HAW HAMBURG

BACHELORTHESIS
Paul Duy An Nguyen

Keyword-spotting für eingebettete Systeme auf Basis eines RNN-Transducers

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Paul Duy An Nguyen

Keyword-spotting für eingebettete Systeme auf Basis eines RNN-Transducers

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Technische Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Andreas Meisel
Zweitgutachter: Prof. Dr. Michael Neitzke

Eingereicht am: 30. Dezember 2021

Paul Duy An Nguyen

Thema der Arbeit

Keyword-spotting für eingebettete Systeme auf Basis eines RNN-Transducers

Stichworte

ML, RNN, RNN-Transducer, LSTM, Keyword-Spotting, Embedded System

Kurzzusammenfassung

Die Signifikanz von Robotern ist in der heutigen Zeit unverzichtbar. Es ist weiterhin ein Problem, Roboter mit der menschlichen Sprache zu steuern. Das Problem kann als Keyword-Spotting Problem modeliert werden, die die Aufgabe hat, bestimmte Sprachbefehle zu erkennen. Neuronale Netzwerke werden immer häufiger im Bereich der Sprachverarbeitung verwendet, welche solche Probleme mit hoher Prezision zu lösen kann. Im Bereich von Embedded-Systemen sind solche Anwendungen limitiert, da neuronale Netzwerke generell hohe Resourcenanforderungen besitzen. Diese Arbeit evaluiert die Anwendbarkeit der RNN-Tranducer (RNN-T) Basisarchitektur als ein Keyword-Spotting System mit Embedded-Systemanforderungen. Die Experimente zeigen, dass sich mit limitierten und nicht optimalen Lerndaten das RNN-T nicht zum einem nutzbarem Optimum konvergiert. Potentielle Gründe werden hinsich der RNN-T Basisarchitektur Auswahl geschildert.

Paul Duy An Nguyen

Title of Thesis

Keywords

ML, RNN, RNN-Transducer, LSTM, Keyword-Spotting, Embedded System

Abstract

The significance of todays robots is greater than ever and society will be unthinkable without it in the future. Even though an active issue in this space, is how we control the robot without our voice. This problem can be categorized in the area of keyword-spotting systems. Today's neural networks can perform with the highest precision in the area of automatic speech-recognition because of recent advancements in this area. The usecase of neural networks is still limited to non-embedded platforms because usual neural networks have high resource utilization. The usability of an RNN-Transducer (RNN-T) as a keyword-spotting-system (KWS) is evaluated with embedded-systems requirements. The experiments show limited results in terms of convergence, probably because of limited and non-optimal training data. Potential reasons are explored as to why an RNN-T might not be an optimal choice as a base for a keyword-spotting-system.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

DNN Deep-Neural Network.

ICC Informatik Compute Cloud.

KWS Keyword-spotting System.

LSTM Long Short-Term Memory.

MFCC Mel-frequency cepstral coefficients.

RNN Recurrent Neural Network.

RNN-T RNN-Transducer.

SGD Stochastic Gradient Descent.

WER Word Error Rate.

1 Einleitung

1.1 Motivation

Neue Spracherkennungssystem, basierend auf Neuronalen Netzwerken, zeigen erhöhte Erkennungsraten und Qualität als herkömmliche Spracherkennungssystemen (z.B. mit DSP und Hidden Markov Modellen). Da generelle Spracherkennungssysteme basierend auf NNs entweder sehr viel Resourcen (Speicher, Rechenleistung, usw) benötigen oder bei der Erkennungsrate suboptimal sind, kann der Fokus auf bestimmte "Keywörter"limitiert werden. Keyword-spotting ist eine Unterkategorie der Spracherkennung, welches versucht nur bestimmte Wörter aus einer Eingabesequenz (z.B. Audio) zu erkennen. Im Bereich von Embedded-Systemen, wie zum Beispiel eines Roboters, können solche "Keyword-spottingsysteme genutzt werden um bestimmte Sprachbefehle zu erkennen und zu interpretieren. Embedded-Systeme besitzen im üblichen aber nur begrenzte Resourcen, welches sich auf das Spracherkennungssystem rücktwirkt. Traditionellen Spracherkennungssystemen, die aus mehrern Schichten bestehen, (wie z.B. audio, preprocessing, feature extraction, prediction, decision making) können vielfältige Algorithmen verwenden und müssen getrennt trainiert werden. End-to-end learning ist ein Bereich der Tiefen Neuoralen Netzwerken (Deep-Neural Network (DNN)s), welches mehrer komplexe Lernsysteme in einem System vereint, und dadurch Pipeline-Systeme vermeidet.

