Identifikation gesprochener Sprache mit Deep Learning

Joel André

Maturaarbeit
Betreut von Beni Keller

Kantonsschule Zug

2019

Inhaltsverzeichnis

1 Einleitung

Sprachoberflächen sind der aktuelle Megatrend in der Tech-Branche. Ein Algorithmus, der natürliche Sprache zuverlässig versteht, ist längst kein Sciene-Fiction mehr. Hight-Tech-Firmen rund um den Globus investieren Milliarden in die Entwicklung solcher Produkte. Wie das iPhone mit dem Touchscreen eine Revolution der Interaktion Mensch-Maschine lancierte, erhofft man sich mit Spracherkennung den nächsten Durchbruch. Sprachgesteuerte Programme sollen weitaus intuitiver zu bedienen sein als Texteingaben. In Zukunft ist die lästige Uhrzeitabfrage-Bewegung zum Handy wahrscheinlich Geschichte. Wir werden bequem danach fragen können.

Moderne Spracherkennung-Systeme sind auf einzelne Sprachen spezialisiert. Aus diesem Grund muss ein sprachunabhängiges System, in erster Linie die verwendete Sprache identifizieren, um anschliessend, das passende Sprachsystem anzuwenden. Dieser Schritt nimmt zum Beispiel die Grammatik der erkannten Sprache zur Hilfe. In dieser Arbeit geht es darum den Prozess der Sprachidentifikation zu implementieren: Aufnahmen mündlicher Äusserungen sollen der korrekten Sprache zugeordnet werden.
Die Idee zu diesem Thema entstand spontan. Für mich war aber seit Anfang an klar, dass ich meine Maturaarbeit im Fach Informatik beschreiten würde. Ich nahm bereits an der Informatikolympiade und dem Freifach *Begabtenförderung Informatik* teil. Mich weiter in diesen Bereich zu vertiefen, war eine logische Konsequenz. Überdies besitzen Informatik-Projekte den Vorteil äusserst ressourcenarm zu sein. Ein Computer mit Internet-Anschluss reicht, um ein innovatives Produkt zu entwickeln, dass die Welt verändern kann. Das nötige Wissen lässt sich ohne lange Ausbildung im Internet finden. Entsprechend erlaubt Informatik Jugendlichen ,bereits Teil der Wirtschaft zu sein. Kein 14-Jähriger kann, ohne teures Labor Medikamente entwickeln, hingegen für die Idee *Blockchain* hätte theoretisch jeder die Mittel gehabt.

Für die Sprachidentifikation beschränke ich mich auf die Methode, *Künstliche Intelligenz*. KI ist ein prominentes junges Teilgebiet der Informatik. In den letzten Jahren wurden enorme Fortschritte gemacht, was dazu führt, dass die Wirtschaft der Forschung weit hinterher hinkt. Dank fortgeschrittener Computer-Hardware und neuen Algorithmen können Maschinen heute Aufgaben lösen, die vor zehn Jahren unmöglich erschienen. Mittlerweile sind zum Beispiel sogar selbst-fahrende Autos nur noch eine Frage der Zeit. Der Traum der Menschen, eine ihr in Intelligenz ebenbürtige Maschine zu entwerfen, wird fast realistisch.
Die klassischen Projekte in KI-Arbeiten sind oft Spiele. Es wird dem Computer beigebracht besser zu spielen als jeder Mensch. So ist es zum Beispiel beim Schachcomputer *Deep blue*[8] oder dem Go-Programm *AlphaGo*[1]. Solche Projekte bieten zwar gute Forschungsobjekte, sind aber an sich keine Anwendungsfelder. Ich wollte in meiner Arbeit etwas programmieren, das tatsächlich einen realen Nutzen besitzt. Sprachidentifikation erfüllt das Kriterium. Sprachidentfikation kann nebst für die Spracherkennung, unter anderem im Anrufcenter angewendet werden. Eine Computer erkennt die Sprache des Kunden und leitet den Anruf an den passenden Mitarbeiter weiter.

Klassische Methoden für die Sprachidentifikation beruhen stark auf Expertenwis-

sen. Analytisch entwickelte, handprogrammierte Verfahren erreichen mit wenig maschinellem Lernen sehr gute Leistungen. In dieser Arbeit wird ein anderer Ansatz gewählt. Der Computer soll möglichst viel selbst erlernen. Dafür beschränke ich mich auf die Technologie *Deep Learning*.
Das Ziel dieser Arbeit lässt sich in zwei Teile gliedern. Zuerst soll KI bzw. Deep Learning beschrieben werden und soweit wie möglich ein Verständnis dafür geschaffen werden. Anschliessend wird die Technologie auf das Problem Sprachidentifikation angewendet. Als zu identifizierende Sprachen gelten Französisch, Englisch und Deutsch. Es werden verschiedene Ansätze probiert und verglichen. Das Produkt soll eine möglichst grosse Fehlerfreiheit besitzen. Um das Produkt zu demonstrieren, wird ein Web-Interface entwickelt, das einem ermöglicht, das Produkt selber zu testen.

Um die Realisierbarkeit dieses Vorhabens zu beurteilen, hatte ich im Vorfeld der Arbeit, bereits das Buch *Neuronale Netze selbst programmieren*[27] aus dem Info-Z gelesen. Dies ermöglichte mir erst, die Fragestellung genauer einzuschränken und den nötigen Aufwand dafür abzuschätzen. Die enthaltenen Informationen waren aber noch lange nicht ausreichend um mit dem Programmieren zu beginnen. Ausschlaggebend war erst später das Buch *Deep Learning with Python*[6], das als theoretische Grundlage dient.
Parallel wurde begonnen das Web-Interface zu entwickeln. Das nötige Know-How besass ich bereits grösstenteils als Vorwissen. Das früh realisierte Interface erlaubte, das Produkt in der Entwicklungsphase live zu testen.
Der nächste Schritt war endlich das Programmieren am Produkt. In dieser Phase gab das Paper *Practical Applications of Multimedia Retrieval. Language Identification in Audio Files*[32] die Richtung vor. Um die Unterschiede zwischen den Sprachen zu erlernen, benötigt mein Programm viele Trainings-Daten. Die Beschaffung dieser grossen Menge Daten und das effiziente Umgehen damit, sind ein nicht zu unterschätzender Aufwand.
Die letzte Phase, das Entwickeln der künstlichen Intelligenz, war am spannendsten. Die Komplexität des Themas erlaubte allerdings oft nur oberflächlich, ein Verständnis zu entwickeln, zumal die mathematischen Grundlagen auf Universitäts- Stufe fehlten.

Im Kapitel **2 Deep Learning** wird versucht, das Verständnis für die verwendeten Technologien der künstlichen Intelligenz weiterzugeben. Das Kapitel bildet das Herz des theoretischen Teils. Das nächste Kapitel **3 Daten** befasst sich mit den Quellen der Daten, den Methoden zur Datenbeschaffung und den Algorithmen zur Datenbearbeitung. Anschliessend werden in Kapitel **4 Modelle** die entwickelten Modelle präsentiert. Kapitel **5 Umsetzung und Resultate** erläutert die Implementierung des System's und zeigt die Auswertung der Modelle. In **6 Diskussion** werden die Ergebnisse in Zusammenhang gestellt und Kapitel **7 Anhang** erlaubt Interessierten einen Einblick in einen Teil des Codes. Der Vollständige Programmiercode der Arbeit ist online öffentlich hier verfügbar: `https://github.com/jotron/deepLID`

2 Deep learning

Die Begriffe *Deep Learning*, *maschinelles Lernen* und *künstliche Intelligenz* werden oft fälschlicherweise auswechselbar verwendet. Es gibt allerdings eine ganz klare, und für das Verständnis wichtige Hierarchie zwischen den Wörtern. Um Klarheit zu verschaffen werden darum alle Gebiete aufgeführt.

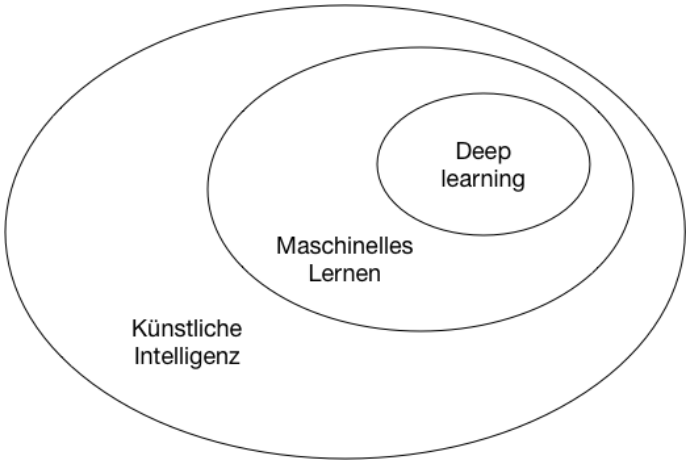


Abbildung 1: Künstliche Intelligenz, Maschinelles Lernen und Deep Learning

Das Gebiet der künstlichen Intelligenz gibt es schon so lange wie den Computer selbst. Die Frage, wie schlau ein Computer werden kann, beschäftigt uns bis heute. Als anerkannte Definition für KI gilt, das Bestreben intellektuelle Aufgaben, die normalerweise von Menschen gelöst werden, zu automatisieren [6, Kap. 1.1.1].

