deepLID

Sprachidentifikation mit Deep Learning

Joel André

Maturaarbeit
Betreut von Beni Keller

Kantonsschule Zug
2019

Inhaltsverzeichnis

1 Einleitung

1.1 Themenwahl

Für mich war seit Anfang an klar, dass ich meine Maturaarbeit im Fach Informatik beschreiten will. Ich interessierte mich schon lange für das Fach, besuchte das Freifach Begabtenförderung-Informatik und war darum auch nicht ganz ohne Vorwissen. Informatik erlaubt wie kein anderes Fach ohne grössere Ressourcen ein konkurrenzfähiges Produkt zu gestalten. Würde ich eine Informatik-Startup lancieren, bräuchte ich dafür nur meinen Computer und einen Internet-Anschluss, nichts anderes. Das nötige Wissen findet man ohne lange Ausbildung im Internet. Auf diese Weise ermöglicht Informatik Jugendlichen wahrhaftig produktiv zu sein.

Das Teilgebiet auf die Künstliche Intelligenz zu fixieren nahm mir längere Zeit. KI hat aber mehrere Vorteile gegenüber anderen Themen.

Erstens ist es extrem aktuell. In den letzten Jahren wurden enorme Fortschritte gemacht was dazu führt, dass die Industrie der Forschung weit hinterher ist. Dank fortgeschrittener Computer-Hardware und neuen Algorithmen können Maschinen heute Aufgaben lösen die vor zehn Jahren unmöglich erschienen. Diese Aufblühende Phase wird generell KI-Sommer genannt.

Zweitens ist KI für mich faszinierend. Künstliche Intelligenz fasziniert den Menschen bereits seit langer Zeit. Das Prinzip, einer Maschine Teile unserer kognitiven Fähigkeiten beizubringen ist für viele der nächste Schritt der menschlichen Evolution. Zuletzt überlege ich mir mein Studium in Richtung Informatik/Ki zu absolvieren. Diese Arbeit ist für mich persönlich also ein Test, ob dies das richtige für mich ist.

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*[5] oder dem Go-Programm *AlphaGo*[1]. Solche Projekte bieten zwar gute Forschungsprojekte sind aber an sich keine Anwendungsfelder. Ich wollte in meiner Arbeit etwas programmieren, das tatsächlich einen realen Nutzen hat. Schlussendlich bin ich auf Sprachidentifikation gestossen. Mit zwei Muttersprachen bin ich perfekt geeignet, mein Projekt zu testen und Sprachmaterial lies sich genügend im Internet finden. Das war ausschlaggebend.

1.2 Ziel der Arbeit

Bei Sprachidentifikation geht es darum, anhand von Audio-Aufnahmen die gesprochene Sprache zu bestimmen. Eine fortgeschrittene Anwendung ist Spracherkennung: Moderne Computer und Mobiltelefone interagieren zunehmlich mit uns über unsere Stimme. Dank schneller Sprachidentifikation kann das Gerät die Grammatik der Sprache dabei zur Hilfe nehmen. Eine weitere Anwendung sind zum Beispiel Support-Anrufe. Eine Computer erkennt die Sprache und leitet den Anruf an den passenden Mitarbeiter weiter.

Klassische Methoden für die Sprachidentifikation beruhen stark auf Expertenwissen. Die Signale werden mehrfach mit komplexen Operationen bearbeitet um dann mit nicht weniger einfacheren Algorithmen ausgewertet zu werden. 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 beschränke ich mich auf 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.

1.3 Vorgehen

Ich hatte a priori der Arbeit bereits das *Neuronale Netze selbst programmieren*[12] aus dem Info-Z gelesen. Dies ermöglichte mir erst meine Fragestellung genau einzuschränken und den nötigen Aufwand abzuschätzen. Die enthaltenen Informationen waren aber noch nicht genügend um mit dem Produkt anzufangen. Deshalb recherchierte ich noch eine Zeit weiter. Besonders hilfreich war das Buch *Deep learning with Python*[4], was als meine theoretische Grundlage diente.

Parallel fing ich an das Web-Interface zu entwickeln. Das nötige Know-How hatte ich bereits grösstenteils als Vorwissen. Das früh realisierte Interface erlaubte mir mein Produkt in der Entwicklungsphase live zu testen.

