Weakly-Supervised Speaker Presence Detection on Podcast Episodes

Hollmann, M.

Weakly-Supervised Speaker Presence Detection on Podcast Episodes

Max Hollmann (s1895621)

Thesis advisor: Dr. Wojtek Kowalczyk

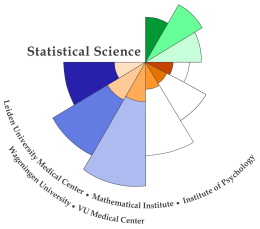
Co-supervisor: Prof. Dr. Peter Grunwald

MASTER THESIS

Defended on November 28, 2018

Specialization: Data Science

Universiteit Leiden



STATISTICAL SCIENCE
FOR THE LIFE AND BEHAVIOURAL SCIENCES

Contents

Chapter 1

Introduction

As a relatively new media format, the infrastructure for cataloging and indexing podcasts is relatively undeveloped. Whereas for movies there is the Internet Movie Database[1] (IMDb) that catalogs movies by who is involved in any way, from directing to composing the soundtrack, no such database exists for podcasts. Interviews are one of the most common formats for podcasts, in which usually one or two hosts speak to one or a few different guests each episode. A catalog that can find all episodes where a particular person was interviewed (like the IMDb providing a list of movies in which an actor has starred) might therefore be of interest to many listeners. The lack of such a directory was the inspiration and motivation behind this work.

The fact that no such directory exists is not surprising since a podcast's metadata (its RSS feed) does not provide information about the people appearing on an episode in a way that could easily be digested by a program. In most instances where a person is being interviewed on an episode, this person will be mentioned in the title or description of the episode, though in a way intended to be understood by humans, not computers (e.g., "In this episode, we talk to..."). Furthermore, a person's name being mentioned in an episode's title or description does not guarantee them appearing on the episode, as in many cases the text might merely indicate that the episode talks *about* them in some way (e.g., "In this episode, we talk about the new book by...").

We present an algorithm that aims to predict with a high degree of accuracy which of the people that are mentioned in an episode's title or description are actually present as speakers on the episode. In order to make it practically feasible to apply this algorithm to the vast number of podcast episodes and speakers in existence, we set the constraint that the algorithm must work without prior knowledge about speakers' voices. This constraint is necessary because training a model to recognize the voice of each individual person that was ever a guest on a podcast would require manually finding examples of their voice, a task that is practically impossible to achieve for the tens or hundreds of thousands of people that have appeared on podcasts.

[1] https://www.imdb.com

In order to make accurate predictions while observing this constraint, our algorithm combines two sources of information that can be obtained without human supervision: the names of people mentioned in an episode's title or description and the characteristics of the voices present on an episode's audio. Neither of these sources alone is enough to determine who is present on an episode – a person might be mentioned for a number of different reasons, and the characteristics of a voice tell us nothing about the identity of the speaker. In combination, however, they are much more informative: If a voice with very similar characteristics occurs on many of the episodes where a certain person is mentioned, it is likely that the voice belongs to this person, and that the person is speaking only on those episodes where these voice characteristics are found.

In the following chapters, we detail the implementation of this algorithm and its components. We evaluate its performance on a large dataset of podcast episodes obtained from the iTunes podcast directory[2], by applying it to a subset of episodes for which we manually label speaker presence. We show that the algorithm can achieve a high level of accuracy under certain conditions and discuss under which circumstances performance degrades. Furthermore, we discuss how due to its modular design, the algorithm can be applied to many other domains by simply substituting certain components.

This work builds on research from various fields of speech processing, such as speaker recognition [1] ("Out of this known set of speakers, whose voice is this?"), speaker verification [2] ("Are these two utterances from the same speaker?"), and speaker diarization [3] ("In this recording of multiple unknown speakers, how many are speaking, and when?"). While it utilizes techniques from all these fields, however, it cannot be appropriately assigned to any one of them. Instead it is, to the best of our knowledge, a novel application of the ideas from these fields.

The structure of this work is as follows: Chapter 2 introduces key concepts that are used throughout this work, Chapter 3 summarizes the results of recent research on methods that we employ, Chapter 4 provides an overview of the various components of the algorithm, Chapter 5 details the implementation and results of the speaker embedding network, Chapter 6 describes the algorithm that brings all components together and reports the results that we obtain, and Chapter 7 examines the results from Chapters 5 and 6 in closer detail and outlines possible directions of future research.

[2] https://itunes.apple.com/us/genre/podcasts/id26?mt=2

Chapter 2

Preliminaries

Though it is impossible to provide comprehensive definitions of all concepts used in this work, we will give a brief description of each of the concepts that are necessary to understand the following chapters.

2.1 Signal Processing

Signal All speech processing begins with recordings of voices. In order to transfer a recording from an analog source such as a microphone or a tape to a computer for further processing, it needs to be *digitized*. Digitization is performed by taking samples of the amplitude of the analog waveform at uniform intervals many times per second. The resulting stream of numbers closely approximates the original waveform and is called a *signal*. The number of samples per second is referred to as the *sampling rating*, a common value being 44 100 Hz. For multi-channel recordings, the signal consists of an equal number of streams (e.g., two streams for a stereo recording).

Fast Fourier Transform A signal can be hard to process directly by speech processing algorithms for various reasons. Therefore, virtually all speech processing systems rely on a higher-order representation of the signal produced by a *fast Fourier transform* (FFT) algorithm. A signal that does not change over time can be approximated by a composition of sine waves of varying amplitudes at different frequencies. FFT reverses this composition and determines the amplitude of the sine waves with a given number of different frequencies, i.e., the power spectrum of the signal. Since a voice recording changes significantly over time, FFT is usually applied to very short frames extracted from the signal, which can be considered as unchanging. These frames are extracted from the signal at uniform intervals (*stride*) and have a fixed *length*. This results in a spectrogram, a three-dimensional representation of the signal with the axes representing time, frequency, and the power of a specific frequency present at a particular time. Spectrograms are commonly visualized as heatmaps with time and frequency as the X and Y axes, and power as the color of a coordinate. The resolution of the spectrogram is determined by

the length and stride of the frames. An introduction to common FFT algorithms can be found in [4].

Mel Scale Human hearing responds to frequencies in a non-linear way – the difference between 100 Hz and 110 Hz is perceived as larger than that between 1000 Hz and 1010 Hz. For this reason, [5] devised the *Mel scale*, which maps a subjective pitch unit called *mel* to frequency measured in Hertz (Hz) in a way that two tones with, for example, a difference of 10 mels are perceived as equally far removed in pitch, no matter where on the frequency scale these points lie. This scale is frequently used in speech processing, as human hearing has evolved at least partially to optimize understanding of human speech.

Mel-Filterbank Energy Mel-filterbank energy (MFE) is a combination of the Mel scale with FFT. A Mel-filterbank consists of a number of triangular filters which are spaced uniformly on the Mel-scale. Filters have a response of 0 at the edges and 1 in the center and overlap with their neighboring filters by half their width. The MFE is computed by applying this filterbank to the output of an FFT.

Mel-Frequency Cepstral Coefficients An MFE spectrogram usually exhibits a very high degree of autocorrelation, which is a problem for many machine learning algorithms. A common solution for this problem is the conversion of MFEs into *Mel-frequency cepstral coefficients* (MFCCs) [6]. MFCCs are derived by applying a discrete cosine transform to the logs of the MFEs of a frame. This results in a representation of the signal which is both lower-dimensional than the MFE spectrogram, as well as decorrelated.

2.2 Speech Processing

Speaker Recognition Speaker recognition is the task of recognizing the speaker of an utterance out of a known set of people. To achieve this, sample utterances of each person are compared in some way to the utterance in question, and a prediction is made based on similarity (see [1] for an overview).

Speaker Verification Speaker verification is closely related to speaker recognition but deals with only a single person. The question in this task is whether an utterance (the *test utterance*) is from a certain person or not. To this end, the test utterance is compared to an *enrollment utterance*, which is known to be from this person. Research commonly distinguishes text-dependent speaker verification, in which enrollment and test utterances contain the same phrase, from text-independent speaker verification. See [2] for an overview of this field.

Speaker Identification Since many of the same techniques to compare utterances can be used in both speaker recognition and verification tasks, the research on these fields is to a large

part overlapping. The two tasks are therefore sometimes grouped together and jointly referred to as speaker identification.

Equal Error Rate The *equal error rate* (EER) is a common performance measure for speaker verification algorithms. Speaker verification usually entails a decision based on a threshold, where two utterances are classified as the same speaker if their distance is lower than the threshold and as not the same speaker if their distance is higher. When this threshold is lowered, the false positive rate declines, but the false negative rate increases, and vice versa when the threshold is increased. At some threshold, the false negative rate is equal to the false positive rate, at which point the error rates are referred to as the EER.

2.3 Machine Learning

Machine learning is a broad field of methods, most of which provide a way to make predictions about an outcome variable based on one or a number of predictors. Models are trained how to make predictions based on a *training set*, a dataset for which both predictors and outcome are known. In mathematical notation, predictor variables are commonly jointly referred to as $\mathbf{X}$ and the outcome as $\mathbf{Y}$. In these terms, the model attempts to create a mapping $f(\mathbf{X}) = \mathbf{Y}$. We employ two different methods in this work, which we describe in the following.

Neural Networks As the name implies, neural networks are loosely based on the working mechanism of the brain. A neural network in its simplest form (called a *feed-forward* neural network) consists of at least two layers, the input layer and the output layer, with the option to add multiple *hidden layers* in between. Each layer is made up of a number of *nodes* (corresponding to neurons in the brain analogy). Nodes are represented simply as numbers that indicate their *activation*. The activations of the input layer are defined through the predictors, i.e., they are equal to $\mathbf{X}$. Each subsequent layer is connected to their predecessor through connections, each of which has a *weight*. The activation of a neuron is the sum of its connected neurons' activations, each multiplied by its respective connection's weight. In a *fully connected layer*, each node is connected to each node of the previous layer. Activations are propagated forward until they reach the output layer, whose activations constitute the prediction for the given input.

In order to train a neural network, the activations of the output layer $\hat{\mathbf{Y}}$ for input $\mathbf{X}$ are compared to the desired output $\mathbf{Y}$ by a *loss function* $L(\mathbf{Y}, \hat{\mathbf{Y}})$. This function is designed in a way that it returns high values for predictions $\hat{\mathbf{Y}}$ that are very different from $\mathbf{Y}$, and lower values the closer the prediction matches the desired output. The network is trained by updating its weights in the opposite direction of the gradient of the loss function. In this way, each layer gradually learns to use the activations of its previous layer to produce activations representing features of the input which are useful to the following layer for making accurate predictions.

Convolutional Networks Convolutional networks are networks that include convolutional layers, and are frequently used for inputs with a spatial structure, such as images. In contrast

to fully connected layers, nodes in a convolutional layer do not have a weight for each of the previous layer's nodes. Instead, a convolutional layer uses a relatively small set of weights to generate activations from spatially constrained areas of the nodes of the previous layer. The same weights are used for all areas of the input. The resulting activations are referred to as *feature maps* since the weights learn to recognize certain features in an area, and the result is a map of the presence of these features throughout the spatial structure of the input.

Deep Learning Deep learning is simply the term commonly used for networks with a large number of hidden layers. Training of such networks has only relatively recently become feasible, due to innovations in network architectures and training methodologies, decreasing cost of computation, and availability of large amounts of training data. The use of deep neural networks (DNNs), and in particular deep convolutional networks (DCNs), have resulted in breakthroughs in many tasks where input data contains information encoded in very complex structures, such as images and audio. Due to this complexity, hand-crafted feature extraction has had limited success in these domains, while DNNs can automatically learn to represent increasingly higher-order features throughout the layers. [7] provide an overview of the innovations in this field.

Embedding An embedding is a numeric representation of some entity, usually with the property that the distances between embeddings in the embedding space are reflective of the semantic similarity of their respective entities. Speaker identification methods frequently use speaker embeddings generated from utterances to evaluate the probability that two utterances are from the same speaker.

Triplet Loss The triplet loss is a loss function introduced by [8] to train neural networks to produce embeddings. It derives its name from operating on triplets of network outputs. Each triplet consists of an anchor (A), a positive example (P), and a negative example (N). In the context of speaker identification, A and P could be embeddings of utterances from the same speaker, and N an embedding from a different speaker. The triplet loss attempts to achieve embeddings where the distance between A and P is smaller than the distance between A and N. We describe this loss function in more detail in Section 5.1.3.

Logistic Regression Logistic regression is a relatively simple model for classification tasks where the outcome variable has only two possible classes, frequently referred to as the positive and the negative class. This method works by finding the coefficient vector $\mathbf{b}$ that best fits the predictors $\mathbf{X}$ to the logit of the probability of the outcome being the positive class, i.e. $\text{logit}(\hat{p}) = \mathbf{Xb}$, where $\text{logit}(p) = \ln(\frac{p}{1-p})$. Classifications are made by predicting the positive class when $\hat{p}$ is greater than some threshold, and the negative class otherwise.

Chapter 3

Related Research

As mentioned in Chapter 1, our algorithm does not fall within the bounds of any single established field of research, but it is utilizing techniques from various fields in the area of speech processing. Some of the core components of the algorithm stem from speaker recognition and verification research.

For more than a decade, most state-of-the-art speaker identification methods relied in some way on Gaussian Mixture Models (GMMs) fitted on MFCC features extracted from the audio signal [1]. One of the most successful methods employing this approach was introduced by [9]. They describe a method to generate a so-called i-vector (short for identification-vector) for an utterance. This vector is derived by concatenating the mean of each component of a GMM trained on the MFCCs of the utterance into a supervector, which is then decomposed into speaker- and channel-dependent components through joint factor analysis. The speaker-dependent component varies mostly with speaker identity and is therefore used as the i-vector. In a speaker identification task, distances between i-vectors of utterances can be used to judge the similarity of the voices.

