Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

# Projektarbeit
# HS15 Studiengang Informatik

## Sprechererkennung mit Convolutional Neural Networks

| | |
|---|---|
| **Autoren** | Yanick Lukic<br>Carlo Vogt |
| **Hauptbetreuung** | Oliver Dürr<br>Thilo Stadelmann |

| | |
|---|---|
| **Datum** | 17. Dezember 2015 |

# Zusammenfassung

Automatische Stimmerkennung ist eine wichtige Basistechnologie in verschiedenen kommerziellen Bereichen. Trotz dieser Relevanz liefern automatische Verfahren deutlich schlechtere Ergebnisse als das menschliche Gehör.

Die vorliegende Arbeit beschäftigt sich damit das Problem der Sprecheridentifikation mithilfe von Convolutional Neural Networks (CNNs) zu lösen. Hierfür wurden verschiedene Experimente durchgeführt und ausgewertet. Zusätzlich wurde ein erster Ansatz im Bereich des Sprecher-Clusterings experimentell evaluiert.

Unsere implementierten Systeme erreichten für die Sprecher-Identifikation von 630 Sprechern eine Genauigkeit von 97% auf Daten die dem System unbekannt waren. Die Testdaten die für die Evaluation verwendet wurden, waren pro Sprecher im Durchschnitt 5 Sekunden lang. State of the Art Verfahren erreichen auf den gleichen Datensätzen knapp 100% [19] [14].

Im Bereich des Sprecher-Clusterings konnten wir zeigen, dass dieses mittels neuronalen Netzwerken möglich ist. Wir erreichten für 20 Sprecher eine Misclassification Rate von 40%, andere Verfahren erreichen mit dem gleichen Experimentaufbau Werte von 0% [22].

Die Resultate zeigen, dass neuronale Netzwerke in der Lage sind aus Audio-Daten geeignete Features zur Identifizierung von Sprechern zu extrahieren. Dabei werden mit State of the Art Verfahren vergleichbare Ergebnisse erreicht. Dies zeigt auf, dass neuronale Netzwerke für die Arbeit mit gesprochener Sprache geeignet sind.

Des Weiteren konnte gezeigt werden, dass Sprecher-Clustering mit neuronalen Netzwerken möglich ist, die erreichte Qualität ist aber noch nicht mit bestehenden Verfahren vergleichbar. Die verwendeten Ansätze scheinen vielversprechend, sollten weiterverfolgt und bei Bedarf erweitert werden.

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

# Erklärung betreffend das selbständige Verfassen einer Projektarbeit an der School of Engineering

Mit der Abgabe dieser Projektarbeit 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 Projektarbeit 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 Projektarbeiten zu Beginn der Dokumentation nach dem Abstract bzw. dem Management Summary mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

# Inhaltsverzeichnis

# 1. Einleitung

In diesem Kapitel werden Ausgangslage, Aufgabenstellung, Motivation und Aufbau der Projektarbeit beschrieben.

## 1.1. Ausgangslage

Diese Projektarbeit wurde am Institut für angewandte Informationstechnologie (InIT) an der ZHAW School of Engineering erarbeitet. Die Betreuer waren Thilo Stadelmann und Oliver Dürr. Die Arbeit baut auf der Bachelorarbeit Deep Learning für automatische Stimmerkennung von Gabriel Eyyi [3] auf. Dieser versuchte das Speaker Clustering Problem mit Convolutional Neural Network zu lösen.

## 1.2. Zielsetzung

Es wird die Problematik der Speaker-Identification vertieft betrachtet. Durch Verwendung von Convolutional Neural Networks soll die automatische Stimmerkennung ausgearbeitet werden. Mit bildlicher Darstellung von Stimmen soll das Convolutional Neural Network in der Lage sein, diese entsprechenden Personen zuzuordnen.

## 1.3. Motivation

In vielen Bereichen der Wirtschaft ist die automatische Stimmerkennung ein wichtiger Grundbaustein. Die Technologien sind jedoch bei weitem nicht so genau wie das menschliche Erkennen von Stimmen. Mit dem Gebrauch von Convolutional Neural Networks (CNNs) könnte dieser Unterschied von automatisierter und menschlicher Stimmerkennung verringert werden. Dies könnte zum Beispiel im Bereich der biometrischen Authentifizierung zu grösserer Akzeptanz der Stimmerkennung führen.

## 1.4. Aufgabenstellung

Mithilfe der Python-Bibliotheken *lasange* und *nolearn.lasagne* ein CNN aufbauen, das anhand von Spektrogrammen Sprecher unterscheiden und erkennen kann.

## 1.5. Aufbau

Im Kapitel 2 Grundlagen werden die relevanten Technologien und Ideen vorgestellt, die in der Projektarbeit verwendet werden. Es stellt einen Einstieg in das Thema der CNNs und Spracherkennung dar. Im folgenden Kapitel 3 Vorgehen und Methoden wird der Aufbau der Versuchsumgebung erklärt und welche Technologien dafür eingesetzt wurden. In Kapitel 4 und 5 werden die Experimente und Auswertungen zu Speaker-Identification und Speaker-Clustering dokumentiert. Ein Ausblick anhand des Fazits vermittelt das letzte Projektarbeitskapitel.

# 2. Grundlagen

In diesem Kapitel werden die notwendigen Grundlagen dieser Arbeit vermittelt.

## 2.1. Deep Learning

Deep Learning ist ein breiter Begriff und verfügt über mehrere Definitionen. Im Buch *Deep Learning - Methods and Applications* von Li Deng und Dong Yu [1] werden fünf verschiedene Definitionen verwendet. Im Folgenden sind zwei davon aufgeführt.

„A class of machine learning techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification.“ [1] S. 199

Deep Learning bezeichnet also nicht eine spezifische Methode, sondern eine breite Palette an Möglichkeiten. Diese werden eingesetzt um über ein mehrstufiges Modell Informationen aus gegebenen Daten zu extrahieren und diese analytisch zu verarbeiten. Die zweite Definition ist umfangreicher.

„Deep learning is a set of algorithms in machine learning that attempt to learn in multiple levels, corresponding to different levels of abstraction. It typically uses artificial neural networks. The levels in these learned statistical models correspond to distinct levels of concepts, where higher-level concepts are defined from lower-level ones, and the same lower-level concepts can help to define many higher-level concepts.“ [1] S. 200

Hier wird auch die am meisten verwendete Methode von Deep Learning erwähnt, die Verwendung von künstlichen Neuronalen Netzwerken. Auf diesen basiert die vorliegende Arbeit.

## 2.2. Künstliche neuronale Netzwerke

Dieser Abschnitt bezieht sich auf das Buch von Michael A. Nielsen [16].

### 2.2.1. Perzeptron

Perzeptronen wurden in den 1950er und 1960er Jahren von Frank Rosenblatt entwickelt. Ein Perzeptron nimmt mehrere binäre Eingabewerte $x_1, x_2, ...$ entgegen. Aus diesen wird dann ein einzelner binärer Ausgabewert generiert.

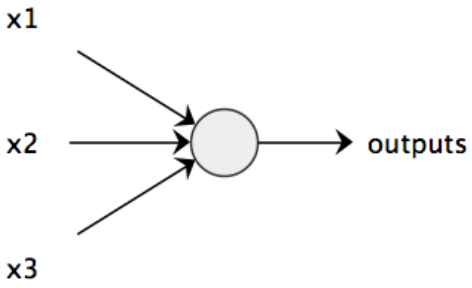



Abbildung 2.1.: Ein Perzeptron.

Jedem Eingabewert wird ein Gewicht zugewiesen, das die Wichtigkeit des Wertes in Bezug auf die Ausgabe darstellt. Ob der Ausgabewert 0 oder 1 ist, hängt davon ab, ob die gewichtete Summe $\sum_j w_j x_j$ grösser oder kleiner als eine definierte Schwelle (Threshold) ist. Die Schwelle und die Gewichte sind dabei reelle Zahlen, die als Parameter eines Neurons gesehen werden. Mathematisch ausgedrückt:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases} \tag{2.1}$$

Eine Alternative für die Schwelle ist der *Bias*. Hierfür wird $\sum_j w_j x_j$ umgeschrieben in $w \cdot x \equiv \sum_j w_j x_j$, dabei sind $w$ und $x$ jeweils Vektoren mit den Gewichten und Eingabewerten. Des Weiteren wird die Schwelle auf die andere Seite der Ungleichung genommen, $b \equiv -\text{threshold}$. Der mathematische Ausdruck sieht dann folgendermassen aus:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \tag{2.2}$$

Der Bias kann dabei als eine Masseinheit gesehen werden, wie einfach es ist, das Perzeptron dazu zu bringen eine 1 als Ausgabe zu generieren.

### 2.2.2. Sigmoid-Neuron (Logistik-Neuron)

Sigmoid-Neuronen sind sehr ähnlich wie Perzeptronen. Sie ermöglichen, dass kleine Anpassungen der Gewichte, auch in kleinen Änderungen der Ausgabe resultieren. Bei Perzeptronen können bereits kleine Anpassungen zur Folge haben, dass die Ausgabe von 0 zu 1 wechselt, was starke Auswirkungen auf den Rest des Netzwerks haben kann. Der Unterschied ist also, dass die Eingabewerte $x_1, x_2, \ldots$ nicht nur 0 und 1, sondern auch z.B. 0.345 sein können. Die Ausgabe wird dabei ebenfalls nicht mit 0 oder 1 angegeben, sondern wird durch $\sigma(w \cdot x + b)$ definiert. $\sigma$ ist dabei die Sigmoid-Funktion (manchmal auch Logistik-Funktion) und wird folgendermassen definiert:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} \tag{2.3}$$

Dabei ist $z = \sum_j w_j x_j - b$ mit den Eingabewerten $x_1, x_2, \ldots$ und den Gewichten $w_1, w_2, \ldots$.

Trotz allem ist das Sigmoid-Neuron sehr ähnlich zum Perzeptron, denn ist $z$ eine sehr grosse positive Zahl, wird die Ausgabe des Neurons näherungsweise 1. Ist $z$ jedoch eine stark negative Zahl, dann gilt $e^{-z} \to \infty$ und $\sigma(z) \approx 0$. Das Sigmoid-Neuron unterscheidet sich also nur dann vom Perzeptron, wenn $z$ eine moderat grosse Zahl ist. In modernen Neuronalen Netzwerken werden meist Sigmoid-Neuronen verwendet.

Die Gegenüberstellung der Graphen eines Sigmoid-Neurons und eines Perzeptrons zeigen deren Unterschied noch klarer auf.

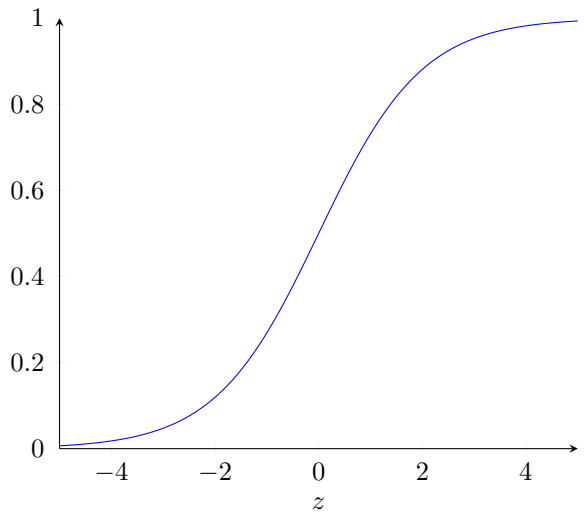



Abbildung 2.2.: Sigmoid-Funktion.

Die Sigmoid-Funktion hat einen fliessenden Übergang von 0 zu 1. Bei einem Perzeptron sieht die Funktion anders aus:

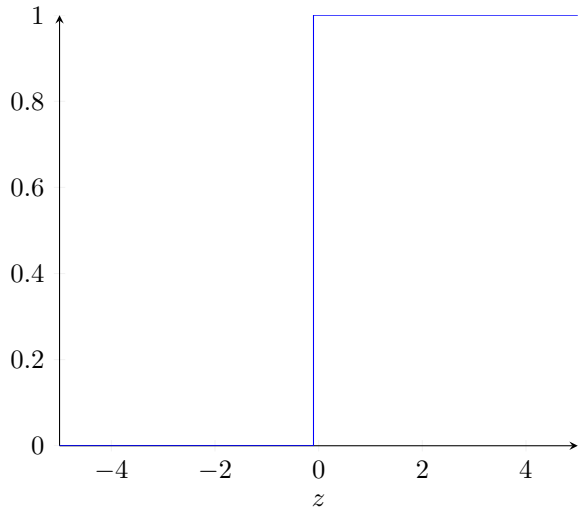



Abbildung 2.3.: Schrittfunktion eines Perzeptrons.

Wäre $\sigma$ also eine Schrittfunktion gäbe es keinen Unterschied zwischen dem Sigmoid-Neuron und einem Perzeptron. Die Sigmoid-Funktion nennt man hier die Aktivierungsfunktion des Neurons. Manchmal werden auch andere Funktionen verwendet. Eine ebenfalls sehr häufig verwendete Aktivierungsfunktion ist der *Rectifier*. Diese ist wie folgt definiert:

$$\varphi(z) = \max(0, z) \tag{2.4}$$

Wird in einem Neuron eine Rectify-Funktion verwendet, spricht man von einer Rectified Linear Unit (ReLU)[25].

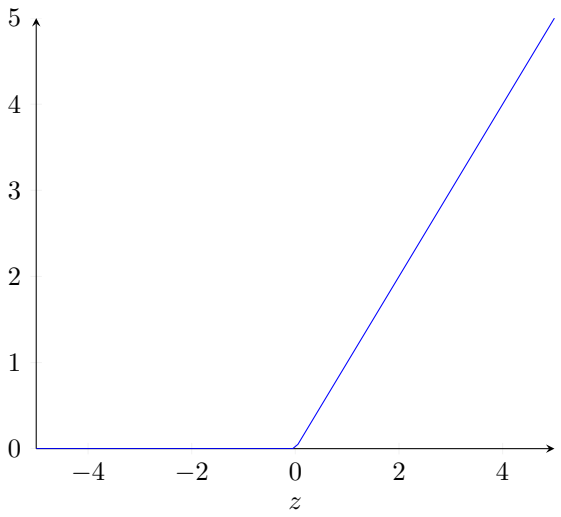



Abbildung 2.4.: Rectified Linear Unit (ReLU) Aktivierungsfunktion [25].