Der nächste Schritt war endlich das programmieren am Produkt. In dieser Phase gab mir das Paper *Practical Applications of Multimedia Retrieval. Language Identification in Audio Files*[17] die richtige Richtung vor.

Um die Unterschiede zwischen den Sprachen zu erlernen braucht mein Programm viele Trainings-Daten. Die Beschaffung dieser grossen Menge Daten und das effiziente umgehen damit, kostete mich mehr Zeit als eingeschätzt und gab mir schlaflose Nächte.

Die Letzte Phase, das entwickeln der künstlichen Intelligenz, war am spannendsten. Die Komplexität des Themas erlaubte mir allerdings oft nur oberflächlich ein Verständnis zu entwickeln weil mir die mathematischen Grundlagen auf Universitäts-Stufe fehlten.

1.4 Aufbau

Im Kapitel **2 Deep Learning** werden die Technologien der künstlichen Intelligenz vorgestellt. Das Kapitel bildet das Herz des instruktiven Teils. Das nächste Kapitel **3 Daten** befasst sich mit den Quellen der Daten, den Methoden zur Datenbeschaffung und zur Datenbearbeitung. Anschliessend wird in Kapitel **4 Web Interface** das Interface und die verwendeten Frameworks vorgestellt. Kapitel **5 Resultate** präsentiert die entwickelten Modelle und vergleicht ihre Fehlerfreiheit. In **6 Diskussion** werden die Ergebnisse in Zusammenhang gestellt und Kapitel **7 Code** erlaubt Interessierten einen Einblick in den nötigen Code.

2 Deep learning

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

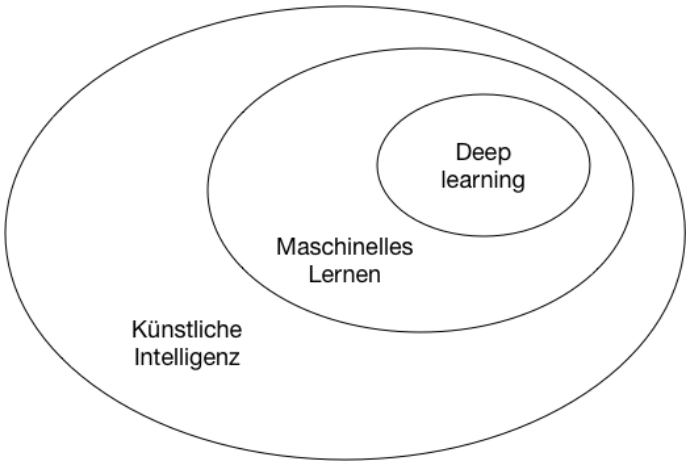

Abbildung 1: Künstliche Intelligenz, Maschinelles Lernen und Deep Learning

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.

Erste Erfolge erreichte man zum Beispiel mit Schachcomputern, die handgeschriebene Regeln befolgten. Diese Form von künstlicher Intelligenz hatte aber schnell Grenzen, da viele Prozesse schlicht zu komplex waren, um sie unter angemessenem Aufwand mit Regeln zu beschreiben. Aus dieser Idee entwickelte sich das Konzept des maschinelles Lernens.

Der Ablauf von maschinellem Lernen ist grundlegend anders als konventionelles Programmieren. Der Entwickler muss keinen Programmcode mit festen Regeln schreiben, im Gegenteil. Man liefert dem Computer Eingabe und Ausgabe, und der Computer lernt die Regeln selbst. Maschinelles Lernen entstand blühte erst in 90'er Jahren auf, wurde aber schnell zum grössten Teilgebiet der künstlichen Intelligenz.

Beim maschinellem Lernen, lernt die Software im Grunde eine nützlichere Darstellungsweise der Daten bzw. der Eingabe. Anhand dieser anderen Darstellungsweise kann der Computer die Antwort einfach erkennen. Wenn der Computer stufenweise nützlichere Repräsentationen bestimmt, kann er zunehmend komplexe Probleme in einfachere Zwischenschritten lösen. Genau das ist deep learning. Es beschreibt also mehr das Konzept von stufenweisem Lernen als eine Methode selber. Ein weit verbreitete Mehode sind allerdings *tiefe künstliche neuronalen Netzen.*[vgl. 4]

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 Verbindungen haben ein Gewicht w. Künstliche Neurone Netze haben sich aber mittlerweile so stark weiterentwickelt, dass sie

nebst der ursprünglichen Idee, nichts mehr mit der Natur zu tun haben.

