Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

Bachelorarbeit Informatik
Frühling 2017

Speaker Clustering mit Metric Embeddings

Autoren	Timon Gygax Jörg Egli
Hauptbetreuer	Thilo Stadelmann Oliver Dürr

Datum	8. Juni 2017

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Erklärung betreffend das selbständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:

……

Unterschriften:

……...

……

……

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Bachelorarbeiten zu Beginn der Dokumentation nach dem Titelblatt mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Zusammenfassung

Das Gruppieren von Audiosegmenten von gleichen Sprechern wird als *Speaker Clustering* bezeichnet. Die Anzahl verschiedener Sprecher sowie die Sprecher selber sind vorgängig nicht bekannt. *Speaker Clustering*-Verfahren müssen Merkmale von Sprechern identifizieren können, welche unabhängig vom Inhalt der Äusserung sind. Seit einigen Jahren werden Neuronale Netzwerke und insbesondere Convolutional Neural Networks (CNNs) für diese Aufgabe eingesetzt. Die Äusserungen werden dazu in Spektrogramme umgewandelt, womit sie zugänglich für die Verarbeitung durch CNNs werden. Somit können bewährte Methoden zur Mustererkennung angewendet werden.

Das Ziel dieser Arbeit liegt darin, auf bestehende Ansätze aufzubauen und diese zu verbessern. In vorhergehenden Arbeiten wurden verschiedene Netzwerkarchitekturen evaluiert und bilden eine solide Ausgangslage. Zusammen mit einer neuen Loss-Funktion soll ein Modell trainiert werden, welches direkt Embeddings aus den Audiosegmenten erzeugt. Diese Embeddings stellen die Featurevektoren der Audiosegmente dar und werden in einem zweiten Schritt mit klassischen Verfahren geclustert.

Die bestehende Architektur wurde in TensorFlow implementiert. Zusammen mit der neuen Lossfunktion wurde das Netzwerk in Betrieb genommen. Für die Experimente wurde der bewährte TIMIT Datensatz verwendet.

Die Experimente bestätigten, dass dieser Ansatz funktioniert und Resultate erzeugt, welche vergleichbar mit denen aus vorhergehenden Arbeiten sind. Damit eignet sich dieser Ansatz für zukünftige Arbeiten, in welchen die Hinzunahme von nicht annotierten Daten geprüft oder die Netzwerkarchitektur weiter verbessert werden könnte.

Im Rahmen dieser Arbeit wurde das bestehende experimentelle Setup kritisch hinterfragt. Um zukünftig die Reproduzierbarkeit und Vergleichbarkeit solcher Experimente zu gewährleisten, werden neue Definitionen und Konzepte vorgeschlagen. Für die Berechnung der Misclassification Rate (MR) wurde eine klare Definition verfasst, welche keinen Interpretationsspielraum lässt. Zudem wurde mit der Durchschnitts-MR ein neues Mass eingeführt, welches unabhängiger von einzelnen Messpunkten ist. Für die Generierung von Sprecherlisten auf dem TIMIT Datensatz wurde ein Konzept entwickelt, welches erlaubt, Trainings- und Testlisten von besserer Diversität und beliebiger Grösse zu erstellen.

Abstract

The task of clustering audio segments from the same speakers is known as *Speaker Clustering*. Neither the number of different speakers, nor the speakers themselves are known to the system in advance. *Speaker Clustering* techniques need to identify voice-based features of speakers that are not dependant on the content of an utterance. In the past few years, Neural Networks and in particular Convolutional Neural Networks (CNNs) have been used for such tasks. Utterances are converted into spectrograms in order to get processed by CNNs. This procedure allows the use of known pattern recognition methods.

The goal of this work is to improve existing approaches. In recently published work, different network architectures have been evaluated and this evaluation is the basis for our work. These architectures and a new lossfunction are used to train a model which produces embeddings directly from audio segments. The embeddings represent feature-vectors of the audio segments are then clustered in a traditional manner.

The existing architecture has been re-implemented using TensorFlow and extended by the new lossfunction. The well-known TIMIT dataset is used for the experiments.

The experiments show that this approach works and the resulting outcomes suggest comparable results to previous work. The idea has proven suitable and can be pursued in future work. Further improvements could be made with the addition of unlabeled data or by enhancing the architecture.

As part of this thesis we critically examined the existing experimental setup. In order to maintain reliability and comparability of such experiments, we suggest several new concepts. A definition for the misclassification rate (MR) is given which leaves no room for interpretation. Additionally, a new measure called mean misclassification rate, which is independent of individual measuring points, is introduced. For the generation of speaker lists from the TIMIT dataset, we propose a new concept which allows greater diversity and arbitrary size for training- and testing lists.

Vorwort

Nach der Projektarbeit zum Thema *Deep Learning* war für uns klar, dass wir unser angeeignetes Wissen in diesem Bereich und insbesondere TensorFlow auch in der Bachelorarbeit anwenden wollten. Mit der Arbeit zum Thema *Speaker Clustering* fanden wir unser Wunschthema und freuten uns über die Möglichkeit, unser Wissen nochmals zu vertiefen.

Einem Netzwerk „das Lernen“ beizubringen war für uns neu und erforderte mehr als einmal unseren ganzen Einsatz. Nicht immer lag eine Lösung in Reichweite und es galt systematisch vorzugehen und mit viel Beharrlichkeit dem Problem auf den Grund zu gehen. Neben der Implementierung waren die Durchführung der Experimente ein weiterer Schlüsselpunkt. Um die Experimente sauber und reproduzierbar zu definieren und durchzuführen, erforderte es ein genaues Verständnis vom experimentellen Setup, der Funktionsweise der Masse, sowie methodisches Vorgehen. Durch diese intensive Auseinandersetzung konnten wir neue Masse und Konzepte vorschlagen, welche, so hoffen wir, für zukünftige Arbeiten nützlich sind.

Mit Patrick Gerber und Sebastian Glinski befasste sich ein zweites Team mit derselben Problematik. Der Austausch und die zweite Perspektive auf das Problem war für uns immer sehr erfrischend. Zusammen mit unseren Betreuern Thilo Stadelmann und Oliver Dürr waren die gemeinsamen Sitzungen stets zielführend, motivierend und eine Schmiede für viele neue mögliche Ansätze und Ideen.

Abschliessend dürfen wir sagen, dass wir mit unserem Resultat sehr zufrieden sind und nochmals Vieles dazu lernen konnten. Wir hoffen, dass unsere Erkenntnisse in folgenden Arbeiten Verwendung finden und unser Beitrag das Thema *Speaker Clustering* weitergebracht hat. Wir bedanken uns recht herzlich bei Thilo Stadelmann und Oliver Dürr für die bereitwillige Unterstützung und die ausserordentlich angenehme Zusammenarbeit!

Inhaltsverzeichnis

1 Einleitung

In diesem Kapital wird die Ausgangslage sowie die Zielsetzung erklärt. Weiter wird auf die Motivation für dieses Ziel eingegangen und der Aufbau dieser Arbeit erläutert.

1.1 Ausgangslage

Für das Problem des *Speaker Clustering* existieren verschiedenste maschinelle Verfahren. Erst kürzlich wurden Verfahren vorgestellt, welche auf Neuronalen Netzwerken basieren und damit neue Möglichkeiten eröffnen.

Lukic et al. zeigten in ihrer Arbeit [1], dass sich Spektrogramme von Audiosegmenten als Input für Convolutional Neural Networks (CNNs) eignen. Anders als bei klassischen Verfahren extrahieren CNNs die benötigten Features selber aus den Eingabedaten. Dazu wurde ein CNN auf die Identifikation von Sprechern trainiert. Die Repräsentation der Spektrogramme in den Layern wurde als Embedding verwendet und diente als Grundlage für das Clustern. Während diese Methode ähnlich gut wie die klassischen Verfahren funktioniert, vermuteten Lukic et al. Potential zur Verbesserung.

In einer zweiten Arbeit zeigte Lukic et al. [2], dass ein direkteres Lernen von Embeddings gleich gut funktioniert, jedoch weniger Trainingsdaten benötigt. Dazu verwendeten sie die KL-Divergenz als Loss-Funktion. Die Eingabedaten werden zu Paaren kombiniert und mit *Weak Labels* versehen (gleicher Sprecher oder verschiedene Sprecher). Von diesen Paaren wird der Loss berechnet.

Hoffer et al. [3] stellten in ihrer Arbeit eine neue Loss-Funktion vor, welche erfolgreich zum Lernen von Embeddings auf dem MNIST Datensatz eingesetzt wurde. Im Gegensatz zum Ansatz von Lukic et al. [2] werden hier die Daten nicht paarweise verarbeitet. Beim sogenannten *Metric Embedding*, wird versucht die Distanz zwischen den Samples gleicher Klasse zu minimieren und gleichzeitig die Distanz zu klassenfremden Samples zu vergrösseren.

1.2 Zielsetzung

Wir wollen, basierend auf der vorhergehenden Arbeiten von Lukic et al. [1] [2], das Speaker Clustering weiter verbessern. Dazu bilden die Erkenntnisse von Lukic et al. [1] die Grundlage für die Gestaltung des Netzwerkes, den Aufbau der Experimente sowie der Hyperparameter. Ausgehend von der von Hoffer et al. [3] vorgestellten Methode zum Lernen von Embeddings, wollen wir versuchen, diese auf die Spektrogramme unserer Sprecher zu übertragen.

Diese Arbeit soll zeigen, inwiefern sich das *Metric Embedding* für das Clustering von Audiodaten eignet und ob sich damit allenfalls eine Verbesserung der bestehenden Ansätzen erzielen lässt.

1.3 Motivation

In unserer heutigen digitalen Welt fallen täglich Unmengen von Daten an. Immer mehr handelt es sich dabei um Bilder, Videos und Audiodateien, welche nur schwierig zu durchsuchen und indexieren sind. Um diese Datenflut bewältigen zu können, benötigt es zunehmend automatische, maschinelle Verfahren.

Für Audiodateien ist *Speaker Clustering* ein wichtiges Instrument um diese zu analysieren. Damit können wertvolle Metadaten gewonnen werden, welche auch nützlich für eine allfällige Spracherkennung sein können. Durch diesen Schritt werden diese Daten zugänglich für viele andere

subsequente Systeme, wie z.B. die Textsuche.

Maschinelle Verfahren haben die menschliche Leistung in dieser Disziplin noch nicht erreicht. Eine Verbesserung der Verfahren würde sich daher direkt bemerkbar machen und für viele Applikationen einen Mehrwert bringen.

1.4 Aufbau der Arbeit

Kapitel 2 gehen wir auf die Grundlagen ein, welche nötig sind für das Verstehen der vorliegenden Arbeit. Wir führen die Arbeiten ein, welche als Basis für unseren Ansatz dienen und veranschaulichen unser Konzept im Kapitel 3. Zusätzlich stellen wir die verwendeten Datensätze vor. Relevante Implementationsdetails, insbesondere die Loss-Funktion sowie der Aufbau des Neuronalen Netzwerkes, werden Kapitel 4 beschrieben. Der experimentelle Aufbau, angefangen bei den Sprecherlisten, über die Spektrogramme bis hin zur Beurteilung der Resultate, werden in Kapitel 5 erläutert und kritisch hinterfragt. Die gemachten Experimente werden in Kapitel 6 beschrieben und die daraus gewonnenen Erkenntnisse in Kapitel 7 dargestellt. Letztlich reflektieren wir in Kapitel 8 unsere Arbeit, geben einen Rückblick auf die Aufgabenstellung und beschreiben mögliche, weiterführende Arbeiten.

2 Grundlagen

Dieses Kapitel erklärt die Grundlagen dieser Arbeit. Wir gehen bewusst nur auf die spezifische Thematik ein und setzten ein Grundwissen im Bereich *Deep Learning* voraus.

2.1 Speaker Clustering

Als *Speaker Clustering* bezeichnet man die Aufgabe, Audiosegmente gleicher Sprecher zu gruppieren. Dabei ist a priori weder bekannt zwischen wie vielen Sprechern unterschieden werden muss, noch wer diese Sprecher sind. Es handelt sich um ein Mapping-Problem der Schwierigkeit $m : n$. Speaker Clustering ist zu unterscheiden von Spracherkennung (engl. *speech recognition*), dem Übersetzen von menschlicher Sprache in digitalen Text. Um Sprecher anhand ihrer Äusserungen clustern zu können, müssen Merkmale (engl. *Features*) gefunden werden, welche unabhängig vom Inhalt sind [4].

Ein möglicher Anwendungsfall für Speaker Clustering ist die Analyse von Sprachaufnahmen (z. B. Gerichtsverhandlungen, Sitzungen). Neben dem Übersetzen der Aussagen in digitale Texte mithilfe von Spracherkennung und das Aufteilen der einzelnen Äusserungen mittels *Speaker Segmentation* könnten zusätzlich Äusserungen von gleichen Sprechern gruppiert und anschliessend annotiert werden. Eine Software, welche Speaker Clustering erfolgreich umsetzt, müsste kein Vorwissen über die Anzahl oder die Identität der aussagenden Personen besitzen, um Cluster von Äusserungen derselben Sprecher zu bilden. Ein anderes, denkbares Szenario wäre der Einsatz bei Medien wie Filmen, Serien, TV-Sendungen oder Hörbüchern, bei denen beispielsweise die Gesamtsprechzeit pro Sprecher als Metadaten gespeichert und für weitere Auswertungen verwendet werden können.

Zum Speaker Clustering verwandte Aufgaben sind:

- die Sprecherverifikation (engl. *speaker verification*), das Entscheiden ob es sich bei einem Audiosegment um eine bestimmte Person handelt oder nicht. Diese Aufgabe entspricht einer binären Entscheidung, das Audiosegment passt zum Sprecher oder nicht. Die Sprecherverifikation kann zum Beispiel als Zugangskontrolle bzw. zur Authentifizierung eingesetzt werden.
- die Sprecheridentifikation (engl. *speaker identification*), das Zuordnen eines Audiosegmentes zu einem von einer a priori bekannten, endlichen Menge von Sprechern. Diese Aufgabe entspricht einem Mapping-Problem der Schwierigkeit $1 : n + 1$. Die Sprecheridentifikation kann beispielsweise bei der Suche eines Sprechers innerhalb eines vorhandenen Datenpools eingesetzt werden.

Speaker verification, *speaker identification* und *speaker clustering* werden als Teilaufgaben der Sprechererkennung (engl. *speaker recognition*) verstanden [4].

2.2 Convolutional Neural Network

Convolutional Neural Networks (CNN) funktionieren ähnlich wie normale Neuronale Netzwerke. Der grosse Unterschied besteht in der Architektur, welche bei CNNs für die Verarbeitung von Bildern konzipiert ist. Diese Architektur besteht aus einer oder mehreren Faltungen (engl. *convolution*) mit anschliessender Vereinigung (engl. *pooling*).

Klassische Verfahren zur Mustererkennung basieren darauf, dass die gewünschten Muster, welche

man als Features bezeichnet, manuell definiert werden. Mit diesen definierten Features kann anschliessend eine Klassifizierung durchgeführt werden. Interessanter ist die automatische Extraktion der Features durch *Convolutional Neural Network*. Das CNN lernt durch Backpropagation selber, welche Features relevant sind. Die Convolution in den ersten Layern eines CNNs stellt lokale Zusammenhänge her (z.B. Kanten, Muster usw.) und übernimmt dabei die Rolle des Features Extraktor, während die späteren Layer die Klassifizierung vornehmen [5].

Diese Eigenschaft machen CNNs interessant für eine Vielzahl von Anwendungen im Bereich der Bild- und Spracherkennung.

Anwendung

Für einen kurzen Überblick für mögliche Anwendungen von *Convolutional Neural Network* stellen wir eine Auswahl vor:

- Bei Googles AlphaGo, einer Künstlichen Intelligenz, welche das Brettspiel *Go* spielen kann, werden CNNs zur Repräsentation der Konstellation des Brettes verwendet [6].
- Die ImageNet Challenge ist ein Wettbewerb für Objekterkennung in Bildern. Neuronale Netzwerke und insbesondere *Convolutional Neural Networks* trugen im Jahre 2012 zum Wendepunkte in der Objekterkennung bei und dominieren seither [7].
- In seiner Arbeit über Dokumentenerkennung übernahm LeCun et al. den Ansatz von *Convolutional Neural Networks* für die Erkennung von handschriftlichen Zeichen und konnte zeigen, dass keine manuell definierten Features nötig sind [8].
- Abdel-Hamid et al. setzten erfolgreich *Convolutional Neural Network* für die Erkennung von Sprachlauten (engl. *phone recogniton*) ein und übertrafen damit andere Arten von Neuronalen Netzwerken [9].

2.3 Lernformen

Ein Neuronales Netzwerk kann man auf verschiedene Arten lernen lassen. Für unsere Arbeit sind die beiden Lernformen *supervised* und *semi-supervised* relevant.

Supervised

Beim *supervised* (deutsch: überwachtes) Lernen existiert für jede Eingabe eine erwartete Ausgabe. Das Netzwerk lernt, welche Ausgabe für welchen Input erwartet wird und passt sich entsprechend an. Der Trainingsdatensatz besteht also aus Tupel von Eingabedaten und erwartetem Resultat [10].

Beispiel Möchte man ein Netzwerk die Klassifikation von Bilder *supervised* lernen lassen, muss man für jedes Bild die richtige Klassenbezeichnung kennen. Der Datensatz besteht demnach aus Bildern mit ihren zugehörigen Labels.

Semi-supervised

Beim *semi-supervised* (deutsch: halbüberwachtes) Lernen existiert nur für einen Teil der Eingaben eine erwartete Ausgabe. Grosse Datensätze komplett zu annotieren ist sehr aufwendig. Häufig hat man daher einen kleineren, annotierten Datensatz und zusätzlich einen grösseren, nicht annotiertem Datensatz. Das Netzwerk versucht aus beiden den maximalen Nutzen zu ziehen [10].

2.4 Spektrogramm

Audio-Dateien können als Spektrogramme visualisiert werden. Ein Spektrogramm, wie in Abbildung 1 gezeigt, ist eine dreidimensionale Darstellung der Frequenz zu einer gewissen Zeit. Die Intensität jeder Frequenz zu einer bestimmten Zeit wird dabei farblich illustriert. Diese farbliche Darstellung ist im Sinne eines besseren, menschlichen Verständnisses erstellt worden. Für die Verarbeitung von Spektrogrammen innerhalb von Computersystemen werden Graustufen verwendet.

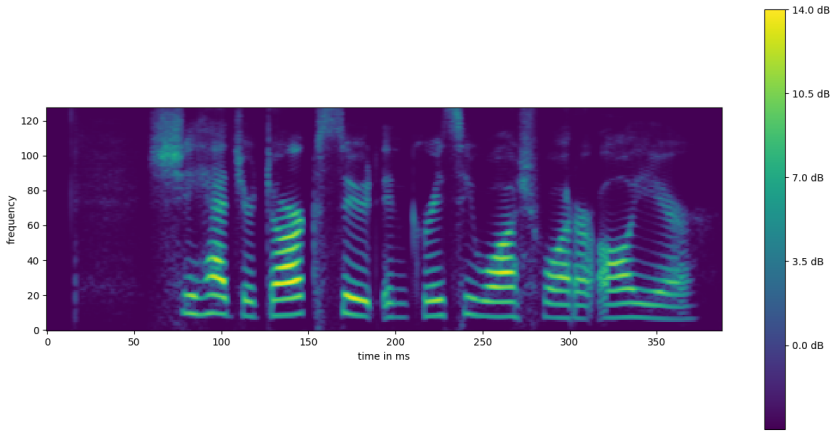

Abbildung 1: Spektrogramm von Sprecherin FLAG0

Die Frequenz-Achse wird üblicherweise nicht in Hertz sondern in Mel (engl. *melody*) angegeben. Bei der Mel-Skala handelt es sich um eine an die menschliche Wahrnehmung angelehnte Frequenzskala [11]. Die logarithmierte Frequenzskala verläuft im Bereich bis 500 Hz nahezu proportional zur logarithmierten Mel-Skala, darüber weichen die beiden Skalen deutlich voneinander ab [12]. Gemäss Shaughnessy [13] beschreibt die Gleichung 2.1 den Zusammenhang zwischen Mel Z und Frequenz f.

$$Z(f) = 1127 \cdot ln(1 \cdot f/700) \tag{2.1}$$

Für die Umrechnung zwischen Hertz und Mel gibt es allerdings verschiedenste andere Funktionen, wobei nicht abschliessend gesagt werden kann, welche dieser Funktionen die menschliche Wahrnehmung am besten widerspiegelt [14].

Mit einer sogenannten *Dynamic Range Compression* wird in einem zweiten Schritt eine elementweise Funktion g, gezeigt in Gleichung 2.2, auf die Spektrogrammwerte x angewendet. Diese sorgt für eine stärkere Ausprägung von wenig prägnanten Werten im hochfrequenten Spektrum. Je mehr dabei die Konstante C erhöht wird, desto prägnanter werden diese Werte [1] [15]. Sowohl Lukic et al. [1] als auch Dieleman et al. [15] haben diesen Wert auf 10'000 gesetzt.

$$g(x) = log(1 + C \cdot x) \tag{2.2}$$

2.5 Embeddings

Für das Clustering müssen Daten in einer Form vorliegen, welche die gewünschten Eigenschaften, nach welchen geclustert werden soll, repräsentieren. Diese Repräsentation wird Embedding genannt und ist ein Featurevektor, welcher durch ein Neuronales Netzwerk erzeugt werden kann [16].

Liegen die Eingabedaten als Bilder vor, werden mit einem *Convolutional Neural Network* die benötigten Features extrahiert und im Embedding dargestellt. Das Embedding ist ein Vektor, welcher für das eigentliche Clustering verwendet wird.

2.6 Clusteranalyse

Unter Clusteranalyse versteht man die Aufgabe, Elemente mit ähnlichen Eigenschaften zu gruppieren. Das Ziel einer Clusteranalyse ist das Finden von neuen Clustern (deutsch: Gruppe). Im Gegensatz dazu werden in der Klassifikation Elemente einem Cluster zugewiesen. Cluster können mithilfe verschiedener Clustering-Algorithmen gebildet werden [17].

Strikt partitionierendes Clustering verlangt, dass alle Elemente genau einem Cluster zugewiesen werden, während bei einem überlappenden Clustering eine Zuordnung zu mehreren Clustern möglich ist. Je nach Anwendung können sogenannte Ausreisser auch keinem Cluster zugeordnet werden [17].

