



**School of
Engineering**

InIT Institute of Applied
Information Technology

Bachelor thesis (IT)

Speaker Clustering for Real-World Data using Deep Learning

Author	Christian Lauener Claude Lehmann
---------------	-------------------------------------

Main supervisor	Prof. Dr. Thilo Stadelmann
------------------------	----------------------------

Date	07.06.2019
-------------	------------



DECLARATION OF ORIGINALITY

Bachelor's Thesis at the School of Engineering

By submitting this Bachelor's thesis, the undersigned student confirms that this thesis is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the Bachelor thesis have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date:

Signature:

.....

.....

.....

.....

The original signed and dated document (no copies) must be included after the title sheet in the ZHAW version of all Bachelor thesis submitted.

Abstract

In recent years substantial progress was made in the areas of speaker recognition and clustering using deep neural networks. Due to the time dependant aspects of speech, recurrent neural networks and long short-term memory architectures are being applied to a wide array of time series problems. Their ability to include previously seen data in their decision making process is highly beneficial for these tasks and leads these neural network architectures to this sort of application.

Increased problem complexity typically requires larger amounts of meaningful training data to produce applicable results. A common approach to overcome long and intensive training phases is to increase the hardware computation capacity. For small scale businesses this is not a financially feasible way.

In this thesis we explore an active learning approach to reduce the amount of training data shown to the neural network while still achieving reasonable results on a complex dataset. We show a modular implementation of this approach, with which we are able to achieve a new record on the TIMIT dataset using 590 speakers and show why our expectations could not be met on the VoxCeleb2 dataset. Additionally, we propose improvements to the system and present ideas for future work.

Zusammenfassung

In den letzten Jahren wurden in den Bereichen Speaker Recognition und Clustering durch die Verwendung von Deep Neural Networks beträchtliche Fortschritte erzielt. Durch die zeitabhängigen Aspekte von gesprochener Sprache werden Recurrent Neural Networks und Long Short-Term Memory Strukturen auf eine Vielzahl von Problemen in diesem Bereich angewendet. Die Fähigkeit dieser Modelle, Entscheidungen auf zeitlich bereits zurückliegende Daten zu stützen, ist äusserst vorteilhaft für diese Art von Aufgaben.

Eine erhöhte Komplexität der Problemstellung erfordert typischerweise eine grössere Menge an Trainingsdaten um befriedigende Resultate zu erzielen. Üblicherweise wird zusätzliche Rechenleistung in Form von Hardware eingesetzt um lange und intensive Trainingsphasen zu unterbinden. Dies ist für kleinere und mittlere Unternehmen aus finanzieller Sicht jedoch oft nicht möglich.

Wir erforschen in dieser Arbeit einen Active Learning Ansatz um die Datenmenge, welche ein neuronales Netzwerk zu Trainingszwecken benötigt, klein zu halten und dennoch akzeptable Resultate auf einem komplexen Datensatz zu erzielen. Wir präsentieren eine modulare Implementation eines solchen Ansatzes, mit welchem wir auf dem TIMIT Datensatz mit 590 Sprechern einen neuen Bestwert erreichen, und zeigen auf weshalb die Resultate auf dem VoxCeleb2 Datensatz nicht den Erwartungen entsprechen. Zusätzlich stellen wir Verbesserungen für das System vor, um es künftig weiter auszubauen.

Preface

During our years of study at ZHAW, we developed a passion for artificial intelligence and quickly realized that both of us wanted our thesis to be in this research field. While it was announced as a work-intensive task, the possibility to contribute in an active research area and, if the results were noteworthy, the opportunity to work on and publish a research paper, encouraged us to take on this challenge.

We would like to thank the following people for their support and valuable inputs throughout this project.

Prof. Dr. Thilo Stadelmann (ZHAW InIT) for giving us the opportunity to work on an interesting and current research problem, his support during the many discussions of ideas and questions throughout the semester and his trust in our ability to tackle this challenging problem.

Mohammadreza Amirian (ZHAW InIT) for his support on debugging the pipeline and getting the project up and running.

Hansjürg Stocker (VSMP/DMK) for leading us through the deceitful maze of complex combinatorics and his ideas on approaching batch sizing mathematically.

Martina Lauener (FFHS) for her valuable input on how to write a scientific report.

Dennis Camera (ZHAW Student) for his critical questions during the many conversations we had during the semester.

Daniel Neururer (ZHAW Student) for working alongside us on the framework during the semester.

Contents

1. Introduction	6
1.1. Motivation	6
1.2. Related Work	6
1.3. Goals	7
1.4. Document Structure	7
2. Theoretical principles	8
2.1. Clustering	8
2.2. Speaker Clustering	8
2.3. Embeddings	9
2.4. Prosodic Features	9
2.5. Spectrograms	10
2.6. Mel Spectrogram	10
2.7. Speech in the Wild	11
2.8. Neural Networks	11
2.9. RNN	12
2.10. LSTM	13
2.11. Bidirectional LSTM	13
2.12. KL-Divergence	14
2.13. Hinge Loss	14
2.14. Active Learning	15
3. Approach / Methodology	16
3.1. Datasets	16
3.2. Preparation	17
3.3. Processing of VoxCeleb2 Data	17
3.4. Adapting to VoxCeleb2	17
3.5. Active Learning Approach	18
3.6. Neural Network Architecture	19
3.7. Misclassification Rate	20
3.8. Homogeneity Score	20
3.9. Completeness Score	21
3.10. Used Software	21
4. Results	22
4.1. Experimental Setup	22
4.2. Datasets	22
4.3. Experiments	23
4.3.1. Experiment #1: TIMIT-100 vs. TIMIT-40	24
4.3.2. Experiment #2: TIMIT-590 vs. TIMIT-40 with modest active learning	26
4.3.3. Experiment #3: TIMIT-590 vs. TIMIT-40 with increased active learning	28
4.3.4. Experiment #4: VoxCeleb2-600 vs. VoxCeleb2-120	30
4.3.5. Experiment #5: VoxCeleb2-600 vs. VoxCeleb2-120 with modest active learning	32
4.3.6. Experiment #6: VoxCeleb2-600 vs. VoxCeleb2-120 with increased active learning	34
4.4. Segment Length Sweet Spot	36
4.5. Active Learning Comparison	36
5. Discussion	38
5.1. Conclusion	38

5.2. Goals	38
5.3. Lessons Learned	39
5.4. Future Work	39
6. Listings	41
Bibliography	41
List of Figures	45
List of Tables	46
Glossary	47
A. Appendix	I
A.1. Official Project Description	II
A.2. Manual Project Setup	III
A.3. Docker setup on the GPU cluster	III
A.4. Continuing Development	IV
A.5. Experiment Commands	V
A.5.1. Experiment #1: TIMIT-100 vs. TIMIT-40	V
A.5.2. Experiment #2: TIMIT-590 vs. TIMIT-40 with modest active learning	V
A.5.3. Experiment #3: TIMIT-590 vs. TIMIT-40 with increased active learning	VI
A.5.4. Experiment #4: VoxCeleb2-600 vs. VoxCeleb2-120	VI
A.5.5. Experiment #5: VoxCeleb2-600 vs. VoxCeleb2-120 with modest active learning	VI
A.5.6. Experiment #6: VoxCeleb2-600 vs. VoxCeleb2-120 with increased active learning	VI
A.6. Resources on USB Flash Drive	VI

1. Introduction

In audio recordings it is often not clear *who* is speaking *when*. *Speaker diarization* tries to solve this by dividing the recordings up into smaller pieces. These pieces only contain one active speaker each and are grouped (or clustered) together with other pieces of the same speaker. This clustering step is referred to as *speaker clustering*. Usually, the speakers are unknown and the algorithm is unaware of the number of total speakers.

Other tasks in the *speaker recognition* field are *speaker verification*, the task of deciding if the speaker in a recording has the same identity as a known speaker and *speaker identification*, where a recording is used to match it against multiple known speakers. In both of these cases, the algorithm is working with a preset of known speakers to compare to and is thus solving a simpler, albeit non-trivial, problem.

The focus of this work lies on the task of *speaker clustering* using Deep Neural Networks on a recently published large dataset containing more complex data in the form of audio recordings with background noise, as it would be the case in a real-world application.

1.1. Motivation

Deep Neural Networks have been applied successfully to many image processing tasks in the past, examples being the work presented by Deng et al. [4] and Cao et al. [1]. This suggested that these types of neural networks may also be successful in various other areas of data processing where the underlying data may be interpreted as an image to leverage progress made in the field of image processing. Speech processing is one example of such an area, where data (voice recordings) can be converted to spectrograms, an image representation of that data, and then processed as image sources. Due to the lack of publicly available large datasets required to do such tasks, only few, highly specific models have been created. In recent years new, larger and more diverse datasets have become available which led to an increase of experiments in this area, including the topic of *speaker clustering*.

Examples of applications in real world scenarios would be analysing business meetings, identifying who speaks when for how long or protocol and identify sessions in court or political debates. Knowledge gained from such an analysis can lead to higher efficiency and a better understanding of human inter-communication in general. Another popular application are the audio assistants developed by some of the biggest tech companies in the industry. Examples being Apples' Siri, Amazons' Alexa or Google Home. These products also use features like speaker identification for security and privacy purposes or clustering when following a conversation with multiple participants.

The problem is not solved in its entirety, although as stated above, many real world applications benefit from solutions being developed. As datasets get bigger and more complex, the efficient use of data and hardware comes to the fore. Examples of other aspects besides scalability where further improvements can be made would be portability to other datasets and improved algorithms tailored to specific applications.

1.2. Related Work

Before neural networks, speech recognition was approached from a statistical standpoint, using predominantly Gaussian mixture models e.g. Jousse et al. [16] or hidden Markov models as in Gales and Young [8]. Kanagasundaram et al. [17] use identity vectors or i-vectors extracted from a Gaussian mixture model to cluster utterances and report equal error rates (EERs) of 9.66% for 20 second long

utterances and 20.92% EER for 5 second utterances on the NIST 2008 speaker recognition evaluation (SRE) dataset [11].

More recent papers utilize specialized neural network architectures such as convolutional and recurrent neural networks. Current state of the art systems such as Xie et al. [40] are able to perform speaker recognition in the wild, reporting equal error rates (EERs) of 3.22% on the VoxCeleb1 test dataset. Zhang et al. [41] presented similar EERs on American phone data, with their best experiment at 1.91% EER. It should be noted that Zhang et al. [41] is using 64 times more data (70 million utterances versus 1.1 million utterances) and it remains unclear if the system proposed by Xie et al. [40] could narrow the gap with similar amounts of data.

Lukic et al. [23] built the basis for this thesis by applying convolutional neural networks to the studio recording dataset TIMIT, achieving misclassification rates (MRs) of 0.1 for 20 and 0.05 for 40 speakers respectively. Stadelmann et al. [35] continued this work with a recurrent neural network architecture exceeding previous results with a MR of 0.0219 for the same 40 speaker test set. Both focused on the network architecture and not the clustering step, where Hibraj et al. [13] presented a perfect 0.0 MR through the use of dominant sets in the clustering step.

1.3. Goals

The Institute of Applied Information Technology¹ (InIT) at the ZHAW School of Engineering conducted multiple studies into speaker clustering approaches, motivated by the rise of neural networks and their capabilities. These studies include Lukic et al. [23] with their use of convolutional neural networks, followed up by Stadelmann et al. [35] and the use of recurrent neural networks. While these two approaches mainly focused on improving the way data is represented for the clustering step, Hibraj et al. [13] improved how speech segments were clustered through the use of so called dominant sets. All of the mentioned approaches are based on the TIMIT dataset, which is composed of studio recordings and as such represent a very narrow part of audio recordings in the everyday world.

The primary goal of this work is to adapt the existing speaker clustering system to the new VoxCeleb2 [2] dataset and outperform previously achieved results on the well-established TIMIT dataset, which will serve as ground truth for the evaluation. An approach has to be found that is able to handle a much more complex dataset and bridge the gap between studio and real-world recordings.

1.4. Document Structure

The second chapter gives a broad overview on the theoretical principles needed and provide explanations of technical terms.

The third chapter describes one possible approach to solve the given task, the ideas influencing this work and elaborates the reasoning behind decision which have been made during this project.

Results are presented in the fourth chapter containing experiments which were conducted throughout the project.

Lastly, the results are discussed and evaluated in the fifth chapter.

¹<https://www.zhaw.ch/en/engineering/institutes-centres/init/>

2. Theoretical principles

This chapter gives a broad overview on the theoretical principles and knowledge required for this work.

2.1. Clustering

Clustering is the task to process an arbitrary amount of objects and group them together according to a metric such as distance between objects to build groups or clusters of the same type. The aim is to end up with clusters that only contain objects of one type and have as many clusters as there are different types. The amount of clusters is usually unknown and not trivial to calculate, such that many algorithms assume a given number of clusters, independent of the true number of clusters. As can be seen in figure 2.1, clustering algorithms' performance depends on the problem at hand and should be tailored to the underlying data, as presented in Fahad et al. [6].

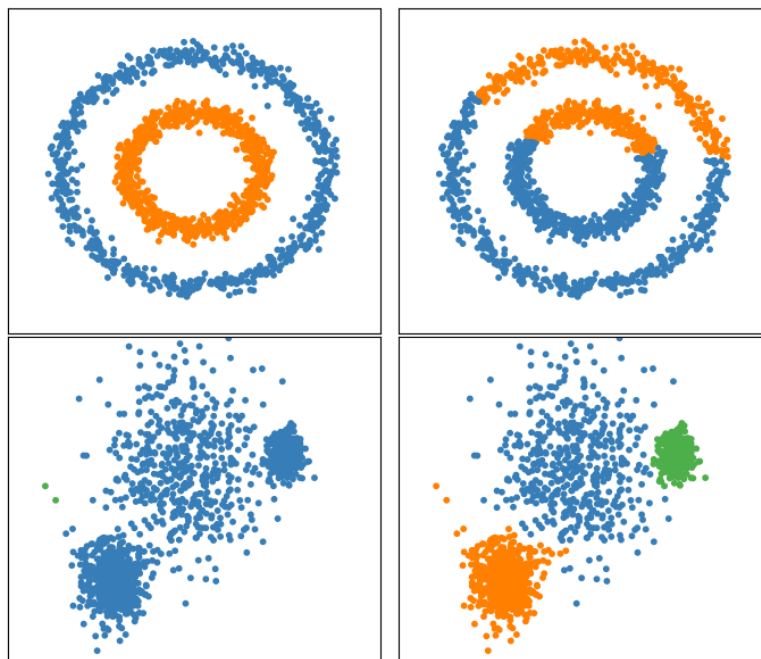


Figure 2.1.: Example of two cluster algorithms on different types of data from scikit-learn.¹

2.2. Speaker Clustering

In speaker clustering, speech segments from unknown speakers are taken and compared to each other with the aim to place them in a high dimensional space and be able to discriminate among individual speakers. Clustering algorithms are used to group speech segments of only one speaker together and

¹The code can be found at https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html.

ignoring other speakers' segments. To be successful, a comparison measure needs to be found that both scores low for segments of the same speaker and high for different speakers. Examples are hierarchical agglomerative clustering in Lukic et al. [23] and Stadelmann et al. [35], the use of dominant sets in Hibraj et al. [13] or the *arccos* function in facial recognition by Deng et al. [4] which was successfully applied to the topic of speaker clustering by Xie et al. [40].