In [6] haben die Authoren gezeigt, dass ein RNN-Transducer Model als Spracherkennungssystem mit embedded Anforderungen genutzt werden kann. Diese Arbeit untersucht, die Möglichkeit eine ähnliche RNN-Transducer Architektur als Keyword-spotting System (KWS) zu verwenden.

1.2 Zielsetzung und Struktur

In dieser Arbeit wird die Nutzbarkeit eines RNN-Transducers für ein Keyword-spotting System untersucht und evaluiert.

In den ersten Kapiteln, werden die Grundlagen von neuronalen Netzwerken und der RNN-Transducer (RNN-T) Basisarchitektur erläutert. Desweiteren, werden Grundlagen und Prior-Art im Bereich Keyword-spotting gegeben. Folgend, wird die zu evaluierende Architektur beschrieben. Anschließend, wird die Methodik, wie die Trainingsdaten aufbereitet werden, beschrieben. Zum Schluss werden die Ergebnisse evaluiert und deren Probleme bzw. mögliche Lösungswege anhand der Word Error Rate (WER)s diskutiert.

2 Rekurrente Neuronale Netzwerke

Ein Ziel der Spracherkennung ist es Sequenzen von Wörtern oder Phoneme zu erkennen. Da Wörter oder Phoneme in mehreren Zeitschritten einer Eingabesequenz vorkommen können, ist es wichtig die Eingabeinformationen und die zughörige Zeitinformationen mit in die Spracherkennung zu verarbeiten.

Mit einem einfachen feed-forward Deep Neural Network (DNN), können die Zeitinformationenn nur mit verwendet werden, wenn gleichzeitig mehrere Zeitfenster in das DNN gefüttert werden. Dieser Ansatz ist sehr limitierend im Vergleich zu RNNs.

2.1 RNN

Rekurrente Neuronale Netzwerke sind neuronale Netzwerke, die eine bestimmte Rückkopplung zwischen den Schichten besitzen, dadurch kann der zeitliche Kontext im Netzwerk abgebildet werden. Die gelernten Informationen einer Eingabe sind in den Aktivierungen h zu finden. In der nächsten Eingabe der Eingabesequenz wird jeweils eine gewichtete Form der vorherigen Aktivierungsinformationen verwendet. Informationen, die zeitlich zurückliegen, können über diesen Mechanismus beibehalten werden. Eine typische Recurrent Neural Network (RNN) Schicht (siehe. 2.1) wird wie folgt abgebildet:

$$h_t = f(W_{xh} \cdot x_t + W_{hh} \cdot h_{t-1} + b_h) \tag{2.1}$$

wobei f eine nicht lineare Aktivierungsfunktion ist.

Für jeden Zeitschritt wird ein neuer *hidden state* h_t aus den folgenden zwei Komponenten erzeugt: Der Eingabevektor x_t, welcher mit dem Gewichtungsmatrix W_{xh} gewichtet wird, und dem vorherigem *hidden state* h_{t-1}, gewichtet mit W_{hh}. Die zweite Komponente

