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

SPEAKER VERIFICATION WITHOUT FEATURE EXTRACTION

VERIFIKACE OSOB PODLE HLASU BEZ EXTRAKCE PŘÍZNAKŮ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. PETER LUKÁČ**
AUTOR PRÁCE

SUPERVISOR **Ing. LADISLAV MOŠNER**
VEDOUCÍ PRÁCE

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2020/2021

Master's Thesis Specification

Student:	**Lukáč Peter, Bc.**
Programme:	Information Technology
Field of study:	Sound, Speech and Natural Language Processing
Title:	**Speaker Verification without Feature Extraction**
Category:	Speech and Natural Language Processing

Assignment:

1. Get acquainted with the problem of speaker verification and with state-of-the-art systems for verification with and without feature extraction.
2. Achieve results obtainable with the RawNet model using existing implementation.
3. Experiment with the RawNet model to gain insight into the effects of the system modules on its behavior.
4. Propose and implement extensions of the model.
5. Compare and discuss results obtained with the modified model, the original RawNet model, and state-of-the-art models using extracted features.

Recommended literature:

- JUNG, Jee-weon, Hee-Soo HEO, Ju-ho KIM, Hye-jin SHIM a Ha-Jin YU. RawNet: Advanced End-to-End Deep Neural Network Using Raw Waveforms for Text-Independent Speaker Verification. In: *Interspeech 2019,* 1268-1272,

Requirements for the semestral defence:

- Items 1 through 3

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Mošner Ladislav, Ing.**
Head of Department:	Černocký Jan, doc. Dr. Ing.
Beginning of work:	November 1, 2020
Submission deadline:	May 19, 2021
Approval date:	October 30, 2020

Abstract

Speaker verification is a field that is still improving its state of the art (SotA) and tries to meet the demands of its use in speaker authentication systems, forensic applications, etc. The improvements are made by the advancements in deep learning, the creation of new training and testing datasets and various speaker recognition challenges and speech workshops. In this thesis, we will explore models for speaker verification without feature extraction. Inputting the models with raw speaker waveform simplifies the pipeline of the systems, thus saving computational and memory resources and reducing the number of hyperparameters needed for creating the features from waveforms that affect the results. Currently, the models without feature extraction do not achieve the performance of the models with feature extraction. By applying various techniques to the models we will try to improve the baseline performance of the current models without feature extraction. The experiments with SotA techniques improved the performance of a model without feature extraction considerably however we still did not achieve the performance of a SotA model with feature extraction. However, the improvement is considerable enough so that we can use the improved model in a fusion with feature extraction model. We also discussed the experimental results and proposed improvements that aim to solve discovered limitations.

Abstrakt

Verifikácia osôb je oblasť, ktorá sa stále modernizuje, zlepšuje a snaží sa vyhovieť požiadavkám, ktoré sa na ňu kladú vo oblastiach využitia ako sú autorizačné systmémy, forenzné analýzy, atď. Vylepšenia sa uskutočňujú vďaka pokrom v hlbokom učení, tvorením nových trénovacích a testovacích dátovych sad a rôznych súťaží vo verifikácií osôb a workshopov. V tejto práci preskúmame modely pre verifikáciu osôb bez extrakcie príznakov. Používanie nespracovaných zvukových stôp ako vstupy modelov zjednodušuje spracovávanie vstpu a teda znižujú sa výpočetné a pamäťové požiadavky a redukuje sa počet hyperparametrov potrebných pre tvorbu príznakov z nahrávok, ktoré ovplivňujú výsledky. Momentálne modely bez extrakcie príznakov nedosahujú výsledky modelov s extrakciou príznakov. Na základných modeloch budeme experimentovať s modernými technikamy a budeme sa snažiť zlepšiť presnosť modelov. Experimenty s modernými technikamy značne zlepšili výsledky základných modelov ale stále sme nedosiahli výsledky vylepšeného modelu s extrakciou príznakov. Zlepšenie je ale dostatočné nato aby sme vytovrili fúziu so s týmto modelom. Záverom diskutujeme dosiahnuté výsledky a navrhujeme zlepšenia na základe týchto výsledkov.

Keywords

speaker verification, featureless extraction, speaker embedding, residual network, RawNet, VoxCeleb1, VoxCeleb2, VoxSRC, Feature Map Scaling, SincNet, Additive Angular Margin loss, fusion

Klíčová slova

verifikácia osôb, bez extrakcie príznakov, obtisk rečníka, residuálne siete, RawNet, VoxCeleb1, VoxCeleb2, VoxSRC, škálovenie máp príznakov, SincNet, Aditívna Uhlová Okrajová funkcia, fúzia

Reference

LUKÁČ, Peter. *Speaker Verification without Feature Extraction*. Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Ladislav Mošner

Rozšířený abstrakt

Verifikácia osôb podľa hlasu bez extrakcie príznakov je smer pre verifikáciu osôb, kde trénujeme a vyhodnocujeme (a aj používame v prípade reálneho nasadenia) modely priamo na časovom priebehu nahrávok rečníkov. Toto robíme za cieľom zjednodušenia spracovávania vstupov. Extrakcia príznakov vyšuje pamäťové a výpočetné nároky, je potrebné zvoliť vhodný typ príznakov z veľkého množstva metod a vyžaduje hyperparametre, ktoré ovplivňujú výsledok extrakcie a zároveň výsledok verfikácie. Na základe naštudovaných zdrojov a vlastných experimentov, v sučastnosti modely pre verifikáciu osôb bez extrakcie príznakov nedosahujú výsledky modelov s extrakciou príznakov. Cieľom tejto práce bolo experimentovať s modernými technikamy pre neurónové siete a zlepšiť výsledky modelu bez extrakcie príznakov. Nakoniec sme vytvorili fúziu so modelom s extrakciou príznakov, aby sme zistili, či takto vylepšený model môže slúžiť vo fúzii. Ako základ pre naše experimenty sme zvolili model nazvaný RawNet, na ktorom sme experimentovaly s rôznymi technikamy.

The VoxCeleb Speaker Recognition Challenge 2020 (VoxSRC-2020) poslúžilo ako prehľad systémov pre verifikáciu osôb. Na zaklade VoxSRC-2020 sme zistili, že súčastnym trendom sú modely založené na residuálných konvolučných neurónových sieťach. Použili sme rovnakú trénovaciu, validačnú a testovanciu sadu ako vo VoxSRC-2020, tj. VoxCeleb1 (validovanie) a VoxCeleb2 (trénovanie). Vedúci práce nám poskytol výsledky modelu pre verifikáciu osôb s extrakciou príznakov, ktorý bol založený na ResNet34 architektúre a trénovaný na VoxCeleb2. Ako príznaky pre tento model boli použité logaritmické mel-frekvenčné bankové energie (fbank) príznaky. Zároveň je využitá Additive Margin (AM) loss funkcia. Tento model dosiahol 1.46% EER. S týmto modelom sme porovnali náš vylepšený model a vytovrili fúziu s ním.

Ako základ naších experimentov sme použili model RawNet, ktorý je založený na schéme pre extrakciu embedingov. V prevej sérii experimentov sme porovnávali niekoľko loss funkcií: Sofmtax loss, Center loss, Speaker Basis loss, Hard Negative Mining loss a Additive Angular Margin (AAM) loss. Z týchto spomenutých loss funkcií najlepšie fungovala AAM loss. Nasledovali experimenty s vylepšenou architektúrou sieťe. Nová architektúra spočívala v prezoskupení konvolučných vrstiev (full pre-activation), batch normalizácie a pooling vrstiev. Toto prinieslo relatívne zlepšenie 8.4%. Augmentácia v dobe testu (TTA) priniesla relatívne zlepšenie 10.9%. Nasledovali porovnania experimentov so Squeeze-and-excitation blokmi a im podobnými metódami ako sú Feature Map Scaling (FMS) a α-Feature Map Scaling (α-FMS). Z týchto metód najviac uspeli FMS a α-FMS. Experimenty so SincNet vrstvou a Layer Normalization vrstvou nedosiahli dobré výsledky, navyše sa 4-krát zvýšil trénovaci čas.

Vo všetkých experimentoch (okrem prvých experimentov s loss funkciami) sme zároveň porovnávali separátne efekty predspracovania dát filtrom s hornou prepusťou a normalizácie dát do rozsahu -1 až 1. V niektorých prípadoch dopadla normalizácia dát lepšie (použitie FMS a α-FMS so Softmax loss), ale v konečom modeli sme použili normalizáciu dát. Konečný model pozostával z: full pre-activation residuálnymi blokmi, α-FMS, TTA evaluáciou, AAM loss a predspracovaním dát horným filtrom. Tento model dosiahol 2.48% EER, čo je relatívne 33% zlepšenie oproti 3.7% EER na začiatku. Tento model bez extrakcie príznakov stále nedosahuje výsledky modelu s extrakciou príznakov. Vytovrili sme fúziu. Najlepšie výsledky dosiahla fúzia v modelom, ktorý využíval FMS namiesto α-FMS (tento model dosiahol 2.5% EER). Táto fúzia dosiahla výsledok 1.45% EER.

V tejto práci sme dokázali vylepšiť baseline model bez extrakcie príznakov do stavu keď ho je možné použiť vo fúzii s modelom s extrakciou príznakov. Na základe experimentov sme vyvodili závery, kde je možné zaviesť ďalšie vylepšenia. Na základe zlepšenia výsledkov

pomocou TTA, rekurentné vrstvy (GRU) je možné nahradiť pomocou attentive statistical pooling. Augmentácia dát na VoxCeleb2 zväčšuje množstvo dát čo zlepšuje výsledky ako sme mohli pozorovať pri VoxSRC-2020. Sú dostupné novšie architektúri ako napr. Res2Net, ktoré demonštrovali dobré výsledky vo VoxSRC-2020. Zároveň by bolo možné použiť nové loss funkcie vyvinuté špeciálne pre verfikáciu osôb ako Generalized End-to-End loss pre verfikáciu osôb.

Speaker Verification without Feature Extraction

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Ing. Ladislav Mošner. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Peter Lukáč
May 16, 2021

Acknowledgements

I would like to thank my supervisor Ing. Ladislav Mošner, for his invaluable guidance and feedback during making this thesis. I would also like to thank the administrators (namely Ing. Tomáš Kašpárek) of SGE cluster at BUT FIT, for providing me with resources that made this thesis possible.

Contents

Chapter 1

Introduction

Current state of the art (SotA) approach to speaker verification is based on deep neural networks (DNNs) that process features extracted from the waveforms. We can get an overview of these systems in various speaker recognition challenges such as VoxCeleb Speaker Recognition Challenge 2020 (VoxSRC) [24]. The challenge contained multiple speaker recognition tasks: Speaker Verification - Closed (which means only provided dataset could be used for training), Speaker Verification-Open (which means data augmentation and training datasets other than provided datasets were allowed), Speaker Verification - Self-supervised (which means only unlabeled training data were allowed) and Speaker diarisation. In this thesis, we will only focus on closed supervised speaker verification. We see that the winners of the verification part of the challenge usually used models that are variations of the residual convolutional networks such as ResNet-34 architecture [8]. The extracted features that these models are trained on are usually Mel-frequency bank energies (fbank) features that typically have dimension in the range of 40-80 or Mel Frequency Cepstral Coefficients (MFCCs).

The hand-crafted features such as fbank features or MFCCs are affected by the hyperparameters such as window length, overlap or the final dimension of the features. That introduces hyperparameters that need to be tuned in the training to get optimal results. It also adds an extra step in the processing that consumes computing power and increases memory requirements. More importantly from the perspective of this thesis, training models that take waveform of a raw speech signal as the input should better fit the original dataset and learn deep features that would not be discovered by the models with feature extraction.

Some of the systems submitted to VoxSRC were also fused systems. In the system fusion, we take two or more different systems and combine their results in evaluation in such way that the final performance will outperform the individual systems. For this to happen, the systems need to have complementary results in evaluation. Systems without feature extraction are great candidates for fusion with systems with feature extraction because they learn deep features in a different way, therefore they could provide complementary discrimination capability for the fusion system.

At the present, models without feature extraction do not achieve the performance of their counterparts with feature extraction. In this thesis, we will explain a baseline model for speaker verification with feature extraction first (see Section 2.3). Then we will explain how a baseline model for speaker verification without feature extraction differs (see Section 2.4) and explain SotA DNN techniques that focus on improving the performance (see Sections 2.5, 2.6 and 2.7). Next, we will explain how models are trained and how we evaluate models

(see Chapter 3). Finally, we will perform series of experiments on models with different techniques and select the best technique to create a model with improved performance that we will compare to a baseline model with feature extraction and then create a fusion of improved model without feature extraction with the baseline model with feature extraction (see Chapter 4).

Chapter 2

Models for Speaker Verification

In this chapter, we will introduce the task of speaker verification (SV) and systems for the SV. First, we will analyze a DNN model for SV with feature extraction as this is currently the most widely researched approach. We will explain used techniques and how they differ from a DNN model without feature extraction. Then we will describe a baseline model for SV without feature extraction and techniques that will be applied to this model.

2.1 Speaker Verification

In SV, we verify if the given two utterances belong to the same speaker. More specifically we will focus on text-independent SV, meaning that in the pair of utterances the speakers say different content and the speakers are generally unknown.

In this thesis, we will use supervised training in which we use labeled data to optimize a model using an iterative optimization algorithm and an objective function or more objective functions at the same time. This is the most common way of training. It is also the most constrained way of training data-wise because we need a huge amount of labeled samples however, the large datasets are plentiful. Self-supervised does not require labeled data however, it is outperformed by the supervised training according to the results in the VoxSRC. The middle ground is semi-supervised training where only some data are labeled. The self-supervised and semi-supervised training are beyond the scope of this thesis.

In the evaluation, we will use Equal Error Rate (EER) as a metric to measure the system's discriminative performance. The evaluation will be performed on a dataset that has no overlap with the training and validation set. Pairs of the utterances from the evaluation set are described by the trials which also have specified ground truth if they are target trial or non-target trial.

2.2 Speaker Verification Systems

The SV system has two parts: front-end and back-end. The front-end is the DNN model. More specifically, in our case, we use DNNs in the speaker-embedding scheme (see Figure 2.1). In the context of DNNs, embeddings are sometimes referred to as x-vectors [27]. The inputs of the DNN are variable-length features of an utterance (in our case raw waveform) from a speaker. The output is fixed-length vector (embedding) that is a vector representation of the speaker. We can interpret the embedding as the speaker's voice fingerprint.

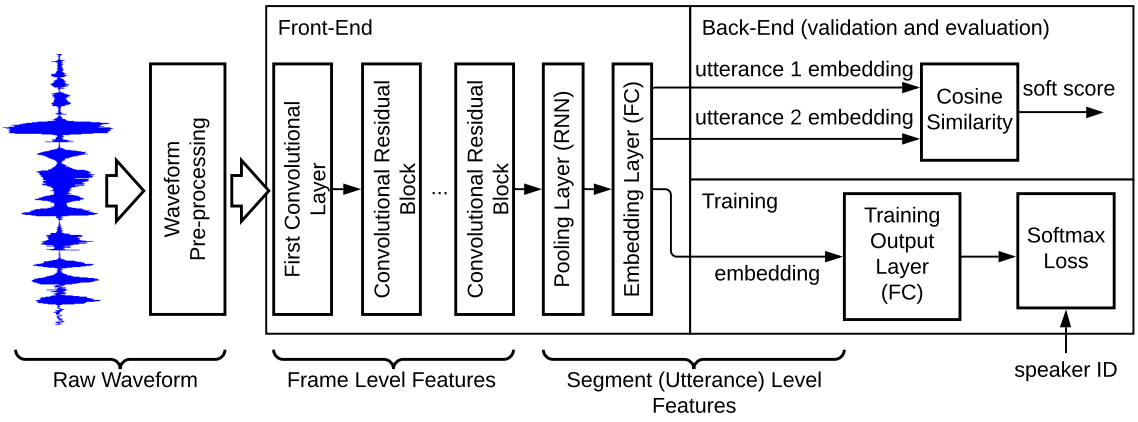


Figure 2.1: Speaker-embedding scheme shown on an example model for speaker verification with raw waveform input, data pre-processing and residual convolutional layers in the Front-End. The Back-End uses cosine similarity. Softmax loss is employed for forming the speaker embeddings during training. The bottom labels describe the meaning of features in various parts of the model.

We will perform class (speaker) identification training in order to train the front-end to be able to map the speaker's voice to embedding. With enough speakers in the training dataset, we achieve the capability of mapping a general speaker to the embedding. The embeddings of one class should be distributed with as little variance as possible and not overlap with other classes.