2.3. Embeddings

Embeddings are used as a different representation of data, while adding useful characteristics due to the position of the embeddings in the vector space. The aim of an embedding is to create a comparable representation with low dimensionality for ease of use.

In the case of word embeddings on the word2vec model by Mikolov et al. [24], vectors from one word to another indicate relations that can be observed with other word pairs as well. For example, the vector from the word *France* to the word *Paris* is identical to the vector from *Italy* to *Rome* or *Japan* to *Tokyo*. In this case, this vector can be used to calculate the capital city for any country.

Speaker embeddings, as used in this thesis, are learned by the neural network. During the training period, the network is tasked to perform speaker identification and intrinsically creates such an embedding representation in the hidden layers. This allows the network to transform any spectrogram into a highly generalized representation for any speaker.

2.4. Prosodic Features

The prosodic (or suprasegmental) features of speech as described by Ladefoged [21] are characteristics beyond vowels and consonants that lie "in between segments". They are distinctive for every voice and are perceived subconsciously by humans to distinguish one voice from another. Naturally, not all of those features vary as much as others, such that a person whispering can still be identified as the same person if they were shouting, even though the volume is very different. We summarize the most important features listed and described in [21] and [39] as follows:

- **Pauses** can both be used as a rhetorical device or to structure the spoken word.
- **Pitch** levels change to emphasize meaning, such as increasing towards the end of a question. However, the pitch stays in a similar range for a given speaker.
- **Stress** describes the emphasis given to certain syllables during speech. Typical properties of stress are for example increased length or differences in timbre.
- **Rhythm** is often taken as a prosodic feature to a certain language or regional accents rather than being a mainly speaker specific.
- **Volume** or **Loudness** is either an indication of emotion (e.g. anger) or a reaction to the environment (e.g. whispering when one does not want to be overheard). As such, this feature is expected to vary wildly between different recordings.
- **Tempo** refers the amount of words being spoken within a certain timeframe. This feature, similarly to volume, may have a high variance between segments.

2.5. Spectrograms

Spectrograms are a different representation for audio data. They were introduced in 1946 by Koenig et al. [19]. In contrast to wavelength diagrams where axes are time and amplitude, the spectrogram displays three dimensions: time, frequency and loudness. As seen in figure 2.2, for each combination of frequency and time the corresponding loudness can directly be seen, where the wavelength diagram, seen figure 2.3, contains the same information hidden behind the Fourier transform.

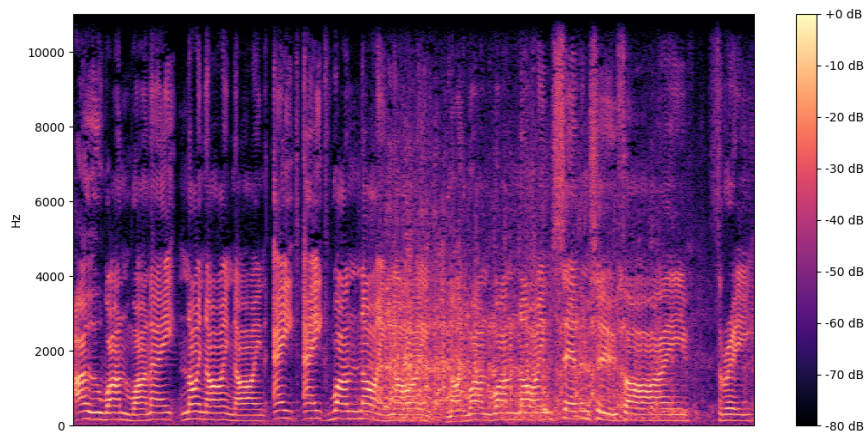


Figure 2.2.: Spectrogram of a sound snippet from a TV-show.²

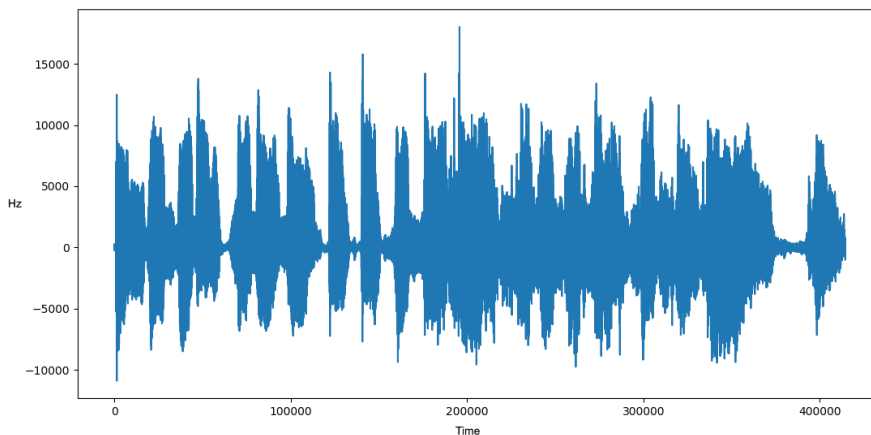


Figure 2.3.: Wavelength diagram of the same file as in figure 2.2.

2.6. Mel Spectrogram

Mel spectrograms represent sound as an image, similar to a regular spectrogram. The sound frequencies are placed along the Mel scale. Stevens [36] introduced the Mel scale in 1937 as follows:

A subjective scale for the measurement of pitch was constructed from determinations of the half-value of pitches at various frequencies. This scale differs from both the musical scale and the frequency scale, neither of which is subjective. (p. 1)

²The sample is taken from the sitcom "The IT-Crowd", series 1, episode 2.

It is therefore closer to the human auditory senses than a linear scaling and resembles how our brain processes sound more closely. Rosen and Howell [28] states that “for auditory signals and human listeners, the accepted range is 20Hz to 20kHz, the limits of human hearing”, while Gelfand [9] defines “hearing is most sensitive in the 2'000 to 5'000 Hz range”. This effect can be observed when comparing figures 2.2 and 2.4. The transformation to a Mel spectrogram effectively reduces the amount of noise in the data by disregarding the parts that would not have been processed by the human brain.

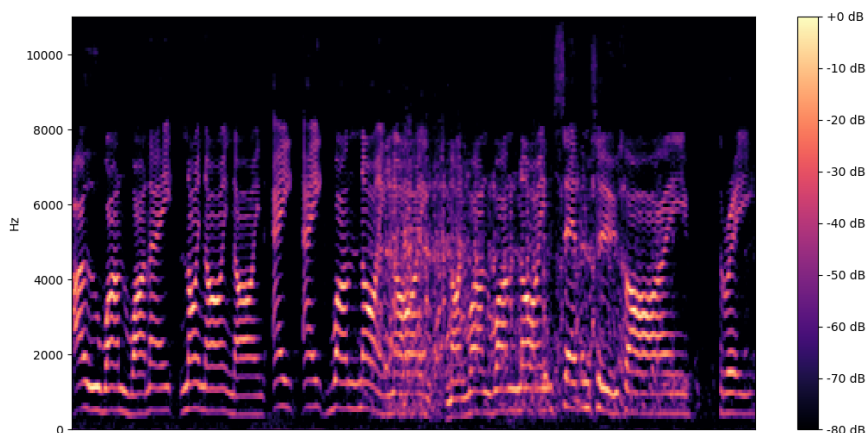


Figure 2.4.: Mel spectrogram of the same sentence as in figure 2.2

2.7. Speech in the Wild

Audio in a real-world scenario does not only consist of the data which is relevant to the task at hand, but also various other information or influences by exterior factors. Some examples of background noise are be cars, trains, other speakers, reverberation effects as well as technical side-effects like a cheap microphone. Unlike professional studio recordings, speakers are not focusing on clear pronunciation in this type of audio recording or may not be voice actors but rather of all ages and professions.

Looking at figure 2.4, the middle part of the diagram seems to merge together compared to other parts of the graph and cannot be separated as clearly. This is caused by background laughter, which can be heard in the audio file and is a fitting example of speech in the wild.

Evaluation on noisy audio data is a focal point of this thesis using a dataset consisting purely of this type of audio recording.

2.8. Neural Networks

Neural Networks (NNs) are a computational model, modelled after neurons in the human brain. They are based on the single neuron perceptron by Rosenblatt [30]. NNs are broken up into different layers of neurons that are interconnected with other neurons of previous and following layers, generally only to directly neighboring layers. There exist no connections between neurons inside the same layer.

NNs can be interpreted as a special case of a directed graph where each neuron is a node and the edges represent the connections from neuron to neuron between layers. NNs have a plethora of variables as each connection has a weight attached to it, while every neuron has a so called bias, which is added to the value of the neuron. These variables need to be evaluated and are crucially important for the performance of the NN.

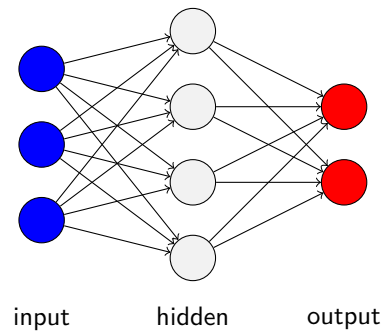


Figure 2.5.: Schematic of a simple neural network with one hidden layer.

Network variables are initialized randomly and are iterated upon during the training step via a process called backpropagation, introduced by E. Rumelhart et al. [5]. This training is done by feeding examples through the network and comparing the networks output with the desired output value. The resulting error is - as the name suggests - propagated backwards through the network, adjusting weights and biases along the way.

The most simple NN has an input layer (the size of the input vector), an output layer with arbitrary size and one layer in between, a so called hidden layer (see figure 2.5). In case of multiple hidden layers, we refer to them as Deep Neural Networks (DNNs).

As described in Lian et al. [22], NN can be interpreted as a one-way function that is constantly improved upon. The usefulness stems from the complexity of the speaker clustering problem, as it cannot be solved by applying simple IF ... ELSE statements. Instead, NNs are trained to learn what exactly distinguishes one voice or speaker from another.

2.9. RNN

Karpathy [18] describes the speciality of recurrent neural networks (RNNs) as follows:

The core reason that recurrent nets are more exciting is that they allow us to operate over sequences of vectors: Sequences in the input, the output, or in the most general case both.

[...]

They accept an input vector x and give you an output vector y . However, crucially this output vector's contents are influenced not only by the input you just fed in, but also on the entire history of inputs you've fed in in the past. (section "Recurrent Neural Networks")

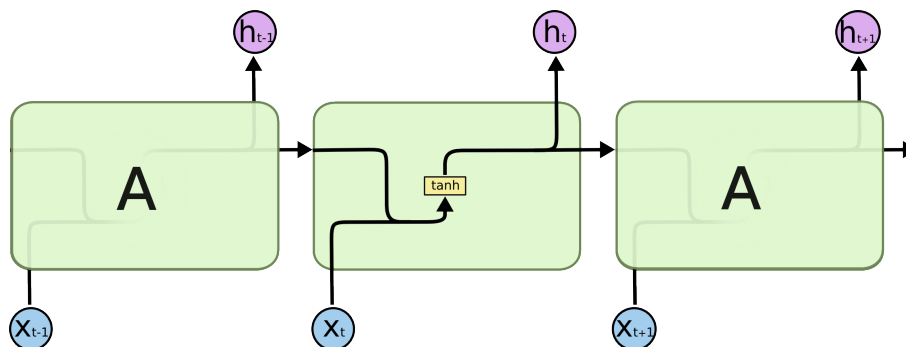


Figure 2.6.: Schematic of multiple RNN cells connected together.[26]

This ability allows the network in the case of speech recognition to take advantage of an intrinsic probabilistic model (as seen in figure 2.9), that for example someone that is talking is likely to still be talking milliseconds earlier and later, or that he is more likely to talk again later than a speaker the network has not seen before.

2.10. LSTM

Long short-term memory networks (LSTMs) were first described by Hochreiter and Schmidhuber [14] as a subtype of, and improvement over RNNs. These networks are special because they can “learn to bridge time intervals in excess of 1000 steps even in case of noisy, incompressible input sequences, without loss of short time lag capabilities” Hochreiter and Schmidhuber [14], p.1.

LSTMs are used as cells that feed into each other, while controlling what needs to be remembered or forgotten. For leveraging this information flow, three logic gates are used: The input, forget and output gates (see figure 2.7, in order from left to right: input, forget and output gates).

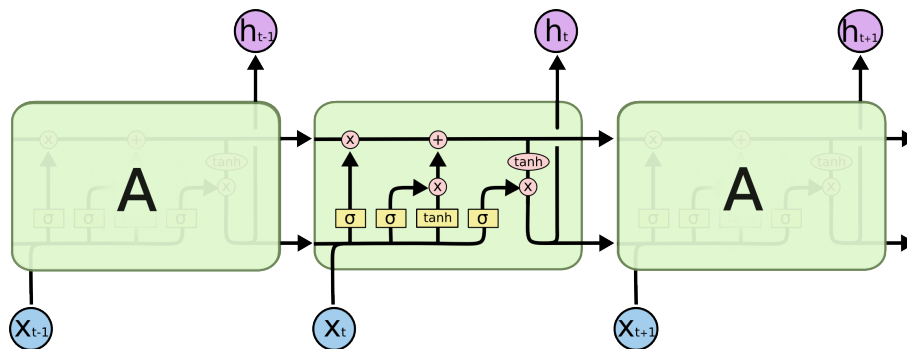


Figure 2.7.: Schematic of connected LSTM cells.[26]

The input gate controls which parts of the information given is remembered for further use. The forget gate decides if something can be forgotten and the output gate controls how the memory is used to calculate its output value. Each of these gates can be seen as separate NNs, trained to specifically accomplish their task. For a more detailed explanation, we refer to the blog post by Olah [26], where the figures are taken from.