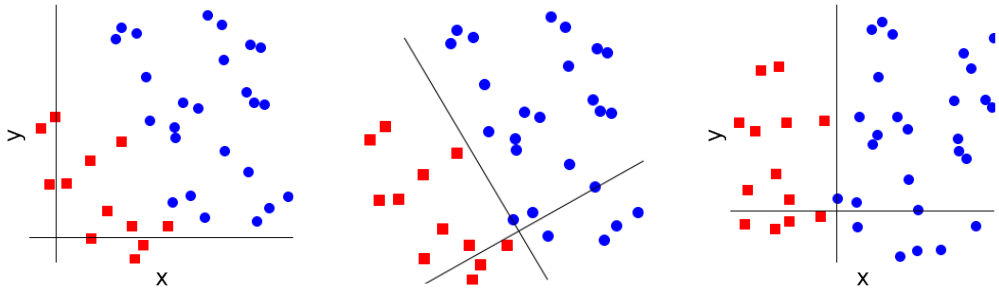
Erste Erfolge erreichte man zum Beispiel mit Schachcomputern, die handgeschriebene Regeln befolgten. Diese Form von künstlicher Intelligenz hat aber schnell Grenzen, da die Lösungen vieler Prozesse schlicht zu komplex sind, um sie unter angemessenem Aufwand mit Regeln zu beschreiben. Mit der Zeit verbreitete sich darum eine neue Methode: Maschinelles Lernen. [vgl. 6, Kap. 1.1.1]

Der Ablauf von maschinellem Lernen ist grundlegend anders als konventionelles Programmieren. Der Entwickler muss keinen Programmcode mit festen Regeln schreiben, im Gegenteil: Er liefert dem Computer Trainigsdaten, und der Computer lernt die Regeln selbst. Die bekannteste Kategorie von Machine Learning Algorithmen ist *Überwachtes Lernen*. Bei überwachtem Lernen bestehen die Trainingsdaten aus Eingaben x und zugehörigen Ausgaben y. Der Algorighmus versucht die Abbildungsfunktion f zwischen den Eingaben und den Ausgaben zu bestimmen.

$$y = f(x)$$

Der Name *Überwachtes Lernen* kommt daher, dass man sich den Lernalgorithmus wie einen Lehrer vorstellen kann der die Abbildungsfunktion überwacht. Die korrekten Ausgaben zu den Eingaben sind bekannt. Die Abbildungsfunktion berechnet iterativ Ausgaben zu den Trainingsdaten und der Lernalgorithmus korrigiert ihn mit den Lösungen. Der Lernprozess hört auf sobald die Abbildungsfunktion soviel wie möglich gelernt hat. Kurzgesagt lernt das System aus Beispielen Muster zu erkennen. Dieser Prozess wird *Training* genannt. Maschinelles Lernen blühte erst in den 90'er Jahren auf, wurde aber schnell zum grössten Teilgebiet der künstlichen Intelligenz. [vgl. 6, Kap. 1.1.2] [vgl. 3]

Beim maschinellem Lernen, lernt die Software im Grunde eine nützlichere Darstellungsweise der Daten bzw. der Eingabe. Anhand dieser anderen Repräsentation kann der Computer die Antwort einfach erkennen. Abbildung 2 zeigt als Beispiel zwei unterschiedliche Repräsentationen von Daten. Um zwischen roten und blauen Elementen zu unterscheiden ist Repräsentation (c) eindeutig praktischer als (a). Um bei (c) zwischen roten und blauen Elementen zu unterscheiden gilt einfach: Falls $x > 0$ ist das Element blau und sonst ist es rot. Bei Deep Learning gehen die Daten durch eine Folge von aufeinander aufbauenden Repräsentationen. Jede Repräsentation ist ein wenig nützlicher um das Problem zu lösen als die Vorherige. Auf diese Weise kann der Computer zunehmend komplexe Probleme, in einfacheren Zwischenschritten lösen. Das *Deep* in Deep Learning entspringt der grossen Anzahl an aufeinanderfolgenden Repräsentationen. Die Verbindung zwischen Deep Learning und überwachtem Lernen ist, dass Deep Learning eine Kategorie von Abbildungsfunktionen beschreibt. Deep Learning bezeichnet das Konzept von stufenweisem Lernen und nicht eine einzelne Methode. Eine weit verbreitete Methode sind allerdings *tiefe künstliche neuronale Netze*. [vgl. 6]



(a) Rohe Daten (b) Koordinatenwechsel (c) Bessere Repräsentation

Abbildung 2: Unterschiedliche Repräsentationen

2.1 Künstliche neuronale Netze

Künstliche neuronale Netze hat man sich, wie der Name schon preisgibt, von der Natur abgesehen. Ähnlich wie in unserem Gehirn gibt es Neuronen bzw. Knoten und dazwischenliegende Verbindungen. Die "Intelligenz" entsteht erst aus dem Zusammenspiel tausender Neuronen. Künstliche Neurone Netze haben sich mittlerweile so stark weiterentwickelt, dass sie nebst der ursprünglichen Idee, nichts mehr mit der biologischen Variante gemeinsam haben.

Der Wert eines Knotens/Neurons ist abhängig von der Summe aller seiner eingehenden Verbindungen. Was der Wert einer Verbindung ist wird weiter unter erklärt. Die Abhängigkeit von der Summe wird durch die Aktivierungsfunktion σ definiert. Die Aktivierungssfunktion ist wichtig, damit das Netzwerk jede mögliche Funktion abbilden kann. Es ist bewiesen dass neuronalen Netze universelle Approximatoren sind [22, Kap. 4]. Eine bekannte Aktivierungsfunktion ist zum Beispiel die *ReLU* Funktion. Sie wird vor allem geschätzt für ihre Simplizität. Die ReLU Funktion unterdrückt negative Werte, bzw. sie ist null für Werte kleiner als null. [vgl. 27] [vgl. 6]

$$\sigma(x) = \text{ReLU}(x) = \max(0, x)$$

Der Wert einer eingehenden Verbindung entspricht dem Produkt des Herkunftsknoten, bzw. des Knoten woher die Verbindung kommt, und dem Gewicht $w \in \mathbb{R}$ der Verbindung. Die Gewichte der Verbindungen sind die Parameter des Netzwerks. Intuitiv kann man sich vorstellen, dass das Netzwerk lernt, wichtigeren Verbindungen grössere Gewichte zuzuordnen und Unwichtigen umgekehrt. Wenn man alles zusammensetzt ergibt sich für den Wert irgendeinen Knotens o_i mit Vorgänger-Knoten $\boldsymbol{h}$ diese Formel:

$$o_i = \sigma\Big(\sum_i h_i \cdot w_i\Big)$$

Die fettgedruckte Notation bedeutet, dass ein Vektor gemeint ist. Normal geschriebene Buchstaben mit Index bezeichnen die Komponenten des Vektors. Mit dieser Formel propagieren sich die Werte der Anfangsknoten durch das ganze Netzwerk. Um das Prinzip anschaulicher zu machen, wird ein Beispiel-Durchlauf vorgeführt. Die verwendeten Gewichte und Eingaben sind willkürlich.

Die Aufgabe des vorgestellten Netzwerks ist es zum Beispiel, zwischen Hund und Katze zu unterscheiden. Die Eingaben x_1 und x_2 sind gewisse Merkmale des zu erkennenden Tieres. 1 bedeutet, dass Tier besitzt das Attribut und 0 das Gegenteil. Die Ausgabe o beschreibt die Wahrscheinlichkeit, dass das Tier eine Katze ist. Daraus folgt, dass Werte $o > 0.5$ für Katze stehen und Werte $o < 0.5$ für Hunde.

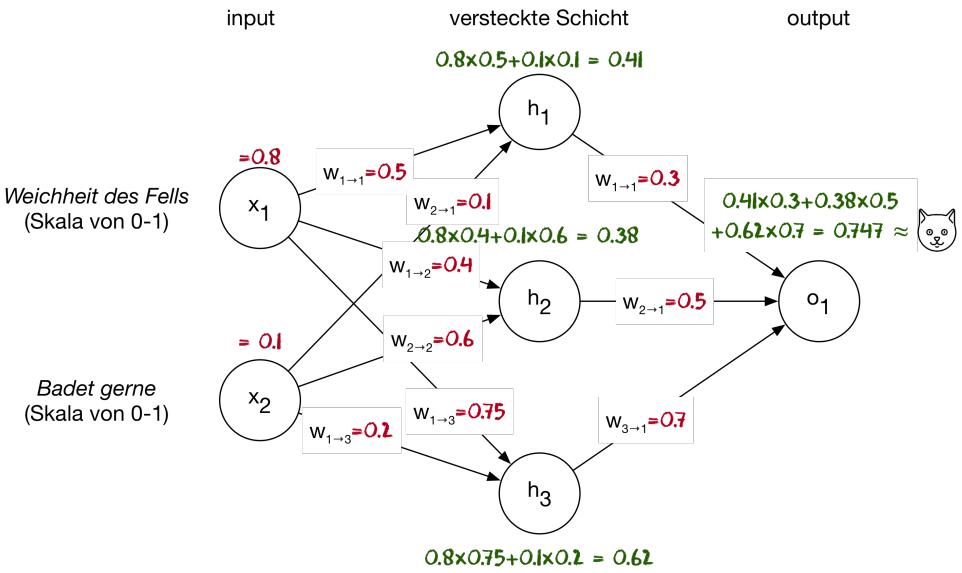


Abbildung 3: Grafische Darstellung eines künstlichen neuronalen Netzwerks anhand eines Beispiels: Rote Zahlen sind festgesetzt und grüne Zahlen werden berechnet.

Die versteckte Schicht ist wichtig damit das Netzwerk eine grössere Anzahl Paramter aus den Daten lernen kann. Je mehr versteckte Schichten es gibt, desto schwierigere Probleme kann das Netzwerk in der Regel lösen. Die Werte jeder versteckten Schichten können als neue Repräsentation der Eingabedaten interpretiert werden. Die Wahl von drei versteckten Knoten im Beispiel ist willkürlich.
Was bis jetzt berechnet wurde nennt man den *Vorwärtspass*. Aus einer Eingabe wurde die Ausgabe berechnet. Daran war aber noch nichts intelligent. Erst jetzt können die Parameter der Funktion, die Gewichte, aus diesem Beispiel lernen (Siehe Abbildung 4). Um diese zu verbessern braucht es eine *Verlustfunktion* die uns angibt, wie weit die Ausgabe vom korrekten Ziel entfernt ist. Eine mögliche Verlustfunktion

ist die absolute Differenz zwischen dem Ziel und der Ausgabe. Wenn man in oberen Beispiel davon ausgeht, dass die Eingabe wirklich zu einer Katze gehört, ergäbe sich:

$$\text{Verlust}(output, target) = |target - output| = |1.0 - 0.747| = 0.253$$

Um den Verlustwert zu minimieren, passt das Netzwerk die Gewichte schrittweise an. Diesen Teil übernimmt der *Optimierer*. Ein einfacher Optimierer ist zum Beispiel das Gradientenverfahren. Details dazu befinden sich in [12].

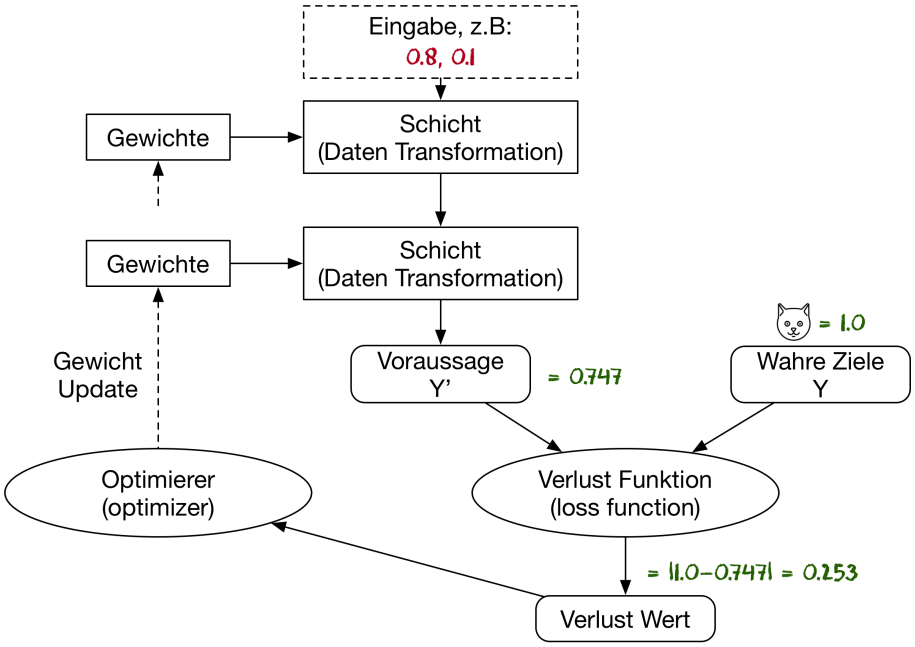


Abbildung 4: Visuelle Darstellung des Lernprozesses anhand eines Beispiels

Da bei diesem Typ von neuronalen Netzwerken alle Knoten miteinander verbunden sind, wird es oft *Dense Neural Network* genannt.
Die mathematischen Operationen in einem neuronalen Netzwerk lassen sich alle als Matrizen-Operationen berechnen. Mit Matrizen kann der Computer auf einer Grafikkarte und mit einem BLAS (*Basic linear algebra system*) sehr effizient rechnen [27] .