### 2.2.3. Architektur von neuronalen Netzwerken

Ein neuronales Netzwerk besteht immer aus einer Eingabe-Ebene (Input-Layer), einer Ausgabe-Ebene (Output-Layer) und dazwischen beliebigen versteckten Ebenen (Hidden-Layers). Alle Ebenen die keine Eingabe- oder Ausgabe-Ebenen sind, sind versteckte Ebenen.

Das Design der Eingabe- und Ausgabe-Ebene ist meist durch die zu bewältigende Aufgabe gegeben. Für die versteckten Ebenen gibt es verschiedene heuristische Methoden.

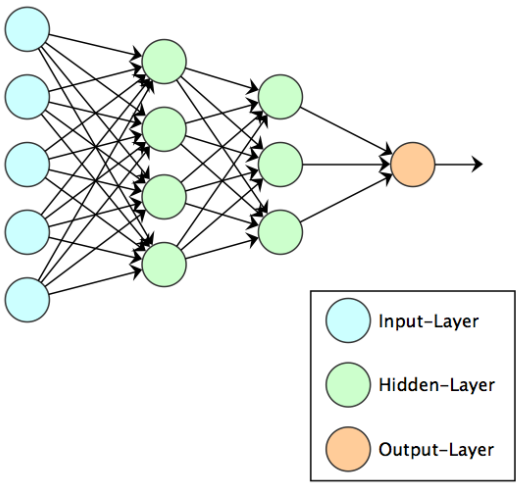



Abbildung 2.5.: Ebenen eines neuronalen Netzwerkes.

Es gibt verschiedene Arten von neuronalen Netzwerken:

- *feedforward*: Es gibt keine Schleifen im Netzwerk. Informationen werden also immer vorwärts gereicht und niemals zurück.
- *recurrent*: In diesen Netzwerken sind Schleifen möglich.

### 2.2.4. Kostenfunktion (cost function, auch lost oder objective function)

Die Kostenfunktion berechnet den Fehler den ein neuronales Netzwerk macht. Sie bezieht sich jeweils auf die aktuelle Gewichte- und Bias-Konfiguration.

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2. \tag{2.5}$$

Dabei steht $w$ für eine Sammlung aller Gewichte und $b$ für alle Biases, $n$ ist die Anzahl Trainingseinheiten. Die Summe wird über alle Trainingseingabevektoren $x$ gebildet, wobei $a$ der Vektor aller Ausgaben ist. $\|v\|$ steht für die Berechnung der Vektorlänge. $C$ bezeichnet die quadratische Kostenfunktion (engl. *quadratic cost function*, *mean squared error* oder *MSE*).

Das Ziel ist nun mit spezifischen Optimierungs- bzw. Trainingsmethoden die Gewichte und Biases so anzupassen, dass $C(w,b)$ möglichst klein wird.

## 2.3. Optimierungsmethoden

Um die Gewichte von neuronalen Netzwerken zu optimieren gibt es verschiedene Methoden. Einige davon werden im Folgenden vorgestellt. Des Weiteren werden die wichtigsten Begriffe der Trainingsphase erörtert.

### 2.3.1. Gewichtsmodifikationsarten

- **Supervised Learning**: Anhand vorgegebener Outputs werden die Gewichte angepasst. [21]

  Klassische Methode: Decision Tree

  Alternative Methode: Support Vector Machine (SVM)

- **Reinforcement Learning**: Es wird kein exakter Output vorgegeben, lediglich ob das Resultat richtig oder falsch ist. [5]
- **Unsupervised Learning**: Es werden keine Outputs vorgegeben. Der Algorithmus muss von den Inputs alleine lernen (Clustering). [21]

### 2.3.2. Backpropagation

Backpropagation bezeichnet ein *Supervised Learning*-Verfahren. Dabei werden die Eingabedaten vorwärts durch ein Netz eingeführt. Anhand des Fehlers wird über die Ausgabeschicht zurück zur Eingabeschicht propagiert. Die Gewichte werden dann abhängig von ihrem Einfluss auf den Fehler angepasst.

Im Folgenden wird der Algorithmus Schritt für Schritt aufgezeigt. $\odot$ beschreibt elementweises Multiplizieren von zwei Vektoren (Hadamard-Produkt) [16].

1. **Input** x: Weise der Eingabeschicht einen entsprechenden Aktivierungswert $a^1$ zu.
2. **Feedforward**: Berechne für jede Schicht $l = 2, 3, ..., L$ den gewichteten Input $z^l = w^l a^{l-1} + b^l$ und den Aktivierungswert $a^l = \sigma(z^l)$.
3. **Output error** $\delta^L$: Berechne den Fehlervektor $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4. **Backpropagate the error**: Führe für jede Schicht $l = L-1, L-2, ..., 2$ die Berechnung $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$ durch.
5. **Output**: Der Gradient der Kostenfunktion ist gegeben durch $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ und $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

### 2.3.3. Accuracy

Um die Qualität eines neuronalen Netzwerks zu bewerten, können verschiedene Messwerte verwendet werden. Meist wird auf die Accuracy (Genauigkeit) des Netzwerks zurückgegriffen. Diese ist wie folgt definiert [12]:

$$accuracy(t) = \frac{t}{N} \tag{2.6}$$

*N* ist dabei die totale Anzahl Objekte und *t* sind die richtig klassifizierten Objekte.

### 2.3.4. Softmax-Funktion

Oft will man aus dem Output eines neuronalen Netzwerkes ablesen können, wie gross die Wahrscheinlichkeit für die jeweiligen Klassen ist. Hierfür wird meistens die Softmax-Funktion verwendet. Diese ist eine Generalisierung der logistischen Funktion, die einen $K$-dimensionalen Vektor $z$ in einen Vektor $\sigma(z)$ mit reelen Werten konvertiert. Dabei haben die Werte einen Definitionsbereich zwischen 0 und 1 und summieren auf 1. Die Funktion ist wie folgt definiert [18] [13]:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{für} j = 1, ..., K \tag{2.7}$$

### 2.3.5. Epochen

Sind alle Trainingsdaten einmal vorwärts und wieder rückwärts durch das Netzwerk gewandert, spricht man von einer Epoche. In einer vor- und rückwärts Iteration sind mehrere Trainingsdaten enthalten, deren Anzahl bezeichnet man als die Batch-Size. Die Gewichte werden dabei jeweils erst nach Abschluss einer Iteration, also nach einem kompletten Durchlauf eines sogenannten Mini-Batches angepasst. [16]
*Beispiel*: Hat man 1000 Trainingsdaten und die Batch-Size beträgt 500, dann benötigt es 2 Iterationen um eine Epoche abzuschliessen.

Für die Implementation von neuronalen Netzwerken ist es wichtig zu wissen, dass die Batch-Size essenziell für den Speicherbedarf des Systems ist. Verfügt die verwendete Hardware über wenig dynamischen Speicher, kann eine kleinere Batch-Size oft die Lösung sein.

### 2.3.6. Overfitting

Overfitting bezeichnet die Situation, wenn ein statistisches Modell oder ein Machine-Learning Algorithmus so gut auf die Trainingsdaten passt, dass es für keine weiteren Daten verwendet werden kann. Dieser Effekt kann auftreten, wenn mehr Variablen als nötig oder zu komplizierte Ansätze, verwendet werden. Das Problem tritt während der Trainingsphase ein. Dabei nimmt der Trainingsfehler ab und der Validierungsfehler steigt an. [8] Um Overfitting zu vermeiden gibt es verschiedene Möglichkeiten [20]:

- Verwenden von Dropout-Ebenen (zufällige Gewichte werden auf 0 gesetzt)
- Training vor dem Eintreten von Overfitting stoppen
- Verschiedene Regularisierungsmethoden

## 2.4. Convolutional Neural Network (CNN)

CNNs sind eine spezielle Form von mehrschichtigen neuronalen Netzwerken [17]. Sie nutzen Lokalität aus [4]. Somit können Verschiebungen im Input automatisch kompensiert werden. Zum Beispiel wenn ein Bild verarbeitet wird, aber das Subjekt nicht immer in der Mitte ist. Sie werden daher auch oft verwendet um ein bis zwei dimensionale Inputs zu verarbeiten. Um diese Lokalitätsverschiebung zu erreichen haben CNNs drei architektonische Ideen: Local receptive fields, shared weights (oder weight replication) und zum Teil spatial oder temporal subsampling. Mit dem Prinzip der local receptive fields können Neuronen elementare visuelle Charakteristiken wie Endpunkte oder Kanten erkennen. Dies gilt auch für Ähnliches in Sprach-Spektrogrammen [11]. Ein Convolution-Layer ist wie in Abbildung 2.6 beschrieben aufgebaut.

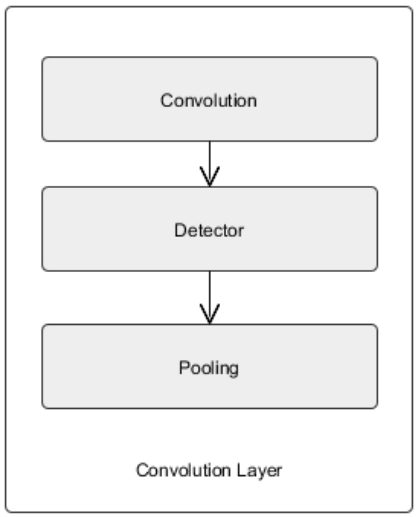



Abbildung 2.6.: Aufbau des Convolution Layers.

### 2.4.1. Convolution

Convolution wird in CNNs, im Unterschied zu anderen neuronalen Netzwerken, anstelle der normalen Matrizenmultiplikation verwenden. Ein Vorteil von Convolution ist, dass es mit unterschiedlichen Inputgrössen funktioniert. Dies wird durch sogenannte Filter ermöglicht. Diese Filter werden im Normalfall kleiner als die Inputgrösse gewählt. In einem nächsten Schritt werden die Filter über den Input geschoben und in jedem Schritt die in Abbildung 2.7 beschriebene Operation ausführt.

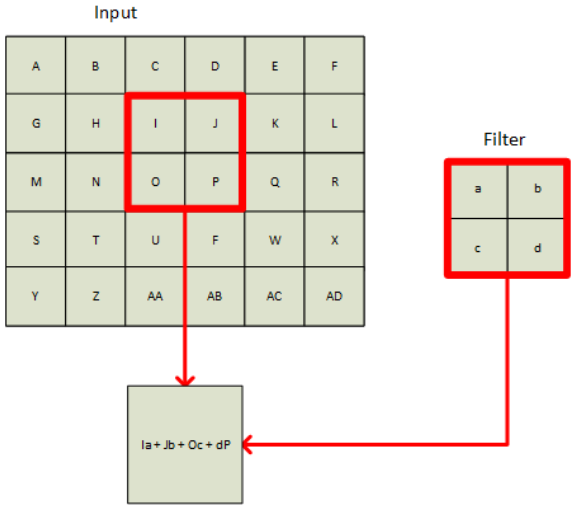



Abbildung 2.7.: Convolution Operation.

Diese Operation wird mit mehreren unterschiedlich gewichteteten Filtern durchgeführt. Filter auf demselben Layer und mit identischer Gewichtung werden zu feature maps zusammengefasst. Es kommt der sogenannte *Detector*-Schritt. In diesem wird auf jeden Convolution-Output eine lineare Aktivierungsfunktion angewandt. Folgend wird im *Pooling*-Schritt dieser Output statistisch mit seinen Nachbarn zusammengefasst. In den meisten CNNs wird in diesem Fall die *Max-Pooling*-Funktion verwenden [7].

## 2.5. Sprechererkennung

Sprechererkennung (engl. *Speaker recognition*) bezeichnet das Erkennen von Personen an ihrer Stimme. Keine zwei Menschen hören sich gleich an, denn die Form des Vokaltrakts, die Kehlkopfgrösse und weitere Organe, die für die Stimmerzeugung benötigt werden, sind stets unterschiedlich. Zusätzlich zu diesen physikalischen Unterschieden hat jeder Sprecher seine charakteristische Sprechart. Diese beinhaltet Akzent, Rhythmus, Intonationsstile, Betonungsmuster, verwendetes Vokabular und noch viele weitere Aspekte. State-of-the-Art Sprechererkennungs-Systeme verwenden eine Anzahl dieser Features um eine möglichst genauer Erkennung zu erreichen. [9]

Die Sprechererkennung lässt sich in drei Bereiche einteilen [21] [9]:

- **Speaker verification**: Binäre Entscheidung ob es sich um eine spezifische Person handelt oder nicht (1:1).
- **Speaker identification**: Gesprochene Sprache einem Sprecher aus einer endlichen Menge von Sprechern zuordnen (1:n).
- **Speaker clustering**: Gesprochene Sprache von mehreren Sprechern aufteilen. So dass sich Sammlungen von Teilen der gesprochenen Sprache bilden die jeweils zu einem Sprecher gehören. Die Anzahl Sprecher ist dabei unbekannt.

Die meisten Sprechererkennungssysteme im Bereich von Speaker verification/identification implementieren folgendes Muster[21]:

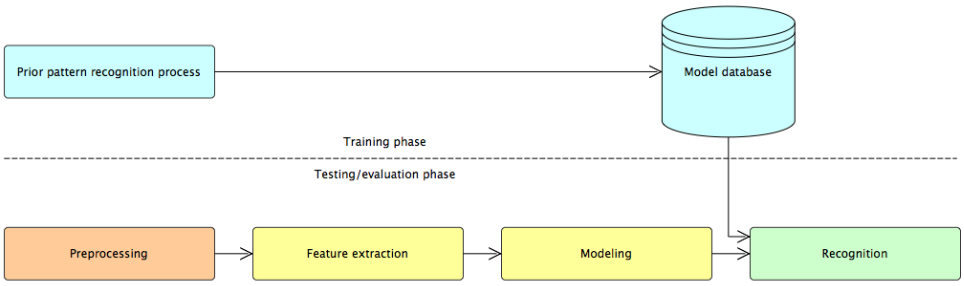



Abbildung 2.8.: Übersicht über einen Sprechererkennungsprozess[21].