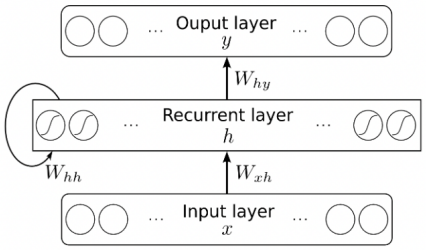

Abbildung 2.1: Ein Rekurrentes neuronales Netz mit einem *hidden layer* und Rückkopplung, um auch vorherige Informationen mitaufzunehmen.
Quelle: Wake-Up Word Detection using LSTM-Neural-Networks, Clemens Vayda, Wolfgang Hrauda

trägt zur Erhaltung der zeitlichen Informationen bei. Zusammen, können die zeitlichen Abhängigkeiten der Eingabesequenz abgebildet und gespeichert werden.

Da in RNNs, der Fehler bei jedem Zeitschritt abhängig von dem vorherigen Zeitschritt ist, muss die Backpropagation auch rückwirkend über die Zeit erfolgen.

Klassische RNNs haben jedoch das Problem, dass bei der Backpropagation während des Tranings, der Fehler entweder "explodiert" (exploding gradient) oder "verschwindet" (vanishing gradient [7]). Nachteile sind die sehr hohen Trainingszeiten und limitierte Erhaltung von zeitlichen Informationen. In [8] haben Hochreiter und Schmidhuver, "Long Short-Term Memory", eine neuartige RNN Architektur vorgestellt, die diese Probleme angehen.

2.2 LSTM

Die LSTM-Architektur [8] erlaubt es neben der Aufnahme von Informationen, auch das Verwerfen von temporalen Informationen, so dass nur kontext-relevante Informationen betrachtet werden, wodurch sich die Aufnahme von Langzeitinformationen verbessert. Die Kernidee der LSTM-Architektur (long short-term memory) ist das Setzten der Gewichtungsmatrix auf 1. Dadurch kann der Fehler durch das Netzwerk ohne Modifikation zurück-propagieren, welches das *vanishing-gradient* Problem löst. Folglich, steigt der Berechnungsaufwands und Speicherbedarf einer LSTM-Zelle. Neben dem *hidden state* h_t, wird auch der cell-state Vektor C_t mit in die nächste LSTM-Zelle weitergegeben. Eine LSTM-Zelle besitzt intern vier Schichten (*output gates*), die steuern welche Informationen gespeichert, vergessen und für die Berechnung des *hidden-state* h_t verwendet werden (Die

Basis-RNN Architektur 2.1 besitzt im Vergleich nur eine Schicht). Es können dadurch Langzeitabhängigkeiten gespeichert werden, im Vergleich zur Basis RNN-Architektur.

In 2.2 werden LSTM-Zellen über mehrere Zeitschritte dargestellt. Die folgenden Gleichungen beschreiben die Funktionsweise einer LSTM-Zelle:

$$i_t = \sigma(W_{xi} \cdot x_t + W_{hi} \cdot h_{t-1} + W_{c_i} \cdot c_{t-1} + b_i) \tag{2.2}$$

$$f_t = \sigma(W_{xf} \cdot x_t + W_{hf} \cdot h_{t-1} + W_{c_f} \cdot c_{t-1} + b_f) \tag{2.3}$$

$$o_t = \sigma(W_{xo} \cdot x_t + W_{ho} \cdot h_{t-1} + W_{c_o} \cdot c_{t-1} + b_o) \tag{2.4}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tanh(W_{xc} \cdot x_t + W_{hc} \cdot h_{t-1} + b_c) \tag{2.5}$$

$$h_t = o_t \cdot \tanh c_t \tag{2.6}$$

x_i ist der Eingabevektor und ist in der Regel die Ausgabe der Aktivierungsfunktions der letzten Schicht. h_{t-1} ist die Zellenausgabe vom vorherigen Zeitschritt, welches hier als weitere Eingabe verwendet wird. Die vier Schichten haben jeweils eine Gewichtungsmatrix W und Bias b. Die Eingabevariablen werden über die Gewichtungsmatrix W gewichtet.