2.11. Bidirectional LSTM

While looking back in time, thanks to the LSTM architecture, is helpful, gaining knowledge of the future is equally important. Schuster and Paliwal [31] describe bidirectional RNNs to do just that:

To overcome the limitations of a regular RNN [...], we propose a bidirectional recurrent neural network (BRNN) that can be trained using all available input information in the past and future of a specific time frame. (p. 2675)

A bidirectional RNN (or LSTM) feeds the input sequence both forward and backward through two separate, but mirrored RNN cell chains simultaneously. In fig 2.8 this is visualized for time steps $t - 1$, t and $t + 1$.

$$V(y) = \max\{0, \text{margin} - l(y)\}$$

2.14. Active Learning

Active Learning (AL), also known as query learning, is usually used to reduce the amount of labelled data needed to train a neural network. This is achieved by training a network on a smaller subset with labelled data and then evaluate the remaining data with said network to decide which parts of the unlabelled data are worth investing time and resources to get accurate labels.

This process tries to avoid labelling data that isn't useful for the network to learn on, as manual labelling of data is a highly expensive task.[32]

The algorithm works as follows:

Require: $n \geq 0$ and is set to number of active learning rounds

Require: T contains data **with** labels

Require: P contains data **without** labels

```
function TRAINMODEL( $T$ ) trains the model e.g. using the fit function.
  DoWORK
end function
```

```
function U( $P$ ) returns the least confident predictions for  $P$ 
  DoWORK
end function
```

```
function LABEL( $u_p$ ) returns labelled  $u_p$ 
  DoWORK
end function
```

```
for  $i \leftarrow 1, n$  do
  TRAINMODEL( $T$ )
   $u_p \leftarrow U(P)$ 
   $u_p \leftarrow LABEL(u_p)$ 
   $T \leftarrow T + u_p$ 
end for
```

3. Approach / Methodology

In this chapter, the approach taken to achieve the goal described in section 1.3 is depicted.

3.1. Datasets

TIMIT Acoustic-Phonetic Continuous Speech Corpus [15]

In the TIMIT dataset, each of the 630 speakers is recorded uttering ten sentences. Every sentence is chosen to include a multitude of different phonetic sounds giving a broad spectrum of characteristics for each speaker. All speakers are recorded in their native American English accents, within a range of eight different dialects. The dataset is biased towards male voices, as 70% of the speakers are male and only 30% female. It is important to note that all speakers were recorded in a studio under similar conditions, removing background noise, cross-talk and volume discrepancies.

This is a paid product and cannot be distributed as part of this work.

VoxCeleb2 [2]

Built with a focus on real-world data, the VoxCeleb2 dataset consists of sentences extracted from videos on YouTube.¹ As Chung et al. [2] stated when introducing it in 2018, “the speech segments are corrupted with real world noise including laughter, cross-talk, channel effects, music and other sounds. The dataset is also multilingual, with speech from speakers of 145 different nationalities, covering a wide range of accents, ages, ethnicities and languages.” (p. 1).

It features 6112 different speakers and has less of a gender imbalance, with 61% male and 39% female speakers. For each speaker there are between 21 and 500 utterances in the dataset.

Due to its enormous size, it is not distributed with this work as well.²

Comparison

	TIMIT	VoxCeleb2
Size	0.6 GB	334 GB ³
Unique speakers	630	6112
Male	438 (70%)	3728 (61%)
Female	192 (30%)	2384 (39%)
Utterances	6300	1'128'246
Utterance length	max. 8 seconds	4 - 68 seconds
Languages	American English	Multinational

Table 3.1.: Direct comparison of the TIMIT and VoxCeleb2 dataset

¹The extraction process uses the YouTube search engine with the name of the speaker and the word 'interview' added. Speaker names are taken from the VGGFace2[1] dataset, which includes predominantly actors, athletes and politicians.

²You can download the dataset at <http://www.robots.ox.ac.uk/~vgg/data/voxceleb/vox2.html>.

3.2. Preparation

The existing code and infrastructure of the project was heavily coupled to the TIMIT dataset used by Lukic et al. [23] and Stadelmann et al. [35] in their work. In addition, the code was in an outdated and unstable state. To reproduce the results from Stadelmann et al. [35] a working and stable baseline had to be created first to be able to compare results reliably when running experiments on the new dataset.

3.3. Processing of VoxCeleb2 Data

We compared the existing TIMIT dataset with the new VoxCeleb2 dataset to identify the core components we would have to adapt so we could process the new one, while maintaining compatibility with the previous dataset and networks. Main differences relevant for the processing, were: file format (WAV/M3A), file and folder structure, amount of speakers, amount of files per speaker and variation in length of audio.

Only supporting WAV file format to create and process spectrograms, changing the file format was not a feasible option so the entire VoxCeleb2 dataset was converted to WAV format, tripling its size in the process. For the conversion we used the openly and freely available FFmpeg solution.

Audio file length varies wildly in the VoxCeleb2 dataset, ranging from 4 to 68 seconds. To avoid having to needlessly add padding to all files shorter than the maximum, a *max_audio_length* parameter was chosen at 8 seconds.⁴ For every file in the VoxCeleb2 dataset used, only the first 8 seconds are considered.

The big increase in the amount of data caused *pickle* file format [7] to reach its limits of only allowing file sizes up to 2GB. We incorporated the option for networks to use the *HDF5* file format [12] not only for networks but also for speaker information allowing almost infinitely large files.⁵ The suite now supports both formats and the networks training and testing components can specify the format which should be used. Whilst an improvement, the problem of the dataset being too large by a considerable margin to be stored in a single file or loaded in memory was not solved. We researched ways of splitting the dataset into multiple partitions, allowing us not to keep it in memory as a whole and switch between these partitions dynamically.

3.4. Adapting to VoxCeleb2

The ZHAW_deep_voice⁶ framework [35] was built for use with the TIMIT dataset. Changing the dataset over to VoxCeleb2 quickly proved to be unsuccessful due to the sheer size difference. The 180-fold increase in total utterances (assuming $\mathcal{O}(n)$ complexity) would turn the training period from e.g. a day on TIMIT into half a year on VoxCeleb2, while covering the dataset similarly during training. As this bachelor thesis is time constrained, a more time efficient approach has to be taken. Furthermore, memory is not endless and the training set should ideally be kept in memory as to avoid unnecessary read and write operations.

The following approaches were evaluated and further explored:

- Exploring different loss functions to converge faster
- Handling data more efficiently
- Updating used software to ensure the newest optimizations are used

³The audio files had to be converted from .m4a to .wav, further increasing the total size.

⁴For reference, the longest recordings in the TIMIT dataset last 8 seconds.

⁵Most Unix systems however limit the number and size of files opened, realistically giving a 1TB file size limit. See <https://support.hdfgroup.org/HDF5/doc/TechNotes/BigDataSmMach.html>.

⁶https://github.com/stdm/ZHAW_deep_voice

- Reducing the training data shown to the network

Different loss functions

We initially considered hard example mining and triplet loss due to the promising results of Wang et al. [38] for facial recognition of 100k identities. Facial recognition being a similarly difficult task: the front and side view of the same face look much different than two different front views of a face. Similarly, Deng et al. [4] were able to define a new loss function using a geometric representation with *arccos*, outperforming the state of the art for facial recognition. A fellow master student explored how to apply these loss functions to speaker recognition and we planned on eventually combining our findings. As such we did not pursue this topic further for the scope of this thesis.

Efficient data handling

The framework uses *pickle* files to store the pre-processed training data. The change from *pickle* to *HDF5* boosts reading times by about 17%. During training, the dataset is only read once, so this only brings a negligible benefit. From a memory standpoint however, *pickle* uses about four times as much memory as *HDF5* when reading data from disk. The decreased memory need enables us to increase the amount of data being read with the same RAM constraints. An important note, file sizes for *.h5* and *.pickle* of the same data are within the margin of a few bytes, the argument, *pickle* uses a more advanced compression algorithm that causes a bigger memory footprint when read does not apply.

Software dependency updates

Due to updates in library dependencies, the epoch training times could be decreased by a factor of 2. We believe the update to *tensorflow* version 1.13.1 to be the cause of this increase, as LSTM modules traditionally were implemented without taking advantage of the GPU architecture available. Using *keras'* CuDNNLSTM modules instead of LSTM yielded no measurable performance gain after upgrading *tensorflow*.

Reducing training data size

Trying to achieve similar performance with a smaller subset instead of the full dataset is a problem domain where plenty of research has been made: **active learning**. Normally, scientist prefer to label as few examples in their dataset as possible to achieve great results and through active learning, the network specifies which data needs to be labelled for further training.

Shen et al. [33] showed for named entity recognition that active learning can be applied to conserve the amount of training data used and still achieve 99% of state of the art performance with as little as 25% of training data. Their approach uses a starting dataset, which is constructed as a subset of all data available. After a few training iterations, the network is used to evaluate which part of the remaining dataset is most useful for the current learning progress.

3.5. Active Learning Approach

Our idea to solve the problem of splitting up the dataset was to use an automated active learning approach. Usually this process is applied if a large **unlabelled** data pool and only small amount of labelled data are available and data labelling is applied manually. Shen et al. [33] showed promising results, achieving up to 99% performance of state of the art systems with as few as 25% of training data used.

We partitioned the VoxCeleb2 dataset into smaller data pools, each pool consisting of a smaller amount of data samples from each speaker in the full set. These pools are loaded as an extension to the current processing pipeline training the network on one base pool and then loading the other pools throughout the active learning process. In each active learning round (*ALR*), a different data pool is loaded (round robin) and uncertainty sampling is performed on the pool **as if** there were no labels available. Then the current training set is expanded with the selected data including the labels. Retraining is then applied to the model. This process is illustrated in figure 3.1.

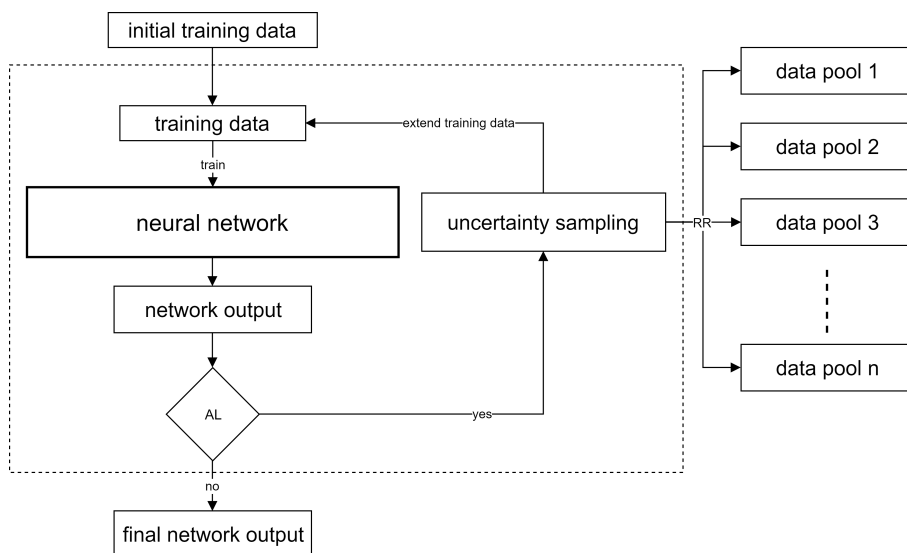


Figure 3.1.: Training process with AL

The uncertainty sampling function (U) is defined by the following formula and used to detect data entries the network is least confident to predict correctly:

$$U(x) = 1 - P(\hat{x}|x)$$

where P is the probability function, " x is the instance to be predicted and \hat{x} is the most likely prediction" (Danka and Horvath [3]). The top n instances with the largest values as samples to be labelled are selected and added to the training set for the next ALR. We set n to 128 in our experiments. An experiment using 20 active learning rounds to retrain the model would therefore be trained on the total $20 * 128 = 2560$ plus the amount of initial samples.

3.6. Neural Network Architecture

The network models built while conducting experiments follow a similar structure. Each network consists of two bidirectional LSTM layers ($ls1$, $ls2$) with a 50% dropout layer in between, three dense layers ($d1$, $d2$, $d3$) with a 25% dropout layer between the first and second dense layer and a softmax layer (ac). The LSTM layers both have a constant width of 256, the first two dense layers are configurable using a *dense_factor* (df) and the third uses the *output_size* (os), which should match the amount of classes the network is expected to differentiate (see figure 3.2).

Mini-batch gradient descent is used to train the network with a batch size of 100. The mini-batches are small snippets with a length $10 * segment_size$ ms, extracted randomly from utterances in the training data. The Adam optimizer is used with standard parameters.

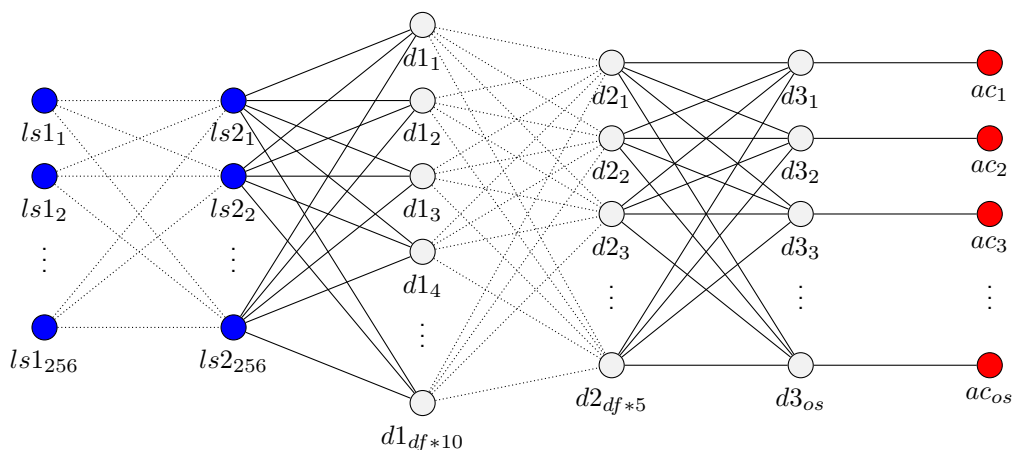


Figure 3.2.: Neural network layout

A combination of Hinge loss and Kullback-Leibler divergence is used as the loss function, outlined in Stadelmann et al. [35], with a margin parameter of 3 (see sections 2.12, 2.13 and the work of Stadelmann et al. [35] for a detailed explanation).