- **Preprocessing**: Alles vor der Feature Extraction, also z.B. die Bereinigung des Signals.
- **Feature Extraction**: Die relevanten Informationen sollen hervorgehoben und die irrelevanten entfernt werden. Dabei geht man nach Methoden vor, die bestimmte Features extrahieren, die als relevant betrachtet werden. Beispiele für solche Methoden sind *Mel Frequency Cepstral Coefficients (MFCCs)* oder *Linear Predictive Coding (LPC)*.
- **Modeling**: Fasst die wichtigsten Informationen aus mehreren Feature-Vektoren zusammen und erstellt daraus ein Modell. Dieses Sprachmodell fungiert als Signatur für diesen Sprecher. Ein oft verwendetes Beispiel für ein solches Modell ist das *Gaussian mixture model (GMM)*.
- **Recognition**: In dieser Phase werden entweder zwei Modelle direkt miteinander verglichen oder die Wahrscheinlichkeit berechnet ob ein Feature-Vektor, einer Stimme, zum Modell einer anderen Stimme gehört.

Der Ansatz von neuronalen Netzwerken ist es, diese Schritte zu lernen. Die Convolution in CNNs ersetzt dabei den Schritt der Feature Extraction.

# 3. Vorgehen und Methoden

Dieses Kapitel beschreibt die Grundüberlegungen zur vorliegenden Arbeit. Im letzten Abschnitt werden die verwendeten Software-Bibliotheken aufgelistet.

## 3.1. Projektvorgehen

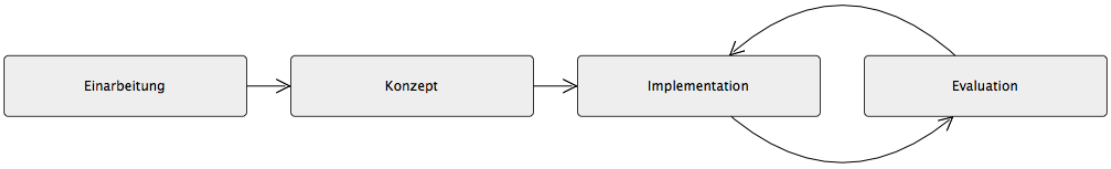



Abbildung 3.1.: Projektvorgehen.

- **Einarbeitung**: Die erste Phase beinhaltet ein erstes Auseinandersetzen mit der Thematik. Dabei werden die notwendigen Grundlagen für das Projekt geschaffen. Dies beinhaltet themenspezifisches, aber auch technologiebasiertes.
- **Konzept**: In diesem Schritt wird die Form der Eingabedaten und die Grundarchitektur bestimmt.
- **Implementation/Evaluation**: In der letzten Phase werden die Architekturen implementiert und durch Experimente evaluiert. Dabei handelt es sich um verschiedene Zyklen. Die Implementationen hängen dabei stark von den jeweiligen Experiment-Ergebnissen ab.

## 3.2. Konzept

Dieser Abschnitt erörtert die Grundüberlegungen, die für den Experimentaufbau notwendig sind.

### 3.2.1. Grundarchitektur

Als Grundarchitektur diente jene von Sander Dieleman und Benjamin Schrauwen, aus deren Arbeit *End-to-end learning for music audio* [2].

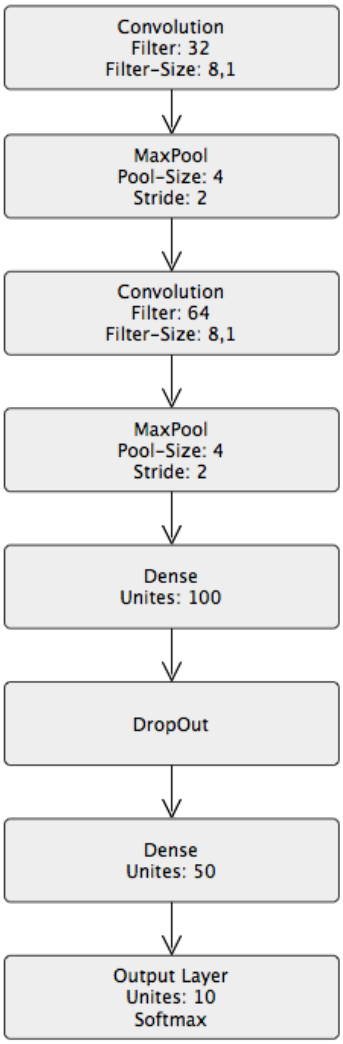



Abbildung 3.2.: Grundarchitektur des neuronalen Netzwerkes.

In der Abbildung 3.2 ist das Netzwerk von Sander Dieleman angepasst dargestellt, um 10 Sprecher zu erkennen. Die Abweichung beschränkt sich auf den Output-Layer.

### 3.2.2. Form der Eingabedaten

Für das neuronale Netzwerk musste ein einheitlicher Eingabetyp definiert werden. Dieser sollte eine Form haben, die möglichst gut verarbeitet werden kann. Für Audiodaten gibt es hierzu verschiedene Möglichkeiten:

- (Mel-)Spektrogramm
- FBE (log filter-bank energies)
- Unkomprimiertes Audio

Die Mel-Skala bezieht die wahrgenommene Frequenz eines reinen Tones zu seiner eigentlich gemessenen Frequenz. Das menschliche Gehör erkennt bei niedrigen Frequenzen kleine Änderungen in der Tonhöhe besser, als bei hohen Frequenzen. Wird die Mel-Skala auf Audio-Features angewendet, sind diese näher an dem, was Menschen hören. Um Frequenz in die Mel-Skala umzurechnen wird folgende Formel verwendet:

$$M(f) = 1125 \cdot ln(1 + \frac{f}{700}) \tag{3.1}$$

Unkomprimiertes Audio konnte von neuronalen Netzwerken bislang schlechter verarbeitet werden als Spektrogramme. Des Weiteren haben sich Spektrogramme für Erkennungsaufgaben im Audiobereich bewährt [2]. Aus diesen Gründen wurde in dieser Arbeit mit Mel-Spektrogrammen gearbeitet. Für die Generierung wurde auf die Python-Bibliothek *librosa* zurückgegriffen. Die Konfigurationen waren dabei wie folgt:

- Sampling rate: 22050
- FFT window: 1024, ergibt eine Framelänge von 45.5ms
- Samples zwischen aufeinander folgenden Frames: 441

So erhält man Mel-Spektrogramme, die für eine Sekunde die Dimension 128x50 haben. 50 Elemente in der x-Achse entsprechen somit einer Sekunde. Auf diesen Mel-Spektrogrammen wurde dann noch eine Kompression angewendet. Dies wurde durch das elementweise anwenden der Funktion $f(x) = log(1 + Cx)$ erreicht. $C$ ist dabei eine Konstante mit der die Kompression kontrolliert werden kann, sie wurde hier auf $10'000$ gesetzt [15] [2].

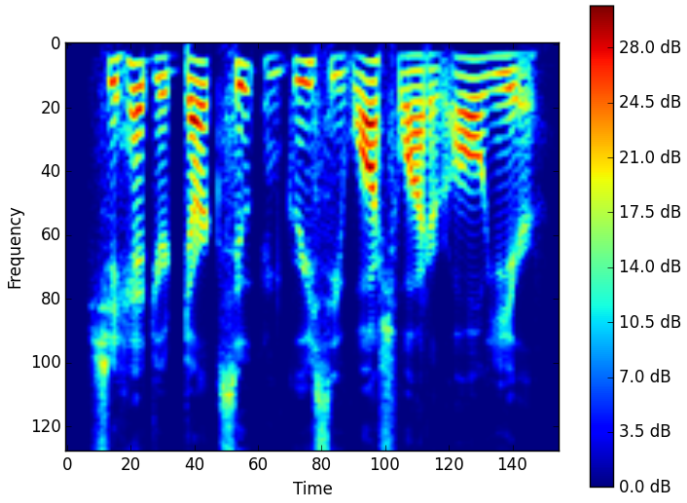



Abbildung 3.3.: Mel-Spektrogramm eines Satzes.

Spektrogramme zeigen in der vertikalen Achse die Frequenz in Hertz und in der horizontalen Achse die Zeit in Sekunden an. Die Farbe stellt die Intensität des Audiosignals in Dezibel dar. Dabei reicht die Farbpalette von Dunkelblau (Stille) bis Dunkelrot (laut).

Nach der Generierung der Spektrogramme wurden diese in einer 4D-Matrix gespeichert. Diese fungiert als Input für neuronale Netzwerke der *lasagne*-Bibliothek. Die Matrix muss zwingend folgende Struktur aufweisen:

$$X[Sprecher, Kanal, Frequenz, Zeit] \tag{3.2}$$

Da lediglich mit Graustufenbildern gearbeitet wurde, existiert lediglich ein Kanal. Die Frequenz wird durch die generierten Spektrogramme vorgegeben und ist überall konstant. Die Breite sollte eigentlich dynamisch sein, jedoch fordert *Theano*, die Bibliothek unter *lasagne*, eine Eingabematrix vom durchgehenden Typ *float*. Für ein dynamisches 2D-Array für die Sprektrogramme wären der Sprecher- und Kanalindex vom Typ *object*, dies wird aber nicht unterstützt. Somit wird die Zeit auf einen möglichst kleinen Maximalwert gesetzt. Dieser muss lediglich genug gross sein, um alle verwendeten Audiodaten aufnehmen zu können. Daraus ergibt sich die Problematik, dass bei den Daten jeweils Bildabschnitte ohne Information gebildet werden. Dieses Problem wurde mit einem Batch-Iterator gelöst, dies ist im Kapitel 3.3 Implementation genauer beschrieben.

Für die Ergebnisse existiert ein 1D-Array:

$$y[Sprecher] \tag{3.3}$$

Dabei sind die Sprecher in $y$ mit ganzen Zahlen durchnummeriert.

### 3.2.3. TIMIT-Datensatz

Für die Daten wurde auf den TIMIT-Datensatz zurückgegriffen. Dieser umfasst 630 Sprecher, mit je 10 Sätzen in Studioqualität. Von diesen sind 192 weiblich und 438 männlich. Die Sprecher sind in die acht Hauptdialekte der USA unterteilt.

## 3.3. Implementation

Im Folgenden werden die Umsetzungen der wichtigsten Aspekte unserer Experiment-Pipeline vorgestellt. Dabei handelt es sich um die Datenaufbereitung und die Trainingsphase.

### 3.3.1. Datenaufbereitung

Mithilfe des Ablaufdiagramms aus Abbildung 3.4 werden in diesem Abschnitt die Kernfunktionen der Datenaufbereitung beschrieben.

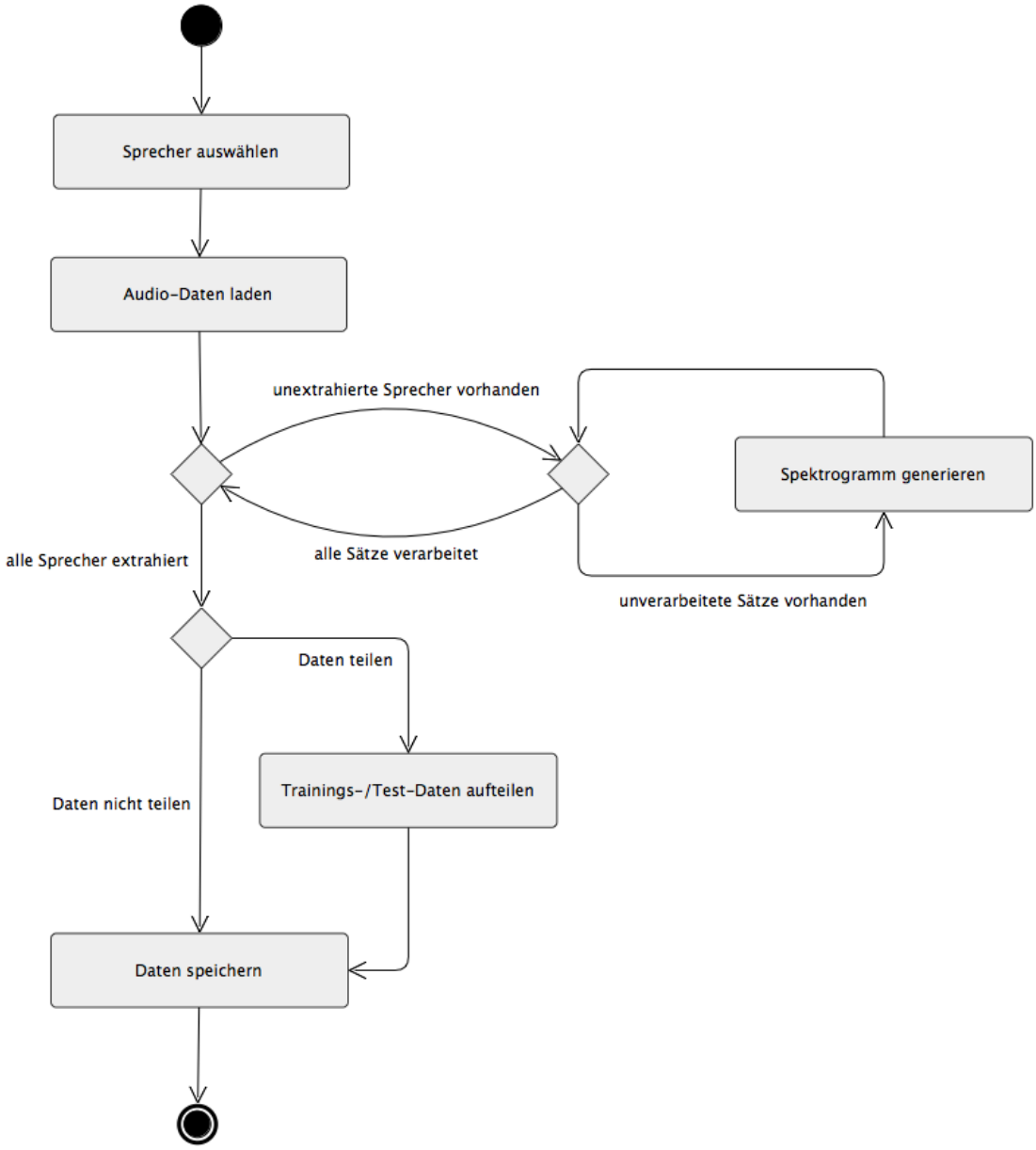



Abbildung 3.4.: Datenaufbereitung.

- **Sprecher auswählen**: Die Auswahl geschieht über Textdateien, die Listen von Sprechern enthalten. Um zufällige Listen mit Sprechern zu generieren, kann *speaker_corpus_generator.py* verwendet werden.
- **Audio-Daten laden**: Die Funktion *DataExtractor.traverse_TIMIT_data* in *data_extractor.py* traversiert die TIMIT-Daten und lädt die entsprechenden WAV-Dateien der Sprecher.
- **Spektrogramm generieren**: Für jede WAV-Datei wird ein Spektrogramm erstellt. Dies geschieht über die Funktion *mel_spectrogram* in *spectrogram_converter.py*.
- **Trainigs-/Test-Daten aufteilen**: Beim Extrahieren der Spektrogramme kann entschieden werden, ob die Daten in Trainings- und Testdaten aufgeteilt werden sollen. Dies geschieht dann durch die Klasse *SpeakerTrainSplit* in *speaker_train_splitter.py*
- **Daten speichern**: Die generierten Daten werden durch die Python-Bibliothek *cPickle* im pickle-Dateiformat gespeichert.