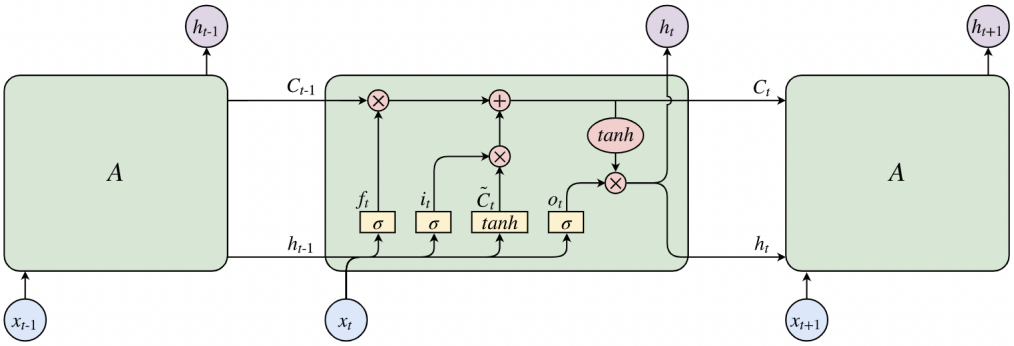

Abbildung 2.2: LSTM-Zellen über mehrere Zeitschritten und deren internen Verbindungen.
Quelle: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

2.3 RNN-Transducer

Ein RNN-Transducer [5] ist ein *end-to-end* und *sequence-to-sequence* Model, d.h. ein Model wird mit einer Sequenz von Informationen (*Features* wie z.B. Wörter in der deutschen Sprache) gefüttert und produziert, ohne weitere Nachbearbeitung und Verarbeitungspipelines, eine andere transformierte Sequenz von Informationen (*Features* wie z.B. Wörter in der englischen Sprache). Bei Transformationensproblemen, wie der Spracherkennung oder Übersetzung von natürlichen Sprachen, eignet sich ein RNN-T. Ein inherentes Problem bei der Sequenzentransformation (*transduction*), ist das Lernen wie Eingabe- und Ausgabesequenz korrelieren (*sequential distortions*). Basis RNN Modelle, sind in der Lage *sequence-to-sequence* Probleme zu modellieren, jedoch müssen Ausrichtungsinformationen, d.h. wie Elemente aus der Eingabesequenz mit Elemente aus der Ausgabesequenz korrelieren, mitgegeben werden. Eine automatische Extrahierung solcher Ausrichtungsinformationen ist nicht trivial. Ein weiteres Problem existiert, wenn die Transformation keine monotone Abbildung von Eingabe und Ausgabe erzeugt (z.B. Ausgabesequenz ist großer oder kleiner als die Eingabesequenz). Die RNN-Tranducer Architektur basiert auf RNNs und addresiert die genanten Problem. Es ist somit in der Lage jegliche Eingabesequenz in eine beliebige Ausgabesequenz zu transformieren.

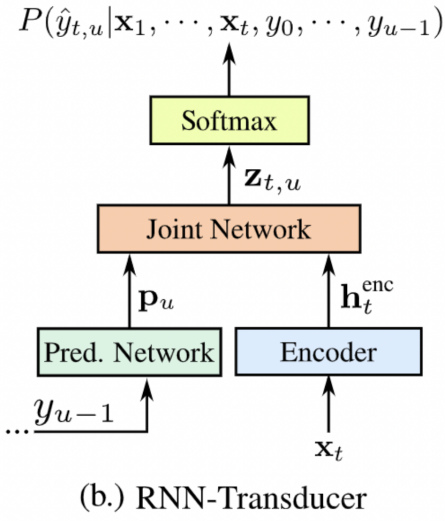

Abbildung 2.3: RNN-Tranducer Model Übersicht.
Quelle: An Overview of End-to-End Automatic Speech Recognition - Wang, Dong and Wang, Xiaodong and Lv, Shaohe [12]