2.6.1 Hierarchisches Clustering

Das hierarchische Clustering wird entweder agglomerativ oder divisiv durchgeführt. Das divisive, hierarchische Clustering (auch *Top-down Verfahren*) geht von einem einzigen, grossen Cluster aus, in welchem sich alle Elemente befinden. Dieser Cluster wird schrittweise aufgeteilt, bis sich schlussendlich jedes einzelne Element in einem eigenen Cluster befindet. Das agglomerative, hierarchische Clustering (auch *Bottom-up Verfahren*) geht von einem Cluster für jedes Element aus und verbindet schrittweise die Elemente, welche am nächsten beieinander liegen, zu einem gemeinsamen Cluster, bis sich schlussendlich alle Elemente in einem Cluster befinden [18].

Mithilfe eines *Thresholds* wird festgelegt an welchem Schwellwert nicht weiter geteilt bzw. verbunden werden sollen. Abhängig vom Threshold entstehen unterschiedlich viele Cluster. Ein Dendogramm, wie in Abbildung 2 gezeigt, visualisiert das hierarchische Clustering. Dabei wurde der Threshold, welcher zu einem optimalen Clustering führt, grün eingezeichnet.

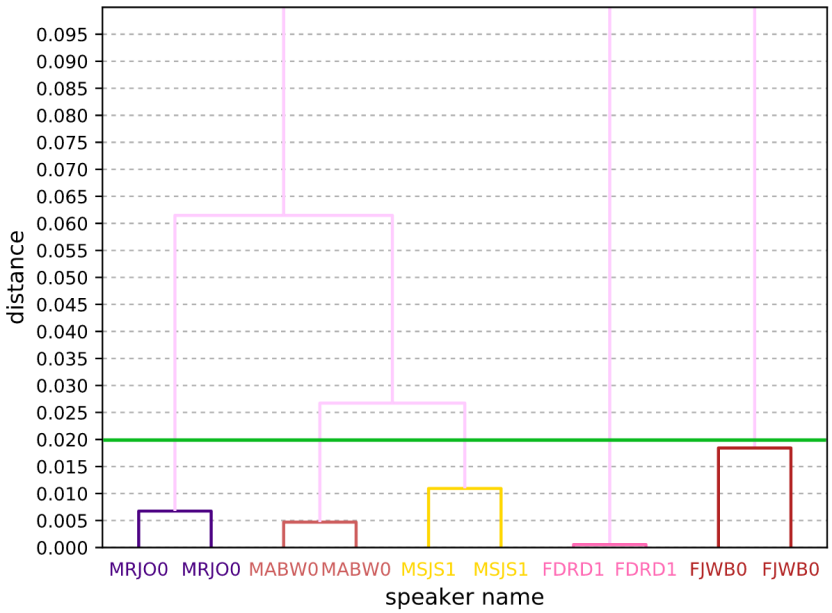

Abbildung 2: Dendogramm eines Clusterings von fünf Sprechern mit je zwei Audiosegmenten. Grün eingezeichnet ist der optimale Schnittpunkt, um die Audiosegmente pro Sprecher zu gruppieren.

3 Konzept

In diesem Kapitel erklären wir, wie sich unser Ansatz zusammensetzt und was die Absichten dahinter waren. Dazu führen wir zuerst die zugrundeliegenden Arbeiten von Lukic et al. [1][2] sowie Hoffer et al. [3] ein und bilden aufgrund dessen in einem zweiten Schritt das Konzept dieser Arbeit.

3.1 Related Work

Unsere Arbeit basiert auf den Publikationen von Lukic et al. über die Verwendung von *Convolutional Neural Networks* und das Lernen von Embeddings für Speaker Clustering [1] [2], sowie Hoffer et al. Arbeit über *Deep Learning* mit *Metric Embeddings* [3].

3.1.1 Speaker Clustering

In ihrer ersten Arbeit untersuchten Lukic et al. [1] die Eignung von *Convolutional Neural Networks* für die Speaker Identification sowie das Speaker Clustering. Zu diesem Zeitpunkt waren klassische Verfahren für die Feature Extraktion (MFFC) und GMM basierte Models üblich [19]. Da *Convolutional Neural Networks* die Fähigkeit, selbständig Features zu lernen und Muster zu erkennen, in einer Vielfalt von Anwendungen erfolgreich demonstriert haben (vgl. Kapitel 2.2), lag es nahe, diese auch bei Spektrogrammen von Audiosegmenten einzusetzen [1].

Lukic et al. trainierte ein *Convolutional Neural Network* zuerst mit dem Ziel der Sprecher Identifikation. Dazu verwendeten sie den TIMIT Datensatz (vgl. Kapitel 3.3.1). Die Audiosegmente wandelten sie in Mel-Spektrogramme von jeweils 128 x 100 Pixel (Frequenz x Zeit) um, welche zusammen mit den zugehörigen Labels als Input für das Netzwerk diente. Für die Loss-Funktion wurde *Cross-Entropy* verwendet und mit dem Minibatch Gradientenverfahren trainiert. Abbildung 3 zeigt den detaillierten Aufbau des verwendeten Netzwerkes [1].

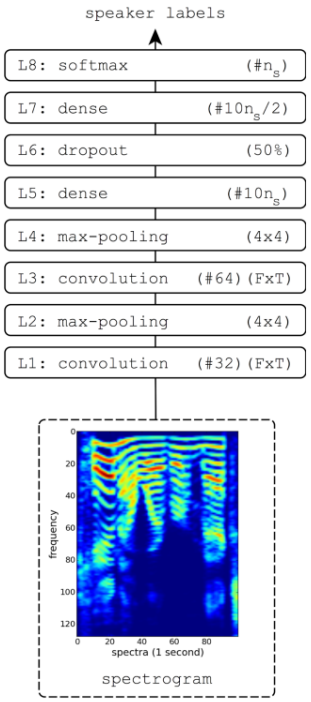

Abbildung 3: Aufbau des Netzwerkes für das Training von Sprecher Identifikation. Nach zwei Wiederholungen von einem Convolutional und Max-Pooling Layer folgen zwei Dense Layer mit einem Dropout Layer dazwischen. Am Ende des Netzwerkes befindet sich der Softmax Layer für die eigentliche Klassifizierung [1].

Das Netzwerk kann nach erfolgreichem Training Audiosegmente bekannten Sprechern zuordnen. Für das Speaker Clustering verwendeten Lukic et al. die Repräsentation der Eingabedaten in einem der Hidden Layer. Die Idee dahinter ist, dass die Hidden Layer auf die relevanten Features trainiert werden, um einen Sprecher identifizieren zu können. Die Hidden Layers repräsentieren demzufolge die Features, welche einen Sprecher ausmachen und eignen sich für das Clustering. Dazu werden beliebige Audiosegmente in das fertig trainierte Netzwerk eingegeben, die Aktivierungen im gewünschten Hidden Layer stellen dann das Embedding (vgl. Kapitel 2.5) dieser Audiosegmente dar [1].

Dieses Netzwerk trainierten sie mit einer 100-Sprecher-, sowie einer 590-Sprecher Liste. Bei dem Testing mit einer 40-Sprecher Liste erreichten sie eine MR von 0.325 für das Netzwerk, welches mit 100 Sprechern trainiert wurde. Das Netzwerk, welches mit 590 Sprechern trainiert wurde, erreicht eine MR von 0.05 mit der gleichen Testliste.

Das Training auf Sprecher Identifikation bezeichnet man als *Surrogate Task*, da das eigentliche Produkt des Netzwerkes (die Klassifizierung) nicht für das Clustering verwendet wird. Stattdessen wird nach dem Training die Repräsentation im gewünschten Hidden Layer als Embedding für den Speaker Clustering Task verwendet und die Klassifierungslayer werden verworfen. Dieser offensichtliche Umweg lässt Optimierungspotential vermuten und so haben Lukic et al. in ihrer zweiten Arbeit [2] Ansätze untersucht, bei denen direkter Embeddings gelernt werden.

Im Unterschied zum Training der Sprecher Identifikation werden bei diesem Ansatz Paare von Audiosegmenten, beziehungsweise von deren Spektrogrammen, gebildet. Diese werden mit sogenannten *Weak Labels* versehen, welche nur angeben, ob die beiden Spektrogramme vom selben Sprecher stammen oder nicht. Pro Paar wird der Loss mithilfe der Kostenfunktion Kullback-Leibler-Divergenz (KL-Divergenz) berechnet. Diese Kostenfunktion hat zum Ziel, die Distanz von gleichen Paaren zu minimieren und vice versa die Distanz von ungleichen Paaren zu maximieren. Abbildung 4 zeigt den Aufbau des verwendeten Netzwerkes. Es basiert grundsätzlich auf

dem vorherigen und wurde mit dem Ziel weitere Verbesserungen zu erzielen um einige Layers ergänzt [2].

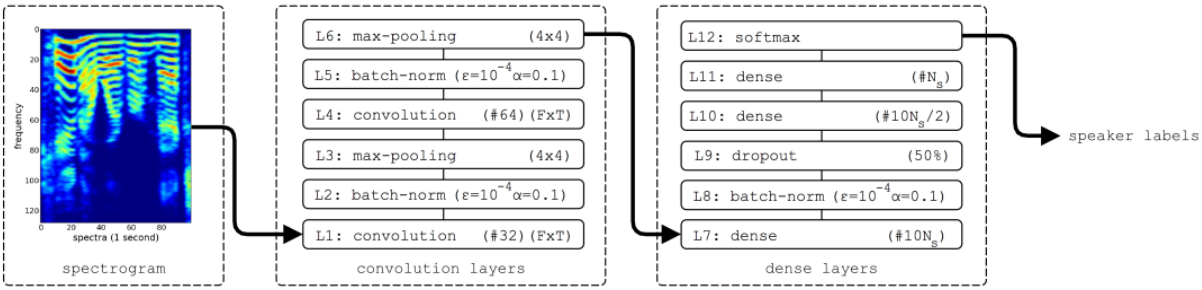

Abbildung 4: Aufbau des Netzwerkes für das Training mit der KL-Divergenz. Im Vergleich zum Netzwerk in Abbildung 3 sind die Batch Normalisierungslayer sowie ein weiterer Dense Layer dazugekommen. [2].

Wie in Abbildung 4 ersichtlich, ist auch hier der letzte Layer ein Softmax. Es werden immer noch nicht direkt Embeddings erzeugt. Die Wahrscheinlichkeitsverteilungen, welche der Softmax Layer erzeugt, werden für die KL-Divergenz benötigt. Die Embeddings, welche für das Clustering verwendet werden, werden nach wie vor aus einem der Hidden Layers gewonnen. Es findet deshalb kein End-to-End Lernen statt. Trainiert wird weiterhin mithilfe eines *Surrogate Task*. Dieser Ansatz ist aber gemäss Lukic et al. deutlich näher am eigentlichen Clustering Problem. Dies resultiert in einer besseren Performance [2].

Die Verbesserungen führten dazu, dass nun für ein Netzwerk, welches mit 100 Sprechern trainiert wurde, ebenfalls eine MR von 0.05 erreicht wird wie beim Testing mit 40 Sprechern. Das sind 83% weniger Trainigsdaten im Vergleich zu ersten Ansatz von Lukic et al. [1]. Aus ihrer Arbeit schliessen sie, dass für zukünftige Arbeiten ein vollständiger *End-to-End*-Ansatz noch weitere Verbesserungen mit sich bringen kann.

3.1.2 Metric Embedding

Damit ein Neuronales Netzwerk generalisiert, auch für unbekannte Daten, ist eine grosse Menge an annotierten Daten nötig. Ist dies nicht der Fall, besteht das Problem des *Overfittings*. Hoffer et al. entwickelten in ihrer Arbeit Lösungen, wie man auch mit wenigen annotierten Daten und einer grösseren Menge von nicht annotierten Daten ähnliche Resultate erzielt. Da nur ein kleiner Teil der Daten annotiert ist, handelt es sich hierbei um ein *semi-supervised* Lernverfahren [3].

Kern der Arbeit von Hoffer et al. ist eine Loss-Funktion, welche jeweils annotierte und nicht annotierte Daten berücksichtigt. Die Loss-Funktion lernt direkt metrische Embeddings (engl. *metric embedding*) aufgrund von Distanzen zwischen den einzelnen Daten. In Gleichung 3.1 sieht man den Loss $L(x_l, z_1, ..., z_c)_L$ für annotierte Daten x_l und den Loss $L(x_u, z_1, ..., z_c)_U$ für nicht annotierte Daten x_u. Die Parameter $\lambda_L, \lambda_U \in [0, 1]$ gewichten die Teile mit den annotierten sowie den nicht-annotierten Daten. $\{z_1, ..., z_c\}$ stellen die Klassenrepräsentanten dar. Aus jeder Klasse wird ein zufälliges Sample ausgewählt, welches diese Klasse für diesen Berechnungsschritt repräsentiert [3].

$$L(x_l, x_u, \{z_1, ..., z_c\}) = \lambda_L L(x_l, z_1, ..., z_c)_L + \lambda_U L(x_u, z_1, ..., z_c)_U \tag{3.1}$$

Da wir in dieser Arbeit ausschliesslich mit annotierten Daten arbeiten, ist für uns nur der erste Summand der Formel relevant und wird hier im Detail beschrieben. Die Gleichung 3.2 zeigt den Loss für annotierte Daten. $F(x)$ stellt dabei das Neuronale Netzwerk dar, x_l ist ein beliebiges, annotiertes Sample. Für jede Klasse I wählt man einen Vertreter z_i aus. z_k ist dabei der Klassenrepräsentant, welcher aus der gleichen Klasse wie x_l kommt.

$$L(x_l, z_1, ..., z_c)_L = -\log(\frac{e^{-\|F(x_l)-F(z_k)\|_2}}{\sum_{i=1}^{c} e^{-\|F(x_l)-F(z_i)\|_2}}) \tag{3.2}$$

Der Dividend widerspiegelt die Distanz zwischen dem aktuellen Sample x_l und dessen Klassenrepräsentant z_k. Das Ziel besteht darin, diese Distanz zu minimieren, da sich diese beiden Samples nachher im selben Cluster befinden sollten. Im Divisor werden die Distanzen vom aktuellen Sample x_l zu allen Klassenrepräsentanten z_i aufsummiert. Diese Summe soll maximiert werden, da die einzelnen Cluster möglichst weit auseinander liegen sollten. Der gesamte Loss L_L wird minimal, wenn der Dividend minimal und der Divisor maximal ist.

Mit diesem Modell erreichten Hoffer et al. [3] State of the Art Resultate auf de MNIST-Datensatz. Damit haben sie gezeigt, dass sich der Einsatz eines *semi-supervised* Lernansatzes, sowie einer Loss-Funktion, welche Embeddings lernt, sinnvoll ist.

3.2 Unser Ansatz

In unserer Arbeit wollen wir den *supervised* - Teil der Loss-Funktion von Hoffer et al. [3] für das Problem des Speaker Clustering verwenden. Da Hoffer et al. in ihrer Arbeit direkt Embeddings gelernt haben, erhoffen wir uns, dass wir damit näher am Problem des Speaker Clusterings sind und auf den Einsatz eines *Surrogate Tasks* verzichten können. Durch diesen *End-to-End*-Ansatz erwarten wir bessere Resultate.

Lukic et al. [1] [2] haben bereits zweimal demonstriert, dass ihre Netzwerkarchitektur für das Training auf TIMIT Spektrogrammen geeignet ist. Daher übernehmen wir die Architektur für die Versuche mit der Loss-Funktion von Hoffer et al [3].

3.3 Verwendete Datensätze

Um unseren Ansatz zu überprüfen, verwendeten wir zwei Datensätze. Zum einen der MNIST-Datensatz, auf welchem auch Hoffer et al. [3] ihren Ansatz trainiert haben, sowie der für Speaker Clustering übliche TIMIT Datensatz.

3.3.1 TIMIT

Der TIMIT-Datensatz ist eine Sammlung von Audioaufnahmen, welche als wav-Dateien vorliegen. Die Aufnahmen wurden in einem Tonstudio aufgenommen und sind von guter akustischer Qualität. Die Sammlung besteht aus Aufnahmen von 630 verschiedenen Sprechern, wovon 438 männlich und 192 weiblich sind. Von jedem Sprecher wurden zehn akustisch reichhaltige Sätze aufgenommen. Die Sprecher stammen aus acht verschiedenen Sprachregionen der USA [20].

3.3.2 MNIST

Der MNIST Datensatz besteht aus insgesamt 70'000 Beispielen von handgeschriebenen Ziffern. Die Ziffern liegen als Graustufenbild vor, von jeweils 28 x 28 Pixel. Alle Ziffern wurden zentriert und auf die gleiche Grösse transformiert. Der Datensatz wurde schon in etlichen Arbeiten verwendet [21] und dient häufig als Grundlage für Tutorials [22]. Abbildung 5 zeigt einige Beispiele aus dem Datensatz.

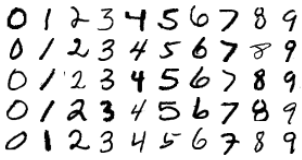

Abbildung 5: Einige Beispiele für Ziffern, welche im MNIST Datensatz enthalten sind.

4 Implementierung

In diesem Kapitel wird auf relevante Punkte in der Implementierung eingegangen. Zum einen wird die grundlegende Architektur des Netzwerkes erklärt, zum anderen die genaue Implementierung der Loss-Funktion.

4.1 Netzwerk

Die Architektur des Netzwerkes wurde von Lukic et al. [1] übernommen und ist in Abbildung 6 gezeigt. Sie unterscheidet sich nur in zwei Punkten von der ursprünglichen Architektur:

- Für die Loss-Funktion war es nötig, den Tensor zu einem Vektor umzuformen. Deshalb wurde vor dem ersten Dense Layer L6 ein Reshape Layer eingefügt. Da das Netzwerk ab diesem Punkt die Daten nicht mehr als Bilder betrachtet, wie es bei den Convolutional Layer der Fall ist, hat diese Umformung keinen Einfluss auf die Interpretation der Daten.
- Als Loss-Funktion kam anstelle der *Cross Entropy* die Loss-Funktion von Hofer et al. [3] zum Einsatz.

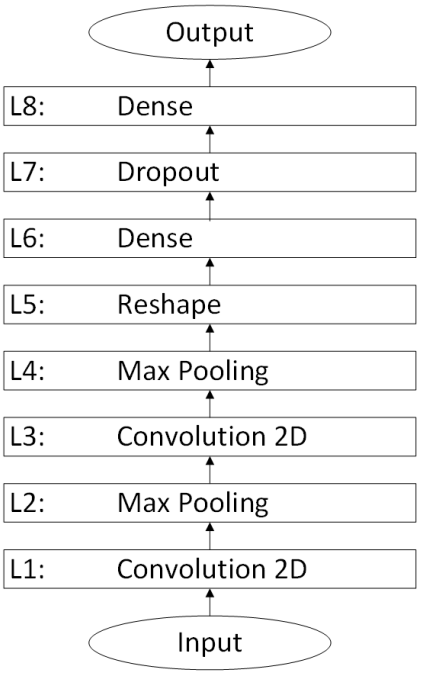

Abbildung 6: Grundlegender Aufbau des Netzwerkes, basierend auf dem Netzwerk von Lukic et al. [1]. Es unterscheidet sich hauptsächlich bei der Loss-Funktion (hier nicht gezeigt) und dem Reshape Layer L5, welcher nötig ist, um die Daten in eine für die Loss-Funktion passende Form umzuwandeln.

Die Konfiguration der einzelnen Layer ist in Tabelle 1 gezeigt. Falls nicht anders erwähnt, wird die Standardkonfiguration von TensorFlow verwendet. Es werden ausschliesslich Layer des TensorFlow Modules `tf.layers` verwendet. n_s bezeichnet die Anzahl Sprecher im Trainingsdatensatz.

Layer	Konfiguration
L1 Convolution 2D	Anzahl Filter = 32 Kernelgrösse = 4x4 Padding = Same Aktivierung = Relu
L2 Max Pooling	Pool Grösse = 4x4 Strides = 2
L3 Convolution 2D	Anzahl Filter = 64 Kernelgrösse = 4x4 Padding = Same Aktivierung = Relu
L4 Max Pooling	Pool Grösse = 4x4 Strides = 2
L5 Reshape	-
L6 Dense	Units = $10 * n_s$ Aktivierung = Relu
L7 Dropout	Rate = 0.5
L8 Dense	Units = $5 * n_s$ Aktivierung = Relu

Tabelle 1: Verwendete Grundkonfiguration der einzelnen Layer. Falls nicht anders angegeben, wird die Standardkonfiguration von TensorFlow verwendet.

Die Dense Layer sind intern aus mehreren Layern aufgebaut. Abbildung 7 zeigt einen Dense Layer mit Relu Aktivierung und einen Dense Layer ohne Aktivierung. Dieser interne Aufbau ist relevant für die Embeddings, welche jeweils aus einem dieser Layer (Relu oder BiasAdd) gewonnen wird.

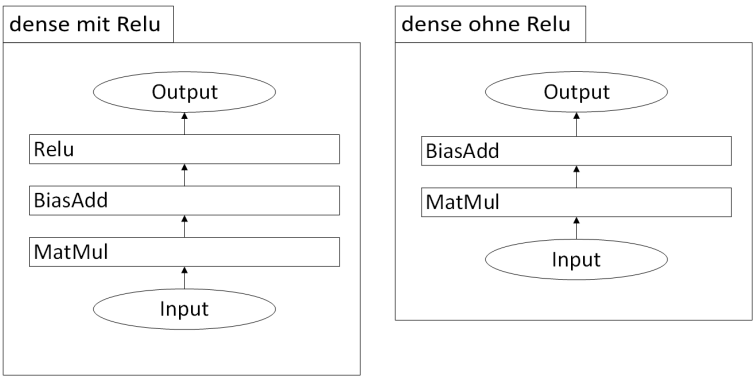

Abbildung 7: Interner Aufbau eines Dense Layers. Der Dense Layer mit Relu Aktivierung (links) hat einen internen Layer mehr, als der Dense Layer ohne Aktivierung (rechts).

4.2 Loss-Funktion

Die Implementierung der Loss-Funktion gemäss Hoffer et al. [3] musste für die Anwendung in unserem Netzwerk auf die Verarbeitung eines ganzen Batches ausgelegt werden. Anders als bei herkömmlichen Loss-Funktionen, kann hier nicht der ganze Batch auf einmal mit einer Matrixoperation verarbeitet werden, sondern nur Element für Element.