Der Wert eines Knotens ist eine Funktion der Summe all er seiner eingehenden Verbindungen. Diese Funktion wird Aktivierungsfunktion σ genannt. Die Aktivierungsfunktion ist wichtig damit das Netwerk auch nicht lineare Repräsentationen lernen kann. Eine bekannte Aktivierungsfunktion ist zum Beispiel die *RELU* Funktion. [12]

$$\sigma(x) = relu(x) = \max(0, x)$$

Der Wert einer eingehenden Verbindung ist errechnet sich aus dem Produkt des Gewichts w und dem Wert des ausgehenden Knotens. Wenn man alles zusammensetzt ergibt sich für den Wert irgendeinen Knotens o mit Vorgänger Knoten h diese Formel:

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

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 sind willkürlich.

Die Aufgabe des vorgestellten Netzwerks könnte zum Beispiel sein, zwischen Hund und Katze zu unterscheiden. Die Eingaben x_1 und x_2 würden dann gewisse Merkmale des Tieres beschreiben. 1 würde heissen das Tier besitzt das Attribut und 0 das Gegenteil. Die Ausgabe o wäre dann die Wahrscheinlichkeit, dass das Tier eine Katze ist.

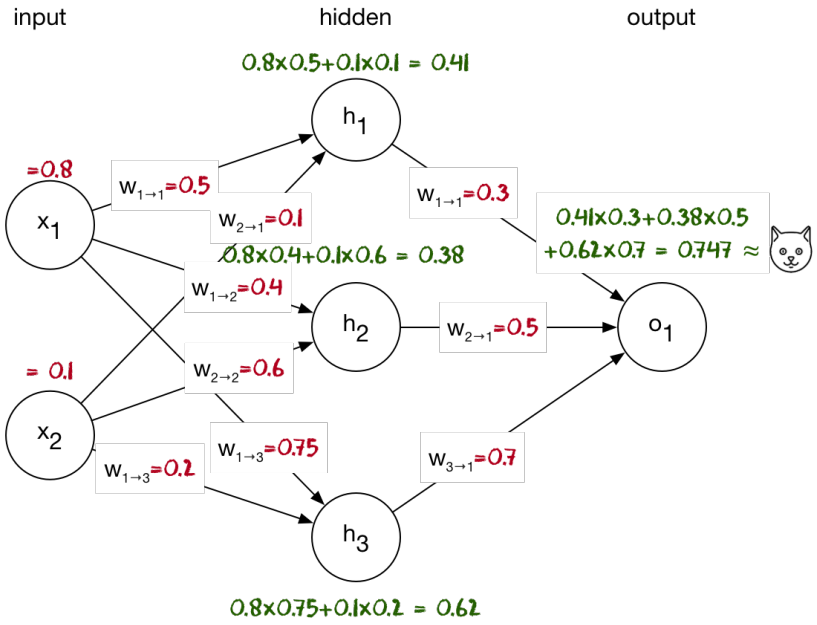

Abbildung 2: Grafische Darstellung eines künstlichen neuronalen Netzwerks anhand eines Beispiels

Die mathematischen Operationen in einem neuronalen Netwerk 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 [12] .

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. Um diese zu verbessern braucht es eine *Verlust Funktion* die uns angiebt, wie weit die Ausgabe vom korrekten Ziel entfernt ist. Wenn man in unserem Beispiel davon ausgeht, dass die Eingabe wirklich zu einer Katze gehört, wäre so etwas möglich:

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

Anhand des Verlustwerts passt das Netzwerk die Gewichte schrittweise an. Diesen Teil übernimmt der *Optimierer*. Ein einfacher Optimierer ist zum Beispiel das Gradientenverfahren [7].