Recently, deep neural networks (DNNs) have proven to perform better than conventional approaches in many areas, including speech recognition [7]. As such, it was an obvious candidate to be applied to speaker identification tasks as well, and indeed, much research has found deep learning approaches to result in superior performance compared to GMM based methods.

A very straightforward approach to generating embeddings has been proposed by [10]. They trained a DNN to identify a set of speakers based on log filterbank energy features extracted from utterances and used the L2 normalized activations of the last hidden layer of the network as speaker embeddings. These embeddings were used in a text-dependent speaker verification task by comparing the embeddings of the enrollment and the test utterance via their cosine distance. They compared the performance to that of a baseline i-vector approach and found that the DNN performed slightly worse under normal conditions. However, when they added background noise to the utterances, the performance of the DNN approach degraded less than that of the i-vector

approach, indicating better robustness to noisy recording conditions.

Similar to [10], [11] first trained a convolutional deep convolutional network (DCN) with an architecture adapted from ResNet [12] on a speaker recognition task, using MFE features as inputs. They then took the training a step further by removing the prediction softmax layer and refining the embeddings generated by the previous layer through training using the triplet loss. In a text-independent speaker recognition task, they found the embeddings generated by this network to result in a 13.7 % relative reduction in EER compared to a baseline i-vector approach. In a text-dependent speaker recognition task, on the other hand, the DNN's EER was 17.1 % higher than that of the i-vector approach.

A similar network architecture, as well as training strategy, was employed by [13]. They trained a DCN with the 50-layer ResNet [12] architecture using the contrastive loss function [14]. This loss function is similar to the triplet loss in that it operates on distances between embeddings. It takes a pair of embeddings as input and produces a loss that encourages same-speaker pairs to have a small distance, and different-speaker pairs to have a larger distance. They trained the network on the VoxCeleb2 dataset, which they introduced in the same paper, and evaluated its performance on speaker pairs from VoxCeleb1 [15], on which it achieved an EER of 4.42 %. This result serves as a useful comparison for our network (see Chapter 5), as we use the same training and test data.

Chapter 4

Approach

In order to build an intuition about how our algorithm works, we will start by describing how a human might approach the problem outlined in Chapter 1. Let's assume that for any person, we can easily find all episodes that mention their name in the title or description (as is the case in our dataset, see Section 6.2). We will call this person a *speaker candidate* for the episodes where he or she is mentioned. To find out on which of these episodes this person is actually speaking, one might listen to all these episodes and see if there is a voice that one recognizes on multiple episodes. If those episodes have nothing else in common, such as being from the same podcast (in which case the voice might be that of the host), it is likely that this voice belongs to the person in question. One might also have mistaken different people with very similar voices to be the same person, so there is some uncertainty inherent in this approach.

Our computational approach works along the same lines. Of course, this requires a few abilities that come naturally to humans but are non-trivial to perform computationally. Comparing voices of speakers from multiple episodes requires two abilities. First, to figure out how many speakers are speaking on an episode, and which parts of the audio signal originate from which speaker. This problem is called speaker diarization, and we describe our approach to it in Section 4.2. Second, to generate a speaker embedding from the audio signal so that voices can be compared quantitatively. We introduce two options to achieve this in Section 4.1 and detail their implementation and performance in Chapter 5. Finally, these components need to be employed to judge the likelihood for each episode of a speaker candidate that he or she is actually speaking on the episode. Chapter 6 deals with our approach to this final part of the problem, and Section 4.3 provides a summary.

4.1 Speaker Embeddings

The first major requirement is the ability to extract speaker embeddings – feature vectors that represent a speaker's voice numerically (see Section 2.3) – from an audio signal. Since podcast episodes vary in terms of audio quality, acoustic environment, recording equipment, and trans-

mission channel, the method used should ideally generate embeddings that are robust to these factors and only vary with speaker identity. Various methods that attempt to satisfy this criterion have been developed for use in speaker identification, verification, and diarization tasks. Prior research suggests that DNN generated speaker embeddings are currently the best-performing option available – especially under noisy conditions and for text-independent comparisons (see Chapter 3). We therefore test the following two methods of obtaining embeddings in this way.

4.1.1 Intermediate Layer of a Neural Network

[10] and [16] describe a method in which a neural network is first trained to recognize a set of speakers from their utterances, and then stripped of its final (prediction) layer. To provide useful input to the prediction layer, the previous layer should have learned to detect various features that characterize a person's voice. Therefore, the outputs of this layer can be used as a representation of a person's voice.

4.1.2 Neural Networks Trained Using the Triplet Loss

A more explicit way to train a neural network to generate embeddings that vary more with speaker identity than with other factors is to use the triplet loss introduced by [8]. This loss function has been used by [11] to train a network to generate speaker embeddings, with very promising results. Due to pretraining on a speaker recognition task (described in Section 5.1.4) we can obtain intermediate layer embeddings as a side-effect. Chapter 5 describes the implementation and training of such a network in detail.

4.2 Speaker Diarization

Given a method of extracting speaker embeddings from audio, we could simply split an episode into short chunks and extract an embedding from each chunk. This would yield a large number of embeddings for each speaker on the episode, as well as some which originate from chunks where multiple speakers are present. These factors might negatively impact runtime performance as well as accuracy of the scoring stage (Section 4.3).

Therefore, in an ideal case, we would extract just a single embedding per speaker. This requires a way of detecting the number of distinct speakers on an episode as well as the segments where each of them speaks, which is a problem referred to in the literature as speaker diarization [3]. We use the LIUM Speaker Diarization Toolkit for this task.

4.2.1 LIUM Speaker Diarization Toolkit

LIUM_SpkDiarization [17] is an open source speaker diarization toolkit. It performs speaker segmentation using the generalized likelihood ratio (GLR) computed within a window moving over the MFCCs of the audio signal. The GLR is used to compare a model using a single multivariate normal distribution fitted on the MFCCs within the window with another model

which uses two distributions, one for each half of the window. When a local maximum of the GLR is detected, the midpoint of the window at this point is used as a segment boundary. In a second step, speakers are merged using hierarchical agglomerative clustering using the same model comparison as a stopping criterion (see [18] for details about this procedure).

In preliminary experiments on manually diarized episodes, we determined that this method generally results in accurate segmentation between speakers, but is conservative in the clustering stage. This means that the diarization detects more speakers than are actually present, and each real speaker's utterances get distributed over multiple detected speakers. However, in most cases, all of a detected speaker's utterances belong to a single real speaker. Due to the way we use the embeddings of the detected speakers (see Chapter 6), this is not a big problem and certainly preferable to the opposite case.

4.3 Speaker Presence Detection

In the final stage of the algorithm, the speaker embeddings extracted from the diarized episodes are used to evaluate the likelihood for each episode that the speaker candidate is actually speaking on it. For this task we employ a logistic regression model, using as predictor a score that utilizes differences between the distributions of distances between embeddings from the same speaker and those of different speakers to capture information about the speaker candidate's presence on episodes. We describe this procedure in detail in Chapter 6.

Chapter 5

Speaker Embedding Network

5.1 Details

We replicate the network architecture and most of the training strategy of [11], who employ an adaptation of the ResNet [12] architecture trained using the triplet loss to generate speaker embeddings.

5.1.1 Architecture

The network architecture is an adaptation from ResNet [12]. It consists of four Conv+Res components (depicted in Figure 5.2), which are made of a convolutional layer followed by three ResBlocks. Each of these convolutional layers has a kernel size of 5×5, a padding of 2 in both frequency and temporal dimension, and a stride of 2 in both dimensions. A ResBlock (introduced by [12], depicted in Figure 5.3), consists of a convolutional layer with a kernel size of 3×3 and a padding of 1 in both dimensions, followed by the ReLU activation function and a second convolutional layer with the same configuration. The inputs to the first layer are added to the output of the second layer, and the result is passed through the ReLU function. The purpose of this arrangement is that the two convolutional layers merely need to learn to *fine-tune* the ResBlock's input, thus allowing the construction of a deeper network without making it harder to train. [12] have demonstrated the effectiveness of this approach.

As shown in Figure 5.1, the number of filters of the initial convolutional layer of the Conv+Res components start at 64 filters and is doubled in each consecutive component, resulting in 512 filters in the last one. This last component is followed by a mean-pooling layer that averages outputs over the temporal dimension, making the network agnostic to input duration. The next layer is a fully connected layer with 512 nodes, optionally followed by a length normalization function (for reasons described in Section 5.1.3.2). The outputs at this point are used as speaker embeddings.

During pretraining (see Section 5.1.4), the embedding layer is passed through the ReLU function

and feeds into a softmax prediction layer. In this case, no length normalization is performed.

We use the ReLU activation function after each convolutional layer throughout the network, and add a Batch Normalization layer [19] before each ReLU function. The complete architecture is provided in Table 5.1.

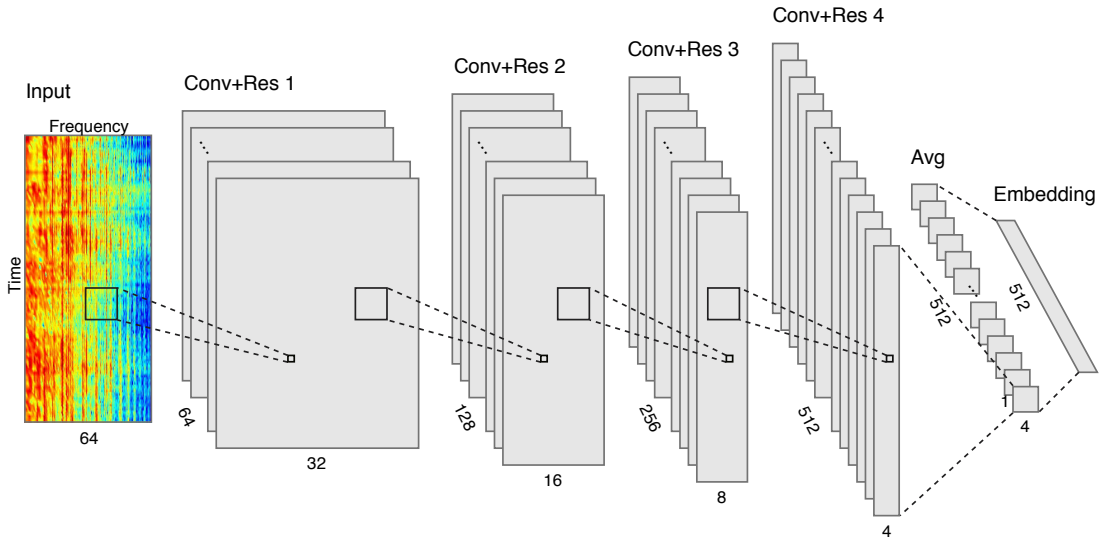


Figure 5.1: Architecture of the speaker embedding network.

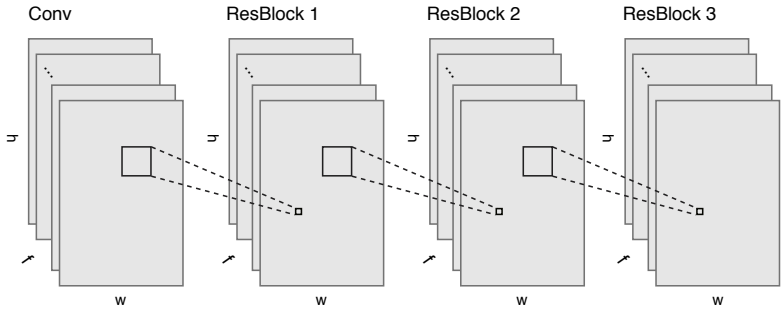


Figure 5.2: Architecture of a Conv+Res component.

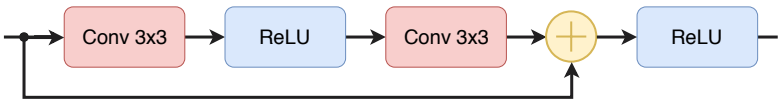


Figure 5.3: Layout of a ResBlock. The convolutional layers use the same number of filters as the convolutional layer preceding the ResBlock and a stride of 1, thus keeping the dimensionality of the output constant.

Layer type	Specification	Stride	Output dim.
Input			300×64
Convolutional	$5 \times 5, 64$	2×2	$150 \times 32 \times 64$
Conv. (ResBlock)	$3 \times \begin{cases} 3 \times 3, 64 \\ 3 \times 3, 64 \end{cases}$	1×1	$150 \times 32 \times 64$
Convolutional	$5 \times 5, 128$	2×2	$75 \times 16 \times 128$
Conv. (ResBlock)	$3 \times \begin{cases} 3 \times 3, 128 \\ 3 \times 3, 128 \end{cases}$	1×1	$75 \times 16 \times 128$
Convolutional	$5 \times 5, 256$	2×2	$38 \times 8 \times 256$
Conv. (ResBlock)	$3 \times \begin{cases} 3 \times 3, 256 \\ 3 \times 3, 256 \end{cases}$	1×1	$38 \times 8 \times 256$
Convolutional	$5 \times 5, 512$	2×2	$19 \times 4 \times 512$
Conv. (ResBlock)	$3 \times \begin{cases} 3 \times 3, 512 \\ 3 \times 3, 512 \end{cases}$	1×1	$19 \times 4 \times 512$
Average	over temporal dim.		4×512
Fully connected	512 units		512
Length norm.	Optional		512
Fully connected	5990 units softmax activation only for pretraining		5990

Table 5.1: Architecture of the speaker embedding network. Convolutional layers are specified as "kernel size, #filters". Consecutive ResBlocks are summarized for brevity.