Für die elementweise Verarbeitung gibt es die TensorFlow Operation `tf.map_fn(fn, tensor)`, welche für jedes Element in der Dimension 0 (Batchdimension) die Funktion `fn` anwendet. Damit lässt sich der Loss gemäss Hoffer et al. für jedes Sample im Batch berechnen. Für die Summierung im Divisor (vgl. Kapitel 3.1.2) wird ein weiteres `tf.map_fn(fn, tensor)` verwendet, um die Distanzen zwischen dem aktuellen Sample und allen Klassenrepräsentanten zu berechnen.

Während den ersten Versuchen produzierte die Loss-Funktion häufig Inf (*infinity*) und NaN (*not a number*) Resultate. NaN Werte entstehen bei TensorFlow, wenn man den Logarithmus von 0 (`tf.log(0)`) berechnet. $-Inf$ Werte sind das Resultat des Logarithmus von sehr kleinen Float32 Zahlen. Da bei der Loss-Funktion von Hoffer et al. nach dem Logarithmus der ganze Ausdruck negiert wird (vgl. Gleichung 3.2), wird aus dem $-Inf$ ein Inf. Um solche Grenzfälle zu verhindern, haben wir die Loss-Funktion umgeformt und ein Epsilon von $1e-6$ eingefügt. Die Gleichung 4.1 zeigt die umgeformte Loss-Funktion mit dem zusätzlichen Epsilon ϵ.

$$L(x_l, z_1, ..., z_c)_L = \|F(x_l) - F(z_k)\|_2 + \log(\sum_{i=1}^{c} e^{-\|F(x_l)-F(z_i)\|_2} + \epsilon) \tag{4.1}$$

Die Division als Ursache für NaN Werte fällt weg und der Logarithmus wird nur noch über einen Teil der Loss-Funktion berechnet. Das zusätzliche Epsilon verhindert dort allfällige 0 Werte. Mit diesen Anpassungen wurde die Loss-Funktion numerisch stabil. Das Listing 4.1 zeigt die Implementierung der Loss-Funktion mit oben genannten Anpassungen.

```
1 loss = tf.map_fn(lambda x: tf.add(tf.norm(tf.subtract(x[0],
      ph_class_input_data[x[1]]), ord='euclidean'), tf.log(tf.add( 1e-6, tf.
      reduce_sum(tf.map_fn(lambda z: tf.exp(tf.negative(tf.norm(tf.subtract(x
      [0], z), ord='euclidean'))), ph_class_input_data), 0)))), (l8,
      ph_map_labels), dtype=tf.float32)
2
3 loss = tf.reduce_mean(loss)
```

Listing 4.1: Implementierung der Loss-Funktion von Hoffer et al. [3] gemäss Gleichung 4.1 in TensorFlow.

Die Variable `l8` beinhaltet den aktuellen Batch mit Samples x_l, von denen der Loss berechnet wird. `ph_class_input_data` beinhaltet die Klassenrepräsentanten $z_1, ..., z_c$ und `ph_map_lables` beinhaltet für jedes Sample x_l das richtige Label, damit der Loss berechnet werden kann. Am Schluss wird der Durchschnitt der einzelnen Losswerte berechnet (`tf.reduce_mean(loss)`).

4.3 Analyse

Um unsere Implementierung zu überprüfen haben wir einige Aspekte dessen genauer überprüft. Diese Analysen werden in diesem Abschnitt dargelegt.

4.3.1 Validierung Loss-Funktion

Um sicherstellen zu können, dass die Loss-Funktion richtig implementiert wurde, wurde eine Referenzimplementierung mit `numpy` erstellt. Anhand von Beispieldaten wurde dann der Loss der `numpy`- und TensorFlow Implementierung verglichen. Dafür generierten wir vier künstliche Samples und zwei Klassenrepräsentanten. Der berechnete Loss aus den beiden Implementierungen stimmt überein. Das Listing C.1 im Anhang C zeigt Referenzimplementierung.

4.3.2 Charakteristik Loss-Funktion

Um die Funktionsweise der Loss-Funktion genauer zu verstehen, wurde anhand kleiner Beispiele Versuche gemacht. Da wir von der von Hoffer et al. [3] vorgeschlagenen Loss-Funktion nur den *supervised* - Teil verwenden, wollten wir sicherstellen, dass dessen Verwendung keine unerwünschten Auswirkungen hatte.

Die Loss-Funktion minimiert einerseits die Intracluster-Distanz (Distanz zwischen Samples derselben Klasse) und maximiert andererseits die Intercluster-Distanz (Distanz zwischen einzelnen Clustern). Die beiden Distanzen sind in Abbildung 8 veranschaulicht.

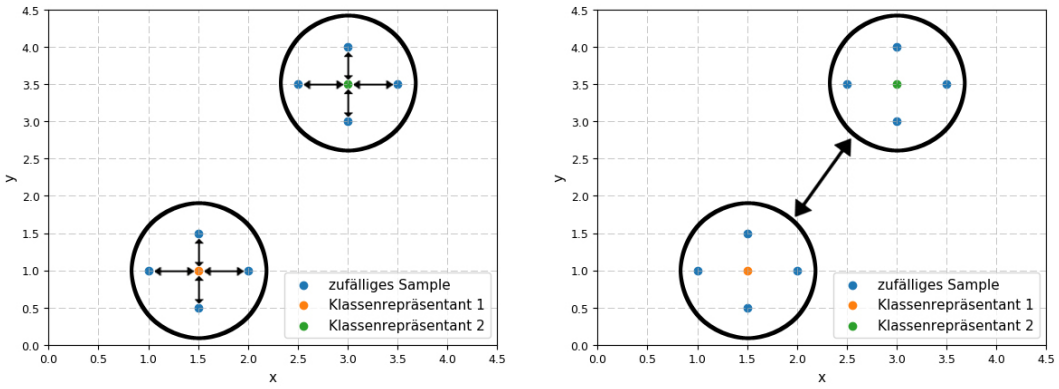

Abbildung 8: Intracluster-Distanz (links) beschreibt die Distanz der Samples innerhalb eines Clusters zum korrespondierenden Klassenrepräsentanten während die Intercluster-Distanz (rechts) die Distanz zwischen den einzelnen Clustern beschreibt.

Die Loss-Funktion wurde bezüglich verschiedener Szenarien analysiert. Um Referenzwerte für den Losswert zu erhalten, wurde eine Grundkonfiguration gemäss Abbildung 9 (links) definiert. Diese Konfiguration stellt ein Batch dar, welcher zufällige Samples (blaue Punkte) sowie zwei Klassenrepräsentanten (grüner und oranger Punkt) enthält. Die vier blauen Punkte um den grünen Punkt entsprechen Samples des grünen Sprechers, während die vier blauen Punkte rund um den orangen Punkt Samples des orangen Sprechers entsprechen. Bei diesem vorteilhaften Szenario wurde ein Loss von 0.0884 errechnet. Werden die beiden Klassenrepräsentanten getauscht und somit ein ungünstiges Szenario geschaffen, steigt der Loss auf einen Wert von 2.5254, da die Samples rund um den Repräsentanten nicht zu demjenigen gehören.

Erneut ausgehend vom Losswert unseres vorteilhaften Szenarios sinkt der Loss, sobald sich die Intracluster-Distanz verringert und steigt sobald sich die Intracluster-Distanz erhöht. Zusätzlich sinkt der Loss für den Fall, dass sich die Intercluster-Distanz erhöht und vice versa.

Falls die Loss-Funktion die Intracluster-Distanz zu stark gewichtet, bestünde die Gefahr, dass alle Samples aller Cluster auf den gleichen Punkt zu liegen kommen. Wird die Intracluster-Distanz um Faktor 10 verringert, hat dies einen weniger grossen Einfluss auf den Loss, als wenn die Intercluster-Distanz um Faktor 10 erhöht wird. Demzufolge gewichtet die Loss-Funktion die „Ferne von verschiedenen Clustern“ grösser, als die „Nähe der Samples innerhalb eines Clusters“. Diese Vermutung wurde anhand zweier Szenarien gestärkt.

- Um zu überprüfen, ob sich die Loss-Funktion in kleineren Grössenordnungen identisch verhält, wurden die Werte der Samples und Repräsentanten um Faktor 10 sowie Faktor 100 verkleinert.
- Es wurde eine weitere Grundkonfiguration gemäss Abbildung 9 (rechts) geschaffen, welche einheitliche Abstände zwischen Samples und Klassenrepräsentaten sowie zwischen den beiden Klassenrepräsentanten aufweist. Damit sollte gezeigt werden, ob das Verhältnis zwischen der Intercluster- und Intracluster-Distanz einen Einfluss hat. Wie beim vorhergehenden Szenario wurden auch in dieser Konfiguration die Werte um Faktor 10 sowie Faktor 100 verkleinert.

In allen Experimenten reagierte der Losswert sensibler auf Veränderungen der Intercluster-Distanz. Somit ist gezeigt, dass die Loss-Funktion bestrebt ist, die Cluster möglichst voneinander zu trennen.

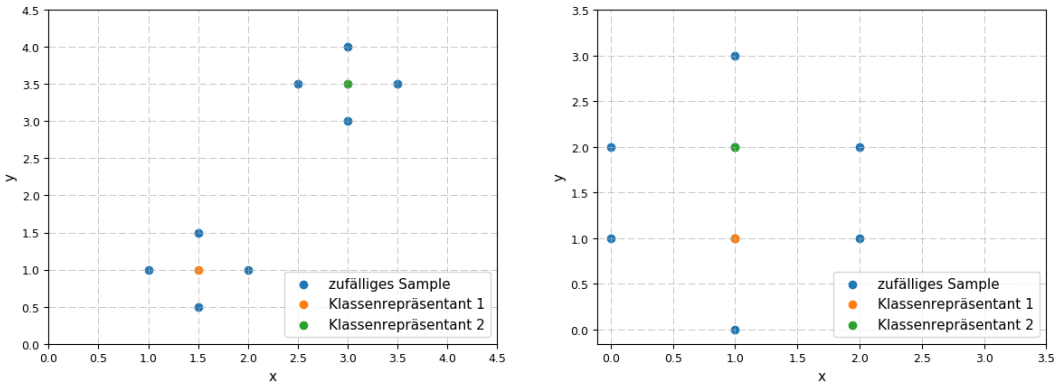

Abbildung 9: Grundkonfiguration für die Analyse der Loss-Funktion in einem vorteilhaften Szenario (links). Szenario zur Ermittlung der Gewichtung von Intracluster- und Intercluster-Distanz mit einheitlichen Abständen zwischen Samples und Klassenrepräsentanten sowie zwischen beiden Klassenrepräsentanten (rechts).

Folgende weitere Charakteristiken haben sich im Rahmen der Analyse gezeigt:

- Bewegt sich einer der Cluster in den negativen Wertebereich, hat dies keinen nachteiligen Einfluss auf den Losswert. Dies ist zu erklären mit der euklidschen Norm in Hoffers et al. [3] Loss-Funktion, welche negative Werte normiert.
- Bewegen sich Werte in der Nähe des Nullpunktes, hat auch dies keinen ungünstigen Einfluss auf den Losswert.
- Der Losswert reagiert stabil auf Szenarien, in denen eines oder mehrere Samples den gleichen Wert wie der entsprechende Klassenrepräsentant aufweist.

4.3.3 Lernfähigkeit

Nach der Implementierung des Netzwerkes war nicht von Anfang an garantiert, dass dieser Aufbau auch lernt. Die Kombination von Netzwerkarchitektur von Lukic et al. [1] und Loss-Funktion von Hoffer et al. [3] war neu. Um dies zu überprüfen, haben wir einerseits einen Versuch mit dem MNIST Datensatz durchgeführt, andererseits die Embeddings auf zwei Dimensionen reduziert, um sie direkt darstellen zu können.

4.3.3.1 2D Embeddings

Die Embeddings, welche das Netzwerk erzeugt, sind bei der Grundkonfiguration jeweils von der Grösse $5 * n_s$, wobei n_s der Anzahl Sprecher entspricht. Solche Embeddings kann man sich nur schwer vorstellen und können nur mithilfe von Verfahren zur Dimensionsreduktion zweidimensional dargestellt werden.

Um den Lernprozess direkt beobachten zu können, haben wir ein Netzwerk mit einer Liste von 10 Sprechern trainiert, welches Embeddings der Grösse 2 erzeugt. Solche Embeddings lassen sich direkt in einem Scatter-Plot darstellen. Die Abbildung 10 zeigt einen solchen Plot von einem Batch während dem Training.

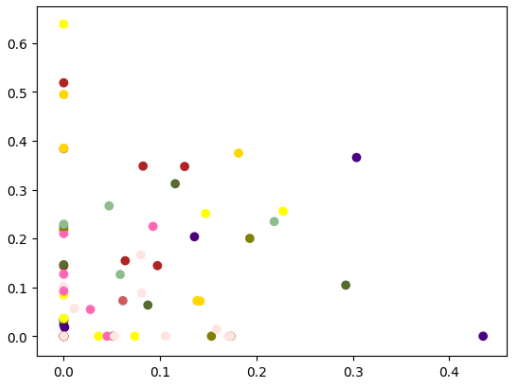

Abbildung 10: Scatter Plot von einem Batch während dem Training. Punkte der gleichen Farben sind Samples vom gleichem Sprecher. Die Punkte sind noch ohne erkennbare Ordnung verteilt, da der Plot ganz zu Beginn des Trainings erzeugt wurde. Der Einfluss der Relu zeigt sich deutlich bei Punkten, welche auf einer oder beiden Dimension auf der 0-Achse liegen.

Es fällt auf, das bei mehreren Punkten eine oder sogar beide Dimensionen den Wert 0 haben. Dies ist zu begründen mit der ReLu Aktivierung im Dense Layer L8, welche das Maximum zwischen 0 und dem aktuellen Wert auswählt. So werden alle negativen Werte auf 0 gesetzt. Für Embeddings ist das nicht gewünscht, da so Informationen verloren gehen.

Um zu verhindern, dass negative Werte auf 0 gesetzt werden, wurde die Relu Aktivierung im letzten Layer L8 entfernt. Die Abbildung 11 zeigt zwei Scatter Plots mit wiederholtem Training ohne Relu Aktivierung im Layer L8. Deutlich sind die Lernfortschritte in der rechten Grafik nach 5000 Steps zu sehen.

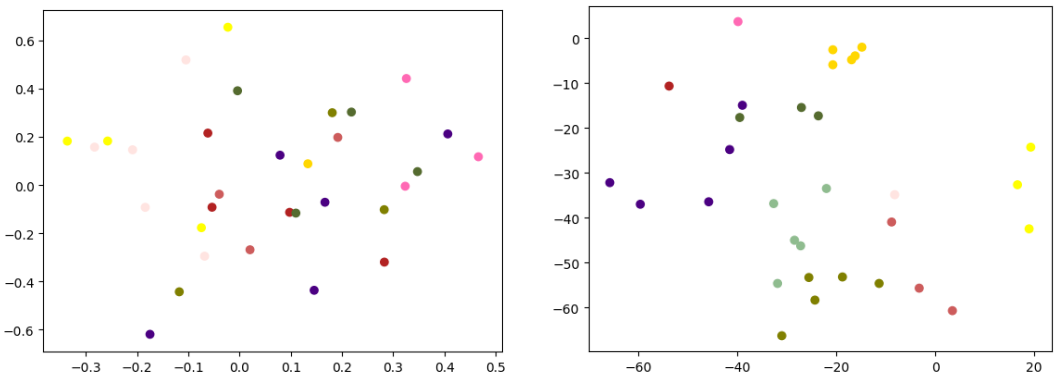

Abbildung 11: Scatter Plot von einem Batch am Anfang des Trainings (links) und nach 5000 Steps (rechts) ohne Relu Aktivierung im letzten Layer L8. Punkte der gleichen Farbe entsprechen Samples von gleichen Sprechern. Die Punkte sind im rechten Plot geordneter und bilden erste Cluster

Die Erkenntnis über den Einfluss der Relu liessen wir in die späteren Experimente (vgl. Kapitel 6) einfliessen.

4.3.3.2 MNIST

Hoffer et al. [3] entwickelte seine Loss-Funktion unter anderem für den MNIST Datensatz. Zur Überprüfung, ob unsere Implementierung korrekt funktioniert, führten wir einen Versuch mit MNIST Daten durch. MNIST Daten sind Bilder, wie auch die Spektrogramme, welche wir aus den TIMIT Daten erzeugen. So lässt sich unser Netzwerk ohne grosse Anpassungen auch mit MNIST Daten speisen.

Die Erwartung war, dass wir mit den einfacheren MNIST Daten schneller erkennen, ob das Netzwerk lernt oder nicht. Da MNIST Bilder kleiner sind (vgl. Kapitel 3.3.2) als unsere Spektrogramme (vgl. Kapitel 5.5), dauert ein Trainingsstep auch nur ein Bruchteil so lange wie bei TIMIT Daten. Weiter sind MNIST Daten im Vergleich zu den Spektrogrammen einfacher, weil das Muster (bei MNIST Ziffern) viel eindeutiger und weniger detailliert ist. Abbildung 12 stellt ein MNIST Sample einem TIMIT Spektrogramm Sample gegenüber.

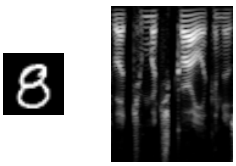

Abbildung 12: Das MNIST Sample (links) besitzt ein eindeutigeres Muster als das TIMIT Spektrogramm Sample (rechts).

Wir trainierten zwei Netzwerke, eines mit dem MNIST Daten, ein zweites mit den TIMIT-Daten, bestehend aus 10 Sprechern. Die Fortschritte beobachteten wir im TensorBoard. Abbildung 13 zeigt einen PCA Plot mit zwei Komponenten nach 1800 Trainingsteps.

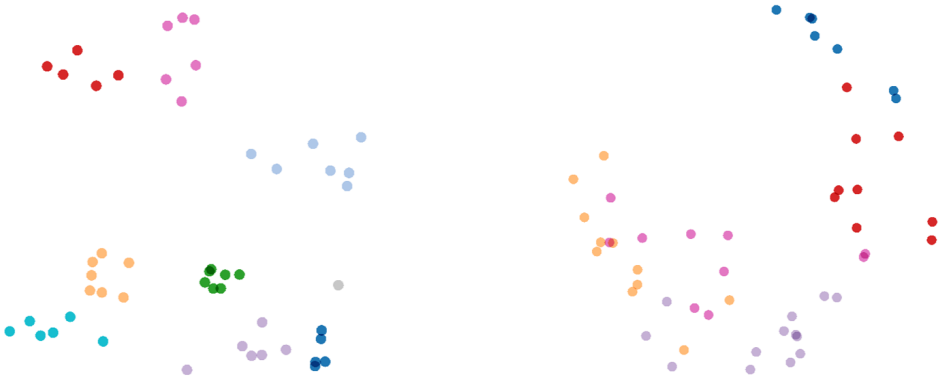

Abbildung 13: Die Abbildungen zeigen jeweils einen PCA Plot mit zwei Komponenten nach 1800 Trainingsstep. Bei den MNIST Daten (links) sind bereits deutlich Cluster zu erkennen, bei dem TIMIT Daten (rechts) überlappen sich einige Daten.

Diese Analyse bestätigt uns, dass das Netzwerk und die Loss-Funktion im Grunde funktionieren. Die MNIST Resultate sehen bereits vielversprechend aus und auch bei den TIMIT Daten sind erste Cluster zu erkennen.

5 Experimenteller Aufbau

5.1 Bedingungen

Alle Experimente werden auf dem GPU-Cluster der ZHAW durchgeführt. Dieser Cluster besteht aus 2 x 8 NVIDIA Titan X Grafikkarten, 24 CPU Kernen des Types Intel Xeon E5-2650 2.20GHz sowie 2 x 512 GB Arbeitsspeicher. Davon wird für ein einzelnes Experiment jeweils ein GPU, zwei CPUs sowie 32 GB Arbeitsspeicher alloziiert.

Um die Experimente in einer gekapselten Umgebung durchführen zu können, wurde jedes einzelne Experimente in einem Docker-Container gestartet. Sämtliche Experimente wurden mit dem gleichen Docker-Image durchgeführt, welches auf dem `tensorflow:1.0.1-gpu`-Image basiert.

Um die WAV-Dateien des TIMIT-Datensatzes für die weitere Verarbeitung einzulesen, wurde die Python Bibliothek `SciPy 0.19.0` sowie `numpy 1.12.1` verwendet. Zur Generierung der Spektrogramme wurde `librosa` in Version `0.5.0` verwendet.

5.2 Konfiguration

Für jedes Experiment wird eine Konfigurationsdatei erstellt, welche alle Parameter enthält, die das Experiment definieren. Dies sind beispielsweise Anzahl Sprecher, Grösse des Batches, Anzahl Units der einzelnen Layer oder Informationen zum Optimizer. Während der Experimente wurden viele unterschiedliche Konfigurationen getestet. Damit diese Experimente nachvollziehbar bleiben, haben wir alle wichtigen Parameter des Netzwerkes und der Validierung in eine Konfigurationsdatei ausgelagert. Das Listing im Anhang B.1 zeigt die Grundkonfiguration und das Listing im Anhang B.2 zeigt eine Beispielkonfiguration eines Experimentes.

Die Grundkonfiguration wird immer geladen und anschliessend mit der eigentlichen Konfigurationsdatei des Experimentes überschrieben. Dadurch müssen die Konfigurationsdatein der Experimente nur noch vom Standard abweichende Konfigurationen enthalten.

5.3 Sprecherlisten

Eines der zentralsten Elemente bei unseren Speaker Clustering Experimenten sind die Sprecherlisten. Das Wichtigste dabei ist, dass kein Sprecher der Testliste auch in der Trainingsliste enthalten ist. Trainings- und Testlisten müssen zwingend disjunkt sein, ansonsten sind die Resultate voreingenommen und nicht valide.

Der TIMIT-Datensatz besteht aus zehn gesprochenen Sätzen von 630 Sprecherinnen und Sprechern (vgl. Kapitel 3.3.1) und ist deshalb für viele Experimente zu gross. In einem ersten Schritt muss deshalb ein Subset von Sprechern ausgewählt und in einer Liste abgelegt werden gemäss Abbildung 14.

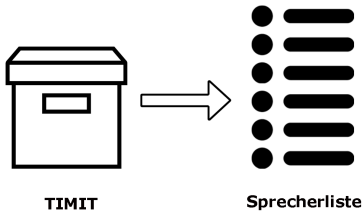

Abbildung 14: Aus dem TIMIT Datensatz wird eine Liste von Sprechern generiert.