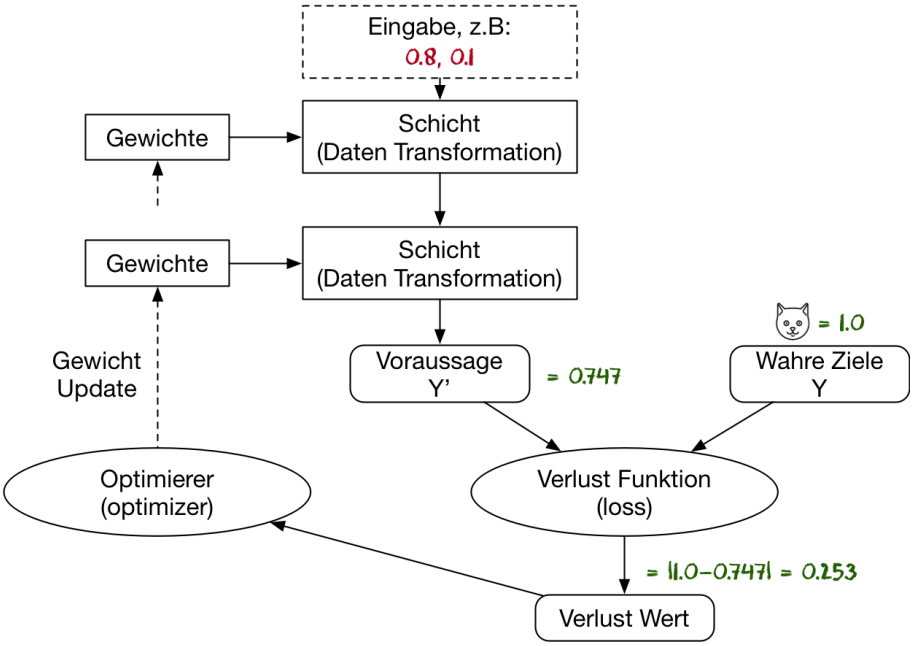

Abbildung 3: Grafische Darstellung des Lern-Prozesses anhand eines Beispiels

Da bei diesem Typ von neuronalen Netzwerken alle Knoten miteinander verbunden sind, wird es oft *dense neural network* genannt. Es ist bewiesen dass solche neuronale Netze jede Funktion abbilden können[10, Kap. 4].

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 heisst, dass ein Muster an einer bestimmten Stelle angetroffen wird, an jeder anderen Stelle ebenfalls erkannt wird. [4]

Um das zu erlauben, teilen gewisse Verbindungen das gleiche Gewicht. In Abbildung 4 (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

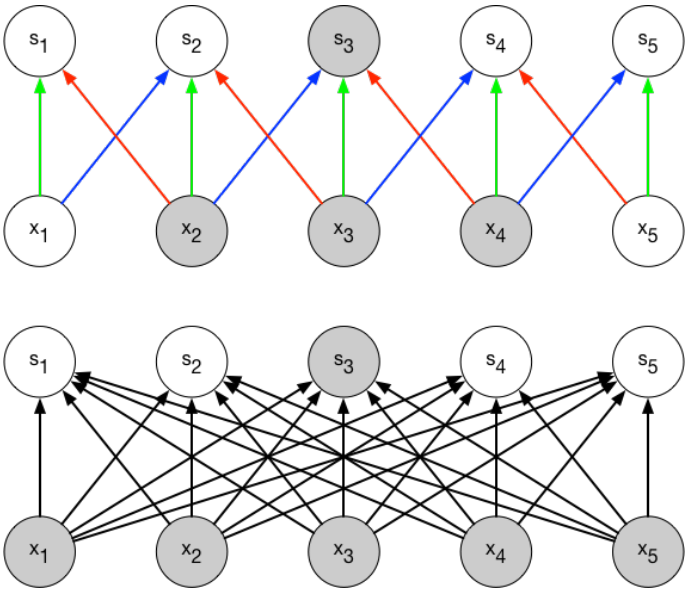

Abbildung 4: (*Oben*)1D Convolution mit *kernel* der Grösse 3. s_3 wird durch 3 inputs beeinflusst. (*Unten*) *Dense Network*. s_3 wird durch alle inputs beeinflusst.[6]

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.