### 3.3.2. Trainigsphase

Dieser Abschnitt beschreibt die wichtigsten Funktionen der Trainingsphase, das Ablaufdiagramm aus Abbildung 3.5 dient dabei zur Veranschaulichung. Für das Trainieren der neuronalen Netzwerke wurde die Python-Bibliothek *lasagne* zusammen mit der darauf aufbauenden Bibliothek *nolearn.lasagne* verwendet.

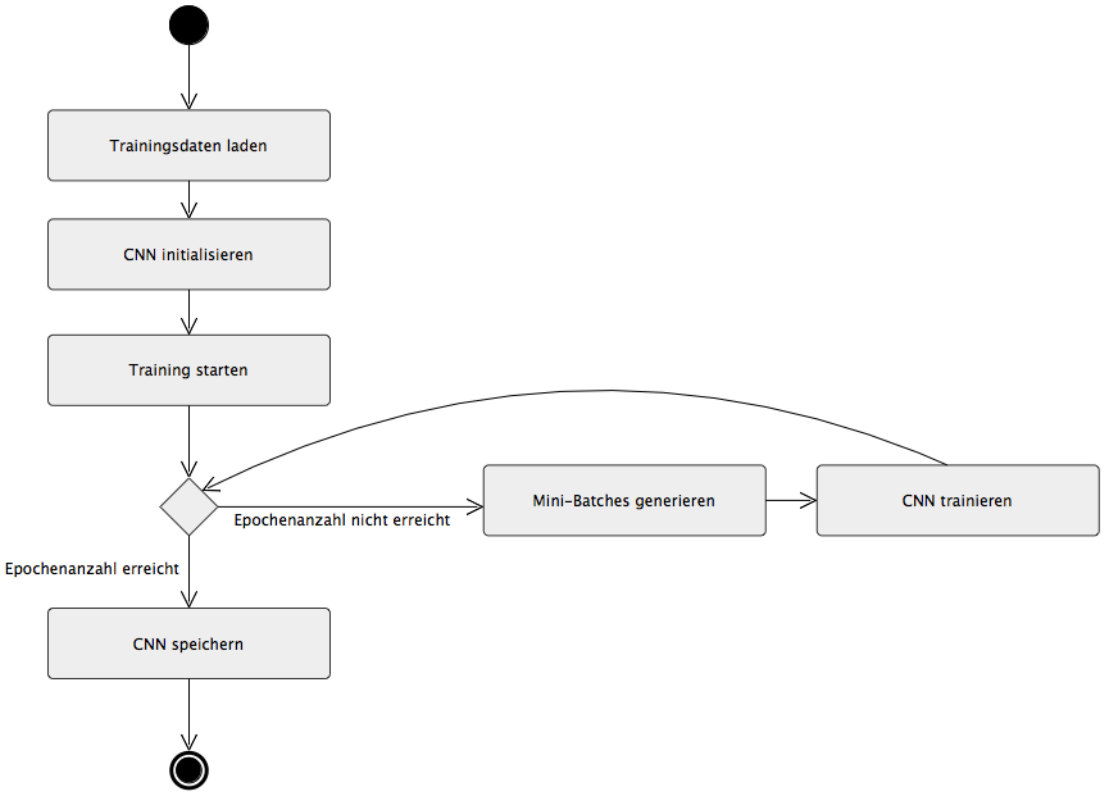



Abbildung 3.5.: Trainingsphase.

- **Trainingsdaten laden**: Die Trainingsdaten liegen im pickle-Dateiformat vor und werden mithilfe der Python-Bibliothek *cPickle* geladen. Dabei erhält man für jeden Satz ein Spektrogramm. Die Aufteilung in Sekundensegmente geschieht später.

- **CNN initialisieren**: Das CNN wird mithilfe der Klasse *NeuralNet* der Python-Bibliothek *nolearn.lasagne* erstellt.
- **Training starten**: Das Training wird über die Funktion *NeuralNet.fit* ausgelöst.
- **Mini-Batches generieren**: Es werden jeweils Sekunden-Segmente aus dem Spektrogramm extrahiert. Dies wird durch einen eigenen Batch-Iterator erzielt. Die Auswahl der Sprecher-Sätze und Segmentpositionen ergibt sich dabei zufällig. Der Batch-Iterator wird weiter unten genauer beschrieben.
- **CNN trainieren**: Es wird mittels Backpropagation trainiert, dies geschieht ebenfalls mittels *nolearn.lasagne*.
- **CNN speichern**: Das trainierte CNN wird mittels *cPickle* im pickle-Dateiformat gespeichert.

#### Batch-Iterator

Um bei der Datenauswahl eine möglichst hohe Zufälligkeit zu erreichen, wurde die Iteration der Batches angepasst. Ein Batch wird aus einer definierten Anzahl von Daten zusammengestellt. Dies erfolgt durch zwei Zufallszahlen. Die erste definiert den auszuwählenden Sprecher und dessen Satz. Mit der zweiten wird dann ein Sekundenfragment aus diesem Satz ausgewählt. Der hintere Teil des Datensatzes, der wie in Kapitel 3.2.2 bereits erwähnt, keine Daten enthält, wird dabei ignoriert.

## 3.4. Verwendete Bibliotheken

Für die Datenaufbereitung und die Implementation des CNNs waren zusätzlich zu den Standard-Bibliotheken von Python, noch weitere nötig. Diese werden im Folgenden kurz, mit der verwendeten Versionsnummer, aufgelistet.

### Datenaufbereitung

- NumPy 1.9.2
- SciPy 0.16.0
- librosa 0.4.1

### CNN

- Theano 0.7.0
- Lasagne 0.2.dev1
- nolearn.lasagne 0.6a0.dev0

# 4. Resultate Identification

Wir haben in den folgenden Experimenten die gewählte Eingabeform überprüft und die optimalen Filtergrössen in der Convolution für Speaker-Identification untersucht. Beim Aufbau nahmen wir an, dass, wie bei Musik, die besten Resultate mit einem Filter in Zeitrichtung erreicht werden [2]. In einem zweiten Teil wurde der Einfluss der Anzahl Trainingsepochen auf das Resultat untersucht.

## 4.1. Datensatz

Aus dem TIMIT-Datensatz wurden 10 weibliche Sprecher eines Dialekts oder 100 zufällige Sprecher genommen, die genauen Listen für die einzelnen Experimente sind im Anhang aufgeführt. Von den 10 Sätzen jedes Sprechers wurden 6 als Trainings- und 2 jeweils für Validierungs- und Testdaten verwendet.

## 4.2. Architektur

Als Architektur wurde als Ausgangslage die im Kapitel 3 Vorgehen und Methoden beschriebene Grundarchitektur verwendet. Anpassungen gab es in der Anzahl *Units* in den *Dense-Layers* und im *Output-Layer*, um die Anzahl Sprecher variieren zu können. Die Convolution wurde wie in den Resultaten beschrieben in der Filtergrösse angepasst. Falls nicht anders vermerkt werden immer 1000 Trainingsepochen durchgeführt. Die Accuracy wird jeweils als Mittelwert über die Validierungsdaten der letzten 100 Trainingsepochen dargestellt.

## 4.3. Experiment 1: Mel-Spektrogramm

### 4.3.1. Ausgangslage

Ziel dieses Experiments war es, die Tauglichkeit von Mel-Spektrogrammen für Speaker-Identification zu überprüfen. Des Weiteren wurde getestet ob die Grundarchitektur mit den gegebenen Daten lernt und welche Accuracy damit erreicht wird. Bei diesem Experiment wurden 10 weibliche Sprecher aus einem Dialekt verwendet. Dabei wurden zwei Konfigurationen eingesetzt.

| Nr. | Lernrate | Batch-Grösse | Epochen |
|---|---|---|---|
| 1 | $10^{-3}$ | 10 | 1000 |
| 2 | $10^{-3}$ | 10 | 10'000 |

Tabelle 4.1.: Konfigurationen von Experiment 1.

### 4.3.2. Auswertung

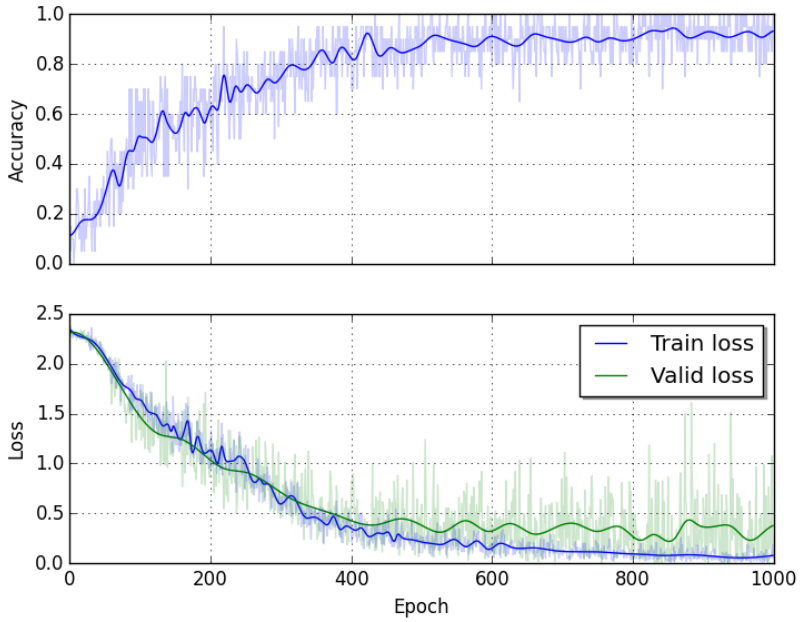



Abbildung 4.1.: Geglättete Genauigkeit und Train-/Valid-Loss bei Konfiguration 1.

In Abbildung 4.1 sieht man klar, dass die Accuracy des Netzwerks zu Beginn stark ansteigt und dann zwischen 80% und 100% abflacht. Über die Validierungsdaten erreicht das Netzwerk in den letzten 100 Epochen eine Accuracy von rund 91.85%. Auf den Testdaten beträgt die Accuracy 85%.

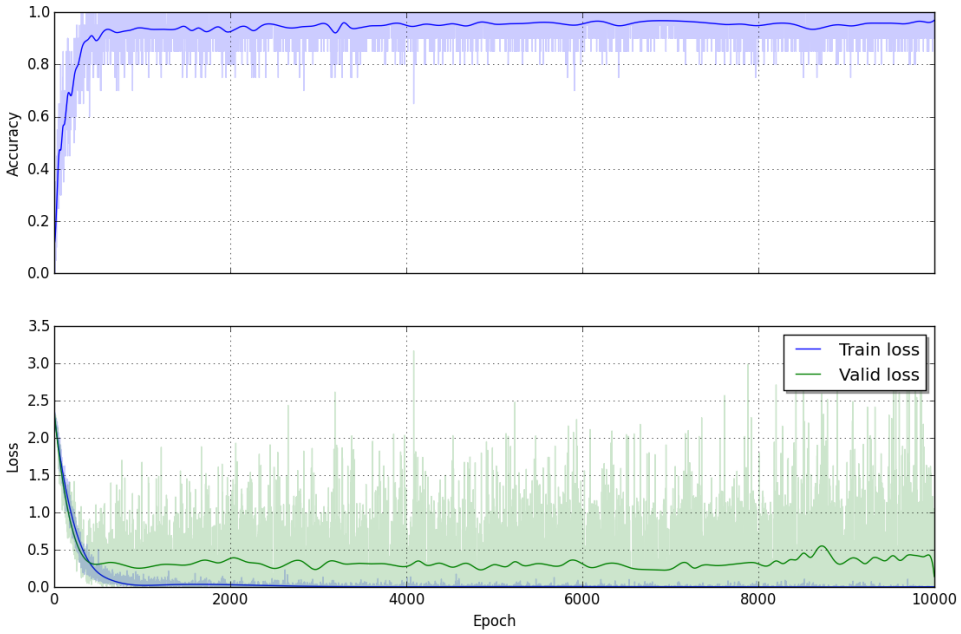



Abbildung 4.2.: Geglättete Genauigkeit und Train-/Valid-Loss bei Konfiguration 2.

Der Abbildung 4.2 ist zu entnehmen, dass auch bei 10'000 Epochen kein Einbruch der Genauigkeit festzustellen ist. Vielmehr wird sie noch geringfügig verbessert. Über die letzten 1000 Epochen erreicht das Netzwerk eine Accuracy von 95.76% auf den Validierungsdaten. Auf den Testdaten beträgt die Accuracy 91%. Mit der Verwendung von Mel-Spektrogrammen erreichten wir bereits Accuracy-Werte von über 80% auf den Testdaten. Dies zeigt, es findet eine erfolgreiche Feature Extraction durch die Convolution Filter auf den Spektrogrammen statt.

## 4.4. Experiment 2 - Convolution-Richtung

### 4.4.1. Ausgangslage

Aus Sander Dielemans Arbeit[2] übernahmen wir, dass die besten Ergebnisse mit einem Convolution Filter in Zeitrichtung erreicht werden. Der zeitliche Zusammenhang wird in diesem Fall gewichtet. Im folgenden Experiment sollte diese These überprüft werden. Drei Netzwerke mit unterschiedlichen Convolution Filtern wurden verglichen. Die Convolution erfolgte dabei in Zeit-, Frequenzrichtung und einer Kombination. Ziel dieses Experimentes war es festzustellen, welche Convolution Richtung für die weiteren Versuche verwendet werden sollte.

### 4.4.2. Auswertung

| **Filter in Zeitrichtung** | **Filter in Frequenzrichtung** | **Accuracy** |
|---|---|---|
| 8 | 1 | 0.76845 |
| 1 | 8 | 0.4745 |
| 8 | 8 | 0.6644 |

Tabelle 4.2.: Filterrichtung.

In Tabelle 4.2 ist zu erkennen, dass die Accuracy des neuronalen Netzwerks, mit einem Convolution Filter in Frequenzrichtung, das schlechteste Ergebnis liefert. Die Kombination ergibt schon klar stärkere Werte. Zu beobachten ist, dass einzig die Convolution mit Filter in Zeitrichtung eine Accuracy über 75% ergibt.