Die in unseren Experimenten verwendete Trainingsliste besteht aus total 100 Sprecherinnen und Sprechern, davon sind 50 weiblich und 50 männlich (vgl. Anhang D.1). Diese Sprecherliste wurde bereits von Lukic et al. [1][2] verwendet und von uns übernommen. Sie ist wie folgt zusammengesetzt: 38 Sprecherinnen und Sprecher aus Sprachregion 1 (New England), 49 Sprecherinnen und Sprecher aus Sprachregion 2 (Northern) sowie 13 Sprecherinnen aus Sprachergion 3 (North Midland).

Eine Testliste mit 40 Sprechern (vgl. Anhang D.2) wurde 2009 von Stadelmann et al. [23] anhand des lexikographisch sortierten Dateipfades erstellt. Dadurch wurden, wie in Abbildung 15 (rechts) ersichtlich, elf Sprecherinnen und Sprecher aus der Sprachregion 1, 26 Sprecherinnen und Sprecher aus der Sprachregion 2 sowie drei Sprecherinnen der Sprachregion 3 selektiert. Diese Testliste wurde von Lukic et al. [1][2] sowie mehreren Projekt- und Bachelorarbeiten an der ZHAW (unter anderem auch von uns) weiterverwendet.

Abbildung 15: Übersicht TIMIT-Datensatz (links), aufgeteilt nach Geschlecht sowie nach Sprachregionen im Vergleich zur verwendeten Trainingsliste mit 100 Sprechern (Mitte) sowie Testliste mit 40 Sprechern (rechts).

Für zukünftige Speaker Clustering Experimente ist es unseres Erachtens sinnvoll, andere Sprecherlisten sowohl für das Training als auch für das Testing zu verwenden. Die aktuelle Trainingsliste limitiert die Grösse der Testlisten, da man sinnvollerweise nur Sprecher von Sprachregionen testet, die auch in der Trainingsliste enthalten sind. Bei neuen Listen ist es wichtig, dass Sprecher von unterschiedlichem Geschlecht sowie von allen unterschiedlichen Sprachregionen vertreten sind, um eine grösst mögliche Diversität zu gewährleisten.

Wir schlagen folgendes Modell zur Generierung von Trainings- und Testinglisten vor: Alle 630 Sprecher des TIMIT-Datensatzes werden zufällig gemischt, in dieser Reihenfolge abgespeichert und bildet die Basisliste (vgl. Anhang D.3). Da die manuelle Konstruktion einer ausgeglichenen und unvorbelasteten Liste sehr schwierig ist, wurde eine zufällige Reihenfolge gewählt. Vom Be-

ginn dieser Basisliste wird aus den ersten x Sprecher eine Liste für das Training des Modells erstellt. Vom Ende derselben Basisliste werden mehrere, aufeinander aufbauende Listen mit y_i Sprecher für das Testen des Modells erstellt. Dabei gilt für jedes Paar von x und y_i die Ungleichung 5.1.

$$x + y_i \leqslant 630 \tag{5.1}$$

Die Verteilung der Sprecher über die Sprachregionen ist sowohl in der Trainingsliste mit 100 Sprechern als auch in den Testlisten mit 40 bzw. 80 Sprechern gleichmässiger verteilt als in den ursprünglichen Listen. Abbildung 16 zeigt die Verteilung der neuen Trainings- und Testlisten, welche eine deutlich bessere Verteilung zeigen als die Verteilung der bestehenden Listen in Abbildung 15 (Mitte und rechts).

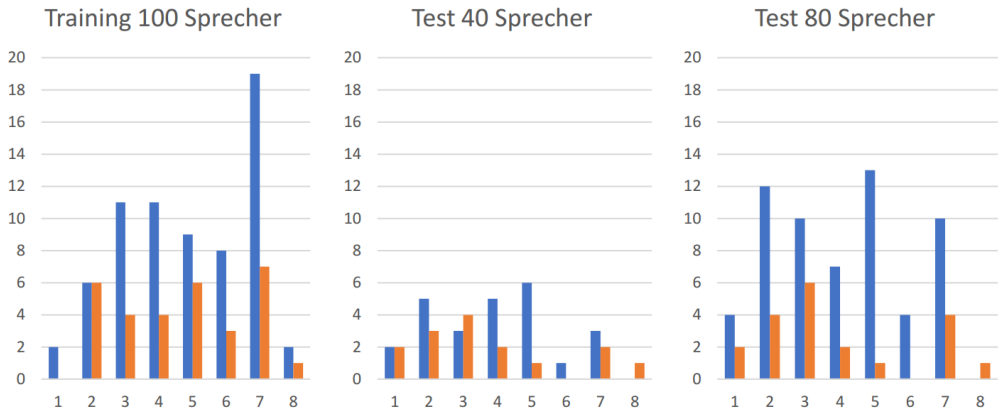

Abbildung 16: Verteilung der vorgeschlagenen, neuen Sprecherlisten. Die Trainingsliste mit 100 Sprechern (links), die Testliste mit 40 Sprechern (Mitte) und die Testliste mit 80 Sprechern (rechts) zeigen eine deutlich bessere Verteilung.

Der Dialekt eines Sprechers widerspiegelt sich in den generierten Spektrogrammen. Lernt ein Neuronales Netzwerk einer Liste mit 100 Sprechern ähnlichen Dialektes, so muss das Netzwerk die Sprecher anhand anderer (vermutlich weniger prägnanten) Merkmalen unterscheiden. Es kann vermutet werden, dass diese weniger prägnanten Merkmale zu schlechteren Clusteringergebnissen führen.

5.4 Trainingsdaten erzeugen

In einem zweiten Schritt wird gemäss Abbildung 17 für jeden gesprochenen Satz jedes Sprechers aus der Sprecherliste ein Spektrogramm erstellt. Die Spektrogramme werden gemäss Gleichung 2.1 in die Mel-Skala überführt und mit einer *Dynamic Range Compression* gemäss Gleichung 2.2 werden wenig prägnante Werte im hochfrequenten Spektrum verstärkt. Die Mel-Spektrogramme werden in einer Pickle-Datei abgelegt und können so für verschiedenen Experimente wiederverwendet werden. Dieses Vorgehen ist für Trainingsliste sowie Testingliste identisch.

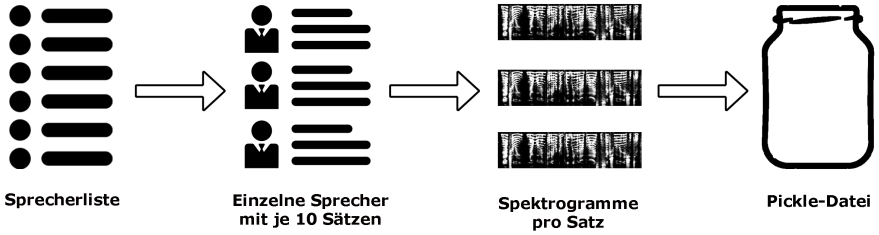

Abbildung 17: Aus der Sprecherliste werden zuerst für jeden einzelnen Sprecher die entsprechenden 10 Sätze geladen. Daraus werden Spektrogramme erzeugt, welche in einer Pickle-Datei abgespeichert werden.

5.5 Training

Ist die Pickle-Datei abgespeichert, kann daraus in einem dritten Schritt gemäss Abbildung 18 ein Batch generiert werden. Der Batch dient als Input für das Neuronale Netzwerk und besteht aus einer Menge von Ausschnitten der Spektrogramme. Diese Ausschnitte der Spektrogramme haben die Grösse 100x128 (Zeit x Frequenz) und werden als Samples bezeichnet. Zusätzlich zum Batch wird ein Ausschnitt pro Sprecher (sogenannte *Klassenrepräsentanten*) benötigt.

Das Neuronale Netzwerk wird über mehrere Iterationen trainiert und periodisch abgespeichert. Diese abgespeicherten Netzwerke werden *Checkpoints* genannt. An jedem dieser Checkpoints lässt sich später ein Testing bzw. eine Auswertung durchführen. Über das TensorBoard kann der Verlauf des Trainings bzw. die Veränderung des Loss beobachtet werden.

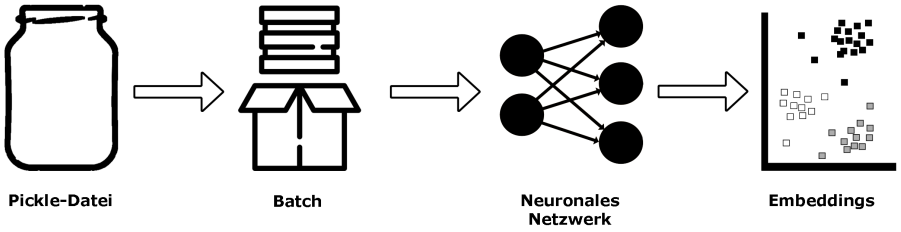

Abbildung 18: Aus der Pickle-Datei wird ein Batch generiert, der zufällige Ausschnitt von Spektrogrammen enthält. Der Batch dient als Input für das Neuronale Netzwerk. Das Netzwerk erzeugt aus dem Input Embeddings, welche der Loss-Funktion zugeführt werden.

5.6 Testing

Um gelernte Modelle zu beurteilen, werden diese mit verschiedenen Listen getestet (vgl. Anhang D.2). Die Spektrogramme aller Sätze von allen Sprecher werden in nicht überlappende Samples der Grösse 100x128 (Zeit x Frequenz) aufgeteilt. Von diesen Samples werden durch das vortrainierte Netzwerk Embeddings erzeugt. Die Embeddings der acht Sätze pro Sprecher bzw. zwei Sätze pro Sprecher werden zu einer Sprecherrepräsentation zusammengefasst. Eine Sprecherrepräsentation entspricht dem Durchschnitt der Embeddings von mehreren Samples. Das Clustering wird somit nicht auf Sampleebene durchgeführt. Die beiden Repräsentationen werden mit einem agglomerativen, hierarchischen Clustering (vgl. Kapitel 2.6.1) gruppiert. Dieser experimentelle Aufbau wurde bereits in verschiedenen Arbeiten [1][2][23] im Bereich Speaker Clustering verwendet und erlaubt ein direktes Vergleichen der Resultate.

Für das sukzessive Zusammenschliessen beim agglomerativen, hierarchischen Clustering wird die Complete-linkage Methode verwendet, welche bereits in anderen Speaker Clustering Arbeiten [1][2][24] erfolgreich eingesetzt wurde. Lukic et al. beschreiben in [1], dass sie mit der Cosinus-Metrik die besten Ergebnisse erzielten. In dieser Arbeit verwenden wir deshalb ebenfalls die Cosinus-Metrik.

5.7 Beurteilung Resultate

Um unseren Ansatz beurteilen zu können, haben wir mit einem Testdatensatz und der *Misclassification Rate* die trainierten Modelle evaluiert. Durch frühere Arbeiten im Bereich Speaker Clustering ist die State of the Art *Misclassification Rate* auf dem verwendeten Datensatz bzw. dessen Teilmengen bekannt und ein direkter Vergleich wird möglich. Für zukünftige Arbeiten habe wir ein neues Mass, die *Durchschnittliche Misclassification Rate* eingeführt.

5.7.1 Misclassification Rate

Um die Qualität des Clusterings zu bestimmen, verwenden wir die *Misclassification Rate* MR. Diese gibt an, wie gross der Anteil der Samples ist, welche einem falschen Cluster zugeordnet sind. Ein tiefer MR-Wert zeugt von einem guten Clustering, bei einer $MR = 0$ wurden alle Audiosegmente richtig gruppiert. Umgekehrt wurden alle Audiosegmente falsch gruppiert, wenn der MR-Wert gleich 1 ist. Ein richtig gewählter *Threshold* ist entscheidend für eine gute MR. Für diese Arbeit berechnen wir die MR an allen Merging-Punkten und wählen jeweils den Threshold mit der besten MR aus. Dieses Vorgehen wurde bereits von Lukic et al. [2] verwendet.

Die MR kann gemäss Gleichung 5.2 berechnet werden. Diese ist definiert durch Kotti et al. [25] bzw. Liu et al. [26].

$$MR = \frac{1}{N} \sum_{j=1}^{N_s} e_j \tag{5.2}$$

Dabei ist N die Anzahl zu gruppierender Audiosegmente. N_s ist die Anzahl Sprecher und e_j die Anzahl falsch zugewiesener Audiosegmente von Sprechern j. In Kotti et al. [25] wird e_j wie folgt beschrieben:

> „Let e_j denote the total number of segments uttered by speaker j that are not mapped to the corresponding cluster.“ [25]

Aus obiger Definition geht nicht eindeutig hervor, was unter einem falsch zugewiesenen Audiosegment verstanden wird beziehungsweise welche Eigenschaft einen Cluster einem Sprecher zugehörig macht.

Wir definieren die MR für diese Arbeit wie folgt:

Definition Für jeden Sprecher wird der passende Cluster gesucht. Ein Cluster wird als passend bezeichnet, wenn die folgenden zwei Bedingungen erfüllt werden:

- Die meisten Samples eines Sprechers müssen in diesem Cluster beinhaltet sein.
- Innerhalb des Clusters darf es keinen anderen Sprecher geben, der mehr Samples hat.

Ist eine dieser Bedingungen nicht erfüllt, wird mit dem Cluster, welcher die nächst kleinere Anzahl an vorkommenden Samples besitzt, fortgefahren. Es kann vorkommen, dass ein Sprecher keinem Cluster zugewiesen wird. Ist ein Sample nicht im Cluster seines Sprechers, wird es als falsch klassifiziert gewertet.

5.7.2 Abgrenzung

Die Resultate aus Lukic et al. 2016 [1] sowie 2017 [2] basieren auf einer anderen MR-Berechnung. In dieser Arbeit wurden sämtliche Experimente sowohl mit unserer Definition (im Folgenden als MR bezeichnet) als auch der Interpretation von Lukic et al. beurteilt. Letztere bezeichnen wir im Folgenden als LMR (Legacy Misclassification Rate).

Bei der Definition der LMR kommen zwei zusätzliche Kriterien dazu:

- Samples, welche alleine in einem Cluster sind, zählen als falsch klassifiziert.
- Alle Samples eines Cluster, welche nicht nur von einem Sprecher stammen, zählen als falsch klassifiziert.

Die Unterschiede in der Definition der MR und LMR lassen sich an Abbildung 19, mit dem in grün eingezeichnet Threshold, erklären.

Beispiel 1 Die ersten drei Segmentpaare wurde korrekt gruppiert. Als Beispiel gehört der erste Cluster offensichtlich zur Sprecherin `FDRD1`, da nur zwei Audiosegmente darin enthalten sind und diese beide von ihr stammen. Diese Klassifizierung ergibt sich bei der MR, als auch bei der LMR

Beispiel 2 Das siebte Audiosegment von Sprecher `MABW0` befindet sich alleine in einem Cluster. Nach unserem Verständniss ist es dadurch nicht zwingend falsch zugeordnet. Man kann aber argumentieren, dass ein Clustering, in welchem sich jedes Audiosegment in einem eigenen Cluster befindet, nicht wünschenswert ist. Deshalb gilt dieses Sample gemäss der LMR als falsch klassifiziert.

Beispiel 3 Beim letzten Cluster handelt es sich gemäss unserer Defintion um den Cluster von Sprecher `MRJO0`, somit ist nur das zweite Audiosegment von `MABW0` falsch zugeordnet. Gemäss der Definition der LMR ist dieser Cluster jedoch als ganzes als falsch zu klassifizieren.

Aus total 10 Audiosegmenten wurde gemäss unserer Definition ein Segment falsch zugeordnet, daraus ergibt sich eine MR von 0.1. Verwendet man die LMR, so wie sie Lukic et al. in ihren Arbeiten [1] [2] verwendet haben, gelten sowohl das einzelne Audiosegment von Sprecher `MABW0` als auch das Cluster mit drei Audiosegmenten als falsch, daraus resultierte eine LMR von 0.4.

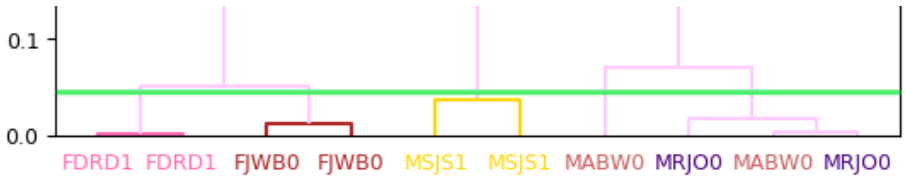

Abbildung 19: Beispiel zur Veranschaulichung der Unterschiede zwischen der MR und der LMR.

5.7.3 Vergleichbarkeit

Unsere Resultate lassen sich nur über die LMR mit den Vorgängerarbeiten vergleichen. Abbildung 20 zeigt den Unterschied zwischen der MR und der LMR über mehrere Checkpoints. Die LMR ist dabei ungefähr doppelt so gross wie die MR. Dies ist auf den Aufbau des Clusterings zurückzuführen (vgl. Kapitel 5.6), welches nur jeweils zwei Repräsentationen von Audiosegmenten pro Sprecher clustert. Im Extremfall wird der Threshold bei 0 gewählt, was einen eigenen Cluster pro Repräsentation zur Folge hätte, wobei die Hälfte als korrekt zugeordnet eingestuft würde. Dies würde in einer MR von 0.5 resultieren.

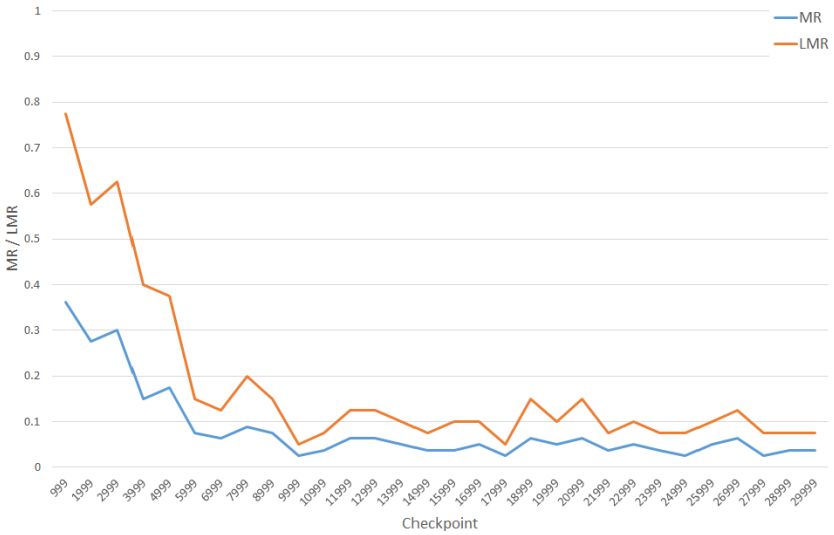

Abbildung 20: Vergleich der Misclassification Rate mit der Legacy Misclassification Rate über mehrere Checkpoints. Die LMR ist dabei ungefähr Faktor 2 grösser als die MR.

5.7.4 Mean Misclassification Rate

Lukic et al. [2] haben ihre Experimente jeweils an den Checkpoints nach 10'000, 15'000, 20'000 sowie 30'000 Iterationen ausgewertet. Wie Abbildung 21 zeigt, kann die Misclassification Rate abhängig vom Checkpoint sehr stark variieren und das Vergleichen bestimmter Punkte verhält sich weniger stabil gegenüber einem Durchschnittswert.

Wir haben uns deshalb entschlossen, ein neues Mass einzuführen. Die *Durchschnittliche Misclassification Rate* ($\varnothing MR$) ist definiert durch Gleichung 5.3. Die $\varnothing MR$ beschreibt den Mittelwert der Misclassification Rate über mehrere Checkpoints.

$$\varnothing MR = \frac{1}{N_c} \sum_{i=1}^{N_c} MR_i \tag{5.3}$$

Dabei ist N_c die Anzahl ausgewerteter Checkpoints. MR_i entspricht der Misclassification Rate von Checkpoint i.

Alle gemachten Experimente werden gemäss der $\varnothing MR$ ausgewertet. Zwecks Vergleichbarkeit mit Vorgängerarbeiten wird zusätzlich die *Durchschnittliche Legacy Misclassification Rate* $\varnothing LMR$ (vgl. Kapitel 5.7.2) ausgewertet. Die $\varnothing LMR$ wird analog Gleichung 5.3 berechnet, wobei MR_i mit LMR_i zu ersetzen ist. LMR_i entspricht der Legacy Misclassification Rate am Checkpoint i.

Für unsere Arbeit werden alle Checkpoints zwischen 10'000 und 30'000 Iterationen in einem Abstand von 1'000 Iterationen ausgewertet. Die ersten Checkpoints der ersten 10'000 Iterationen haben wir nicht in die Bewertung miteinbezogen, da das Neuronale Netzwerk bis zu diesem Zeitpunkt noch sehr stark lernt und die MR deshalb stark variiert. Abbildung 21 zeigt diesen Verlauf sowie den rot eingezeichneten Schnitt.

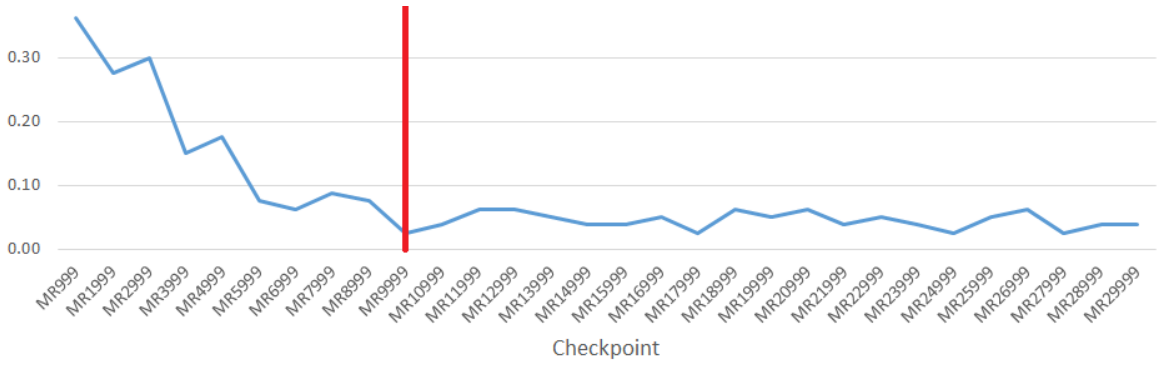

Abbildung 21: Verlauf der MR über die ausgewerteten Checkpoints mit der rot eingezeichneten Grenze bei 10'000 Iterationen.