2.2 Convolutional Neural Networks

Convolutional Neural Networks sind eine sehr weit verbreitete Methode im Feld von *Computer Vision*. Der fundamentale Unterschied zwischen dem oben besprochenen *dense network* und einem *CNN* ist, dass ein *CNN* lokale Muster erkennen kann, wo hingegen das vorherige Netzwerk nur globale Muster erkennen konnte. Das bedeutet, dass ein Muster, das an einer bestimmten Stelle angetroffen wird, an jeder anderen Stelle ebenfalls erkannt wird. [6]

Um das zu erlauben, teilen gewisse Verbindungen das gleiche Gewicht. In Abbildung 5 (Oberer Teil) sind das die gleichfarbigen Verbindungen. Weniger Gewichte führen zusätzlich dazu, dass das Netzwerk schneller lernen kann.

Ein weiterer Vorteil von *Convolutional Neural Networks* ist, dass sie eine räumliche Hyrarchie von Mustern erlernen können. Wenn die Eingabe das Bild einer Katze

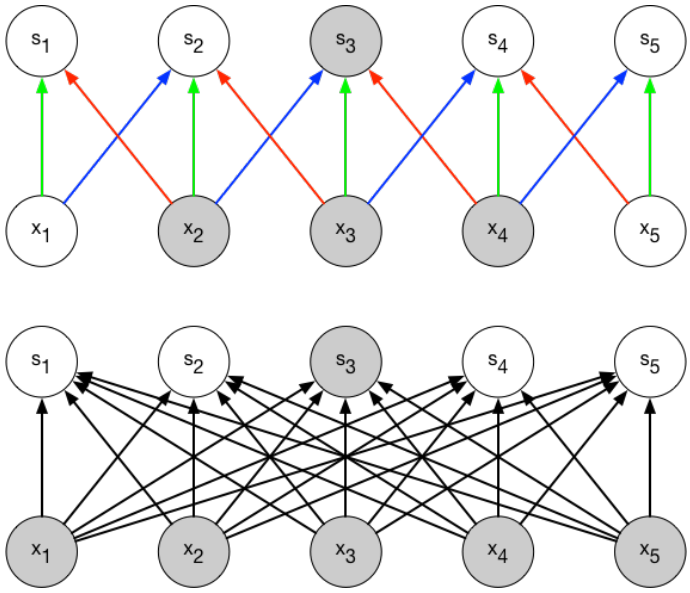


Abbildung 5: (*Oben*)1D Convolution mit *kernel* der Grösse 3. s_3 wird durch 3 inputs beeinflusst. (*Unten*) *Dense Network*. s_3 wird durch alle inputs beeinflusst.[11]

ist, wird zum Beispiel die erste Schicht unterschiedliche Kanten erkennen, die zweite Schicht dann einzelne Merkmale (z.b Augen), und so weiter.

Damit das gilt, muss aber der analysierte Bereich eines Knotens, von Schicht zu Schicht grösser werden. Deshalb wird meisten nach jedem *Convolution Layer* ein *Pooling Layer* gesetzt. Das *Pooling Layer* fasst mehrere Datenpunkte zusammen um dem nächsten Netzwerk eine grösseren Analysebereich zu verschaffen. Ein oft verwendetes Pooling-Verfahren ist *Max-Pooling*: Angrenzende Knoten werden zusammengefasst durch ihr Maximum.

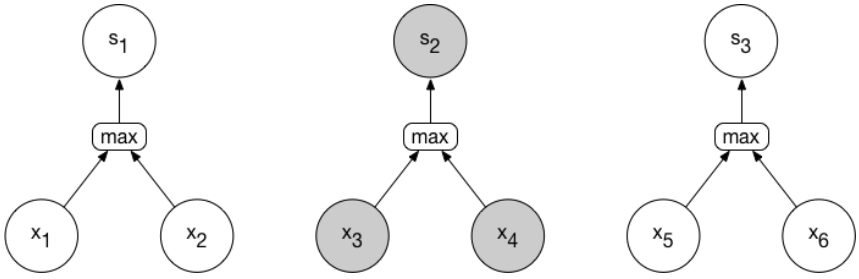


Abbildung 6: 1D Max-Pooling layer: z.B s_2 ist $\max(x3, x4)$

2.3 Recurrent Neural Networks

Eine gemeinsame Eigenschaft von allen *Dense Neural Networks* und CNN's ist, dass sie keinen Speicher haben. Bei jedem Vorwärtspass berechnet das Netzwerk alles von neuem ohne Erinnerungen an vorherige Durchlaufe. Dieses Verhalten ist das absolute Gegenteil vom menschlichem Denkprozess. Wenn wir einen Satz lesen, durchgehen wir ihn Wort nach Wort und merken uns den vorherigen Kontext.

Recurrent Neural Networks (RNN) bilden diesen Prozess vereinfacht nach. Sie besitzen eine interne wiederkehrende Schleife die dem Netzwerk Informationen aus

dem vorherigen Durchlauf bereitstellt (Siehe Abbildung 7). Die RNN Zelle berechnet dann die nächste Ausgabe sowohl aus der neuen Eingabe, wie auch mit den Erinnerungen der letzten Ausgabe. [6]

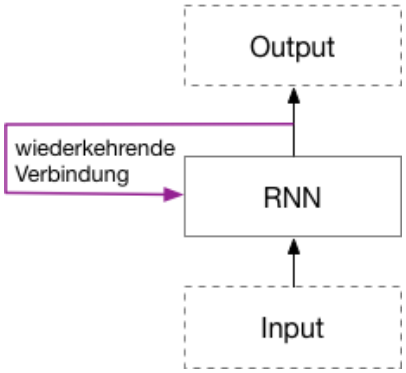


Abbildung 7: RNN mit Schlaufe

Der Vorgang lässt sich grafisch über die Zeit aufgerollt darstellen (Abbildung 8). In dieser Darstellung fällt auf, dass das Netzwerk theoretisch für jeden Schritt eine Ausgabe besitzt. Die zwischenliegenden Ausgaben sind vor allem wichtig, wenn man eine weitere Schicht an das Netzwerk anhängen will. Sonst behält man meist nur die letze Ausgabe, da diese indirekt Informationen über alle anderen beinhaltet.

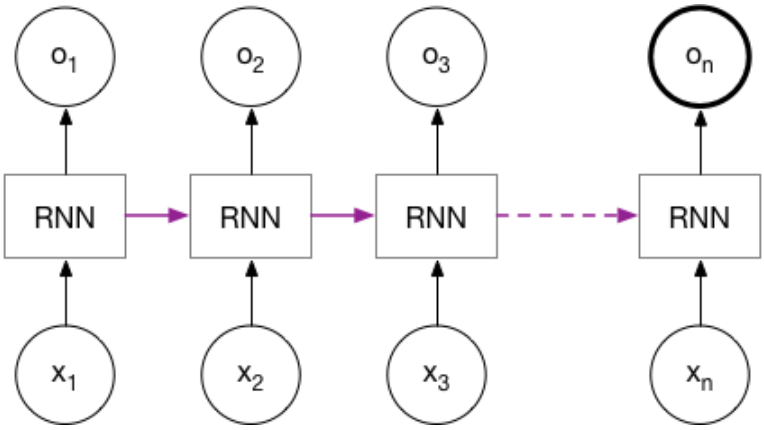


Abbildung 8: RNN aufgerollt über die Zeit

Das ganze Prinzip ergibt jedoch nur Sinn, wenn frühere Eingaben tatsächlich einen Einfluss auf spätere Ausgaben haben. Eine praktische Anwendung ist das Verarbeiten von zeitlichen Sequenzen wie Wetterdaten und Sprache. Die einzelnen Lernbeispiele werden zeitlich zerteilt und stückweise dem Netzwerk gefüttert. Eine fortgeschrittene Implementation von RNN Zellen ist unter anderem die *Gated recurrent unit* (GRU)[4]. Durch das Einführen von unterschiedlichen wiederkehrenden Verbindungen wird verhindert, dass ältere Signale langsam verschwinden, bzw. vergessen werden [6].

3 Daten

3.1 Datenquellen

Es gibt keine frei zugänglichen umfassenden Datensets für Sprachidentifikation. Datensets wie das *NIST Language Recognition Evaluation*[23] sind nur unter teuren Gebühren zugänglich. Wie verwandte Arbeiten empfehlen [32], wird darum ein eigenes Datenset zusammengestellt. Es werden gleichmässig Daten zu den Sprachen Deutsch, Englisch, und Französisch gesammelt. Die Daten stammen vom *Voxforge*[33] Datenset, von *Youtube*[35] und von *Librivox*[17].

Voxforge ist ein open-source Datenset für Spracherkennung. Es besteht aus vielen kurzen (1-10s), von Benutzern hochgeladenen, Audiodateien. Die englische Sprache dominiert mit rund 120h Audio das Datenset. Über alle drei Sprachen verteilt sind 190 Stunden Ton verfügbar. Die Audioqualität variiert je nach Benutzer.

Die Sprache ist langsam und deutlich verständlich. Sie hört sich eher künstlich an im Vergleich zu einem natürlichem Gespräch. Die Anzahl unterschiedlicher Sprecher ist gering.

Youtube dient als Quelle für abwechslungsreiche Sprache. Es werden populären Nachrichtenkanäle wie CNN, ARD, etc. verwendet (Siehe Tabelle 1). Die Aufnahmen werden oft von diversen Hintergrundgeräuschen begleitet und die Variation der Sprecher gross, da oft fremde Gäste eingeladen werden. Kehrseite ist, dass nicht garantiert werden kann, dass alle Aufnahmen tatsächlich die richtige Sprache beinhalten. Sendepausen und fremdsprachige Interviews kommen vereinzelt vor.

Sprache	Kanäle
Französisch	France24, FranceInfo
Deutsch	ARD, ZDF
Englisch	CNN, BBC

Tabelle 1: Youtube Kanäle

Librivox ist ein öffentlich abrufbares Hörbuch Datenset. Anstatt selbst Hörbücher zu selektieren, wird eine vorgefertigte Selektion verwendet [25]. Verfügbar sind sieben Stunden Aufnahmen mit 90 verschiedenen Sprechern.