3.7. Misclassification Rate

This metric has been used to measure the speaker clustering performance. It is based on the following formula with w equal to the number of not or wrongly clustered instances and n equal to the total number of instances:

$$MR = \frac{w}{n}$$

For example, if 10 utterances out of a total of 100 utterances are clustered wrongly, the MR is calculated as $10/100 = 0.1$. For a more detailed explanation on how to calculate w and interpret the result, we refer to the work of Niederer and Heusser [25], which defines it thoroughly.

3.8. Homogeneity Score

The ZHAW_deep_voice framework built by Niederer and Heusser [25] uses the homogeneity score defined by Rosenberg and Hirschberg [29]. The original definition states:

A clustering must assign **only** those data points that are members of a single class to a single cluster. That is, the class distribution within each cluster should be skewed to a single class, that is, zero entropy. [...] In the perfectly homogeneous case, this value, $H(C|K)$, is 0. (p.411)

This metric observes how well the system is able to spread out data points of different classes, such that the distance between classes is maximized. For a more detailed explanation, we refer to the works of Niederer and Heusser [25] and Rosenberg and Hirschberg [29].

3.9. Completeness Score

As with the homogeneity score above, the completeness score used is defined by Rosenberg and Hirschberg [29]. The original definition states:

Completeness is symmetrical to homogeneity. In order to satisfy the completeness criteria, a clustering must assign **all** of those data points that are members of a single class to a single cluster. (p.411)

This metric pays attention to how well the system is able to minimize the distance between data points of the same class. For a more detailed explanation, we refer to the works of Niederer and Heusser [25] and Rosenberg and Hirschberg [29].

3.10. Used Software

The software programs listed below have been used during the process of this work. We recommend using a lightweight editor and using standard *pip* package manager to reduce installation overhead.

Environment

- Microsoft Windows 10, Ubuntu 16.04 LTS
- Visual Studio Code, Version 1.33.1
- Python 3.7
- FFmpeg (audio conversion) (ffmpeg.org)
- Inkscape, version 0.92.4

Notable Python modules

We only list the most important modules and not all included dependencies. For a full list consult the *Dockerfile* in the attached source code.

- Tensorflow, Version 1.13.1
- keras Version 2.0.8
- Librosa, Version 0.5.1 (Spectrogram extraction)

Documentation

- Overleaf (LaTeX online collaboration tool), www.overleaf.com

Docker

To run the experiments on the GPU cluster, *Docker containers* are required. Docker is a software program which runs so-called containers as virtual machines where further programs can be executed in a dedicated environment. Dependencies can be defined and will be installed on the fly when the virtual machine is started. It is cross-platform compatible and enables users to work in their encapsulated environment as a super user without influencing the general infrastructure used.

4. Results

4.1. Experimental Setup

Global Hyperparameters

Throughout the experiments we kept certain parameter values unchanged. These values are based on the previous work conducted with this suite [10, 23, 25, 34, 35]. Changing the dataset, applying the new active learning approach, network architecture, and hyperparameter values would have led to incomparable results. We decided that preserving the parameters listed in table 4.1 with values from previous work would reduce the discrepancies between the projects and aid in the acquisition of comparable results.

The following values were used throughout the experiments as constants:

Description	Value
Batch size	100
Learning rate ¹	0.001
β_1 ¹	0.9
β_2 ¹	0.999
ϵ ¹	$1e^{-8}$
Decay ¹	0.0

Table 4.1.: Hyperparameters used in the experiments section

Hardware

Experiments were run on a GPU cluster provided by the ZHAW Institute of Applied Information Technology (InIT). The cluster consists of four compute nodes. On each node there are two *Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz 12 Cores* and eight *Nvidia TITAN Xp GPUs* available. Each of the experiments were run as a slurm job with the *default student QOS*.² For each experiment run, **one** CPU core and **one** GPU were used, allocating **32 GB** of memory.

Total runtime averaged at about ≈ 20 hours for network training. The evaluation process took ≈ 20 minutes on the smaller TIMIT experiments and ≈ 5 hours on all VoxCeleb2 experiments. Evaluation was performed on twelve checkpoints, ten of which are at equidistant intervals, one at the very end of training and another where the validation loss function performed best.

4.2. Datasets

We define the following data subset to be used in the experiments. We used different sets for train and test configurations and did not split a set into two parts. The lists of all speakers in each set can be found in the source code attached³ or on the GitHub repository.

¹Standard parameters are used for the Adam optimizer.

²QOS information can be found at https://info.cloudlab.zhaw.ch/pages/gpu/ressource_limits.html#available-qos.

³Check the README file on the enclosed USB drive for information on where to find these lists

timit_speakers_40_clustering_vs_reynolds_cluster Referring to [35]⁴, this set is distinct from the any other TIMIT sets and is only used during the test phase. It contains 10 utterances per speaker totalling 400 utterances.

timit_speakers_100_50w_50m_not_reynolds_cluster This subset contains both 50 male and female speakers of the TIMIT set and matches the test set used in [35]. It contains 10 utterances per speaker totalling 1'000 utterances.

timit_speakers_590_clustering_without_reynolds_cluster This subset contains all TIMIT speakers except those used in the test set *timit_speakers_40_clustering_vs_reynolds_cluster*. It contains 590 speakers with 10 utterances per speaker totalling 5'900 utterances.

vox2_speakers_120_test_cluster Based on the test set introduced by Chung et al. [2] this set contains 120 speakers with a total of 36'237 utterances.⁵ In this set, the number of utterances per speaker varies.

vox2_speakers_5994_dev_600_base We compiled a list of the 600 speakers with the most utterances from VoxCeleb2 to use as a training set. It contains 445 utterances per speaker totalling 118'815'000 utterances.

This dataset was split into 20 equal partitions of 23 utterances each for every speaker, with the exception of the last partition containing only 8 utterances. Per partition there are $23 * 600 = 13'800$ utterances, and for the last one $8 * 600 = 4'800$.

The decision, to only use 600 speakers, was made to have a number of speaker reflecting the larger TIMIT 590 closely and to reduce the memory footprint required to process. We planned on expanding the dataset step by step to ensure comparability, which ultimately could not be pursued because of poor performance on the existing set and the time constraints of this work.

To use the desired dataset, two variables *train_data* and *val_data* have to be changed directly in the controller python file. These hardcoded values were retained since the *speaker_factory* class also contains these hard-coded values and they refer directly to each other. Unlike other parameters these variables are strictly bound to available and set up sets in the solution.

4.3. Experiments

In this chapter, an overview of each experiment conducted and evaluations of the results will be given. A total of 13 experiments were run, 6 of which will be elaborated in detail. In the following sections, the remaining experiments will be discussed as an ensemble, as primarily multiple *segment_sizes* and *out_layers* for the speaker embeddings were compared against each other, following Stadelmann et al. [35] and their search for a sweet spot both for the layer from which to extract speaker embeddings from and the length of the *segment_size*.

For evaluation, speaker clustering is done on two utterances per speaker. These two utterances are created using 80% and 20% of all available utterances for that speaker respectively. The network is then expected to match each pair of speaker utterances, why the misclassification rate, completeness score and homogeneity score plots all range from *zero* to *twice the number of speakers*. Under ideal conditions, the network should perform optimally where the true number of speakers is positioned.

To ensure a comfortable reading experience, each experiment is displayed on two separate pages. The first page contains the setup, reasoning why we ran the experiment and its result. The second page contains graphics or plots supporting the reader in understanding the results visually.

⁴A *timit_* prefix was added to the existing speaker lists to distinguish them more easily from the VoxCeleb2 sets.

⁵Identity metadata can be found on <http://www.robots.ox.ac.uk/vgg/data/voxceleb/vox2.html> under the metadata section. The 120 speaker test set contains all speakers with the value *test* in the *Set* column. On the official page the set is reported to contain 118 speakers, while there are 120 identities given. We assume this is because only 118 of the speakers contain audio files and the remaining two are empty. To reflect the total number of speakers that are defined in the metadata set, we decided to name it 120 instead of 118.

4.3.1. Experiment #1: TIMIT-100 vs. TIMIT-40

Setup

Configuration entry	Value
Training dataset	timit_speakers_100_50w_50m_not_reynolds_cluster
Test dataset	timit_speakers_40_clustering_vs_reynolds_cluster
Segment size	40
Dense Factor	100
Output Size	100
Output Layer	2 (first dense layer)
Epochs trained (total)	1000
Active learning	deactivated

Table 4.2.: Setup for Experiment #1: TIMIT-100 vs. TIMIT-40

Reasoning

This experiment tries to recreate the results from Stadelmann et al. [35] and Niederer and Heusser [25] by using their ZHAW_deep_voice⁶ framework with as few adjustments as possible to get a ground truth from which further experiments are evaluated. For comparability the same metrics and plot types as by Niederer and Heusser were used. The configured hyperparameters are shown in table 4.2.⁷

Since only the pre-processing steps were adjusted to use the *hdf5* format instead of the *pickle* format, the same result as Stadelmann et al. and Niederer and Heusser reported, is expected to be reproduced.

Result

Misclassification rate, completeness score as well as homogeneity score could be reproduced with nearly identical results⁸ when compared with Niederer and Heusser [25]. The outlier curves after 100 and 200 epochs respectively can be easily spotted where the network clearly was still in process of coarse tuning, while the MR improved almost constantly with additional epochs. This experiment reaches a plateau somewhere around 800 epochs where it assumes 41 clusters for the 40 different speakers, exactly as observed in Stadelmann et al. [35]. This experiment confirms that the network itself did not change through the addition of active learning code. See figure 4.1 for a visual representation of the result.

⁶The repository can be found on https://github.com/stdm/ZHAW_deep_voice

⁷The commands used to start the experiments can be found in the appendix section A.5.

⁸We refer to figure 11 of their work, found on page 21.

Plot

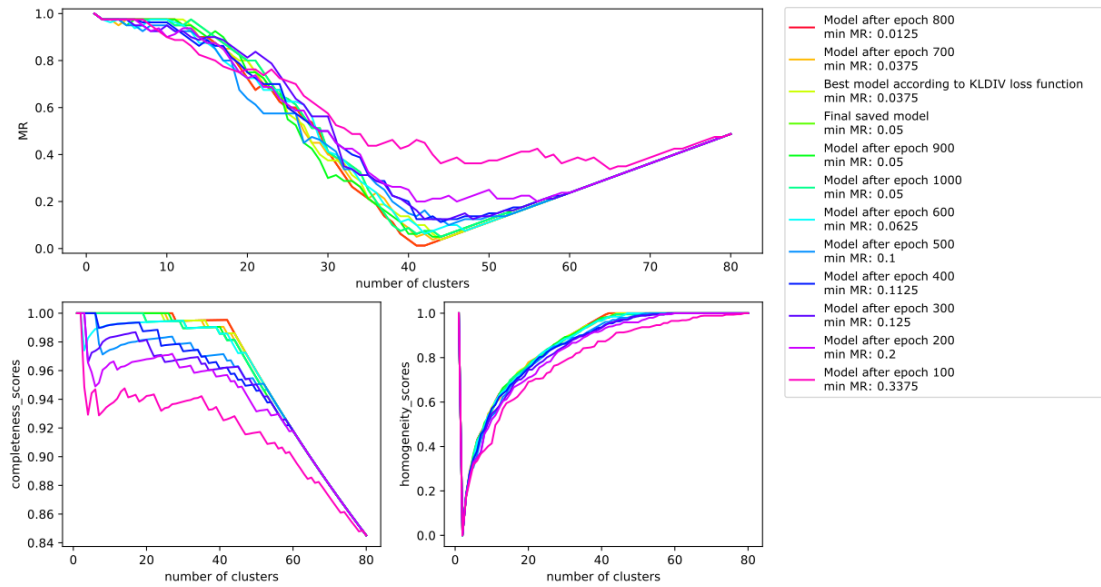


Figure 4.1.: Plot for Experiment #1: TIMIT-100 vs. TIMIT-40

4.3.2. Experiment #2: TIMIT-590 vs. TIMIT-40 with modest active learning

Setup

Configuration entry	Value
Training dataset	timit_speakers_590_clustering_without_reynolds_cluster
Test dataset	timit_speakers_40_clustering_vs_reynolds_cluster
Segment size	40
Dense Factor	100
Output Size	590
Output Layer	2 (first dense layer) and 3 (second dense layer)
Epochs trained (total)	1000
Active learning	# of epochs before active learning starts: 200 # of rounds: 16 # of epochs per round: 50

Table 4.3.: Setup for Experiment #2: TIMIT-590 vs. TIMIT-40 with modest active learning

Reasoning

As with the previous experiment, Niederer and Heusser [25] settled on a 100 speaker training set for their best results and were not able to fully use the whole TIMIT dataset. In this experiment, we explore how the active learning approach differs from their result. Due to the increased size and number of speakers in this training set, it is expected that this experiment to perform at least similarly to Experiment #1: TIMIT-100 vs. TIMIT-40.

Result

As in all of the previous work[23, 25, 35], this experiment performs worse than training on the smaller dataset. Niederer and Heusser [25] trained a different network architecture on this dataset and achieved a MR of 0.625, while Lukic et al. [23] reported as low as 0.05 and a MR of 0.125 with the speaker embeddings extracted at the matching layer for this experiment. For the first dense layer used to extract the speaker embeddings, we achieved similar results to Lukic et al. [23], resulting in a MR of 0.1125 in this experiment. When extracting embeddings from the second dense layer however, the lowest MR reported was 0.0625 outperforming the expected result in this scenario (see figure 4.2).

It is noteworthy that even the very first checkpoint at 100 epochs (min. MR of 0.25) is already performing much better than its counterpart in the previous experiment (min. MR of 0.3375) on the 100 speaker training set. At this point, the network is still training on the initial base training set and has not been increased through the active learning rounds.