6 Experimente

In diesem Kapitel erklären wir die durchgeführten Experimente, welche Erwartungen wir hatten und was für Erkenntnisse wir daraus gewonnen haben. Alle Experimente sind gemäss Kapitel 5 aufgebaut. Im Anhang E befindet sich eine Übersicht aller durchgeführten Experimente und deren Merkmale. Nachfolgend sind die gemeinsamen Einstellungen beschrieben.

- In der Grundkonfiguration verwenden wir eine Batchgrösse von 32 Samples. Dies wurde von Hoffer et al. [3] übernommen.
- Es wird 30'000 Iterationen lang trainiert. Diese Anzahl an Iterationen hat sich bereits bei Lukic et al. [2] bewährt.
- Jeweils nach 1'000 Iteration wird das Modell abgespeichert (Checkpoint).
- Als Optimierungsalgorithmus für das Gradientenverfahren wird Adadelta verwendet. Dessen Konfiguration ist im Anhang B ersichtlich. Lukic et al. untersuchten verschiedene Algorithmen in ihrer zweiten Arbeit [2] und kamen zum Schluss, dass Adadelta die besten Ergebnisse liefert.

Die Tabellen mit den Resultaten der folgenden Experimenten zeigen jeweils die $\varnothing MR$ und die $\varnothing LMR$ für jeden getesteten Layer der Experimente (vgl. Kapitel 5). Es wurde mit der 40-Sprecher Liste getestet (vgl. Anhang D). Die Tabellen sind aufsteigend nach der Spalte $\varnothing MR$ sortiert (symbolisiert durch ↑). Die Spalte *Layer* bezeichnet den Layer aus dem Netzwerk, aus welchem die Embeddings für das Testing generiert wurden.

6.1 Referenz

Die ersten beiden Experimenten dienen als Referenz für die aufbauenden Experimente. Sie basieren auf der Architektur, welche in Kapitel 4.1 beschrieben ist. Mit diesen beiden Experimenten wollen wir herausfinden, wie sich die Erkenntnisse im Bezug auf die Relu im letzten Layer (vgl. Kapitel 4.3.3.1) auswirken. Weiter wollen wir überprüfen, welcher Layer für die Generierung der Embeddings am geeignetsten ist. Lukic et al. [2] erzielten die besten Resultate im Dense Layer direkt nach dem CNN.

#1 Standard mit Relu Dieses Experiment basiert exakt auf der Grundarchitektur (vgl. Kapitel 4.1).

#2 Standard ohne Relu Dieses Experiment basiert exakt auf der Grundarchitektur (vgl. Kapitel 4.1), mit der einzigen Abweichung, dass im letzten Dense Layer L8 keine Aktivierung vorhanden ist.

6.1.1 Resultate

Experiment	Layer	$\varnothing MR \uparrow$	$\varnothing LMR$
#2 Standard ohne Relu	L6 Relu	**0.0440**	0.0940
#1 Standard mit Relu	L6 Relu	0.0673	0.1405
#2 Standard ohne Relu	L6 BiasAdd	0.0887	0.1833
#2 Standard ohne Relu	L8 BiasAdd	0.0940	0.1988
#1 Standard mit Relu	L8 BiasAdd	0.1214	0.2583
#1 Standard mit Relu	L6 BiasAdd	0.1232	0.2714
#1 Standard mit Relu	L8 Relu	0.1238	0.2881

Tabelle 2: Resultate der Experimente #1 und #2. Der Layer L6 liefert durchgehend bessere Resultate als der Layer L8. Ebenso scheint das Experiment #2 (ohne Relu) durchgehend besser abzuschneiden als das Experiment #1.

Das Experiment #2 Standard ohne Relu sowie der Layer L6 liefern tendenzielle bessere Ergebnisse. Das lässt sich auch bei der Auswertung mit der 60-Sprecher und 80-Sprecher Liste beobachten.

Das bessere Ergebnis des Experiment #2 war nach der Analyse während der Implementierung (vgl. Kapitel 4.3.3.1) zu erwarten. Für die weiterführenden Experimente bauen wir deshalb auf Experiment #2 auf.

Interessant ist, dass der Layer L6 insgesamt besser abschneidet, als der Layer L8. Mit der Loss-Funktion von Hoffer et al. [3], welche nach dem Layer L8 kommt, sollte das Netzwerk direkt dort Embeddings lernen. Dieses Verhalten ist identisch mit dem, welches Lukic et al. [2] in ihrer Arbeit beobachtet haben. Dies lässt vermuten, das selbst die Methode von Hoffer et al. noch nicht völlig dem angestrebten *End-to-End* Lernansatz entspricht. Ebenfalls interessant ist, dass der Relu Layer im Dense Layer L6 besser abschneidet wie der BiasAdd Layer. Wir haben keine Erklärung, weshalb das Abschneiden von Informationen zu besseren Resultaten führt.

6.2 Anzahl Units

Die Experimente in Kapitel 6.1 zeigten, dass der Layer L6 besser abschneidet als der Layer L8. Die beiden Layer unterscheiden sich durch die Anzahl Units, der Layer L6 besitzt doppelt so viele Units wie L8. Mit nachfolgenden Experimenten wollen wir untersuchen, ob eine veränderte Anzahl der Units eine Verbesserung hervorbringt. Aufgrund der besseren Performance des Layer L6 könnte man darauf schliessen, dass eine höhere Anzahl der Units bessere Resultate erbringt.

#3 1000 Units Dieses Experiment basiert auf dem Experiment # 2 Standard ohne Relu, die Anzahl Units im Layer L8 wurde jedoch von 500 auf 1000 Units erhöht.

#4 300 Units Dieses Experiment basiert auf dem Experiment # 2 Standard ohne Relu, die Anzahl Units im Layer L8 wurde jedoch von 500 auf 300 Units reduziert.

6.2.1 Resultate

Experiment	Layer	$\varnothing MR \uparrow$	$\varnothing LMR$
#3 1000 Units	L6 Relu	0.0935	0.2024
#4 300 Units	L6 Relu	0.1169	0.2548
#3 1000 Units	L8 BiasAdd	0.1351	0.3000
#3 1000 Units	L6 BiasAdd	0.1452	0.3119
#4 300 Units	L8 BiasAdd	0.1619	0.3393
#4 300 Units	L6 BiasAdd	0.1650	0.3810

Tabelle 3: Resultate der Experimente #3 und #4. Das Experiment #3 mit mehr Units schneidet besser ab, als das Experiment mit weniger Units. Jedoch bewegen sich beide Experimente unter dem Niveau vom Referenz Experiment #2.

Experiment #3 und #4 sind beide schlechter als das Experiment #2. Tendenziell sind mehr Units im Layer L8 besser als weniger. Jedoch scheinen 500 Units, wie im Referenz Experiment #2, ideal zu sein. Auch hier bestätigen die Auswertungen mit der 60-Sprecher und 80-Sprecher Liste obige Resultate.

Diese beiden Experimente bestätigen, dass die Architektur gemäss Lukic et al. [1] und insbesondere die Anzahl Units im Layer L8, geeignet für das Clustering von Spektrogrammen dieser Grösse ist.

6.3 Anzahl Layers

In ihrer zweiten Arbeit fügten Lukic et al [2] ihrem Netzwerk einen weiteren Dense Layer hinzu und erzielten damit bessere Resultate. Die nachfolgenden Experimente untersuchen, wie sich ein zusätzlicher oder weggelassener Dense Layer auf die Ergebnisse auswirken.

#5 Ohne Dense Layer L8 Dieses Experiment basiert auf dem Experiment #2 Standard ohne Relu, jedoch wurde der Layer L8 weggelassen und die Aktivierung im Dense Layer L6 weggelassen.

#6 Mit Dense Layer L9 Dieses Experiment basiert auf dem Experiment #1 Standard mit Relu mit einem zusätzlichen Dense Layer L9, ohne Aktivierung. Die Anzahl Units im Layer L8 betrugen 800, im neuen Layer L9 500.

6.3.1 Resultate

Experiment	Layer	$\varnothing MR \uparrow$	$\varnothing LMR$
#6 Mit Dense Layer L9	L6 Relu	0.0470	0.1083
#6 Mit Dense Layer L9	L8 BiasAdd	0.0738	0.1655
#6 Mit Dense Layer L9	L6 BiasAdd	0.1006	0.2095
#6 Mit Dense Layer L9	L8 Relu	0.1042	0.2369
#5 Ohne Dense Layer L8	L6 BiasAdd	0.1048	0.2095
#6 Mit Dense Layer L9	L9 BiasAdd	0.1179	0.2595

Tabelle 4: Resultate der Experimente #5 und #6. Beide Experimente sind schlechter als das Referenz Experiment #2. Der zusätzliche Dense Layer L9 liefert aber bessere Resultate als das Experiment #5 mit einem Dense Layer weniger.

Wiederum übertreffen die beiden Experimente #5 und #6 das Referenz Experiment #2 nicht. Das Experiment #6 kommt aber sehr nahe an die $\varnothing MR$ des Referenz Experiments und liefert mit dem Layer L6 Relu das zweitbeste Ergebnis über alle gemachten Experimente.

Auch hier wurden die Feststellungen von Lukic et al. [2] bestätigt, ein zusätzlicher Dense Layer L9 liefert eine bessere $\varnothing MR$ als das Experiment mit einem Layer weniger.

6.4 Batchgrösse

Die Grösse des Batches wurde von Hoffer et al. [3] übernommen und beträgt bei allen vorhergehenden Experimenten 32. Lukic et al. verwendeten jedoch Batchgrössen von 100 Samples. Das Experiment #7 soll zeigen, welche von beiden Batchgrössen zu bevorzugen ist. Bei nur 32 Samples pro Batch fliesst nur ein Bruchteil aller vorhandenen Sprecher in die Backpropagation ein. Bei einer Batchgrösse von 100 ist eine gleichmässigere Verteilung der Sprecher pro einzelner Backpropagation gegeben, was zu besseren Resultaten führen könnte.

#7 Batchgrösse 100 Dieses Experiment basiert auf dem Experiment #2 Standard ohne Relu, die Batchgrösse wurde von 32 auf 100 Samples vergrössert.

6.4.1 Resultate

Experiment	Layer	$\varnothing MR \uparrow$	$\varnothing LMR$
#7 Batchgrösse 100	L6 Relu	0.0655	0.1310
#7 Batchgrösse 100	L8 BiasAdd	0.1000	0.2083
#7 Batchgrösse 100	L6 BiasAdd	0.1113	0.2298

Tabelle 5: Resultat des Experiments #7. Die Ergebnisse sind schlechter als die des Referenz Experiments #2.

Experiment #7 ist schlechter als das Referenz Experiment #2. Das Netzwerk hat bei gleicher Anzahl Iterationen viel mehr Daten gesehen (ungefähr Faktor 3). Das wirkt sich aber auch negativ auf die Laufzeit des Experiments aus, welche sich von üblichen 20 Stunden auf nahezu 60

Stunden verdreifacht. Trotz der längeren Laufzeit und der grösseren Menge an Trainingsdaten pro Iteration wird hier kein neues Bestergebnis auf der 40-Sprecher Liste erreicht.

Auf den beiden grösseren Testlisten mit jeweils 60 und 80 Sprechern, übertrifft das Experiment #7 das Referenz Experiment #2 knapp und liefert damit die besten Resultate auf diesen Listen.

7 Resultate

In diesem Kapitel wollen wir die Ergebnisse aus dem Kapitel 6 in einen grösseren Kontext stellen und darin auswerten. Dabei vergleichen wir alle Experimente miteinander. Weiter vergleichen wir unsere Resultate mit den Resultaten von Lukic et al. [2] und erläutern unsere Erkenntnisse zum experimentellen Aufbau.

7.1 Gesamtvergleich

Dieser Abschnitt vergleicht die einzelnen Experimente untereinander. Dabei werden zusätzlich zur 40-Sprecher Liste, die 60-Sprecher- und die 80-Sprecher Liste (vgl. Anhang D) betrachtet.

Die nachfolgenden Tabellen sind wiederum identisch aufgebaut, wie bei den Resultaten der Experimenten (vgl. Kapitel 6).

7.1.1 40 Sprecher

Experiment	Layer	$\varnothing MR \uparrow$	$\varnothing LMR$
#2 Standard ohne Relu	L6 Relu	**0.0440**	0.0940
#6 Mit Dense Layer L9	L6 Relu	0.0470	0.1083
#7 Batchgrösse 100	L6 Relu	0.0655	0.1310
#1 Standard mit Relu	L6 Relu	0.0673	0.1405
#6 Mit Dense Layer L9	L8 BiasAdd	0.0738	0.1655

Tabelle 6: Fünf beste Resultate gemäss $\varnothing MR$ auf der 40-Sprecher Liste. Das Experiment #2 Standard ohne Relu schneidet am besten ab, was bereits aus Kapitel 6 bekannt war.

Das Experiment #2 Standard ohne Relu liefert das beste Ergebnis auf dem Layer L6 Relu. Diese Resultate sind bereits aus Kapitel 6 bekannt. Im Anhang F ist das dazugehörige Dendogramm gezeigt. Tabelle 6 zeigt die besten Ergebnisse über alle Experimente. Dabei ist deutlich zu erkennen, das der Layer L6 Relu überdurchschnittlich oft in den fünf besten Rängen vertreten ist und der geeignetste Layer für die Erzeugung der Embeddings zu sein scheint.

Abbildung 22 zeigt den Verlauf der MR von den fünf besten Resultaten über alle 30 Checkpoints. Man erkennt, wie die MR nach ca. 10'000 Iterationen langsam stabil wird. Das Resultat von Experiment #2 Standard ohne Relu und von Experiment #6 Mit Dense Layer L9 liegen häufig unter den anderen Kurven, was in einer tieferen $\varnothing MR$ resultiert. Bei beiden Resultaten, wurde der Layer L6 Relu zur Generierung der Embeddings verwendet.

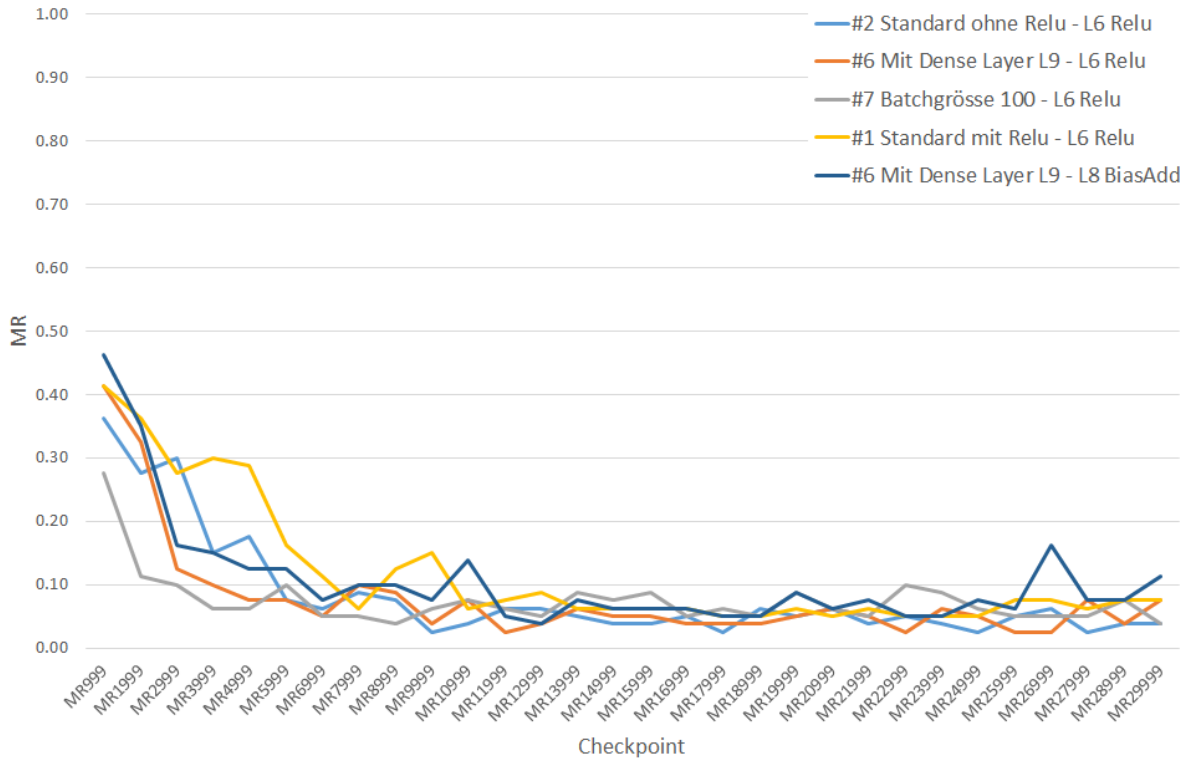

Abbildung 22: Verlauf der MR über 30'000 Iterationen der fünf besten Resultate. Zusätzlich zum Experimentname wurde der verwendete Layer für die Generierung der Embeddings angegeben.

7.1.2 60 Sprecher

Experiment	Layer	$\varnothing MR \uparrow$	$\varnothing LMR$
#7 Batchgrösse 100	L6 Relu	**0.0647**	0.1357
#6 Mit Dense Layer L9	L6 Relu	0.0671	0.1556
#2 Standard ohne Relu	L6 Relu	0.0687	0.1675
#1 Standard mit Relu	L6 Relu	0.0810	0.1810
#6 Mit Dense Layer L9	L8 BiasAdd	0.0857	0.2079

Tabelle 7: Fünf beste Resultate gemäss $\varnothing MR$ auf der 60-Sprecher Liste. Das Experiment #7 Batchgrösse 100 schneidet am besten ab.

Die Resultate sehen hier sehr ähnlich aus wie bei der 40-Sprecher Liste (vgl. Kapitel 7.1.1). Die drei besten Resultate sind sehr nahe beieinander, das Experiment #7 Batchgrösse 100 schneidet jedoch leicht besser ab. Die grössere Batchgrösse scheint sich positiv für grössere Testinglisten auszuwirken. Abgesehen davon sind die gleichen Experimente in den fünf besten Resultaten vertreten. Ebenso zeigt sich das gleiche Bild bei den Layern zur Generierung der Embeddings.

Die $\varnothing MR$ liegt bei der 60-Sprecher Liste leicht höher als bei der 40-Sprecher Liste, was aber zu erwarten war. Das Clustering von mehr unbekannten Sprechern ist schwieriger und wirkt sich direkt auf die $\varnothing MR$ aus.

7.1.3 80 Sprecher

Experiment	Layer	$\varnothing MR \uparrow$	$\varnothing LMR$
#7 Batchgrösse 100	L6 Relu	**0.0801**	0.1667
#6 Mit Dense Layer L9	L6 Relu	0.0831	0.1923
#2 Standard ohne Relu	L6 Relu	0.0840	0.1821
#1 Standard mit Relu	L6 Relu	0.0854	0.1929
#7 Batchgrösse 100	L6 BiasAdd	0.0994	0.2048

Tabelle 8: Fünf beste Resultate gemäss $\varnothing MR$ auf der 80-Sprecher Liste. Das Experiment #7 Batchgrösse 100 schneidet, wie auch auf der 60-Sprecher Liste, am besten ab.

Auch bei der 80-Sprecher Liste zeigt sich ein ähnliches Bild wie bei der 60-Sprecher Liste (vgl. Kapitel 7.1.2). Das Experiment #7 Batchgrösse 100 schneidet auch hier am besten ab und ist hier sogar ein zweites Mal mit dem Layer L6 BiasAdd vertreten. Dies bekräftigt die Annahme, dass sich die grössere Batchgrösse positiv auf grössere Sprecherlisten auswirkt. Jedoch sind die Verbesserungen minimal im Vergleich zu der erheblich grösseren Laufzeit (vgl. Kapitel 6.4).

Die $\varnothing MR$ liegt hier nochmals eine Stufe höher als bei der 60-Sprecher und der 40-Sprecher Liste.

7.2 Vergleich Lukic et al.

Dieser Abschnitt vergleicht unsere Resultate mit denen von Lukic et al. aus ihrer zweiten Arbeit [2]. Da Lukic et al. die LMR nach 10'000, 15'000, 20'000 und 30'000 Iterationen als Masse verwendet haben, werden in Tabelle 9 die selben Masse verwendet. Zusätzlich sind für zukünftige Vergleiche die neuen Masse ebenfalls dargestellt. Für das Testing wurde die gleiche 40-Sprecher Liste (vgl. Anhang D) verwendet.

Exp.	Layer	MR					LMR				
		$\varnothing$	10k	15k	20k	30k	$\varnothing$	10k $\uparrow$	15k	20k	30k
#2	L6 Relu	**0.044**	0.025	0.038	0.050	0.038	0.094	**0.050**	0.075	0.100	0.075
#6	L6 Relu	0.047	0.038	0.050	0.050	0.075	0.108	0.100	0.100	0.125	0.200
#7	L6 Relu	0.066	0.063	0.075	0.088	0.038	0.131	0.125	0.150	0.175	0.075
#2	L8 BiasAdd	0.094	0.050	0.075	0.113	0.088	0.199	0.125	0.150	0.225	0.175
#3	L6 Relu	0.094	0.050	0.088	0.113	0.138	0.202	0.125	0.200	0.275	0.325

Tabelle 9: Fünf beste Resultate gemäss LMR nach 10'000 Iterationen. Die erreichte LMR von 0.05 nach 10'000 Iterationen ist dabei auf dem gleichen Niveau wie das beste Resultat von Lukic et al. [2].

Die Resultate von Lukic et al. [2] werden hier ebenfalls erreicht. Unser Netzwerk, trainiert im Experiment #2 Standard ohne Relu, erreicht nach 10'000 Iterationen die selbe LMR von 0.05. Da die LMR, sowie die MR, zwischen verschiedenen Checkpoints stark schwanken kann (vgl. Kapitel 5.7.4), ist es jedoch eher Zufall, dass gerade beim Checkpoint nach 10'000 Iterationen die

tiefste LMR erzielt wurde. Trotzdem lässt sich daraus schliessen, dass unser Modell zu ähnlichen Resultaten fähig ist, wie das von Lukic et al.

7.3 Trainingsloss

An der Losskurve sieht man, wie gut das Netzwerk lernt. Ein tiefer Loss bedeutet, dass das Netzwerk sich erfolgreich an die Trainingsdaten anpassen kann. Die Losskurven von allen durchgeführten Experimenten sind in der Abbildung 23 ersichtlich.