5.1.2 Input Features

As inputs for the network, we extract Mel-filterbank energy features from the audio signal. Multi-channel signals are converted to mono by taking the mean across channels at each time point. We use 64 filters between 50 Hz and 5000 Hz for the Mel-filterbank and extract energies from frames of 20 ms duration with a stride of 10 ms. Thus, we obtain a matrix of 300×64 values per 3 s utterance. The energies are scaled to a mean of 0 and a variance of 1 within each frequency band of each utterance. We use the SpeechPy [20] Python package to obtain these features.

5.1.3 Triplet Loss

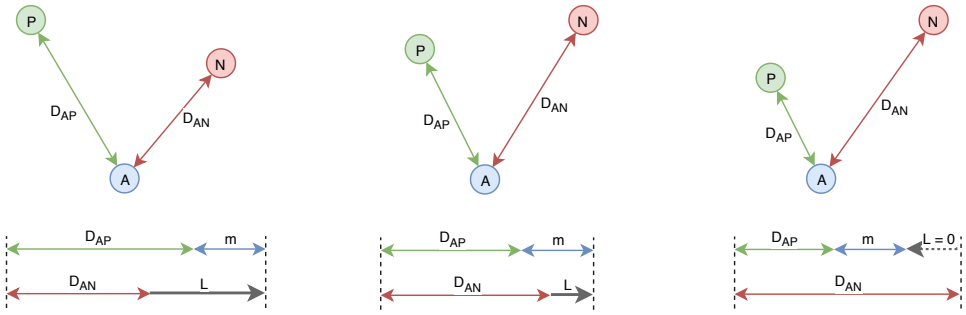
The triplet loss [8] operates on triplets of embeddings $(\mathbf{e}^A, \mathbf{e}^P, \mathbf{e}^N)$, originating from an anchor A, a positive example P, and a negative example N. The anchor and positive examples are two different utterances by the same speaker, and the negative example is an utterance of a different speaker. The loss function

$$L = \frac{1}{N} \sum_{i=1}^{N} \Big[\mathrm{D}(\mathbf{e}_i^A, \mathbf{e}_i^P) - \mathrm{D}(\mathbf{e}_i^A, \mathbf{e}_i^N) + m \Big]^+ \tag{5.1}$$

where $[.]^+$ denotes $\max(0, .)$, aims to make the distance between the anchor and the positive example $\mathrm{D}(\mathbf{e}^A, \mathbf{e}^P)$ smaller than the distance between the anchor and the negative example $\mathrm{D}(\mathbf{e}^A, \mathbf{e}^N)$ by at least a margin m (see Figure 5.4 for an illustration). Formally, it attempts to get all triplets to conform to the criterion

$$\mathrm{D}(\mathbf{e}_i^A, \mathbf{e}_i^P) + m \leq \mathrm{D}(\mathbf{e}_i^A, \mathbf{e}_i^N). \tag{5.2}$$

This results in embeddings where utterances of the same speaker are mapped closely together, and utterances of different speakers are mapped further apart, which is a very desirable property for the purpose of our algorithm, since these embeddings will lend themselves to further processing steps such as clustering (as [21] demonstrate for the subject of faces).



(a) The negative example is closer than the positive example to the anchor, making the loss for this triplet larger than m.

(b) The positive example is now closer than the negative example to the anchor, but not yet by at least m, thus decreasing the loss to a value of less than m, but still greater than 0.

(c) Now the distance between the positive example and the anchor is smaller than that of the negative by more than m, so the loss becomes 0 for this triplet.

Figure 5.4: Illustration of a triplet in three stages of learning.

5.1.3.1 Triplet Mining

Since the components of the loss function for triplets where the criterion in Equation (5.2) is satisfied are 0, the gradient of the total loss L will get flatter the better the network becomes, thus slowing down learning. To combat this, various schemes have been proposed that generate triplets in a way that only triplets not satisfying Equation (5.2) are used in training. These procedures are called triplet mining.

One can differentiate between two basic categories of these procedures: Offline and online triplet mining. In offline mining methods, the network training is interrupted periodically, and the latest network is used to calculate the embeddings of the training data, which are then assembled into triplets for the following training iteration. Examples of this method are [22] and [11]. In online triplet mining, triplets are generated on-the-fly from the embeddings in the current batch, thus always operating on embeddings generated using the latest network weights. Since online triplet

mining methods require considerably less computing overhead and have proven more effective than offline methods [23], we focus solely on online methods in this paper.

Since each triplet requires two different examples of the same speaker, it is important to ensure that each speaker that is present within a batch is represented by multiple utterances. To this end, a batch can be constructed by sampling P people, and from each person U utterances, resulting in a batch-size of $B = P \times U$. Within such a batch, a total number of $PU(PU - U)(U - 1)$ triplets can be generated. We will call this set of all possible triplets within a batch $\mathcal{T}_B$.

[23] introduce two online triplet mining methods – Batch All and Batch Hard. Batch All simply utilizes all triplets in $\mathcal{T}_B$ that do not satisfy Equation (5.2). In Batch Hard, each embedding in the batch is used as the anchor example once and is matched with the positive and negative examples that maximize the loss for this particular triplet. We use a Python implementation of the Batch All and Batch Hard methods from [24].

5.1.3.2 Length Normalization

It is common practice to normalize the embeddings to a length of 1, thus constraining them to the surface of the unit hypersphere [11, 21, 25, 26]. Since only the distance between embeddings is considered in the loss function, unnormalized embeddings are translation invariant and therefore allow the network to scale the embeddings up infinitely, which can lead to NaNs in the weights and output. Length normalization is also required for a regularization that we describe in the next section. We run experiments both with and without a length normalization layer appended after the embedding layer, and use the cosine distance in the triplet loss when length normalization is applied.

5.1.3.3 Regularization

Various regularization terms have been proposed that can be added to the triplet loss (e.g. [25, 26]). We experiment with the global orthogonal regularization described by [26]. They show that the expected value of the cosine similarity of two points sampled uniformly from the surface of a d-dimensional hypersphere is 0, and the variance of this value is $\frac{1}{d}$. Therefore, if the embeddings generated by the network with length normalization are well spread-out in the embedding space, most pairs of embeddings from utterances of different speakers should have cosine similarities close to 0. The global orthogonal regularization encourages this property by adding the term

$$L_{gor} = M_1^2 + \left[M_2 - \frac{1}{d}\right]^+ \tag{5.3}$$

to the triplet loss, where $[.]^+$ denotes $\max(0, .)$,

$$M_1 = \frac{1}{N}\sum_{i=1}^{N}\langle \mathbf{e}_i^A\rangle^\intercal \langle \mathbf{e}_i^N\rangle \tag{5.4}$$

and

$$M_2 = \frac{1}{N}\sum_{i=1}^{N}\left(\langle \mathbf{e}_i^A\rangle^\intercal \langle \mathbf{e}_i^N\rangle\right)^2 \tag{5.5}$$

are the sample first and second moments of the cosine similarities between anchors and negatives in $\mathcal{T}_B$ (we use $\langle \mathbf{v}\rangle$ to denote the unit vector in the direction of $\mathbf{v}$; $\langle \mathbf{v}\rangle = \frac{\mathbf{v}}{\|\mathbf{v}\|}$).

5.1.4 Pretraining

We experiment with pretraining the network by appending a fully connected layer with a softmax activation function and training it to identify speakers from their utterances. For triplet training, the prediction layer is removed, and optionally the length normalization layer appended. This procedure has been shown to improve the performance of the final triplet-trained network [11, 23].

Since the fully connected layer prior to the prediction layer needs to learn representations that help the prediction layer to discriminate between speakers, pretraining also allows us to obtain the intermediary layer embeddings described in Section 4.1.1 for comparison to embeddings generated using the triplet-trained network.

5.1.5 Single Layer Training

If the pretraining led to a relatively high accuracy, this shows that the convolutional layers of the network have learned representations that are useful to the speaker identification task. Since these same representations are likely just as useful for the task of separating speakers in the triplet training stage, it might not be necessary to train these layers any further. We therefore experiment with *single layer training* (SLT) in the triplet stage, meaning that we apply the backpropagation only to the embedding layer, leaving the rest of the weights in their pretrained state. We anticipate that this will lead to faster convergence, as optimizing a single layer drastically reduces the size of the search space and cost of gradient computation.

5.2 Data and Evaluation

5.2.1 Dataset

We use the VoxCeleb2 dataset [13] for the training of the neural network. This dataset consists of a total of 1 128 246 utterances by 6112 distinct speakers that were extracted from 150 480 YouTube videos.

[13] compiled the dataset automatically by searching YouTube for the names of people contained in the VGGFace2 dataset [27] appended by the word "interview" and downloading the first 100 resulting videos. In each video, a convolutional neural network trained to recognize the identities from VGGFace2 is used to identify the frames where the person is visible. Finally, SyncNet [28] is used to estimate in which frames the person is audibly speaking by correlating the person's mouth movement with the video's audio track.

This method of constructing the dataset has the consequence that the utterances stem from varying recording conditions, acoustic environments, and microphones. We anticipate that this variance in the training data will lead the network to generalize well over the different recording conditions of podcast episodes.

As [13] only provide links to the videos and timestamps for the segments where each person is speaking, we use the Luigi job orchestration framework [29] to download the videos and extract the utterances of each speaker[1]. 9187 of the videos were not available on YouTube at the time of download, reducing by a small percentage the total number of utterances, but not speakers, present in the dataset.

Since we intend to employ the network on 3 s segments of audio, we also cut the utterances from VoxCeleb2 into 3 s long segments. This increases the number of utterances to 2 214 804. Table 5.2 summarizes the statistics of the resulting dataset.

	# or Mean	SD	Min	Max
People	6112			
Videos	141 293			
Utterances	2 214 804			
Videos per person	23.25	14.22	1	91
Utterances per video	15.68	25.70	1	719
Utterances per person	362.37	344.51	5	2192

Table 5.2: Statistics of VoxCeleb2 after splitting into 3 s segments

5.2.2 Data Splits

To ensure that none of the utterances used for validation have been seen by the network during either pretraining or triplet training, and because the two stages of training require splits fulfilling different criteria, we create two different validation sets. In the following, we define $\mathcal{D} = \{\mathbf{x}_{p,v,u}\}$ as the complete dataset, where p is a person, v a video of person p, and u an utterance from video v.

Triplet training validation set In order to evaluate how well the network can separate people not seen during training, we create splits such that each person is present in either the

[1] We published the Luigi pipeline to download and construct the VoxCeleb and VoxCeleb2 datasets at https://github.com/maxhollmann/voxceleb-luigi

training set or the validation set, but not both. We randomly select 122 people $\mathbf{P}_{val,triplet}$ (2 % of all people in $\mathcal{D}$) to be used in validation, and define the validation set for the triplet training stage as $\mathcal{D}_{val,triplet} = \{\mathbf{x}_{p,v,u} \in \mathcal{D} \mid p \in \mathbf{P}_{val,triplet}\}$.

Pretraining validation set Since multiple utterances can originate from the same video, but each video only contains utterances by a single speaker, the network might overfit to recognize the audio characteristics of a video instead of the voice of a speaker. In order to prevent leakage of this overfitting into the validation set, we ensure that all utterances of any given video are assigned to only one of the splits. Thus, we randomly select 2825 videos $\mathbf{V}_{val,pre}$ (2 % of all videos in $\mathcal{D}$) and use all utterances originating from them as the validation set for the pretraining $\mathcal{D}_{val,pre} = \{\mathbf{x}_{p,v,u} \in \mathcal{D} \mid v \in \mathbf{V}_{val,pre}\}$. To ensure that every person present in $\mathcal{D}_{val,pre}$ is also present in $\mathcal{D}_{train}$, we perform the random selection from the set of videos in $\mathcal{D} \setminus \mathcal{D}_{val,triplet}$ with one video of every person removed.

Training set The training set $\mathcal{D}_{train} = \mathcal{D} \setminus (\mathcal{D}_{val,pre} \cup \mathcal{D}_{val,triplet})$ is composed of all utterances that are not used in either of the two validation sets and is used for both stages of the training. This is to ensure that the pretrained network's weights that are carried over to the triplet training stage are untouched by any data in $\mathcal{D}_{val,triplet}$.

The characteristics of each data split are described in Table 5.3.

	$\mathcal{D}_{train}$	$\mathcal{D}_{val,pre}$	$\mathcal{D}_{val,triplet}$
People	5990	2167	122
Videos	135 744	2825	2800
Utterances	2 124 032	45 004	46 706

Table 5.3: Characteristics of the data splits.