In the evaluation, we will use the front-end to map utterances to their embeddings. Then the back-end classifier will compare the embeddings of those utterances to score their similarity. Cosine-similarity is the simplest way to calculate the similarity of the embeddings. The similarities from all pairs of the evaluation trials and their ground truths will be used to calculate the performance metric of the system. In the SV scenario, when we need a hard decision, we would choose similarity threshold according to the scenario requirement.

By looking at the VoxSRC 2020 [24] we see that the front-end models for the embedding extraction with feature extraction are commonly based on ResNext, Res2Net or ResNet-34 architectures. The organizers also provided a baseline model to help new participants to get started that consisted of Fast ResNet-34. We will explain the functionality of the model for SV on ResNet architecture [8]. The ResNet architecture provides convolutional layers used for processing variable-length input. The following layers in the model such as pooling layer, embedding layer and output layer will be virtually the same in the models with and without feature extraction.

The main difference will be in the convolutional layers. Models with feature extraction use 2-dimensional convolutional layers that process 2-dimensional inputs (that may have multiple channels, see Subsection 2.3.1) and also have 2-dimensional outputs (that also might have multiple channels). This is ideal for 2-dimensional inputs such as fbank features or MFCCs. Models without feature extraction will use 1-dimensional convolutional layers that have 1-dimensional inputs and 1-dimensional outputs (also may have multiple channels) as this is ideal for processing raw waveforms.

The winners of the VoxSRC 2020 also applied numerous techniques to improve performance such as improvements to the convolutional layers, loss functions, all that can be applied to models without feature extraction as well.

2.3 Baseline Model for Speaker Verification with Feature Extraction

We will look at the ResNet [8] (there are different variations such as ResNet-34 (see Table 2.2 for the model for SV using ResNet-34) or ResNet50 but they only differ in the configuration of the hyperparameters) and how it can be used for speaker verification with some modification. The ResNet architecture was designed for processing highly correlated 2-dimensional inputs, therefore it is predominantly used in image processing. This is also suitable for processing features used in speech processing that have these properties such as fbank features.

ResNet consists of 2D convolutional layers and one pooling layer behind the first convolutional layer. The number of convolutional layers often reflects a specific codename (34 convolutional layers in ResNet-34).

In the following subsections, we discuss convolutional layers and residual blocks.

2.3.1 Convolutional Layers

A convolution layer [20] applies a convolution across the input feature maps and outputs new feature maps. In the context of the 2D convolution layers, the feature maps are 2D arrays that are arranged in one or several channels (see Figure 2.3). Convolution is a discrete operation that is local, linear and translation invariant. Because we apply the convolution locally, the output feature value depends only on the local input features. Using the convolutional layers we can extract features from the variable size input. The size of the output feature maps then depends on the size of the input feature maps. This is very useful for processing the variable-length utterances. The convolution being translation invariant means that we apply the same operation across every place of the input feature maps. The convolution will extract desired features from the input regardless of their position in the input feature maps. This is useful for extracting features from the sequences such as speaker utterances that contain relevant information in various places.

The convolution is performed using the convolution filters (kernels). In the case of the 2D convolution, the kernel is a 3D array (also can be imagined as 2D filters arranged in one or several channels) of scalar parameters (coefficients of the filter). The output of the kernel is the sum of the multiplications of the matching coefficients and values in the input feature maps. The output from every position is then written into output feature maps. The step of the kernel in every direction is given by the stride. The spacing between kernel elements may vary in every direction as well. This is usually called dilation. Padding is an optional hyperparameter that enables the kernel to reach over the borders of the feature maps to create output feature maps that are the same size as the input feature maps. The kernel size, stride, dilation and padding are set hyperparameters while kernel coefficients are trained parameters.

The feature maps usually get smaller after the convolutional layer is applied due to the kernel size, stride and dilation and the fact that the kernel usually can not reach outside the input feature maps. The output feature maps are of the same size as the input feature maps minus the size of the kernel (including the dilation) and divided by the step of the

Table 2.2: Model for SV with feature extraction based on ResNet-34 architecture. The first hyperparameter of conv2D is $n \times n$ which is the size of the kernel channels. The second hyperparameter is the number of kernels. Pool represents statistical pooling over time. Flatten layer flattens the 2-dimensional output of the pooling layer into the 1-dimensional tensor that can be feed-forwarded into a fully connected (FC) Embedding layer. Thick line after Embedding layer represents the embedding output that is used during evaluation. Output layer is used during training. The horizontal line in the residual blocks represents where the output of BN is added to the identity connection, following the original residual block architecture show in Figure 2.5.

Layer	Parameters
conv2D	3×3, 64 BN ReLU
Res Block	$\left\{\begin{array}{c}\text{conv2D } 3\times 3, 64\\ \text{BN}\\ \text{ReLU}\\ \text{conv2D } 3\times 3, 64\\ \text{BN}\\ \hline \text{ReLU}\end{array}\right\} \times 3$
Res Block	$\left\{\begin{array}{c}\text{conv2D } 3\times 3, 128\\ \text{BN}\\ \text{ReLU}\\ \text{conv2D } 3\times 3, 128\\ \text{BN}\\ \hline \text{ReLU}\end{array}\right\} \times 4$
Res Block	$\left\{\begin{array}{c}\text{conv2D } 3\times 3, 256\\ \text{BN}\\ \text{ReLU}\\ \text{conv2D } 3\times 3, 256\\ \text{BN}\\ \hline \text{ReLU}\end{array}\right\} \times 6$
Res Block	$\left\{\begin{array}{c}\text{conv2D } 3\times 3, 256\\ \text{BN}\\ \text{ReLU}\\ \text{conv2D } 3\times 3, 256\\ \text{BN}\\ \hline \text{ReLU}\end{array}\right\} \times 3$
pool	mean and std. pooling
	Flatten
Embedding	FC 256
Output	FC 5994

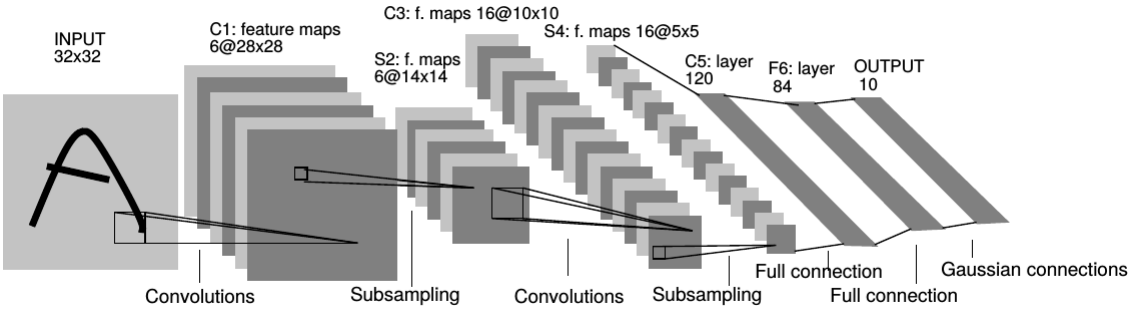


Table 2.3: Architecture of LeNet-5 [20] shows how convolutions are applied to the feature maps to obtain new feature maps.

filter. An exception is a configuration with a stride of one and added padding that is usually filled with zeros, thus allowing the kernel to reach borders and corners. The convolutional layers with zero padding are needed in residual blocks.

As mentioned, the feature maps are arranged in one or several channels (let us consider C channels). The kernel of the convolutional layer has one or several channels (let us consider C' kernel channels) that have the same size and stride. The kernel can process the input feature maps with any number of channels. At first, each kernel channel performs single-channel convolutions ($*$) on every feature map. Single-channel convolutions are then summed together to get a feature map of that kernel channel. The output of the convolutional layer are feature maps that were created by concatenating the feature maps of all kernel channels. The number of the channels in the output feature maps is then the number of kernel channels in that convolutional layer. If we consider input feature maps $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_C]$, output feature maps $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_{C'}]$, set of convolutional filters $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_{C'}]$ and bias $\mathbf{b} = [b_1, b_2, ..., b_{C'}]$, then each output feature maps channel is computed as:

$$\mathbf{u}_{c'} = \left(\sum_{c=1}^{C} \mathbf{v}_{c'} * \mathbf{x}_c \right) + b_{c'}. \tag{2.1}$$

In processing the features such as power fbank features, we stack multiple convolutional layers together. The first layers have fewer kernel channels than the deeper layers. In the convolutional network, we increase the number of channels in the deeper layers. The first layers will extract low-level features (because the kernels are usually small such as 2x2-7x7, they see small segments of the features) and deeper layers will be extracting high-level features (such as parts of phonemes or entire phonemes that are characteristic for certain speaker) that are useful for forming speaker embeddings.

2.3.2 Residual Blocks

Very deep convolutional neural networks (VDCNN) do not converge very well, having higher training error and evaluation error. This has been demonstrated on VDCNN having more than 20 convolutional layers [8]. The reason for that is not overfitting. The reason for that are vanishing gradients during training. The loss function gradient is found using backpropagation. Each layer gradient is multiplied following the chain rule. In architectures

with many layers, the final gradient is very small therefore the updates after every batch will be very small as well.

One of the solutions to that are residual connections [8] (hence the name ResNet). The residual blocks in Figure 2.4 consist of several convolutional layers. The residual connection (usually identity connection) takes the input of the block and adds it to the output of the last layer. If we add the identity to the gradient of the convolutional layers, the overall gradient of the block is then bigger and we can stack many residual blocks together without having a small final gradient.

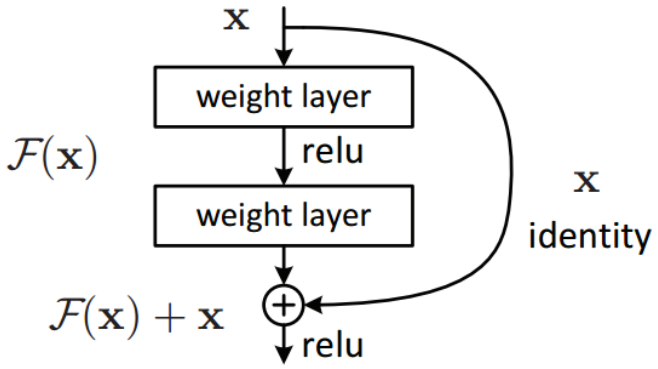


Table 2.4: A very simple residual block introduced in ResNet [8].

It has been demonstrated that the VDCNN is able to discover more features than the shallow counterparts and perform better [8]. The residual block shown in Figure 2.4 is the simplest implementation.

Residual Block Architectures

The convolutional layers are used with Batch Normalization (BN) layers that accelerate convergence and activation functions that provide non-linearity necessary for discriminative properties. Because BN normalizes values in feature maps it is best placed before the activation function which then leads to stable activation values [13]. There are several architectures of residual blocks (see Figure 2.5) with different orders of the two convolutional layers with BN and activation function (ReLU in our case).

The first three architectures [9] 2.5(a), (b) and (c) are the most straightforward use of BN and ReLU as they only add BN and ReLU after the convolutional layers. They differ in the placement of the addition of the identity connection. In residual networks, we avoid the problem of vanishing gradients by limiting the number of operations in the information propagation that diminish the gradient. The original architecture 2.5(a) uses ReLU after addition which could be placed elsewhere to limit the problem of vanishing gradient even more. The architecture with BN after addition 2.5(b) impedes the information propagation even more and architecture with ReLU before addition 2.5(c) negatively affects the representational ability because the ReLU output is only non-negative. As a result, both underperformed the original architecture [9]. The proposed full pre-activation architecture [9] 2.5(e) shifts the layers so that BN and ReLU are used before the convolutional layer. With convolutional layer before addition, the representational ability is increased and the

gradient is not diminished by ReLU after addition. The full pre-activation architecture outperformed the original architecture [9].

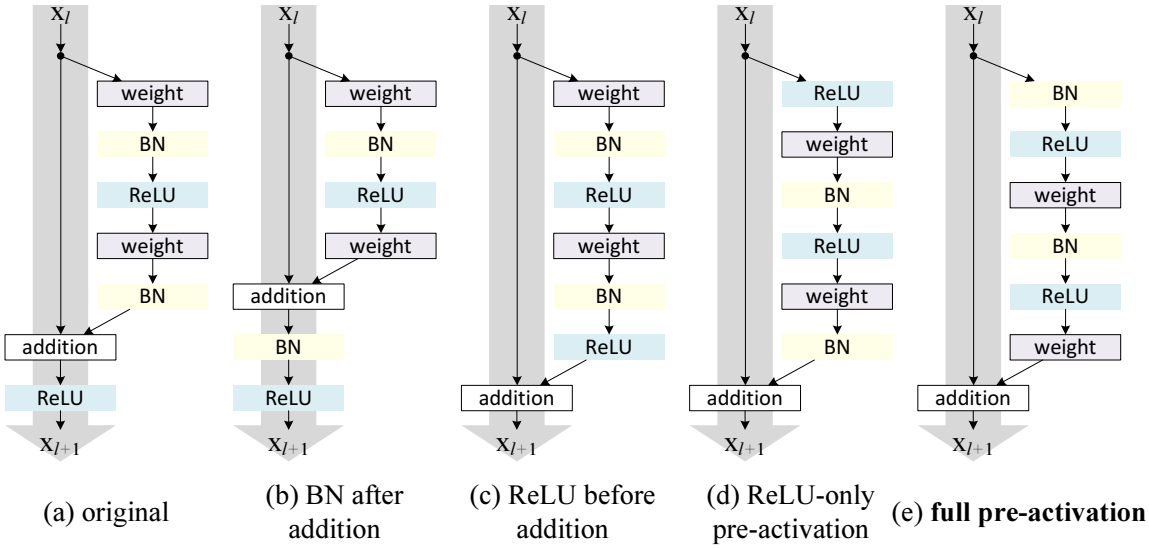


Table 2.5: Multiple residual block architectures [9] that use convolutional layers (weight) with batch normalization (BN) and activation functions (ReLu) only differ in order of the layers. Full pre-activation (e) is the best performing solution [9].

2.3.3 Statistics Pooling

The output of the convolutional layers are feature maps. In the case of 2D convolutional layers, they are 3D arrays as $\mathbb{R}^{H\times W\times C}$. The length W of the feature maps varies depending on the length of the input. Because the convolutional layers process the input and feature maps locally, they keep the important features for forming speaker embeddings in the same positions in the feature maps as they pass through convolutional layers. The important information that is scattered in the feature maps needs to be aggregated in order to form the speaker embedding. Statistics pooling is a simple way to obtain statistics from variable-length inputs. In the case of the baseline model with feature extraction, mean and standard deviation over time (length W) is used.

2.4 Baseline Model for Speaker Verification without Feature Extraction

For the baseline model for verification without feature extraction, we will use a model called RawNet [14]. RawNet's architecture shown in Table 2.6 is based on some other models for SV without feature extraction [17][15]. RawNet has some improvements over these models in terms of activation functions, new layers, better pooling layer, etc. This will give us a decent baseline performance of a system without feature extraction, so we can focus our experiments on more advanced techniques rather than updating older models with currently used NN features (activation functions, residual block configurations, etc.).

There are similarities to ResNet except that RawNet uses 1D convolutional layers in residual blocks to take raw waveforms as input. Residual blocks architecture follows the

original pattern in figure 2.5. The first activation function (leaky ReLU) is placed after the batch normalization. The second activation function is in the original place after the identity addition. The maximum pooling was added to reduce the size of the deep features after every block.

The last convolutional layer provides frame level features of the utterance in their respective position coming from the waveform. The length of the output of the last convolutional layer depends on the length of the input as explained. In order to form embeddings from these features, we have to use a pooling layer that will aggregate frame level features into utterance (speaker) level features of a fixed length. For this purpose we use GRU.

The utterance level features are finally transformed into embeddings by a fully connected layer with a linear activation. We do this to get speaker embeddings in the desired hyperspace with the desired dimensionality. For hyperparameter optimization reasons, the pooling layer and embedding layer dimension may differ. With the embedding layer, this forms the front-end section used in the validation and evaluation. We normalize embedding length (baseline length is 10) before sending them to the next layer. That is because our back-end classifier uses only embedding angles for classification. Then in the output layer, the embedding direction will be the only thing relevant in the training and evaluation while optimizing size would be pointless.

In the training phase, we will use a fully connected output layer with linear activation after the embedding layer. The output layer will transform embeddings into logits. Logits can be used by a loss function to have discriminative properties for each speaker according to the speaker label. Because the output layer is fully connected, discriminative logits also imply discriminative embeddings. Also, some loss functions can be applied directly on the embeddings layer to give embeddings directly the desired properties. The size of the output layer will depend on the number of speakers in the training dataset.