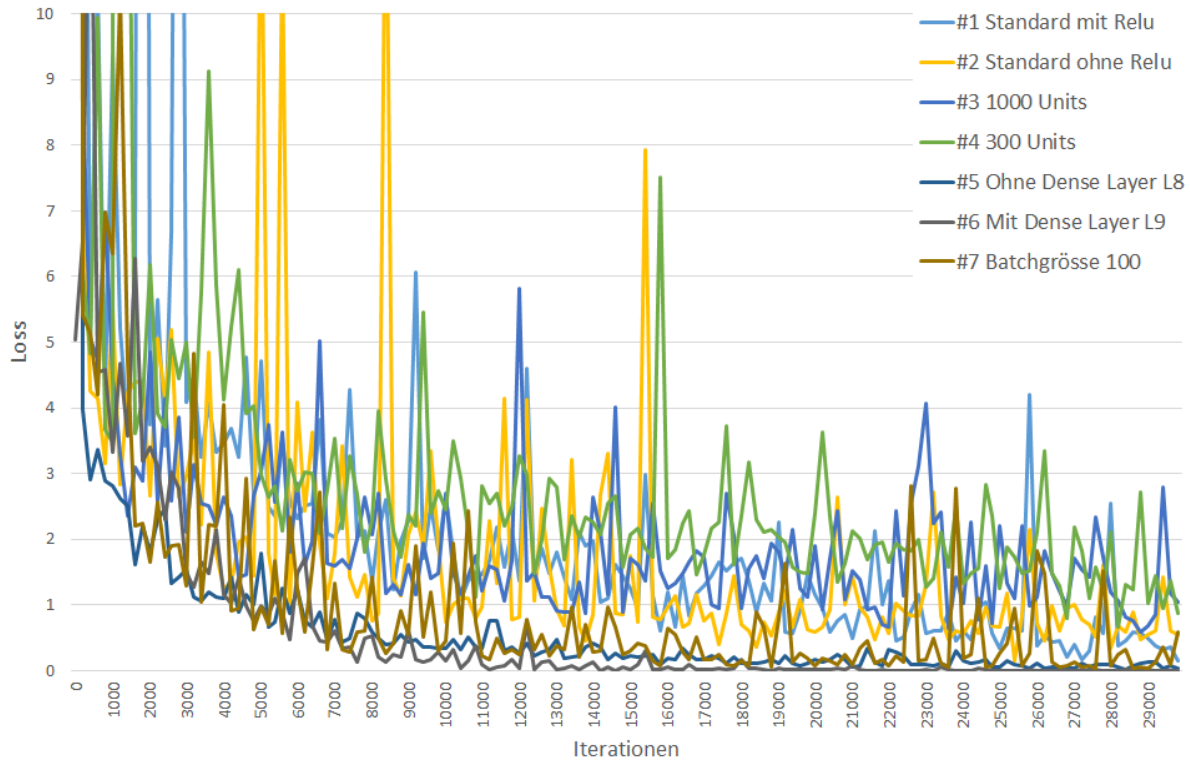

Abbildung 23: Trainings Loss aller durchgeführten Experimente. Es wurden 30'000 Iterationen trainiert und jeweils nach 200 Steps wurde der Loss gespeichert.

Auffällig ist, dass das beste Modell auf der 40-Sprecher Liste, #2 Standard ohne Relu, von einigen anderen Experimenten im Trainingsloss unterboten wurde. Die Experimente #7 Batchgrösse 100, #6 Mit Dense Layer L9 und #5 Ohne Dense Layer L8 haben alle eine Losskurve, welche deutlich unter der von Experiment #2 liegt.

Wir vermuten die Ursache dafür in folgenden zwei Punkten:

- Overfitting: Die sehr gute Anpassung der Netzwerke an die Trainingsdaten und das Erreichen von weniger guten Ergebnissen im Testing auf den Testdaten sind Zeichen für ein Overfitting.
- *End-to-End* Lernen: Dass ein guter Trainingsloss erreicht wird, die Resultate, gemessen mit der $\varnothing MR$, jedoch schlechter sind, deutet darauf hin, dass der verwendete Ansatz noch nicht ganz *End-to-End* ist. Das bedeutet, dass ein optimierter Loss nicht zwingend die $\varnothing MR$ minimiert.

Unabhängig von der Ursache schliessen wir daraus, dass eine vielversprechende Losskurve nicht zwingend eine gute $\varnothing MR$ bedeutet.

7.4 Zusammenfassung Experimente

Dieser Abschnitt fasst die gemachten Erkenntnisse aus den Resultaten der Experimente zusammen.

- Der letzte Layer vor der Loss-Funktion sollte keine Aktivierung besitzen, da diese die Embeddings auf einen positiven Wertebereich begrenzt.
- Der Layer L6, direkt nach dem CNN, eignet sich durchgehend am besten für die Erzeugung der Embeddings. Dies konnte bereits Lukic et al. [2] beobachten, jedoch verwendeten sie für das Training einen *Surrogate Task*. Es ist deshalb nicht vollständig klar, warum das bei unserem Ansatz mit der Loss-Funktion von Hoffer et al. [3], welche direkt Embeddings lernt, immer noch der Fall ist. Eine mögliche Ursache wäre, dass es noch immer kein vollständiger *End-to-End* Ansatz ist.
- Die Relu im Layer L6 übertrifft die Resultate des Layers L6 BiasAdd. Dies steht im Gegensatz zu den Beobachtungen im Layer L8, wo sich die Aktivierung negativ auf das Resultat ausgewirkt hat. Die Gründe, warum die Eingrenzung des Wertebereichs der Embeddings eine Verbesserung des Resultats zur Folge haben, sind uns unklar.
- Ein tieferer Trainingsloss bedeutet nicht unbedingt eine bessere $\varnothing MR$. Gründe dafür sind ein mögliches Overfitting oder ein Zeichen dafür, dass der verwendete Ansatz noch nicht vollständig *End-to-End* ist.
- Eine grössere Batchgrösse wirkt sich vor allem positiv auf die Auswertung mit den 60-Sprecher und 80-Sprecher Listen aus. Jedoch benötigt ein Experiment mit grösserer Batchgrösse erheblich mehr Laufzeit.

7.5 Experimentelles Setup

Neben den Experimenten hinterfragten wir das bisherige Experimentelle Setup und entwickelten einen Vorschlag für zukünftige Experimente. Dieser Abschnitt fasst die gemachten Vorschläge und die Beweggründe dazu zusammen.

7.5.1 Misclassification Rate

Die Misclassification Rate, welche von Kotti et al. [25] definiert wurde, lässt Interpretationsspielraum. Beim Vergleich von unserer eigenen Implementation der MR Berechnung mit der bestehenden von Lukic et al. fiel die unterschiedliche Auffassung der MR auf.

Im Zuge unserer Arbeit haben wir die MR nach unserem Verständnis genau definiert und verwenden in dieser Arbeit jeweils beide Interpretationen zum Vergleich. Wir schlagen vor, dass zukünftige Arbeiten die MR nach unserer Definition verwenden. Sie entspricht eher der Definition von Kotti et al. [25] und wurde bereits in anderen Arbeiten von Stadelmann et al. [23] [4] verwendet.

7.5.2 Mean Missclassification Rate

Für die Auswertung unserer Resultate haben wir ein neues Mass definiert. Da die MR über verschiedene Checkpoints stark variieren kann, macht es aus unserer Sicht Sinn, den Durchschnitt über mehrere Checkpoints zu verwenden. Dieses Mass $\varnothing MR$ berechnet sich aus dem Durchschnitt der MR-Werte aller Checkpoints ab Iteration 10'000 bis 30'000. Die $\varnothing MR$ verspricht stabiler und unabhängiger von einzelnen Checkpoints zu sein.

Wir empfehlen für zukünftige Arbeiten ebenfalls die $\varnothing MR$ zu verwenden.

7.5.3 Sprecherlisten

Die bisherige übliche Trainingsliste mit 100 verschiedenen Sprechern, sowie die existierende 40-Sprecher Liste für das Testing, bestehen nur aus Sprechern der Sprachregionen 1 bis 3. Für die Auswertung der Experimente wären Testlisten mit deutlich mehr Sprechern sehr interessant. Diese können jedoch nur realisiert werden, wenn Sprecher von anderen Sprachregionen verwendet werden. Da diese nicht in der Trainingsliste enthalten sind, würde man das trainierte Modell mit unbekannten Dialekten konfrontieren. Bessere Resultate würde man erzielen, wenn man in der Trainingsliste alle Sprechregionen vertreten hätte, sowie ähnlich viele Frauen wie Männer enthalten sind.

Wir schlagen ein Konzept vor, mit welchem man Trainings- und Testlisten beliebiger Grösse erstellen kann, welche die oben erwähnten Anforderungen an eine gleichmässige Verteilung erfüllen. Dafür haben wir alle 630 Sprecher in einer Liste zusammengefasst und zufällig sortiert. Mit den erstellen Skripten kann man nun Trainingslisten der gewünschten Grösse erstellen, sowie unterschiedlich grosse, aufeinander aufbauende Testlisten mit den verbleibenden Sprechern.

Mit unserer Basisliste ist die Nachvollziehbarkeit zukünftiger Experimente gewährleistet. Die Grösse der Trainingsliste und der Testliste ergeben mit der Basisliste zusammen eine eindeutige, reproduzierbare Liste. Zudem ist maximale Flexibilität gegeben, da Trainingslisten von beliebiger Grösse erstellt werden können. Wir denken, dass dieses Konzept geeignet ist für die Verwendung in weiterführenden Arbeiten zu diesem Thema und komplexere, sowie vielfältigere Experimente zulässt.

8 Diskussion

In diesem Kapitel interpretieren wir die gemachten Resultate und geben einen Ausblick über mögliche Stossrichtungen für weitere Experimente.

8.1 Rückblick auf die Aufgabenstellung

Unser Netzwerk mit der Loss-Funktion von Hoffer et al. [3] funktioniert und produziert Embeddings, welche für den Task des *Speaker Clustering* verwendet werden können. Der Vergleich mit den Resultaten von Lukic et al. [2] zeigt, dass wir nach ihren Massen gleich gute Ergebnisse erzielen. Die erreichte LMR vo 0.05 nach 10'000 Iterationen entspricht dem besten Ergebnis von Lukic et al. Damit haben wir gezeigt, dass dieser Ansatz grundsätzlich funktioniert und durchaus für weitere Experimente interessant ist.

Im Rahmen unserer Arbeit haben wir uns stark mit der Berechnung der MR sowie dem Aufbau und der Zusammensetzung der Sprecherlisten auseinander gesetzt. Dabei sind uns verschiedene Interpretationen der MR und die Einseitigkeit der Sprecherlisten aufgefallen. Für die MR haben wir eine genaue Definition erstellt und empfehlen zukünftigen Arbeiten diese zu verwenden. Für eine noch bessere Vergleichbarkeit definierten wir die durchschnittliche MR ($\varnothing MR$), welche die MR über mehrere Checkpoints mittelt. Mit einem neuen Konzept für die Erstellung der Trainings- und Testlisten wollen wir zukünftig flexiblere und komplexere Experimente ermöglichen.

Die Arbeit zeigt, dass der verwendete Ansatz zum Lernen von Embeddings grundlegend funktioniert und dabei Resultate erzeugt, welche mit den bisherigen mithalten können. Die Auseinandersetzung mit dem experimentellen Setup und den neuen vorgeschlagenen Massen und Konzepten bilden eine gute Grundlage für zukünftige Arbeiten und erleichtert deren Vergleichbarkeit und Reproduzierbarkeit.

8.2 Ausblick

Eine mögliche Weiterverfolgung dieses Ansatzes mit der Loss-Funktion von Hoffer et al. [3] wäre aus unserer Sicht sehr vielversprechend. Die bereits vorhandenen guten Resultate lassen sich möglicherweise durch weitere Optimierungen verbessern. Dabei sehen wir im Zusammenhang mit der Loss-Funktion von Hoffer et al. Potential in folgenden Bereichen:

- Optimierungen an der Netzwerkarchitektur und deren Parameter. Als vielversprechend haben sich die grössere Batchgrösse sowie die Experimente mit zusätzlichem Layer erwiesen.
- Einbau der Batch Normalisierung, wie es bereits Lukic et al. in ihrer zweiten Arbeit [2] realisiert haben.
- Crossentropie als zusätzliche Komponente der Loss-Funktion verwenden. Romanov et al. [27] zeigten in ihrer Arbeit, dass durch die gewichtete Summe der Crossentropie und einer Loss-Funktion für Embeddings (in unserem Fall Loss-Funktion von Hoffer et al. [3]) die Clustering Resultate verbessert werden können.

Die Loss-Funktion von Hoffer et al. besitzt einen zweiten Teil für *unsupervised* Lernen. Diesen zu verwenden, wäre eine weitere Möglichkeit und würde es erlauben, nicht annotierte Trainingsdaten zu verwenden. Damit könnte man auf grösseren Datensätzen trainieren und wäre nicht mehr auf den TIMIT Datensatz limitiert.

Einige der Experimente haben Hinweise geliefert, welche darauf hindeuten, dass der verwendete Ansatz noch nicht vollständig *End-to-End* ist. Dies ist auf der einen Seite die Diskrepanz zwischen ausserordentlich gutem Trainingsloss und weniger guten Resultaten bei der $\varnothing MR$. Auf der anderen Seite ist der positive Effekt der Relu-Aktivierung auf die Embeddings fragwürdig und ungeklärt. Aufgrund dieser zwei Indizien halten wir es für eine sinnvolle Möglichkeit, in diese Richtung weitere Untersuchungen anzugliedern und neue Ansätze zu evaluieren, welche dem *End-to-End* Ansatz eher gerecht werden.

9 Literatur

[1] Y. Lukic, C. Vogt, O. Dürr und T. Stadelmann, „Speaker Identification and Clustering Using Convolutional Neural Networks“, 2016.

[2] Y. Lukic, C. Vogt, O. Dürr und T. Stadelmann, „Learning embeddings for speaker clustering based on voice equality“, 2017 (Eingereicht an MLSP).

[3] E. Hoffer und N. Ailon, „Semi-supervised deep learning by metric embedding“, S. 1–9, 2016. Adresse: http://arxiv.org/abs/1611.01449.

[4] T. Stadelmann, *Voice modeling methods for automatic speaker recognition*, PhD thesis, 2010.

[5] Y. LeCun und Y. Bengio, „Convolutional networks for images, speech, and time series“, *The handbook of brain theory and neural networks*, Bd. 3361, Nr. April 2016, S. 255–258, 1995.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. Den, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, M. Leach, K. Kavukcuoglu, T. Graepel und D. Hassabis, „Mastering the Game of Go with Deep Neural Networks and Tree Search“, Nr. 1, S. 1–37, ISSN: 0028-0836. DOI: 10.1038/nature16961.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg und L. Fei-Fei, „ImageNet Large Scale Visual Recognition Challenge“, *International Journal of Computer Vision*, Bd. 115, Nr. 3, S. 211–252, 2015, ISSN: 15731405. DOI: 10.1007/s11263-015-0816-y. arXiv: 1409.0575. Adresse: https://arxiv.org/abs/1409.0575.

[8] Y. Lecun, L. Bottou, Y. Bengio und P. Haffner, „Gradient-Based Learning Applied to Document Recognition“, *Proc. IEEE 1998*, 1998.

[9] O. Abdel-hamid, L. Deng und D. Yu, „Exploring Convolutional Neural Network Structures and Optimization Techniques for Speech Recognition“, *14th Annual Conference of the International Speech Communication Association, (INTERSPEECH 2013)*, Nr. August, S. 3366–3370, 2013, ISSN: 15206149.

[10] S. Russell und P. Norvig, *Artificial Inteligence: A Modern Approach (3rd ed)*, 3rd. 2009, S. 1152, [Online; Zugriff am 08.06.2017], ISBN: 9780136042594. Adresse: http://aima.cs.berkeley.edu/.

[11] M. Muller, D. P. W. Ellis, A. Klapuri und G. Richard, „Signal processing for music analysis“, *IEEE Journal of Selected Topics in Signal Processing*, Bd. 5, Nr. 6, S. 1088–1110, Okt. 2011, ISSN: 1932-4553. DOI: 10.1109/JSTSP.2011.2112333.

[12] U. zu Köln, *Mel skala*, [Online; Zugriff am 29.05.2017]. Adresse: http://phonetik.phil-fak.uni-koeln.de/240.html.

[13] D. O'Shaughnessy, *Speech communication: Human and machine*, Ser. Addison-Wesley series in electrical engineering. Addison-Wesley Pub. Co., 1987, ISBN: 9780201165203. Adresse: https://books.google.ch/books?id=mHFQAAAAMAAJ.

[14] *Musical perception*, no one claims yet to have determined 'the' mel scale, 1970.

[15] S. Dieleman und B. Schrauwen, *End-to-end learning for music audio*. Mai 2014, S. 6964–6968. DOI: 10.1109/ICASSP.2014.6854950.

[16] O. Dürr, T. Stadelmann, Tolkachev, Sick und Stampfli, „Beyond imagenet – deep learning in industrial practice“, in *Applied Data Science – Lessons Learned for the Data-Driven Business*, M. Braschler, T. Stadelmann und K. Stockinger (Eds.), Hrsg., Spinger, 2018 (to appear).

[17] L. Kaufman und P. J. Rousseeuw, *Finding groups in data: An introduction to cluster analysis.* John Wiley, 1990.

[18] S. C. Johnson, „Hierarchical clustering schemes“, *Psychometrika*, Bd. 32, Nr. 3, S. 241–254, 1967, ISSN: 1860-0980. DOI: 10.1007/BF02289588. Adresse: http://dx.doi.org/10.1007/BF02289588.

[19] D. A. Reynolds, „Speaker identification and verification using gaussian mixture speaker models“, *Speech Communication*, Bd. 17, Nr. 1, S. 91 –108, 1995, ISSN: 0167-6393. DOI: http://dx.doi.org/10.1016/0167-6393(95)00009-D. Adresse: http://www.sciencedirect.com/science/article/pii/016763939500009D.

[20] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett und N. L. Dahlgren, *Darpa timit acoustic phonetic continuous speech corpus cdrom*, [Online; Zugriff am 08.06.2017], 1993. Adresse: https://catalog.ldc.upenn.edu/ldc93s1.

[21] Y. LeCun, C. Cortes und C. J. Burges, [Online; Zugriff am 08.06.2017]. Adresse: http://yann.lecun.com/exdb/mnist/.

[22] TensorFlow, *Getting started*, [Online; Zugriff am 15.05.2017]. Adresse: https://www.tensorflow.org/get_started/.

[23] T. Stadelmann und B. Freisleben, „Unfolding speaker clustering potential“, in *Proceedings of the seventeen ACM international conference on Multimedia - MM '09*, New York, New York, USA: ACM Press, 2009, S. 185, ISBN: 9781605586083. DOI: 10.1145/1631272.1631300. Adresse: http://portal.acm.org/citation.cfm?doid=1631272.1631300.

[24] H. Ghaemmaghami, D. Dean, S. Sridharan und D. A. van Leeuwen, „A study of speaker clustering for speaker attribution in large telephone conversation datasets“, *Comput. Speech Lang.*, Bd. 40, Nr. C, S. 23–45, Nov. 2016, ISSN: 0885-2308. DOI: 10.1016/j.csl.2016.03.005. Adresse: https://doi.org/10.1016/j.csl.2016.03.005.

[25] M. Kotti, V. Moschou und C. Kotropoulos, *Speaker segmentation and clustering*, 2008.

[26] D. Liu und F. Kubala, *Online speaker clustering*, Apr. 2003.

[27] A. Romanov und A. Rumshisky, „Forced to Learn: Discovering Disentangled Representations without Exhaustive Labels“, *ICLR*, 2017.

[28] A. Krishnamurthy, „High-dimensional clustering with sparse gaussian mixture models“, 2011.

[29] B. Logan, „Mel frequency cepstral coefficients for music modeling“, in *In International Symposium on Music Information Retrieval*, 2000.

[30] J. Shlens, „A Tutorial on Principal Component Analysis“, 2014. DOI: 10.1.1.115.3503. arXiv: 1404.1100. Adresse: http://arxiv.org/abs/1404.1100.

[31] X. Glorot, A. Bordes und Y. Bengio, „Deep sparse rectifier neural networks“, *AISTATS '11: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, Bd. 15, S. 315–323, 2011, ISSN: 15324435. DOI: 10.1.1.208.6449. arXiv: 1502.03167.

10 Glossar

Dense Layer Ein *Dense Layer* (oder *fully-connected Layer*) verbindet alle Input-Units mit allen Output-Units. Diese Eigenschaft ermöglicht es gelernte Features zu kombinieren. Häufig wird der Dense Layer deshalb in der höheren Layern eines Neuronalen Netzwerkes eingesetzt.

Dropout Layer Ein Dropout Layer verwirft zufällig einen Prozentsatz der Daten im Tensor. Dies wird gemacht um das Netzwerk vor Overfitting zu schützen.

GMM *Gaussian Mixture Models* (GMM) ist ein probabilistisches Model, welches auf der Gausschen Normalverteilung basiert. Es kann für Clustering-Aufgaben eingsetzt werden [28].

Hidden Layer Als Hidden Layer bezeichnet man alle Layer in einem Netzwerk, welche zwischen Input und Output Layer liegen.

MFFC *Mel-Frequenz-Cepstrum-Koeffizienten* (MFCC) ist ein Verfahren zur Feature Extraktion. MFCCs werden bei der Spracherkennung und bei der Erkennung von Musikstücken eingesetzt [29].

numpy Eine Python-Bibliothek für wissenschaftliche Berechnungen.

overfitting *Overfitting* (deutsch: Überanpassung) passiert, wenn sich ein Neuronales Netzwerk zu fest an die Trainingsdaten anpasst und danach nicht mehr fähig ist, generelle Probleme zu lösen.

PCA Mit einem PCA (*principal component analysis*, dt. Hauptkomponentenanalyse) Diagramm kann man höher dimensionale Daten vereinfachen, indem man sie durch weniger Dimensionen annähert und danach darstellen kann [30].

Pickle Eine Pickle-Datei ist ein Dateiformat um serialisierte Python Objekte abzuspeichern.

Relu Relu ist eine Aktivierungsfunktion und bedeutet *rectified linear unit* [31].

