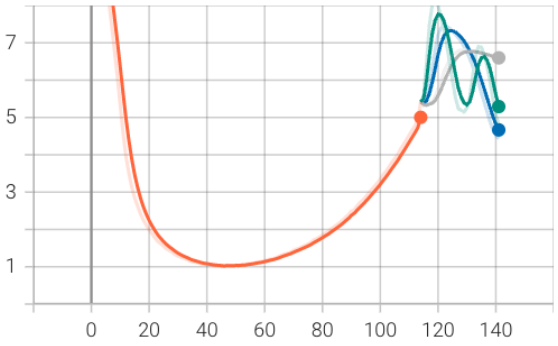
3.5.1 Result

The results are more than surprising when all three of these experiments achieved better EER levels than Baseline. Based on the order from the Table 3.5, the EER of each experiment is as follows: 10.54% (green), 10.61% (blue) and 10.68% (gray). From this, we can conclude that we managed to find a training configuration that can achieve an overall enhancement of the resulting system. This success can be followed by several other experiments that could benefit even more from the SWA algorithm. Unfortunately, the time-consuming nature of the individual experiments does not allow me to build on this success.

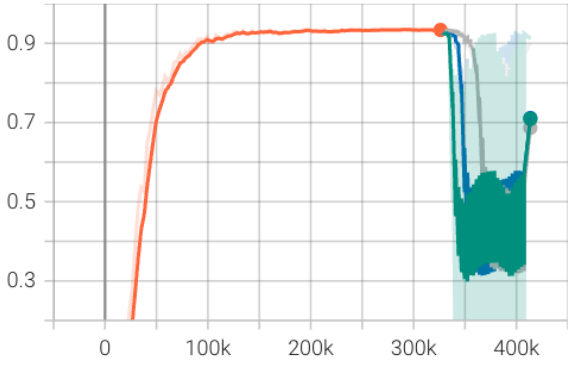
[1] Created by Python list comprehension (2 refers to the one full cycle which is 2π):
[(2 * num_cycles / swa_iters) * math.pi * i for i in range(0, swa_iters + 1)]