Eine Übersicht des RNN-Tranducer Modells ist in 2.3 abgebildet. Die Grundkomponenten sind ein *Encoder*, *Prediction Network* und ein *Joint Network*. Folgende Variablen werden in der Grafik verwendet: $x = (x_1, x_2, ..., x_T),\ x \in X^*$ beschreibt die Eingabesequenz mit einer beliebigen Länge T, wobei X den Eingabenraum (Die Eingabe-Labels) kennzeichnet. $y = (y_1, y_2, ..., y_U),\ y \in Y^*$ beschreibt die Eingabesequenz mit einer beliebigen Länge U, wobei Y den Ausgaberaum (Die Ausgabe-Labels) kennzeichnet. Die Ausgabe des Netzwerkes ist eine bedingte Wahrscheinlichkeitsverteilung: $P(a \in \bar{y}^*),\ \bar{y} = y \cup \varnothing$. $\varnothing$ beschreibt die *null* Ausgabe, welches wichtig für die Ausrichtung (im Paper auch *alignment*) der Eingabesequenz und Ausgabesequenz ist, da es schlaggeben die Ausrichtung beschreibt. $\mathbf{p}_u$ und $\mathbf{h}_t^{enc}$ beschreiben den *hidden state* des *Encoder*s und Pred. Networks. In der einfachen Konfiguration sind *Encoder* und Pred. Network unidirektionale und gestapelte Schichten bestehend aus *LSTM*-Zellen (siehe 2.4), die letzlich die Wahrscheinlichkeitsverteilung P definiert. Das *Pred. Network* versucht aus der vorherigen Ausgabe den nächsten Ausgabezustand $\mathbf{p}_u$ vorauszubestimmen, daher wirkt die Komponente autoregressive. Das *Join Network* ist ein einfacher *feed-forward* Netzwerk, welches $\mathbf{p}_u$ und die gelernten *hidden features* $\mathbf{h}_t^{enc}$ aus dem *Encoder*, kombiniert und eine *softmax*/Wahrscheinlichkeitsverteilung zurückgibt.

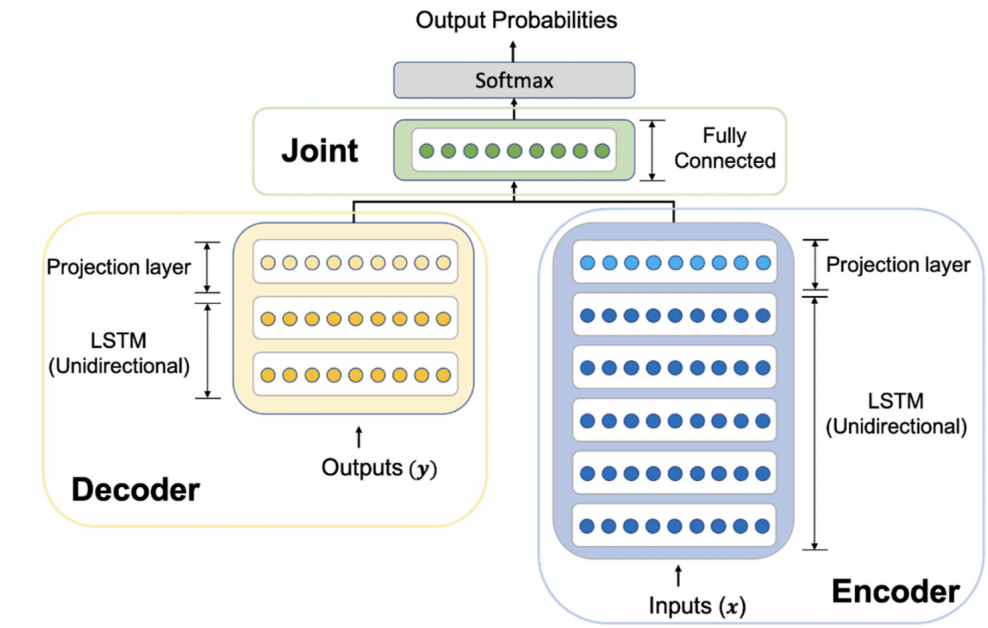

Abbildung 2.4: RNN-Tranducer Model. Encoder und Decoder (*Pred. Network*) bestehen aus gestapelten unidirektionalen LSTM Schichten.
Quelle: An Effective Learning Method for Automatic Speech Recognition in Korean CI Patients' Speech [10]