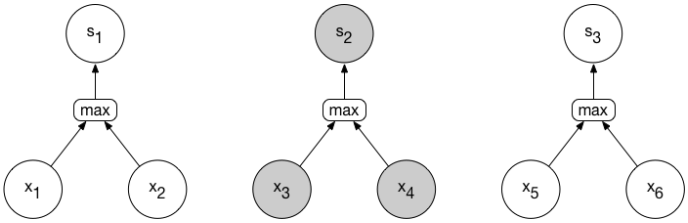

Abbildung 5: Abbildung eines 1D Max-Pooling layer. s_2 ist $\max(x3, x4)$

2.3 Recurrent Neural Networks

Eine gemeinsame Eigenschaft von allen *dense neural networks* und CNN's ist, dass sie keinen Speicher haben. Bei jedem Vowärtspass berechnet das Netzwerk alles von neuem ohne Erinnerungen an vorherige Durchläufe. 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 6). Die RNN Zelle berechnet dann die nächste Ausgabe sowohl aus der neuen Eingabe wie auch mit den Erinnerungen der letzten Ausgabe. [4]

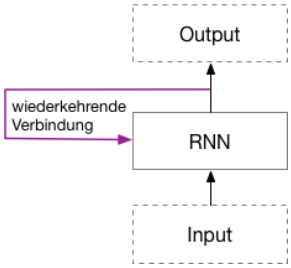

Abbildung 6: RNN mit Schlaufe

Der Vorgang lässt sich grafisch über die Zeit aufgerollt darstellen (Abbildung 7). 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.

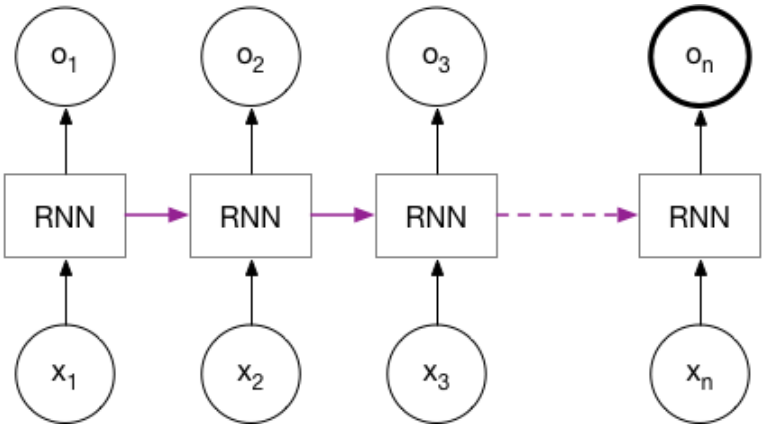

Abbildung 7: RNN aufgerollt über die Zeit

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 nach zeitlich zerteilt und stückweise dem Netzwerk gefüttert. Eine fortgeschrittene Implementation von RNN Zellen ist unter anderem die *Long short-tem memory* (LSTM) Zelle [8]. Durch das einführen von unterschiedlichen wiederkehrenden Verbindungen wird verhindert, dass ältere Signale langsam verschwinden [4].

2.4 Generational Adversial Neural Networks

3 Daten

3.1 Daten Auswahl

Es gibt keine frei zugänglichen umfassenden Datasets für Sprachidentifikations-Aufgaben. Datensets wie das *NIST Language Recognition Evaluation*[11] sind nur unter teuren Gebühren zugänglich. Wie verwandte Arbeiten empfehlen [17], wird darum ein eigenes Datenset zusammengestellt. Es werden gleichmässig Daten zu den Sprachen Deutsch, Englisch, und Französisch gesammelt. Die Daten stammen einerseits vom *Voxforge*[18] Datenset und von *Youtube*[20].

Voxforge ist ein open-source Datenset für die 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 es 190h. 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.

Youtube dient als Quelle für natürlichere Sprache. Es werden angesehene Nachrichtenkanäle wie CNN, ARD, etc. verwendet. Die Sendungen haben den Vorteil, dass oft fremde Gäste eingeladen werden, was zu einer grossen Variation der Sprecher führt. Ausserdem kommen oft natürliche Hintergrundgeräusche dazu. Zum Schluss ist die Datenmenge die hier erhältlich ist, praktisch unbeschränkt gross.