3.2 Daten Auswahl

Die Modelle werden grundsätzlich mit den Daten von Voxforge und Youtube trainiert und ausgewertet. Es werden zwei unterschiedliche Datenquellen verwendet um die *Stichprobenverzerrung* zu minimieren. Stichprobenverzerrung bedeutet, dass das Datenset nicht repräsentativ für alle Sprachaufnahmen ist. Bei Voxforge ist zum Beispiel die Gefahr, dass das Modell sich überanpasst an die geringe Anzahl Sprecher. Anstatt die Sprache zu erkennen, könnte das Modell das Mikrofon des Sprechers identifizieren und jedem Sprecher eine Sprache zuordnen. Die Leistung des Modells für neue Sprecher wäre dann nicht besser als ein Zufallsgenerator.
Um die Stichprobenverzerrung zu messen werden die Modelle zusätzlich auf dem

Netz	Voxforge	Youtube	Librivox
Trainingset	56h	56h	-
Validationset	7h	7h	-
Testset	7h	7h	8.5h

Tabelle 2: Daten Verteilung

Librivox Datenset ausgewertet. Die Daten von Librivox teilen keine systematischen "Fehler" wie zum Beispiel Sprecher mit den Trainingsdaten. Das Librivox-Testset besteht aus insgesamt 2 Stunden Aufnahmen.
Von Youtube und Voxforge werden gemeinsam 139 Stunden Audiodaten heruntergeladen, was 100'000 5s Aufnahmen entspricht. Die Datenmenge wird bewusst klein gehalten, weil grössere Datenmengen ressourcenaufwändiger und ineffizienter wären. Die einzelnen Sprachen und Quellen sind zu gleichen Teilen repräsentiert.
Die Daten werden weiter in 80% Trainingsdaten, 10% Testdaten und 10% *Validationset* gespalten. Das Validationset wird verwendet um während dem Training zu beobachten, wie das Modell auf neue Daten reagiert. Die *Hyperparamter* (Parameter die das Netzwerk nicht selber lernen kann, z.B die Anzahl Knoten) werden manuell so angepasst dass das Netzwerk möglichst gut auf dem Validationset abschneidet. Tabelle 2 gibt eine Übersicht über die Verteilung der Daten.

3.3 Preprocessing

Sprache besteht aus Wörtern und verschiedene Sprachen unterscheiden sich grundsätzlich durch einen anderen Wortschatz. Wörter lassen sich wiederum in verschiedene Laute spalten. Sprachen unterscheiden sich also auch an den verschiedenen Abfolgen von Lauten, manchmal sogar an den verwendeten Lauten selbst. Die kleinste relevante Einheit für Spracherkennung, sowohl für den Menschen wie die Maschine, ist wahrscheinlich ein Laut.

Wenn der Computer mit dem Mikrofon aufnimmt, misst er kleinste Druckunterschiede, bzw. Schallwellen. Eine unkomprimierte Audiodatei zeigt den Schalldruck über die Länge der Aufnahme, siehe Abbildung 9 *(oben)*. Die einzelne Schallwelle ist für den Menschen nicht erkennbar, deshalb ist sie auch für die Sprache von keiner Bedeutung. Erst mehrere Schallwellen, bzw. die daraus folgende Frequenz lässt sich als Laut hören. Das Verfahren um aus einer Schallwelle die unterschiedlichen Frequenzen zu bestimmen heisst *Fourier-Transformation* [29]. Falls dem Netzwerk als Eingabe rohe Schallwellen gefüttert werden, muss es dieses Verfahren erlernen, um dann aus den Lauten die Sprache erkennen zu können. Allerdings ist die Fourrier-Transformation zu erlernen ein zusätzlicher Aufwand und fordert den Computer darum mehr. Um dem Algorithmus die Aufgabe zu erleichtern, kann man ihm darum als Eingabe die berechneten Laute anstatt der rohen Schallwelle geben.

Die Prozedur dem Algorithmus bereits vorgerechnete Werte zu füttern, heisst *Preprocessing* und ist weit verbreitet im Feld von *Machine learning*. Das Vorrechnen ist unter dem Namen *Feature Engineering* bekannt. *Features*, also Merkmale z.b Laute werden aus den Rohen Daten extrahiert. [vgl. 6]

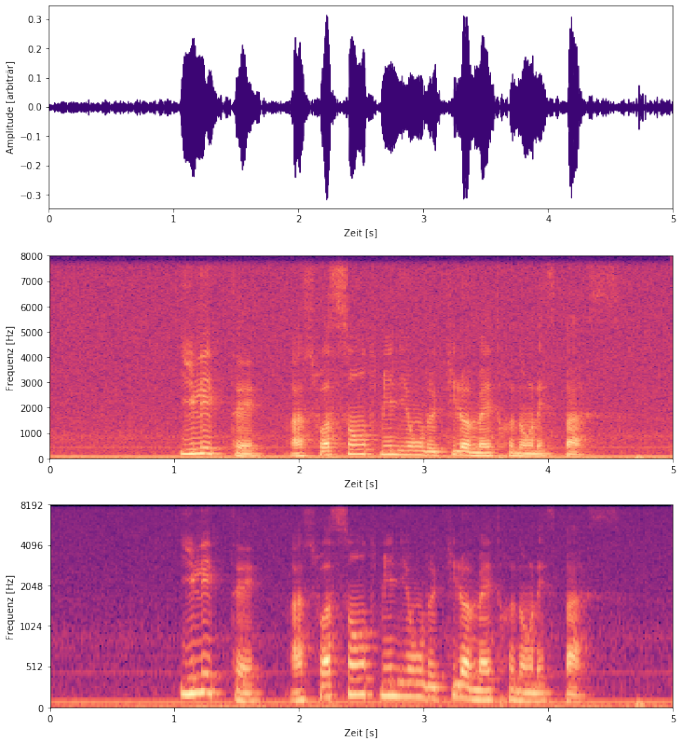


Abbildung 9: Audio-Preprocessing: *(oben)* Rohe Schallwelle, *(mitte)* Dezibel-Spectrogramm, *(unten)* Mel Dezibel-Spectrogramm

3.3.1 Spektrogramme

Spektrogramme sind grafische Darstellungen eines Hörsignals nach der Anwendung der Fourrier-Transformation[29, 'Spectrograms'], ersichtlich in Abbildung 9 *(Mitte)*. Die Lautstärke der Frequenzen wird in die Logarithmische Dezibell-Skala umgerechnet. Es hat sich empirisch gezeigt, dass die Wahl der Dezibell-Skala einen positiven Effekt auf die Modelle hat. Frequenzen über 10kHz können abgeschnitten werden, da menschliche Sprache grösst Teils darunter abläuft [30]. Im Spektrogramm lassen sich von Auge Muster erkennen und unterscheiden. Daraus folgt die Idee für die Klassifizierung Bilderkennungsmethoden zu verwenden. Der visuelle Charakter der Spektrogramme erlaubt zum Beispiel das verwenden von *2D Convolutional Neural Networks*.

3.3.2 Mel Filtering

Spektrogramme haben relativ viele Datenpunkte und sind deshalb recht aufwändig zu verarbeiten. Ein weiter *preprocessing* Schritt wird deshalb oft angewendet: *Mel Filtering*[34]. Die Frequenzen werden dabei in grössere Eimer gepackt. Unter 1kHz sind die Eimer linear verteilt und darüber logarithmisch, siehe Abbildung 9 *(unten)*. Das Modell entspricht unserer Hörfähigkeit, die recht präzise unter 1kHz arbeitet, höhere Frequenzen aber schlecht unterscheiden kann [30].

3.3.3 Implementation

Die Oben genannten Transformationen können vor dem Training direkt auf die Daten angewendet und abgespeichert werden. In diesem Fall hätte man die rohen Daten

löschen können. Allerdings sollte in dieser Arbeit sowohl an den rohen Daten wie auch an der Transformation flexibel experimentiert werden können, weswegen die Daten unbedingt beibehalten werden mussten.

Anstatt die Transformationen selber zu implementieren wird ein Framework verwendet. In diesem Fall bietet sich an *kapre*[5] an. Mit Kapre lassen sich währende dem Training die Daten in Echtzeit verarbeiten. Das Training dauert dabei aber 20% länger. Konkret verhält sich *kapre* wie eine Schicht vor dem eigentlichen Neuronalen Netzwerk.

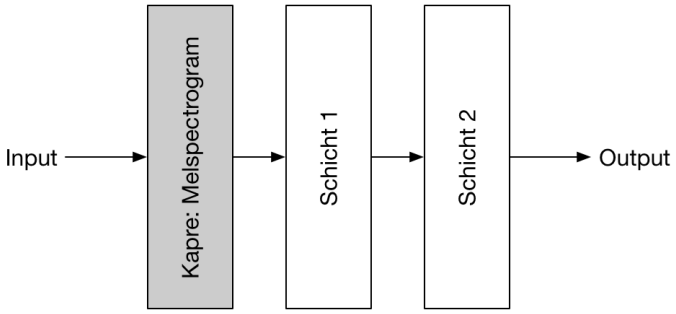


Abbildung 10: *Kapre*[5] als Schicht

4 Modelle

Im folgenden Teil werden die entwickelten Deep Learning Modelle vorgestellt. Für jedes Modell ist der grundlegende Ablauf derselbe (siehe Abbildung 11). Die Aufnahme wird vorab auf die Länge 5s zugeschnitten. Anschliessend wird ein Spektrogramm generiert. Die Werte des Spektrogramms werden in die Dezibell Skala umgerechnet und dann normiert. Die Grösse des Spektrogramms unterscheidet sich zwischen den Modellen. Schliesslich berechnet ein Neuronales Netzwerk aus dem Spektogramm eine Vorhersage für die drei Sprachen.
Insgesamt werden drei grundsätzlich verschiedene Neuronalen Netzwerk Architekturen vorgestellt. Die ersten zwei Modelle sind Convolutional Neural Networks während das dritte Modell eine Kombination zwischen CNN und Recurrent Neural Network ist. Jedes Modell endet mit einer Ausgabeschicht der Grösse drei. Jeder der der drei Werte steht für die Wahrscheinlichkeit einer Sprache. Um zu garantieren, dass die Summe der Werte 1 gibt wird die Aktivierungsfunktion *softmax* verwendet:

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=0}^{n} \exp(x_j)}$$

Beispiel:

$$\text{softmax}(\begin{bmatrix}0.5 & 0.8 & 0.3\end{bmatrix}) = \begin{bmatrix}0.32 & 0.43 & 0.26\end{bmatrix} = \begin{bmatrix}p_{FR} & p_{EN} & p_{DE}\end{bmatrix}$$