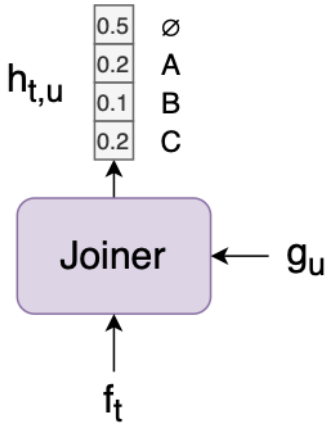

Abbildung 2.5: Beispiel Ausgabe eines RNN-Tranducers. Mit drei Labels (A, B, C) und dem *null* Element.
Quelle: https://lorenlugosch.github.io/posts/2020/11/transducer/

3 Keyword Spotting System (KWS)

Diese Arbeit beschäftigt sich mit der Evaluierung eines RNN-Tranducers als Basismodell für ein limitiertes Keyword-Spotting System (limitiert auf einer fester Liste von definierten Schlüsselwörtern; siehe A.1). In [6] haben die Authoren ein RNN-T Modell als Spracherkennungs System mit embedded Anforderungen demonstiert, daher wird in der Arbeit ein RNN-T als Grundlage für ein KWS verwendet.

3.1 Grundlagen

Typische Keyword-Spotting Systeme (KWS) bestehen aus einem *feature extractor* und einem Klassifikator basierend auf ein neuronales Netz (siehe. 3.1).

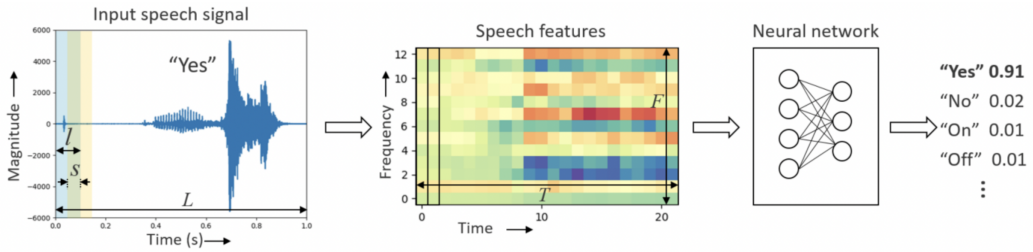

Abbildung 3.1: Typisches Keyword-Spotting System bestehend aus dem Eingabesignal in der Zeitdomaine, der *feature extraction* und dem neuronalem Netzwerks als Klassifikator.
Quelle: Hello Edge: Keyword Spotting on Microcontrollers - [14]

Zunächst wird das Sprachsignal (als Eingabe) mit der Länge L in überlappende Fenster (*overlapping frames*) mit der Länge l und dem Stride s unterteilt. Die Unterteilung resultiert in $T = \frac{L-l}{s} + 1$ Fenster (*frames*). Für jedes *frame* werden F Sprachfeatures (*speech features*) extrahiert, sodass TxF *features* für das Eingabesignal enstehen. Die Mel-Frequenz-Cepstrum-Koeffizienten (Mel-frequency cepstral coefficients (MFCC)) werden oft zur automatischen Spracherkennung zusammen mit Neuronalen Netzwerken