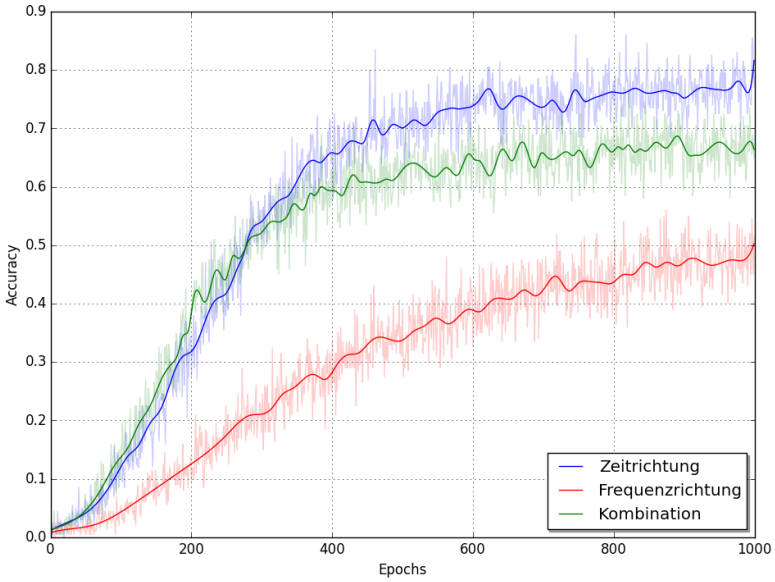



Abbildung 4.3.: Accuracies von Validierungsdaten.

In Abbildung 4.3 ist zu erkennen, dass die Kombination der Filterrichtungen schneller lernt, jedoch bereits nach 400 Trainingsepochen abzuflachen beginnt. Der Unterschied zum Training mit dem Filter in Zeitrichtung ist, bis zu diesem Zeitpunkt, nur gering. Die Zeitrichtung-Accuracy wird jedoch immer besser. Die Convolution mit Filter in Frequenzrichtung ist von Anfang an viel schwächer und kommt nie auf annähernd gleiche Werte wie die anderen Accuracies.

Für die weiteren Experimente verwendeten wir die Convolution nur in Zeitrichtung. Da dies die besten Ergebnisse ergab. Der Zusammenhang von gleichen Frequenzen über die Zeit ergibt besseren Aufschluss über den Sprecher als der Zusammenhang unterschiedlicher Frequenzen. In jedem Fall führt die Berücksichtigung eines grösseren Frequenzbereichs zu einer Verschlechterung der Accuracy.

## 4.5. Experiment 3 - Convolution in Zeitrichtung

### 4.5.1. Ausgangslage

In Experiment 2 konnte verifiziert werden, dass die besten Resultate mit Convolution in Zeitrichtung erzielt werden. In Experiment 3 versuchten wir die optimale Filtergrösse zu finden. Um Features zu extrahieren, die es ermöglichen einen Sprecher richtig zu erkennen, gingen wird davon aus, dass mindestens 130ms benötigt werden [22]. Eine Sekunde wird durch eine 50x128 (Zeit x Frequenz) grosse Matrix dargestellt. Dies bedeutet, dass wir die besten Resultate mit 6 Elementen in der Zeitachse erwarteten.

### 4.5.2. Auswertung

| Filter Zeitrichtung | Filter Frequenzrichtung | Acc. mit Dropout | Acc. ohne Dropout |
|---|---|---|---|
| 5 | 1 | 0.76205 | 0.71645 |
| 6 | 1 | 0.78410 | 0.70590 |
| 7 | 1 | 0.77535 | 0.72550 |
| 8 | 1 | 0.76845 | 0.69805 |
| 9 | 1 | 0.76900 | 0.71280 |
| 10 | 1 | 0.74750 | 0.68605 |

Tabelle 4.3.: Filtergrösse in Zeitrichtung.

In Tabelle 4.3 werden für jede Filtergrösse die Mittelwerte der Accuracy über die letzten 100 Trainingsepochen dargestellt. Jeweils ein Durchlauf mit und ohne Dropout. In den Versuchen mit Dropout fliesst jedes Mal ein Zufallsfaktor in das Ergebnis ein, dies verhindert einen genauen Vergleich der Werte. Unterschiedliche Trainingsdurchläufe können also stark variieren. Die besten Resultate erhielten wir mit einem (7,1) Filter, was 140ms in Zeitrichtung entspricht.

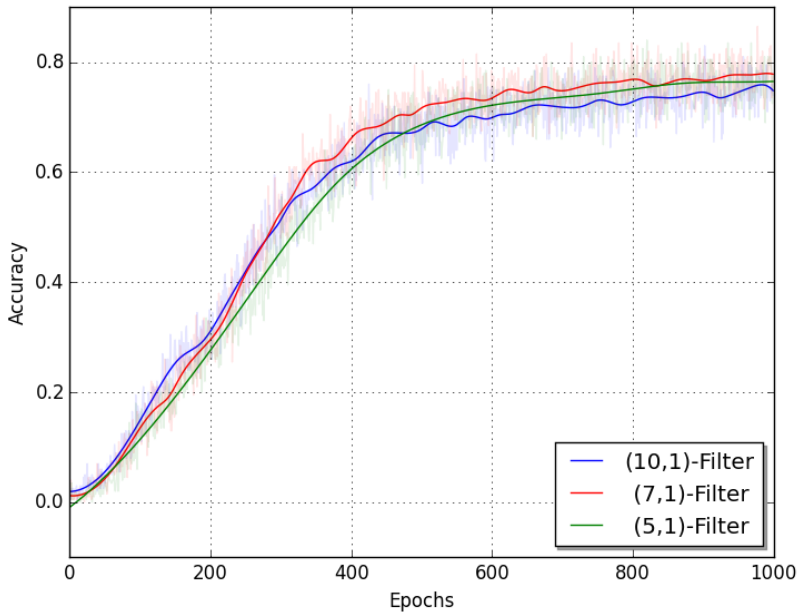



Abbildung 4.4.: Accuracies von Trainings mit Filtergrössen (10,1),(7,1) und (5,1).

In Abbildung 4.4 wird die Accuracy der Netze mit den Filtergrössen (10,1), (7,1) und (5,1) in Zeitrichtung dargestellt. Mit (10,1) und (5,1) wird der grösste und kleinste Filter aufgeführt. Der Filter (7,1) hat, wie in Tabelle 4.3 aufgelistet, die besten Werte ergeben. An den Kurven ist zu erkennen, dass die Unterschiede sehr gering sind, ohne Glättung wären sie nicht auseinander zu halten.

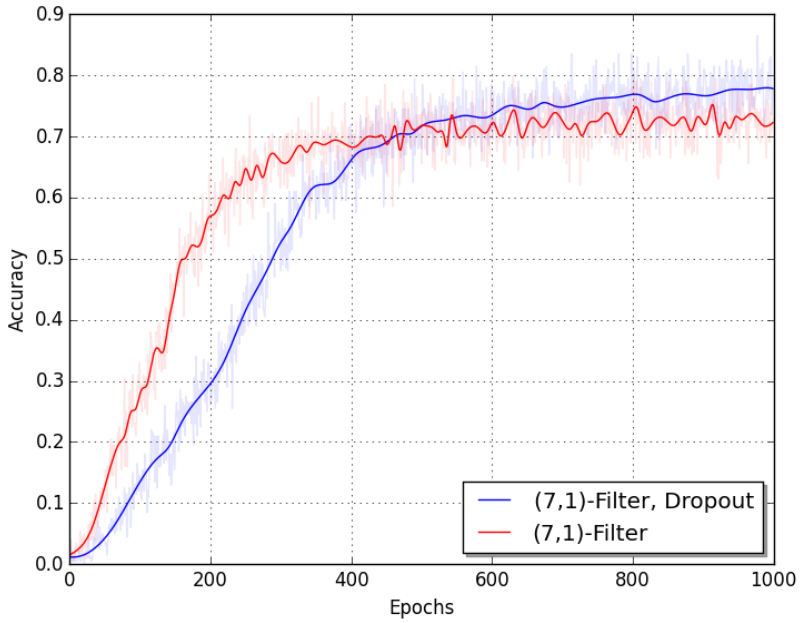



Abbildung 4.5.: Accuracies mit und ohne Dropout.

In Abbildung 4.5 ist der Accuracy-Vergleich eines Netzwerks einmal mit und einmal ohne Dropout aufgezeigt. Der Dropout-Layer führt ab 500 Trainingsepochen zu einem deutlich besseren Ergebnis. Zuvor war das Netzwerk ohne Dropout besser. Der zu Beginn schlechtere Accuracy-Wert ist damit zu erklären, dass der Dropout-Layer das Training verlangsamt.

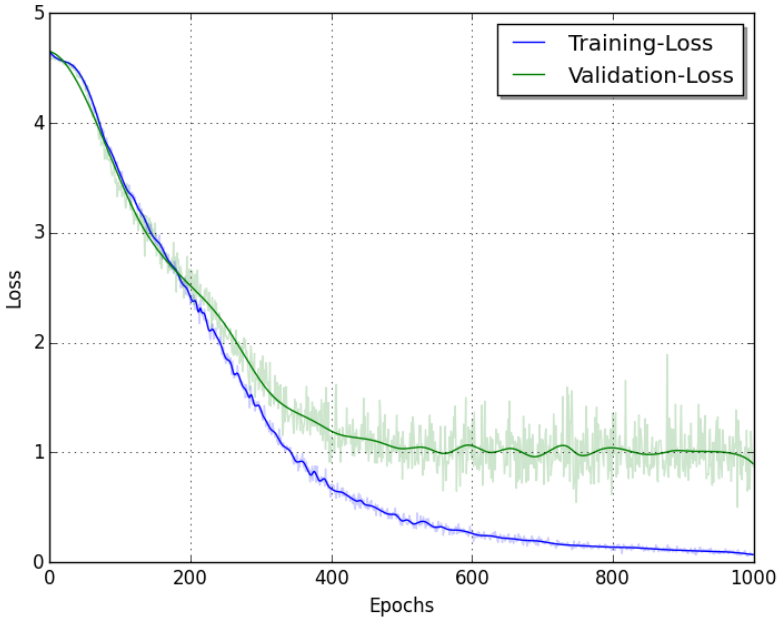



Abbildung 4.6.: Training- und Validation-Loss mit Filtergrösse (7,1).

Das es zu keinem Overfitting kommt, ist in der Abbildung 4.6 zu erkennen. Für Overfitting müsste die

Validation-Loss wieder ansteigen.

Für die beste Feature-Extraktion benötigen wir mit diesem neuronalen Netzwerk einen Convolution-Filter von 140ms. Dies bestätigt die Erkenntnis von Thilo Stadelmann, dass sprecherrelevante Informationen am besten in Zeitspannen im Bereich von 130ms zu erkennen sind [22].

## 4.6. Experiment 4 - Anzahl Epochen

### 4.6.1. Ausgangslage

Die Anzahl Epochen im Training haben einen wichtigen Einfluss auf das neuronale Netzwerk. In Experiment 4 wollten wir testen, welchen Unterschied im Resultat festzustellen ist, wenn die Epochen-Anzahl variiert wird. Dabei wurden zwei identische Netzwerke verwendet. Die Filtergrösse wurde auf (8,1) konfiguriert.

### 4.6.2. Auswertung

| Anzahl Epochen | Accuracy |
|---|---|
| 1000 | 0.69805 |
| 10000 | 0.80695 |

Tabelle 4.4.: Filterrichtung.

Mit dem zusätzlichen Training des Netzwerks konnte eine Verbesserung von ungefähr 10% erreicht werden.

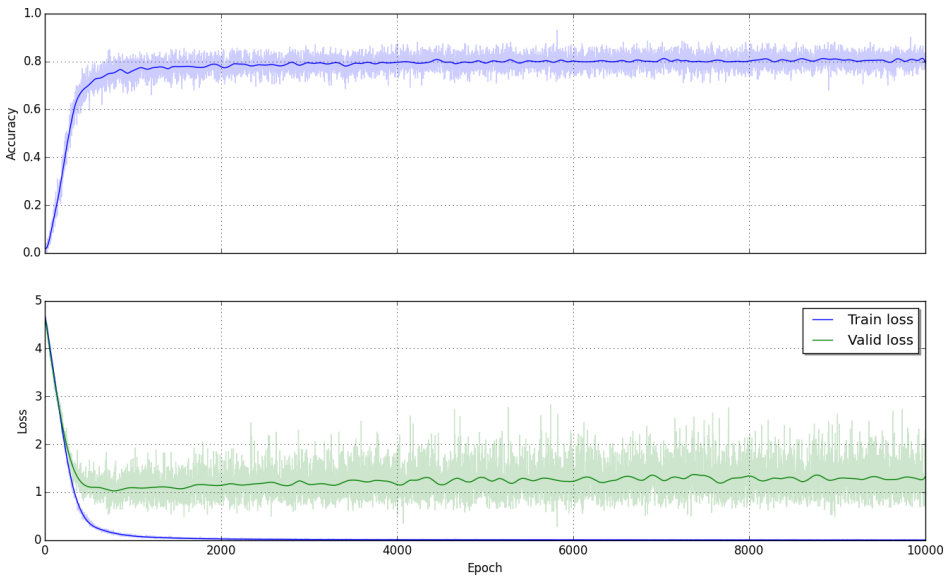



Abbildung 4.7.: Training- und Validation-Loss in einem mit 10000 Epochen trainierten Netz.

Die Accuracy nimmt ab 1000 Trainingsepochen nur noch geringfügig zu. Ebenfalls ist in Abbildung 4.7 zu erkennen, dass noch kein Overfitting entsteht. Mit einer grösseren Anzahl an Trainingsepochen kann nochmals eine Verbesserung der Erkennung erreicht werden. Für die finale Auswertung sollte daher ein stärker trainiertes Netzwerk verwendet werden.