2.4.1 Recurrent Layers

A recurrent layer is an optional layer for the models for SV. Baseline model with feature extraction (see Table 2.2) for instance uses mean and standard deviation pooling over time (see Subsection 2.3.3). The recurrent neural networks (RNNs) were designed for processing sequences of input features and they can also be used for pooling and forming speaker embeddings. The use of the RNN instead of the mean and standard deviation pooling is a more sophisticated approach that we will use in the model without feature extraction.

The RNN layer is composed of the RNN blocks. The RNN layer processes inputs in sequential a fashion as displayed in Figure 2.7 where each block calculates its output based on the current input and output of the previous block. We will consider the Gated Recurrent Unit (GRU) [4] blocks for this example since different blocks may have different inputs and outputs. In the case of one-directional, single-layer RNN, the blocks are repeated in a sequence as many times as is the length of the input feature maps. Each GRU block has two inputs (feature input and hidden input) and one output (hidden feature output). In this model architecture, the feature input is one element of the feature maps that contains all channels. The hidden input takes the hidden feature output of the previous block. The first block takes the initial hidden feature vector that is set to all zeros.

As the hidden feature vector is passed through, the GRU blocks modify the hidden feature vector, send it to another block and copy it to the output. The output of each block is then based on the current and all previous feature inputs. The output of the last block,

Table 2.6: RawNet architecture. The meaning of the hyperparameters is the same as in Table 2.2. The identity in the residual blocks is taken before the first convolutional layer. The horizontal line in the residual blocks represents where identity is added. That means identity is summed with the output of the batch normalization before LeakyReLU activation. The GRU hyperparameter is the size of the hidden feature. The bold line after Embedding layer is where the speaker embedding is extracted. It is also the final layer in evaluation. Output layer is used for training. The size of the output layer is specific for the number of speakers in the training dataset.

Layer	Parameters
conv1D	3, 128, stride 3 BN LeakyReLU
Res Block	{ conv1D 3, 128 BN LeakyReLU conv1D 3, 128 BN LeakyReLU MaxPool3 } × 2
Res Block	{ conv1D 3, 256 BN LeakyReLU conv1D 3, 256 BN LeakyReLU MaxPool3 } × 4
pool	GRU 1024
Embedding	FC 1024
Output	FC 5994

therefore, aggregates the information from the entire feature maps. The output of the last block is passed to the fully connected embedding layer.

The blocks in the RNN layer contain the same weights. Using the RNNs we can process even the long sequences using a small number of parameters. The computational cost is however high due to many operations being applied sequentially.

Gated Recurrent Unit

Gated Recurrent Unit (GRU) [4] brings an improvement over the vanilla RNNs that suffer from the gradient vanishing. The hidden mechanism of the vanilla block sums the linear transformations of the input and hidden feature and then applies non-linearity such as *tanh*. In long sequences, the gradient of these operations becomes very small and the RNN will not converge. The vanilla RNNs do not learn long-term dependencies very well.

GRU overcomes the problem of vanishing gradients by using a gating mechanism shown in Figure 2.8 that adaptively remembers or forgets the propagated information and then

[2]https://colah.github.io/posts/2015-08-Understanding-LSTMs/

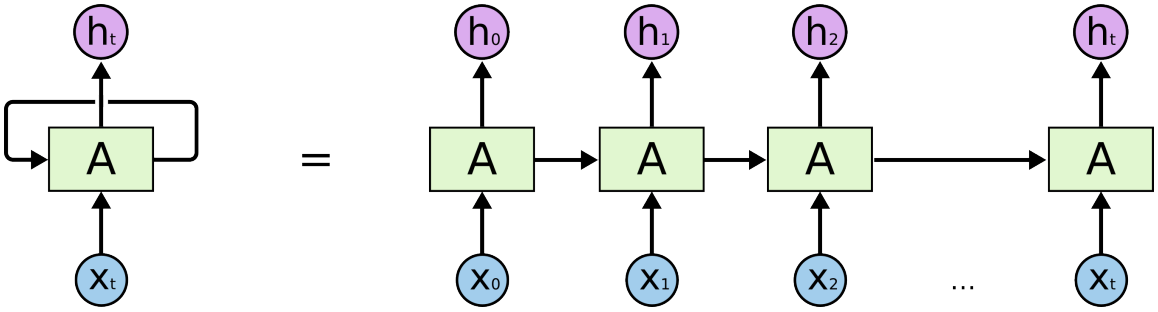


Figure 2.7: Unrolled recurrent NN[2] shows processing of the sequence of features X of length t. The output h is passed to the next block as a hidden state. The last output h_t then aggregates information from the entire sequence.

modifies its value. This is similar to the Long Short-Term Memory (LSTM) [11] however GRU achieves this at a lower computational cost and a lower number of hidden features.

At first, the hidden feature $\mathbf{h}_{t-1}$ and input feature $\mathbf{x}_t$ passed to the reset gate that uses weight set $\mathbf{W}_r$ and $\mathbf{U}_r$ and outputs reset value $\mathbf{r}_t$ (see Equation 2.2). The update gate performs the same operation with weight set $\mathbf{W}_z$ and $\mathbf{U}_z$ and outputs update value $\mathbf{z}_t$ (see Equation 2.3). The reset value $\mathbf{r}_t$, weight set $\mathbf{W}$ and $\mathbf{U}$ and hidden feature $\mathbf{h}_{t-1}$ are used to calculate new hidden feature $\tilde{\mathbf{h}}_t$ (see Equation 2.4). The hidden feature $\mathbf{h}_{t-1}$ is either propagated to the current hidden output $\mathbf{h}_t$ to the next block if the update gate output $\mathbf{z}_t$ is low or new hidden feature $\tilde{\mathbf{h}}_t$ is propagated if the update gate output $\mathbf{z}_t$ is high (see Equation 2.5). The modified hidden feature is either emphasized or diminished by the value of the update gate.

$$\mathbf{r}_t = \sigma\left([\mathbf{W}_r\mathbf{x}_t] + [\mathbf{U}_r\mathbf{h}_{t-1}]\right), \tag{2.2}$$

$$\mathbf{z}_t = \sigma\left([\mathbf{W}_z\mathbf{x}_t] + [\mathbf{U}_z\mathbf{h}_{t-1}]\right), \tag{2.3}$$

$$\tilde{\mathbf{h}}_t = \tanh([\mathbf{W}\mathbf{x}_t] + [\mathbf{U}(\mathbf{r}_t \odot \mathbf{h}_{t-1})]) \tag{2.4}$$

$$\mathbf{h}_t = \mathbf{z}_t\tilde{\mathbf{h}}_t + (1 - \mathbf{z}_t)\mathbf{h}_{t-1} \tag{2.5}$$

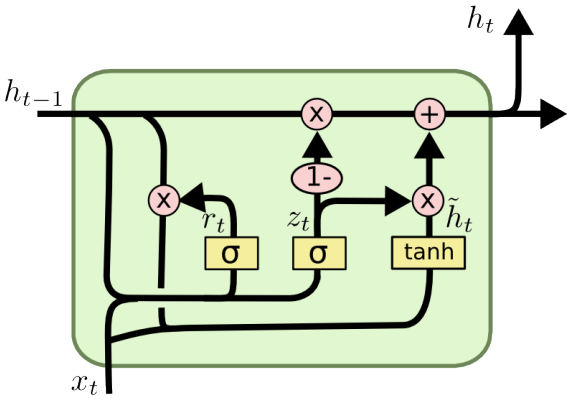


Figure 2.8: The internal connection[3] of the hidden mechanism.

[3] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

2.5 Loss Functions

A loss function is an objective function which value indicates how well a model performs a certain task. In the following subsections, the loss functions will focus on the tasks of discrimination among the classes or direct optimization of the embeddings or combination of those. In machine learning, we call them loss functions because loss is a value that we want to minimize to achieve the optimal state. The loss functions have to be numerically differentiable functions because they will be used in a gradient descend algorithm that needs a gradient of that function.

Usually, the loss functions are applied to the output of the last layer. We will explore different kinds of loss functions that also operate directly on the embeddings (Center Loss) or the output layer parameters (Basis Loss) to give embeddings desired properties that are supposed to increase DNN's embeddings discrimination quality.

In the following subsections we will consider the output layer weights as a matrix of trainable parameters $\mathbf{W} \in \mathbb{R}^{d \times n}$, the biases as a vector of trainable parameters $\mathbf{b} = [b_1, b_2, ..., b_n]^T, (b \in \mathbb{R})$ and embedding vector $\mathbf{x} = [x_1, x_2, ..., x_d]^T, (x \in \mathbb{R})$ where d is the size of the embedding and n is the number of classes.

2.5.1 Softmax Loss

Softmax loss [3, p. 209] is a very common loss function that is used for the task of class identification. In class identification, we predict the posterior probability of an example belonging to each of M classes, where the sum of all probabilities is equal to one. In training, we have a label (in this case one-hot encoded labels) associated with every training example. The softmax loss is a combination of two functions: softmax activation function and cross-entropy [3, p. 206] (CE) loss function. It is beneficial to understand CE loss separately because we may use it in softmax loss variations [6] later.

First, we get logits $y_{1..M}$ from the output layer with the weight matrix $\mathbf{W}$ and bias vector $\mathbf{b}$ and input (the embedding) into that layer $\mathbf{x}$ as:

$$y_m = \mathbf{W}_m^T \mathbf{x} + \mathbf{b}_m. \tag{2.6}$$

The softmax activation $\sigma(y_m)$ of the m-th logit from the output layer with M classes is calculated as:

$$\sigma(y_m) = \frac{e^{y_m}}{\sum_{j=1}^{M} e^{y_j}}. \tag{2.7}$$

The softmax activation output is normalized to be in the range of 0-1 and therefore, the sum of all outputs is 1. This is why softmax activation is used as a class posterior probability interpretation. The output of the output layer is a vector $\mathbf{y} = [y_1, y_2, ..., y_M]^T$. If we apply softmax function σ on every element of the output vector $\mathbf{y}$, we get vector of posterior probabilities $\mathbf{z} = [z_1, z_2, ..., z_M]^T$. Output of the network given by each example is associated with one-hot encoded label $\mathbf{l} = [l_1, l_2, ..., l_M]^T$. One-hot encoding means that the vector is filled with zeros except for one element at the index of the encoded class that is set to one. In order to create a loss function from class probabilities and class labels we use the CE loss:

$$L_{CE} = -\frac{1}{N} \sum_{i=1}^{N} \mathbf{l}_i^T \log(\mathbf{z}_i) \tag{2.8}$$

where $\mathbf{l}_i$ represents the label of the i-th example, $\mathbf{z}_i$ represents the posterior probability vector of the i-th example and N is the size of the dataset. For every example, CE selects one logit from the output layer based on the one-hot encoded label. We see that the CE value is minimal when the sum of logarithms of selected posterior class probabilities is maximal. Together we get the final softmax loss as:

$$L_S = -\frac{1}{N}\sum_{i=1}^{N} \mathbf{l}_i^T \log\left(\frac{e^{\mathbf{W}^T\mathbf{x}_i+\mathbf{b}}}{\sum_{j=1}^{M} e^{\mathbf{W}_j^T\mathbf{x}_i+\mathbf{b}_j}}\right), \tag{2.9}$$

where $\mathbf{x}_i$ represents the embedding of the i-th example. From the perspective of embeddings, the softmax loss enforces linear inter-class separation of the embeddings in the hyperspace they exist in. Linear inter-class separation has small effect on distributions of embeddings within a class and across the classes. We would like to enforce low intra-class covariance (within one class) and high inter-class covariance (among the classes). Those are the properties that should provide good generalization and improve evaluation performance.

2.5.2 Center Loss

Center loss [31] is an attempt to enhance class embeddings by decreasing intra-class covariance (see Figure 2.9). It also demonstrate good results in SV [10]. For every class, we will consider embedding center $\mathbf{c} = [c_1, c_2, ..., c_d]^T$, where d is the size of the embedding. Center loss will decrease the intra-class covariance by minimizing the average squared Euclidean distance of embeddings of each class from the center $\mathbf{c}$ of that class. The loss functions is then defined as:

$$L_C = \frac{1}{2N}\sum_{i=1}^{N} ||\mathbf{x}_i - \mathbf{c}_{y_i}||_2^2, \tag{2.10}$$

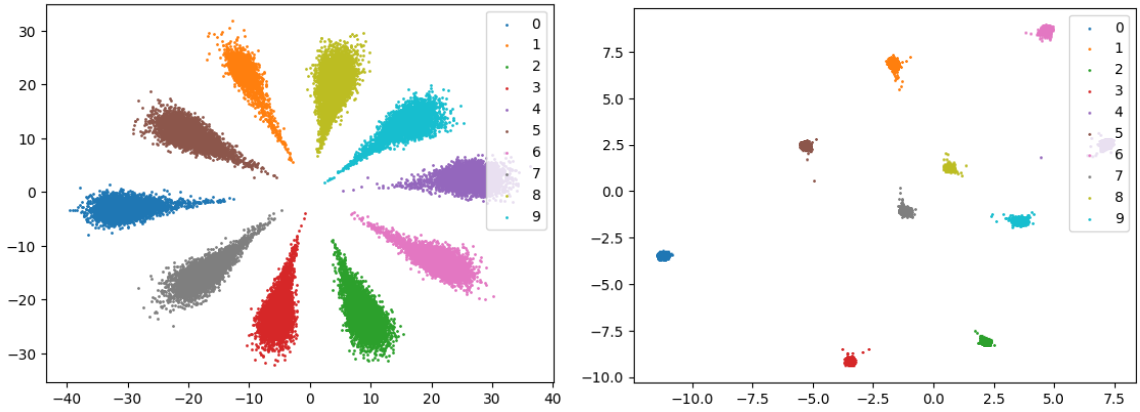
where $\mathbf{x}_i$ is the embedding of the i-th utterance, $\mathbf{c}_{y_i}$ is the center of i-th utterance class and N is the size of the dataset. From the perspective of the optimization algorithm, the centers $\mathbf{c}_{y_i}$ are treated as constants in the computational graph.

Center loss only affects embeddings to their respective class center. Used alone, the class centers would stay in their initial position (randomly initialized) and not have great inter-class separability. The job of increasing inter-class separability is offloaded to different loss function such as softmax loss. During training, the embeddings are affected by two or more loss functions. When softmax loss is optimizing the embeddings for the inter-class separability we need a function that will allow the class centers to move around from their initial position.

We will supplement a function that will update centers to a new position. The centers will move towards the current average center of the embeddings. The function that will calculate the difference between the current and new positions of the centers is:

$$\mathbf{\Delta c}_j = \frac{\sum_{i=1}^{N} \delta(y_i = j)(\mathbf{c}_j - \mathbf{x}_i)}{1 + \sum_{i=1}^{N} \delta(y_i = j)}, \tag{2.11}$$

where, N is the size of batch, $\mathbf{x}_i$ the embedding of the i-th sample, $\mathbf{c}_j$ current center of the j-th class, $\mathbf{\Delta}c_j$ is difference we will add to the center $\mathbf{c}_j$ of the j-th class, $\delta(condition)$ takes value of 1 if the *condition* is satisfied, else 0. The difference $\mathbf{\Delta c}_j$ is added to the current center $\mathbf{c}_j$ being scaled by α, that is suggested [10] at 0.5. This is performed manually after each pass of batch (or batch in stochastic training).



(a) Center loss with $\lambda = 10^{-3}$

(b) Center loss with $\lambda = 1$

Figure 2.9: Comparison of effects of the Center loss on the 2-dimensional embeddings with different λ values. The examples were trained on MNIST dataset.

We see that center loss pulls embeddings toward embeddings that are constants and we manually update them. Some implementations[4] suggest using a separate gradient descend algorithm to update the centers. This separate algorithm would treat embeddings as constants and centers as trainable parameters while the learning rate of this algorithm would be as high as previously used α (usually set to 0.5).

As mentioned, we have to use center loss along with another loss function that will provide some inter-class variation. It is commonly used in combination with softmax loss such as: $L = L_S + \lambda L_C$ where center loss has small weight factor λ, suggested at 10^{-3} according to the experiments [10].

2.5.3 Speaker Basis Loss

Speaker basis [10] loss is an attempt to increase inter-class covariance by weight regularization approach. This loss has to work together with another loss that is using logits where computation is based on the embeddings and output layer weights. This is because speaker basis loss assumes a correlation between weights of the output layer and embeddings. Speaker basis loss is suggested to work with softmax loss (as $L = L_S + L_{BC}$) [10] or hard negative mining (as $L = L_H + L_{BC}$) [10].