benutzt [3]. MFCCs beschreiben das Eingabesignal in der Zeitdomäne als kompakte Darstellung des Frequenzspektrums. In 3.1 bestehen die extrahierten *features* aus MFCCs, welche dann im Klassifikator zu einer Wahrscheinlichkeitsverteilung der Keywörter verarbeitet werden. Eine Problemematik solcher Systeme ist die konstante Eingabelänge L. Eine zu kleine oder große Länge kann zu starken Reduktionen der Erkennungsrate führen. Es müssen unteranderem die Trainingsdaten speziell aufbereitet werden, da sonst Ausrichtungsprobleme (*alignment issues*) auftreten können. Eine kontinuiertliche (*streaming*) Signalverarbeitung ist daher nicht effektiv möglich. Andere Spracherkennungssysteme basierend auf *Hidden Markov Models* (HMM) [13] existieren und erzeugen vernünftige Erkennungsraten, sind jedoch sehr schwierig zu trainieren und sind sehr rechenaufwendig. Andere Verfahren, die rein Rekurrente Neuronale Netwerke verwenden erreichen höhere Erkennungsraten als HMM, leiden jedoch an hohen Berechnungszeiten und weiteren Aufwand Trainingsdaten aufzubereiten [4].

3.2 Ansatz mit RNN-Tranducer

Keyword-Spotting Systeme können auch als *sequence to sequence* Systeme modelliert werden. Im RNN-Tranducer Kontext bieten sich Vorteile im Bereich des Trainings an, da Trainingsdaten mit wenig Aufwand aufbereitet werden können. Mit dem obigen RNN-Transducer Model 2.3 kann auch eine kontinuiertliche Ausführung (*streaming*) des Models garantiert werden ohne weiteres *windowing* des Eingabesignales.

4 Methodik

In diesem Abschnitt wird das Modell-Design, die Aufbereitung der Trainingsdaten und die Trainingsprozedur beschrieben.

4.1 Framework

Die Python Programmiersprache, kombiniert mit der TensorFlow Framework von Google, wird für die gesamte Implementierung dieser Arbeit verwendet. In TensorFlow werden Mathematische Operationen in einem *computational graph* dargestellt, welches sich auch in serialisierbarer Form abgespeichern lässt. Am *computational graph* können Verfahren wie das automatische Differenzieren (*auto-differentiation*) und weitere Graphenoptimierungen ausgeführt werden. Im Rahmen dieser Arbeit werden erweiterte Graphenoptimierung verwendet (siehe 4.4) um das tranierte Model "embedded system"s tauglich zu machen. Genauer wird Tensorflow-Lite genutzt um das tranierte Model in ein TensorFlow-Lite Modell zu konvertieren. TensorFlow-Lite Modelle können starke Reduktionen der Modellgröße und reduzierte Evaluierungszeiten erreichen, im Vergleich zu konventionellen TensorFlow Modellen.

4.2 Datensatz und Datenaufbereitung

Das KWS in dieser Arbeit limitiert sich auf eine kleine Schlüsselwortauswahl, bestehend aus ca. 26 englischen Befehlswörtern (siehe A.1).

Typische *sequence-to-sequence* Modelle benötigen im Vergleich zu anderen Verfahren mehr Trainingsdaten, da im Kontext der automatischen Spracherkennung, Akoustikmodell und Sprachmodell in kombinierter Form zusammen trainiert werden müssen. Spezialisierte Keyword Datensätze, wie *Speech Command Dataset* [11] von Google, sind daher

nicht optimal, da nur sehr kurze Audiosignale von den einzelnen Keywörtern im Datensatz enthalten sind. Es kann daher sinvoll sein größere Datensätze zu nutzen, die im Bereich der automatischen Spracherkennung (*speech-to-text*) verwendet werden (d.h. längere gesprochene Audiosignale mit Transkript).

Für die Evaluierung wird Mozilla's *Common Voice* Datensatz (Version 5.1) in der Englischen Variante verwendet. Der Datensatz besteht aus ca. 1450 Stunden verifiziertem und Englisch gesprochenen Audiosignalen mit Transkript. In komprimierter Form umfasst der Datensatz ein Datengröße von ca. 50GB.

Der Datenaufbereitungsprozess besteht aus der *Feature Extraction* von den Audiosignalen und der Aufbereitung der zugehörigen Text-Transkripte (auch Labels genannt).