5.2.3 Evaluation

5.2.3.1 Triplet Training

As a measure of performance for the triplet trained network, we employ the equal error rate (EER) of a speaker verification task performed within each batch. We simulate a speaker verification task by considering all possible pairs of $(\mathbf{e}^A, \mathbf{e}^P)$ and $(\mathbf{e}^A, \mathbf{e}^N)$ within $\mathcal{D}_{val,triplet}$. A pair is classified as the same speaker if $\mathrm{D}(\mathbf{e}_1, \mathbf{e}_2) < t$, so we can define the classification function as

$$f_t(\mathbf{e}_1, \mathbf{e}_2) = \begin{cases} 1 & \text{if } \mathrm{D}(\mathbf{e}_1, \mathbf{e}_2) < t \\ 0 & \text{if } \mathrm{D}(\mathbf{e}_1, \mathbf{e}_2) \geq t \end{cases} \tag{5.6}$$

Thus, the error rate of the classification of positive pairs is $E_{AP} = \frac{1}{N_{AP}} \sum_{i=1}^{N_{AP}} 1 - f_t(\mathbf{e}_i^A, \mathbf{e}_i^P)$, and that of negative pairs $E_{AN} = \frac{1}{N_{AN}} \sum_{i=1}^{N_{AN}} f_t(\mathbf{e}_i^A, \mathbf{e}_i^N)$. There exists a value of t at which the error

rates for positive and negative pairs are approximately equal, at which $EER = E_{AP} \approx E_{AN}$.

5.2.3.2 Pretraining

We evaluate models in the pretraining stage by their classification accuracy on the validation set. A high accuracy suggests that the network has learned representations that are useful to discriminate between people's voices, and will thus provide a good starting point for the triplet training stage.

As hinted at in Section 5.1.4, we also evaluate the intermediary layer embeddings obtained from the network during pretraining. To this end, we use the pretrained network without the prediction layer to generate embeddings from $\mathcal{D}_{val,triplet}$ and calculate the EER. Even though the length normalization layer is never used during training in this stage, we evaluate the performance of the intermediary layer embeddings both with and without length normalization applied.

5.2.3.3 Evaluation of the Final Model

From all model configurations that we test, we select the model and epoch with the lowest EER on $\mathcal{D}_{val,triplet}$ for use on the speaker presence detection task. To obtain an unbiased estimate of the performance of this model, we determine its EER on a dataset that was not involved in either training nor model selection. In the paper introducing VoxCeleb2, [13] propose a speaker verification test set called VoxCeleb1-E, consisting of 581 480 pairs of utterances sampled from the smaller VoxCeleb1 [15] dataset. Using this test set allows us to directly compare our model's performance to that of [13], as they report the performance of their model on the same data.

Because the utterances in this dataset vary in duration, we split them into 3 s segments, and use the mean of the embeddings generated from these segments as the embedding for an utterance.

5.3 Experimental Setup

5.3.1 Training Methodology

In both stages, we train the network using distributed synchronous SGD [30] on up to eight NVIDIA Tesla K80 GPUs. Unless explicitly stated otherwise, we use a learning rate of 0.05 and no momentum. For the triplet loss, we always use a margin m of 0.2.

Performance of the model on the validation set is evaluated after every epoch, and training stopped manually once the relevant performance measure stabilizes or declines. We store snapshots of the network's weights after every epoch, enabling us to use the weights from the best-performing epoch for continued training, evaluation, or use in the speaker presence detection task (Chapter 6).

During the triplet training stage, we assemble batches by randomly assigning the people present in the training set into groups of P people. Each group is then formed into a batch by randomly

sampling U utterances for each person in that group. Thus, only a fraction of all utterances is used in each epoch, allowing us to follow the training progress in relatively small increments. We use $P = 5$ and $U = 20$ in all experiments (these parameters are introduced in Section 5.1.3.1).

5.3.2 Experimental Conditions

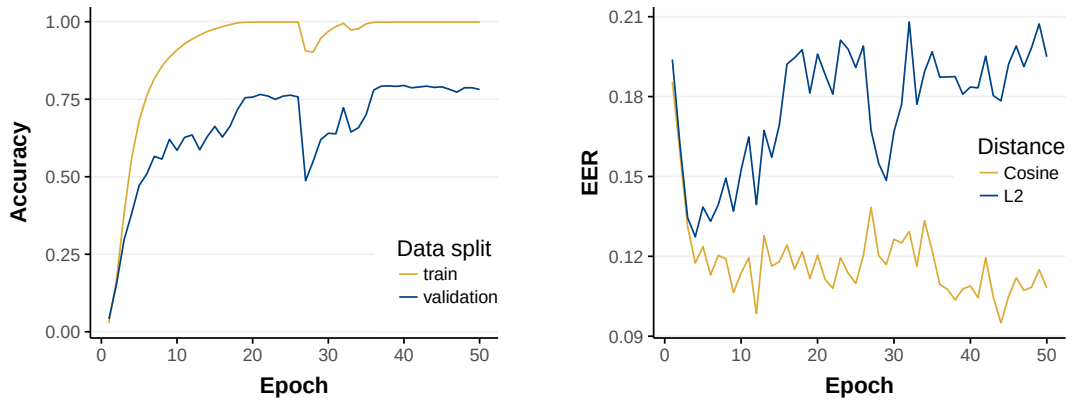
Because the training of the network is computationally intensive, and the elements described in Section 5.1 allow a large number of different configurations of the network and training procedure, we cannot explore all possible conditions. We therefore start with a few simple configurations, and iteratively modify elements of the best-performing models. This allows us to explore the configurations that have a good chance of being successful within an acceptable timeframe.

5.4 Results

5.4.1 Pretraining

The performance measures of the softmax-trained prediction model are shown in Figure 5.5. The model started to significantly overfit after a few epochs (see Figure 5.5a) but nonetheless achieved a top-1 accuracy on the validation set of 79.43 % after 40 epochs, which, on the task of recognizing a person out of a set of 5990 people, is a respectable performance.

The performance of the intermediary layer embeddings generated from this model is displayed in Figure 5.5b. Interestingly, even though the length normalization layer was not used during training, the performance of these embeddings on a speaker verification task is significantly better when the cosine distance is used. The best EER of 9.5 % was achieved after 44 epochs using the cosine distance.



(a) Classification accuracy on the training and validation sets.

(b) EER on the triplet validation set, using Cosine and Euclidean distance.

Figure 5.5: Performance measures of the softmax trained model.

5.4.2 Triplet Training

Unless stated otherwise, we compared embeddings generated by networks trained with length normalization using the cosine distance, and networks trained without length normalization using the Euclidean distance.

5.4.2.1 Mining Method and Length Normalization

As an initial experiment in the triplet training stage, we compared four network configurations: with and without the length normalization layer, and in both cases using the Batch All and the Batch Hard triplet mining strategy. None of the models were pretrained, and none used the global orthogonal regularization (GOR). As Figure 5.6 shows, the models trained using Batch All mining converged a lot faster than those using Batch Hard and achieved better performance. The best EER was achieved by the model without length normalization using the Batch All mining method after 189 epochs, at 10.6 %.

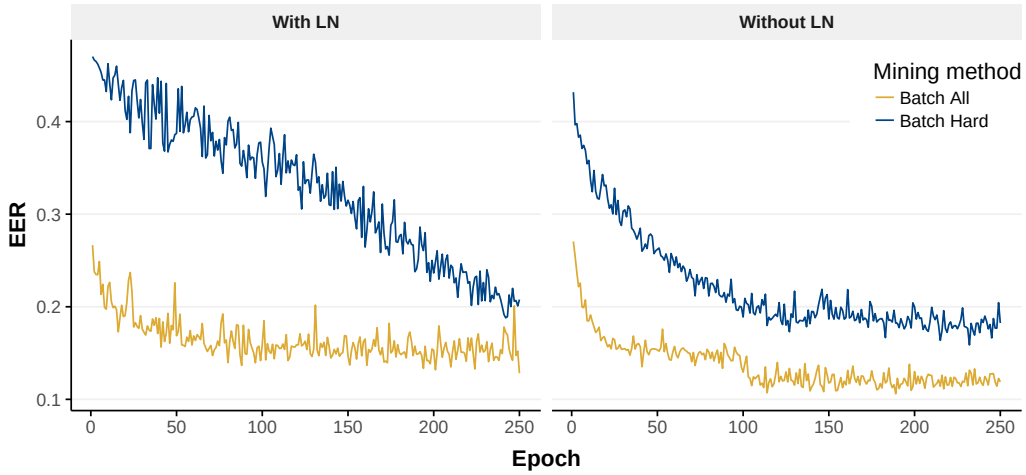


Figure 5.6: Triplet training with and without length normalization, using Batch All and Batch Hard mining methods. All without pretraining and GOR.

5.4.2.2 Pretraining, Learning Rate, and Single Layer Training

In the second batch of experiments, we evaluated the effects of pretraining, learning rate, and single layer training on the model performance. To this end, we trained two models with a learning rate of 0.05 and two with a learning rate of 0.005. One of each was trained using SLT. All models in this batch were initialized with the weights of the softmax trained network after epoch 40 (see Section 5.4.1), did not use the length normalization layer, and were trained using the Batch All mining method.

Even though these models were trained without length normalization and using the Euclidean distance, they performed better when their embeddings were compared using the cosine distance (see Figure 5.7). The difference in performance is quite small, except in the case of the model trained with SLT and a learning rate of 0.005, where using the cosine distance led to an average relative reduction of 10 % of the EER in comparison to the Euclidean distance. This model produced the best EER out of this batch, 7.51 %, after just four epochs of triplet training and using the cosine distance.

The model with a learning rate of 0.05 and no SLT sticks out from the other models in this batch, as it regressed to an EER of 25.63 % after the first epoch of triplet training. None of the models trained with a learning rate of 0.005 have this issue, nor does the model trained with SLT and a learning rate of 0.05.

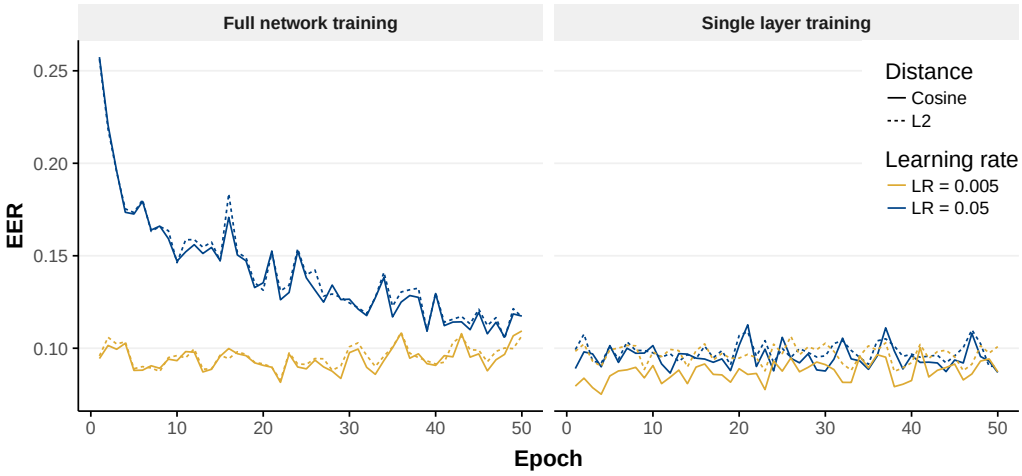


Figure 5.7: Triplet training with pretraining, with and without single layer training, and with learning rates of 0.05 and 0.005. Evaluated using both cosine and Euclidean distance.

5.4.2.3 Global Orthogonal Regularization

To evaluate whether applying the global orthogonal regularization (GOR) could yield further improvements of the model performance, we adapted the best-performing configuration up to this point – with pretraining, using a learning rate of 0.005, and both with and without SLT. Since the theory behind the GOR only applies to embeddings on the surface of a hypersphere (see Section 5.1.3.3), we used the length normalization layer for all models in this batch. We trained models with this configuration with GOR weights of 0.0, 0.5, and 1.0.

The results are shown in Figure 5.8. Due to the relatively high variation in EER from epoch to epoch, it is not obvious at all if the GOR has an effect on the performance. The best EER of 7.23 % was achieved by the model with a GOR weight of 0.5, trained without SLT, and after a

single epoch, though it is questionable if this model performed best due to the GOR weight or by chance.

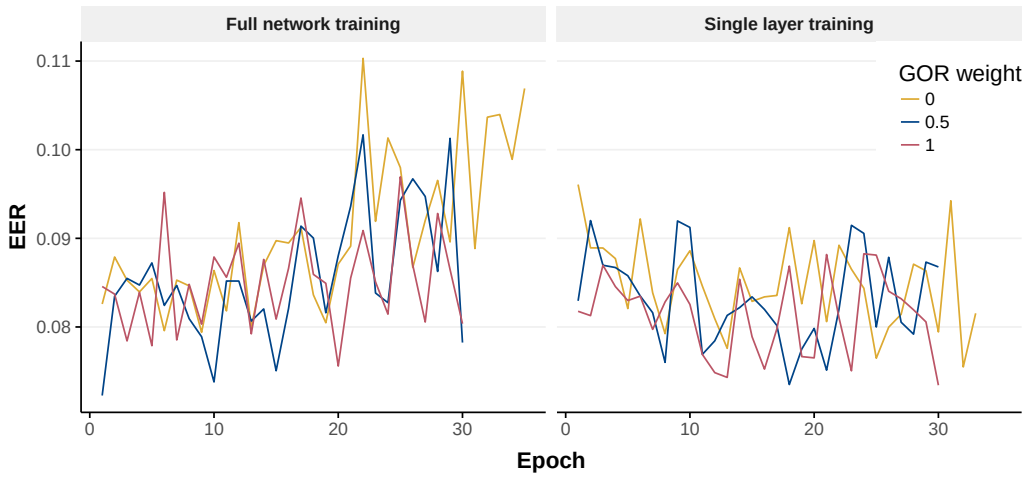


Figure 5.8: Triplet training with different GOR weights, with and without single layer training. All with pretraining and with a learning rate of 0.005.