Sprache	Voxforge	Youtube
Französisch	20h	50h
Deutsch	22h	60h
Englisch	23h	200h

(a) Verteilung der Datenmenge

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

(b) Youtube Kanäle

Tabelle 1: Daten Auswahl

3.2 Daten Splitt

Zu grosse Datenmengen sind für das Training problematisch. Sie sind technisch aufwendig und sehr langsam in der Verarbeitung. Zudem ist es immer das Ziel eines Algorithmus, mit möglichst wenigen Daten auszukommen. Darum wird das Datenset auf 100'000 Stücke reduziert, was ungefähr 139h insgesamt ausmacht. Die Daten stammen zufällig zu gleichen Teilen sowohl aus Youtube wie von Voxforge. Die einzelnen Sprachen sind gleichgewichtig repräsentiert.

Das Datenset wird wiederum in *Trainingset*, *Validationset*, und *Testset* aufgeteilt zu den Anteilen 80%, 10% und 10%. Das *Trainingset* wird, wie der Name preisgibt, für das Training verwendet. Das *Validationset* wird verwendet um 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. Zuletzt wird das *Testset* in Betracht gezogen, um die Leistung auf gänzlich neue Daten zu erforschen.

3.3 Preprocessing

Sprache besteht aus Wörtern und Wörter sind grundsätzlich eine Abfolge von Lauten. Verschiedene Sprachen unterscheiden sich 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 also 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 8 *(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* [15]. Das Neuronale Netz müsste dieses Verfahren erlernen um dann aus den Lauten die Sprache erkennen zu können. Allerdings ist die Fourrier-Transformation sehr komplex und fordert den Computer darum viel. Um dem Algorithmus die Aufgabe zu erleichtern kann man ihm darum als Eingabe die berechneten Laute geben anstatt der rohen Schallwelle.

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. 4]

3.3.1 Spektrogramme

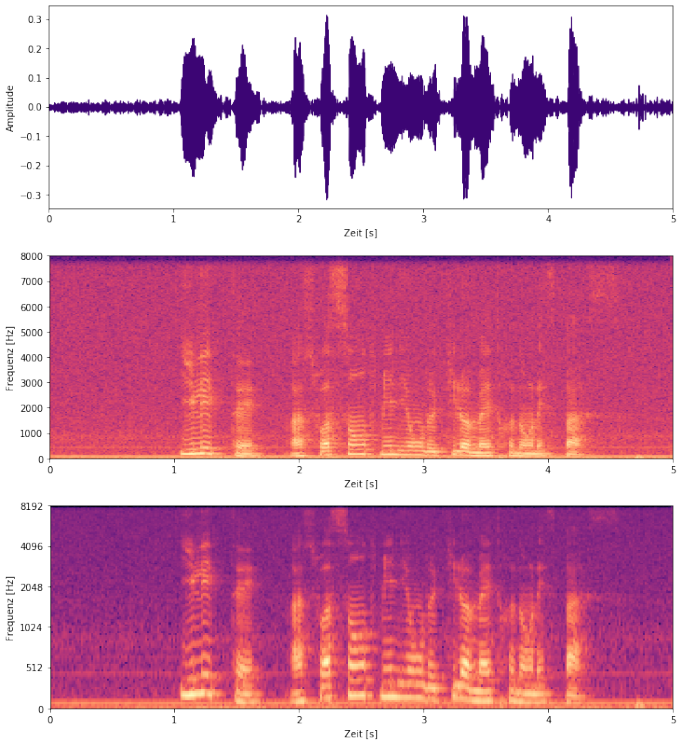

Abbildung 8: Audio-Preprocessing: *(oben)* Rohe Schallwelle, *(mitte)* Dezibel-Spectrogramm, *(unten)* Mel Dezibel-Spectrogramm

Spektrogramme sind grafische Darstellungen eines Hörsignals nach der Anwendung der Fourrier-Transformation[15, 'Spectrograms'], ersichtlich in Abbildung 8 *(Mitte)*. Frequenzen über 10kHz können abgeschnitten werden, da menschliche Sprache übermässig darunter abläuft[16]. Im Spektrogramm lassen sich von Auge Muster erkennen und unterscheiden. Der visuelle Charakter der Spektrogramme erlaubt ausserdem das verwenden von *Convolution Neural Networks*.