TensorBoard Das TensorBoard ist ein Visualisierungstool, welches in TensorFlow integriert ist.

A Aufgabenstellung

zhaw School of Engineering

[Online Administration Praktische Arbeiten]

[Dept. T Admin Tools]

zurück Logout

Bachelorarbeit 2017 - FS: BA17_stdm_4

Allgemeines:

Titel: Machine Learning for Speaker Clustering
Anzahl Studierende: 2

Betreuer:

HauptbetreuerIn: Thilo Stadelmann, stdm
NebenbetreuerIn: Oliver Dürr, dueo

Zugeteilte Studenten:

Diese Arbeit ist zugeteilt an:
- Jörg Egli, eglijoe1 (IT)
- Timon Gygax, gygaxtim (IT)

Fachgebiet:

BV	Bildverarbeitung
DA	Datenanalyse
SOW	Software

Studiengänge:

IT	Informatik
WI	Wirtschaftsingenieurwesen

Zuordnung der Arbeit :

InIT	Institut für angewandte Informationstechnologie

Infrastruktur:

benötigt keinen zugeteilten Arbeitsplatz an der ZHAW

Interne Partner :

IDP	Institut für Datenanalyse und Prozessdesign

Industriepartner:

Es wurden keine Industriepartner definiert!

Beschreibung:

Automatische Stimmerkennung ist eine wichtige Basistechnologie in verschiedenen kommerziellen Bereichen (z.B. biometrische Authentifizierung, Gesprächsanalyse für Teledienste, Medienmonitoring). Trotzdem liefern automatische Verfahren deutlich schlechtere Ergebnisse als menschliches Hören. Forschung der Betreuer hat kürzlich einen Weg gezeigt, diese Lücke zu schliessen: Mittels Verfahren des maschinellen Lernens, wie sie in der Analyse von Bildern zum Einsatz kommen, lässt sich der zeitliche Verlauf von Sprache (der "Klang") extrahieren und in einem für die Stimme charakteristischen Modell abspeichern. Insbesondere Deep Convolutional oder Recurrent Neural Networks (CNNs / RNNs) bieten sich hier an, die in den letzten Jahren zu einem Milliardengeschäft und heissesten Eisen in der Machine Learning Forschung geworden sind.

Ziel

Ziel dieser Bachelorarbeit ist es, "Speaker Clustering" weiterzuentwickeln. Hierzu ist ein geeigneter Ansatz aus der Literatur gemeinsam mit den Betreuenden auszuwählen, zu implementieren und experimentell mit dem wissenschaftlichen State of the Art zu vergleichen.

Vorgehen

- Kennenlernen des Hintergrunds von Speaker Clustering
- Einarbeiten in die relevante Literatur; einen Einblick liefert Rippel et al., "Metric Learning with Adaptive Density Discrimination", 2016
- Implementieren eines geeigneten Ansatzes (Auswahl zusammen mit Betreuern)
- Geeignetes Wählen von Einstellungen anhand der Leistung auf gegebenen Testdaten
- Evaluieren anhand systematischer Experimente und Vergleich mit existierendem Baseline Ansatz
- Darstellung und Diskussion der eigenen Ergebnisse im Licht des Standes der Forschung (BA-Bericht, Vortrag)

Informations-Link:

Unter folgendem Link finden sie weitere Informationen zum Thema:
https://dublin.zhaw.ch/~stdm/?page_id=77

Voraussetzungen:

Die Arbeit setzt kein Vorwissen im Bereich Machine Learning oder Stimmerkennung voraus. Freude am Programmieren und Experimentieren sowie Lust auf ein Auseinandersetzen mit einer wissenschaftlichen Fragestellung sind hingegen wichtig. So kann die Arbeit als Einstieg in forschungsnahes Arbeiten genutzt werden.

In vorangegangenen Projekt- und Bachelorarbeiten wurden bereits Ansätze positiv evaluiert und Ergebnisse international durch Studierende präsentiert. Eine experimentelle Umgebung für Stimmerkennung mit CNNs steht zur Verfügung, in der bekannte Datensätze geladen, analysiert und Ergebnisse ausgewertet werden können.

zurück Logout

B Konfiguration

Die verwendete Grundkonfiguration der einzelnen Experimente ist in Listing B.1 dargestellt. Die experimentspezifische Konfigurationsdatei überschrieb nur noch abweichende Parameter. Dies sind insbesondere der Experimentname und die Beschreibung. Es ist gezeigt in Listing B.2. Viele der Parameter wurden nie überschrieben und wurden nur zwecks zukünftiger Erweiterbarkeit als Parameter definiert.

```
[exp]
name = undefined ; Name of experiment, has to be unique
desc = undefined ; Description of experiment
dataset = timit

[train]
name = undefined ; Name of trainingslist
list = undefined
pickle = undefined
sentences_per_speaker = 10
total_speakers = 10

[validation]
name = speakers_40_not_clustering_vs_reynolds
list = ../data/speaker_lists/speakers_40_not_clustering_vs_reynolds.txt
pickle = ../data/training/TIMIT_extracted/
    speakers_40_not_clustering_vs_reynolds.pickle
sentences_per_speaker = 10
total_speakers = 40
samples = 50

[test]
output_layer = undefined ; Layer for embeddings generation during testing

[test_list1]
name = speakers_40_clustering_vs_reynolds
list = ../data/speaker_lists/speakers_40_clustering_vs_reynolds.txt
pickle1 = ../data/training/TIMIT_extracted/
    speakers_40_clustering_vs_reynolds_8.pickle
pickle2 = ../data/training/TIMIT_extracted/
    speakers_40_clustering_vs_reynolds_2.pickle
total_speakers = 40
sentences_per_speaker = 10
sentences_pickle1 = 8
sentences_pickle2 = 2

[test_list2]
name = speakers_60_clustering
list = ../data/speaker_lists/speakers_60_clustering.txt
pickle1 = ../data/training/TIMIT_extracted/speakers_60_clustering_8.pickle
pickle2 = ../data/training/TIMIT_extracted/speakers_60_clustering_2.pickle
total_speakers = 60
sentences_per_speaker = 10
sentences_pickle1 = 8
sentences_pickle2 = 2

[test_list3]
name = speakers_80_clustering
list = ../data/speaker_lists/speakers_80_clustering.txt
pickle1 = ../data/training/TIMIT_extracted/speakers_80_clustering_8.pickle
pickle2 = ../data/training/TIMIT_extracted/speakers_80_clustering_2.pickle
total_speakers = 80
sentences_per_speaker = 10
sentences_pickle1 = 8
sentences_pickle2 = 2
```

```
53
54 [spectrogram]
55 duration = 100 ;spectrogram width
56 intervall = 15 ;input data widht
57 frequency_elements = 128 ;spectrogram height
58
59 [output]
60 log_path = ../data/experiments/logs/
61 net_path = ../data/experiments/nets/
62 plot_path = ../data/experiments/plots/
63
64 [data]
65 timit_path = ../data/training/TIMIT_extracted/
66 mnist_path = ../data/training/MNIST/
67
68 [net]
69 batch_size = 32
70 max_iter = 30000 ; Number of iteration
71 sum_iter = 200 ; steps between summary write and overview print
72 chkp_iter = 1000 ;steps between checkpoint file write
73 norm_on = False
74 norm_eps = 1e-4
75 norm_mom = 0.1
76 pool_size = 4
77 pool_strides = 2
78 conv_kernel = 4
79 conv_pad = same
80 conv1_filter = 32
81 conv4_filter = 64
82 dropout_rate = 0.5
83 dense7_factor = 10
84 dense10_factor = 5
85 dense11_factor = 5
86
87 [optimizer]
88 name = adadelta ; [adadelta, adam, nesterov]
89 learning_rate = 1 ; adadelta: 1, adam: 0.001, nesterov: 0.001
90 eps = 1e-6 ; adadelta: 1e-6, adam: 1e-8, nesterov: -
91 momentum = 0.9 ; nesterov only
92 beta1 = 0.9 ; adam only
93 beta2 = 0.99 ; adam only
94 rho = 0.95 ; adadelta only
```

Listing B.1: Grundkonfiguration der Experimente.

```
[exp]
name = 026_100_layer11
desc = standard-configuration, 100 speakers, cpu-compatible. Based on 020
    plus one additional dense layer l11 (l10 activation=Relu, l11 activation=
    None)

[train]
list = ../data/speaker_lists/speakers_100_50w_50m_not_reynolds.txt
pickle = ../data/training/TIMIT_extracted/speakers_100_50w_50m_not_reynolds.
    pickle
sentences_per_speaker = 10
total_speakers = 100

[net]
dense10_factor = 8
dense11_factor = 5

[test]
output_layer = l11_dense/dense/BiasAdd:0
```

Listing B.2: Experimentspezifische Konfiguration, mit Beschreibung des Experiments und der, von der Grundkonfiguration abweichenden, Parameter.

C Skripte

Dieses Kapitel beinhaltet relevante Skripte, welche in dieser Arbeit verwendet wurden.

C.1 Überprüfung Loss-Funktion

Die Überprüfung der Loss-Funktion wird in Listing C.1 gezeigt.

```
import os
import numpy as np
import tensorflow as tf

X = np.array([
  [1, 2, 3, 4, 5, 6],
  [6, 1, 4, 8, 6, 8],
  [9, 7, 2, 2, 3, 3],
  [2, 4, 7, 1, 8, 9],
], dtype=np.float32)

Z = np.array([
  [2, 2, 3, 4, 4, 6],
  [2, 3, 7, 2, 9, 9],
], dtype=np.float32)

X_labels = np.array([0, 1, 1, 0])

# Numpy
result = np.zeros((4))
for i, x in enumerate(X):
  dividende = np.exp(np.negative(np.linalg.norm(np.subtract(x, Z[X_labels[i
    ]]), ord=2)))

  divisor = np.zeros((1))
  for z in Z:
    dist = np.exp(np.negative(np.linalg.norm(np.subtract(x, z), ord=2)))
    divisor = np.add(divisor, dist)
  result[i] = np.negative(np.log(np.divide(dividende, divisor)))

result = np.mean(result)
print('Ausgabe numpy')
print(result)

# Tensorflow
x_sym = tf.placeholder(tf.float32, shape=(4, 6))
x_labels_sym = tf.placeholder(tf.int32, shape=(4))
z_sym = tf.placeholder(tf.float32, shape=(2, 6))

loss = tf.map_fn(lambda x: tf.add(tf.norm(tf.subtract(x[0], z_sym[x[1]]), ord
    ='euclidean'), tf.log(tf.reduce_sum(tf.map_fn(lambda z: tf.exp(tf.
    negative(tf.norm(tf.subtract(x[0], z), ord='euclidean'))), z_sym), 0))),
    (x_sym, x_labels_sym), dtype=tf.float32)

loss = tf.reduce_mean(loss)

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
loss_value = sess.run(loss, feed_dict={x_sym: X, x_labels_sym: X_labels,
    z_sym: Z})

print('Ausgabe TensorFlow')
print(loss_value)
```

Listing C.1: Vergleich der Implementierung der Loss-Funktion mit `numpy` und TensorFlow.

C.2 Generierung Sprecherlisten

```
import numpy as np

source_path = '../data/exp_setup_fs17/speakers_all.txt'
target_path = '../data/exp_setup_fs17/exp_setup_fs17.txt'

with open(source_path, 'r') as source:
  speakers = [speaker for speaker in source]

np.random.shuffle(speakers)

with open(target_path, 'w') as target:
  for speaker in speakers:
    target.write(speaker)
```

Listing C.2: Skript zur Erzeugung der Basisliste. Wurde nur für die initiale Erzeugung der Basisliste verwendet und ist hier nur zu Dokumentationszwecken gezeigt.

```
import argparse
import os

import numpy as np

parser = argparse.ArgumentParser(description='Creating speakerlists for training and testing a neural network.')
parser.add_argument('-train', type=int, dest='traininglist_size', default='100', help='Number of speakers which are used for training.')
args = parser.parse_args()

path = '../data/exp_setup_fs17/'
source_file = 'exp_setup_fs17_baselist.txt'
total_speakers = 630
testing_lists = np.arange(20, total_speakers - args.traininglist_size, 20)

# open the pre-shuffled list with all 630 speakers
with open(path + source_file, 'r') as source:
  speakers = [speaker for speaker in source]
  training_speakers = speakers[0:args.traininglist_size]

# create folder for lists if not exists
listfolder = os.path.join(path, 'traininglist_{}'.format(args.traininglist_size))
if not os.path.exists(listfolder):
  os.makedirs(listfolder)
# create the traininglist
with open(os.path.join(listfolder, 'traininglist_{}.txt'.format(args.traininglist_size)), 'w') as target:
  for training_speaker in training_speakers:
    target.write(training_speaker)
  print('traininglist_{}.txt created!'.format(args.traininglist_size))

# create the corresponding testing-lists
for testing_list in testing_lists:
  with open(os.path.join(listfolder, 'testlist_{}.txt'.format(testing_list)), 'w') as target:
    testing_speakers = np.flipud(speakers[total_speakers - testing_list:total_speakers])
    for testing_speaker in testing_speakers:
      target.write(testing_speaker)
    print('testlist_{}.txt created!'.format(testing_list))
```

Listing C.3: Skript zur Generierung von Sprecherlisten. Als Parameter wird die gewünschte Grösse der Testliste benötigt. Das Skript erstellt dann die Trainingsliste und eine Sammlung von Testlisten verschiedener Grössen. Anwendung: `python create_speakerlist.py -train 100`

D Sprecherlisten

Für die Experimente wurden vordefinierte Sprecherlisten verwendet. Die nachfolgenden Listings definieren die einzelnen Listen. Die Bezeichnungen der Sprecher sind aus dem TIMIT-Datensatz.

D.1 Training

Sprecherliste, welche für das Training des Netzwerkes verwendet wurde.

100 Specher

Enthält 50 weibliche und 50 männliche Sprecher.

- FCJF0
- FDAW0
- FDML0
- FECD0
- FETB0
- FJSP0
- FKFB0
- FMEM0
- FSAH0
- FSJK1
- FSMA0
- FTBR0
- FVFB0
- FVMH0
- FAEM0
- FAJW0
- FCAJ0
- FCMM0
- FCYL0
- FDAS1
- FDNC0
- FDXW0
- FEAC0
- FHLM0
- FJKL0
- FKAA0
- FLMA0
- FLMC0
- FMJB0
- FMKF0
- FMMH0
- FPJF0
- FRLL0
- FSCN0
- FSKL0
- FSRH0
- FTMG0
- FALK0
- FCKE0
- FCMG0
- FDFB0
- FDJH0
- FEME0
- FGCS0
- FGRW0
- FJLG0
- FJLR0
- FLAC0
- FLJD0
- FLTM0
- MCPM0
- MDAC0
- MDPK0
- MEDR0
- MGRL0
- MJEB1
- MJWT0
- MKLS0
- MKLW0
- MMGG0
- MPGH0
- MPGR0
- MPSW0
- MRAI0
- MRCG0
- MRDD0
- MRSO0
- MRWS0
- MTJS0
- MTPF0
- MTRR0
- MWAD0
- MWAR0
- MARC0
- MBJV0
- MCEW0
- MCTM0
- MDBP0
- MDEM0
- MDLB0
- MDLC2
- MDMT0
- MDPS0
- MDSS0
- MDWD0
- MEFG0
- MHRM0
- MJAE0
- MJBG0
- MJDE0
- MJEB0
- MJHI0
- MJMA0
- MJMD0
- MJPM0
- MJRP0
- MKAH0
- MKAJ0
- MKDT0

D.2 Testing

Specherlisten, welche für das Testing des trainierten Netzwerkes verwendet wurden.

40 Sprecher

- MPGL0
- MSTK0
- FCMR0
- MWBT0
- MCEM0
- MRGG0
- MBJK0
- MPDF0
- MDAB0
- MMDM2
- MGWT0
- FDAC1
- FPKT0
- FJAS0
- MCCS0
- MMDB1
- MDLD0
- MJAR0
- FAKS0
- FELC0
- MTAS1
- MSJS1
- FJEM0
- MWEW0
- MREB0
- FJRE0
- MWVW0
- FRAM1
- MRCZ0
- MABW0
- FCMH0
- MRJO0
- FPAS0
- FJWB0
- FDRD1
- MJSW0
- FKMS0
- FSLB1
- MTMR0
- MDBB0

60 Sprecher

- FAKS0
- FDAC1
- FELC0
- FJEM0
- MDAB0
- MJSW0
- MREB0
- MRJO0
- MSJS1
- MSTK0
- MWBT0
- FCMR0
- FDRD1
- FJAS0
- FJRE0
- FJWB0
- FPAS0
- FRAM1
- FSLB1
- MABW0
- MBJK0
- MCCS0
- MCEM0
- MDBB0
- MDLD0
- MGWT0
- MJAR0
- MMDB1
- MMDM2
- MPDF0
- MPGL0
- MRCZ0
- MRGG0
- MTAS1
- MTMR0
- MWEW0
- MWVW0
- FCMH0
- FKMS0
- FPKT0
- MBDG0
- MBWM0
- MCSH0
- MCTW0
- MGJF0
- MGLB0
- MHPG0
- MJBR0
- MJES0
- MJJG0
- MJMP0
- MJVW0
- MKCH0
- MLNT0
- MMAB0
- MMDH0
- MMJR0
- MMWH0
- MRTK0
- MTAA0

80 Sprecher

- FAKS0
- FDAC1
- FELC0
- FJEM0
- MDAB0
- MJSW0
- MREB0
- MRJO0
- MSJS1
- MSTK0
- MWBT0
- FCMR0
- FDRD1
- FJAS0
- FJRE0
- FJWB0
- FPAS0
- FRAM1
- FSLB1
- MABW0
- MBJK0
- MCCS0
- MCEM0
- MDBB0
- MDLD0
- MGWT0
- MJAR0
- MMDB1
- MMDM2
- MPDF0
- MPGL0
- MRCZ0
- MRGG0
- MTAS1
- MTMR0
- MWEW0
- MWVW0
- FCMH0
- FKMS0
- FPKT0
- MBDG0
- MBWM0
- MCSH0
- MCTW0
- MGJF0
- MJES0
- MJJG0
- MJMP0
- MJVW0
- MKCH0
- MLNT0
- MMAB0
- MMDH0
- MMJR0
- MMWH0
- MRTK0
- MTAA0
- MTDT0
- MTHC0
- MWJG0
- FADG0
- FCFT0
- FCRH0
- FDMS0
- FEDW0
- FGJD0
- FJLM0
- FJMG0
- FLBW0
- FLKD0
- FMAF0
- FMCM0
- FNMR0
- FREW0
- FRNG0
- FSEM0
- MBNS0

D.3 Basisliste

Die zufällig sortierte Liste aller 630 Sprecher, welche wir als Grundlage für die Generierung von neuen Trainings- und Testlisten vorschlagen. Im Gegensatz zu den vorhergehenden Listen ist hier die Reihenfolge relevant.

1. MBAR0
2. MRTK0
3. MSMS0
4. FMJB0
5. FGMD0
6. MDLF0
7. MDMA0
8. MSVS0
9. MRRE0
10. FCRZ0
11. MRVG0
12. FJAS0
13. FTAJ0
14. MTAB0
15. MJLS0
16. MDVC0
17. MRTC0
18. MEAL0
19. MRXB0
20. MSAT1
21. FGWR0
22. MRGG0
23. MHPG0
24. FPKT0
25. MPFU0
26. MPAM0
27. FGDP0
28. MREE0
29. MKES0
30. MJDG0
31. MJAC0
32. FLMK0
33. MPAR0
34. MPLB0
35. MTDP0
36. MMDB0
37. MMDM1
38. MTKD0
39. FBLV0
40. MDLC1
41. MCHH0
42. FDRW0
43. MSRR0
44. MJBR0
45. FEDW0
46. FMAH0
47. FSKL0
48. MRHL0
49. FSJS0
50. MCAL0
51. MMRP0
52. MESD0
53. FAJW0
54. MAKB0
55. FSPM0
56. MTPR0
57. FGCS0
58. MMDM2
59. MRGM0
60. FUTB0
61. MSDH0
62. FKAA0
63. MRJM1
64. MKLN0
65. MJDA0
66. MAEB0
67. MDED0
68. MFWK0
69. FPMY0
70. MRJM3
71. MCLM0
72. FDAS1
73. MTEB0
74. MWRP0
75. FBCH0
76. MHJB0
77. FCFT0
78. MTMN0
79. MCPM0
80. MFXS0
81. FAWF0
82. FSLS0
83. MBPM0
84. FTLH0
85. MJFR0
86. MMSM0
87. FREW0
88. MBNS0
89. MTJM0
90. FJRP1
91. MKDB0