Die Funktion eignet sich generell für Klassifizierung, denn sie bildet die Werte auf eine Wahrscheinlichkeitsverteilung über die berechneten Ausgabeklassen ab [11, S. 180-184]. Der Wert einer Ausgabeklasse lässt sich auf diese Weise als die Wahrscheinlichkeit, dass die Eingabe zu ihm gehört, interpretieren. Für alle anderen Schichten wird immer die Aktivierungsfunktion *ReLU* verwendet.
Damit die Ziele des Netzwerks die gleichen Anzahl Dimensionen wie die Ausgaben haben, werden sie *One-hot*[6] kodiert. Das bedeutet, dass die richtige Sprache die Wahrscheinlichkeit 1.0 besitzt und alle anderen 0. Das Ziel für eine französische Aufnahme ist also $\begin{bmatrix}1.0 & 0.0 & 0.0\end{bmatrix}$, für eine englische Aufnahme $\begin{bmatrix}0.0 & 1.0 & 0.0\end{bmatrix}$, etc..
Obwohl die Ziele die gleiche Form wie die Ausgaben haben lässt sich der Verlustwert nicht so leicht berechnen wie beim Anfangsbeispiel. Die Standard Verlustfunktion in diesem Fall ist *Categorical Crossentropy*[6]. Für das Ziel $\boldsymbol{p}$ und die Voraussage $\hat{\boldsymbol{p}}$ ist die Funktion folgendermassen definiert:

$$\text{H}(\boldsymbol{p}, \hat{\boldsymbol{p}}) = -\sum_i p_i \cdot \log \hat{p}_i$$

Das diese Verlustfunktion sinnvoll ist, lässt sich intuitiv verstehen: Das Ziel des Netzes soll es sein, die Wahrscheinlichkeit der richtigen Sprache zu maximieren. So soll zum Beispiel bei $\boldsymbol{P} = \begin{bmatrix}0 & 1.0 & 0\end{bmatrix}$ und $\hat{\boldsymbol{P}} = \begin{bmatrix}P_o & P_1 & P_2\end{bmatrix}$, P_1 maximiert werden. Die Wahrscheinlichkeit der richtigen Sprache lässt sich einfach berechnen:

$$\text{L}(\boldsymbol{p}, \hat{\boldsymbol{p}}) = \sum_i p_i \cdot \hat{p}_i$$

Im Beispiel oben fallen P_0 und P_2 weg und $1 \cdot P_1$ bleibt übrig, was genau der Wahrscheinlichkeit der richtigen Sprache entspricht. Ab hier lässt sich nachvollziehen, das genau wie wir L maximieren wollen, $-L$ (wie der Verlustwert) minimiert werden

soll. Da log eine monotonische Funktion ist entspricht $\min(p)$ auch $\min(\log(p))$. Das Resultat ist genau die Categorical Crossentropy. [vgl. 10]

$$\max[\mathrm{L}(\boldsymbol{p}, \hat{\boldsymbol{p}})] = \min[-\mathrm{L}(\boldsymbol{p}, \hat{\boldsymbol{p}})]$$

$$\min[-\mathrm{L}(\boldsymbol{p}, \hat{\boldsymbol{p}})] = \min[-\mathrm{L}(\boldsymbol{p}, \log(\hat{\boldsymbol{p}}))] = \mathrm{H}(\boldsymbol{p}, \hat{\boldsymbol{p}})$$

Die Parameter der Modelle, wie zum Beispiel die Grösse des Spektrogramms wurden empirisch bestimmt. Insgesamt wurden fast 200 Experimente durchgeführt. Allerdings konnte längst nicht jede mögliche Kombination von Parametern probiert werden, da jeder Durchlauf zeitaufwändig und nicht parallelisierbar ist.

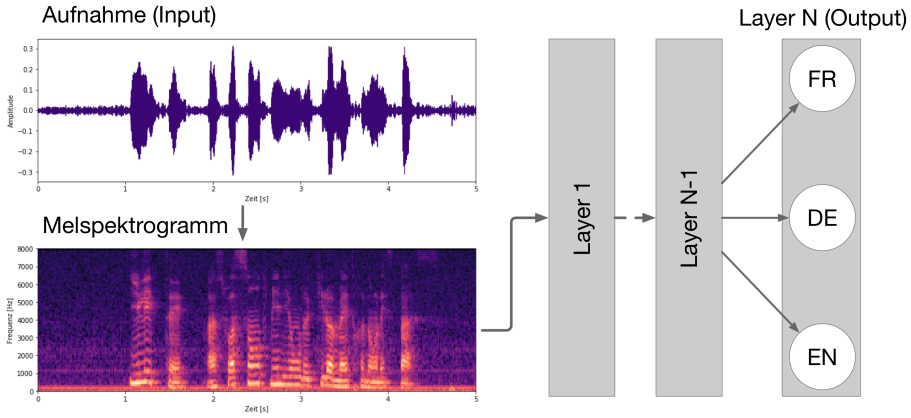


Abbildung 11: Grundlegender Aufbau aller Modelle

4.1 CNN

Das CNN Netzwerk akzeptiert Spektrogramme der Grösse 28x313, wobei 28 verschiedene Mel-Frequenzeimer berechnet werden an 313 Zeitpunkten. Das Netz besteht aus drei Convolution-MaxPooling Blocks und zwei Dense Schichten (Abbildung 12). Die Convolution-Grösse ist immer 3x3 und MaxPooling geschieht im Bereich 2x2. Die Anzahl Convolution's (Kanäle) nimmt von 64 auf 128 zu.
Die *Fully Connected* Schicht besteht aus 512 Knoten mit 30% *Dropout*. Bei Dropout wird ein zufälliger gewählter Anteil der Eingabe der Schicht mit 0 ersetzt. Das Netz lernt dann, ohne diese Verbindungen auszukommen und somit mehrere Verbindungen einzelnen starken Verbindungen vorzuziehen. Die Eigenschaft hat einen positiven Einfluss auf die Robustheit des Netzes gegenüber neuen Daten [6, Kap. 4.4.3].
Die Architektur entspricht massgeblich der vorgestellten Architektur in [32] plus Dropout.

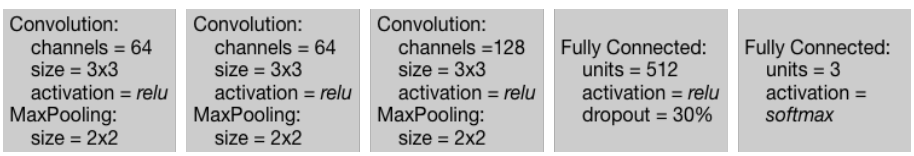


Abbildung 12: CNN Architektur

4.2 MobileNet-CNN

Andere Arbeiten hatten gezeigt, dass grosse CNN Architekturen die auf dem *ImageNet*[9] Datenset (Riesiges Bilddatenset mit tausenden Bidklassen) stark sind, ebenfalls für Audioklassifizierung geignet sind [13]. Um Leistung zu sparen wurden Experimente mit *MobileNet* durchgeführt. MobileNet ist eine bekannte CNN Architektur entwickelt von *Google* [28, (V2)]. Das Modell ist speziell entwickelt für Mobilgeräte, die in der Regel schwächere Hardware als Desktop-Computer besitzen. Das Ziel war ein effizientes Netzwerk für Bilderkennung im grossen Umfang, wie z.B ImageNet. Da das Modell in dieser Arbeit nur zwischen drei Klassen unterscheiden muss, wird die Anzahl Knoten im Netzwerk so weit wie möglich verkleinert, bzw. der Parameter α im Paper wird auf 0.25 gesetzt.
Das Netzwerk besitzt insgesamt 21 Schichten. Das Netzt funktioniert am bessten mit Eingaben der Grösse 224x224, der Bildgrösse in ImageNet. Die Spektrogramme werden darum auf dieses Format skaliert.

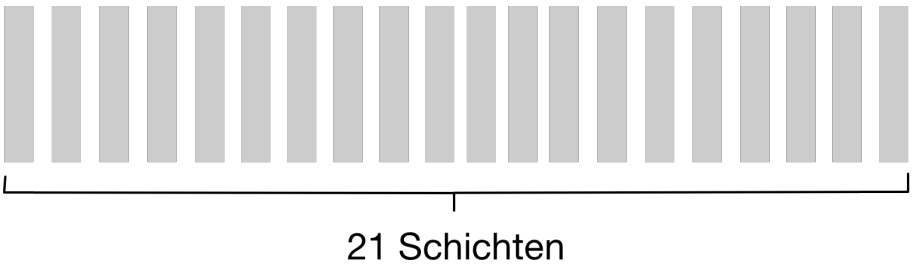


Abbildung 13: MobileNet Architektur

4.3 CRNN

Convolutional Recurrent Neural Networks, eine Mischung aus CNN und RNN, wurden bereits erfolgreich auf Sprachidentifikation angewendet [2][14]. RNN's für Tonaufnahmen zu verwenden ist sinnvoll, denn Audiodaten sind naturgemäss zeitliche Folgen, also ideal für RNN's. RNN's haben aber den Nachteil, dass sie langsam lernen. Darum soll der CNN Teil im Vorfeld aus dem Spektrogramm eine kompaktere Repräsentation berechnet, z.b eine Abfolge von Phonemen. Da die Lernzeit proportional zur Eingabegrösse ist, lernt der RNN teil mit dieser Eingabe schneller. Es gibt keine Möglichkeit zu verifizieren, ob das CNN wirklich Phoneme berechnet aber die Resultate zeigen, dass das CRNN besser abschneidet als das CNN.
Der CNN Teil besteht aus fünf Blocks von Convolution, MaxPooling und *Batch Normalization* (Siehe Abbildung 14). Die Idee von *Batch Normalization* ist, dass die Ausgabe der vorherigen Schicht so normalisiert wird, dass sie den Durchschnitt 0 und die Varianz 1 besitzt. *Batch Normalization* bringt den vor allem den Vorteil, dass das Modell schneller konvergiert, also schneller bessere Leistungen erbringt. [16]
Die Anzahl Convolutions steigert sich schrittweise von 16 auf 64. Das Convolution-Fenster hat immer die Grösse 4x4 und MaxPooling das Fenster 2x2. Bei den Convolutions wird 0.01 *L2-Regularisation* verwendet:

$$Verlust = Verlust + \sum_i w_i^2$$

Das bedeutet, dass dem Verlustwert zusätzlich das Quadrat der Gewichte addiert wird [11, Kap. 7.1.1]. Übermässig hohe Gewichte werden überproportial gewertet. Die Regularisation hat das gleiche Ziel wie Dropout, das Netz robuster zu gestalten.