5.4.3 Summary

Table 5.4 lists the configuration and performance of all networks that we experimented with. The best EER out of all experiments – 7.23 % – was achieved by the pretrained network trained with a learning rate of 0.005, using the Batch All triplet mining strategy, with the length normalization layer applied, and with a GOR weight of 0.5. While the difference to many other configurations is only marginal, some elements of the training that we experimented with are clearly shown to have an impact on performance. The two obvious ones are triplet mining strategy and pretraining. Both networks trained using the Batch Hard strategy were much slower to converge and resulted in lower performance than those using Batch All, and all of the best-performing networks were pretrained. Furthermore, the lower learning rate of 0.005 seems to benefit the pretrained networks, especially when single layer training is not applied.

We evaluated the performance of the best model on the VoxCeleb1-E test set (see Section 5.2.3), where it achieved an EER of 8.79 % – a 21.6 % relative increase compared to the EER achieved on the validation set.

5.4.4 Distribution of Embedding Distances

Figure 5.9 shows the distribution of distances between same-speaker embedding pairs and different-speaker pairs from different stages in the training progress of the best-performing model. After the first epoch of the pretraining, same-speaker and different-speaker distances are already quite

Training						Best performance		
Pretraining	Mining	LN	SLT	GOR	LR	Dist.	Epoch	EER (%)
yes	Batch All	yes	no	0.5	0.005	cos	1	7.23
yes	Batch All	yes	yes	1.0	0.005	cos	30	7.34
yes	Batch All	yes	yes	0.5	0.005	cos	18	7.35
yes	Batch All	no	yes	0.0	0.005	cos	4	7.51
yes	Batch All	yes	yes	0.0	0.005	cos	32	7.55
yes	Batch All	yes	no	1.0	0.005	cos	20	7.56
yes	Batch All	yes	no	0.0	0.005	cos	9	7.94
yes	Batch All	no	no	0.0	0.005	cos	22	8.16
yes	Batch All	no	yes	0.0	0.050	cos	12	8.66
this	n/a	no	no	n/a	0.050	cos	44	9.50
yes	Batch All	no	no	0.0	0.050	L2	48	10.52
no	Batch All	no	no	0.0	0.050	cos	189	10.59
no	Batch All	yes	no	0.0	0.050	cos	250	12.82
no	Batch Hard	no	no	0.0	0.050	cos	232	15.00
no	Batch Hard	yes	no	0.0	0.050	cos	278	17.33

Table 5.4: All tested model configurations, sorted by their best performance.

distinct. The separation is much stronger in epoch 44, which was the best-performing epoch from the pretraining in terms of the EER. Epoch 1 of the triplet training – the overall best-performing epoch out of all our experiments – shows slightly better separation, but the more striking change is the strong skewness of the different-speaker distances.

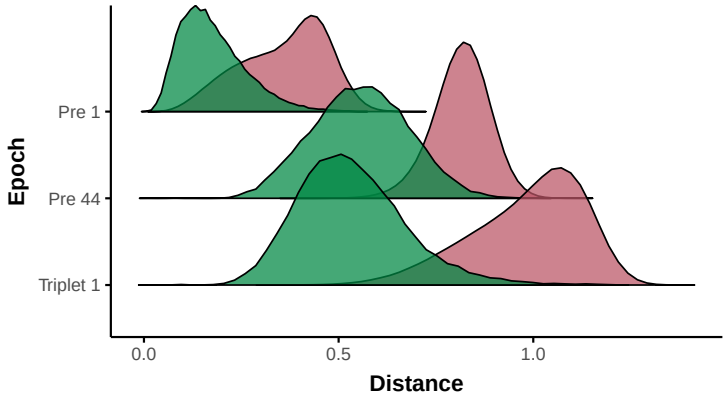


Figure 5.9: Distribution of cosine distances from same-speaker (green) and different-speaker (red) embedding pairs randomly sampled from the triplet validation data. Generated using epochs 1 and 40 from the pretraining model and epoch 4 from the best-performing triplet trained model.

Chapter 6

Speaker Presence Detection

The speaker presence detection (SPD) stage of our approach is where the speaker embeddings and information about which episodes mention a person's name in their title or description are combined to generate predictions about who is actually present on an episode. At the beginning of Chapter 4, we outlined how a human might intuitively approach this problem. In the following, we will explain how we translate this approach into computational steps.

The hypothetical person tasked with "manually" solving this problem is faced with the following situation: He has the name of a person, and a directory full of audio files, each belonging to an episode. The only information he has is that all episodes in this directory mention the person's name in their title or description. In order to find out which episode the person is speaking on, he first listens to all the episodes and pays very close attention to the voices, trying to remember each one. He then listens to each episode again, and for each voice takes note on how many of the other episodes he remembers hearing a voice that sounded similar enough that it might actually be the same person speaking. In a third step, he evaluates the likelihood for each episode that the person of interest is speaking on it. If an episode contained a voice that he recognized on many other episodes as well, then he can be quite sure that this is the voice of the person in question, and therefore that the person is speaking on this episode.

Step one, listening to all episodes and remembering the voices, is performed computationally by first diarizing each episode to find out in which segments different speakers are speaking (see Section 4.2). Then the speaker embedding network from Chapter 5 is used to extract an embedding for each speaker detected by the diarization. In step two, the algorithm goes through each diarized speaker from each episode and counts on how many of the other episodes a speaker with the same voice is recognized (*speaker recognition rate*). For each episode, the count of the speaker that was recognized the most is stored. In step three these counts are used in a logistic regression model to predict the probability for each episode that the person is speaking on it.

We will explain each step of this process in detail in the following section.

6.1 Details

6.1.1 Terminology

We will start this section by introducing some terminology that will be used throughout the rest of this chapter.

A *mention* is a pair (p, E) of a person p and an episode E, where p is mentioned in the title or description of E. We refer to p as a *speaker candidate* of E, since being mentioned suggests, but does not guarantee, that someone is actually speaking on an episode. The set of people who are speaking on an episode is referred to as $\mathbf{S}_E$. If $p \in \mathbf{S}_E$, i.e., p is actually speaking on E, we call (p, E) a *positive mention*, otherwise it is a *negative mention*.

The set of episodes for which p is a speaker candidate is called $\mathbf{E}_p$. Because the SPD algorithm relies on the assumption that the speaker candidate p is the only commonality of a set of episodes, we define $\mathbf{E}^*_{p,E}$ to be the set of episodes in $\mathbf{E}_p$ that are not from the same podcast as E and do not share another speaker candidate besides p.

For a person p and a set of episodes $\mathbf{E}$, we call the proportion of episodes in $\mathbf{E}$ on which p is a speaker the base rate of p in $\mathbf{E}$:

$$\text{BaseRate}(p, \mathbf{E}) = \frac{\| \{E \in \mathbf{E} \mid p \in \mathbf{S}_E\} \|}{\|\mathbf{E}\|} \tag{6.1}$$

One special case of a base rate is the *proportion of positive mentions* (PPM) of a person p, which is the base rate of p in $\mathbf{E}_p$.

6.1.2 Speaker Embedding Extraction

To extract speaker embeddings from an episode, it is downloaded, converted to the wave file format, and then diarized using the LIUM Speaker Diarization Toolkit [17] (described in Section 4.2). The diarization results in a list of detected speakers, each with a list of time frames indicating during which segments of the audio they are speaking. From the episode's audio signal the input features of the speaker embedding network are computed (see Section 5.1.2) and assigned to the detected speakers. Each speaker's features are then split into chunks corresponding to 3 s of audio and fed to the network. The resulting embeddings are averaged, yielding a single embedding for every speaker. We will refer to the set of speaker embeddings extracted from an episode as $\mathcal{E}_E$.

6.1.3 Speaker Recognition Rate

In the next step, the embeddings are used to compute what we call the *speaker recognition rate* (SRR) of a mention (p, E). The idea is that for negative mentions the SRR will be close to 0, while the SRR of positive mentions will reflect the PPM of p.

The first step towards obtaining the SRR for a mention (p, E) is to count, for each speaker embedding $\mathbf{e}$ in $\mathcal{E}_E$, the number of other episodes E_{other} in $\mathbf{E}^*_{p,E}$ on which we recognize this speaker's voice. This is achieved by defining a radius r and counting the number of episodes E_{other} that have at least one speaker embedding $\mathbf{e}_{other}$ with a cosine distance of less than r to the embedding $\mathbf{e}$. Since we want to arrive at a single score for each mention (p, E), we then select the highest count from the embeddings from E, which we call N_{max}.

While this number contains information about the likelihood of the person speaking on the episode, it is not yet comparable between mentions of different people as it is dependent on the number of mentions of a person. The final step to compute the SRR is therefore the conversion of N_{max} into a proportion: $\text{SRR} = \frac{N_{max}}{\|\mathbf{E}^*_{p,E}\|}$. Algorithm 1 demonstrates the computation of the SRR of a mention using pseudo-code.

Algorithm 1 Algorithm to determine the SRR for a mention.

Input: A mention (p, E)
Output: $\text{SRR}_{p,E}$

$N_{max} \leftarrow 0$
for $\mathbf{e} \in \mathcal{E}_E$ **do**
    $\text{N} \leftarrow 0$
    for $E_{other} \in \mathbf{E}^*_{p,E}$ **do**
        if $\text{any}(\{\text{D}(\mathbf{e}, \mathbf{e}_{other}) < r \mid \mathbf{e}_{other} \in \mathcal{E}_{E_{other}}\})$ **then**
            $\text{N} \leftarrow \text{N} + 1$
    if $N > N_{max}$ **then**
        $N_{max} \leftarrow N$
return $\frac{N_{max}}{\|\mathbf{E}^*_{p,E}\|}$

6.1.3.1 Rationale of the SRR

To clarify why the SRR is a suitable predictor for the presence of a person on an episode, we will describe in detail the variables underlying its computation.

Given the vast number of podcast episodes, any given person can be present on only a tiny fraction of them. Thus, on random episodes, the base rate of a particular person speaking on them is extremely low. On episodes where the person is mentioned in the title or description, that base rate will clearly be much higher, though it is still likely to be less than 1, as people might be mentioned for various reasons apart from being present on an episode. In terms of the terminology introduced above,

$$0 \approx \text{E}(\text{BaseRate}(p, \mathbf{E}_{all})) < \text{E}(\text{BaseRate}(p, \mathbf{E}_p)) < 1 \tag{6.2}$$

where E(.) is the expected value and $\mathbf{E}_{all}$ is the set of all episodes in existence.

The set of episodes $\mathbf{E}_p$ where a particular person p is a speaker candidate also contains many other speakers. For any speaker except p, however, $\mathbf{E}_p$ is essentially a random sample from $\mathbf{E}_{all}$. Due to the low base rate mentioned above, it is therefore unlikely that any speaker except p is present on more than one of the episodes $\mathbf{E}_p$. There are exceptions to this statement: A certain podcast might mention person p on every episode as a sponsor of the podcast, in which case the host would be speaking on multiple episodes of $\mathbf{E}_p$ while p never speaks on this podcast. Another possible exception is that p might often get interviewed together with an affiliated person. These exceptions are the reason why the computation of the SRR uses $\mathbf{E}^*_{p,E}$, where these cases are excluded. Because $\mathbf{E}^*_{p,E}$ does not include E itself, for a mention (p, E) and a person q, who is a speaker on E, the following is true:

$$0 \approx \mathrm{E}(\mathrm{BaseRate}(q, \mathbf{E}^*_{p,E})) < \mathrm{E}(\mathrm{BaseRate}(p, \mathbf{E}^*_{p,E})) < 1 \tag{6.3}$$

If the speaker embedding network was perfect, meaning that embeddings from the same speaker would always have a distance of 0 while embeddings of different speakers would have a larger distance, r could be set to an infinitesimal value. In this case, the SRR of a positive mention (p, E) would always be exactly equal to $\mathrm{BaseRate}(p, \mathbf{E}^*_{p,E})$. For a negative mention (p, E), where p is not speaking on E, N_{max} would originate from the embedding of some other person q. Thus, the SRR would be exactly equal to $\mathrm{BaseRate}(q, \mathbf{E}^*_{p,E})$, which would almost always be 0.

6.1.3.2 Example

To provide a more intuitive understanding of the SRR, we will demonstrate its computation by means of a very simple hypothetical scenario: Suppose a person p is mentioned on five episodes $\mathbf{E}_p$ and is speaking on three of them. All these episodes are from different podcasts, and no other people are mentioned on them, so for any mention (p, E) of person p, $\mathbf{E}^*_{p,E}$ is simply $\mathbf{E}_p \setminus \{E\}$. The diarization has detected two speakers on each episode, so we have extracted a total of ten embeddings, two per episode.

The panels in the top part of Figure 6.1 show these embeddings in a two-dimensional space. The color indicates the episode an embedding was extracted from, while letters symbolize the person an embedding belongs to. Note that the algorithm only knows the episode an embedding belongs to, not the person. The left column shows an idealized situation, where the embedding network manages to separate the embeddings of different people very clearly. The three embeddings of person p (designated as X) are very close to each other, while embeddings from different people are much further apart. The distribution of all pairwise distances between the embeddings is shown below the panels, separately for same-speaker pairs (green) and different-speaker pairs (red). (See Figure 5.9 for the same kind of distributions generated by our real embedding network.)

The middle and right column show embeddings extracted from the same audio segments but with a progressively worse performing speaker embedding network. In the middle panel, p's

embeddings are still vaguely identifiable as belonging to the same speaker, though it is not obvious whether D and A also belong to the same person or not. In the right column, the situation is even less clear. Without the information provided by the letters, purple X would appear to be from the same person as D, and the other two Xs might as well each be from a separate speaker. Note, however, that on average, even in the high error situation the same-speaker distances are still smaller than different-speaker distances.