As mentioned, we have the output weight matrix $\mathbf{W}^{d \times M}$, where d is the size of embedding and M is the number of speakers. Then if $\mathbf{W} = [\mathbf{W}_1, \mathbf{W}_2, ..., \mathbf{W}_M]$, we will refer to the $\mathbf{W}_i$ as speaker bases. Because with softmax loss, in order to get logits, the described computation is performed as a dot product of embeddings and speakers respective $\mathbf{W}_i$ (bases) from the $\mathbf{W}$, we see that there is a correlation between each speaker's embedding and its respective basis during the training. By optimizing the bases we can enforce certain properties on embeddings.

In order to increase inter-class covariance, we will be decreasing the cosine of angles between all speaker bases in the loss function. We get speaker basis loss as:

[4] https://github.com/KaiyangZhou/pytorch-center-loss

[4] https://github.com/KaiyangZhou/pytorch-center-loss

$$L_{BC} = \sum_{i=1}^{M} \sum_{j=1, i \neq j}^{M} cos(\mathbf{W}_i, \mathbf{W}_j), \tag{2.12}$$

where M is the number of speaker bases.

2.5.4 Hard Negative Mining Loss

In the field of producing feature embeddings, by hard negative mining, we generally understand methods that focus on improving the class separation by selecting the hard negative pairs. The hard negative pairs are training examples that are likely to be mistaken in training. The hard negative pairs are selected in the training phase.

The hard negative mining method developed for the speaker verification that does not require mining of hard negative pairs has been proposed [10]. We are using the cosine of the angles between the speaker embeddings and bases to minimize the possibility of incorrect embedding classification. The hard negative mining function is defined as:

$$L_H = \frac{1}{N} \sum_{i=1}^{N} \sum_{\mathbf{W}_h \in \mathcal{H}_i} log(1 + exp(cos(\mathbf{W}_h, \mathbf{x}_i) - cos(\mathbf{W}_{y_i}, \mathbf{x}_i))), \tag{2.13}$$

where N is the size of the batch, $\mathbf{x}_i$ is the embedding of the i-th example, $\mathbf{W}_{y_i}$ is the basis of the y_i-th speaker. $\mathcal{H}_i$ is the set of H speaker bases that are closest to the y_i-th speaker embedding but do not belong to that speaker. The $\mathcal{H}_i$ represents the set of speakers bases that are likely to be mistaken. The set of closest bases of other speakers $\mathcal{H}_i$ is then an implementation of the idea of hard negatives applied to the speaker bases.

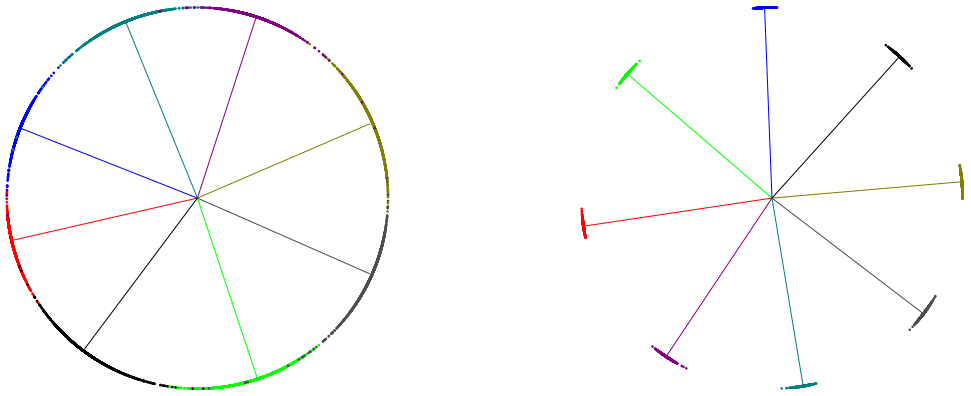
Because we use the cosine of the angle between embeddings and bases, this loss will be minimal when each speaker's embeddings will have high similarity with their respective basis and low similarity with the H nearest bases.

Hard Negative Mining loss itself is using a variation of logits, the cosine of the embedding and the basis. Therefore, we will be using it instead of softmax loss, only with speaker basis loss as $L = L_H + L_{BC}$. H is a hyperparameter that gives the best results at a value of 50 for the VoxCeleb2 dataset according to the experiments [10].

2.5.5 Additive Angular Margin Loss

Additive Angular Margin (AAM) [6] loss was originally developed for the purpose of deep face recognition in which it outperformed other loss functions. It solves the same problem of increasing class separability. It has been used in models for SV as well [1]. The AAM increases the class separability by enforcing an angular margin between the classes. It operates on the fully connected output layer, however, the bias is not used, because AAM loss is based on the angles between the embeddings and bases. AAM loss is based on the softmax loss, therefore it will be used instead of the softmax loss.

In the softmax loss we had logits calculated as a dot product of the embedding $\mathbf{x}$ and output layer bases $\mathbf{W}_i$ as $y_i = \mathbf{W}_i^T \mathbf{x} + \mathbf{b}_i$. Now we will use cosine of the angle θ between the embedding and the basis as the similarity and scale it by using constant s. Then we add angular margin m to it so we get $s \cos(\theta + m)$. The function for computing the elements of the output vector $\mathbf{y} = [y_1, y_2, ..., y_M]^T$ is defined as:



(a) Softmax Loss (b) Additive Angular Margin Loss

Figure 2.10: Comparison [6] of effects of Softmax loss and AAM loss on the normalized embeddings.

$$y_i = f_{AAM}(\theta_i) = \frac{e^{s(\cos(\theta_i + m))}}{e^{s(\cos(\theta_i + m))} + \sum_{j=1, j \neq i}^{M} e^{s(\cos(\theta_j))}}, \tag{2.14}$$

where θ_i is the angle between the embedding and the basis $\mathbf{W}_i$ and M is the number of speakers. We only add margin penalty m to the i-th angle (θ_i). The f_{AAM} value is high when the θ_i is low because we take the cosine of θ_i. Adding margin m to the θ_i will decrease the logit value. In training, the CE will force the value of f_{AAM} to be high. However, because we are minimizing the total angle of θ_i and m, the angle θ_i (the actual angle of the embedding and basis) will be under the margin m, therefore creating the margin separating the classes (see Figure 2.10). By using CE, we will get the final AAM loss function:

$$L_{AAM} = -\frac{1}{N} \sum_{i=1}^{N} \log \frac{e^{s(cos(\theta_{y_i} + m))}}{e^{s(cos(\theta_{y_i} + m))} + \sum_{j=1, j \neq y_i}^{M} e^{s(cos(\theta_j))}}, \tag{2.15}$$

where N is the number of samples in the dataset, θ_{y_i} is the angle between i-th embedding and the basis of the y_i-th speaker and M is the number of speakers. According to [6], scale s was suggested at 64 and margin m at 0.5 [6]. This was however used for the task of deep face recognition. The [17] shows a system for speaker recognition with feature extraction that uses AAM loss with $s = 30$ and $m = 0.2$. This will be taken into account when choosing the hyperparameters.

2.6 Squeeze-and-Excitation Networks

In the Subsection 2.3.1 we have discussed that the feature maps that are the output of the convolutional layers are multi-channel arrays where each channel is independently produced by the sum of convolutions of one kernel channel. There is little relation between the channels in the feature maps. The Squeeze-and-Excitation Networks [12] (and their modifications in subsections 2.6.2 and 2.6.3) aim to improve the representational power by discovering the interdependencies between the channels and applying recalibration of the channels. This is performed by the Squeeze-and-Excitation (SE) blocks that take the

feature maps as the input and output recalibrated feature maps with the same dimensions. The original implementation [12] suggests using the SE block in the end of the non-identity branch after the batch normalization. Different placements will be explored as well.

2.6.1 Squeeze-and-Excitation Blocks

The SE blocks [12] perform two operations, *squeeze* and *excitation* to obtain recalibration parameters, hence their name. We will consider the input of the SE block 2.11 which is a feature maps $\mathbf{U} \in \mathbb{R}^{H \times W \times C}$ as $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_C]$, where H, W, C are height, width and number of channels of the feature maps and $\mathbf{u}_i \in \mathbb{R}^{H \times W}$ are individual channels. For the 1-dimensional convolutional networks, we can consider $H = 1$. In the terms of processing speaker waveforms, we can imagine W dimension as time.

The $\mathbf{F}_{sq}$ operation performs the *squeeze* to obtain the global spatial information about the feature maps. The information is stored in the channel descriptor $\mathbf{z} \in \mathbb{R}^C$ where each value represents a statistic from the respective channel. The [12] suggests gathering the statistics as global average pooling from every channel. However, different aggregation methods are possible. The global average pooling channel descriptor is obtained as:

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i, j). \tag{2.16}$$

The $\mathbf{F}_{ex}$ operation performs the *excitation*. The *excitation* captures the interdependencies between the channels from the information in the channel descriptor $\mathbf{z}$ and creates a scale descriptor $\mathbf{s} \in \mathbb{R}^C$ that will be used for the recalibration. In order to do so, the *excitation* uses a simple gating mechanism. First, the *excitation* has to be able to learn non-linear dependencies in the channel descriptor. For that the *excitation* uses fully connected layer $\mathbf{W}_1 \in \mathbb{R}^{\frac{C}{r} \times C}$ and ReLU activation function δ. Then another fully connected layer $\mathbf{W}_2 \in \mathbb{R}^{C \times \frac{C}{r}}$ with a sigmoid activation function σ is applied after that to create the scale descriptor $\mathbf{s}$ with all values in the range between 0 and 1 that allows multiple channels to be emphasized independently in a non-mutually-exclusive manner. The whole gating mechanism is defined as:

$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W}) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z})) \tag{2.17}$$

The hyperparameter r is positive an integer that divides the C. By increasing the r, we create a bottleneck between the $\mathbf{W}_1$ and $\mathbf{W}_2$. The [12] has demonstrated that values up to 16 reduce dimensionality and therefore computational complexity as well, however, do not decrease the performance.

The output feature maps $\tilde{\mathbf{X}} \in \mathbb{R}^{H \times W \times C}$ as $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, ..., \tilde{\mathbf{x}}_C]$ is then created by recalibration of the channels in the feature maps $\mathbf{U}$ by the $\mathbf{F}_{scale}$ function. The $\mathbf{F}_{scale}$ uses the scale descriptor $\mathbf{s}$ and multiplies each channel $\mathbf{u}_c$ with its respective scale s_c:

$$\tilde{\mathbf{x}}_c = \mathbf{F}_{scale}(\mathbf{u}_c, s_c) = s_c \mathbf{u}_c. \tag{2.18}$$

2.6.2 Feature Map Scaling

The feature maps scaling (FMS) is a method that was proposed in [16] specifically to improve the performance of the RawNet by deriving more discriminative representations of the feature maps. It is focused on discovering interdependencies between the feature maps

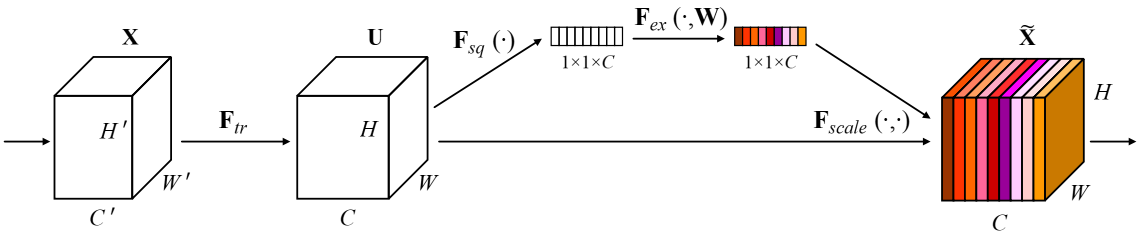


Figure 2.11: The application [12] of the SE block on the feature maps.

and rescaling the feature maps similarly to the SE networks. In addition, multiple methods as shown in Figure 2.12(a, b, c, d) for applying the scale descriptor. Also, the FMS blocks have been suggested to be placed after the residual block.

The filter statistics are obtained the same way as in the SE blocks, by filter-wise global average pooling from the input feature maps $\mathbf{c} = [\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_F]$, where F is the number of the feature map and $\mathbf{c}_i \in \mathbb{R}^T$, where T is the length of the filter. The squeeze and excitation bottleneck is omitted. Instead, the scale vector $\mathbf{r} = [r_1, r_2, ..r_F]$ (scale descriptor) is obtained by feed-forwarding the statistics vector through single fully-connected layer $\mathbf{W} \in \mathbb{R}^{F \times F}$ and applying the sigmoid σ activation function to obtain values in the range between 0 and 1 that can independently scale the filter in the feature maps.

Multiple scale vector applications have been proposed [16] to obtain the rescaled feature maps $\mathbf{c}' = [\mathbf{c}'_1, \mathbf{c}'_2, ..., \mathbf{c}'_F]$. First one being an additive method (see Equation 2.19). The multiplicative method is identical to the SE block method (see Equation 2.20). Finally, the combinations where proposed [16] with different order of the operations (see Equations 2.21 and 2.22).

$$\mathbf{c}'_f = \mathbf{c}_f + r_f \tag{2.19}$$

$$\mathbf{c}'_f = \mathbf{c}_f \cdot r_f \tag{2.20}$$

$$\mathbf{c}'_f = (\mathbf{c}_f + r_f) \cdot r_f \tag{2.21}$$

$$\mathbf{c}'_f = \mathbf{c}_f r_f + r_f \tag{2.22}$$

From the above mentioned methods, the best performance was achieved with the (2.22). The method using two separate scale vectors where one vector is used in addition and the other one in multiplication has also been proposed [16], however, this method did not outperform (2.22).

2.6.3 α-Feature Map Scaling

α-Feature Maps Scaling [18] (α-FMS) builds upon the FMS block. α-FMS still uses the scale vector $\mathbf{s}$ that consists of the independent rescaling values. The limitation of the scale vector from the FMS is that the values are limited in the range from 0 to 1 and that, it is used both for addition and multiplication. The α is a trainable parameter that is used instead of the scale vector in the addition in the rescale method. It is added to each feature maps filter (channel) as displayed in Figure 2.13 as a constant similarly as a bias would be used. The α is able to adapt to any value and therefore improves the discriminative

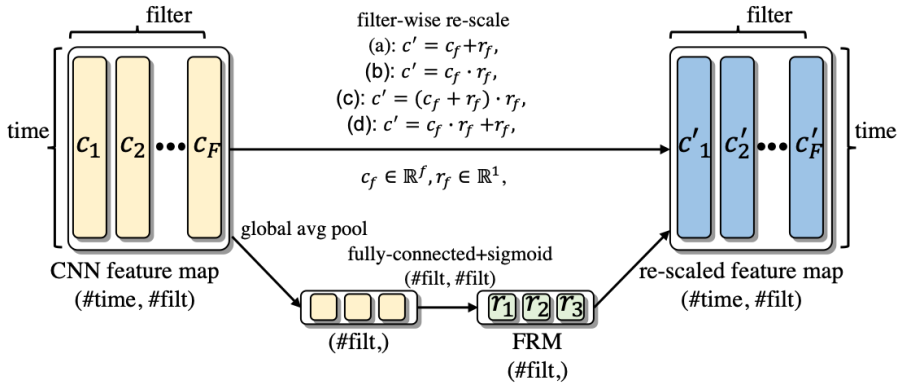


Figure 2.12: Feature Maps scaling pipeline [16] shown on 1-dimensional multi-channel feature maps. The output feature maps can be rescaled in multiple ways: $(a), (b), (c), (d)$.

capability of the feature maps as it was demonstrated. The α can be used as a scalar $\alpha \in \mathbb{R}$ or a vector $\alpha \in \mathbb{R}^F$, while the vector version performs better [18]. The input feature maps $\mathbf{M} = [\mathbf{m}_1, \mathbf{m}_2, ..., \mathbf{m}_F]$ is then rescaled to the output feature maps $\mathbf{M}' = [\mathbf{m}'_1, \mathbf{m}'_2, ..., \mathbf{m}'_F]$ as:

$$\mathbf{m}'_f = (\mathbf{m}_f + \alpha_f) \cdot s_f. \tag{2.23}$$

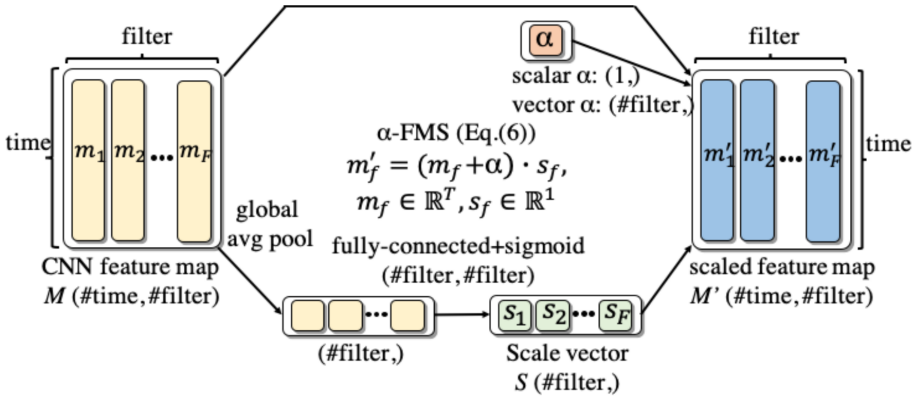
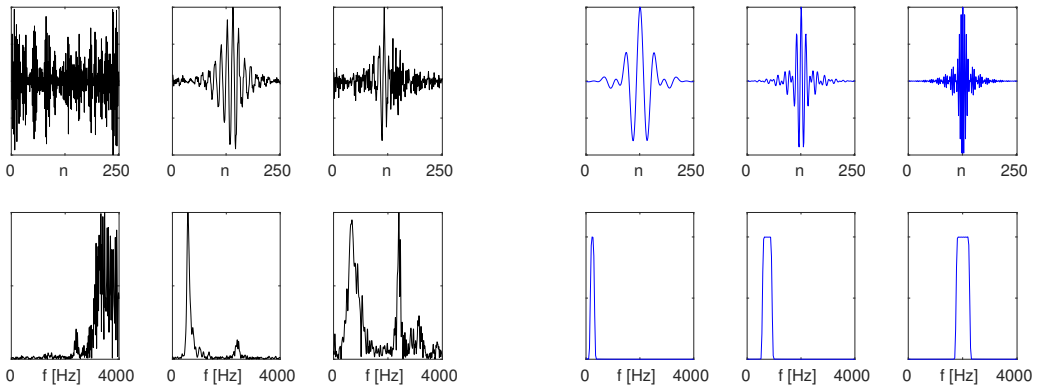


Figure 2.13: α-Feature Maps Scaling [18] with α added to the rescale method.