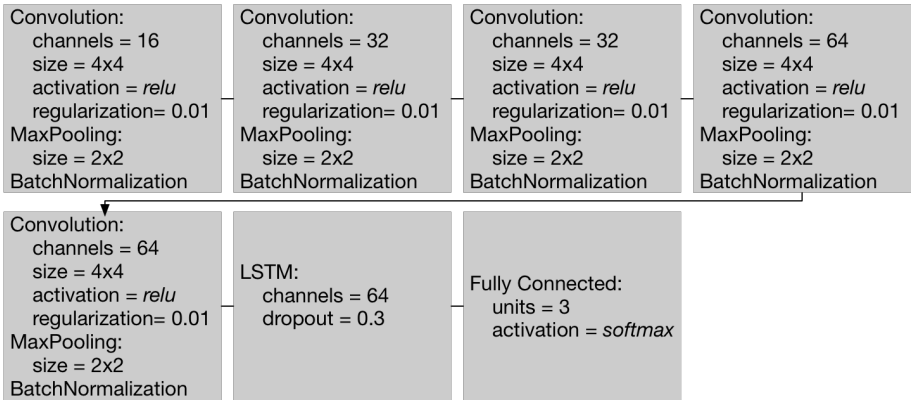


Abbildung 14: CRNN Architektur

5 Umsetzung und Resultate

5.1 Implementierung

Was in dieser Arbeit theoretisch gemacht wurde, wurde bis hierhin ausführlich beschrieben. Die praktische Implementierung aller Schritte wurde aber noch nicht behandelt. Dieses Kapitel erklärt die genaue Umsetzung aller Teile und nennt die verwendeten Werkzeuge. In anderen Worten wird hier die praktische Arbeit anschaulich gemacht. Für alle Teile der Arbeit gilt, dass die Programmiersprache *Python* ist. Python eignet sich generell für Deep Learning weil viele Bibliotheken für die Sprache geschrieben sind. Ausserdem hat Python den Vorteil, dass es einsteigerfreundlich ist. Python kann entweder als Script geschrieben werden, also als Datei die ausgeführt wird, oder in einem interaktiven *Jupyter Notebook*[26]. Jupyter Notebook's bestehen unter anderem aus mehreren Textzellen die einzeln ausgeführt werden können aber den gleichen Thread teilen. Das macht Jupyter Notebooks unglaublich praktisch für das Experimentieren und Darstellen von Daten.

Die Arbeit lässt sich in vier Teile gliedern. Abbildung 15 zeigt die Teile schematisch. Als erstes müssen die zu verwendeten Daten aus den verschiedenen Quellen gesammelt werden. Bevor die Daten aber heruntergeladen werden muss in manchen Fällen eine Selektion geschehen. So kann bei Youtube aufgrund der riesigen Anzahl an verfügbaren Videos unmöglich die gesamte Datenmenge gesammelt werden. Aus diesem Grund wurde bei Youtube manuell nach geeigneten Kanälen bzw. Playlists gesucht. Nach dem klar ist was gesammelt werden soll kann man mit Herunterladen starten. Dafür werden Python-Scripts geschrieben die die beträchtliche Menge Daten automatisiert herunterladen. Für die YouTube Daten wird die externe Bibliothek *Youtube-dl*[36] benutzt.

Der zweite Schritt ist das Erforschen der Daten. Darunter wird zum Beispiel das herausarbeiten von wichtigen Unterschieden zwischen den Datenquellen verstanden.

Abbildung 15: Überblick über die Arbeitsschritte

Unter anderem wurde in diesem Schritt eine einheitliche Länge und Abtastrate der Aufnahmen festgelegt. Für diesen Teil sind Jupyter Notebook's ideal. Für das einlesen der Audiodaten wird die Bibliothek *Librosa*[20] verwendet und alfällige Grafiken werden mit der Bibliothek *Matplotlib*[15] erstellt.
Als nächstes können die Daten korrekt strukturiert werden. Das heisst das die Daten einerseits in Training, Validation und Testdaten gespalten werden während sie gleichzeitig nach der Sprache geordnet sind. Vor dem definitiven abspeichern der Daten werden sie auch auf die richtige Länge zugeschnitten. Das Format der Daten ist ein *Numpy*-Array[24]. Numpy stellt praktische Funktionen für numerische Operationen bereit und kann die Daten effizient abspeichern. Programmiert wird dieser Teil in einer Mischung aus Python-Scripts und Jupyter Notebook's.
Zum Schluss kommt Machine Learning ins Spiel. Der letze Schritt ist das Experimentieren mit unterschiedlichen Modellen. Erst jetzt werden Netzwerke trainiert und ausgewertet. Die zwei wichtigen Bibliotheken für den Machine Learning Teil sind *Keras*[7] und *Tensorflow*[19]. Tensorflow ist die Machine Learning Bibliothek von Google. Dank Tensorflow müssen keine neuronalen Netze von Grund auf programmiert werden. Tensorflow übernimmt alle Arbeit ausser das tunen von Hyperparametern. Keras ist lediglich ein einfaches Interface für Tensorflow und andere Machine Learning Bibliotheken. Der Code für das Trainieren eines einfachen Keras Modells findet sich im Anhang.

5.2 Auswertung an Voxforge und YouTube

An dieser Stelle werden die drei Modelle ausgewertet. Es wird gemessen wie gut die Modelle Aufnahmen klassifizieren. Die verwendete Messgrösse um die Modelle zu vergleichen ist die Genauigkeit. Die Genauigkeit beschreibt den Anteil an richtig klassifizierten Aufnahmen:

$$Genauigkeit = \frac{\#\text{ Korrekt klassifizierte Aufnahmen}}{\#\text{ Total Aufnahmen}}$$

Genauigkeit ist in diesem Fall eine angebrachte Messgrösse, weil die Daten gleichmässig über die Sprachen verteilt sind. Ist dies nicht der Fall kann die Genauigkeit täuschen. Wenn zum Beispiel 90% der Daten Deutsch wären, würde hypothetisch ein Modell dass immer Deutsch ausgiebt 90% Genauigkeit haben. Obwohl das hypothetische Modell extrem naiv ist, täuscht die hohe Genauigkeit. Im Fall dieser Arbeit wurde darum ausdrücklich auf eine gleichmässige Verteilung der Daten geachtet.
Die Modelle werden separat an allen drei Datenquellen ausgewertet. Für jedes Datenset wurden alle Modelle drei Mal trainiert. Die schlussendliche Genauigkeit ist die beste der drei Versuchen. Die Genauigkeiten am Voxforge Testset sind in Tabelle 3 geschrieben. Das CRNN führt mit fast 97%, dicht gefolgt vom CNN. Das MobileNet besitzt eine mehr als 10% tiefere Genauigkeit. Obwohl der Unterschied zwischen CRNN und CNN absolut klein erscheint, ist er relativ zum Fehler beträchtlich. Das CRNN hat eine mehr als 30% kleinere Fehlgenauigkeit als das CNN.

Netz	Architektur	Genauigkeit
CNN	3 Conv	95.5%
CRNN	4 Conv + GRU	96.9%
MobileNet	28 Conv + Dense	86.1%

Tabelle 3: Beste Voxforge Genauigkeit

Die Genauigkeiten am Youtube Testset sind in Tabelle 4 geschrieben. Sämtliche Modelle haben am Youtube Testset eine tiefere Genauigkeit als am Voxforge Testset. Das MobileNet vollzieht mit einem Verlust von 5.9% Genauigkeit den grössten Fall. Das CNN und CRNN besitzen beide eine Genauigkeit von 94.7%.
Die bisherigen Resultate zeigen, dass das CRNN insgesamt die höchste Genauigkeit hat. Trotzdem heisst das nicht, dass das CRNN in jedem Fall besser ist als die anderen beiden Modelle. Der Beweis dafür ist eine Kombination aller Modelle oft besser ist als die einzelnen Modelle. Ein Modell berechnet aus dem Durchschnitt der drei Modelle erreicht bei Voxforge eine Genauigkeit von 97.9%. Das bedeutet eine erneute Verkleinerung der Fehlgenauigkeit um mehr als 30%.

Netz	Architektur	Genauigkeit
CNN	3 Conv	94.7%
CRNN	4 Conv + LSTM	94.7%
MobileNet	28 Conv + Dense	80.2%

Tabelle 4: Beste YouTube Genauigkeit

Das CRNN hat zwar die höchste einzelne Genauigkeit aber das kommt mit einem Preis in Form der Trainingsdauer. Das CRNN braucht mehr als drei mal so lang zu trainieren wie das CNN. Das Genauigkeit/Zeit Verhältnis des CNN ist bedeutend grösser als das des CRNN. Die Trainingszeiten sind in Tabelle 5 aufgetragen.

Netz	Trainingsdauer
CNN	20 min
CRNN	70 min
MobileNet	60 min

Tabelle 5: Trainingsdauer

Eine weitere Methode neben der Genauigkeit um die Modelle auszuwerten ist ein Wahrheitsmatrix aufzustellen. Die Reihen der Matrix entsprechen der Sprache der Aufnahmen während die Spalten die berechnete Antwort des Algorithmus unterscheiden. Das Feld oben links zeigt zum Beispiel die Anzahl Aufnahmen die Französisch sind und als Französisch klassifiziert wurden. Die Wahrheitsmatrix gibt viel mehr Details her als die Genauigkeit alleine. Abbildung 16 zeigt die Matrix vom CRNN Modell an den kombinierten Testdaten von Youtube und Voxforge.
Das interessante was an dieser Matrix ablesbar ist, ist dass das Modell Englisch und Deutsch relativ oft verwechselt. Französisch und Englisch werden ebenfalls öfter verwechselt als Französisch und Deutsch.

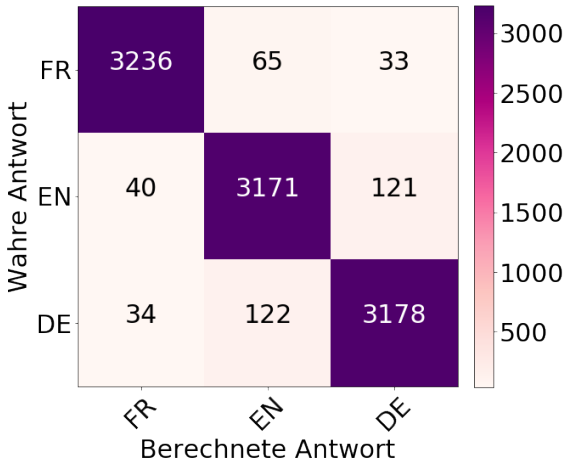


Abbildung 16: Wahrheitsmatrix von CRNN an Voxforge und Youtube (Total 10'000 Aufnahmen)

5.3 Auswertung an Librivox

Die Genauigkeit bei den Voxforge und Youtube Daten war serh hoch weil die Modelle die beiden Quellen bereits kannten. Obwohl das Testset unterschiedlich ist vom Trainingsset, teilen sie viele Gemeinsamkeiten. Die Librivox Daten sind den Modellen hingegen komplett fremd. Mit der Auswertung an den Librivox Daten wird klar wie gut das Modell generalisiert, bzw. wie gross die Stichprobenverzerrung ist.
Tabelle 6 zeigt die Genauigkeiten auf dem Librivox Testset. Das CNN verliert im Vergleich zu vorher rund 6% Genauigkiet. Das CRNN hingegen erreicht immer noch 94.1% Genauigkiet. Am schlimmisten geht es dem MobileNet, das auf eine Genauigkeit von 56.8% sinkt. Anhand dieses Verlust lässt sich klar ablesen, dass das MobileNet sich an die Trainingsdaten überangepasst hat. Das CRNN und CNN auf der anderen Seite generalisieren überraschend gut.