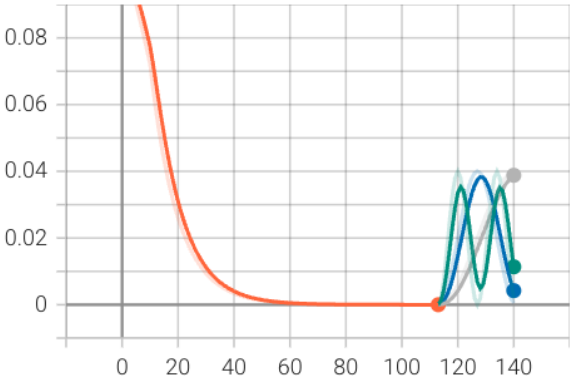
(a) CV Loss



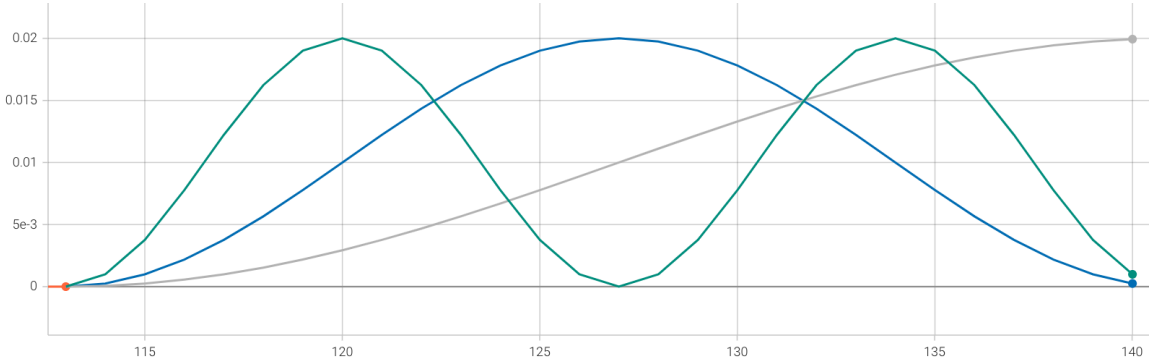
(b) Training Loss



(c) CV Accuracy

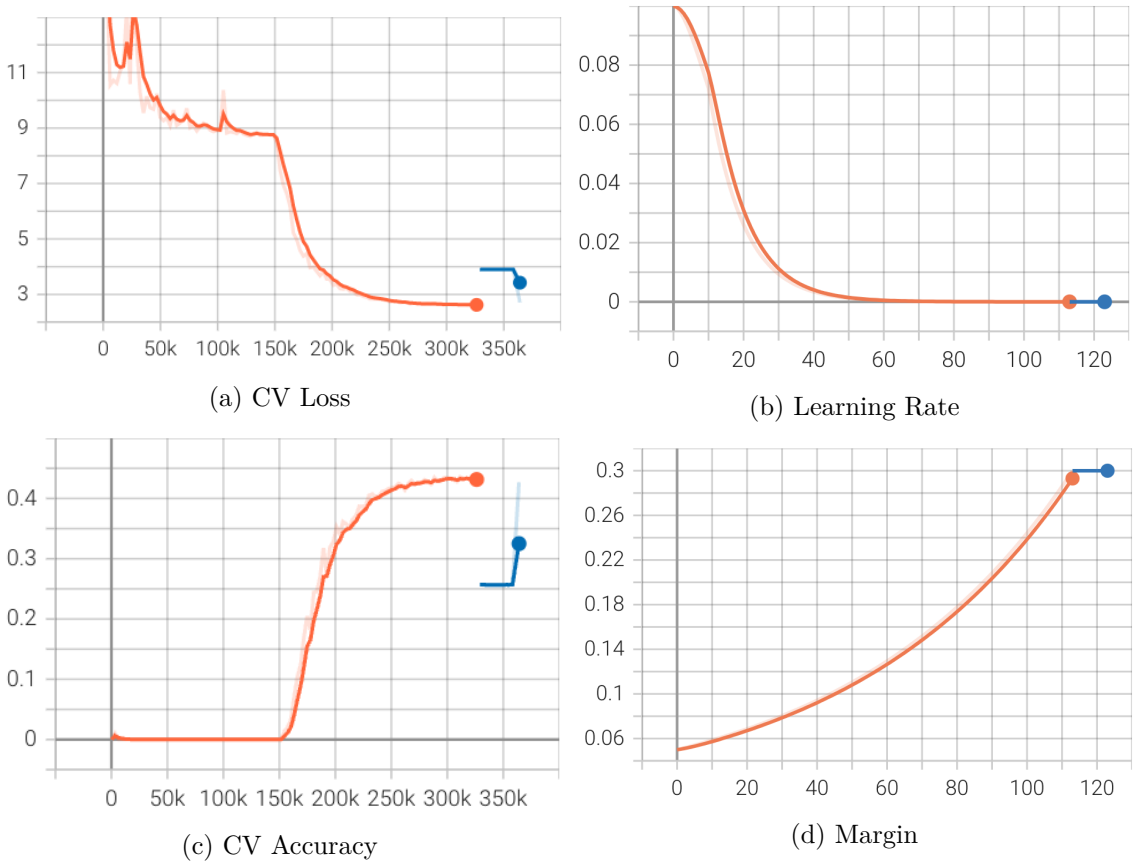


(d) Learning Rate



(e) Effective Learning Rate detail

Figure 3.6: Experiment number 5: Figures 3.6a and 3.6c show a sharp deterioration in system performance with consequent rapid oscillations. Thanks to the smoothing of the edges, we can see at least the approximate behavior of the Loss and Accuracy values, which eventually reach the level of the original orange curve (again, smoothed but pale variants indicate the final value). This course indicates a promising result in terms of the individual systems overall performance. Figure 3.6b shows the tendency of the Loss value to decrease when the LR value decreases and vice versa. Figure 3.6e shows detail of the true LR . Similarly as in the previous experiment in Figure 3.5, except the fact it displays course throughout the SWA training.