Plot

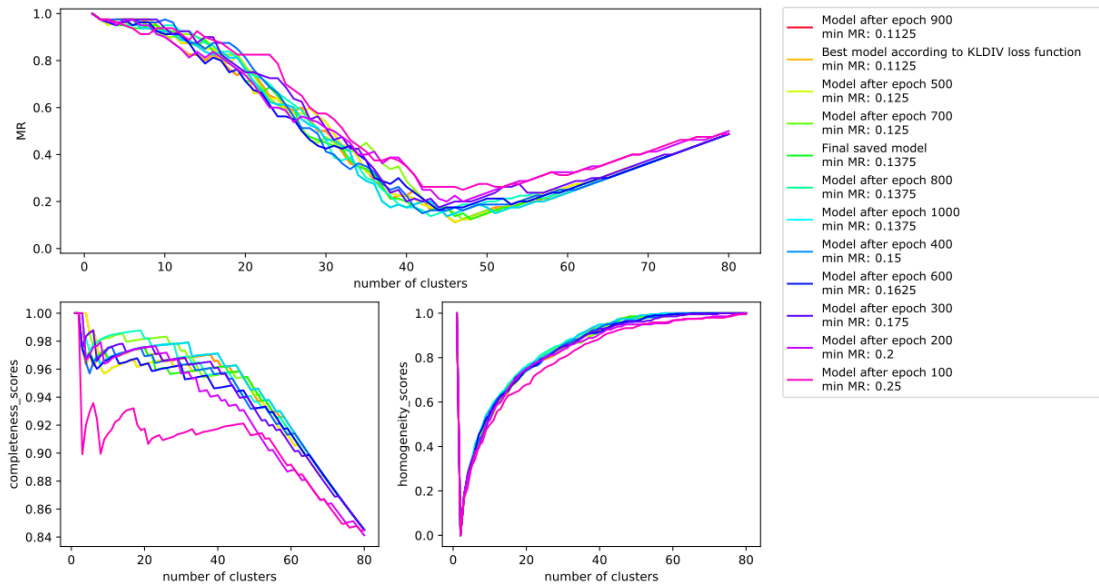


Figure 4.2.: Plot for Experiment #2: TIMIT-590 vs. TIMIT-40 with modest active learning

4.3.3. Experiment #3: TIMIT-590 vs. TIMIT-40 with increased active learning

Setup

Configuration entry	Value
Training dataset	timit_speakers_590_clustering_without_reynolds_cluster
Test dataset	timit_speakers_40_clustering_vs_reynolds_cluster
Segment size	40
Dense Factor	100
Output Size	590
Output Layer	2 (first dense layer) and 3 (second dense layer)
Epochs trained (total)	1000
Active learning	# of epochs before active learning starts: 50 # of rounds: 38 # of epochs per round: 25

Table 4.4.: Setup for Experiment #3: TIMIT-590 vs. TIMIT-40 with increased active learning

Reasoning

The results from Experiment #2: TIMIT-590 vs. TIMIT-40 with modest active learning show at least comparable results to previous experiments on the 590 speaker training set. Here the pre-training period without active learning rounds is decreased and each active learning round lasts fewer epochs. Adding more useful data earlier is expected to improve the performance of this network quicker and ultimately outperform the previous experiment due to the fact that the network is able to decide more often which samples help the most in its learning process.

Result

Compared to the previous experiment Experiment #2: TIMIT-590 vs. TIMIT-40 with modest active learning, this network is able to almost cut the MR in half culminating at a value of 0.0625 (see figure 4.3), which confirms our expectations that more frequent active learning rounds help with model performance. The performance of early checkpoints varies more, so that these checkpoints perform worse than before indicating that shorter pre-training periods decrease the quality of early active learning rounds. Likely, a balance of *just enough* pre-training needs to be found for optimal use. Similarly to the results previously observed when using the second dense layer to extract the embeddings, this experiment also yielded a MR of 0.0375 (see figure 4.4), exceeding the 0.050 MR result from Lukic et al. [23] and showing the same tendency as the previous experiment, that with a larger dataset, extracting the speaker embeddings from deeper layers boosts the overall performance.

Plot

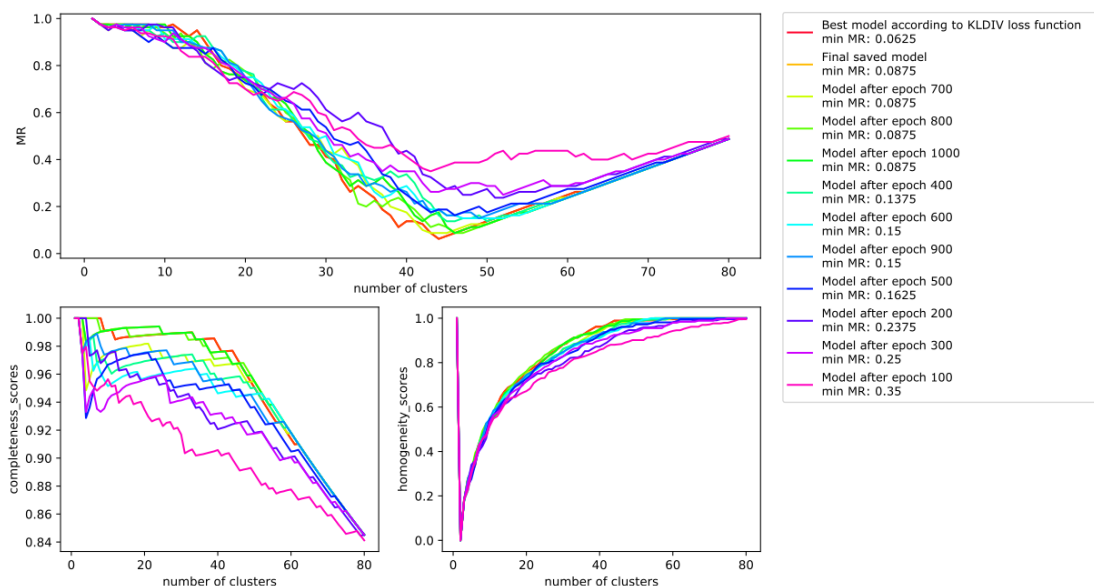


Figure 4.3.: Plot for Experiment #3: TIMIT-590 vs. TIMIT-40 with increased active learning (embeddings extracted from the first dense layer)

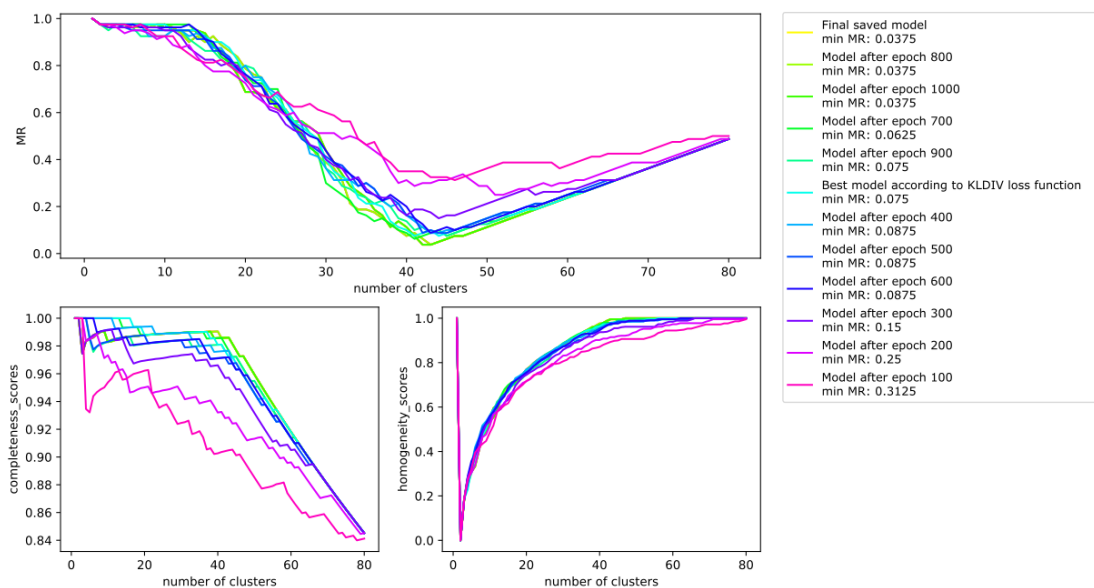


Figure 4.4.: Plot for Experiment #3: TIMIT-590 vs. TIMIT-40 with increased active learning (embeddings extracted from the second dense layer)

4.3.4. Experiment #4: VoxCeleb2-600 vs. VoxCeleb2-120

Setup

Configuration entry	Value
Training dataset	vox2_speakers_5994_dev_600_base
Test dataset	vox2_speakers_120_test_cluster
Segment size	40
Dense Factor	100
Output Size	600
Output Layer	2 (first dense layer)
Epochs trained (total)	1000
Active learning	deactivated

Table 4.5.: Setup for Experiment #4: VoxCeleb2-600 vs. VoxCeleb2-120

Reasoning

This experiment only trains on a smaller subset of VoxCeleb2 with 23 files for each of the 600 speakers, similar to TIMIT's 10 files for each of their 590 speakers. It aims to change as little as possible compared to the experiments 4.3.2 and 4.3.3, hence active learning is deactivated. Due to the increased complexity of the dataset compared to TIMIT, it is expected to perform worse in terms of MR. However, we should be able to improve on the MR metric through later experiments with the addition of active learning.

Result

As expected, the network performs much worse with a min. MR in the 0.40 range instead of the MR below 0.10 from earlier TIMIT experiments (see figure 4.5). While the earlier MR plots curves seemed to reach their minimum close to the true number of clusters (in this case at 120), the MR plateaus around the 120 cluster mark but reaches the minimum closer to about 170 clusters. A similar discrepancy can be found in the completeness scores, where the TIMIT experiments were close to 1.00 completeness almost up to the true number of speakers and only then really started to drop monotonously. Instead, the completeness scores drop as low as 0.65 and recover, but never exceed 0.90. This is expected to be caused by the much more complex combination of both speech and non-speech elements in the recordings. The progress of the loss curve (see figure 4.6) did not indicate any overfitting as another potential reason of performance decrease.

Plot

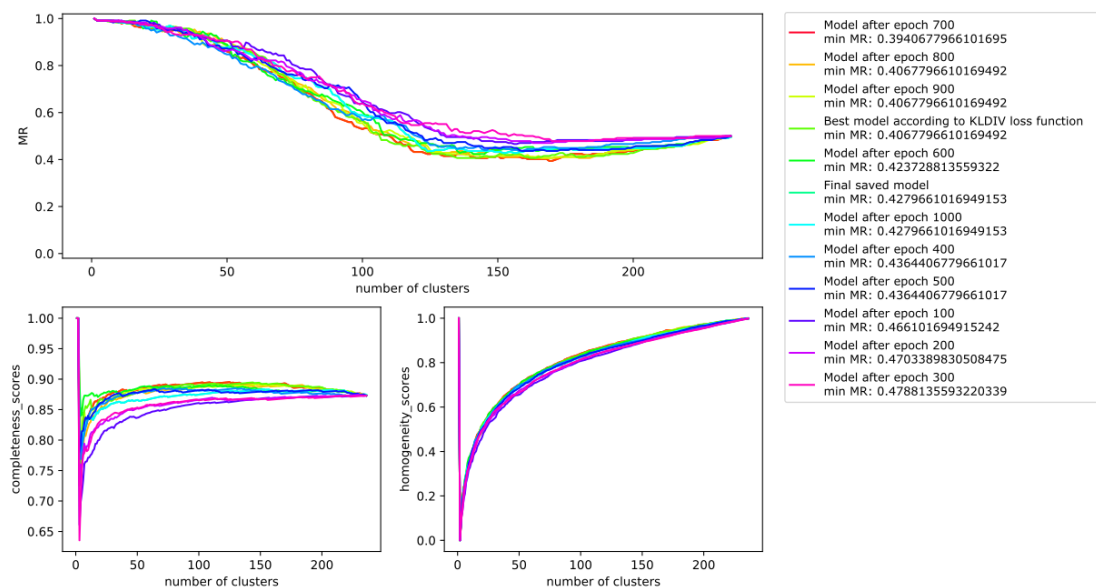


Figure 4.5.: Plot for Experiment #4: VoxCeleb2-600 vs. VoxCeleb2-120

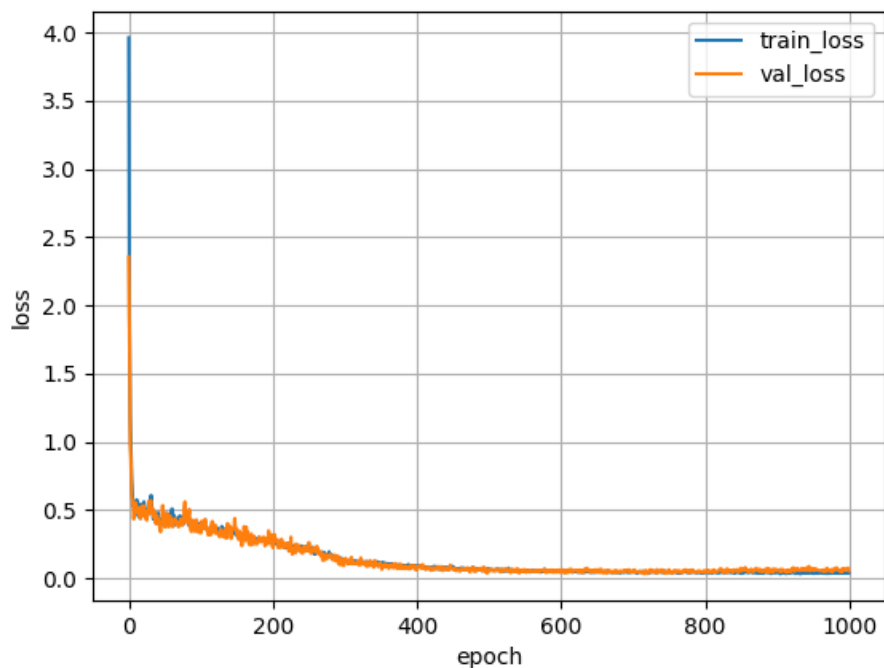


Figure 4.6.: KL-divergence loss for Experiment #4: VoxCeleb2-600 vs. VoxCeleb2-120

4.3.5. Experiment #5: VoxCeleb2-600 vs. VoxCeleb2-120 with modest active learning

Setup

Configuration entry	Value
Training dataset	vox2_speakers_5994_dev_600_base
Test dataset	vox2_speakers_120_test_cluster
Segment size	40
Dense Factor	100
Output Size	600
Output Layer	2 (first dense layer) and 3 (second dense layer)
Epochs trained (total)	1000
Active learning	# of epochs before active learning starts: 200 # of rounds: 16 # of epochs per round: 50

Table 4.6.: Setup for Experiment #5: VoxCeleb2-600 vs. VoxCeleb2-120 with modest active learning

Reasoning

Having a baseline with the VoxCeleb2 dataset, we conducted experiments using the employed active learning process. For a reasonable comparison, the active learning configuration used, is the same as in the previous experiments with the full TIMIT dataset with 590 speakers.