2.7 SincNet

In the Subsection 2.3.1 we have discussed the operation of convolutional layers. We know that the first convolutional layer has single-channel input (the raw waveform). The operation of the kernels in the first convolutional layer is identical to the definition of the convolution using the finite impulse response (FIR) filter in the digital signal processing (DSP) if the kernels have a stride of 1 and no biases with one exception that, the order of the coefficients is reversed. That means, we can approach the first convolutional layer as a series of filters with trainable coefficients. Because the kernels in the first convolutional



(a) Normal convolutional filters coefficients (first row) and their frequency responses (second row)

(b) incNet filters coefficients (first row) and their frequency responses (second row)

Figure 2.14: Comparison [26] of standard convolutional filters and band-pass filters implemented by SincNet

layer are the most important for extracting the features from the raw waveform we want them to extract an efficient representation of the input signal.

The idea of SincNet [26] is to enforce the filters of the input layer to act as a series of band-pass filters (see Figure 2.14). The parameters of the band-pass filters (the cut-off frequencies) are then trainable parameters. Because each filter is defined by cut-off frequencies, SincNet also reduces the number of parameters needed in the first convolutional layer from LF to $2F$, where F is the number of filters in the layer and L is the size of the kernel.

In frequency domain a band-pass filter G is defined as a difference of two rectangular functions:

$$G[f, f_1, f_2] = rect\left(\frac{f}{2f_2}\right) - rect\left(\frac{f}{2f_1}\right), \tag{2.24}$$

where f_1 is lower cut-off frequency and f_2 is higher cut-off frequency. The f_2 has to be initialized higher than f_1 in order to create a meaningful initial band-pass filter. The cut-off frequencies in all filters are initialized in accordance with the Mel scale where the bands are initialized next to each other with no overlap and no space between them. The last band ends at half of the sampling frequency. Covering the whole spectrum in the initialization helps faster converge.

In order to get the filter function in the time domain, we have to apply inverse Fourier transform. The rectangular function becomes the *sinc* function, hence the name SincNet. The band-pass convolution function g is then defined as:

$$g[n, f_1, f_2] = 2f_2 sinc(2\pi f_2 n) - 2f_1 sinc(2\pi f_1 n). \tag{2.25}$$

The $sinc(x)$ equals $sin(x)/x$ ($sinc(0)$ equals 1). We have a limited length of the filter to length L, therefore the ends of the filter are truncated. Because the Discrete Fourier Transform is calculated from a periodic signal, the Discrete Fourier Transform of such truncated filter will contain undesired artifacts. To mitigate this issue we use Hamming

window w on the filter by multiplying the functions. The hamming window makes the ends of the filter close to zero:

$$w[n] = 0.54 - 0.46 \cdot cos\left(\frac{2\pi n}{L}\right) \tag{2.26}$$

$$g_w[n, f_1, f_2] = g[n, f_1, f_2] \cdot w[n], \tag{2.27}$$

Finally, we can apply each band-pass filter with hamming window $g_w[n, f_1, f_2]$ using the convolution $*$ to the input signal $x[n]$ to get the filtered output signal $y[n]$:

$$y[n] = x[n] * g[n, f_1, f_2]. \tag{2.28}$$

In the digital signal processing, the convolution $*$ of the signal $x[n]$ and filter $h[n]$ of the length L is defined as:

$$x[n] * h[n] = \sum_{l=0}^{L-1} x[l] \cdot h[n-l]. \tag{2.29}$$

If we have created the filter $g_w[n, f_1, f_2]$ with trainable parameters f_1 and f_2 and reverse the order of the filter coefficients, then the convolution $*$ is equivalent with convolutional layer convolution that has no bias, zero padding and stride equal to 1, and therefore is easy to use in NNs. It is also a fully differentiable function and therefore suited for optimization by the gradient descend.

The length of the filters L is recommended at 251 while using 80 filters, however, we stick to 128 filters as this is the original RawNet hyperparameter. The L of 251 was recommended for the input signal with sampling frequency $16kHz$, same as we use.

SincNet was suggested as a replacement for the first convolutional layer, however it was also suggested using Layer Normalization before SincNet [26].

2.8 Layer Normalization

The purpose of Layer Normalization [2] (LN) is similar to the purpose of Batch Normalization (BN). That is reducing the effects of covariate shift [2] by normalizing the input feature values to a limited range. LN uses the same normalization statistic as BN: mean $\boldsymbol{\mu}$ and standard-deviation $\boldsymbol{\sigma}$, however, they are computed in a different way. The LN transposes the direction (see Figure 2.15) of computation of the normalization statistics used in BN. BN computes $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ across the batch. LN computes the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ across the features. Let $\mathbf{a}$ be the input feature and H the size of this feature. Then we calculate the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ as:

$$\mu_i = \frac{1}{H}\sum_{i=1}^{H} a_{i_j}, \tag{2.30}$$

$$\sigma_i = \sqrt{\frac{1}{H}\sum_{j=1}^{H}(a_{i_j} - \mu_i)^2}, \tag{2.31}$$

where $i \in N$, N is the size of the batch and i_j is the j-th element in the i-th feature in batch. The statistics are then applied in normalization to obtain the output of the normalization layer $\mathbf{y}$:

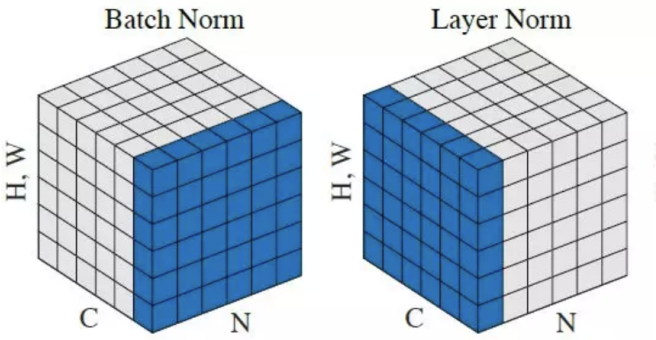


Figure 2.15: Demonstration[6] of statistics computation in Batch Normalization and Layer Normalization

$$y_i = \frac{\mathbf{a_i} - \mu_i}{\sigma_i} * \boldsymbol{\gamma} + \boldsymbol{\beta} \tag{2.32}$$

The $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are gain and bias parameters that are applied to every output feature y_i with element-wise multiplication and element-wise addition. The gain and bias are trainable parameters that are used for the magnitude reparametrization of the feature elements. Gain is initialized at a vector of ones and bias is initialized at a vector of zeros. Because these parameters are of fixed size the input of fixed length has to be used. This is addressed in the section about evaluation 3.3.3.

Because the LN and BN stabilize the covariate shift by normalization of the features, their application in NNs speeds up convergence and reduces training time. In the case of LN, this is not the only motivation. LN is preprocessing and rescaling layer for SincNet.

[6] https://paperswithcode.com/method/layer-normalization

Chapter 3

Training and Evaluation

This chapter describes a detailed setup for training, validation and evaluation, and how the results are obtained. The authors of the RawNet did not specify how exactly they obtained the final results such as numbers of repeating the same experiment or doing early stopping on the validation trials, etc. At first, we will train the baseline RawNet model in its original configuration to get the baseline results relative to our setup. Then we will proceed to apply new techniques and compare the results.

For the training and experiments, we used datasets VoxCeleb1 [23] and VoxCeleb2 [5]. These datasets were also used in VoxSRC-2020 [24]. In the speaker recognition challenges, the participants are provided with the training and evaluation data and evaluation trials without the ground truth. The ground truth with results was released after the deadline when the results of the participants are submitted.

3.1 Datasets

The VoxCeleb datasets contain data obtained from YouTube videos. All speech recordings contain no silence. The single-channel waveforms are sampled at $16kHz$ and encoded as 16-bit Signed Integers. The waveforms are converted to single-precision floating-point datatype and scaled in the range from -1.0 to 1.0 when they are used in the experiments.

VoxCeleb1 dataset used in VoxSRC-2020 for the validation purposes [24]. In the development set, there are 1,211 speakers and 148,642 utterances. In the test set, there are 40 speakers and 4,874 utterances. VoxCeleb1 comes with the evaluation list of trial pairs for the test set. There are two more lists of trial pairs (trial-H, trial-E) however they are drawn from the development set and can not be used for evaluation if we train or validate on VoxCeleb1.

VoxCeleb2 was used in the VoxSRC-2020 and it comes with 5,994 speakers, 1,092,009 utterances in the development set and 118 speakers and 36,237 utterances in the test set. There are no trials for the VoxCeleb2 which means it is only used for training. As the VoxSRC-2020 suggests, we should use VoxCeleb2 for training and VoxCeleb1 for validation.

VoxSRC-2020 did not specify if test trials, they used for the evaluation in the challenge is the same trials that is now provided with the VoxCeleb1 test set. We use the trials that now come with the VoxCeleb1 and are generally used to evaluate speaker verification models in various studies, however, we can not use the VoxSRC-2020 results with certainty.

The VoxCeleb1 test set and its list of trial pairs will be used for the evaluation and it is going to be the only evaluation set. The VoxCeleb1 development set will be used for

validation with trials that we generate ourselves. Trial-H and trial-E then will not be used because they already use the same utterances as the validation trials.

3.2 Training

We perform stochastic gradient-based optimization [3, p. 240] of the loss functions described in the Section 2.5, using an optimizer described in Subsection 3.2.2 based on the stochastic gradient descent method. In stochastic training, one iteration over dataset (epoch) is split into batches and weights of the model are updated after passing each batch.

The VoxCeleb1 and VoxCeleb2 are very extensive in their size and training takes lots of time. In order to reduce training time and a number of experiments and get reasonably reproducible experiments results, we have reduced the number of epochs from the original 40 to 25 and used same the pseudorandom number generator seed for the weight initialization and other random aspects of training such as batch shuffle. We select the final weights of an experiment by the best value from on validation trials. The validation trials are also used to estimate overfitting or underfitting.

The size of batches was based on the size of the memory. The optimal size of the batch to fill the 8GB VRAM is 60.

Training models without feature extraction on VoxCeleb1 have shown overfitting of the models, precisely speaker overfitting. The solution to that was two-step training with a pre-training scheme [15] that used more residual blocks and global average pooling layer instead of recurrent layer. By using the VoxCeleb2 dataset, we have more speakers and more novel data, therefore we can omit the pre-training phase [14]. VoxCeleb2 without pre-training also shows better performance [14] than VoxCeleb1 with pre-training.

3.2.1 Data Preparation

The datasets are split into several folders that represent the individual speakers. Each speaker's folder is split into multiple sessions, each session contains several recordings from that speakers' session. The length of recordings varies from less than 3 seconds to over 10 seconds. The recordings are sampled at $16kHz$. The datatype of the waveforms for training is a single-precision floating-point number in the range from -1 to 1.

Data in each batch needs to be of the same size. For that reason, we use constant-length chunks of the recordings 3.69 seconds long (this length is suggested by RawNet [14]). The recordings that are longer are cut while the beginning of the chunks is chosen randomly. At the sample rate of $16kHz$, that accounts for 59,049 samples for each waveform in the batch. Recordings that are shorter than 3.69 seconds are repeated enough times to fill the length of 59,049 samples. For the validation and evaluation, we use full-length recordings, and therefore validation and evaluation data are passed through one waveform at a time.

The early implementation of RawNet suggests [14] using pre-emphasis, following the [28]. The pre-emphasis in this case is high pass filter that is implemented as a finite impulse response filter with three coefficients: $b_0 = -0.97$, $b_1 = 0$, $b_2 = 1$.

Another pre-processing method introduced in [16] is simple normalization by the maximal absolute value of the waveform. Both pre-processing techniques will be compared in the experiments.

3.2.2 Optimizer

The purpose of the optimizer is to perform stochastic gradient-based optimization [3, p. 240]. This means that we minimize the value of the loss function by calculating the first-order gradient of the loss function and discovering its minimum (there are many minimums) by moving against the direction of that gradient. The current value of the loss function is calculated by feed-forwarding the current batch on the current parameters of the model. In the context of deep learning, the gradient is partial derivatives of the loss function with respect to its parameters (model's weights and biases). The gradient describes how the loss value will be affected (increase or decrease) if we change the parameters in a certain way. In order to move towards a minimum of the function, we subtract the partial derivatives from the respective parameters with a certain learning rate. We iterate the process on all batches and multiple epochs.

Because stochastic algorithms perform updates of the parameters on every batch, that leads to gradient descend that changes the direction often because a single batch does not represent the entire dataset well and we only get an approximation of the gradient for the entire dataset. However, the advantage is that such algorithms are less likely to stop in the local minimum.

The optimizer used in all experiment is Adam [19] as this optimizer demonstrated competitive results [19]. More precisely, we are using the AMSGrad variation. Adam introduces gradient moment estimates. The exponential decay rates for the moment estimates are new hyperparameters that are left at default values $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

The learning rate α is a hyperparameter which purpose is to scale the size of the update of the parameters from the computed gradient. With too large updates the algorithm would possibility diverge from a minimum. With too small updates the algorithm would take too much time to converge or possibility get stuck in a local minimum. The solution to that is learning rate decay. In learning rate decay, we update the learning rate throughout the training, starting at a high learning rate allowing the algorithm to take large steps in the beginning to find the minimum quicker and then when the learning rate decreases, the algorithm is able to find minimum without diverging from it. The initial learning rate is default Adam learning rate $\alpha = 0.001$. After every batch, the α is divided by $1 + 10^{-4} * t$, where t is the number of the batch in the whole training.

Weight regularization

The purpose of regularization [3, p. 256] is to improve the generalization of the model. The generalization is the property of the model to perform well on the different datasets than the dataset the model was trained on. More precisely we want the model to generalize across the evaluation and training dataset. A common instance of bad generalization if overfitting. The overfitting means that the model covers the training dataset with too much certainty while the validation and evaluation sets are misfitted.

One of the solutions for that is weight regularization. Weight regularization adds another objective function to the final loss function that directly regulates the weights. The weight regularization is implemented in the optimizer. Adam implements the regularization as decoupled weight decay regularization [21] which is an L2 (second norm) weight penalty that is modified to work along with Adam's adaptive learning rate. The weight decay decreases the absolute weight values. The idea of decreasing the weights to obtain better generalization is that with too high weight values the model will get very specific discriminative properties that will only suffice a specific dataset which is the training dataset.

The rate of the weight decay is a hyperparameter which default value is suggested at 10^{-3} [21]. With too low weight decay the overfitting may occur. With too high weight decay the model might learn too slowly and underfit. We will use a weight decay rate of 10^{-4} and increase the rate in the case of overfitting.