(a) CV Loss

(b) Learning Rate

(c) CV Accuracy

(d) Margin

Figure 3.7: Experiment number 6: we can see the decrease of the Loss value on the CV data in two phases. The first phase is related to the Figure 3.7c, when the accuracy on the CV data does not increase, but the Loss value decreases abruptly up to the level of 150k data samples, where Loss and Accuracy begin to behave inversely. The Figures 3.7b and 3.7d displays identical value courses as in the first experiment because they refer to the same training configuration. As for the SWA part of the training, Loss and Accuracy—for reasons unknown to me—do not follow the values of the standard training process. Although their value may appear to be constant, however, it very gently decreases and rises, respectively. But after the BN update at the end of training, the Loss and Accuracy made both a jump towards the better solution, which caused little progress in final performance.

3.6 Experiment 6

In this experiment, the same network structure (ResNet34) as in the first one is used, but the optimization algorithm changed. For detailed configuration see Table 3.6.

This experiment serves only as an example of different training optimization method with identical specifications. Training accuracy decreased significantly with the substitution of mini-batch SGD for the Adam algorithm. The basic Adam training with the exact training specifications as in the experiment from Section 3.1 achieved EER equal to 22.98%.

Table 3.6: Training configuration used in the sixth experiment with involded opitmizer Adam.

	Baseline	SWA model
Optimizer	Adam	Adam + SWA
Number of iterations	113	11
Initial LR	0.1	$1e^{-6}$
Final LR	$1e^{-6}$	$1e^{-6}$

3.6.1 Result

The last experiment showed that using the same training configuration as in the first experiment, but with a different optimization algorithm is not the right way to achieve the better or even similar performance. After involving the SWA optimization algorithm with constant LR equal to $1e^{-6}$ and 10% additional iterations, slightly better accuracy was achieved, EER 22.96%. Unlike the first experiment, the added iterations led to a slight improvement in the overall system performance. Although more iterations and subsequent changes in the approach to LR would lead to a more significant improvement, optimization of the Adam algorithm is not the aim of this work. Therefore, no other experiments are performed using the Adam optimizer.

Table 3.7: Summary of individual experiments results. The results are sorted by EER value in ascending order (lowest first). Note that LR course equal to cosine annealing means, before the LR value is set to constant, cosine annealing is applied to make continuous transition. Also $\lim_{\to\infty} 0$ does not explian exact course, it omits cosine course when continuously dropping from value 0.1 to the point where it starts to behave according to lim prescription. For displaying the baselines EERs the italic font is used.

Configuration	Optimizer	Iterations	LR course	EER (%)
Exp. no. 4.3	SGD + SWA	28	vertical step 0.01	**10.38**
Exp. no. 5.1	SGD + SWA	28	cosine (2 cycles)	10.54
Exp. no. 4.2	SGD + SWA	28	cosine annealing 0.007	10.55
Exp. no. 5.2	SGD + SWA	28	cosine (1 cycle)	10.61
Exp. no. 5.3	SGD + SWA	28	cosine (0.5 cycle)	10.68
Baseline	SGD	113	$\lim_{\to\infty} 0$	*11.17*
Exp. no. 4.1	SGD + SWA	28	sloping step 0.005	11.21
Exp. no. 2	SGD + SWA	28	const $1e^{-6}$	11.26
Exp. no. 1	SGD + SWA	11	const $1e^{-6}$	11.27
Exp. no. 3	SGD + SWA	28	const 0.01	11.54
Exp. no. 6	Adam + SWA	11	const $1e^{-6}$	22.96
Baseline	Adam	113	$\lim_{\to\infty} 0$	*22.98*

3.7 Results Summary

Let us now examine the achieved results and summarize the course of the experiments. During previous experiments, where different approaches were used to determine LR value during SWA training, as well as different lengths of the training process, we did not achieve significant success. Although some approaches, especially in experiments from Section 3.4 and 3.5, achieved promising performance on training data, this performance did not reflect a large extent on the system's EER. The improvement achieved is satisfactory in terms of the number of experiments and revealed new knowledge for future improvement. The comparison of the individual results is summarized in Table 3.7. The most useful knowledge gained to improve the performance of the system using the SWA algorithm is, in particular, a sufficiently large LR value in combination with a gradual transition. The second also a promising option is to use a cyclic course of the LR value with sufficient amplitude. As for the length of the training process using the SWA algorithm, the approach of 25% additional iterations proved to be more effective than only 10%.