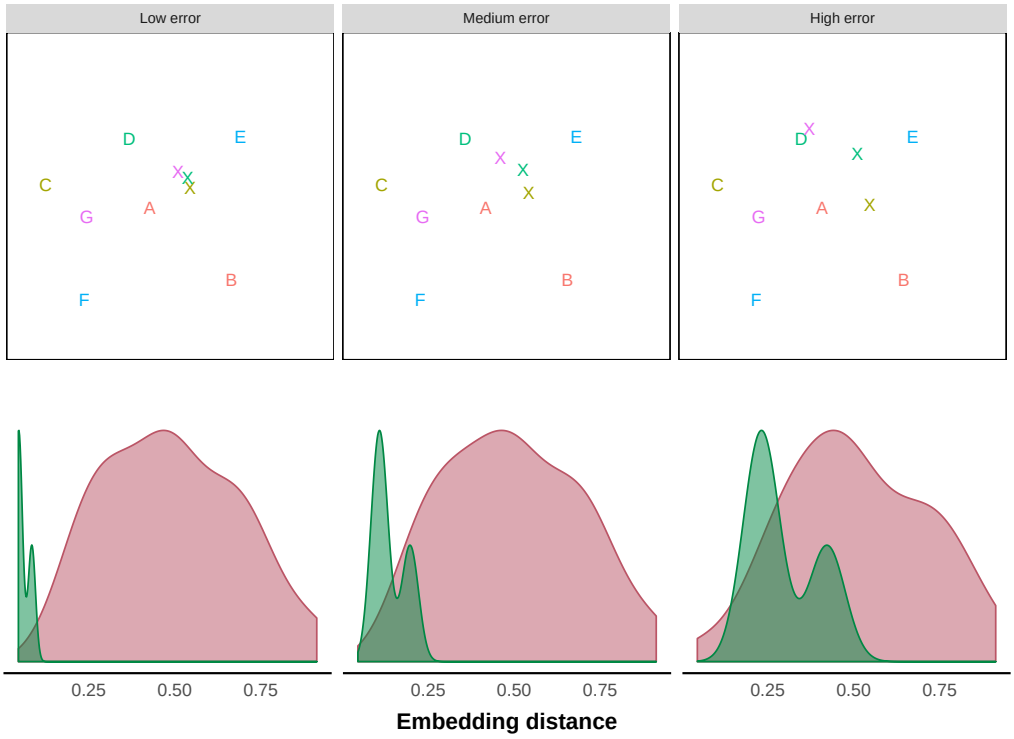


Figure 6.1: Two-dimensional speaker embeddings simulated for five episodes (coded as color) from speech from a total of nine people (identity coded as letter).

Figure 6.2 shows how the SRR of each episode is generated in the same low, medium, and high error situations. The circles are drawn around the embeddings with a radius of r, the hyper-parameter introduced in Section 6.1.3. The labels indicate the number of other episodes with at least one embedding within the circle (i.e., with a distance of less than r to the embedding at the center). In graphical terms, this is the number of different-colored embeddings within the circle, excluding the embedding that the circle belongs to. To keep this graphic somewhat clear, we only display the circle and count for the embedding of each episode with the highest count, i.e., the N_{max} of each episode.

The plots below the panels show the distribution of the N_{max} of the episodes. The green distribution represents the N_{max} of episodes where p is speaking (purple, green, and olive) while the red one represents the N_{max} of the episodes where p is not speaking (red and blue). Note

that because $\mathbf{E}^*_{p,E}$ contains four episodes for each mention in this scenario, the SRR is calculated by dividing N_{max} by four.

In the low error situation, the embeddings of p are so close together that a small value for r performs perfectly. All episodes on which p is speaking capture each other in their respective circle, while none of the other embeddings have any other episode within their circle. Thus, episodes where p is speaking have an N_{max} of 2 and an SRR of 0.5, while both are 0 for the other episodes. Note that for an episode E where p is speaking, $\mathbf{E}^*_{p,E}$ contains the two other episodes where p is speaking as well, and the two episodes where p is not speaking, so SRR = $\text{BaseRate}(p, \mathbf{E}^*_{p,E}) = 0.5$. This is the perfect situation mentioned at the end of Section 6.1.3.1.

In the medium error situation, r needs to be set to a higher value to provide good results. This has the consequence that red A now includes olive X in its circle and therefore in its episode's N_{max}, putting its SRR at 0.25. Vice versa, red A is also included in olive X's radius, so since all Xs still capture each other, one of them now has an inflated SRR of 0.75. In the high error situation, even more mistakes occur, leading the distributions of SRRs of positive and negative mentions to overlap quite strongly. However, since p's embeddings are on average closer to each other than the embeddings of other people, the expected value of the SRR is still higher for positive mentions than for negative ones, which is reflected in the distribution.

Both the low error and the high error situations result in SRRs that, with an adequate decision boundary, allow a perfect classification into positive and negative mentions. Even in the high error situation only one out of the five mentions would be misclassified, demonstrating how imperfect embeddings can still be used for this algorithm.

6.1.4 Prediction Model

We know that the higher the SRR of a mention (p, E), the higher the probability that it is positive, meaning that p is speaking on E. However, we do not yet know the exact mapping between the SRR and the probability of the mention being positive. To determine this relationship, we employ a logistic regression model using the SRR as the predictor to classify mentions as positive or negative.

Since SRRs are values in the interval $[0, 1]$, we transform them using the logit function in order to map the values onto a scale that extends beyond 0 and 1, and to give more space to values close to the extremes. To prevent this transformation from yielding infinite values, we shrink the SRRs to the interval $[0.001, 0.999]$ prior to transformation. The complete transformation is thus $\text{logit}(0.001 + 0.998 \times \text{SRR})$.

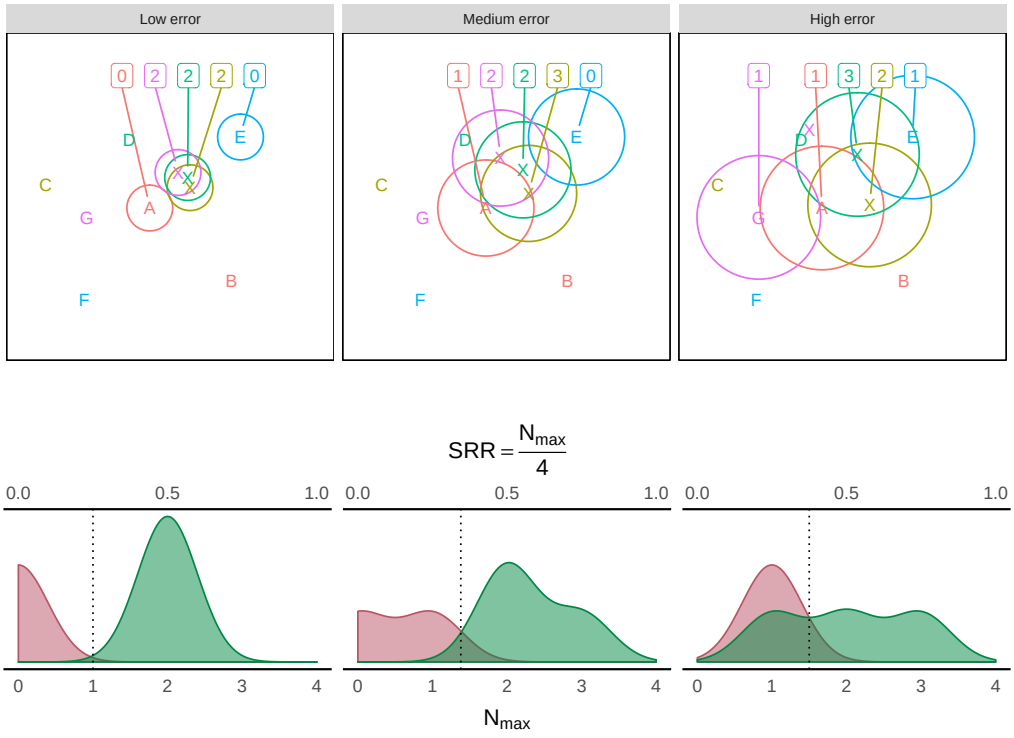


Figure 6.2: Visualization of the computation of SRRs. Dotted lines indicate decision boundaries for the best classification accuracy.

6.2 Dataset

For training and evaluation of the prediction model and the hyper-parameter r, we use a dataset of podcast episodes obtained from the iTunes podcast directory[1]. For each episode, the dataset includes the title and description of the episode, a URL to the episode's audio file, the podcast it appeared on, as well as a list of names mentioned in the title or description. Names were extracted using the Stanford Named Entity Recognizer [31], and validated by checking whether a Wikipedia page or a Twitter account with that name exists. We use these names as the speaker candidates of an episode.

As the mentions in this dataset are unlabeled, we select 50 people at random and manually label all associated mentions by whether the person is present on the episode ("speaker") or not ("non-speaker"). The number of mentions per person is exponentially distributed – 61 % of people are mentioned only once, 93 % less than 10 times, and 97 % less than 20 times. As the results of the algorithm for people with very few mentions would likely not be of great interest to a hypothetical audience of a podcast directory such as described in Chapter 1, we restrict the random selection to people that have been mentioned at least 20 times, leaving 39 203 people to

[1] https://itunes.apple.com/us/genre/podcasts/id26?mt=2

be sampled from. Furthermore, we perform the sampling with a probability of selection that is equal to a person's number of mentions divided by the total number of mentions.

6.3 Training and Evaluation

6.3.1 Hyper-parameter Search

To find the optimal value for the hyper-parameter r, we run a cross-validation evaluation procedure (described in the next section) for different values of r. In a first iteration, we test values between 0 and 1 evenly spaced in intervals of 0.01. In the following iterations, we refine the search around the value of r that led to the highest accuracy, until the two highest accuracies span a range of 10^{-3} or less. It is not necessary to search the range of r above 1, as a cosine distance of 1 indicates orthogonality and a radius of this value or higher cannot plausibly lead to good performance.

6.3.2 Cross-Validation for the Hyper-parameter Search

To estimate the out-of-sample accuracy of the model during the hyper-parameter search, we employ a cross-validation scheme where each person's mentions are used as the validation set once. Since all episodes where a person is mentioned are used in determining the SRR of each of their mentions, this method ensures that information that was in any way involved in model training will not be used for evaluation.

From each fold, we obtain the unnormalized confusion matrix, i.e., the number of true positives, false positives, true negatives, and false negatives. We calculate a total confusion matrix by summing up the counts of each individual matrix in each quadrant. This matrix allows us to estimate the average performance of the model, by calculating accuracy, precision, and recall from it. Because we use unnormalized confusion matrices from each fold, these averages will be weighted by the number of mentions in each fold (i.e., person).

6.3.3 Evaluation of the Final Model

Since we select the model based on the above-mentioned cross-validation, we require a separate test set to obtain an unbiased estimate of the performance of the final model. We therefore withhold data from 17 of the 50 people from the hyper-parameter search and use this as the test set.

Using the best r that we find in the hyper-parameter search, we train a model on the data of all 33 people in the training set and evaluate the performance of this model on the test set. We perform this evaluation both for the complete test set and for each person in the test set separately, to gain insight about the connections between attributes of a person and the model performance on this person's mentions.

6.4 Results

6.4.1 Descriptives of Training and Test Set

Figure 6.3 shows the distribution of the number of mentions and proportion of positive mentions (PPM) of the people in the training and test sets. Due to the distribution of the number of mentions of people described in Section 6.2, most people in both sets have been mentioned on 20 to 50 episodes, but examples of more frequently mentioned people exist in both sets. The PPMs are quite evenly distributed over the full range, though the test set contains a cluster of people towards the high end of this variable. In total, the training set contains 1660 mentions from 33 people, of which 40.4 % are positive, and the test set contains 777 mentions from 17 people, with 47.4 % being positive.

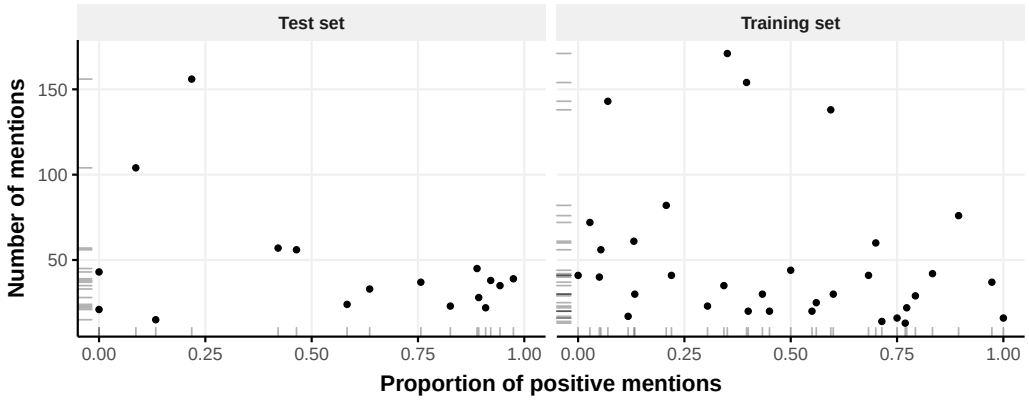


Figure 6.3: Number of mentions and PPM of people in the training and test set.