Early checkpoints of 100 and 200 epochs are expected to perform better than without active learning, but are vary of how the increased complexity of this dataset influences the active learning process. Due to the mediocre performance of the previous experiment, a huge performance gain is not expected.

Result

The previously observed performance improvement when using *out_layer* 3 instead of 2 did not prevail in this experiment and further experiments conducted with VoxCeleb2. Because the overall performance of the VoxCeleb2 experiments is no match compared to the TIMIT experiments, it has yet to be confirmed whether for larger datasets a different layer to extract the embeddings would continuously result in a performance improvement. Furthermore, the loss function (see figure 4.8) in line with the min. MR (see figure 4.7, value of ≈ 0.40) across epoch checkpoints indicates potential overfitting on the training data seen; the best min MR performance is achieved around the epoch 400 mark, after which the MR increases again and almost reaches levels of the epoch 100 checkpoint.

Plot

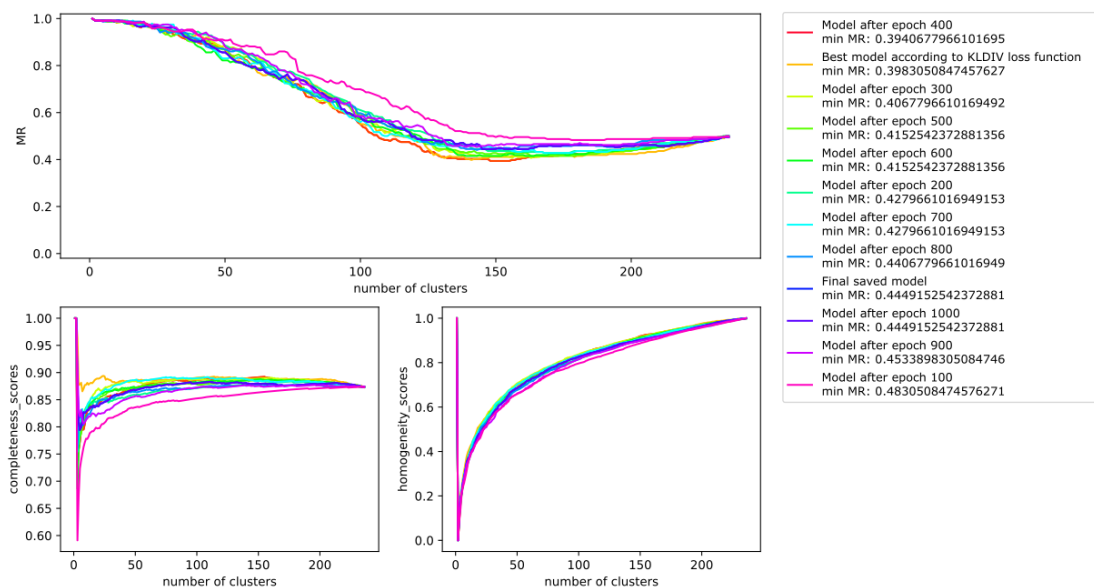


Figure 4.7.: Plot for Experiment #5: VoxCeleb2-600 vs. VoxCeleb2-120 with modest active learning (embeddings extracted from the first dense layer)

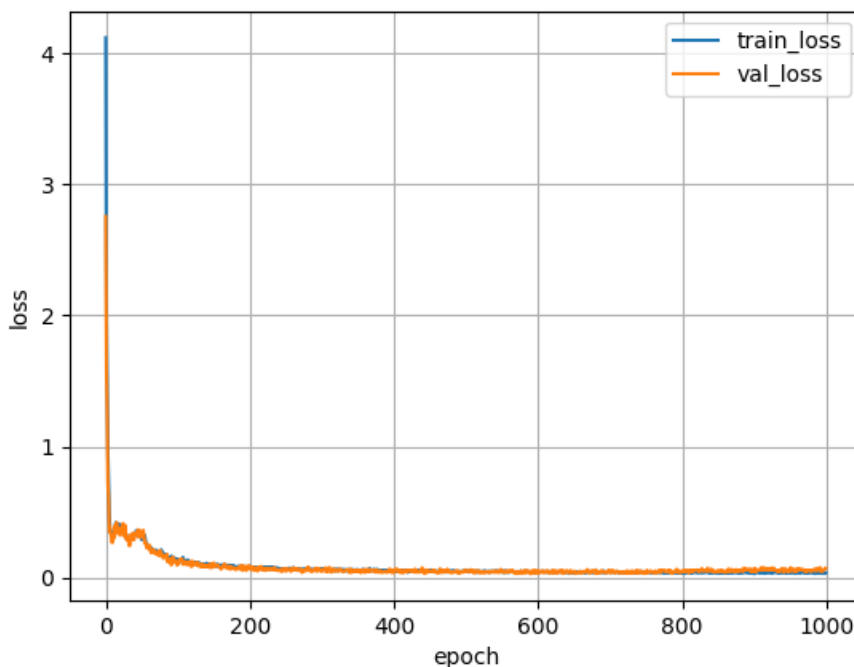


Figure 4.8.: KL-divergence loss for Experiment #5: VoxCeleb2-600 vs. VoxCeleb2-120 with modest active learning

4.3.6. Experiment #6: VoxCeleb2-600 vs. VoxCeleb2-120 with increased active learning

Setup

Configuration entry	Value
Training dataset	vox2_speakers_5994_dev_600_base
Test dataset	vox2_speakers_120_test_cluster
Segment size	40
Dense Factor	100
Output Size	600
Output Layer	2 (first dense layer) and 3 (second dense layer)
Epochs trained (total)	1000
Active learning	# of epochs before active learning starts: 200 # of rounds: 38 # of epochs per round: 25

Table 4.7.: Setup for Experiment #6: VoxCeleb2-600 vs. VoxCeleb2-120 with increased active learning

Reasoning

Based on the previous experiment and analogue to the process used with the TIMIT experiments, the number of active learning rounds were increased in this experiment, shortening the iterative retraining to have comparable result to both experiment #5 in 4.3.5 and experiment #4 in 4.3.4. At the same time, the pre-training period to harness the active learning samples earlier, is reduced. This experiment is expected to once more achieve better results, but the performance difference between TIMIT and VoxCeleb2 may well ignore the benefits of active learning.

Result

This experiment performs *much* worse. Contrary to every other experiment conducted, the MR, completeness and homogeneity plots are very close together across all epoch checkpoints (see figure 4.9). It is evident that this experiment failed, as there is barely any change in min. MR between epochs.

Comparing the loss function to both previous VoxCeleb2 experiments, the starting plateau is roughly twice as big (1.0 versus 0.5) and does not decrease as consistently. While the loss function starts decreasing around the 50 epoch mark where the first active learning round begins, this can be also accounted for by the way the KL-divergence is calculated, as the value of this function decreases with the amount of files in the training set.⁹

The network is clearly overfitting after epoch 400, where the validation loss starts diverging from the training loss (see figure 4.10). The reason for this could be a particularly bad initial state, such that the network was not able to pick useful examples through uncertainty sampling and did not possess the ability to recover. When rerunning the experiment, the network was still overfitting, but not as terribly as in the initial run. This supports our assumption of the network being in a bad initial state when starting the active learning process. The min. MR value dropped to ≈ 0.41 improving from the previous ≈ 0.5 albeit still bad.

⁹For more details see section 2.12.

Plot

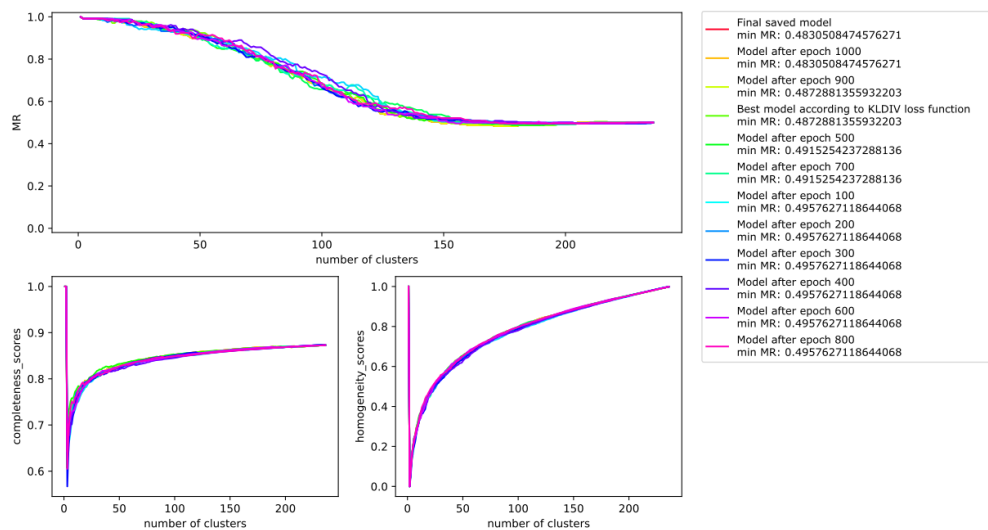


Figure 4.9.: Plot for Experiment #6: VoxCeleb2-600 vs. VoxCeleb2-120 with increased active learning (initial run)

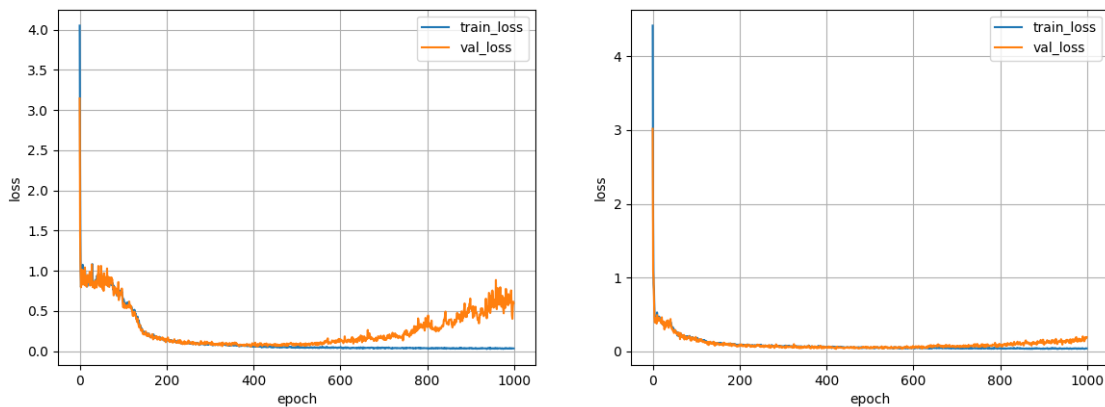


Figure 4.10.: KL-divergence loss for Experiment #6: VoxCeleb2-600 vs. VoxCeleb2-120 with increased active learning (initial run on the left, rerun on the right side)

4.4. Segment Length Sweet Spot

Referring to Stadelmann et al. [35], the experiments on the VoxCeleb2 dataset were evaluated according to their *out_layer* and *seg_size* parameters. As figure 4.11 shows, the experiment on VoxCeleb2 achieve the best results with a segment length of 400ms, extracting speaker embeddings from the first dense layer. The second dense layer outperforms the first dense layer only for a segment length of 700ms on VoxCeleb2. On the TIMIT 590 it performs better than the first dense layer overall. For *seg_size* this confirms the *sweet spot* described in Stadelmann et al. [35] at a length of 400ms not only for the TIMIT but also the VoxCeleb2 dataset. A *sweet spot* for the output layer used to extract the speaker embeddings could not be observed, due to contradicting observations from either the large TIMIT dataset opposed to the VoxCeleb2 experiments.

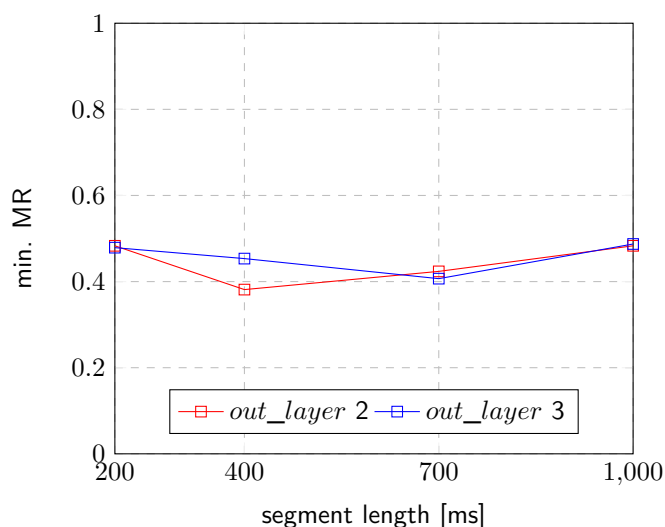


Figure 4.11.: Comparison of different segment lengths and embedding extraction layers for VoxCeleb2, trained on 600 speakers and evaluated against the 120 speaker training set

4.5. Active Learning Comparison

To see how effective active learning can be on both TIMIT and VoxCeleb2 datasets, the experiments show a "no active learning", "modest active learning" and an "increased active learning" setup. The first one gives a baseline result to compare against the other two setups. The second shows the benefits of active learning with a decent pre-learning period and the third setup focuses on applying active learning earlier to increase performance more quickly.

The experiments strongly support a performance increase on the TIMIT dataset. Compared to the network trained by Lukic et al. [23] with a MR of 0.125 with their first dense layer as extraction layer, our active learning approach measures a MR of 0.0625 in Experiment #3: TIMIT-590 vs. TIMIT-40 with increased active learning, exceeding their result by a factor of ≈ 2 . Even more so, if we consider the best extraction layer for both experiments: Lukic et al. [23] were able to report a MR as low as 0.050 while our best experiment on TIMIT Experiment #3: TIMIT-590 vs. TIMIT-40 with increased active learning achieves a 0.0375 MR.

With more frequent active learning rounds (and thus uncertainty sampling), this approach is able to achieve even better results: In Experiment #2: TIMIT-590 vs. TIMIT-40 with modest active learning a MR of 0.1125 could be achieved, while Experiment #3: TIMIT-590 vs. TIMIT-40 with increased active learning was able to beat this metric on the same extraction layer by a factor of ≈ 2 (0.0625).

It should be stated however, that it does not serve as a master key. If the network is not able to reasonably gauge its (un)certainty, active learning is able to theoretically add disadvantageous samples and lead the neural network to a local minimum. One such example would be Experiment #6: VoxCeleb2-600 vs. VoxCeleb2-120 with increased active learning, where in the first run the state of the network seems to be rather faulty compared to previous experiments and ends up overfitting heavily. Rerunning this experiment lead to more reasonable loss behavior and better results, but still performed worse than Experiment #5: VoxCeleb2-600 vs. VoxCeleb2-120 with modest active learning. This might be an indication that the network is not able to achieve a meaningful intermediate state from which to start adding useful samples.