3.8 Implementation Details

After describing the course of individual experiments, it is desirable to state the implementation details used in this work. This section is a brief guide to used tools and technologies. Additionally, the section provides a closer look into the most critical scripts that enable system manipulation and the helper automation scripts.

The whole system is compounded of multiple different components based on different technologies. The recognition model itself is implemented in *Python* with the help of *Pytorch* library. *Bash* scripts were used for manipulation with the model. For illustration,

manipulation can be understood as executing the model training or the x-vector extraction process. Many Bash scripts make the workflow more automated, thus much more comfortable. Here are listed some of them:

- **train.ResNet34Themos.fmargin03.v01.warmup10.2gpus.32x4.1ep.sh** - sets environment variables required for successful execution of the training script. It also contains a command along with the parameters for configuration needed by the training script itself. The name determines the model used for training and other specifications (metric type, number of GPUs used for training, ...). Therefore the name change is dynamic. For each different training configuration, a new script is needed. According to the name, a new folder is created where the whole training (except the processed utterances, they are all stored in one place so they can be shared between researchers) data is stored and later also the x-vectors with the evaluation results.
- **extract.gpu.paja_collection.test.sh**, **extract.gpu.paja_collection.train.sre_04_12_cuts_with_aug.sh** - for training and evaulation (enroll and test together) backend set generates x-vector extraction commands for execution on GPUs, which is managed by the SGE on FIT.
- **cat_scps.all.sh** - for each backend set (training, enrol, test) backend set loops through the all extracted x-vectors and concatenates them into the one long one, to make manipulation easier. Then the file index is used to reference a specific x-vector.
- **run_plda.train_sre_04_12_cuts_with_aug.eval_pajaset01.lda200.sh** - run the backend training process needed for the following final system evaluation. In the first step, he first trains the model on a training set. The second step evaluates the overall performance of the system.
- **task_to_fix.py** - automates the extraction of the missing (previously failed) x-vectors. Script (Python) checks successfully extracted x-vectors, creates files with commands for extracting missing ones and submits only the missing jobs to the SGE.

Despite workflow automation, there are still many processes that cannot be (not efficient) automated and therefore must be performed manually. These processes include, in particular, copying the scripts listed above into working folders to extract x-vectors and evaluate the system's accuracy and subsequent script handling. Furthermore, manual work is necessary while creating training scripts with a specific configuration. Working with SGE when reserving machines for training, checking the progress and subsequent release after training, etc.

In addition, the training statistics should be used to create graphs. For these purposes, the *tensorboard* is used. Throughout the training process the individual values are stored by "torch.utils.tensorboard.SummaryWriter" and from one level above the folder—where the desired data are saved—the command "tensorboard –bind_all –logdir <foldername-with-data>" can be executed. This command launches the server providing data visualization for all stored data from the training process. All of the used Figures in Chapter 3 are taken from tensorboard visualization.

3.9 Problems Encountered

As is often the case when implementing or launching someone else's project, several problems can make development uncomfortable. This work is no exception, where I encountered

many technical problems caused by the "new" environment, incomplete data and many others. This section summarizes the main development issues I have encountered and might have helped someone for future debugging.

The first part of the work was to replicate the baseline. With the help of my supervisor, I had to learn how to manipulate the model using the Sun Grid Engine (SGE) to submit the jobs etc. After receiving the already trained model and access to GPUs at the faculty, generating x-vectors from the evaluation set was necessary. The problem occurred when submitting the command to SGE. A different version of Python was installed in my main working folder ("/homes/eva/xs/xsovad06"). The solution was to unset "PYTHON_PATH" and "PYTHONHOME" in the ".bashrc" file and use Python installed in Ondrej Glembek's workspace.

The initial setup of the training script was not as challenging as the extraction experience, but the training baseline brought an insidious problem. After debugging the details of the training script, the training finally started. At first, for easier debugging, only one GPU was used. After smooth training for a few iterations, the process was aborted and launched on 4 GPUs. The training was interrupted every second iteration (about 35 minutes), and I had to start it again manually. The problem was in the limited time for the CPU on the machine. The solution was to add the command "ulimit -t unlimited" to the file ".bashrc" in the working folder.