3.3.2 Mel Filtering

Spektrogramme haben relativ viele Datenpunkte und sind deshalb recht aufwändig verarbeiten. Ein weiter *preprocessing* Schritt wird deshalb oft angewendet: *Mel Filtering*[19]. Die Frequenzen werden dabei in grössere Eimer gepackt. Unter 1kHz sind die Eimer linear verteilt und darüber logarithmisch, siehe Abbildung 8 *(unten)*. Das Modell entspricht unserer Hörfähigkeit, die recht präzise unter 1kHz arbeitet, höhere Frequenzen aber schlecht unterscheiden kann[16].

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*[3] 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 Neuralen Netzwerk.

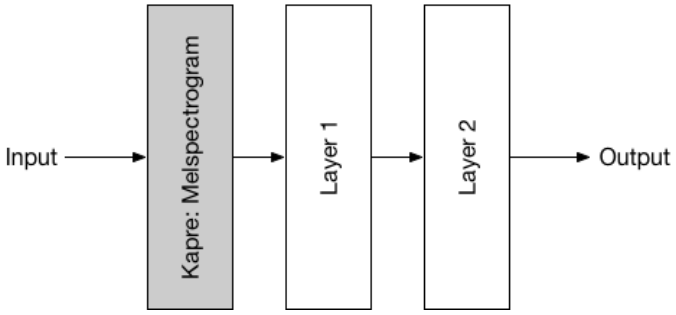

Abbildung 9: *Kapre*[3] als Schicht

4 Web Interface

Es war seit Anfang an klar, dass ein Web-Interface gestaltet werden sollte. Das Web-Interface ist ideal um die Leistung der Software hautnah zu erleben. Man kann mit jedem internetfähigen Gerät (mit Mikrofon) darauf zugreifen.

Die Idee ist einfach. Man nimmt direkt im Browser eine kurze Audioaufnahme auf, lädt diese hoch zum Server, und dieser antwortet mit der geschätzten Sprache. Bevor man die Aufnahme hochlädt kann man sie optional selbst abspielen.

4.1 Frontend

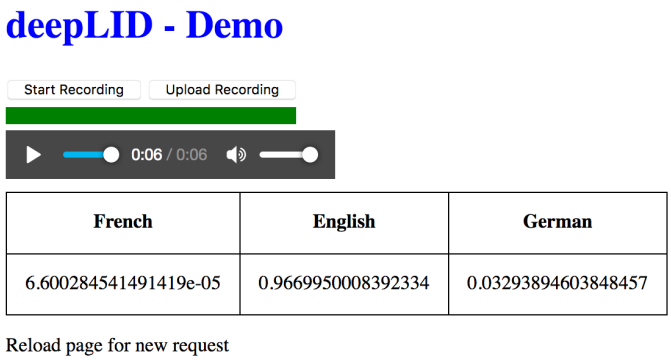

Abbildung 10: Web-Interface Screenshot

Das Design ist in einfachem *HTML* und *CSS* geschrieben, die Logik mit *Javascript*. Um im Browser Audioaufnahmen nehmen zu können, wird *RecordRTC*[9] verwendet. *RecordRTC* ist eine unter der *MIT* Lizenz veröffentlichte *Javascript* Bibliothek für Medienaufnahmen. Sie ist kompatibel mit vielen modernen Browsern und Geräten.

Nach der Aufnahme wird die Datei über AJAX POST an den Server verschickt. Der Server antwortet mit einem XML-Snippet der Resultate. Dieses wird wiederum direkt in die Webseite eingebaut.