5. Discussion

In this chapter we will review the presented results of this project, discuss if and how the goals defined at the beginning were reached and list possible improvements or further work that should be completed in the future.

5.1. Conclusion

The proposed active learning approach was able to reproduce similar results on the TIMIT datasets compared to previous work [23, 25, 35] and even improve on the results by Lukic et al. [23] for the TIMIT-590 training set. We believe that not only file size and memory issues can be kept in check with the active learning approach, but also faster training times can be achieved in general.

We were unable to achieve a comparable performance on the VoxCeleb2 dataset and believe the following issues to be at least partially responsible:

VoxCeleb2 dataset Unlike the TIMIT dataset, VoxCeleb2 features a wide range of background noise, different recording settings and general non-speech elements. We presume the network is unable to distinguish between speech and non-speech characteristics well enough, causing non-speech aspects to manifest themselves as distinctive features of a particular speaker, e.g. speaker A is often recorded with heavy wind and speaker B with cars driving in the background.

Network complexity The neural network architecture is mostly unchanged compared to the pair-wise_lstm network used in Niederer and Heusser [25], due to the desire to have small changes in-between experiments for ease of comparison. As the conducted experiments show a wide performance gap, an architecture with a vastly increased capacity in terms of deepness or wideness could narrow said gap.

Network components Both state of the art systems forgo the use of LSTM networks and apply a CNN together with ResNet layers with Xie et al. [40] or different RNN types (gated recurrent units) altogether in the case of Zhang et al. [41], where an earlier system using LSTMs was replaced.

Batch size While we are unable to support this statement with a mathematical proof at this point, increasing the amount of classes to learn by a factor of 6 reduces the backpropagation adjustments for every class by the same amount. By that logic, the network is unable to reasonably cover every class as thoroughly as before and optimizes any class it has never seen down to zero. Zero, because the labels are one-hot encoded vectors that *always* contain zero in every row where the speaker has not been seen during training. Tang et al. [37] use bidirectional LSTMs successfully in conjunction with time delay neural networks on 20'000 speakers using a batch size of 256, this suggests that batch size alone cannot be the sole issue.

5.2. Goals

The project aimed to utilize the existing framework used at the ZHAW InIT, adapt it to the new dataset VoxCeleb2 and find an approach to make use of it, outperforming previously achieved results on the established TIMIT dataset. Looking at the results described in chapter 4 this goal could only be achieved partially. Nevertheless, we consider this work to be successful and give the following reasons to support this statement:

- The ZHAW_deep_voice framework can now leverage an active learning approach, surpassing earlier work on the 590 speaker TIMIT dataset.
- VoxCeleb2 can now be used with ZHAW_deep_voice and a baseline could be set for future work.
- Improvements were made to more easily work with new and different datasets.
- Various proposals to increase the performance on VoxCeleb2 and address problems discussed in section 5.1, are presented in the upcoming section 5.4 "Future Work".

5.3. Lessons Learned

During such a project, there are always findings on what could have been done better and lessons learned from that, which we will apply in future work. The following main takeaways are noteworthy and may prove useful for readers as well:

- Test parameters at the start of every function and maybe even use debug logs to visualize their values. This can help a lot in finding incorrectly passed values through many layers of code and to find magic constants.
- Employ sanity checks for your network as early as possible to identify potential errors while changing values incrementally. Rerun the sanity checks after every change.
- Software Engineering tasks are an important part of implementing and testing an algorithm. Do not disregard the time it takes to adapt existing systems to new approaches or algorithms as they might have assumptions built in which hinder a simple transition.
- When running test on external or decentralized infrastructure, make sure to plan spare time for errors, as you have little control over that system and debugging or rerunning experiments may take long. Run sanity checks on the local system beforehand to make sure you waste as little time deploying as possible.

5.4. Future Work

Based on discoveries, takeaways and shown results we recommend the following proposals to further improve the system:

Computation Performance during Training The neural network architecture in tandem with the libraries used is not fully taking advantage of parallelization. During the uncertainty sampling step for example, the neural network model is not changing and could be run on multiple threads simultaneously to speed up the active learning rounds or perform them on bigger subsets.

Decouple Active Learning By the end of this project, the active learning algorithm is coupled to the *pairwise_lstm_vox2* network. The code itself is mostly decoupled and should be moved to the common section or a new modules section. Existing networks could be extended to be configurable to use this approach, using the implementation of the *pairwise_lstm_vox2* network as an example.

Additional Data The active learning approach is capable of dynamically adding more data. Additional data may help covering the overfitting problem and improve the performance. This proposal could not be pursued to the full extent during this project due to time constraints with retraining the models created. It is also limited by physical resources of the machine used for training.

Batch Generator The current implementation(s) of the batch generator(s) require all data given to be held in memory. With a huge dataset such as VoxCeleb2, it will not be feasible in the near future to train on the entire dataset - see section 4.2. A possibility to implement a batch generator which itself handles partitions, gets samples randomly from the disk and passes them to the network should be explored. The change to the *HDF5* file format should alleviate some of the read/write operation performance concerns.

Max Audio Length Data retrieved from audio files is cut off after the configured max audio length. As of this project this value is set to 8 seconds. While in the TIMIT dataset all samples are between 3 and 8 seconds, VoxCeleb2 offers audio lengths between 4 and over 60 seconds. There is a lot of potentially helpful data in later parts of the files. This increases extraction file sizes and memory usage further. If pursued, file size and memory usage once again need to be addressed.

CNN over LSTM With the highly successful application of CNN to the VoxCeleb2 dataset as in its introductory paper by Chung et al. [2] or in Xie et al. [40], both using a ResNet variation, re-evaluating the current architecture or combining both approaches could improve the performance.

Wider and deeper network With the indication of a performance improvement when using the second dense layer for embedding extraction shown in experiments Experiment #2: TIMIT-590 vs. TIMIT-40 with modest active learning and Experiment #3: TIMIT-590 vs. TIMIT-40 with increased active learning, further research on using different layers for that task should be conducted. The possibility to deepen or widen the network should also be explored.

Thinner network Being the complete opposite to the previous proposal, this option could yield improvements nevertheless. Referring to Xie et al. [40] or, together with active learning, Shen et al. [33] thin networks were used to reduce the duration of the training phase. Xie et al. [40] compensated the smaller network with an increased amount of training data used.

Loss Function The use of different loss functions was initially considered but not further pursued. A collaboration with a fellow master student was planned, where he evaluated loss functions for speaker recognition, in particular the *additive angular margin loss* proposed by Deng et al. [4]. While it was shown that this special loss function works well in facial recognition, Xie et al. [40] applied it successfully to speaker recognition as well.

Batch Sizing Identified as a potential performance issue, it might be reasonable to try bigger batch sizes to increase the odds of seeing more classes and avoiding the vanishing effects of absentees. A mathematical model could be created to get a sense for the amount of classes the network is able to see during each batch.

6. Listings

Bibliography

- [1] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *International Conference on Automatic Face and Gesture Recognition*, 2018.
- [2] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. Voxceleb2: Deep speaker recognition. *CoRR*, abs/1806.05622, 2018. URL <http://arxiv.org/abs/1806.05622>.
- [3] Tivadar Danka and Peter Horvath. modAL: Uncertainty sampling, 2018. URL https://modal-python.readthedocs.io/en/latest/content/query_strategies/uncertainty_sampling.html.
- [4] Jiankang Deng, Jia Guo, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. *CoRR*, abs/1801.07698, 2018. URL <http://arxiv.org/abs/1801.07698>.
- [5] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back propagating errors. *Nature*, 323:533–536, 10 1986. doi: 10.1038/323533a0.
- [6] Adil Fahad, Najlaa Alshatri, Zahir Tari, Abdullah Alamri, Ibrahim Khalil, Albert Y Zomaya, Sebti Foufou, and Abdelaziz Bouras. A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE transactions on emerging topics in computing*, 2(3):267–279, 2014.
- [7] Python Software Foundation. Pickle module and file documentation. URL <https://docs.python.org/3/library/pickle.html>. Accessed: 2019-06-02.
- [8] M.J.F. Gales and Steve Young. The application of hidden markov models in speech recognition. *Foundations and Trends in Signal Processing*, 1:195–304, 01 2007. doi: 10.1561/20000000004.
- [9] Stanley Gelfand. *Essentials of Audiology*. Thieme, 2011.
- [10] F. Gishamer and P. Hürlimann. Speaker Recognition & Diarization on Realistic Data. *Project on behalf of ZHAW*, 2018.
- [11] NIST Multimodal Information Group. 2008 nist speaker recognition evaluation test set, 2011. URL <https://catalog.ldc.upenn.edu/LDC2011S08>.
- [12] The HDF Group. HDF5 specifications, 2014. URL <https://support.hdfgroup.org/HDF5/doc/Specs.html>. Accessed: 2019-06-02.
- [13] Feliks Hibraj, Sebastiano Vascon, Thilo Stadelmann, and Marcello Pelillo. Speaker clustering using dominant sets. *CoRR*, abs/1805.08641, 2018. URL <http://arxiv.org/abs/1805.08641>.
- [14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9: 1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.
- [15] William M. Fisher Jonathan G. Fiscus David S. Pallett Nancy L. Dahlgren Victor Zue John S. Garofolo, Lori F. Lamel. Timit acoustic-phonetic continuous speech corpus, 1993. URL <https://catalog.ldc.upenn.edu/LDC93S1>.
- [16] Vincent Jousse, Simon Petitrenaud, Sylvain Meignier, Yannick Estève, and Christine Jacquin. Automatic named identification of speakers using diarization and asr systems. pages 4557–4560, 04 2009. doi: 10.1109/ICASSP.2009.4960644.
- [17] Ahilan Kanagasundaram, Robbie Vogt, David Dean, and Sridha Sridharan. Plda based speaker recognition on short utterances. 06 2012.

- [18] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. 05 2015. URL <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>. Accessed: 2019-06-02.
- [19] W Koenig, HK Dunn, and LY Lacy. The sound spectrograph. *The Journal of the Acoustical Society of America*, 18(1):19–49, 1946.
- [20] Solomon Kullback. *Information Theory and Statistics*. Wiley, New York, 1959.
- [21] Peter N. Ladefoged. Phonetics. 08 2014. URL <https://www.britannica.com/science/phonetics/Suprasegmentals>. Accessed: 2019-06-02.
- [22] Shiguo Lian, Jinsheng Sun, and Zhiqian Wang. One-way hash function based on neural network. *CoRR*, abs/0707.4032, 2007. URL <http://arxiv.org/abs/0707.4032>.
- [23] Yanick Lukic, Carlo Vogt, Oliver Dürr, and Thilo Stadelmann. Speaker identification and clustering using convolutional neural networks. pages 1–6, 09 2016. doi: 10.1109/MLSP.2016.7738816.
- [24] Tomas Mikolov, G.s Corrado, Kai Chen, and Jeffrey Dean. Efficient estimation of word representations in vector space. pages 1–12, 01 2013.
- [25] S. Niederer and B. Heusser. Deep learning für speaker clustering. *Project on behalf of ZHAW*, 2017.
- [26] Christopher Olah. Understanding lstm networks. 08 2015. URL <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. Accessed: 2019-06-02.
- [27] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural Comput.*, 16(5):1063–1076, May 2004. ISSN 0899-7667. URL <http://dx.doi.org/10.1162/089976604773135104>.
- [28] Stuart Rosen and Peter Howell. *Signals and Systems for Speech and Hearing*. Emerald, 2011.
- [29] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. pages 410–420, 01 2007.
- [30] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain [j]. *Psychol. Review*, 65:386 – 408, 12 1958. doi: 10.1037/h0042519.
- [31] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, Nov 1997. ISSN 1053-587X. doi: 10.1109/78.650093.
- [32] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.
- [33] Yanyao Shen, Hyokun Yun, Zachary C. Lipton, Yakov Kronrod, and Animashree Anandkumar. Deep active learning for named entity recognition. *CoRR*, abs/1707.05928, 2017. URL <http://arxiv.org/abs/1707.05928>.
- [34] N. Simmler and A. Trabi. Voice Recognition with Deep Neural Networks. *Bachelor Thesis*, 2018.
- [35] Thilo Stadelmann, Sebastian Glinski-Haefeli, Patrick Gerber, and Oliver Dürr. Capturing suprasegmental features of a voice with rnns for improved speaker clustering. In *ANNPR*, 2018.
- [36] S. S. Stevens. A Scale for the Measurement of the Psychological Magnitude Pitch. *Acoustical Society of America Journal*, 8:185, 1937. doi: 10.1121/1.1915893.
- [37] Yun Tang, Guohong Ding, Jing Huang, Xiaodong He, and Bowen Zhou. Deep speaker embedding learning with multi-level pooling for text-independent speaker verification. *CoRR*, abs/1902.07821, 2019. URL <http://arxiv.org/abs/1902.07821>.
- [38] Chong Wang, Xue Zhang, and Xipeng Lan. How to train triplet networks with 100k identities? *CoRR*, abs/1709.02940, 2017. URL <http://arxiv.org/abs/1709.02940>.
- [39] the free encyclopedia Wikipedia. Prosody (linguistics). URL [https://en.wikipedia.org/wiki/Prosody_\(linguistics\)#Phonology](https://en.wikipedia.org/wiki/Prosody_(linguistics)#Phonology). Accessed: 2019-06-03.

- [40] W. Xie, A. Nagrani, J. S. Chung, and A. Zisserman. Utterance-level aggregation for speaker recognition in the wild. In *International Conference on Acoustics, Speech, and Signal Processing (Oral)*, 2019.
- [41] Aonan Zhang, Chong Wang, John Paisley, Quan Wang, and Zhenyao Zhu. Fully supervised speaker diarization. In *Arxiv*, 2018. URL <https://arxiv.org/abs/1810.04719>.