A problem with incomplete or damaged training set data also occurred. This problem was solved by the DCGM staff. He had to generate the new data (process the VoxCeleb2 database), which took several weeks because of other technical issues. No model could be trained during this period.

There are some cases when the x-vector extraction may fail. The reasons for this failure are many. During this thesis work, several technical problems happened on some machines where I trained the DNN and extracted x-vectors. When this failure happens, the x-vectors extraction must have been executed again. I keep the list of machines that work correctly and update him when other machines fail. After this phenomenon occurred quite often, I created a script to automate the extraction of the missing x-vectors described in Section 3.8.

It is also worth mentioning the struggle with implementing the main training script. To implement the SWA algorithm into the existing state of the training script, I came across a solution that was not purely using the Pytorch library. The problem was using the Kaldi tool to provide data for training and not the Pytorch data loader. A case is a different approach to processing data throughout the training phase—custom handling tasks like training one iteration, validating model on CV data after each iteration, etc. Furthermore, a custom LR scheduler solution and not a Pythorch scheduler. The complexity and scope of the training script did not lead to a complete rewrite of my solution, as it came to its current state through long-term use and debugging. Instead, I incorporated the functionality of the SWA algorithm and debugged a given solution. This leads to multiple errors while trying to use Pytorch build-in function. For example, function "torch.optim.swa_utils.update_bn()" was not used after SWA training because of the different data handling while doing the forward pass. I had to use a custom implementation of the forward pass to update BN statistics at the end of the SWA training.

Chapter 4

Conclusion

Finally, the time has come to summarize this work's overall achievements and to look at possible future ways to upgrade the DNNs training process, thus the system's recognition performance.

Despite all the technical, time and other problems I encountered during my research, I finally achieved satisfactory results. Several follow-up experiments had to be carried out, during which the experience gained was always implemented in the following one. This chain of small improvements has led to overcoming the baseline performance in most cases. The difference between the baseline and experiments is not dramatic, but it is living proof that the tested SWA algorithm is beneficial and can be implemented in practice. Baseline with applied mini-batch SGD optimizer achieved EER equal to **11.17%**. In contrast, the best EER obtained in the experiment is equal to **10.38%**. The training configuration, whereas the LR course is **the vertical step** from value $1e^{-6}$(the last value of the previous training) **to constant 0.01** was used and the training last for additional **25%** iterations (corresponding to 28 iteration). Also, promising results came from applying the cyclic course of LR value. In addition, the future application of the acquired knowledge may ensure even greater success. Sufficiently large LR value during SWA training, or its cyclical variant, after averaging, will reach closer to the optimum of the Loss space.

4.1 Future Work

There is still the opportunity to perform many other experiments that could reveal an enhancement in the overall performance of the recognition system. In the first step, I would focus on different configurations using the SWA algorithm for the training process. There is an almost unlimited number of options for engaging the SWA algorithm and benefitting from its capabilities. For example, try different lengths of the training process (number of iterations), and different function courses that prescribe the LR value or the time (concrete iteration) when the SWA starts to be active. It would be worthwhile to try the SWA configuration according to the procedure in the introductory blog[1] from Pytorch. The next possible step is to experiment with the neural network structure. It is adding or removing layers, resizing individual layers, etc.

[1]Blog can be found on the official Pytorch webpage, see https://pytorch.org/blog/pytorch-1.6-now-includes-stochastic-weight-averaging/

Bibliography

[1] *National Institute of Standard and Technology*. Available at: http://www.nist.gov/speech/tests/spk/index.htm.

[2] Bai, Z. and Zhang, X.-L. *Speaker Recognition Based on Deep Learning: An Overview*. arXiv, 2020. DOI: 10.48550/ARXIV.2012.00931. Available at: https://arxiv.org/abs/2012.00931.

[3] Bai, Z., Zhang, X.-L. and Chen, J. Speaker Verification by Partial AUC Optimization With Mahalanobis Distance Metric Learning. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. 2020, vol. 28, p. 1533–1548. DOI: 10.1109/TASLP.2020.2990275.

[4] Cassidy, R. J. and Smith III, J. O. *Auditory Filter Bank Lab - Stanford University*. Available at: https://ccrma.stanford.edu/realsimple/aud_fb/aud_fb.pdf.