6.4.2 Hyper-Parameter Search

Figure 6.4 plots average accuracy, precision, and recall for the values of r that we tested in the hyper-parameter search. The value of r that led to the best accuracy was 0.1685, at which point the model achieved, averaged over the validation sets, an accuracy of 81.9 %, a precision of 76.8 %, and a recall of 79.0 %. When r was 0 or rose above 0.83, the logistic regression predicted the more frequent class "not speaking" for all mentions, thus leading to an accuracy equal to the base rate of this class, and a recall of 0 %. This indicates that when using these values for r, the SRR does not contain any information that is useful to predict speaker presence.

6.4.3 Final Model

6.4.3.1 Distribution of the SRRs

Figure 6.5 shows the distribution of the logit-transformed SRRs obtained using $r = 0.1685$, for positive and negative mentions separately. It indicates that the SRR can be used to achieve a

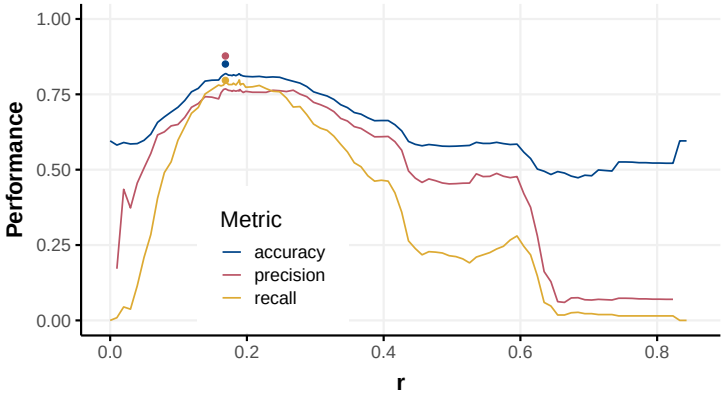


Figure 6.4: Performance metrics of models over a range of values for r. The dots mark the performance of the final model on the test set.

considerable, though not perfect, separation between positive and negative mentions.

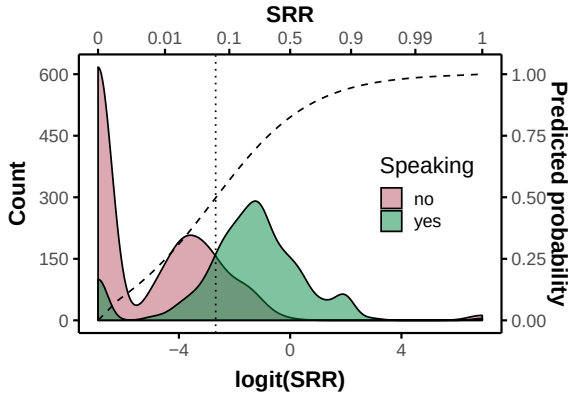


Figure 6.5: Distribution of logit-transformed SRR scores for positive and negative mentions. The dashed line indicates the final model's predicted probability of a mention being positive, the dotted line the decision boundary at $\hat{p} = 0.5$.

6.4.3.2 Performance of the Prediction Model

The model trained on the full training set using $r = 0.1685$ achieved an accuracy of 85.1 %, a precision of 87.7 %, and a recall of 79.6 % on the test set (indicated by the points in Figure 6.4). These values are very close to those obtained from the cross-validation during the hyper-parameter search, with the exception of precision, where the value obtained from the test set is quite a bit higher. This suggests that the value of r selected during the hyper-parameter search is likely not significantly overfitted to the training data.

To gain a more fine-grained insight into the performance of the model, we obtained the performance metrics for each person separately. In Figure 6.6 these per-person metrics are plotted against the PPM of a person, together with the metrics obtained from the cross-validation during the hyper-parameter search. The plot shows that both precision and recall tend to be lowest for people where only a small fraction of mentions is positive. This drop-off is most dramatic for people with less than around 20 % positive mentions. Above this value, both metrics increase slowly but steadily, showing the best performance for people at the high end of the PPM. The accuracy of the model does not follow the same trend. Instead, it shows a slightly U-shaped curve, with the lowest accuracy for people that have about 20 % to 50 % positive mentions.

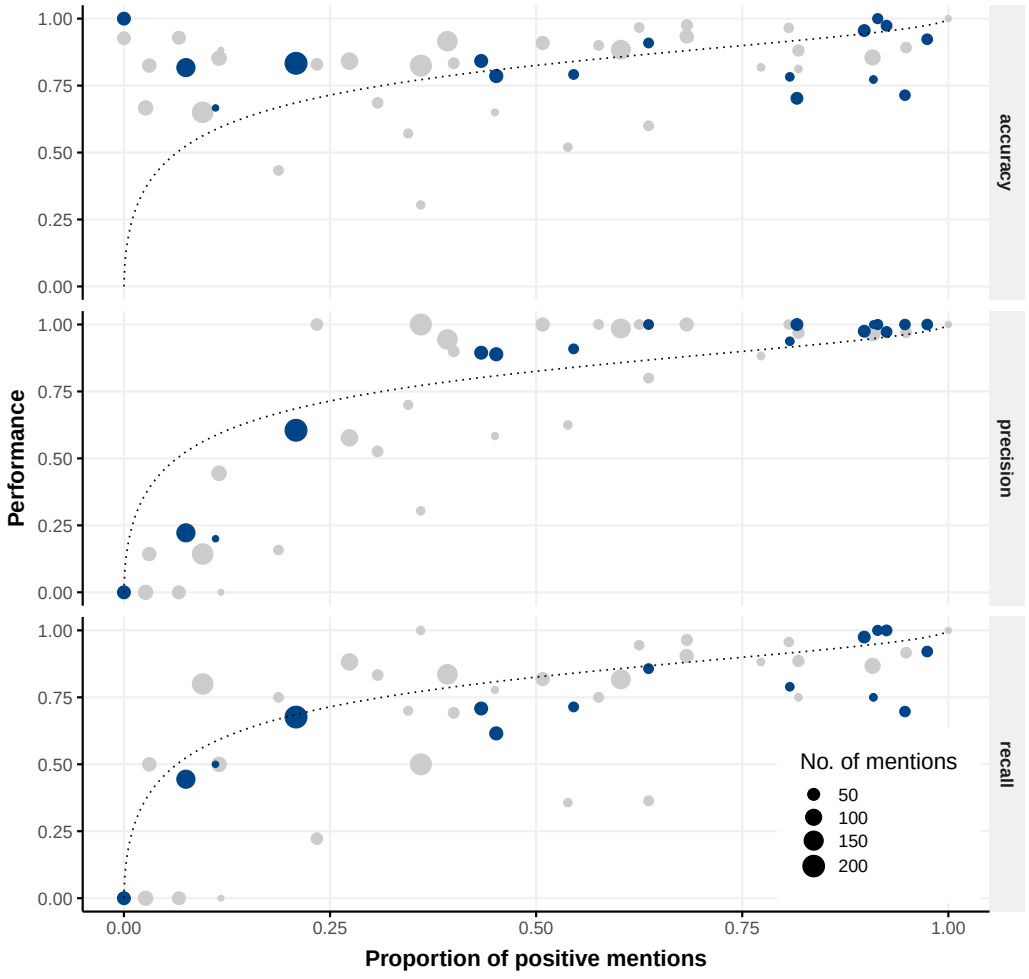


Figure 6.6: Performance metrics of the final model for each person plotted against their PPM. Grey points show the validation metrics for each person from the hyper-parameter search. The dotted line indicates the model's predicted probability of a mention to be of class "speaker" (though the x-axis does not directly indicate SRR in this plot, the SRR is a measure that is designed to reflect the PPM).

Chapter 7

Discussion

7.1 Speaker Embedding Network

Overall, our experiments on the speaker embedding network have confirmed that a DCN trained using the triplet loss is an adequate method to generate speaker embeddings from speech recorded in a variety of environments, using different recording equipment, and partly containing background noises, such as the VoxCeleb datasets [13, 15]. We showed that training using the triplet loss leads to significantly better embeddings over the intermediary layer embeddings resulting from training on a speaker recognition task. Our best model achieved an EER of 8.79 % on the VoxCeleb1-E [13] test set.

7.1.1 Performance Compared to Other Research

Using the same network architecture, [11] achieved EERs between 1.13 % and 5.92 %, thus significantly lower than ours. However, their description of the datasets that they used to evaluate performance suggests that they contain utterances that are quite homogeneous for each speaker. The VoxCeleb datasets, on the other hand, contain utterances from a variety of different conditions for most speakers, making a verification task a lot more demanding.

[13] used a slightly different network architecture and loss function, but evaluated their performance on VoxCeleb1-E, the same test set that we used. Using a model with a 34-layer ResNet [12] architecture, they achieved an EER of 4.42 % on this test set. Our best model, with an EER of 8.79 %, made almost twice as many mistakes. There are some differences between our approach and theirs that might explain part of this difference. First, their network consisted of 50 layers, allowing it to learn representations on more levels of abstraction than ours which used only 29 layers. Second, they describe multiple methods of generating an embedding from a long utterance but do not specify which one they used on the VoxCeleb1-E test set. The method that we used – taking the mean of embeddings generated from slices of the utterance (see Section 5.2.3.3) – performed slightly worse than the other methods in their experiments on other test data. Thus, assuming that they employed one of the better-performing methods

of embedding aggregation on VoxCeleb1-E, this might explain a part of the difference in EER between their approach and ours.

7.1.2 Experiments

Our experiments yielded several results that were unexpected or interesting and warrant closer examination, which we will go into in the following sections.

7.1.2.1 Cosine vs. Euclidean Distance

In several cases, the embeddings performed better when compared using the cosine distance, even when the network was trained without length normalization. The most striking example of this was the pretraining, where the EERs using both distance measures improved very similarly in the first few epochs, but then diverged. While the EER using the cosine distance continued to improve, achieving a performance of 9.5 %, the EER using the Euclidean distance started to increase up to a level of around 19 %. A possible reason for this behavior might be that DNNs trained using cross-entropy tend to be overly confident in their predictions [32]. Thus, the embedding layer might be generating activations that are exaggerated in comparison to the actual presence of a certain voice feature in an utterance. Since the cosine distance is not affected by the difference in magnitude of the embeddings, it might be neutralizing this exaggeration, leading to a more accurate representation of the difference between two utterances.

This difference between the two distance measures also occurred for the network trained using SLT with a learning rate of 0.005 described in Section 5.4.2.2. Though the network was trained without length normalization, the cosine distance consistently outperformed the Euclidean distance throughout the triplet training (see Figure 5.7). As the triplet training should reverse any exaggeration of features that might be inherited from the pretraining, the explanation offered above is unlikely to apply to this case. However, the triplet training itself might lead to some overfitting, in which case the cosine distance would act as a kind of regularization, leading to better generalization to the validation data.

7.1.2.2 Local Minima

One of the models described in Section 5.4.2.1 showed an unusual training progress: The EER seemed to have converged after around 70 epochs, but then showed a sudden drop of roughly 3 % (see the right plot in Figure 5.6). As this was the only training run exhibiting this pattern, this indicates that models may be getting stuck at local minima in the parameter space, which they are unlikely to escape from. Thus, model performance might benefit from methods that deal with this issue, such as training models with the same configuration multiple times with different weight initializations.

7.1.2.3 Triplet Mining Method

In the experiments where we used the Batch Hard mining method, the model learned much slower than using the Batch All method. This is surprising, as the Batch Hard method is designed to select hard triplets in order to prevent learning from slowing down due to a flattening gradient. We suspect that since the Batch Hard method uses much fewer triplets from a batch than the Batch All method, it is possible that fewer nodes of the network reach a positive activation, and thus fewer weights have a non-zero gradient (since the ReLU function is flat below zero). If this is the case, a smaller part of the weights would be updated in each epoch than using Batch All, which might negate the effect of the Batch Hard mining method to increase the steepness of the gradient.

7.1.2.4 Single Layer Training

Four out of the five best-performing models were using single layer training after the pretraining (Table 5.4). This suggests that, as we speculated in Section 5.1.5, this method can help the embedding layer quickly adjust to the change in task from pretraining to triplet loss training. The most dramatic display of this effect was shown by the difference between the two models with a learning rate of 0.05 described in Section 5.4.2.2. While the model using SLT performed very well after just a single epoch, the model not using SLT performed as if it had not been pretrained at all (see Figure 5.7). The low performance of the latter can be explained by the fact that no length normalization was used for these models, and the performance of the pretraining model using Euclidean distance was quite poor. The fact that the model using SLT performed so well regardless shows that SLT is an effective tool to adapt a layer to a change in demands, without "confusing" the rest of the network in the process.

While we either used SLT for the complete triplet training or not at all, the best use case for this method might be to employ it during the first few epochs after pretraining and then continue to train the full network as normal once the embedding layer has adapted. This would allow the rest of the network to learn better representations for the new task, after the embedding layer, for which the change in requirements is likely to be much greater, has learned to make the best use possible of the prior layer's pretrained representations.

7.2 Speaker Presence Detection

The results that we obtained from our approach to the speaker presence detection task clearly show that our algorithm indeed managed to extract useful information from the combination of mentions and voice features. The accuracy of 85.1 % on the test set is high enough to set our results apart from a naive classifier always predicting the more frequent class, which would attain an accuracy of 52.6 %.

7.2.1 Performance Differences Between People

The per-person performance metrics presented in Section 6.4.3.2 provide an interesting insight into the conditions under which the model performs well. The drop in precision and recall that can be seen in Figure 6.6 for people with very few positive mentions was to be expected. The SRR contains a certain level of noise since speaker embeddings are not perfect and some embeddings will fall within the radius r of an embedding from another speaker just by chance. That is why, as Figure 6.5 shows, the SRRs of negative mentions are distributed close to 0, but do not all equal 0. Thus, even if the SRRs of the positive mentions of a person with a low proportion of positive mentions were exactly equal to this proportion, they might fall below the decision boundary of the logistic regression model, as it needs to account for the SRRs of negative mentions that are inflated due to noise.