4.2 Backend

Der Webserver läuft in der Programmiersprache *Python*. Als Framework wird mit *Flask*[13] verwendet. Da sowohl der *Flask* wie auch *Keras* mit Python laufen, kann der Webserver das trainierte Modell ohne zusätzliche Hürden verwenden. Der Code findet sich unter 7.1

5 Resultate

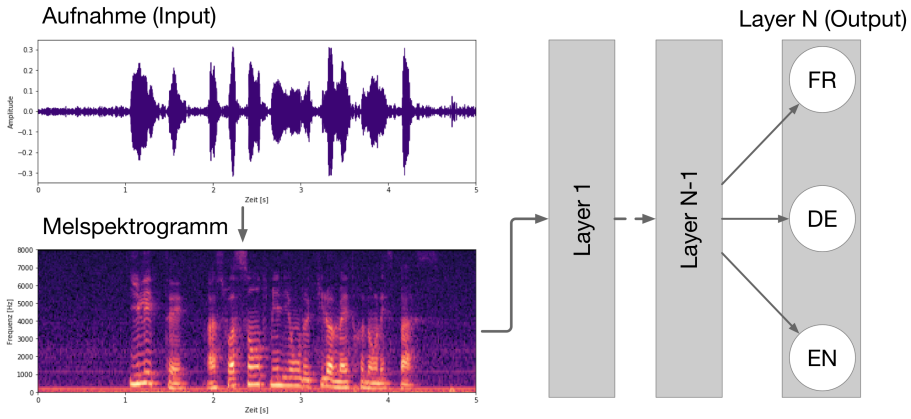

Abbildung 11: Allgemeiner Aufbau

5.1 Modelle

5.1.1 CNN

Convolution: channels = 64 size = 3x3 activation = *relu* MaxPooling: size = 2x2	Convolution: channels = 64 size = 3x3 activation = *relu* MaxPooling: size = 2x2	Convolution: channels =128 size = 3x3 activation = *relu* MaxPooling: size = 2x2	Fully Connected: units = 512 activation = *relu* dropout = 30%	Fully Connected: units = 3 activation = *softmax*

Abbildung 12: CNN Architektur

basiert auf [17]

5.1.2 InceptionV3-CNN

5.1.3 CRNN

basiert auf [2]

5.1.4 GAN

basiert auf [14]

5.2 Auswertung

5.2.1 Auf Test-Datenset

5.2.2 Auf Beta-Datenset

Netz	Architektur	Accuracy	Loss
CNN	3 Conv	90%	0.02
CRNN	4 Conv + LSTM	94%	0.015
CNN	28 Conv + Dense	87%	0.03

Tabelle 2: Top Accuracy

6 Diskussion

6.1 Frühere Arbeiten

6.2 Ausbaumöglichkeiten

6.2.1 Data Augmentation

7 Anhang

7.1 Webserver Code

```
@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        request_file = request.files['file']
        if request_file:
            # Virtuelle Datei generieren
            tmp_file = io.BytesIO(request_file.stream.read())
            # Abfrage an Modell
            pred = predict(tmp_file)
            # HTML als Antowrt senden
            return render_template('prediction.html',
                                   french_prob=str(pred[0]),
                                   english_prob=str(pred[1]),
                                   german_prob=str(pred[2]))
    else:
        # Web-Interface Standard Seite verschicken
        return render_template('site.html')
```

7.2 Beispiel-Modell Code

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

# 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 7500 samples, validate on 3750 samples
# Epoch 1/9
# 7500/7500 [==============================] - 17s 2ms/step -
    loss: 1.0368 - acc: 0.4888 - val_loss: 1.0159 - val_acc:
    0.5152
# Epoch 2/9
# 7500/7500 [==============================] - 13s 2ms/step -
    loss: 0.8200 - acc: 0.6319 - val_loss: 1.1726 - val_acc:
    0.4509
# ...
```

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.

[3] 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.

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

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

[6] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

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

[8] Sepp Hochreiter und Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), S. 1735–1780. ISSN: 0899-7667.

[9] Muaz Khan. *WebRTC*. URL: https://github.com/muaz-khan/RecordRTC (besucht am 23. 07. 2018).

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

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

[12] Tariq Rashid. *n*. Heidelberg: O'Reilly, 2017.

[13] Armin Ronacher. *Flask*. URL: http://flask.pocoo.org/ (besucht am 23. 07. 2018).

[14] Peng Shen u. a. "Conditional Generative Adversarial Nets Classifier for Spoken Language Identification". In: *INTERSPEECH*. 2017.

[15] 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).

[16] 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.

[17] 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%20Audio%20Paper%20Thomas%20Werkmeister%2C%20Tom%20Herold.pdf.

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

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

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

Abbildungsverzeichnis