[5] Dehak, N., Kenny, P., Dehak, R., Dumouchel, P. and Ouellet, P. Front-End Factor Analysis For Speaker Verification. 2010, PP, no. 99, p. 1 –1. DOI: 10.1109/TASL.2010.2064307. ISSN 1558-7916.

[6] Dehak, N., Kenny, P. J., Dehak, R., Dumouchel, P. and Ouellet, P. Front-End Factor Analysis for Speaker Verification. *IEEE Transactions on Audio, Speech, and Language Processing*. 2011, vol. 19, no. 4, p. 788–798. DOI: 10.1109/TASL.2010.2064307.

[7] Deng, L., Li, J., Huang, J.-T., Yao, K., Yu, D. et al. Recent Advances in Deep Learning for Speech Research at Microsoft. In:. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), May 2013. Available at: https://www.microsoft.com/en-us/research/publication/recent-advances-in-deep-learning-for-speech-research-at-microsoft/.

[8] Duchi, J., Hazan, E. and Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* JMLR.org. jul 2011, vol. 12, null, p. 2121–2159. ISSN 1532-4435.

[9] Garcia Romero, D. *Analysis of i-vector length normalization in Gaussian-PLDA speaker recognition systems*. 2011. Submitted to ICSLP 2011.

[10] Garcia Romero, D. and Espy Wilson, C. Y. Analysis of i-vector length normalization in speaker recognition systems. In: *Proc. Interspeech 2011*. 2011, p. 249–252. DOI: 10.21437/Interspeech.2011-53.

[11] Goodfellow, I. J., Bengio, Y. and Courville, A. *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. 200-220 p. http://www.deeplearningbook.org.

[12] Graves, A. Generating Sequences With Recurrent Neural Networks. *ArXiv*. 2013, abs/1308.0850.

[13] Guo, H., Jin, J. and Liu, B. *Stochastic Weight Averaging Revisited*. 2022.

[14] He, K., Zhang, X., Ren, S. and Sun, J. *Deep Residual Learning for Image Recognition*. 2015.

[15] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*. 2012, abs/1207.0580. cite arxiv:1207.0580. Available at: http://arxiv.org/abs/1207.0580.

[16] Ioffe, S. and Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: Bach, F. and Blei, D., ed. *Proceedings of the 32nd International Conference on Machine Learning*. Lille, France: PMLR, 07–09 Jul 2015, vol. 37, p. 448–456. Proceedings of Machine Learning Research. Available at: https://proceedings.mlr.press/v37/ioffe15.html.

[17] Kenny, P. Bayesian speaker verification with Heavy–Tailed Priors. In: *Proc. of Odyssey 2010*. Brno, Czech Republic: [b.n.], June 2010. Http://www.crim.ca/perso/patrick.kenny, keynote presentation.

[18] Kingma, D. P. and Ba, J. *Adam: A Method for Stochastic Optimization*. 2017.

[19] Krizhevsky, A., Sutskever, I. and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C. J. C., Bottou, L. and Weinberger, K. Q., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012, vol. 25. Available at: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[20] Martin, A. and Greenberg, C. *Human Assisted Speaker Recognition in NIST 2010 Speaker Recognition Evaluation*. Speech and Language Processing Technical Committee Newsletter, 2010-06-30 2010. Available at: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=907326.

[21] Nagrani, A., Chung, J. S. and Zisserman, A. VoxCeleb: A Large-Scale Speaker Identification Dataset. In: *Interspeech 2017*. ISCA, Aug 2017. DOI: 10.21437/interspeech.2017-950. Available at: https://doi.org/10.21437%2Finterspeech.2017-950.

[22] Nair, V. and Hinton, G. E. Rectified Linear Units Improve Restricted Boltzmann Machines. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. Madison, WI, USA: Omnipress, 2010, p. 807–814. ICML'10. ISBN 9781605589077.

[23] Ondřej, G. *Optimization of gaussian mixture subspace models and related scoring algorithms in speaker verification*. 2012. PhD dissertation. Brno University of Technology.

[24] Polyak, B. T. and Juditsky, A. B. Acceleration of Stochastic Approximation by Averaging. *SIAM Journal on Control and Optimization*. 1992, vol. 30, no. 4, p. 838–855. DOI: 10.1137/0330046. Available at: https://doi.org/10.1137/0330046.