3.3 Evaluation

In the evaluation (and validation, which has the same approach) we take the trained model and take the output of the embedding layer to get the speaker's embedding. The speaker verification performance is measured on the evaluation trials that belong to the test dataset. The evaluation trials are specified in a list of trials. Each trial (also called test or) contains a pair of speaker recordings. The trials also contain ground truth that specifies if the recordings belong to the same speaker, although the ground truth might not always be provided such as in the VoxSRC-2020 challenge when the ground truths are unknown to the participants however the organizers can evaluate the results submitted by the participants. The evaluation trials with ground truths are released after the submission deadline.

The length of the evaluation trials is 37,720 pairs. The list of trials can be longer than the test dataset because one recording can be paired with multiple other recordings. The output of the evaluation (and validation as well) is a list of soft scores that describe the similarity of the pairs where a higher number means higher similarity. The soft scores are not bounded to any range and hard decisions are also not required for the metrics that we use.

A simple way to calculate the similarity of the embeddings from a pair of recordings is the cosine similarity. The cosine similarity simply returns the cosine of the angles of the embedding vectors $\mathbf{x_A}$ and $\mathbf{x_B}$:

$$cos(\theta) = \frac{\mathbf{x_A} \cdot \mathbf{x_B}}{||\mathbf{x_A}||||\mathbf{x_B}||} \tag{3.1}$$

The cosine similarity returns values in the range from -1 (lowest similarity) to 1 (highest similarity).

3.3.1 Validation and Validation Trial Lists

Validation is used for tuning the right hyperparameters for the model. The validation results are obtained as SV trials on the validation dataset and validation trials. We perform validation throughout the training after every epoch. It is commonly used to detect overfitting. Overfitting occurs when the results of the validation start diverging. The weights of the epoch with the best validation results are also used for the evaluation of that model.

The reason for performing validation as the SV is that the entire training dataset is used for the discriminative class training, so we can not validate speakers outside of that dataset other way than using their embeddings for the SV. Also, we prefer to measure SV performance during the validation which is more relevant than the class discrimination performance.

The validation trial was generated by ourselves. It contains 40,000 pairs of recordings where the recordings were taken from the VoxCeleb1 dataset. The trial was generated with a balanced number of positive and negative ground truths. The first part contains 20,000 pairs of randomly chosen recordings where each recording is paired with a different random recording from the same speaker. This part has all positive ground truths. In the

second that has all negative ground truths we take 20,000 pairs, each composed of different speakers. Then for each speaker, we choose random recording. We generated only one validation trial that we used in all our experiments.

3.3.2 Detection Error Tradeoff

If we have relative scores of similarity and ground truths from a certain test we can calculate Detection Error Tradeoff [22] (DET) curve for that model on that dataset. The Detection Error Tradeoff purpose is very similar to the receiver operating characteristic (ROC) curve. The DET curve is a relation between two tradeoff error types: false negative rate (missed detections) axis and false positive rate (false alarms) axis. We can now interpret rates as probabilities. A real system will have some rate of overlap in the classification, giving some false positives or false negatives.

By moving along the DET curve we get different rates of false positives and false negatives. Each point on the DET curve is associated with a threshold value of the similarity score that gives certain rate of false positives and false negatives. Generally, the system with a lower overall DET curve is better (see Figure 4.5).

From the evaluation standpoint, there is one point on the DET curve that is important for the evaluation and that is the equal error rate (EER). EER is the point on the DET curve where the rates of false positives and false negatives are the same. We will use EER to evaluate all our experiments. The lower the EER the better.

In real-world speaker verification use, the DET curve can be used to determine the threshold that will be either very conservative with low false positives rates or more liberal where false positive rate should be higher. An instance of a system with low false positives rates for can be a security system with speaker verification. A system with higher false positive can be optimal for forensic applications where discovering the target is more important than the presence of false positives. In the case of cosine similarity, we would obtain a cosine threshold for a hard decision from the point on the DET curve. The DET curve can also be used to compare the capabilities of different systems, showing that some systems will perform better for tasks where low false positives rates are needed and vice versa.

3.3.3 Test Time Augmentation

In normal evaluation and validation, we use full-length recordings that we feed into the model and the recurrent layer will process input sequences of any length. We will refer to this as a full time evaluation. Test Time Augmentation (TTA) [5] is an augmentation method performed in the evaluation or validation. In TTA we split each recording into segments of fixed length with some overlap, extract embeddings of individual segments and take the mean of those embeddings as the final embedding. TTA was demonstrated to improve evaluation results [16]. We will also use TTA in some cases when we can only process recordings of fixed length such as when we use Layer Normalization with gain and bias vectors (see Section 2.8). In our case, we use the same length of segments as training segments (59049 samples) with 20% overlap (that accounts for 11810 samples). If the last segment is shorter than 59049 samples the overlap of the last segment is increased so the last segment fills 59049 samples. Recordings shorter than 59049 samples are repeated enough times to fill 59049 samples.

Chapter 4

Experiments

This chapter describes our approach for improving the performance of the system for speaker verification without feature extraction and the results obtained in our experiments. We have started with the baseline configuration of the RawNet [14] (described in Section 2.4) and obtained the baseline results with our training setup described in Chapter 3. Then we proceeded to perform experiments with new techniques and improve the baseline performance.

Each section of this chapter focuses on a certain set of techniques such as loss functions, etc. In every section, we will perform experiments with the goal of finding the best technique. The comparisons of EERs (see Section 3.3.2) of different experiments are compared in absolute values, however, also in relative values of percents between two EERs. We call relative comparison Relative Error Rate (RER). At the end of each section we will compare and discuss the results. The best solutions from every section will be used in combination to create the best system. The performance of the best system will be compared to a model with feature extraction and also be used in fusion with that model.

As a code base we have used RawNet [14] implementation[1] by its authors and performed necessary modification ourselves.

4.1 Loss Function Experiments

The experiments with loss functions described in Section 2.5 were performed with the model with the original architecture (Table 2.6), data pre-emphasis and full time evaluation (Subsection 3.3.3).

4.1.1 Softmax Loss

This experiment (Figure 4.1) with the softmax loss function is equivalent to the baseline setup and it is used to replicate the proclaimed results of the authors of the RawNet [14]. RawNet was supposed to yield 3.6% EER. In our conditions we achieved 3.7% EER. All other loss functions will have to improve on this result to be considered in the final model.

4.1.2 Center Loss

First, we performed two experiments comparing placement of the center loss before and after the normalization of embeddings (the output of the embedding layer is normalized before feed-forwarding into the output layer) (see Section 2.4). The placement before the

[1] https://github.com/Jungjee/RawNet/

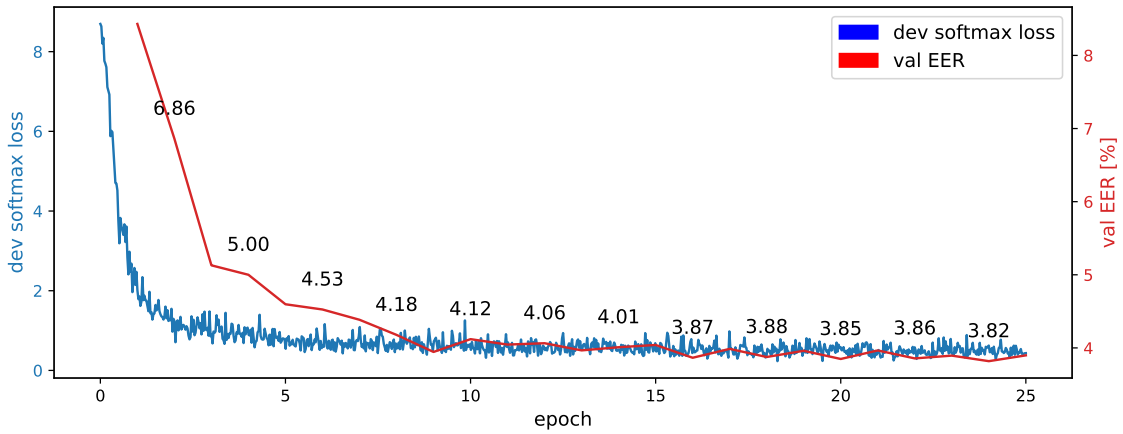


Figure 4.1: Training progress of the softmax loss and validation results. The best validation EER was achieved after the 24th epoch.

embedding normalization achieved EER of 4.15% and placement after the embedding normalization achieved EER of 4.06%. The version placed after the normalization of the embeddings performed better for understandable reason that if minimize the average squared distance of the embeddings from their respective class center after the normalization of the embeddings we make the task easier by not minimizing their average squared length as well.

At this point, we used implementation[2] that utilized separate gradient a descend optimizer for updating the position of the class centers. This is achieved by using center loss function L_C value as a loss that is minimized by this separate optimizer. The separate optimizer is using the embeddings as constants and class centers as trainable parameters which is the opposite of how the parameters are used in the main optimizer.

After that, we implemented the class center update function as described in [10]. In this version we manually update the class centers to the average of their respective class embeddings according to the equation described in Subsection 2.11. This version performed better (achieving EER of 3.91%) than the first implementation. The reason for that might be that gradient descend uses squared distance of the centers from the class average center. That might result in too large steps in gradient update. The manual center update has very predictable step.

The center loss was used in combination with softmax loss with recommended weight $\lambda = 0.001$ and recommended [10] update weight $\alpha = 0.5$: $L = L_S + \lambda L_C$. The combination of center loss and softmax loss did not outperform the softmax loss alone. We will add speaker basis loss to this combination.

4.1.3 Speaker Basis Loss with Center Loss

Speaker Basis Loss showed minor improvement (3.79% EER) when used with the softmax loss with weight 1.0 compared to the center loss with softmax loss. However, the speaker basis loss and center loss were intended to be used together with softmax loss. The total loss function is: $L = L_S + L_{BS} + \lambda L_C$ where λ is suggested at 0.001 according to [10]. This combination was also used by the authors of the RawNet in the early implementations trained on VoxCeleb1.

[2] https://github.com/KaiyangZhou/pytorch-center-loss

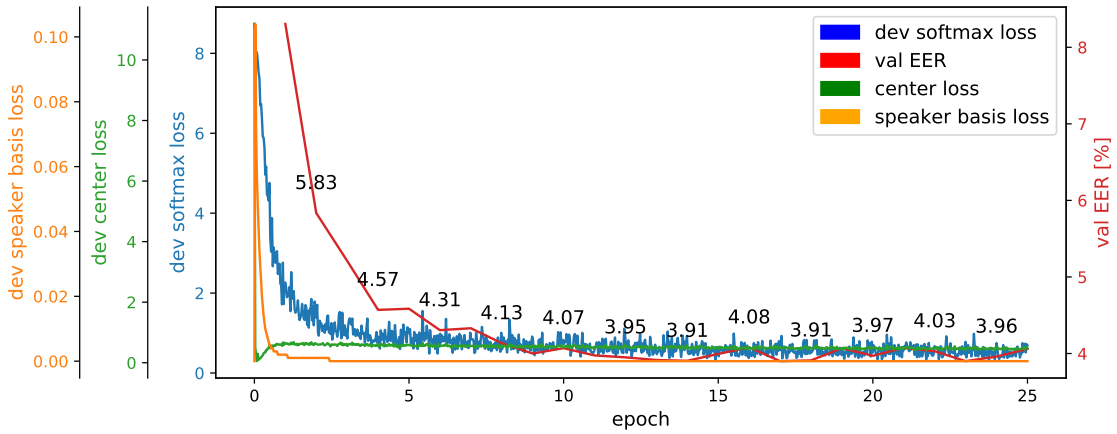


Figure 4.2: Center Loss (green) with Basis Loss (Orange) and Softmax (Blue) training progress. The best evaluation EER was achieved after the 18th epoch.

The softmax with speaker basis and center loss achieved EER of 3.84% (Figure 4.2). That is relative 3.7% EER increase from the softmax loss alone. The softmax loss alone also showed better validation results.

After that, we tried to improve this result by making changes to how the output logits are calculated. Before the embedding is multiplied with the output layer weights, we normalize the length of the embedding to one and the length of the weight bases to 1 as well. In theory, this should allow the basis loss and softmax loss to be trained more independently.

The results of this modification were poor. Achieving only about 16% EER. We concluded that the change of the length of the embeddings from the original 10 to 1 was the cause of this because the center loss operates in Euclidean space not based on angle, thus using the center loss with the wrong length. After we reverted to the original length of the embedding of 10, we compared the results of experiments with the original $\lambda = 0.001$ and increased $\lambda = 0.01$. These experiments yielded results of 6.59% EER with $\lambda = 0.001$ and 6.72% EER with $\lambda = 0.01$.

After unsatisfying results with normalization of the output layer weight bases, we reverted to the original version without the normalization of the output layer weight bases and embedding length of 10. Now we again performed an experiment with increased $\lambda = 0.01$. This yielded EER of 4.39% at the 17th epoch. After that, the validation results started diverging which points to overfitting. We tried another experiment with increased weight decay (described in Subsection 3.2.2) from original 10^{-4} to 10^{-3}. This avoided overfitting but caused underfitting because this experiment yielded only 5.52% EER.

These experiments showed that the hyperparameters suggested by [10] are already well tuned and modifications of the hyperparameters did not improve the results. The EER of 3.84% represents relative 3.78% EER increase over the baseline result of 3.7%. For comparison, the experiments [10] on the models with feature extraction (64-dimensional Mel-filterbank) trained on the VoxCeleb1 showed relative 15.8% EER decrease and relative 11% EER decrease when trained on VoxCeleb2.

4.1.4 Hard Negative Mining Loss

The first experiment with hard negative mining loss was performed with standard convolutional filters and band-pass filters weight decay 10^{-4} and the hyperparameter that sets the number of closest bases H was set to 50 [10]. This yielded the EER of 4.75%. This best validation result was on at the 16th epoch, then the validation results slightly diverged suggesting slight overfitting. We ran another experiment with weight decay of 10^{-3}. This resulted in too much weight regularization that caused strong underfitting and nearly stopped the progress of the validation results.

We decided to conduct experiments with slight changes to the weight decay to discover the optimal weight decay. We conducted three experiments with weight decays of $4 \cdot 10^{-4}$, $2 \cdot 10^{-4}$ and $5 \cdot 10^{-5}$. These experiments yielded EER of 10.4%, 6.4% and 5.5% respectively. The weight decay of $4 \cdot 10^{-4}$ and $2 \cdot 10^{-4}$ caused slight underfitting again as the validation progress continually converged however at a slower rate. The weight decay of $5 \cdot 10^{-5}$ showed overfitting as the validation progress reached the best EER after the 11th epoch. The original value of 10^{-4} seems to be the best value for weight decay.

With that, we proceeded to tune the hyperparameter H which specifies the number of closest neighbor bases (see Section 2.5.4). We conducted three experiments with H values of 30, 75 and 100, achieving EER of 4.54%, 5.54% and 7.2% respectively. The $H = 30$ with EER of 4.54% is the only value that improved over the original $H = 50$ with EER of 4.75%.

On the models with feature extraction [10], the Hard Negative Mining loss achieved relative 28.6% EER improvement over softmax loss on VoxCeleb1 with $H = 100$ and relative 21% EER improvement over softmax loss on VoxCeleb2 with $H = 50$. We have discovered that for the model without feature extraction, better H is 30. However, the model without feature extraction did not improve over softmax loss with Hard Negative Mining loss.

4.1.5 Additive Angular Margin Loss

Based on the analysis of a model for speaker recognition [1] and original the implementation of AAM [6], we have chosen the margin hyperparameter for the first experiment as $m = 0.3$ as this is somewhat middle ground between $m = 0.2$ [1] and m $m = 0.5$ [6]. The scale was set as $s = 30$ according the model for speaker recognition [1] because scale as a hyperparameter can greatly depend on a task as demonstrated by the experiments with center loss. We used existing implementation[3] of the AAM.

We have also used a function for gradually increasing the margin from 0 to 0.3 to allow embeddings to fit better early in the training. The margin is updated with high granularity on the batch basis to avoid big changes in the objective function with a faster increase in the beginning and a slower increase when reaching the maximal margin. The function for updating the margin is defined as $m(1 - e^{-0.3(i+b/N)})$, where i is the index of the epoch, b is the index of the batch, N is the number of batches in one epoch and m is the final margin.

In the early experiment, the AAM loss showed signs of overfitting as the model validation reached the best EER very early (around the 5th epoch) and then started diverging. In order to improve generalization, we increased the weight decay from 10^{-4} to 10^{-3}.

This configuration yielded better results (see Figure 4.3), achieving the EER of 3.38%. Then another two experiments followed with increased margin to 0.4 and 0.5 achieving 3.14% EER and 3.53% EER respectively.