92. FJEN0
93. MDLD0
94. MRMS1
95. FCAG0
96. MRPC0
97. MPRK0
98. MTMT0
99. FCLT0
100. MDSS0
101. MRDS0
102. MDAC0
103. FETB0
104. FJWB1
105. MDWH0
106. MJFC0
107. MMWH0
108. MWRE0
109. MWEW0
110. FDJH0
111. MDTB0
112. MCDD0
113. MTJS0
114. MLNS0
115. FCRH0
116. MJWT0
117. FJRE0
118. MTKP0
119. MTCS0
120. FDAW0
121. FDMS0
122. FCMM0
123. MLSH0
124. MRDM0
125. MVLO0
126. FLAG0
127. MPGR0
128. FRNG0
129. FCMH1
130. FLNH0
131. MRWA0
132. MROA0
133. FMGD0
134. MRMS0
135. MKLW0
136. FBJL0
137. FCDR1
138. FDMY0
139. MAJC0
140. MJJJ0
141. MEGJ0
142. MJTC0
143. MPGH0
144. MDAS0
145. MMBS0
146. MRJH0
147. FJKL0
148. MDRM0
149. MHMR0
150. MRAB1
151. MHBS0
152. MTJG0
153. MEDR0
154. MWAD0
155. MGES0
156. MRPP0
157. FLHD0
158. MKJO0
159. MAHH0
160. MCRC0
161. MKLR0
162. MBCG0
163. FJSP0
164. MARW0
165. MPEB0
166. MKAG0
167. MBMA0
168. MFMC0
169. MRJM0
170. MMEB0
171. MAEO0
172. MCAE0
173. MMAB0
174. FALK0
175. FHXS0
176. MFRM0
177. MSMC0
178. MMDS0
179. MMLM0
180. FLAC0
181. FLEH0
182. FMJU0
183. MJRA0
184. MDWM0
185. MKDD0
186. MMDH0
187. FDAC1
188. MJKR0
189. MBWP0
190. MLLL0
191. MKXL0
192. MLJB0
193. MCLK0
194. MSAS0
195. MSAH1
196. FNMR0
197. FADG0
198. MBBR0
199. MCEF0
200. MLEL0
201. MKLS1
202. MPWM0
203. MGAF0
204. MESG0
205. MDSJ0
206. MJDM1
207. MRAB0
208. FRAM1
209. FSDC0
210. MREM0
211. MCMB0
212. MCSH0
213. MRLK0
214. MTWH0
215. FMMH0
216. FMCM0
217. MMVP0
218. MSFV0
219. FPAF0
220. MKAM0
221. MWAR0
222. FMPG0
223. MSAT0
224. MRFL0
225. MRPC1
226. FLKM0
227. MJAI0
228. MDHS0
229. MKLS0
230. MWBT0
231. MJDE0
232. MSTF0
233. FEXM0
234. FELC0
235. FLJD0
236. MDCM0
237. FSAG0
238. MRJB1
239. FLJG0
240. FSJK1
241. MPSW0
242. FCYL0
243. FMML0
244. MGAW0
245. MTQC0
246. FMAF0
247. FJXP0
248. MDSS1
249. MMDG0
250. MWVW0
251. FJSK0
252. MCXM0
253. MHRM0
254. FGRW0
255. FRJB0
256. MGLB0
257. MBOM0
258. MLBC0
259. MMDM0
260. MCEW0
261. FSGF0
262. FKDW0
263. MTPF0
264. MBML0
265. MTPP0
266. MSFH0
267. MGRL0
268. MGSL0
269. MTWH1
270. FSKP0
271. FBAS0
272. MWSB0
273. FSJG0
274. MJVW0
275. MWDK0
276. MDLR0
277. FDKN0
278. FLMA0
279. MJJM0
280. MREH1
281. MJMM0
282. FTBR0
283. MTAA0
284. MTRC0
285. MJRH1
286. FCEG0
287. MKDT0
288. MSJS1
289. MDHL0
290. FCJF0
291. MMGK0
292. MPAM1
293. MRMB0
294. MSRG0
295. FDNC0
296. MRAI0
297. FPLS0
298. FJEM0
299. MDLC0
300. FSMS1
301. MRJS0
302. FCMR0
303. FPAS0
304. FJCS0
305. MJWS0
306. MLJH0
307. MJSR0
308. FCAL1
309. FKLH0
310. MJJG0
311. MBTH0
312. FHLM0
313. FGJD0
314. FSRH0
315. FJLR0
316. MTXS0
317. FKLC0
318. MAJP0
319. FJSJ0
320. FSLB1
321. MGJF0
322. MDNS0
323. MSLB0
324. MKLT0
325. FAEM0
326. MJXA0
327. MRLJ0
328. MJRG0
329. FSMA0
330. MRES0
331. MTDT0
332. MJFH0
333. MMXS0
334. FVKB0
335. MDWA0
336. MSDB0
337. MRWS0
338. MEJL0
339. MBGT0
340. MILB0
341. MFGK0
342. MJPM0
343. MPRB0
344. FPAD0
345. MDPK0
346. MCCS0
347. FDTD0
348. MTAT0
349. FSCN0
350. FSEM0
351. FLOD0
352. MPAB0
353. MMWS0
354. MBSB0
355. FKDE0
356. MJHI0
357. FNTB0
358. MJEE0
359. MMAA0
360. MRDD0
361. FMJF0
362. FKLC1
363. MWSH0
364. MDKS0
365. FPAZ0
366. MHIT0
367. MTLC0
368. MDRB0
369. MRWS1
370. MRBC0
371. MTBC0

372. MREW1
373. FSAK0
374. FMBG0
375. MKCL0
376. MPGL0
377. MMJB1
378. FJXM0
379. MBMA1
380. FJDM2
381. FKFB0
382. FEEH0
383. MJJB0
384. MCSS0
385. FEAC0
386. MGRT0
387. FNLP0
388. MJDM0
389. MBDG0
390. FISB0
391. FSMM0
392. MJAR0
393. FALR0
394. MFXV0
395. MDLC2
396. MJTH0
397. MWCH0
398. MCDC0
399. MRJO0
400. MDPS0
401. MBJK0
402. MRCS0
403. MGMM0
404. MGWT0
405. FASW0
406. MMAG0
407. MREB0
408. MCDR0
409. FMKC0
410. MDPB0
411. FJRB0
412. MDLS0
413. FLBW0
414. MTRT0
415. MERS0
416. MPGR1
417. FPAB1
418. FHES0
419. MSJK0
420. FMEM0
421. FSDJ0
422. MWGR0
423. MTHC0
424. MRMG0
425. MADD0
426. MNET0
427. FJHK0
428. MESJ0
429. MAFM0
430. MGXP0
431. MJPM1
432. FKKH0
433. MRML0
434. MTAT1
435. MRLR0
436. FSBK0
437. MRCZ0
438. MRTJ0
439. MJRH0
440. FMLD0
441. MABC0
442. FJMG0
443. FBCG1
444. FEAR0
445. MMPM0
446. MPRD0
447. FDML0
448. MRKO0
449. MEJS0
450. MZMB0
451. MMGC0
452. MRRK0
453. MDLB0
454. MMJR0
455. FBMJ0
456. FSAH0
457. MJEB1
458. MMGG0
459. MDBP0
460. MNTW0
461. MDAW1
462. MEWM0
463. MSES0
464. MJLN0
465. MPDF0
466. MDLM0
467. MKRG0
468. MCTM0
469. FGMB0
470. MGAR0
471. MKCH0
472. MRJR0
473. MAPV0
474. MJMP0
475. MJXL0
476. MRAM0
477. MDDC0
478. MGJC0
479. FJWB0
480. FLAS0
481. MBJV0
482. MCEM0
483. MMAR0
484. FLET0
485. MRAV0
486. MTMR0
487. FDFB0
488. MJRK0
489. FBMH0
490. MWJG0
491. MJAE0
492. MEFG0
493. FSXA0
494. MDAC2
495. MJMD0
496. MARC0
497. MKJL0
498. FCAJ0
499. MBEF0
500. MGAG0
501. MMEA0
502. MRJT0
503. MSFH1
504. MDRD0
505. MNJM0
506. MPCS0
507. FLJA0
508. MDWD0
509. MSMR0
510. MTAS0
511. MVJH0
512. FJLG0
513. FPAC0
514. MRMH0
515. MHXL0
516. MRGS0
517. FDXW0
518. MJMA0
519. FMKF0
520. MRLJ1
521. MDAB0
522. MMCC0
523. FTLG0
524. FREH0
525. MCTW0
526. FAPB0
527. MRLD0
528. MTAS1
529. MMAB1
530. MKAH0
531. MTRR0
532. FCJS0
533. FVFB0
534. MDEF0
535. MJEB0
536. FJLM0
537. MPMB0
538. MTER0
539. FTBW0
540. MJDH0
541. MLNT0
542. MJLB0
543. FKMS0
544. FAKS0
545. FHEW0
546. MFER0
547. MDJM0
548. FTMG0
549. MLJC0
550. FCKE0
551. MGAK0
552. MDLH0
553. MDBB0
554. MCMJ0
555. MJRF0
556. MDBB1
557. MMDB1
558. FDHC0
559. MDWK0
560. MKDR0
561. MSEM1
562. MJRP0
563. MLIH0
564. MRJM4
565. MDMT0
566. MSDS0
567. MHMG0
568. MRCW0
569. MJWG0
570. MTPG0
571. MGRP0
572. MADC0
573. FKSR0
574. MPPC0
575. MJBG0
576. MVRW0
577. FCMH0
578. MMWB0
579. MJSW0
580. MTML0
581. FSKC0
582. MWAC0
583. FPJF0
584. MDLR1
585. MRSO0
586. MJLG1
587. MCTH0
588. MJES0
589. MAKR0
590. MDSC0
591. MTLB0
592. FRLL0
593. MNLS0
594. FSJW0
595. MDEM0
596. MMWS1
597. FDRD1
598. FNKL0
599. FSSB0
600. MSTK0
601. FJSA0
602. FMAH1
603. MJDC0
604. MCHL0
605. MKAJ0
606. FLMC0
607. FLTM0
608. MGSH0
609. FLKD0
610. MBWM0
611. MCRE0
612. FEME0
613. MCTT0
614. MPRT0
615. MRKM0
616. MRCG0
617. FCAU0
618. MRFK0
619. MRSP0
620. MJPG0
621. MDCD0
622. FVMH0
623. FECD0
624. MTJU0
625. MMAM0
626. MTLS0
627. MWEM0
628. MABW0
629. FCMG0
630. MTDB0

E Übersicht Experimente

#	Experiment	Besonderheit
1	Standard mit Relu	Basiskonfiguration, dient als Ausgangslage für alle späteren Experimente. Denselayer L8 hat eine Relu-Aktivierung und 500 Units. Die Batchgrösse ist 32.
2	Standard ohne Relu	Die Aktivierung im letzten Dense Layer L8 wurde weggelassen.
3	1000 Units	Basiert auf #2. Der Dense Layer L8 hat jedoch 1000 Units, statt 500.
4	300 Units	Basiert auf #2. Der Dense Layer L8 hat jedoch 300 Units, statt 500.
5	Ohne Dense Layer L8	Basiert auf #2. Dense Layer L8 wurde weggelassen, sowie die Aktivierung im Dense Layer L6.
6	Mit Dense Layer L9	Basiert auf #2. Ein weiterer Dense Layer L9 wurde angehängt. Die Aktivierung von L8 ist Relu, bei L9 wurde sie weggelassen.
7	Batchgrösse 100	Basiert auf #2. Die Batchgrösse wurde auf 100 erhöht.

Tabelle 10: Übersicht der durchgeführten Experimente und ihre Besonderheiten.

Die nachfolgenden Abschnitte zeigen die Resultate aller Experimente pro Testliste, sortiert nach der $\varnothing MR$.

E.1 40 Sprecher

Name Doku	Outplut Layer	min. MR	Threshold	min. MR @ ckpt	ø MR ab 10k	Diagramm	min. LMR2	min. LMR @ ckpt2	ø LMR ab 10k	LDiagramm
#2 Standard ohne Relu	L6 Relu	0.0250	MR9999	MR9999	0.0440		0.0500	LMR9999	0.0940	
#6 Mit Dense Layer L9	L6 Relu	0.0250	MR11999	MR11999	0.0470		0.0500	LMR11999	0.1083	
#7 Batchgrösse 100	L6 Relu	0.0375	MR8999	MR8999	0.0655		0.0750	LMR8999	0.1310	
#1 Standard mit Relu	L6 Relu	0.0500	MR17999	MR17999	0.0673		0.1000	LMR18999	0.1405	
#6 Mit Dense Layer L9	L8 BiasAdd	0.0375	MR12999	MR12999	0.0738		0.1000	LMR12999	0.1655	
#2 Standard ohne Relu	L6 BiasAdd	0.0625	MR14999	MR14999	0.0887		0.1250	LMR14999	0.1833	
#3 1000 Units	L6 Relu	0.0500	MR9999	MR9999	0.0935		0.1000	LMR27999	0.2024	
#2 Standard ohne Relu	L8 BiasAdd	0.0375	MR10999	MR10999	0.0940		0.0750	LMR10999	0.1988	
#7 Batchgrösse 100	L8 BiasAdd	0.0500	MR16999	MR16999	0.1000		0.1250	LMR3999	0.2083	
#6 Mit Dense Layer L9	L6 BiasAdd	0.0875	MR9999	MR9999	0.1006		0.1750	LMR13999	0.2095	
#6 Mit Dense Layer L9	L8 Relu	0.0500	MR18999	MR18999	0.1042		0.1000	LMR18999	0.2369	
#5 Ohne Dense Layer L8	L6 BiasAdd	0.0875	MR6999	MR6999	0.1048		0.1750	LMR16999	0.2095	
#7 Batchgrösse 100	L6 BiasAdd	0.0500	MR11999	MR11999	0.1113		0.1000	LMR11999	0.2298	
#6 Mit Dense Layer L9	L9 BiasAdd	0.0750	MR15999	MR15999	0.1179		0.1750	LMR15999	0.2595	
#4 300 Units	L6 Relu	0.0500	MR28999	MR28999	0.1179		0.1250	LMR28999	0.2548	
#1 Standard mit Relu	L8 BiasAdd	0.0750	MR10999	MR10999	0.1214		0.1500	LMR18999	0.2583	
#1 Standard mit Relu	L6 BiasAdd	0.0875	MR7999	MR7999	0.1232		0.1750	LMR7999	0.2714	
#1 Standard mit Relu	L8 Relu	0.0750	MR19999	MR19999	0.1238		0.1750	LMR15999	0.2881	
#3 1000 Units	L8 BiasAdd	0.0875	MR8999	MR8999	0.1351		0.1750	LMR8999	0.3000	
#3 1000 Units	L6 BiasAdd	0.1000	MR14999	MR14999	0.1452		0.2000	LMR14999	0.3119	
#4 300 Units	L8 BiasAdd	0.1125	MR28999	MR28999	0.1619		0.2500	LMR26999	0.3393	
#4 300 Units	L6 BiasAdd	0.1000	MR21999	MR21999	0.1750		0.2250	LMR21999	0.3810	

E.2 60 Sprecher

Name Doku	Outplut Layer	min. MR	Threshold	min. MR @ ckpt	ø MR ab 10k	Diagramm	min. LMR2	min. LMR @ ckpt2	ø LMR ab 10k	LDiagramm
#7 Batchgrösse 100	L6 Relu	0.0333	MR18999	MR18999	0.0647		0.0667	LMR18999	0.1357	
#6 Mit Dense Layer L9	L6 Relu	0.0250	MR22999	MR22999	0.0671		0.0500	LMR22999	0.1556	
#2 Standard ohne Relu	L6 Relu	0.0417	MR17999	MR17999	0.0687		0.1167	LMR21999	0.1675	
#1 Standard mit Relu	L6 Relu	0.0500	MR17999	MR17999	0.0810		0.1000	LMR17999	0.1810	
#6 Mit Dense Layer L9	L8 BiasAdd	0.0500	MR22999	MR22999	0.0857		0.1333	LMR14999	0.2079	
#7 Batchgrösse 100	L6 BiasAdd	0.0583	MR11999	MR11999	0.0901		0.1167	LMR11999	0.1921	
#7 Batchgrösse 100	L8 BiasAdd	0.0500	MR18999	MR18999	0.1032		0.1167	LMR18999	0.2230	
#5 Ohne Dense Layer L8	L6 BiasAdd	0.0667	MR17999	MR17999	0.1048		0.1333	LMR17999	0.2254	
#6 Mit Dense Layer L9	L6 BiasAdd	0.0833	MR12999	MR12999	0.1079		0.1833	LMR12999	0.2413	
#2 Standard ohne Relu	L8 BiasAdd	0.0667	MR10999	MR10999	0.1159		0.1333	LMR10999	0.2587	
#2 Standard ohne Relu	L6 BiasAdd	0.0750	MR23999	MR23999	0.1159		0.1500	LMR23999	0.2540	
#3 1000 Units	L6 Relu	0.0833	MR20999	MR20999	0.1167		0.1833	LMR20999	0.2540	
#1 Standard mit Relu	L8 BiasAdd	0.0833	MR18999	MR18999	0.1345		0.1833	LMR18999	0.2968	
#4 300 Units	L6 Relu	0.0917	MR28999	MR28999	0.1357		0.2167	LMR22999	0.2897	
#1 Standard mit Relu	L8 Relu	0.0917	MR26999	MR26999	0.1369		0.2167	LMR26999	0.3048	
#6 Mit Dense Layer L9	L8 Relu	0.1000	MR10999	MR10999	0.1397		0.2333	LMR12999	0.3159	
#1 Standard mit Relu	L6 BiasAdd	0.1167	MR10999	MR10999	0.1544		0.2667	LMR10999	0.3278	
#3 1000 Units	L8 BiasAdd	0.1167	MR8999	MR8999	0.1560		0.2167	LMR8999	0.3317	
#6 Mit Dense Layer L9	L9 BiasAdd	0.1000	MR6999	MR6999	0.1591		0.2167	LMR24999	0.3508	
#3 1000 Units	L6 BiasAdd	0.1083	MR14999	MR14999	0.1631		0.2167	LMR14999	0.3333	
#4 300 Units	L8 BiasAdd	0.1333	MR28999	MR28999	0.1774		0.3000	LMR28999	0.3825	
#4 300 Units	L6 BiasAdd	0.1000	MR20999	MR20999	0.1774		0.2167	LMR20999	0.3690	

E.3 80 Sprecher

Name Doku	Outplut Layer	min. MR	Threshold	min. MR @ ckpt	ø MR ab 10k	Diagramm	min. LMR2	min. LMR @ ckpt2	ø LMR ab 10k	LDiagramm
#7 Batchgrösse 100	L6 Relu	0.0500	MR3999	MR3999	0.0801		0.1000	LMR3999	0.1667	
#6 Mit Dense Layer L9	L6 Relu	0.0563	MR18999	MR18999	0.0831		0.1250	LMR18999	0.1923	
#2 Standard ohne Relu	L6 Relu	0.0625	MR15999	MR15999	0.0840		0.1375	LMR12999	0.1821	
#1 Standard mit Relu	L6 Relu	0.0625	MR18999	MR18999	0.0854		0.1375	LMR18999	0.1929	
#7 Batchgrösse 100	L6 BiasAdd	0.0688	MR28999	MR28999	0.0994		0.1500	LMR11999	0.2048	
#5 Ohne Dense Layer L8	L6 BiasAdd	0.0813	MR16999	MR16999	0.1069		0.1625	LMR18999	0.2214	
#3 1000 Units	L6 Relu	0.0563	MR12999	MR12999	0.1149		0.1125	LMR12999	0.2423	
#6 Mit Dense Layer L9	L8 BiasAdd	0.0938	MR6999	MR6999	0.1152		0.2000	LMR14999	0.2631	
#2 Standard ohne Relu	L6 BiasAdd	0.0813	MR12999	MR12999	0.1235		0.1625	LMR12999	0.2625	
#6 Mit Dense Layer L9	L6 BiasAdd	0.1000	MR20999	MR20999	0.1271		0.2125	LMR20999	0.2714	
#1 Standard mit Relu	L8 BiasAdd	0.1000	MR20999	MR20999	0.1343		0.2250	LMR20999	0.3065	
#7 Batchgrösse 100	L8 BiasAdd	0.0813	MR3999	MR3999	0.1343		0.1375	LMR3999	0.2935	
#4 300 Units	L6 Relu	0.1063	MR21999	MR21999	0.1363		0.2500	LMR28999	0.3036	
#2 Standard ohne Relu	L8 BiasAdd	0.0500	MR17999	MR17999	0.1402		0.1250	LMR17999	0.2988	
#1 Standard mit Relu	L6 BiasAdd	0.1000	MR16999	MR16999	0.1438		0.2375	LMR16999	0.3119	
#6 Mit Dense Layer L9	L8 Relu	0.1188	MR6999	MR6999	0.1622		0.2500	LMR6999	0.3512	
#3 1000 Units	L8 BiasAdd	0.1313	MR8999	MR8999	0.1688		0.2750	LMR8999	0.3530	
#3 1000 Units	L6 BiasAdd	0.1250	MR14999	MR14999	0.1711		0.2500	LMR12999	0.3577	
#1 Standard mit Relu	L8 Relu	0.1188	MR10999	MR10999	0.1765		0.2500	LMR10999	0.3869	
#6 Mit Dense Layer L9	L9 BiasAdd	0.1375	MR6999	MR6999	0.1836		0.2500	LMR6999	0.3905	
#4 300 Units	L6 BiasAdd	0.1313	MR21999	MR21999	0.1869		0.3125	LMR21999	0.4054	
#4 300 Units	L8 BiasAdd	0.1562	MR10999	MR10999	0.1914		0.3625	LMR10999	0.4298	

F Dendogramm

Vom besten Resultat des Experimentes #2 Standard ohne Relu zeigt die Abbildung 24 das Dendogramm.

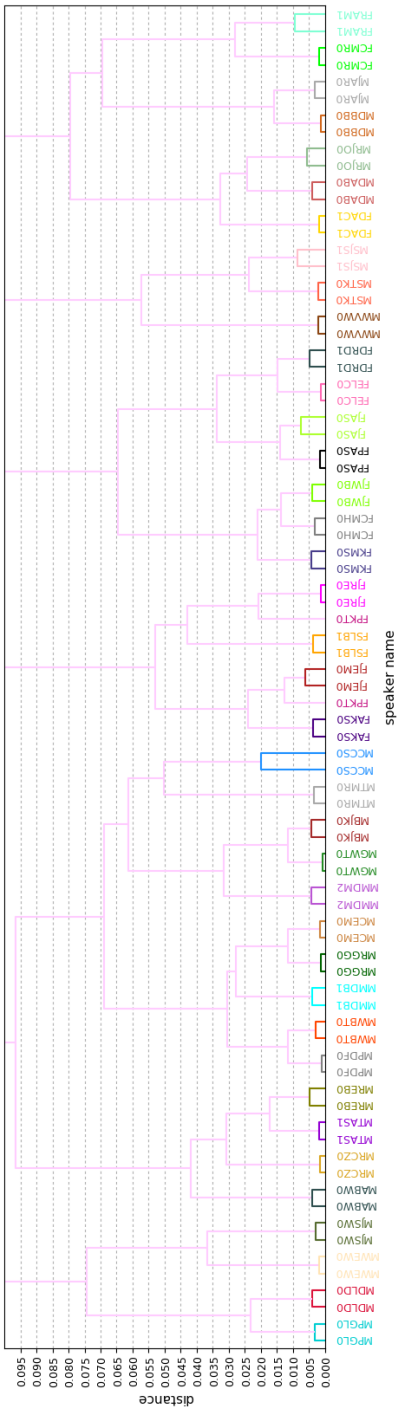

Abbildung 24: Dendogramm vom Experiment #2 Standard mit Relu, Embeddings wurden im Layer L6 Relu erzeugt.

G CD

Auf der beiliegenden CD sind folgende Inhalte enthalten:

- Software Code
  - Konfigurationsdateien der Experimente
  - Vollständige Auswertung der Experimente (Dendogramm, t-SNE Plot, MR und LMR)
  - Verwendete Sprecherlisten für Training und Testing
- neue Sprecherlisten und die dazu gehörende Basisliste
- Übersicht Analyse Experimente (Excel)