Netz	Architektur	Genauigkeit
CNN	3 Conv	88.1%
CRNN	4 Conv + LSTM	94.1%
MobileNet	28 Conv + Dense	56.8%

Tabelle 6: Beste Librivox Genauigkeit

Genau wie bei Voxforge und Youtube lässt sich wieder eine Wahrheitsmatrix zu detaillierten Auswertung formen. Abbilunng 17 zeigt die Matrix des CRNN Modells. Das auffällige an dieser Matrix ist, dass Englisch relativ selten mit anderen Sprachen verwechselt wird. Deutsch und Französich werden hingegen oft fälschlicherweise als Englisch erkannt.

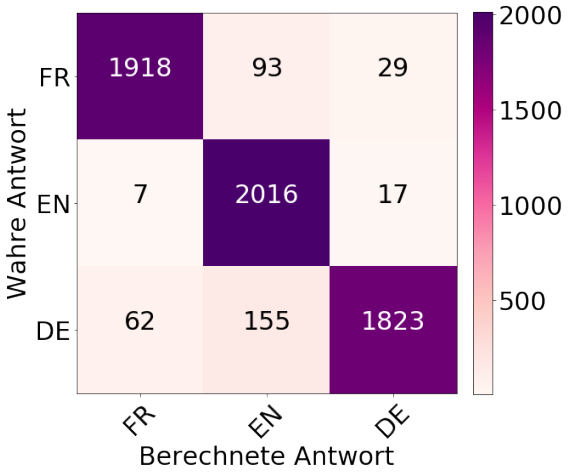


Abbildung 17: Wahrheitsmatrix von CRNN an Librivox (Total 6120 Aufnahmen)

6 Diskussion

Im Rahmen dieser Arbeit wurde ein System zum klassifizieren von Sprachaufnahmen implementiert. Ein hybrides Modell aus CNN und RNN besass die höchste Genauigkeit. Das Gewinner CRNN Modell erreicht beim Librivox Testset, das von den Trainingsdaten unabhängig ist, eine Genauigkeit von 94.1%. Das CNN Modell folgt in Genauigkeit an zweiter Stelle mit 88.1% beim Librivox Testset. Bei weiterm die tiefste Genauigkeit besitzt das MobileNet Modell bei den Librivox Testdaten mit 56.8%. weit hinter den ersten beiden Modellen. Die hohe Korrektheit der ersten zwei Modelle bestätigt, dass tiefe neuronale Netze eine angemessene Methode für Sprachidentifikation sind.

Der Vergleich der Resultate mit anderen Arbeiten ist nur mit Vorsicht möglich. Der wichtigste Unterschied ist, dass andere Daten verwendet werden. Weil die Genauigkeit des Modells stark von der Menge und der Stichprobenverzerrung der Daten abhängt, können dementsprechend nur andere Resultate entstehen. Glücklicherweise sind Arbeiten zu Automatischer Sprachidentifikation im Internet jedoch frei erhältlich, denn das Problem ist in der Informatik recht jung. Erst in den 70'er Jahren wurde man für das weiterleiten von Telefonanrufen auf das Thema aufmerksam. Eine lange Zeit galt, dass die bessten Systeme diejenigen waren, die die fortgeschrittensten sprachlichen Merkmale berechneten. Der Nachteil an diesen Systemen ist, dass die Erweiterung auf andere Sprachen mit einem grossem Aufwand verbunden ist. [18]
Erst mit dem Aufkommen von Deep Learning als konkurrenzfähige Methode im letzen Jahrzehnt kamen wieder Systeme mit weniger Preprocessing auf [6]. 2009 erreichte Grégoire Montavon mit Deep Learning für 3 Sprachen auf dem Voxforge Datenset eine Genauigkeit von 80.1% [21]. Das Voxforge Datenset hat sich seither verändert. 2016 wurde mit einem sehr ähnlichen Modell wie das CNN Modell dieser Arbeit Genauigkeiten von 93% und 85% für 2 Sprachen respektive 4 Sprachen gemessen [32]. Die Autoren haben ihre Modelle mit dem Montavon Modell verglichen und gezeigt, dass sie es deutlich übertreffen. Aus diesem Grund kann angenommen werden, dass die hier vorgestellten Modelle, sein System ebenfalls deutlich übertreffen. Hrayr et al. [14] stellen im selben Jahr ein CRNN Modell für einen Wettbewerb vor, dass zwischen 176 Sprachen mit 99.67% Genauigkeit differenzieren kann. Ein weiteres CRNN Modell von Bartz et al. [2] erreicht 2017 96% bei der Unterscheidung von vier Sprachen.
Die in dieser Arbeit vorgestellten System leisten vergleichbare Genaugigkeiten wie die genannten Modelle. 94.1% ist eine überraschend hohe Genauigkeit unter anbetracht das das Youtube Datenset nicht fehlerfrei ist. Das Datenset wurde bekanntlich nicht manuell gesäubert. Es können dementsprechend Aufnahmen ohne Sprache oder mit fremdsprachigen Interviews enthalten sein. Der Anteil von fehlerhaften Aufnahmen ist jedoch wahrscheinlich klein. Der Fall auf 80% Genauigkeit beim Librivox Datenset ist erwartbar. Das Modell ist für die Youtube und Voxforge Daten optimiert und nicht für Hörbücher. Die Librivox Daten haben zum Beispiel unterschiedliche Hintergrundgeräusche und Aufnahmeprotokolle. 80% bleibt in jedem Fall ein relativ grosse Genauigkeit. Ein Zufallsgenerator würde im Kontrast nur 33% erreichen. Es kann also aus dem Resultat abgelesen werden, dass das Modell angemessen generalisiert.

Bis jetzt wurde das Modell nur an öffentlich verfügbaren Daten ausgewertet wo klar war welche Sprache gesprochen wurde. In der Realität soll das Modell aber helfen die Sprache von unbekannten Aufnahmen zu erkennen. In anderen Worten soll das Modell in der Praxis angewendet werden. Dafür wurde ein Interface programmiert wo Benutzer sich selbst Aufnehmen und das Modell damit abfragen können. Das Interface ist in Form einer Webseite auf den Schulservern frei verfügbar. Die meisten Mobilgeräte und Computer sind kompatibel. Abbildung 18 zeigt die Seite nach dem hochladen einer zufälligen Aufnahme. Die Umgebung bei Smartphone Aufnah-

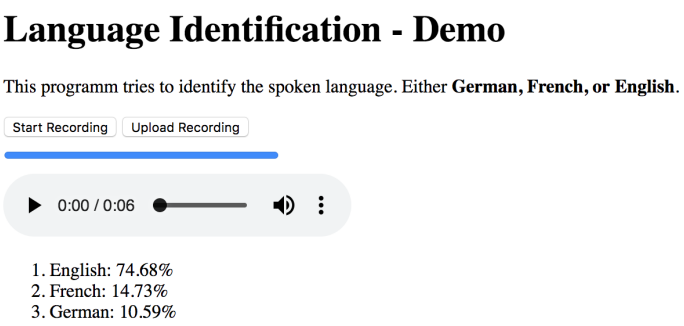


Abbildung 18: Interface Schnappschuss

men ist erneut eine andere wie die von Voxforge und YouTube. Die Genauigkeit die dort erreicht wird ist eine weitere Art die Generalisierbarkeit des Modells zu messen. Aus Neugier wurde ein sehr kleines Datenset aus Aufnahmen von Verwandten und Freunden zusammengestellt. Es wurde darauf geachtet, dass die Sprache deutlich verständlich ist. Insgesamt sind es nur 70 ungleichmässig verteilte Aufnahmen. Die Resultate daraus sind wegen der kleinen Anzahl und Varianz kaum signifikant. Das Programm entdeckt in 70% der Fälle die richtige Sprache. Eine Programm mit Genauigkeit 70% ist in mancher Hinsicht kein guter Klassifizierer. Kommerziell einsetzbar ist das System damit sicherlich nicht. Für ein Callcenter würde das bedeuten, dass fast jeder dritte Anruf dem falschen Mitarbeiter zugewiesen wird. Das ist aus Sicht der Kunden katastrophal. Also stellt sich die Frage woher der Fehler kommt, bzw. was man noch verbessern kann.

An erster Stelle sind die Daten ein Problem. Die Trainingsdaten sind nicht repräsentativ für alle möglichen Daten, unter anderem Handy-Aufnahmen. Um die Stichprobenverzerrung weiter zu minimieren muss man mehr Daten beschaffen. Die Daten sollten so vielfältig wie möglich sein. Es ist nutzlos unendlich viele Daten zu besitzen wenn die sie sich alle sehr ähnlich sind. Einzelne relevante Merkmale sind unter anderem das Geschlecht, die Stimmhöhe, Hintergrundgeräusche, die Qualität der Aufnahme und die Geschwindigkeit. Je mehr Kombination es gibt desto besser. Was macht man aber, wenn man nur eine begrenzte Menge an Daten zur Verfügung hat? In diesem Fall gibt es eine Methode namens *Data Augmentation*. Die Lösung ist, aus den vorhanden Daten weitere Daten zu erschaffen. Manche Merkmale lassen sich nämlich automatisch anpassen. Beispielsweise kann der Computer ohne Problem die Geschwindigkeit und die Qualität der Aufnahme adaptieren. Der Computer passt bei Data Augmentation in jedem Durchlauf zufällig die Merkmale der einzelnen Aufnahmen an. Im Endeffekt wird dem Netzwerk also nie genau die gleiche Aufnahme gezeigt. Das verhindert das das Netzwerk sich an eine endli-

che Menge von Daten überanpasst. Zudem wird die Stichprobenverzerrung dadurch minimiert. In der Regel hat Data Augmentation immer einen positiven Effekt. Darum ist Data Augmentation eine mächtige und weit verbreitete Methode. In dieser Arbeit wurde primär keine Data Augmentation angewendet, weil man die falsche Vorstellung hatte, dass sowieso genug Daten vorhanden wären. Es stellt sich aber im Nachhinein heraus, dass Data Augmentation immer besser ist als dem Netzwerk die gleichen Daten mehrmals zu zeigen.
An zweiter Stelle sind Mängel beim Training. Manchmal sinkt die Genauigkeit abrupt um im nächsten Durchlauf wieder zu steigen. Die genauen Daten dazu sind online verfügbar. Die Schwankungen können nur daran liegen, dass das Validationsset und das Testset zu klein sind. Das führt dazu, dass sie leicht beeinflussbar sind. Eine Erhöhung auf 25% der gesamten Daten ist für beide zu empfehlen. Andere Hyperparamter oder ganz andere Modelle könnten ebenfalls besser sein. Allerdings wird für viele Experimente ein leistungsstarker Computer benötigt. Mehr Leistung ist immer hilfreich.
Noch mehr Verbesserungspotenzial liegt wahrscheinlich im Verfahren *Learning from Between-class Examples for Deep Sound Recognition*(2017)[31]. Yuji et al. stellen eine vielversprechende Methode vor, die die Genauigkeit in jedem ihrer Versuche verbessert. Die Idee ist es, Aufnahmen von verschiedenen Klassen zu kombinieren um eine Mischaufnahme zu erfinden deren Ziel ebenfalls zwischen den Klassen liegt. Vereinfacht angewendet auf Sprachidentifikation könnte das bedeuten, dass eine deutsche und eine französische Aufnahme gleichzeitig abgespielt werden um zu einer Aufnahme kombiniert zu werden. In dieser Aufnahme würden also zwei Sprecher parallel sprechen. Das Ziel der Aufnahme wäre ebenfalls ein Mix z.B $\begin{bmatrix}0.5 & 0 & 0.5\end{bmatrix}$. Das schöne am Verfahren ist, dass es einfach zu verstehen ist und doch sehr wirksam. Leider reichte die Zeit nicht aus um die Methode zu implementieren.