[3] https://github.com/cvqluu/Angular-Penalty-Softmax-Losses-Pytorch

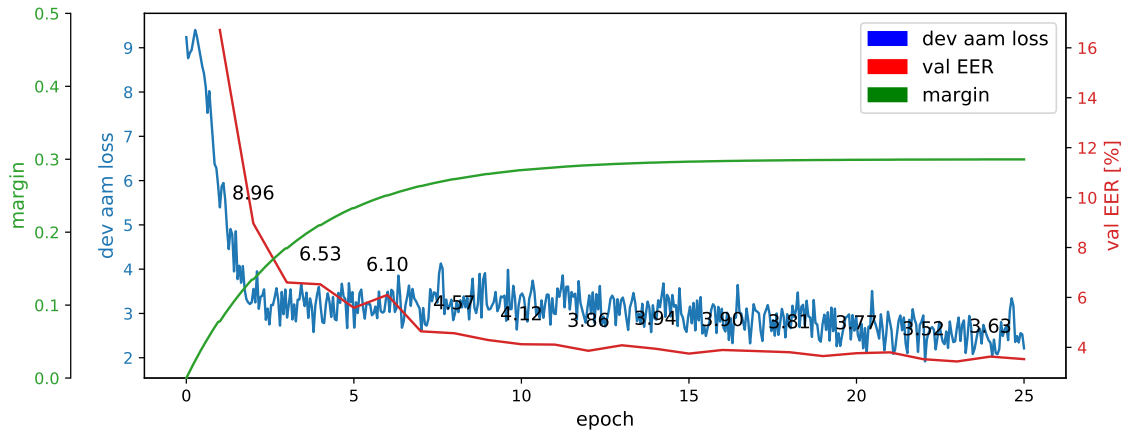


Figure 4.3: AAM loss validation (red) with weight decay of 10^{-3} does not show any signs of overfitting. The AAM loss seems to diverge sometimes (around 5th epoch), that is due to increasing the margin and making the task more difficult. The maximum margin is 0.3 and scale is 30

The margin of 0.4 seems to be the best margin for the original architecture configuration presented in Table 2.6 with weight decay 10^{-3}, achieving 15.1% RER improvement over softmax loss.

4.1.6 Conclusion on the Loss Functions Experiments

We have proven on a baseline model for SV without feature extraction that only some loss functions that focus on embedding optimization have the potential to improve SV performance. The AAM was the only loss function that outperformed (see Table 4.1) the softmax loss.

We discovered that even if we use large datasets such as VoxCeleb2 with many speakers we may still encounter speaker overfitting. The AAM loss was also the only function that responded well to the changes of weight decay. AMM and Hard Negative Mining loss was not used in [14] neither with VoxCeleb1 or VoxCeleb2.

All presented loss functions (see Section 2.5) improved embedding extraction capabilities on the models with feature extraction [10] compared to softmax loss. Since AAM is the only loss function that improved embedding extraction for the model without feature extraction, that demonstrates the specific way of how models without feature extraction extract deep features and aggregate them into embeddings compared to models with feature extraction.

We will continue to use softmax loss in the following experiments that will focus on different aspects of the model because AAM loss might distort results in those experiments as it requires increased weight decay and two additional hyperparameters.

4.2 Residual Block Architecture Improvements

As a next step, we performed experiments with improved residual block architecture (Section 2.3.2). We replaced original residual blocks (see (a) in Figure 2.5) with full pre-activation blocks (see (e) in Figure 2.5) as this architecture showed the best results [9]. Because in full pre-activation architecture Batch Normalization and ReLU activation func-

Table 4.1: Comparison of tested loss functions and their hyperparameters that achieved the best results. wd stands for weight decay. We used full time evaluation and data pre-emphasis.

Loss	Hyperparameters	EER [%]
Softmax	wd=10^{-4}	3.7
Softmax + λCenter + Speaker Basis	wd=10^{-4}, $\lambda = 0.001$	3.84
Hard Negative Mining + Speaker Basis	wd=10^{-4}, $H = 30$	4.54
Additive Angular Margin	wd=10^{-4}, m=0.4, s=30	**3.14**

tion precede the convolutional layer, we skip the first pair of BN and ReLU in the first block so that the convolutional layer is the first layer that is fed with the input after the initial layers (see RawNet architecture in 2.6). The rest of the model hyperparameters remains unchanged and softmax loss was used.

We compared training the model with data pre-emphasis and data normalization (see Subsection 3.2.1) to discover a better approach. We also compared using full time evaluation and test time augmented evaluation (see Section 3.3.3).

Table 4.2: Comparison of data pre-emphasis and data normalization with full time evaluation (FT) and test time augmented evaluation (TTA) on the baseline model (see Table 2.6) now with full pre-activation residual blocks (see (e) in Figure 2.5).

	FT EER [%]	TTA EER [%]
Pre-emphasis	3.39	3.02
Normalization	3.47	3.16

The experiment with pre-emphasis and full time evaluation that differs only in the residual block architecture from the first experiment with softmax loss (see Subsection 4.1.1) achieved EER of 3.39% (see Table 4.2) compared to the original EER of 3.7%. That is relative 8.4% EER improvement. Because of that improvement, we will use full pre-activation residual blocks for all following experiments.

Full pre-activation residual blocks with data normalization also improved over the original architecture however not as much. EER of 3.47% is only relative 6.1% EER improvement. We will continue with experiments comparing data pre-emphasis and data normalization and see how they responded to different architectures.

Test time augmentation evaluation improved results of both data pre-emphasis and data normalization even more achieving EER of 3.02% and 3.16% respectively. The fact that TTA evaluation improves the results might indicate that the deep feature aggregation by the GRU has certain limitations when processing long sequences of deep features. In TTA evaluation the length of the evaluation segments is the same as the length of the training segments (see Subsection 3.3.3), which might be contributing factor to better speaker embedding aggregation because the length of the training and evaluation segments is the same.

Another contributing factor might simply be the too large length of some evaluation segments. Most recordings in the evaluation trials are longer than the 3.69 seconds long segments that we use for training and some are over 12 seconds long. The GRU has a gated mechanism that allows a previous hidden feature to be „forgotten“ and updated (see Subsection 2.4.1) which diminishes the importance of older inputs. This might be suitable

for the task of statistical machine translation which GRU was developed for [4] where recent inputs matter more than older inputs, however, from the perspective of aggregating the speaker embedding every input fed into RNN has the same information weight.

In the following experiments, TTA evaluation results will be the primary results for selecting the best techniques.

4.3 Squeeze-and-Excitation Networks Experiments

The Squeeze-and-Excitation networks experiments focus on exploration of SE blocks (see Subsection 2.6.1) and methods with a similar focus: Feature Map Scaling (FMS) (see Subsection 2.6.2) and α-Feature Map Scaling (α-FMS) (see Subsection 2.6.3). The placement of these methods in the residual blocks is after addition (see Figure 2.5). We used both data pre-emphasis and data normalization. The full pre-activation residual blocks were used and softmax loss was used to avoid possible distortion and hyperparameter cross-dependency of the AAM loss. We evaluated the models with full time evaluation and test time augmented evaluation as well. We used existing implementations of SE blocks[4], FMS[5] and α-FMS[6].

SE bocks have the reduction hyperparameter r that specifies the bottleneck of the two matrices (see Subsection 2.6.1). This is set to 16 as this brings the reduction of training parameters and training time but does not impact performance (see Subsection 2.6.1).

For the FMS we used implementation with *mul-add* method (see Subsection 2.6.2).

In terms of α-FMS, we used version that uses vector α parameter (see Subsection 2.6.3).

Table 4.3: Comparison of SE, FMS and α-FMS with waveform pre-emphasis (pre-emp) or waveform normalization (norm) and FT evaluation or TTA evaluation.

	Data Preparation	FT EER [%]	TTA EER [%]
SE	pre-emp	3.31	2.94
SE	norm	3.28	2.65
FMS	pre-emp	3.24	2.75
FMS	norm	3.02	**2.57**
α-FMS	pre-emp	3.39	2.75
α-FMS	norm	3.1	**2.56**

The experiments (see Table 4.3) showed that SE networks and SE networks inspired methods improved in all experiments across the board compared to the architecture without them. An interesting observation is that, in these experiments, SE blocks, FMS and α-FMS responded well to data normalization performed with it better than with data pre-emphasis. The α-FMS with data normalization and TTA was the best method (achieving 2.56% EER), however, the FMS with data normalization and TTA (achieving 2.57% EER) is trailing very close behind to decide which method would be best suited for the final composition of the model. Therefore, we will use both and compare them again in the final composition.

[4] https://github.com/ai-med/squeeze_and_excitation
[5] https://github.com/Jungjee/RawNet/
[6] https://github.com/Jungjee/RawNet/

4.4 SincNet Experiments

When experimenting with SincNet, we decided to perform multiple experiments with individual methods to discover individual effects of those methods and then how they interact with each other. Those methods are Layer Normalization (see Section 2.8), data normalization (see Subsection 3.2.1) and SincNet (see Section 2.7). We already use full pre-activation residual blocks, softmax loss and TTA evaluation which is necessary during evaluation because LN can only process segments of fixed length (see Section 2.8). We used the existing implementation of SincNet[7] and LN[8] that comes with initialization of gain and basis.

The first experiment (see Table 4.4) was about the comparison of the effects of LN as the first layer and before the first convolution layer (no SincNet yet) as it was suggested by the authors of SincNet [26]. This was performed both with data pre-emphasis and data normalization. We see that LN alone decreased performance (see Table 4.2 where full pre-activation residual blocks with softmax achieve 3.02% EER with data pre-emphasis and 3.16% EER with data normalization).

Table 4.4: Effects of Layer Normalization in the first row also compares use of data pre-emphasis and data normalization. Layer Normalization with SincNet in the second row results with data pre-emphasis (pre-emp) and data normalization (data norm). Everything is evaluated using TTA evaluation.

	data pre-emp (EER [%])	data norm (EER [%])
Layer Normalization	3.27	3.57
Layer Normalization + SincNet	3.16	3.33

Next, we performed experiments with SincNet as it was intended to be used with LN before SincNet [26]. Again, we performed the experiments with data pre-emphasis and data normalization. This yields 3.16% EER for data pre-emphasis and 3.33% EER for data normalization (see Table 4.4), which is an improvement over LN alone however not an improvement over architecture without SincNet and LN.

A strange phenomenon was observed during training of the models with LN and SincNet and that was rapidly increased training time. One epoch with a model without SincNet and LN takes from 1:15 to 1:55 hours depending on how many methods are applied and what loss functions we use. One epoch with the model with SincNet and LN takes 7:55 hours.

We ran another single experiment with SincNet, data normalization and without LN. This was undertaken to discover the effects of SincNet without LN, but also to debug the long training. This experiment yielded only 4.5% EER. However, the training time of this experiment reverted back to the normal time of one epoch taking 1:52 hours.

In the training script, we placed two timers. The first timer measured the time of forwarding the batch through the model and calculating loss and the second timer measured the time of back-propagation of the loss and update of the parameters. The forward and back-propagation time on LN alone took on average 7.27 and 240.38 milliseconds and on SincNet alone it took on average 10.78 and 291.55 milliseconds. Only when LN and SincNet were used together the forward and back-propagation time was on average 11.27 and 1283.58 milliseconds. We see that forward time did not increase much, however, back-propagation

[7] https://github.com/mravanelli/SincNet
[8] https://github.com/mravanelli/SincNet

time increased about four times compared to the experiments where SincNet and LN were used alone. This indicates the high computational complexity of the back-propagation graph caused by the combination of LN and SincNet.

The SincNet was supposed to increase the speed of convergence of the NN and increase interpretability of the feature extraction in the first layer [26]. We could not achieve these results and decided not to continue with the experiments with SincNet also due to the problem of multiplied training time.

4.5 Deeper Architecture Experiments

The residual convolution networks were shown to have increased performance when very deep architectures were used [8]. For this experiment, we decided to take the current model architecture (full pre-activation residual blocks) and increase its depth including the number of the convolutional layer channels in the deeper layers. The deeper model architecture for SV verification without feature extraction is shown in Table 4.4. Softmax loss was employed again.

The experiment with this architecture yielded only 4.1% EER with full time evaluation and data pre-emphasis, which is above normal architecture achieving 3.39% EER (see Table 4.2).

Models with high a number of parameters typically require larger training datasets. The overfitting was not observed in the validation results. We have limited number of epochs to 25 mainly due to long training time (see Section 3.2), however, models with more parameters also typically require more passes over the dataset. Due to these limitations we decided not to continue in finding the cause of the increased EERa and discontinued the experiments with deeper architectures. Original RawNet architecture (see Table 2.6) seems to be well optimized for the current setup.

4.6 Combination of the Best Methods

The best methods tested so far are AAM loss, full pre-activation residual blocks, FMS and α-FMS. The FMS and α-FMS achieved very similar results which means we will compose two models with these methods to see how they respond to the AAM loss and pick a better method. Data pre-emphasis performed better on the model without FMS and α-FMS while data normalization performed better with FMS and α-FMS. Therefore, we will test both data pre-emphasis and data normalization again. TTA is again the preferred evaluation method.

The first series of experiments combined (see Table 4.5) full pre-activation residual blocks, FMS and AAM loss. The margin of AAM loss was selected at 0.3, 0.4 and 0.5 for multiple experiments with weight decay of 10^{-3} as this was the best weight decay according to the experiments with loss functions (see Section 4.1). The best result was achieved with data pre-emphasis and a margin of 0.4. With TTA evaluation, the best result was 2.5% EER.

In the second series of experiments (see Table 4.6) we replaced FMS with α-FMS. We also skipped experiments with a margin of 0.5 as they showed some overfitting in the experiments with FMS. In this case, the combination of α-FMS and AAM loss achieved the best EER of 2.48% with a margin of 0.3. This is also the best combination of methods and hyperparameters so far.

Figure 4.4: Modified RawNet with increased depth as two new residual blocks were added with increased number of kernel channels to 512. See Table 2.6 for the hyperparameters explanation.

Layer	Parameters
conv1D	3, 128, stride 3 BN LeakyReLU
Res Block	{ BN LeakyReLU conv1D 3, 128 BN LeakyReLU conv1D 3, 128 MaxPool3 } × 2
Res Block	{ BN LeakyReLU conv1D 3, 256 BN LeakyReLU conv1D 3, 256 MaxPool3 } × 4
Res Block	{ BN LeakyReLU conv1D 3, 512 BN LeakyReLU conv1D 3, 512 MaxPool3 } × 2
pool	GRU 1024
Embedding	FC 1024
Output	FC 5994

Table 4.5: Feature Map Scaling (FMS) experiments with Additive Angular Margin, full time evaluation vs. test time augmented evaluation and data pre-emphasis vs. data normalization.

	Data Preparation	margin	FT EER [%]	TTA EER [%]
FMS	pre-emp	0.5	3.76	2.96
FMS	norm	0.5	4.01	3.47
FMS	pre-emp	0.4	2.96	**2.5**
FMS	norm	0.4	2.94	2.68
FMS	pre-emp	0.3	2.72	2.52
FMS	norm	0.3	2.82	2.69

An interesting observation on these experiments is that, the models with FMS or α-FMS and AAM loss performed better when they used data pre-emphasis (see Table 4.5 and 4.6) compared to the models with FMS or α-FMS and softmax loss that performed better with data normalization (see Table 4.3).

Table 4.6: α-Feature Map Scaling (α-FMS) experiments with Additive Angular Margin, full time evaluation vs. test time augmented evaluation and data pre-emphasis vs. data normalization.

	Data Preparation	margin	FT EER [%]	TTA EER [%]
α-FMS	pre-emp	0.4	3.15	2.88
α-FMS	norm	0.4	3.44	3.04
α-FMS	pre-emp	0.3	2.62	**2.48**
α-FMS	norm	0.3	4.86	4.47

4.7 System Fusion

The purpose of system fusion is to create a system that combines the results of more systems (we will use a fusion of two systems) to achieve results that are better than the results of the individual systems. In order for this to work, the individual systems should achieve complementary results. The system fusion will work well when one system will be able to correct the mistakes of the other system. This means that the systems should have for instance a different approach for the extraction of the features, therefore we will fuse a model with feature extraction and a model without feature extraction that we created in Section 4.6. The overall performance of the systems should not be too different because worse performing system might create a fusion system which results will not be improved, instead, they will be between the two systems.

We will perform system fusion on the score level which means we will use soft scores of both systems to obtain a fused score. In this case, we use the mean score of the systems. Ideally, the systems should calculate the embedding similarity the same way so the soft scores are in the same range and have a similar EER threshold. In this case, both systems use cosine similarity (see Subsection 3.3).