Bis zu diesem Punkt basierte die Speaker Identification stets auf den einzelnen Sekunden-Segmenten. Andere Arbeiten zu diesem Thema verwendeten jedoch stets mehr als nur eine Sekunde [19] [14]. Unsere Testdaten beinhalten durchschnittlich 5 Sekunden. Mit der Verwendung der kompletten Testdaten sollte theoretisch eine bessere Zuordnung der Sprecher möglich sein. Hierfür extrahierten wir alle Segmente aus den Testdaten, dies geschah ohne Überlappung. Die erhaltenen Segmente wurden anschliessend nacheinander vom Netzwerk analysiert. Anhand der gebildeten Output-Vektoren konnte dann eine Identifikation ausgeführt werden. Dabei wurden verschiedene Verfahren untersucht:

- Maximaler Wert in allen Output-Vektoren.

$$\bar{x}_{max} = max_{1 \leqslant i \leqslant n} x_i \tag{4.1}$$

- Geometrischer Mittelwert über alle Output-Vektoren

$$\bar{x}_{geom} = \sqrt[n]{\prod_{i=1}^{n} x_i} \tag{4.2}$$

- Arithmetischer Mittelwert über alle Output-Vektoren

$$\bar{x}_{arithm} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{4.3}$$

Dabei ist $n$ die Anzahl Output-Vektoren und $x_i$ der Wert für den Sprecher $x$ in Output-Vektor $i$. Die Berechnungen beziehen sich jeweils auf einen Sprecher über alle Output-Vektoren. Anschliessend wurde bei den Verfahren jeweils der höchste Wert über alle erhaltenen $\bar{x}$ verwendet.

Beide Netzwerke erreichten mit den Verfahren die folgenden Accuracies:

- Maximaler Wert: 97%
- Geometrischer Mittelwert: 98%
- Arithmetischer Mittelwert: 98%

Die Berechnung mit dem maximalen Wert ergibt drei falsch klassifizierte Sprecher, beim geometrischen und arithmetischen Mittelwert waren es hingegen nur zwei. Wir stellten fest, dass es sich, bei den falsch klassifizierten Sprechern, immer um eine knappe Entscheidung handelte. Der zweitbeste Sprecher wäre jeweils der richtige gewesen und unterschied sich um lediglich $10^{-3}$ vom falsch ausgewählten Sprecher.

Mit diesem lezten Verfahren war kein Unterschied, zwischen den schwächer und stärker trainierten Netzwerken, festzustellen. Wir schliessen deshalb darauf, dass die zusätzlichen 9000 Epochen nur eine geringe Verbesserung mit sich bringen. Über alle Testdaten erreichten wir sehr gute Accuracies. Um diese mit anderen Arbeiten vergleichen zu können, muss der Versuch über die gesamten TIMIT-Daten ausgeführt werden.

## 4.7. Experiment 5 - Gesamter TIMIT-Datensatz

### 4.7.1. Ausgangslage

Um einen Vergleich mit anderen Arbeiten zu ermöglichen, führten wir in diesem Experiment die Speaker-Identification mit allen 630 Sprechern des TIMIT-Datensatzes durch. Für dieses Experiment verzichteten wir auf Validierungsdaten, so konnten wir für jeden Sprecher mit 8 Sätzen trainieren und hatten 2 Sätze als Testdaten. Die zusätzlichen Daten sollten eine klare Verbesserung für das Training geben. Es war anzunehmen, dass das Resultat tiefer ausfällt als nur bei 100 Sprechern, da das Netzwerk nun 6.3 mal mehr Sprecher lernen musste, obwohl wir das Netzwerk erneut in den Dense-Layer und im Output-Layer um die entsprechenden Werte erweiterten.

### 4.7.2. Auswertung

Das Training dauerte bedeutend länger als bei den bisherigen Netzwerken. Bislang betrug die Trainingszeit jeweils zwei bis vier Stunden. Dieses Netzwerk benötigte rund 17 Stunden für 1000 Epochen.

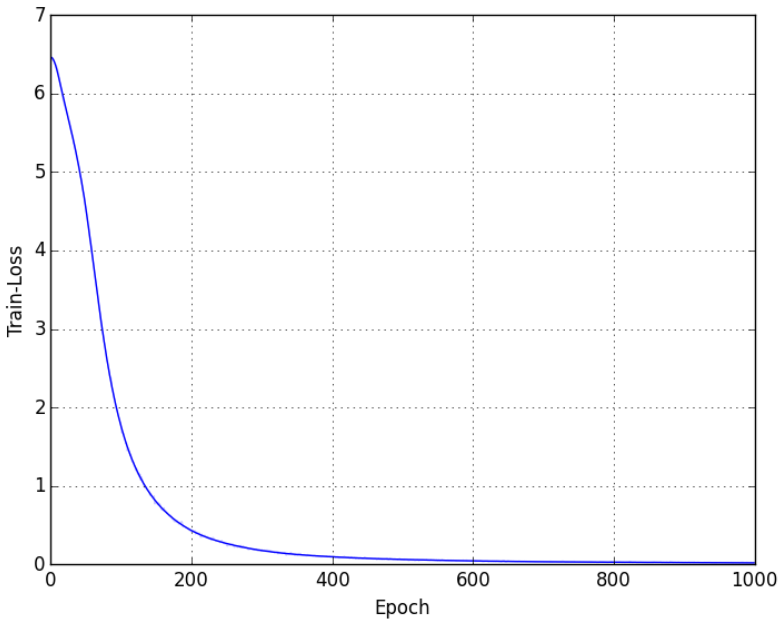



Abbildung 4.8.: Trainings-Loss des Netzwerks für 630 Sprecher.

Wie in Abbildung 4.8 ersichtlich, nahm der Trainings-Loss deutlich schneller ab, als bei den anderen Experimenten. Benötigte es bei den Netzwerken für 100 Personen noch über 400 Epochen um einen Trainings-Loss von unter 0.5 zu erreichen sind es hier nur noch 200 Epochen. Wir führten dies auf die zusätzlich verwendeten Trainingsdaten zurück, konnten dies aber nicht weiter verfolgen.

Für die Berechnung der Accuracy verwendeten wir den Ansatz aus Experiment 4, die Identifikation auf Basis der gesamten Testdaten auszuführen. Wir entschieden uns den arithmetischen Mittelwert zu verwenden, da dieser mit dem geometrischen die besten Ergebnisse ergab. Dabei erhielten wir eine Accuracy von 97%. Das bedeutet 19 Sprecher wurden falsch klassifiziert.

In diesem Experiment konnten wir zeigen, dass mit unserem Verfahren auch gute Ergebnisse erzielt werden, wenn mehr Sprecher verwendet werden. Die erreichte Accuracy von 97% ist vergleichbar mit

den knappen 100% von anderen Arbeiten auf dem TIMIT-Datensatz. Diese Arbeiten verwendeten jeweils einen Ansatz mit dem *Gaussian Mixture Model (GMM)* [19] [14].

## 4.8. Fazit

Wir konnten zeigen, dass CNNs fähig sind aus Mel-Spektrogrammen sprecherrelevante Features zu extrahieren und diese zu lernen. Die besten Ergebnisse konnten mit einer Convolution in Zeitrichtung erreicht werden.

Auf den Validierungsdaten erreichte eine Filtergrösse von 140ms die besten Accuracies. Thilo Stadelmann nannte in seiner Arbeit einen Bereich um 130ms als möglicherweise optimale Zeitspanne um sprecherspezifische Informationen zu extrahieren [22]. Wir konnten diese Annahme mit unserem Ergebnis bestätigen.

Der Dropout-Layer verbesserte bei jedem getesteten Netzwerk die Accuracy. Jedoch trat auch bei den Netzwerken ohne einen Dropout-Layer kein Overfitting auf. Wir nehmen deshalb an, dass die höhere Komplexität der Netzwerke mit Dropout-Layer die Verbesserung verursacht. Ein Beweis wird jedoch noch benötigt.

Wird die Accuracy über alle Testdaten berechnet, resultiert dies in signifikant besseren Ergebnissen. Dabei erwiesen sich der arithmetische und der geometrische Mittelwert über allen Output-Vektoren als optimal.

Bei der Speaker-Identification über den gesamten TIMIT-Datensatz erreichen wir eine Accuracy von 97% bei durchschnittlich fünf Sekunden Testdaten. Dies kommt sehr nahe an die State-of-the-Art Verfahren, die auf denselben Daten knapp 100% erreichen. Im Kapitel 6 Ausblick werden noch einige Verbesserungsmöglichkeiten aufgezeigt, mit denen möglicherweise noch bessere Werte erreicht werden können.

# 5. Resultate Clustering

Wir haben versucht mit vortrainierten Netzwerken, aus den Speaker Identification Experimenten, ein Teilproblem des Sprecher-Clusterings zu lösen. Dabei wollten wir ermitteln ob ein Clustering möglich ist und ob die Dropout-Layer des Netzwerks einen Einfluss auf die Clustering-Qualität hat.
Wir gingen davon aus, dass wenn dem neuronalen Netzwerk unbekannte Sprecher übergeben werden, daraus Vektoren resultieren, die keinem Sprecher eindeutig zuweisbar sind. Diese Vektoren sollten jedoch für jeden Sprecher, einem möglichst eindeutigen Muster entsprechen. So erhält man für jedes eingegebene Segment einen mehrdimensionalen Vektor. Diese Vektoren sollte man dann mithilfe einer adäquaten Distanzformel, wie der Cosinus-Distanz[23], clustern können. Optimalerweise folgen die Vektoren für jeden Sprecher einem Muster und bilden somit sprecherweise Cluster.

## 5.1. Datensatz

Es wurde mit drei verschiedenen Teilmengen der TIMIT-Daten gearbeitet. Diese haben jeweils 5, 20 und 40 Sprecher. Die 5 Sprecher sind dabei in den 20 Sprechern enthalten und diese wiederum in den 40. Dieselben Sprecher verwendeten Thilo Stadelmann und Bernd Freisleben in ihrer Arbeit *Unfolding Speaker Clustering Potential: A Biomimetic Approach*[22].

## 5.2. Architektur

Als Architektur wurde die Konfiguration mit einem Filter von 8 Einheiten in Zeitrichtung und 1 Einheit in Frequenzrichtung verwendet. Um zu verhindern, dass keine Schnittmenge der Sprecher im trainierten Netzwerk und dem Cluster-Datensatz vorhanden ist, wurden nochmals zwei neue Netzwerke mit 100 Sprechern für 1000 Epochen trainiert. Eines mit Dropout-Layer und eines ohne.

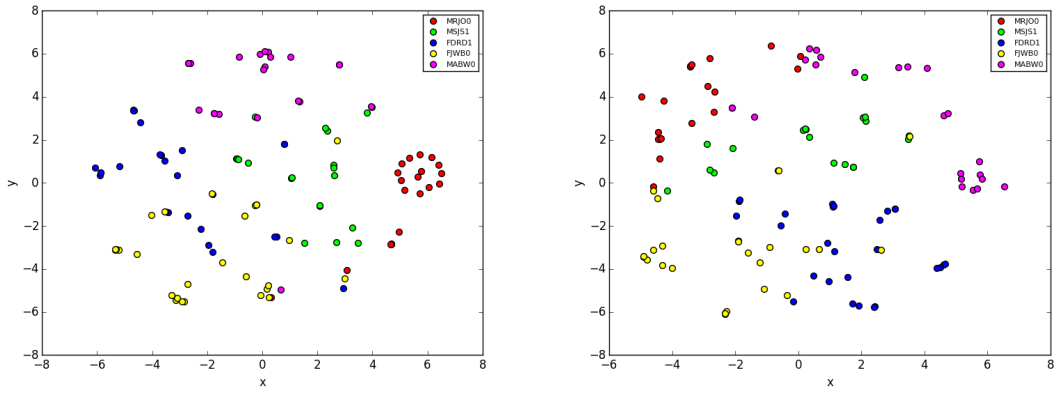
## 5.3. Experiment 1 - Segment-Clustering

### 5.3.1. Ausgangslage

In diesem Experiment wurde untersucht ob ein Clustering überhaupt möglich ist. Dabei haben wir die TIMIT-Teilmenge mit fünf Sprechern verwendet. Für die Visualisierungen der Cluster wurde das t-SNE Verfahren genutzt[24].

### 5.3.2. Auswertung

Die Resultate zeigen, dass sich bei beiden Netzwerken Cluster bilden. Diese erstrecken sich über abgegrenzte Flächen, die sich teilweise überlappen. Ebenfalls kann man beobachten, dass sich die zwei weiblichen Sprecherinnen von den männlichen abgrenzen. Das Netzwerk verwendet also Features, die durchaus relevant für die Unterscheidung zwischen Mann und Frau sind.




(a) Ohne Dropout. (b) Mit Dropout.

Abbildung 5.1.: t-SNE Plot der Sprechersegmente von fünf Sprechern.

Auf Basis der t-SNE Plots kann nicht klar gesagt werden, ob der Dropout-Layer einen negativen oder positiven Einfluss auf das Ergebnis hat. Des Weiteren ist auch die Qualität der Cluster nur schwer zu bestimmen. Um dem entegegenzuwirken verwendeten wir hierarchisches Clustering. Dabei geht es darum, über alle Segmente zu iterieren und bei jeder Iteration diejenigen zusammenzufassen, die sich am nächsten sind. Die Darstellung erfolgt in einem Dendrogramm.

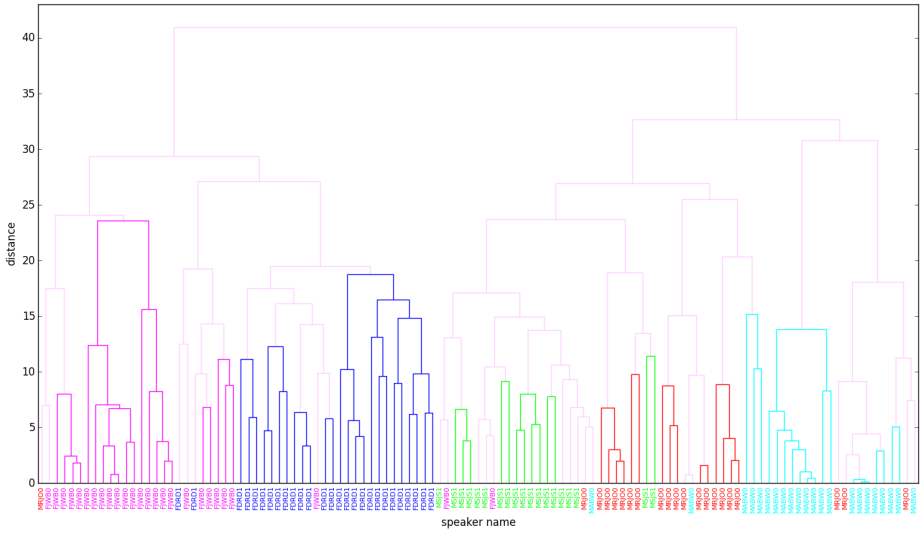



Abbildung 5.2.: Dendrogramm bei fünf Sprechern, Netzwerk mit Dropout.