The reason that the accuracy does not suffer from this in the same way is that incorrectly classifying the small number of positive mentions of a person with a low PPM as negative only decreases the accuracy for this person by exactly their PPM. This also explains the slight U-shape of the accuracy. The SRRs of the positive mentions of people with a moderately low PPM are still in danger of falling below the decision boundary. Since they now account for a larger percentage of this person's mentions, accuracy is impacted more. The higher the PPM of a person, the less likely their positive mentions are to be misclassified, as their SRRs become large enough not to be pushed below the decision boundary by the noise, and therefore the accuracy shows a trend upwards for people with about 30 % or more positive mentions.

7.2.2 Recommendations for Future Research

There are many aspects of this work where we see the potential for improvement that we could not explore due to time constraints. We will lay out the most important ones here.

7.2.2.1 Improvements in the Speaker Embedding Network

Since the predictive power of the SRRs relies heavily on the quality of the speaker embeddings, any improvement of the speaker embedding network would cascade to the performance of the speaker presence detection. Based on the results of our experiments, we suspect that significant improvements could be attained from several changes.

The simplest change to implement is the method with which embeddings are used to compare longer utterances. In experiments on different methods, [13] have found the method we used (see Section 6.1.2) to perform between 5 % and 12 % worse in terms of relative EER than other methods. The best method they found was to take a sample of ten 3 s segments from the utterance and extract an embedding from each one. As a measure of distance between two utterances, the mean of the distances between each possible pair of their embeddings is used.

Further improvements in performance could likely be achieved from changes in the network

architecture. In experiments using the ResNet [12] architecture, [13] found that using 50 layers resulted in a 16.8 % relative reduction in EER compared to using 34 layers.

Other options that we did not have time to explore include optimizing the margin m of the triplet loss, using a decreasing learning rate schedule over the training progress, and testing different parameters for the audio feature extraction (e.g., frequency range, number of filters).

7.2.2.2 Information Not Captured by the SRR

While the SRR has proved to contain enough information to classify mentions with decent accuracy, there is some information that is not being used by our algorithm in its present form that would likely help to achieve better performance.

The most obvious drawback of the SRR is that it evaluates distances between embeddings purely by comparing them to the threshold r in a binary fashion. This is a quite rudimentary way to make use of the fact that the closer two embeddings are to each other, the more likely it is that they are from the same person. A score that weights embeddings according to their distance in some way would likely serve as a better predictor of a person's presence on an episode.

Another aspect that is currently ignored by the algorithm is the number of embeddings extracted from each episode. The more speaker embeddings an episode contains, the higher the probability that this episode will be included in other episodes' N_{max} due to noise in the embeddings. This fact could relatively easily be accounted for by decreasing the influence of an episode on other episodes' N_{max} in proportion with the episode's number of embeddings.

7.2.2.3 Speaker Diarization

We employ speaker diarization in order to narrow down the number of speaker embeddings that we obtain from each episode, while still capturing an embedding from every speaker. While diarization does fulfill this purpose, it is not a perfect fit for this use case. We are less interested in knowing exactly when each speaker starts and stops speaking, and more in obtaining samples of each speaker's voice, preferably in their normal, everyday way of speaking to minimize variance. A simple way of achieving this would be to extract embeddings from all 3 s segments of an episode, and then cluster these embeddings using the mean-shift algorithm. As this is a mode-finding algorithm, this would likely yield cluster centers that represent each speaker's voice in it's most frequent form. Mean-shift clustering has in fact been used with success in a diarization task before [33].

7.2.2.4 Training on a Larger Dataset

As we had to manually label the mentions used to train and test the SPD algorithm we were quite limited in the size of the labeled data that we could obtain. Furthermore, since these mentions were distributed over just 50 people, it is quite likely that this sample is not fully representative

of the population. It would therefore add much fidelity to the results if the algorithm could be tested on a larger dataset.

There is a way to obtain such a dataset without the work of manually labeling mentions on podcast episodes. The VoxCeleb datasets [13, 15] contain, for each person, a list of videos where this person is being interviewed (the lists were obtained by searching YouTube for the name and the word "interview"). It would thus be possible to extract the audio from these videos to obtain something that resembles podcast episodes quite closely. A dataset with a mix of positive and negative mentions could then be simulated by sampling for each person varying numbers of this person's videos and random other people's videos. In this way, many kinds of different situations could be simulated, allowing a very fine-grained evaluation of how the algorithm performs in different circumstances.

7.2.3 Application in Other Domains

As hinted at in Chapter 1, the algorithm presented in this work can relatively easily be generalized and applied to other domains as well. An obvious example is a set of photos scraped from websites, where the text close to the image is known. The text could be searched for names, and face embeddings extracted from the image. By applying our algorithm to this data, names of people could automatically be matched with the faces that are most likely to belong together.

For a more ambitious example, suppose a robot is moving around in a public space, recording both video of the scenery and the conversations of people in its vicinity. Suppose also that the robot's software includes reliable speech recognition that can determine the nouns in the conversations, and an object embedding network, which extracts embeddings representing distinct objects from the video feed. Combining these two elements using our algorithm, the robot could autonomously learn the names of the objects that it sees without explicit supervision, much like humans learn languages growing up.

In general, the algorithm can be applied to any domain where media are annotated in a way that hints at the presence of some entity in the content, and embeddings can be extracted from that content to represent the entities contained within it. Humans are currently creating immense amounts of media of this kind. An algorithm such as ours that can be trained on a modest amount of data and then be used to gain more certainty about the content of the media might therefore prove very useful for a wide variety of purposes.

7.3 Conclusion

We introduced a novel algorithm that utilizes various elements from the area of speech processing to detect the presence of speakers on podcast episodes, by combining them with uncertain information about speaker presence from podcasts' metadata. Though we do not know of any prior research that is similar enough in nature to put our results in perspective, this work has

proven that our proposed approach to the speaker presence detection task can determine speaker presence with an accuracy far above the baseline of a naive classifier. Due to the limited timeframe of this project, many options for improvement have remained unexplored which have the potential to improve the performance significantly.

References

[1] Clark D Shaver and John M Acken. "A Brief Review of Speaker Recognition Technology". In: *Electrical and Computer Engineering Faculty Publications and Presentations* 350 (2016). URL: https://pdxscholar.library.pdx.edu/ece%7B%5C_%7Dfac/350.

[2] F E Eric Bimbot et al. "A Tutorial on Text-Independent Speaker Verification". In: *EURASIP Journal on Applied Signal Processing* 4 (2004), pp. 430–451. ISSN: 1687-6172. DOI: Doi10.1155/S1110865704310024. arXiv: arXiv:1011.1669v3.

[3] M.H. Moattar and M.M. Homayounpour. "A review on speaker diarization systems and approaches". In: *Speech Communication* 54.10 (Dec. 2012), pp. 1065–1103. ISSN: 01676393. DOI: 10.1016/j.specom.2012.05.002.

[4] P. Duhamel and M. Vetterli. "Fast fourier transforms: A tutorial review and a state of the art". In: *Signal Processing* 19.4 (Apr. 1990), pp. 259–299. DOI: 10.1016/0165-1684(90)90158-U. URL: https://www.sciencedirect.com/science/article/pii/016516849090158U?via%7B%5C%%7D3Dihub.

[5] S. S. Stevens, J. Volkmann, and E B Newman. "A Scale for the Measurement of the Psychological Magnitude Pitch". In: *The Journal of the Acoustical Society of America* 8.3 (1937), pp. 185–190. ISSN: 00014966. DOI: 10.1121/1.1915893. URL: http://scitation.aip.org/content/asa/journal/jasa/8/3/10.1121/1.1915893.

[6] Min Xu et al. "HMM-based audio keyword generation". In: *Advances in Multimedia Information Processing - PCM*. 2004, pp. 566–574. ISBN: 3-540-23985-5. DOI: 10.1007/978-3-540-30543-9. URL: http://link.springer.com/chapter/10.1007/978-3-540-30543-9%7B%5C_%7D71.

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 0028-0836. DOI: 10.1038/nature14539. URL: http://www.nature.com/articles/nature14539.

[8] Jiang Wang et al. "Learning Fine-grained Image Similarity with Deep Ranking". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2014), pp. 1386–1393. ISSN: 10636919. DOI: 10.1109/CVPR.2014.180. arXiv: 1404.4661.

[9] Najim Dehak et al. “Front-End Factor Analysis for Speaker Verification”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 19.4 (2011). DOI: 10.1109/TASL.2010.2064307.

[10] Ehsan Variani et al. “Deep Neural Networks for Small Footprint Text-dependent Speaker Verification”. In: *IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP)*. 2014.

[11] Chao Li et al. “Deep Speaker: an End-to-End Neural Speaker Embedding System”. In: *arXiv preprint arXiv:1705.02304* (2017).

[12] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *arXiv* (2015). arXiv: 1512.03385.

[13] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. “VoxCeleb2: Deep Speaker Recognition”. In: *arXiv*. 2018.

[14] Sumit Chopra, Raia Hadsell, and Yann LeCun. “Learning a Similarity Metric Discriminatively, with Application to Face Verification”. In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (2005), pp. 349–356. ISSN: 10636919. DOI: 10.1109/CVPR.2005.202. URL: http://yann.lecun.com/exdb/publis/pdf/chopra-05.pdf.

[15] Arsha Nagrani, Joon Son Chung, and Andrew Zisserman. “VoxCeleb: A Large-Scale Speaker Identification Dataset”. In: *Proc. Interspeech 2017*. 2017, pp. 2616–2620. DOI: 10.21437/Interspeech.2017-950. URL: http://www.robots.ox.ac.uk/%7B~%7Dvgg/publications/2017/Nagrani17/nagrani17.pdf%20http://dx.doi.org/10.21437/Interspeech.2017-950.

[16] Mickael Rouvier, Pierre-Michel Bousquet, and Benoit Favre. “Speaker Diarization Through Speaker Embeddings”.

[17] Sylvain Meignier and Teva Merlin. “LIUM_SpkDiarization: An Open Source Toolkit for Diarization”. In: *Proc. CMU SPUD Workshop, March 2010, Dallas (Texas, USA)*. 2010.

[18] S Chen and P Gopalakrishnan. “Speaker, Environment and Channel Change Detection and Clustering via the Bayesion Information Criterion”. In: *Proc. DARPA Broadcast News Transcription and Understanding Workshop* 6 (1998), pp. 127–132. ISSN: 0898-2643. DOI: 10.1177/0898264313499931.

[19] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *arXiv* (2015). ISSN: 0717-6163. DOI: 10.1007/s13398-014-0173-7.2. arXiv: 1502.03167.

[20] Amirsina Torfi. *SpeechPy: Speech recognition and feature extraction*. 2017. DOI: 10.5281/zenodo.840395.

[21] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “FaceNet: A unified embedding for face recognition and clustering”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 07-12-June. 2015, pp. 815–823. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298682. arXiv: 1503.03832.

[22] Ben Harwood et al. "Smart Mining for Deep Metric Learning". In: *arXiv:1704.01285v3*. 2017.

[23] Alexander Hermans, Lucas Beyer, and Bastian Leibe. "In Defense of the Triplet Loss for Person Re-Identification". In: *arXiv* (2017). arXiv: 1703.07737. URL: http://arxiv.org/abs/1703.07737.

[24] Olivier Moindrot. *tensorflow-triplet-loss*. URL: https://github.com/omoindrot/tensorflow-triplet-loss.

[25] Vijay Kumar B G, Gustavo Carneiro, and Ian Reid. "Learning Local Image Descriptors with Deep Siamese and Triplet Convolutional Networks by Minimising Global Loss Functions". In: *arXiv* (Dec. 2015). arXiv: 1512.09272.

[26] Xu Zhang et al. "Learning Spread-out Local Feature Descriptors". In: *Proceedings of the IEEE International Conference on Computer Vision*. Vol. October 20. 2017, pp. 4605–4613. ISBN: 9781538610329. DOI: 10.1109/ICCV.2017.492. arXiv: 1708.06320.

[27] Qiong Cao et al. "VGGFace2: A dataset for recognising faces across pose and age". In: *arXiv* (2017). DOI: 10.1109/FG.2018.00020. arXiv: 1710.08092. URL: http://arxiv.org/abs/1710.08092.

[28] Joon Son Chung and Andrew Zisserman. "Out of time: automated lip sync in the wild". URL: http://www.robots.ox.ac.uk/%7B~%7Dvgg/publications/2016/Chung16a/chung16a.pdf.

[29] Erik Bernhardsson, Elias Freider, and Arash Rouhani. *Luigi*. URL: https://github.com/spotify/luigi.

[30] Jianmin Chen et al. "Revisiting Distributed Synchronous SGD". In: *arXiv* (Apr. 2016). arXiv: 1604.00981. URL: http://arxiv.org/abs/1604.00981.

[31] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. "Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling". In: *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*. 2005, pp. 363–370. DOI: 10.3115/1219840.1219885. URL: https://nlp.stanford.edu/software/CRF-NER.shtml.

[32] Chuan Guo et al. "On Calibration of Modern Neural Networks". In: *arXiv* (2017). ISSN: 1938-7228. arXiv: 1706.04599.

[33] Mohammed Senoussaoui et al. "A Study of the Cosine Distance-Based Mean Shift for Telephone Speech Diarization". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 22.1 (Jan. 2014), pp. 217–227. DOI: 10.1109/TASLP.2013.2285474. URL: http://ieeexplore.ieee.org/document/6633085/.