[25] Prince, S. J. and Elder, J. H. Probabilistic Linear Discriminant Analysis for Inferences About Identity. In: *2007 IEEE 11th International Conference on Computer Vision*. 2007, p. 1–8. DOI: 10.1109/ICCV.2007.4409052.

[26] Przybocki, M., Martin, A. and Le, A. NIST Speaker Recognition Evaluations Utilizing the Mixer Corpora 2004, 2005, 2006. 2007-09-01 2007.

[27] Psutka, J., Müller, L., Matoušek, J. and Radová, V. *Mluvíme s počítačem česky*. Academia, january 2006. 489–526 p. ISBN 80-2001-309-1.

[28] Qian, N. On the momentum term in gradient descent learning algorithms. *Neural Networks*. 1999, vol. 12, no. 1, p. 145–151. DOI: https://doi.org/10.1016/S0893-6080(98)00116-6. ISSN 0893-6080. Available at: https://www.sciencedirect.com/science/article/pii/S0893608098001166.

[29] Raschka, S. and Mirjalili, V. *Python machine learning : machine learning and deep learning with python, scikit-learn, and tensorflow 2*. Packt Publishing, Limited, 2019. 37-38 p. Available at: https://www.worldcat.org/oclc/1135663723.

[30] Robbins, H. E. A Stochastic Approximation Method. *Annals of Mathematical Statistics*. 2007, vol. 22, p. 400–407.

[31] Sadjadi, O., Greenberg, C., Singer, E., Mason, L. and Reynolds, D. *NIST 2021 Speaker Recognition Evaluation Plan*. NIST SRE, 2021-07-12 04:07:00 2021. Available at: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=932697.

[32] Sadjadi, O., Greenberg, C., Singer, E., Reynolds, D., Mason, L. et al. The 2018 NIST Speaker Recognition Evaluation. In:. INTERSPEECH, Graz, AT, 2019-09-15 00:09:00 2019. Available at: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=927673.

[33] Sarangi, S., Sahidullah, M. and Saha, G. Optimization of data-driven filterbank for automatic speaker verification. *Digital Signal Processing*. 2020, vol. 104, p. 102795. DOI: https://doi.org/10.1016/j.dsp.2020.102795. ISSN 1051-2004. Available at: https://www.sciencedirect.com/science/article/pii/S1051200420301408.

[34] Seyed, Greenberg, C., Singer, E., Olson, D. and Mason, L. *NIST 2020 CTS Speaker Recognition Challenge Evaluation Plan*. NIST 2020 CTS Speaker Recognition Challenge, 2020-07-29 2020.

[35] Seyed, Greenberg, C., Singer, E., Reynolds, D., Mason, L. et al. The 2019 NIST Speaker Recognition Evaluation CTS Challenge. In:. The Speaker and Language Recognition Workshop: Odyssey 2020, Tokyo, -1, 2020-05-18 2020. Available at: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=929506.

[36] Seyed, Kheyrkhah, T., Tong, A., Greenberg, C., Olson, D. et al. The 2016 NIST Speaker Recognition Evaluation. In:. Interspeech 2017, Stockholm, -1,

2017-08-20 2017. Available at:
https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=922849.

[37] Snyder, D., Garcia Romero, D., Sell, G., Povey, D. and Khudanpur, S. X-Vectors: Robust DNN Embeddings for Speaker Recognition. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, p. 5329–5333. DOI: 10.1109/ICASSP.2018.8461375.

[38] Tieleman, T., Hinton, G. et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*. 2012, vol. 4, no. 2, p. 26–31.

[39] Variani, E., Lei, X., McDermott, E., Moreno, I. L. and Gonzalez Dominguez, J. Deep neural networks for small footprint text-dependent speaker verification. In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2014, p. 4052–4056. DOI: 10.1109/ICASSP.2014.6854363.

[40] Wan, V. and Campbell, W. Support vector machines for speaker verification and identification. In:. February 2000, vol. 2, p. 775 – 784 vol.2. DOI: 10.1109/NNSP.2000.890157. ISBN 0-7803-6278-0.