Das Dendrogramm ist von unten nach oben zu lesen. Wir haben jedem Sprecher eine eindeutige Farbe zugeordnet. Werden zwei Segmente richtig verbunden, ist die Verbindung im Dendrogramm in der Farbe des Sprechers gehalten. Wenn Segmente falsch zugeordnet werden, ist die Verbindung in einem hellen Pink eingefärbt.
Die Segmente werden relativ gut geclustert. Erneut sieht man, dass sich die weiblichen Sprecherinnen klar abgrenzen und mehrheitlich nur untereinander überschneiden. Es wurde die *Complete*-Linkage mit der *Manhatten*-Metrik verwendet. Mit der *Cosinus*-Metrik konnten ebenfalls gute Ergebnisse erzielt werden. Da die Dendrogramme der beiden Netzwerke ähnlich ausfielen, wurde an dieser Stelle auf jenes ohne Dropout verzichtet.
Mit diesem Experiment konnten wir zeigen, dass mit dem verwendeten Ansatz Clustering möglich ist.

## 5.4. Experiment 2 - Clustering Qualität

### 5.4.1. Ausgangslage

Unser Ansatz macht Clustering möglich. Jedoch konnten wir noch keine klare Aussage über die Qualität der Cluster machen. Dieses Experiment sollte in einem Vergleich mit anderen Clusteringansätzen [22], das Potential unseres Ansatzes aufzeigen. Dabei wurde das Clustering mit den TIMIT-Teilmengen von 20 und 40 Sprechern durchgeführt.
Für jeden Sprecher wurden zwei Segmente gebildet. Das eine besteht aus allen Sekunden-Segmenten von acht Sätzen und das andere aus jenen der restlichen zwei Sätze. Dies geschah durch das Addieren und anschliessende Normalisieren der Output-Vektoren des neuronalen Netzwerks. So erhält man für jeden Sprecher zwei Segmente. Perfektes Clustering wäre, wenn nun alle zwei Segmente immer zusammenfinden bevor ein Segment falsch zugeordnet würde.

### 5.4.2. Auswertung

Als Messgrösse für die Qualität des Clusterings verwendeten wir die Misclassification Rate (MR) [10]. Diese ist wie folgt definiert:

$$MR = \frac{1}{N} \sum_{j=1}^{N_s} e_j \tag{5.1}$$

Dabei beschreibt $N$ die Anzahl Audio-Segmente, $Ns$ die Anzahl Sprecher und $e_j$ jene Segmente von Sprecher j, die nicht richtig zugewiesen sind. Die Misclassification Rate nimmt jeweils einen Wert zwischen 0 und 1 an. Ein kleiner Wert steht für eine kleine Wahrscheinlichkeit falsch zugewiesener Segmente.

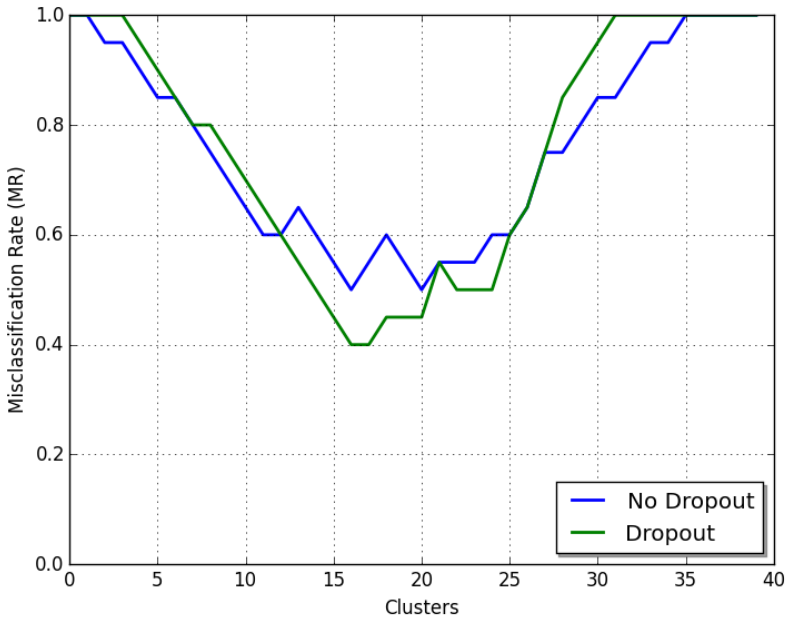



Abbildung 5.3.: Misclassification Rate bei 20 Sprechern mit und ohne Dropout.

Wie aus Abbildung 5.3 ersichtlich wurde auf dem Dropout-Netzwerk eine Misclassification Rate von 40% erreicht. Ohne die Anwendung eines Dropout-Layers fällt diese mit 50% leicht schlechter aus.

Um die Segmente zusammenzufassen, verwendeten wir die *Complete*-Linkage in Verbindung mit der *Cosinus*-Metrik. Durch die Verwendung der *Cityblock*-Metrik erhält man vergleichbare Ergebnisse, andere Metriken wie die *Euclidean*-Metrik ergaben über alle durchgeführten Versuche massiv schlechtere Misclassification Rates.
Aus der Misclassification Rate ist ersichtlich nach wie vielen Iterationen das Clustering optimal gestoppt werden sollte. Bei zwei Segmenten pro Sprecher ist die Anzahl Iterationen auch die vom System geschätzte Anzahl Sprecher im System. Für das Netzwerk mit Dropout ist der optimale Stopp also nach 16 Iterationen und bei jenem ohne Dropout bei 16 oder 20 Iterationen.

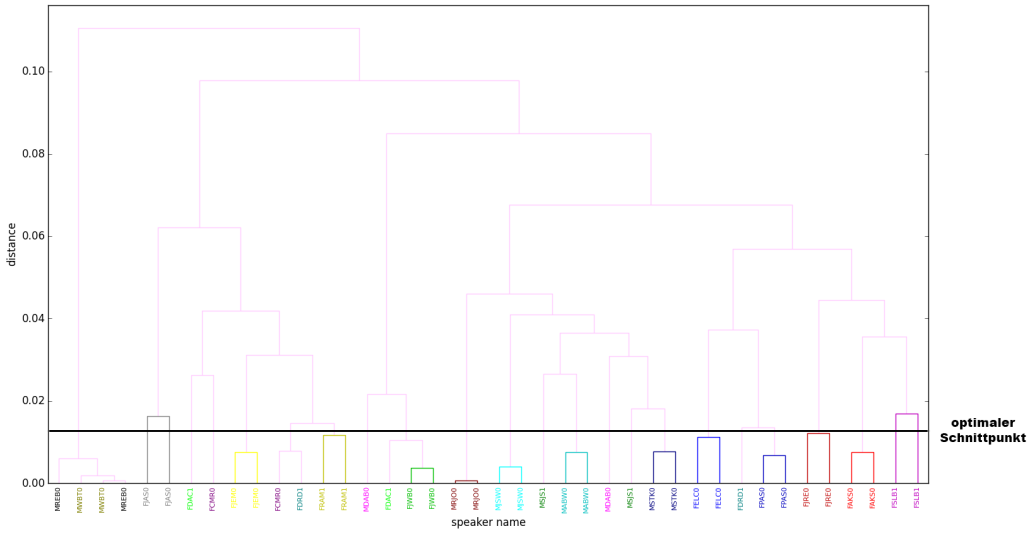



Abbildung 5.4.: Dendrogramm bei 20 Sprechern ohne Dropout.

Dem Dendrogramm, für das Netzwerk ohne Dropout, aus Abbildung 5.4 ist zu entnehmen, dass 13 von 20 Segment-Paaren korrekt gebildet werden. Verwendet man jedoch den optimalen Schnittpunkt der Misclassification Rate, werden zwei Segmente zu spät zusammengefasst. Somit ist das best erreichbare Ergebnis 11 Segment-Paare zu bilden.

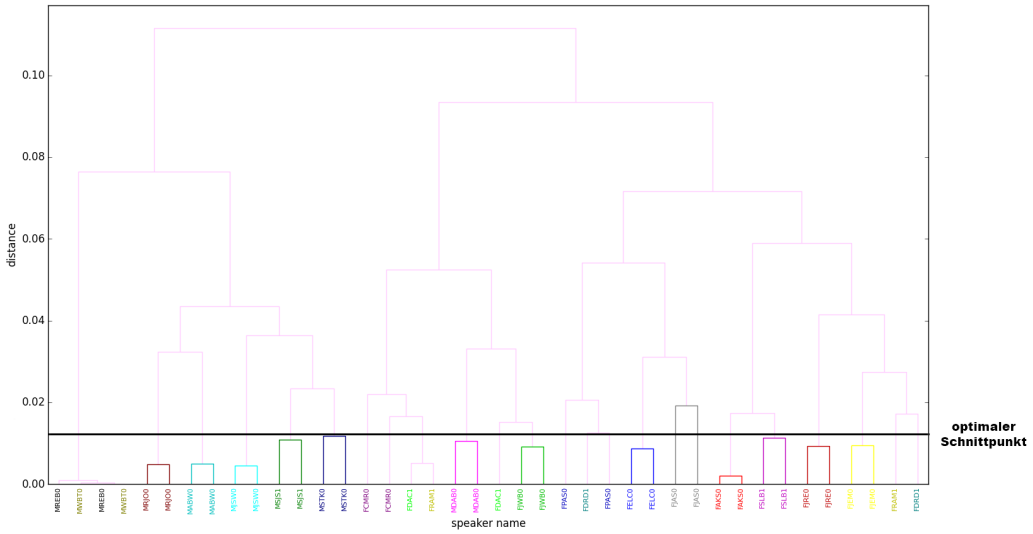



Abbildung 5.5.: Dendrogramm bei 20 Sprechern mit Dropout.

Beim Dendrogramm für das Netzwerk mit Dropout ist zu beobachten, dass zwar gleich viele Segment-Paare wie bei jenem ohne Dropout gebildet werden können, jedoch kann der optimale Schnittpunkt besser gesetzt werden. Man verliert hier nur ein Segment-Paar und kann somit 12 Segment-Paare richtig bilden. Der Dropout-Layer hat hier also einen positiven Einfluss auf die Clustering-Qualität.

Dieselben Versuche haben wir auch mit der Sprecher-Teilmenge von 40 Sprechern ausgeführt. Dort wurde auf dem Dropout-Netzwerk eine Misclassification Rate von 65% erreicht, jedoch wurde die *Complete*-Linkage mit der *Cityblock*-Metrik verwendet. Mit der Cosinus-Metrik erreichte man eine Misclassification Rate von 70%. Auch bei den 40 Sprechern fiel die Misclassification Rate auf dem Netzwerk ohne Dropout klar schlechter aus.

Zum Vergleich: Wir erreichten beim Versuch mit 20 Sprechern eine Misclassification Rate von 40% und mit 40 Sprechern, eine von 65%. Thilo Stadelmann und Bernd Freisleben erreichten in ihrer Arbeit für die gleichen 20 Sprecher eine Misclassification Rate von 0% und für die 40 Sprecher eine von 12.5% [22].

In diesem Experiment konnten wir zeigen, dass das Clustering mit neuronalen Netzwerken durchaus möglich ist. Die Qualität des Clusterings im gegebenen Experiment ist jedoch deutlich schlechter als jene in anderen Arbeiten.

## 5.5. Fazit

Der verwendete Ansatz macht Clustering von Audio-Daten möglich. Vergleicht man die Qualität des Resultats mit anderen Arbeiten, erhält man deutlich schlechtere Ergebnisse. Jedoch verfügen die verwendeten neuronalen Netzwerke lediglich über einen Ausgabevektor der Grösse 100. Wir nehmen an, dass ein grösserer Vektor für mehr Diversität unter den Sprechern führen könnte und so auf den verwendeten Datensätzen deutlich bessere Ergebnisse liefern würde.

# 6. Ausblick

In diesem Kapitel wird beschrieben, wie auf die Resultate dieser Arbeit aufgebaut werden kann.

## 6.1. Speaker Identification

Im Bereich der Sprecher-Identifikation konnten wir gute Ergebnisse erreichen. Jedoch bestehen noch in einigen Bereichen Verbesserungsmöglichkeiten und Unklarheiten.

### 6.1.1. Spektrogramm-Generierung

Wir verwendeten die Standard-Sampling Rate von 22'050 der *librosa*-Bibliothek. Die TIMIT-Daten liegen jedoch in einer Sampling Rate von 16'000 vor. Es könnte bei der Generierung der Spektrogramme die Sampling Rate der Audio-Dateien verwendet werden. Des Weiteren könnten die Samples zwischen aufeinander folgenden Frames verdoppelt werden. So würde man eine doppelt so hohe Auflösung in Zeitrichtung erhalten. Es wäre denkbar, dass das neuronale Netzwerk so noch besser geeignete Features extrahieren könnte.

### 6.1.2. Notwendige Testdatenlänge

Mit durchschnittlich fünf Sekunden Testdaten erreichen wir eine Genauigkeit von 97%. Es wäre interessant zu evaluieren, wie sich die Genauigkeit verhält wenn man die Testdaten verkürzt.

### 6.1.3. Komplexität des neuronalen Netzwerks

Für alle Experimente wurde die Grundarchitektur des Netzwerks aus *End-to-end learning for music audio* [2] verwendet. Die grösste Variation bestand im Entfernen des Dropout-Layers für einige Netzwerke. Diese schnitten jedoch immer schlechter ab, als jene mit Dropout-Layer. Da sie jedoch ebenfalls kein Overfitting aufwiesen, liegt die Vermutung nahe, dass die zusätzliche Komplexität des Dropout-Layers die Verbesserung zur Folge hat. Diese Vermutung könnte durch erhöhen der Komplexität des Netzwerks untersucht werden.

## 6.2. Speaker Clustering

Es konnte gezeigt werden, dass Speaker Clustering möglich ist. Jedoch sollte die Qualität noch gesteigert werden. Im Folgenden werden einige mögliche nächste Schritte beschrieben.

### 6.2.1. Genauere Segment-Analyse

Im t-SNE Plot ist ersichtlich welche Audio-Segmente ähnliche Output-Vektoren haben, also eine geringe Distanz aufweisen. Dabei handelt es sich oft um homogene Cluster von Segmenten desselben Sprechers. Es kommt aber vor, dass sich inhomogene Cluster mit Segmenten von vielen verschiedenen Sprechern bilden. Wir nehmen an, dass es sich dabei um nur schwer zuzuordnende Segmente handelt. Also Segmente die entweder keine Sprache enthalten oder Sprache die keine charakteristischen Merkmale aufweist, die aber oft in der gesprochenen Sprache vorkommen.