Zusammenfassend lässt sich sagen, dass immer noch Potential nach oben bleibt. Nebst dem Produkt ist immer auch der Weg das Ziel. Es wurde enorm viel gelernt und Erfahrung gesammelt über Deep Learning. Auf der einen Seite zeigt das Projekt wie erstaunlich einfach künstliche Intelligenz in der Praxis ist. Ohne jahrelanges Studium ist man in der Lage das meiste zu verstehen und zu implementieren. Nichts daran ist magisch. In vielerlei Hinsicht ist Deep Learning momentan keine absolute Wissenschaft mit wahr und falsch. Ein grosser Teil der Optimierung besteht aus heuristischen Experimenten. Bei Hyperparametern gibt es noch grossen Forschungsbedarf. Wenn einmal klar ist wie diese gesetzt werden müssen, wird Deep Learning wirklich bereit sein für jedermann. In Zukunft wird es wohl einfachere Inferfaces geben, denen man nur das Problem angeben muss und der Rest automatisch erledigt wird. Auf der anderen Seite ist Deep Learning mathematisch ein kompliziertes Gebiet. Die Grundlagen können in Zukunft nur noch anspruchsvoller werden. Um wirklich mitforschen zu wollen, sind Mathematikkentnisse auf Universitätsstufe unabdingbar. Lineare Algebra ist hier extrem nützlich. Ein ebenfalls nicht zu unterschätzender Teil war das Sammeln und Vorbereiten der Daten. Die Vorstellung man könnte sofort an das Experimentieren ran gehen ist trügerisch. Wenn ein Deep Learning System von null aus aufgebaut wird, spendet man mindestens so viel Zeit bei der Vorbereitung der Daten wie beim Experimentieren. Die wichtige Frage ist wie lange Deep Learning noch so populär bleiben wird. Wird es in ein paar Jahren überhaupt noch relevant sein? Der Nutzen wird auf jedem Fall bleiben während die

mediale Aufmerksamkeit wahrscheinlich sinken wird. Es wird gehofft das der Leser etwas nützliches für die Zukunft gelernt hat

7 Anhang

7.1 Beispiel-Modell Code

```
# Importieren von Bibliotheken
from keras import models, layers
from kapre.time_frequency import Melspectrogram

# Laden von Daten
data_path = '../path/to/data'
train_data, train_labels = DataFeed.Dataset.create(data_path,
                                    ['train/voxforge', 'train/youtube
                                        '],
                                    num=50000)
val_data, val_labels = DataFeed.Dataset.create(data_path,
                                    ['val/youtube', 'val/voxforge'],
                                    num=10000, shuffle=True)

# Definieren des Modells
architecture = [
    Melspectrogram(n_dft=512, input_shape=(1, 5 * 16000,),
                   padding='same', sr=16000, n_mels=28,
                   fmin=0.0, fmax=10000, power_melgram=1.0,
                   return_decibel_melgram=False,
                   trainable_fb=False,
                   trainable_kernel=False),
    layers.Conv2D(64, (3, 6), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 6), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 6), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(1024, activation='relu'),
    layers.Dense(3, activation='softmax')
]

model = models.Sequential(architecture)

# Zusammenbauen des Modells
model.compile(optimizer='Rmsprop',
              metrics=['accuracy'],
              loss='categorical_crossentropy')

# Trainieren des Modells
model.fit(train_data, train_labels,
          batch_size=64,
          epochs=9,
          validation_data=(val_data, val_labels))
```

Output:

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/13
- 44s - loss: 0.8041 - acc: 0.6301 - val_loss: 0.7148 - val_acc: 0.7094
Epoch 2/13
- 42s - loss: 0.5119 - acc: 0.7950 - val_loss: 0.4190 - val_acc: 0.8291
Epoch 3/13
- 42s - loss: 0.4299 - acc: 0.8334 - val_loss: 0.3484 - val_acc: 0.8664
Epoch 4/13
...
```

Literatur

[1] *AlphaGo — Wikipedia, The Free Encyclopedia*. 2018. URL: https://en.wikipedia.org/wiki/AlphaGo (besucht am 19. 10. 2018).

[2] Christian Bartz u. a. "Language Identification Using Deep Convolutional Recurrent Neural Networks". In: *CoRR* abs/1708.04811 (2017). URL: http://arxiv.org/abs/1708.04811 (besucht am 17. 01. 2019).

[3] Jason Brownlee. *Supervised and Unsupervised Machine Learning Algorithms*. URL: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ (besucht am 18. 01. 2019).

[4] Kyunghyun Cho u. a. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). URL: http://arxiv.org/abs/1406.1078.

[5] Keunwoo Choi, Deokjin Joo und Juho Kim. "Kapre: On-GPU Audio Preprocessing Layers for a Quick Implementation of Deep Neural Network Models with Keras". In: *Machine Learning for Music Discovery Workshop at 34th International Conference on Machine Learning*. ICML. 2017.

[6] François Chollet. *Deep learning with Python*. Shelter Island, NY: Manning Publications Co., 2018.

[7] François Chollet u. a. *Keras*. https://keras.io. 2015.

[8] *Deep Blue (chess computer)— Wikipedia, The Free Encyclopedia*. 2018. URL: https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer) (besucht am 19. 10. 2018).

[9] J. Deng u. a. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*. 2009.

[10] Rob DiPietro. *A Friendly Introduction to Cross-Entropy Loss*. 2016. URL: https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/ (besucht am 18. 01. 2019).

[11] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. (Besucht am 17. 01. 2019).

[12] *Gradient descent — Wikipedia, The Free Encyclopedia*. 2018. URL: https://en.wikipedia.org/wiki/Gradient_descent (besucht am 11. 09. 2018).

[13] Shawn Hershey u. a. "CNN Architectures for Large-Scale Audio Classification". In: *CoRR* abs/1609.09430 (2016). URL: http://arxiv.org/abs/1609.09430 (besucht am 17. 01. 2019).

[14] Hrant Khachatrian Hrayr Harutyunyan. *Combining CNN and RNN for spoken language identification*. 2016. URL: https://yerevann.github.io/2016/06/26/combining-cnn-and-rnn-for-spoken-language-identification/ (besucht am 15. 12. 2018).

[15] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3 (2007), S. 90–95.

[16] Sergey Ioffe und Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). URL: http://arxiv.org/abs/1502.03167 (besucht am 17. 01. 2019).

[17] *Librivox*. URL: https://librivox.org/ (besucht am 24. 12. 2018).

[18] Kay M. Marc A. / Berkling. “Automatic Language Identification”. In: *MIST-1999* (1999).

[19] Martín Abadi u. a. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (besucht am 17. 01. 2019).

[20] Brian McFee u. a. *librosa/librosa: 0.6.1*. Mai 2018. URL: https://doi.org/10.5281/zenodo.1252297 (besucht am 17. 01. 2019).

[21] Grégoire Montavon. “Deep learning for spoken language identification”. In: (Jan. 2009).

[22] Michal Nilson. *Neural Networks and Deep Learning*. Dez. 2017. URL: http://neuralnetworksanddeeplearning.com/index.html (besucht am 21. 07. 2018).

[23] *NIST 2017 Language Recognition Evaluation*. URL: https://www.nist.gov/itl/iad/mig/nist-2017-language-recognition-evaluation (besucht am 24. 07. 2018).

[24] Travis Oliphant. *Guide to NumPy*. Jan. 2006. URL: http://www.numpy.org/ (besucht am 15. 01. 2019).

[25] Tomasz Oponowicz. *Spoken language dataset*. URL: https://github.com/tomasz-oponowicz/spoken_language_dataset (besucht am 24. 12. 2018).

[26] Fernando Pérez und Brian E. Granger. “IPython: a System for Interactive Scientific Computing”. In: *Computing in Science and Engineering* 9.3 (Mai 2007), S. 21–29. URL: https://ipython.org (besucht am 15. 01. 2019).

[27] Tariq Rashid. *n*. Heidelberg: O’Reilly, 2017.

[28] Mark Sandler u. a. “Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation”. In: *CoRR* abs/1801.04381 (2018). URL: http://arxiv.org/abs/1801.04381 (besucht am 17. 01. 2019).

[29] Julius O. Smith. *Mathematics of the Discrete Fourier Transform (DFT)*. online book, 2007 edition. URL: http://ccrma.stanford.edu/~jos/mdft/ (besucht am 11. 09. 2018).

[30] S. S. Stevens, J. Volkmann und E. B. Newman. “A Scale for the Measurement of the Psychological Magnitude Pitch”. In: *The Journal of the Acoustical Society of America* 8.3 (1937), S. 185–190.

[31] Yuji Tokozume, Yoshitaka Ushiku und Tatsuya Harada. “Learning from Between-class Examples for Deep Sound Recognition”. In: *CoRR* abs/1711.10282 (2017).

[32] Thomas Werkmeister Tom Herold. “Practical Applications of Multimedia Retrieval. Language Identification in Audio Files”. In: (2016). URL: https://github.com/twerkmeister/iLID/blob/master/Deep%5C%20Audio%5C%20Paper%5C%20Thomas%5C%20Werkmeister%5C%2C%5C%20Tom%5C%20Herold.pdf (besucht am 17. 01. 2019).

[33] *Voxforge*. URL: `http://www.voxforge.org/` (besucht am 24. 07. 2018).

[34] Hsiao-Wuen Hon Xuedong Huang Alex Acero. *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*. Prentice Hall PTR, 2001.

[35] *Youtube*. URL: `https://www.youtube.com/` (besucht am 24. 07. 2018).

[36] *Youtube-dl*. `https://rg3.github.io/youtube-dl/`. (Besucht am 17. 01. 2019).

Abbildungsverzeichnis