List of Figures

2.1. Cluster examples	8
2.2. Spectrogram example	10
2.3. Wavelength example	10
2.4. Mel spectrogram of the same sentence as in figure 2.2	11
2.5. Schematic of a simple neural network with one hidden layer.	12
2.6. Schematic of multiple RNN cells connected together.[26]	12
2.7. Schematic of connected LSTM cells.[26]	13
2.8. Structure of a bidirectional RNN.[31, p. 2675]	14
3.1. Training process with AL	19
3.2. Neural network layout	20
4.1. Plot for Experiment #1: TIMIT-100 vs. TIMIT-40	25
4.2. Plot for Experiment #2: TIMIT-590 vs. TIMIT-40 with modest active learning	27
4.3. Plot for Experiment #3: TIMIT-590 vs. TIMIT-40 with increased active learning (embeddings extracted from the first dense layer)	29
4.4. Plot for Experiment #3: TIMIT-590 vs. TIMIT-40 with increased active learning (embeddings extracted from the second dense layer)	29
4.5. Plot for Experiment #4: VoxCeleb2-600 vs. VoxCeleb2-120	31
4.6. KL-divergence loss for Experiment #4: VoxCeleb2-600 vs. VoxCeleb2-120	31
4.7. Plot for Experiment #5: VoxCeleb2-600 vs. VoxCeleb2-120 with modest active learning (embeddings extracted from the first dense layer)	33
4.8. KL-divergence loss for Experiment #5: VoxCeleb2-600 vs. VoxCeleb2-120 with modest active learning	33
4.9. Plot for Experiment #6: VoxCeleb2-600 vs. VoxCeleb2-120 with increased active learn- ing (initial run)	35
4.10. KL-divergence loss for Experiment #6: VoxCeleb2-600 vs. VoxCeleb2-120 with in- creased active learning (initial run on the left, rerun on the right side)	35
4.11. Comparison of different segment lengths and embedding extraction layers for VoxCeleb2, trained on 600 speakers and evaluated against the 120 speaker training set	36

List of Tables

3.1. Direct comparison of the TIMIT and VoxCeleb2 dataset	16
4.1. Hyperparameters used in the experiments section	22
4.2. Setup for Experiment #1: TIMIT-100 vs. TIMIT-40	24
4.3. Setup for Experiment #2: TIMIT-590 vs. TIMIT-40 with modest active learning	26
4.4. Setup for Experiment #3: TIMIT-590 vs. TIMIT-40 with increased active learning	28
4.5. Setup for Experiment #4: VoxCeleb2-600 vs. VoxCeleb2-120	30
4.6. Setup for Experiment #5: VoxCeleb2-600 vs. VoxCeleb2-120 with modest active learning	32
4.7. Setup for Experiment #6: VoxCeleb2-600 vs. VoxCeleb2-120 with increased active learning	34

Glossary


In this section you will find terms and abbreviations with short explanations or references to sections where these can be found.

Abbr	Abbreviation
AL	Active Learning: A process in which the model is used to determine which data samples would be best to achieve better results. Often used to evaluate, what additional data should be labelled.
ALR	Active Learning Round: A single iteration of the active learning training process described in this thesis.
CPU	Central Processing Unit
DNN	Deep Neural Networks, see chapter 2.8
GB	Gigabyte - unit of measurement for disk space or memory. Equals to one billion Bytes.
GMM	Gaussian Mixture Model, a statistical model in which multiple Gaussian distributions are used to represent the total data.
GPU	Graphics Processing Unit
HDF5	A standardized file format, introduced by the HDF Group [12]
HMM	Hidden Markov Model, a probabilistic model using the Markov chains. [12]
InIT	Institute of Applied Information Technology at ZHAW
KLDIV	Kullback-Leibler divergence, see chapter 2.12
LSTM	Long Short-Term Memory: A recurrent neural network (RNN) architecture with some form of intrinsic memory.
M3A	Audio File Format - used to store audio files on a disk
MB	Megabyte - unit of measurement for disk space or memory. Equals to one million Bytes.
NN	Neural Network, see chapter 2.8
PICKLE	File format used to store data, popular in python projects.
QOS	Quality of service, how the individual resource limits are defined on the ZHAW GPU cluster.
RAM	Random Access Memory - temporary memory storage with higher speed than hard drives or flash disks
RNN	Recurrent Neural Network, see chapter 2.9
TIMIT	A speech corpus containing studio recordings. See chapter 3.1
VoxCeleb2	A speech corpus containing real-world recordings. See chapter 3.1
WAV	Waveform Audio File Format - used to store audio files on a disk
ZHAW	Zurich University of Applied Sciences
ZHAW_deep_voice	A framework for speaker clustering, created by Niederer and Heusser [25]

A. Appendix

A.1. Official Project Description

Zürcher Hochschule
für Angewandte Wissenschaften



**School of
Engineering**

Speaker Clustering for Real-World Data using Deep Learning

BA19_stdm_02

BetreuerInnen: Thilo Stadelmann, stdm
Fachgebiete: Bildverarbeitung (BV)
Datenanalyse (DA)
Digitale Signalverarbeitung (DSV)
Software (SOW)

Studiengang: IT
Zuordnung: Institut für angewandte Informationstechnologie (InIT)

Gruppengrösse: 2

Kurzbeschreibung:
Please get in touch (stdm@zhaw.ch) if you are interested in this topic and want to perform exceptional work with a high likelihood for a scientific publication.

Die Arbeit ist vereinbart mit:
Christian Lauener (lauenchr)
Claude Lehmann (lehmac1)

Weiterführende Informationen:
<https://stdm.github.io/research/#voice>

Saturday 01. June 2019 12:01

A.2. Manual Project Setup

We refer to the README docs provided both in the attachment and the GitHub repository, as well as the command line help on how to get this suite up and running locally. All of the steps required are documented in these files.

A.3. Docker setup on the GPU cluster

Docker is used in conjunction with *singularity* to create an executable runtime environment on the GPU cluster provided by ZHAW.¹ Unlike most *docker* containers, this is merely a box in which to execute the Python files.

At the time of this thesis, in June of 2019, the libraries used are mostly up to date and handcrafted to not conflict with each other. Should you be using the code at a later time, make sure that you at least tried running it with exactly these library versions. From experience, the combination of *numpy*, *tensorflow* and *keras* is usually the culprit.

The following steps are needed to build and launch your very own *docker* environment:²

1. Clone GitHub repository from github.com/edualc/ZHAW_deep_voice³
2. Install *docker*
3. Register an account on *dockerhub* at <https://hub.docker.com/>
4. Build the *docker* container using the following command:

```
1 docker built -t <docker_hub_username>/<container_identifier>
```

5. Login to *dockerhub* on the command line:

```
1 docker login
```

6. Create a repository on *dockerhub* with the same name as your `container_identifier`
7. Upload the container to your *dockerhub* repository:

```
1 docker push <docker_hub_username>/<container_identifier>
```

8. Connect to the ZHAW GPU cluster i.e. via SSH

The following commands assume you are connected to the GPU cluster and have a `docker` and `development` folder in your home directory. Feel free to arrange your development files however you prefer, this is just to ensure these command work without path

9. Move to the *docker* folder and download your container:

```
1 cd ~/docker
2 singularity pull docker://<docker_hub_username>/<
  container_identifier>
```

10. Move to the *development* folder and clone the GitHub repository.

```
1 cd ~/development
2 git clone <repo-url>
```

¹See <https://info.cloudlab.zhaw.ch/pages/gpu.html> to access the complete documentation of the ZHAW GPU cluster.

²Most of the steps below can be found on the official ZHAW GPU cluster documentation.

³Be sure to also check the original repository owned by our supervisor Thilo Stadelmann at github.com/stdm/ZHAW_deep_voice, as there is a chance the framework received further updates.

11. Move into the *ZHAW_deep_voice* directory and ensure you have correct project state:

```
1 cd ./ZHAW_deep_voice
2 git branch
3 git status
```

12. Start a *screen*:

```
1 screen -S <your_screen_identification>
```

13. (*Optional*) To see a list of all your active *screen* sessions:

```
1 screen -list
```

14. (*Optional*) To reattach to an existing screen:

```
1 screen -r <your_screen_identification>
```

15. Ensure you are still inside the *ZHAW_deep_voice* directory

16. (*VariantA*) Start your Python environment (this does not start the framework directly and will not shut down when your experiments conclude):

```
1 srun --pty --ntasks=1 --cpus-per-task=1 --mem=32G --gres=gpu:1
  singularity shell ~/docker/<container_identifier>.sing
```

17. (*VariantB*) Start an experiment directly, executing it until completion:

```
1 srun --pty --ntasks=1 --cpus-per-task=1 --mem=32G --gres=gpu:1
  singularity exec ~/docker/<container_identifier>.sing ~/
  development/ZHAW_deep_voice/controller.py <insert args>
```

18. (*Optional*) Terminating the screen (if a program is running, it will be terminated as well):

```
1 exit
```

19. (*Optional*) Key binding for leaving the screen (already running programs such as the framework will keep running):

Ctrl+A, Ctrl+D

IMPORTANT Resources on the GPU cluster are usually sparse. Consider running long experiments only using the `exec` variant *B*, as to free up the resources used as soon as your experiment concludes. If left untouched, variant *A* will only terminated after 10 days! If you are trying to debug however, the `shell` variant *A* is very handy. Just be aware that inside a *screen* only the last fifty or so lines of output are displayed - logging the output of your `srun` command into a log file can make debugging easier.

A.4. Continuing Development

During development, it is best to create small and simple datasets and run the framework locally instead of on the GPU cluster. Call the `controller.py` file as per normal:

```
1 python controller.py <args>
```

Compared to the original *ZHAW_deep_voice* we were given, two main components were heavily modified to change the code to work with our VoxCeleb2 dataset:

Speaker pickle generation (-setup flag) Generation of the *pickle* files for the TIMIT dataset was heavily coupled to the structure of said dataset. Assumptions were made that for every speaker there are exactly 10 utterances. We rewrote most of the code in the `speaker.py` file to accommodate varying utterance numbers and the `speaker_train_splitter.py` used to split datasets. It is also possible to create datasets that are partitioned into multiple files, for more details have a look at `speaker.py` and the arguments used in the `speaker_factory.py`.

Active learning Based on the `pairwise_lstm` controller, a second `pairwise_lstm_vox2` controller exists as a complete copy of the first one. In this folder, files were changed to create compatibility with the new dataset, as well as adding the whole active learning process. Since the active learning using our active learning rounds is hugely independent of the dataset, it should not be hard to extract and apply it to the other existing or completely new network controllers. The dataset assumption for active learning is that there exist at least **two** dataset files named after the dataset identifier (e.g. `vox2_speakers_5994_dev_600_base`) where the base dataset for pre-learning is named `vox2_speakers_5994_dev_600_base.h5` and all the partitions considered during active learning rounds are named `vox2_speakers_5994_dev_600_base_0.h5`, `vox2_speakers_5994_dev_600_base_1.h5`, etc. with an incremental partition index.

There is also a network controller called `lstm_arc_face`, which is based on the research of fellow master student Daniel Neururer. He explored the use of the *ArcFace* loss function introduced by Deng et al. [4] on the same code base using the TIMIT dataset. While many commits on the repository point to changes on this network, the implementations are completely separated.

A.5. Experiment Commands

The experiments were run on the GPU cluster inside a `screen`⁴ using the following commands. To be able to use these commands make sure the folder structure is set up accordingly and you have completed the setup process. Also you will need to adapt the used network controller to be configured for the specific datasets mentioned in each experiments' configuration table.

A.5.1. Experiment #1: TIMIT-100 vs. TIMIT-40

```
1 srun --pty --ntasks=1 --cpus-per-task=1 --mem=32G --gres=gpu:1
  singularity exec ~/docker/zhaw_deep_voice.simg python ~/
  development/ZHAW_deep_voice/controller.py -n pairwise_lstm_vox2 -
  train -test -plot -dense_factor 100 -output_size 100 -epochs_total
  1000 -alr 0 -seg_size 40
```

A.5.2. Experiment #2: TIMIT-590 vs. TIMIT-40 with modest active learning

```
1 srun --pty --ntasks=1 --cpus-per-task=1 --mem=32G --gres=gpu:1
  singularity exec ~/docker/zhaw_deep_voice.simg python ~/
  development/ZHAW_deep_voice/controller.py -n pairwise_lstm_vox2 -
  train -test -plot -dense_factor 100 -output_size 590 -epochs_total
  1000 -alr 16 -epochs_pre_alr 200 -seg_size 40
```

⁴A *screen* is used to keep a session open during training, as disconnecting from the GPU cluster would terminate the session and ultimately stop the execution.

A.5.3. Experiment #3: TIMIT-590 vs. TIMIT-40 with increased active learning

```
1 srun --pty --ntasks=1 --cpus-per-task=1 --mem=32G --gres=gpu:1
  singularity exec ~/docker/zhaw_deep_voice.simg python ~/
  development/ZHAW_deep_voice/controller.py -n pairwise_lstm_vox2 -
  train -test -plot -dense_factor 100 -output_size 590 -epochs_total
  1000 -alr 38 -epochs_pre_alr 50 -seg_size 40
```

A.5.4. Experiment #4: VoxCeleb2-600 vs. VoxCeleb2-120

```
1 srun --pty --ntasks=1 --cpus-per-task=1 --mem=32G --gres=gpu:1
  singularity exec ~/docker/zhaw_deep_voice.simg python ~/
  development/ZHAW_deep_voice/controller.py -n pairwise_lstm_vox2 -
  train -test -plot -dense_factor 100 -output_size 600 -epochs_total
  1000 -alr 0 -epochs_pre_alr 1000 -seg_size 40
```

A.5.5. Experiment #5: VoxCeleb2-600 vs. VoxCeleb2-120 with modest active learning

```
1 srun --pty --ntasks=1 --cpus-per-task=1 --mem=32G --gres=gpu:1
  singularity exec ~/docker/zhaw_deep_voice.simg python ~/
  development/ZHAW_deep_voice/controller.py -n pairwise_lstm_vox2 -
  train -test -plot -dense_factor 100 -output_size 600 -epochs_total
  1000 -alr 16 -epochs_pre_alr 200 -seg_size 40
```

A.5.6. Experiment #6: VoxCeleb2-600 vs. VoxCeleb2-120 with increased active learning

```
1 srun --pty --ntasks=1 --cpus-per-task=1 --mem=32G --gres=gpu:1
  singularity exec ~/docker/zhaw_deep_voice.simg python ~/
  development/ZHAW_deep_voice/controller.py -n pairwise_lstm_vox2 -
  train -test -plot -dense_factor 100 -output_size 600 -epochs_total
  1000 -alr 38 -epochs_pre_alr 50 -seg_size 40
```

A.6. Resources on USB Flash Drive

This document is handed in together with a USB flash drive containing the following information:

- This document as a PDF-File
- A snapshot of the source code by the time this project was concluded
- The stored network models for each of the experiments, including network checkpoints, calculated result pickles and plots
- Scripts used to generate various graphics in this report
- Graphics used in this report