### 6.2.2. Neuronales Netzwerk erweitern

Wir verwendeten in unseren Experimenten ein Netzwerk, das auf 100 Sprecher trainiert war. Der TIMIT-Datensatz würde es jedoch erlauben, die verwendeten 40 Sprecher auf einem Netzwerk das auf 590 Sprecher trainiert ist, zu clustern. Dabei hätte man den Vorteil, dass der Ausgabevektor 5.9 mal grösser wäre. Dies sollte für mehr Diversität unter den erstellten Sprecherprofilen führen, was auch ein besseres Clustering zur Folge haben könnte.

### 6.2.3. Clustering auf Basis der Convolution Features

In unserem Ansatz versuchen wir das Clustering anhand der Output-Vektoren des neuronalen Netzwerkes zu erreichen. Dabei wird jedoch ignoriert, dass das Netzwerk durch die Dense-Layers versucht den Input-Vektor einem klaren Sprecher zuzuordnen, obwohl es diesen gar nicht kennt. Nach der Convolution hat man jedoch sprecherspezifische Features. Dort ist es auch irrelevant ob diese einem bekannten oder unbekannten Sprecher gehören. Es wäre also interessant, das Clustering anhand der extrahierten Features zu versuchen. In der Bildverarbeitung wurden bereits ähnliche Ansätze verwendet um stilistische Charakteristiken aus Bildern zu extrahieren [6].

# 7. Verzeichnisse

# Literaturverzeichnis


[1] Li Deng and Dong Yu. Deep learning: methods and applications. *Foundations and Trends in Signal Processing*, 7(3–4):197–387, 2014.

[2] Sander Dieleman and Benjamin Schrauwen. End-to-end learning for music audio. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6964–6968. IEEE, 2014.

[3] Gabriel Eyyi. Deep learning für automatische stimmerkennung. 2015.

[4] Alex Fandrianto, Ashley Jin, and Aman Neelappa. Speaker recognition using deep belief networks. 2012.

[5] K. F. Wender G. D. Rey. *Neuronale Netze. Eine Einführung in die Grundlagen, Anwendungen und Datenauswertung*. Hans Huber, 2010.

[6] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[7] Ian Goodfellow, Aaron Courville, and Yoshua Bengio. Deep learning. Book in preparation for MIT Press, 2015. URL http://goodfeli.github.io/dlbook/.

[8] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[9] Tomi Kinnunen and Haizhou Li. An overview of text-independent speaker recognition: From features to supervectors. *Speech communication*, 52(1):12–40, 2010.

[10] Margarita Kotti, Vassiliki Moschou, and Constantine Kotropoulos. Speaker segmentation and clustering. *Signal processing*, 88(5):1091–1124, 2008.

[11] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10), 1995.

[12] Charles E Metz. Basic principles of roc analysis. In *Seminars in nuclear medicine*, volume 8, pages 283–298. Elsevier, 1978.

[13] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048, 2010.

[14] Andrew Morris, Dalei Wu, and Jacques Koreman. Gmm based clustering and speaker separability in the timit speech database. *IEICE Transactions on Fundamentals of Communications, Electronics, Informatics and Systems*, 85, 2004.

[15] Mathias Muller, Daniel PW Ellis, Anssi Klapuri, and Guilhem Richard. Signal processing for music analysis. *Selected Topics in Signal Processing, IEEE Journal of*, 5(6):1088–1110, 2011.

[16] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[17] Dũng Việt Phạm. Online handwriting recognition using multi convolution neural networks. In *Simulated Evolution and Learning*, pages 310–319. Springer, 2012.

[18] Kevin L Priddy and Paul E Keller. *Artificial neural networks: an introduction*, volume 68. SPIE Press, 2005.

[19] Douglas A Reynolds. Speaker identification and verification using gaussian mixture speaker models. *Speech communication*, 17(1):91–108, 1995.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[21] Thilo Stadelmann. *Voice Modeling Methods for Automatic Speaker Recognition*. PhD thesis, Philipps-Universität Marburg, 2010.

[22] Thilo Stadelmann and Bernd Freisleben. Unfolding speaker clustering potential: a biomimetic approach. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 185–194. ACM, 2009.

[23] Alexander Strehl, Joydeep Ghosh, and Raymond Mooney. Impact of similarity measures on web-page clustering. In *Workshop on Artificial Intelligence for Web Search (AAAI 2000)*, pages 58–64, 2000.

[24] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[25] Matthew D Zeiler, Marc'Aurelio Ranzato, Rajat Monga, Min Mao, Kun Yang, Quoc Viet Le, Patrick Nguyen, Alan Senior, Vincent Vanhoucke, Jeffrey Dean, et al. On rectified linear units for speech processing. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3517–3521. IEEE, 2013.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# A. Anhang

## A.1. Projektmanagement

### A.1.1. Offizielle Aufgabenstellung

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

**Sprechererkennung mit Convolutional Neural Networks**
**PA15_stdm_3**

| | |
|---|---|
| BetreuerInnen: | Thilo Stadelmann, stdm<br>Oliver Dürr, dueo |
| Fachgebiete: | Datenanalyse (DA)<br>Software (SOW) |
| Studiengang: | IT / WI |
| Zuordnung: | Institut für angewandte Informationstechnologie (InIT) |
| Interne Partner: | Institut für Datenanalyse und Prozessdesign (IDP) |
| Gruppengrösse: | 2 |

**Kurzbeschreibung:**
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 Neural Networks (CNNs) bieten sich hier an, die in den letzten Jahren zu einem Milliardengeschäft und heissesten Eisen in der Machine Learning Forschung geworden sind.

In einer früheren Bachelorarbeit wurde am InIT bereits eine experimentelle Umgebung für Stimmerkennung mit CNNs implementiert, in der bekannte Datensätze geladen, analysiert und Ergebnisse ausgewertet werden können. Ziel dieser Projektarbeit ist es nun, die Stimmerkennung ("Speaker Identification") mittels CNNs weiterzuentwickeln und experimentell zu evaluieren.

**Inhalt:**

- Kennenlernen des Hintergrunds automatischer Sprechererkennung
- Einarbeiten in die relevante Literatur zu CNNs
- 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
- Präsentation der Ergebnisse (Bericht, Demo)

**Voraussetzungen:**
Die Betreuer haben langjährige Erfahrung mit automatischer Sprechererkennung und stehen mit Rat, Tat und viel Einsatz zu Seite. Aus den Vorgängeraufgabenstellungen sind bereits mehrere hervorragende studentische Arbeiten hervorgegangen. Bei Interesse ist eine Fortsetzung in BA und Masterstudium möglich.

Vorkenntnisse in Sprechererkennung, Maschine Learning oder Convolutional Neural Networks etc. sind nicht Voraussetzung. Das nötige Anwenderwissen wird im Rahmen dieser PA vermittelt und erarbeitet. Wir erwarten lediglich Freude am Experimentieren und Programmieren.

**Die Arbeit ist reserviert für:**
Yanick Lukic (lukicyan)
Carlo Vogt (vogtcar1)

**Weiterführende Informationen:**
https://dublin.zhaw.ch/~stdm/?page_id=77

Dienstag 14. Juli 2015 17:17

### A.1.2. Projektplan

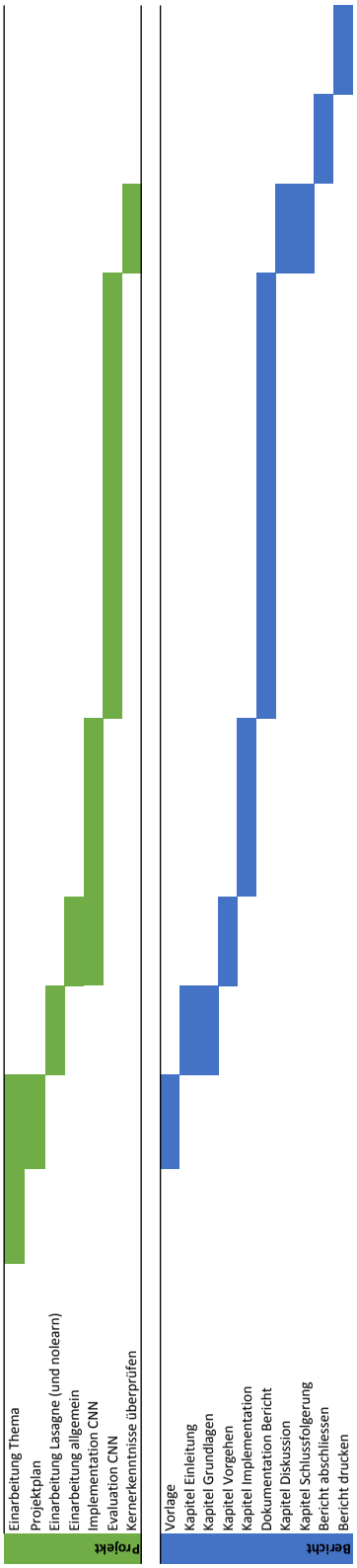



Abbildung A.1.: Projektplan.

## A.2. Weiteres

### A.2.1. Beschreibung der elektronischen Daten

Dieser Arbeit liegen einige elektronische Daten bei. Im Folgenden wird deren Struktur kurz erklärt.

- **0_Dokumentation**: Enthält die Dokumentation im PDF-Format.
- **1_Speaker_Identification**: Enthält die trainierten Netzwerke, Trainingslogs und eine Zusammenfassung zu den durchgeführten Experimenten zum Thema Speaker Identification.
- **2_Speaker_Clustering**: Enthält die zwei vortrainierten Netzwerke, die für das Speaker Clustering verwendet wurden. Des Weiteren sind noch einige Diagramme von nicht näher beschriebenen Experimenten vorhanden.
- **3_Python_Code**: Dieses Verzeichnis beinhaltet den gesamten Code, der verwendet wurde. Aus speichertechnischen Gründen mussten wir auf das Beilegen von aufbereiteten Trainingsdaten verzichten.

| | | 06112015001 | 06112015002 | 06112015003 | 06112015004 | 06112015005 | 11112015001 | 12112015001 | 12112015002 | 12112015003 | 13112015001 | 13112015002 | 13112015003 | 16112015001 | 17112015001 | 25112015001 | 30112015001 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Conv2DLayer** | **num_filters** | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 |
| | **filter_size** | 8.1 | 8.1 | 1.8 | 8.8 | 10.1 | 7.1 | 5.1 | 5.1 | 6.1 | 6.1 | 7.1 | 9.1 | 9.1 | 10.1 | 8.1 | 8.1 |
| **MaxPool2DLayer** | **pool_size** | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| | **stride** | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| **Conv2DLayer** | **num_filters** | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |
| | **filter_size** | 8.1 | 8.1 | 1.8 | 8.8 | 10.1 | 7.1 | 5.1 | 5.1 | 6.1 | 6.1 | 7.1 | 9.1 | 9.1 | 10.1 | 8.1 | 8.1 |
| **MaxPool2DLayer** | **pool_size** | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| | **stride** | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| **DenseLayer** | **num_units** | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 6300 |
| **DropoutLayer** | | inactiv | activ | activ | activ | activ | activ | inactiv | activ | inactiv | activ | inactiv | inactiv | activ | inactiv | inactiv | activ |
| **DenseLayer** | **num_units** | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 3150 |
| **epochs** | | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 10000 | 1000 |
| **last 100 Validation** | | 0.69805 | 0.76845 | 0.4745 | 0.6644 | 0.7475 | 0.77535 | 0.71645 | 0.76205 | 0.7059 | 0.7841 | 0.7255 | 0.7128 | 0.769 | 0.68605 | 0.80695 | - |
| **on random 100'000 Test data** | | 0.699 | 0.772 | 0.488 | 0.65 | 0.725 | 0.767 | 0.695 | 0.695 | 0.701 | 0.772 | 0.695 | 0.702 | 0.76 | 0.681 | 0.798 | 0.765 |
| **on all test data** | | 0.92 | 0.98 | | | | 0.97 | | | | 0.94 | | | | | 0.98 | 0.97 |

Abbildung A.2.: Resultate der Speaker Identification Experimente.

### A.2.3. Sprecherlisten

**Speaker Identification - 10 Sprecher**

| | | | |
|---|---|---|---|
| FCJF0 | FECD0 | FKFB0 | FSJK1 |
| FDAW0 | FETB0 | FMEM0 | |
| FDML0 | FJSP0 | FSAH0 | |

**Speaker Identification - 100 Sprecher**

| | | | |
|---|---|---|---|
| MSES0 | FJEM0 | FKDW0 | MCEW0 |
| FJSK0 | FGRW0 | MZMB0 | MBNS0 |
| FSCN0 | FMMH0 | FMBG0 | MTLB0 |
| MCHL0 | MREB0 | MLJC0 | FJCS0 |
| MJRP0 | MRAB0 | FJSP0 | FKDE0 |
| FDAW0 | MGXP0 | MKRG0 | MGLB0 |
| MLEL0 | MMWB0 | MRAI0 | MCDD0 |
| MJLG1 | FHEW0 | MRPP0 | FMEM0 |
| MLIH0 | MJAC0 | MMWS0 | FCRZ0 |
| MRGS0 | MGMM0 | MKLS0 | FSBK0 |
| FCMH0 | MJMD0 | FSJG0 | MDVC0 |
| MKLT0 | FDTD0 | MRJS0 | FMJU0 |
| FMCM0 | FMML0 | MGWT0 | MRML0 |
| MGAR0 | FLAS0 | MGSH0 | MJSR0 |
| MJFC0 | FMLD0 | MMWH0 | MCTH0 |
| FKLC1 | MMWS1 | FPKT0 | FREH0 |
| MTQC0 | MMGG0 | MBBR0 | MHXL0 |
| MWDK0 | FJRP1 | MMJB1 | FETB0 |
| FDML0 | FPAF0 | MSVS0 | MJWT0 |
| MAKB0 | FSGF0 | MMDS0 | FGDP0 |
| MCCS0 | FAPB0 | FBMH0 | MDMA0 |
| MWAC0 | MLSH0 | FCAL1 | MRJT0 |
| FJRE0 | FBAS0 | MJLB0 | |
| FCLT0 | MRCS0 | MPFU0 | |
| MGRP0 | MAJP0 | FJHK0 | |
| MBWM0 | MRMB0 | MJXL0 | |

**Speaker Clustering - 40 Sprecher**

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