[41] Yang, G., Zhang, T., Kirichenko, P., Bai, J., Wilson, A. G. et al. SWALP : Stochastic Weight Averaging in Low Precision Training. PMLR. 09–15 Jun 2019, vol. 97, p. 7015–7024. Proceedings of Machine Learning Research. Available at: https://proceedings.mlr.press/v97/yang19d.html.

[42] Ye, F. and Yang, J. A Deep Neural Network Model for Speaker Identification. *Applied Sciences*. 2021, vol. 11, no. 8. DOI: 10.3390/app11083603. ISSN 2076-3417. Available at: https://www.mdpi.com/2076-3417/11/8/3603.

[43] Zeinali, H., Wang, S., Silnova, A., Matějka, P. and Plchot, O. *BUT System Description to VoxCeleb Speaker Recognition Challenge 2019*. arXiv, 2019. DOI: 10.48550/ARXIV.1910.12592. Available at: https://arxiv.org/abs/1910.12592.

Appendix A

Structure of the enclosed CD

This chapter brings more information about the structure and content of the enclosed CD. In addition, it clarifies the purpose and possible use of the attached information.

The very nature of this work declares the content and possible use, or replication of the results. Since this is a research activity, not a functional product that is easily replicable, the content is a source code folder. Due to the limited memory possibilities of this carrier, individual functional models are omitted. However, they are available on the school server ("/pub/users/xsovad06/BP/"). Here you can find all the trained models during the individual experiments, the course of the training processes and all the manipulation scripts, some of which are listed in the Section 3.8. A complete description of the structure and contents of the included CD can be found in the root folder in the **README.md** file. If necessary, replicate some experiments or some manipulations to obtain information. You can contact my supervisor or me personally.

Appendix B

ResNet Structure

In this section, you can find the complete ResNet structure in Figure B.1.

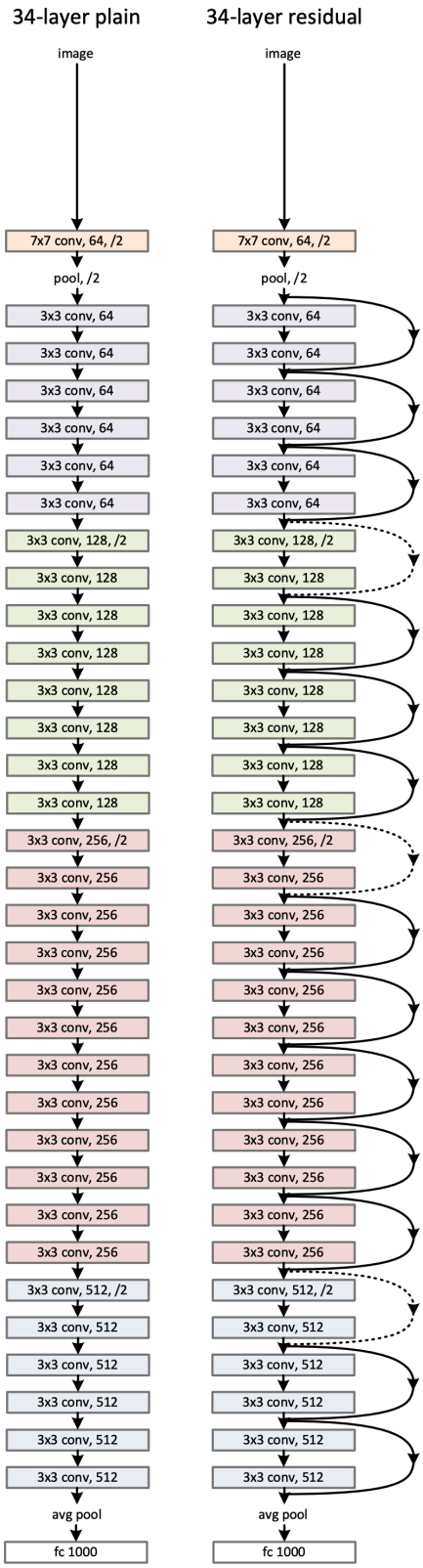


Figure B.1: Complete structure of ResNet (34-layers) used in this work compared to plain DNN without residual blocks. Figure taken from original paper [14].