We were provided with evaluation trial soft scores and details of a system with feature extraction thanks to our supervisor. The architecture of this model is described in Section 2.3 and Table 2.2. This model was also trained on the VoxCeleb2 dataset. The model was trained on log Mel-frequency bank energies (fbank) features with a dimensionality of 40. The fbank features are extracted from waveforms sampled at 16kHz. The fame length is 25 milliseconds and the frame-shift is 10 milliseconds. The batch size was 128 and the input length was 200 frames. The model was trained using Stochastic Gradient Descend (SGD) optimizer with momentum set to 0.9. The loss function was Additive Margin (AM) [30] loss with a margin of 0.2 (exponentially increasing to that threshold) and scale 30. AM loss also enforces margin between the classes as AAM does, however, AM does not add margin to the angle of the embedding and basis but rather it subtracts margin from cosine of the embedding and basis angle. This model achieved EER of 1.46%.

Our best model is based on the experiments in Section 4.6. It is based on the RawNet architecture (see Section 2.4) with full pre-activation residual blocks (see Subsection 2.3.2), α-Feature Map Scaling, Additive Angular Margin loss (see Subsection 2.5.5) with a margin of 0.3 and scale of 30, data pre-emphasis (see Subsection 3.2.1), test time augmented evaluation (see Subsection 3.3.3) trained with weight decay of 10^{-3}. This model achieved EER of 2.48%.

The fusion of these models (see Figure 4.5) achieved EER of 1.49%. This is higher than the model with feature extraction. We decided to try fusion with the second best model that only differs in using FMS instead of α-FMS and achieved EER of 2.5% (see Section

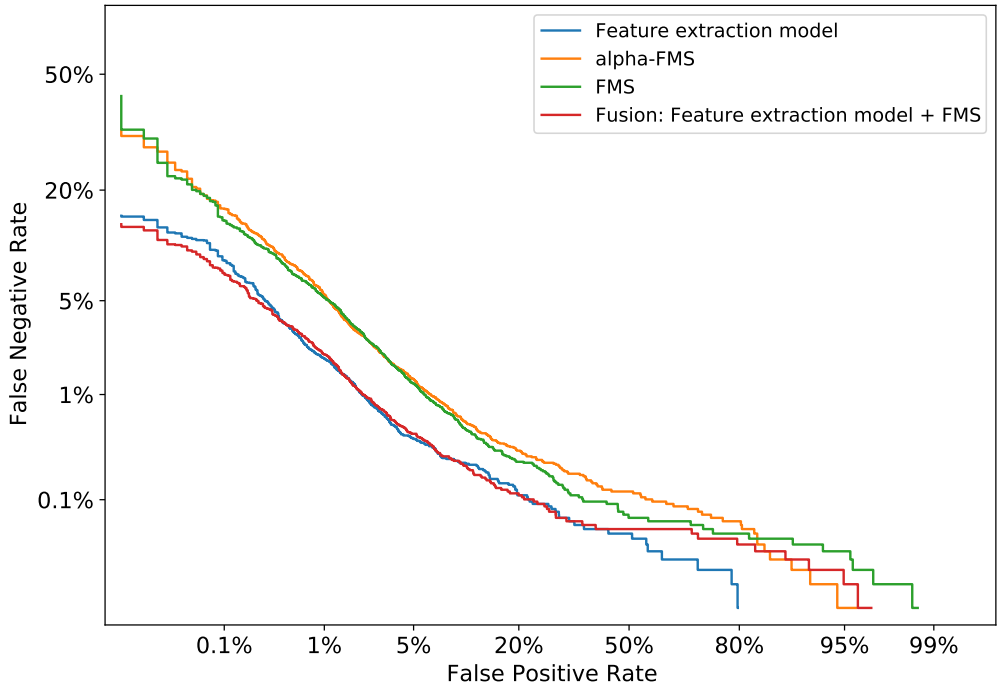


Figure 4.5: Detection Error Tradeoff curves of the following systems: Feature extraction model: model with feature extraction (1.46% EER), alpha-FMS: model without feature extraction with α-Feature Map Scaling (2.48% EER), FMS: model without feature extraction with Feature Map Scaling (2.5% EER), Fusion: fusion of the model with feature extraction and model without feature extraction with FMS (1.45% EER).

4.6). Fusion with this models achieved EER of 1.45% which is a tiny improvement. The probable cause of minor improvement in system fusion is the fact that the models with feature extraction still do not achieve results that are close enough to the models with feature extraction.

Chapter 5

Conclusion

The goal of this thesis was to get acquainted with speaker verification and models for speaker verification with and without feature extraction that utilize the speaker embedding extraction scheme. Then we applied state-of-the-art techniques on top of the baseline model without feature extraction (RawNet) to improve its performance, and compared it to a baseline model with feature extraction and created a fusion of the systems.

The SotA techniques focused on various aspects of the architecture and also training and evaluation. The baseline model achieved 3.7% EER with softmax loss. First, we experimented with various loss functions: Softmax loss, Center loss, Speaker Basis loss, Hard Negative Mining loss and Adding Angular Margin loss. The Adding Angular Margin performed best, achieving 3.14% EER.

The order of the Batch Normalization, Convolutional Layers and activation functions in the residual blocks matters as the full pre-activation architecture performed best.

Increasing the model's depth did not yield good results. The solution to that could be increasing the number of epochs or using a larger dataset than VoxCeleb2 or using data augmentation.

Methods inspired by Squeeze-and-Excitation (Feature Map Scaling and α-Feature Map Scaling) allowed to discover interdependencies in feature maps, thus increasing performance with α-FMS edging over FMS.

SincNet was supposed to discover a more meaningful representation of the first layer convolution filter. However, we could not achieve the results claimed by the authors of SincNet. It also had a very long training time.

The results of data pre-emphasis and data normalization were swinging from one another depending on the other employed techniques. In the final model, data pre-emphasis performed best.

Test Time Augmentation evaluation improved results across the board. This points toward limitations of GRU when aggregating long sequences of inputs when evaluating long recordings.

The final model (see Section 4.6) achieved an EER of 2.48%. That is 33% RER improvement over the baseline with EER of 3.7%. An important observation is RER improvements of some methods in different conditions. As more methods were stacked, the effects on RER improvements started decreasing. For instance, AAM loss achieved 15.1% RER improvement over the baseline RawNet with softmax loss (see Subsection 4.1.5). In the final composition of the methods (see Section 4.6) it only achieved 3% RER improvement over softmax loss. This indicates that the potential of current architecture and methods is exhausted on the current dataset.

The difference in the performance of the models with feature extraction and without feature extraction is still substantial. The provided model with feature extraction trained on the same dataset achieved an EER of 1.46%. The best model without feature extraction achieved only 2.48% EER (see Section 4.6). The fusion of those systems using the mean of soft scores did not improve results, achieving only 1.49% EER. However, fusion with the second best model without feature extraction achieving EER 2.50% (see Section 4.6) improved the fusion results achieving 1.45% EER, showing that the best model might not be able to correct the results of the other model correctly.

5.1 Future Work

There are several techniques that address the mentioned limitations. Some of them were not implemented due to limited time, while some are out of the scope of this thesis.

New architectures are available in the form of Res2Net that allow the multi-scale representation of the features and demonstrate performance gains [7]. Res2Net was already used in some models submitted to VoxSRC 2020.

Models trained on larger datasets tend to perform better. Larger datasets could also allow training deeper models with more parameters. In the open version of the SV part of VoxSRC 2020, participants were allowed to use datasets outside of the provided VoxCeleb2 dataset and data augmentation was allowed. That led to better results.

The recurrent neural network showed limitations when aggregating long inputs. This could be addressed by using Attentive Statistics Pooling for Deep Speaker Embedding [25].

More loss functions are also available such as Generalized End-to-End Loss for Speaker Verification [29].

Bibliography

[1] Alam, J., Boulianne, G., Burget, L., Dahmane, M., Diez Sánchez, M. et al. Analysis of ABC Submission to NIST SRE 2019 CMN and VAST Challenge. In: *Proc. Odyssey 2020 The Speaker and Language Recognition Workshop*. 2020, p. 289–295. DOI: 10.21437/Odyssey.2020-41.

[2] Ba, J. L., Kiros, J. R. and Hinton, G. E. *Layer Normalization*. 2016.

[3] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 0387310738.

[4] Cho, K., Merriënboer, B. van, Gulcehre, C., Bahdanau, D., Bougares, F. et al. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, October 2014, p. 1724–1734. DOI: 10.3115/v1/D14-1179.

[5] Chung, J. S., Nagrani, A. and Zisserman, A. VoxCeleb2: Deep Speaker Recognition. In: *INTERSPEECH*. 2018.

[6] Deng, J., Guo, J., Xue, N. and Zafeiriou, S. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, p. 4685–4694. DOI: 10.1109/CVPR.2019.00482.

[7] Gao, S.-H., Cheng, M.-M., Zhao, K., Zhang, X.-Y., Yang, M.-H. et al. Res2Net: A New Multi-Scale Backbone Architecture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Institute of Electrical and Electronics Engineers (IEEE). Feb 2021, vol. 43, no. 2, p. 652–662. DOI: 10.1109/tpami.2019.2938758. ISSN 1939-3539.

[8] He, K., Zhang, X., Ren, S. and Sun, J. Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, p. 770–778. DOI: 10.1109/CVPR.2016.90.

[9] He, K., Zhang, X., Ren, S. and Sun, J. Identity Mappings in Deep Residual Networks. In: Leibe, B., Matas, J., Sebe, N. and Welling, M., ed. *Computer Vision – ECCV 2016*. Cham: Springer International Publishing, 2016, p. 630–645. ISBN 978-3-319-46493-0.

[10] Heo, H.-S., Jung, J. weon, Yang, I.-H., Yoon, S.-H., Shim, H. jin et al. End-to-End Losses Based on Speaker Basis Vectors and All-Speaker Hard Negative

Mining for Speaker Verification. In: *Proc. Interspeech 2019*. 2019, p. 4035–4039. DOI: 10.21437/Interspeech.2019-1986.

[11] Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. *Neural Computation*. november 1997, vol. 9, no. 8, p. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. ISSN 0899-7667.

[12] Hu, J., Shen, L. and Sun, G. Squeeze-and-Excitation Networks. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, p. 7132–7141. DOI: 10.1109/CVPR.2018.00745.

[13] Ioffe, S. and Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. JMLR.org, 2015, p. 448–456. ICML'15.

[14] Jung, J. weon, Heo, H.-S., Kim, J. ho, Shim, H. jin and Yu, H.-J. RawNet: Advanced End-to-End Deep Neural Network Using Raw Waveforms for Text-Independent Speaker Verification. In: *Proc. Interspeech 2019*. 2019, p. 1268–1272. DOI: 10.21437/Interspeech.2019-1982.

[15] Jung, J. weon, Heo, H.-S., Yang, I., Shim, H. jin and Yu, H.-J. Avoiding Speaker Overfitting in End-to-End DNNs Using Raw Waveform for Text-Independent Speaker Verification. In: *INTERSPEECH*. 2018.

[16] Jung, J. weon, Kim, S. bin, Shim, H. jin, Kim, J. ho and Yu, H.-J. Improved RawNet with Feature Map Scaling for Text-Independent Speaker Verification Using Raw Waveforms. In: *Proc. Interspeech 2020*. 2020, p. 1496–1500. DOI: 10.21437/Interspeech.2020-1011.

[17] Jung, J.-W., Heo, H.-S., Yang, I.-H., Shim, H.-J. and Yu, H.-J. A Complete End-to-End Speaker Verification System Using Deep Neural Networks: From Raw Signals to Verification Result. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, p. 5349–5353. DOI: 10.1109/ICASSP.2018.8462575.

[18] JungJee weon, ShimHye jin, KimJu ho and YuHa Jin. α-feature map scaling for raw waveform speaker verification. *The Journal of the Acoustical Society of Korea*. The Acoustical Society of Korea. september 2020, vol. 39, no. 5, p. 441–446.

[19] Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In: Bengio, Y. and LeCun, Y., ed. *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[20] Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. 1998, vol. 86, no. 11, p. 2278–2324. DOI: 10.1109/5.726791.

[21] Loshchilov, I. and Hutter, F. Decoupled Weight Decay Regularization. In: *ICLR*. 2019.

[22] Martin, A., Doddington, G., Kamm, T., Ordowski, M. and Przybocki, M. A. The DET curve in assessment of detection task performance. In: *EUROSPEECH*. 1997.

[23] Nagrani, A., Chung, J. S. and Zisserman, A. VoxCeleb: a large-scale speaker identification dataset. In: *INTERSPEECH*. 2017.

[24] Nagrani, A., Chung, J. S., Huh, J., Brown, A., Coto, E. et al. *VoxSRC 2020: The Second VoxCeleb Speaker Recognition Challenge*. 2020.

[25] Okabe, K., Koshinaka, T. and Shinoda, K. Attentive Statistics Pooling for Deep Speaker Embedding. *Interspeech 2018*. ISCA. Sep 2018. DOI: 10.21437/interspeech.2018-993.

[26] Ravanelli, M. and Bengio, Y. Speaker Recognition from Raw Waveform with SincNet. In: *2018 IEEE Spoken Language Technology Workshop (SLT)*. 2018, p. 1021–1028. DOI: 10.1109/SLT.2018.8639585.

[27] 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.

[28] Vergin, R. and O'Shaughnessy, D. Pre-emphasis and speech recognition. In: *Proceedings 1995 Canadian Conference on Electrical and Computer Engineering*. 1995, vol. 2, p. 1062–1065 vol.2. DOI: 10.1109/CCECE.1995.526613.

[29] Wan, L., Wang, Q., Papir, A. and Moreno, I. L. Generalized End-to-End Loss for Speaker Verification. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, p. 4879–4883. DOI: 10.1109/ICASSP.2018.8462665.

[30] Wang, F., Cheng, J., Liu, W. and Liu, H. Additive Margin Softmax for Face Verification. *IEEE Signal Processing Letters*. 2018, vol. 25, no. 7, p. 926–930. DOI: 10.1109/LSP.2018.2822810.

[31] Wen, Y., Zhang, K., Li, Z. and Qiao, Y. A Discriminative Feature Learning Approach for Deep Face Recognition. In: Leibe, B., Matas, J., Sebe, N. and Welling, M., ed. *Computer Vision – ECCV 2016*. Cham: Springer International Publishing, 2016, p. 499–515. ISBN 978-3-319-46478-7.

Appendix A

Contents of the included storage media

The root folder on the SD card contains:

- `experiments_project/`: scripts and source code for the experiments
- `DP.pdf`: this thesis in `.pdf` format
- `thesis_src/`: source code for this thesis

Appendix B

Manual

All source codes for the experiments are located in the `experiments_project/` folder on the media. All experiments and scripts are supposed to be executed from the root folder `experiments_project/` and all paths are relative to `experiments_project/`. Almost all experiments were performed on the SGE cluster at BUT FIT. `python>=3.6` and `CUDA>=10.2` are required.

It is recommended to create `python` virtual environment[1] and it is required for running the script on the SGE cluster. The list of `python` modules for the experiments is included in `requirements.txt` and they can be installed in the current environment as:

```
python -m pip install -r requirements.txt.
```

The `ReadMe.md` in the root folder describes detailed parameters of the scripts for experiments and references of the used parts of the code.

Folder `experiment_configs/` contains `.yaml` configurations of most experiments performed in this thesis. `experiment_configs/ReadMe.md` describes in detail the parameters of configuration files.

An experiment can be executed using the train script `train.py`:

```
python train.py experiment_configs/config.yaml
```

This will store model weights and optimizer state in the experiment folder after each epoch. Experiments can be re-executed with the same script and parameters from the last saved weights and state. The validation and evaluation results are then saved in `[experiment_folder]/val_eer.txt` and `[experiment_folder]/eval_eer.txt`. Soft scores are available in `[experiment_folder]/results/`.

The evaluation and validation are enabled in all configuration scripts by default, however, it can be performed separately or different kind of evaluation or validation can be performed such as TTA if default evaluation was FT.

An example of using `eval.py` script is following. The output is then directed to `stdout` and soft scores are available in `[experiment_folder]/tta_res/results/`:

```
python eval.py experiment_configs/config.yaml --eval tta --save tta_res
```

A fusion can be performed using the `fuse.py` script that takes two files with soft scores as arguments and directs fusion EER to `stdout` for instance:

[1] https://docs.python.org/3/tutorial/venv.html

```
python fuse.py exp_1/results/24_eval.txt exp_2/results/24_eval.txt
```

If an experiment is to be submitted to the SGE cluster at BUT FIT, it is possible to create execution script by editing provided „template“ script with full training command for training `sge_train_example.sh` and evaluation `sge_eval_example.sh` or using `sge_train.sh` or `sge_eval.sh` that take same arguments as `train.py` and `eval.py`. Note that absolute path to the virtual environment has to be manually edited in these scripts. The script can be submitted as: `qsub [script]`.