Für die Anwendung eines limitierten Keyword-Spotting-Systems wäre ein solcher Datensatz nicht direkt nutzbar, daher werden die Transscripte aus dem Datensatz wie folgt verarbeitet:

1. Nur die Einträge aus dem Datensatz, die Schlüsselworte enthalten, werden beibehalten. Die restlichen Einträge werden ignoriert:

```
This is a cat. << DISCARD, da kein Keyword.
Please open the car and put the cat inside. << KEEP, da
    "open" ein Keyword ist.
```

Listing 4.1: Beispiele von Transscripten, die beibehalten (KEEP) oder verworfen werden (DISCARD).

2. Die Mehrheit der Eingaben besitzen nur ein bis drei Schlüsselwörter und können beim Trainieren zu einer *class imbalance* führen, da weniger Schlüsselwörter existieren als Wörter, die ignoriert werden sollen. Eine Lösung für dieses Problem ist das Umschreiben der Wörter zu deren zugehörigen Sprachlaute. Es werden zusätlich noch zwei weitere Zeichen (<, >) verwendet, um die Grenzen der Wörter zu definieren.

```
Hey Buddy. -> < HH EY > < B AH D IY >
Open the car -> < OW P AH N > < DH AH > < K AA R >
```

Listing 4.2: Transformation von Wörtern zu Sprachlaute.

cmudict (Carnegie Mellon Pronouncing Dictionary) ist ein Ausprachwörterbuch, welches für die Transformation verwendet wird.

3. Damit keine weiteren *class imbalance* zwischen mehr vorkommenden Sprachlaute, und nicht Schlüsselwörtern vorkommen, werden nur Einträge betrachtet, die gleichviele Sprachlaute von Schlüsselwörten und nicht Schlüsselwörten besitzen. Resultierend, wird der zweite Eintrag aus 4.2 verworfen.

Die *Feature Extraction* besteht aus dem Berechnen der MFCCs (Mel-frequency cepstral coefficients [3]; siehe 3.1) anhand der Audiodaten. Die Audiodaten wurden im vorraus dekodiert (Common Voice Daten sind als MP3 encodiert) und runter auf 16kHz Mono gesampled.

Zusätzlich werden die preparierten Daten aufgeteilt in Trainings- und Validierungsdaten (Evaluierungsdaten). Sie werden vor dem Training in *TensorFlow-Record* Daten abgespeichert.

Die Anzahl und Größe des Datensets nach der Aufbereitung sind in 4.1 dargestellt.

Tabelle 4.1: Anzahl und Größe des aufbereiteten Datensatzes.

Daten	**Anzahl der Audio/Transkript Paare**	**Datengröße**
Trainingsdaten	211254	36 GiB
Evaluierungsdaten	8305	1.5 GiB

4.3 RNN-Transducer Modell

Das Modell für die Evaluierung basiert auf einem RNN-Tranducer (wie in 2.3 beschrieben). Die Auswahl der Schichten innerhalb den *Encoder* und *Pred. Network* Komponenten basieren auf der verwendeten RNN-T Architektur aus dem Paper *Streaming End-to-end Speech Recognition For Mobile Devices* [6]. Der *Encoder* besteht aus mehreren LSTM und *Layer Normamlization* ([1]) Schichten. In [1] beschreiben die Authoren, dass die Anwendung von *Layer Normamlization* zwischen den LSTM-Schichten die "dynamische Stabilität"des Modells signifikant verbessert. Die LSTM-Schichten sind unidirektional ausgerichtet, welches überhaupt die *streaming* Funktionsweise erlaubt, da Daten zwischen den Zeitschritten nicht zurückgeführt werden. Später im *Encoder* wird auch eine *Time Reduction* Schicht [2] verwendet um zusätzlich die Größendimensionen zwischen Anfang und Ende zu Reduzieren. Am Ende beider Komponenten befinden sich eine *Fully-Connected* Schicht.

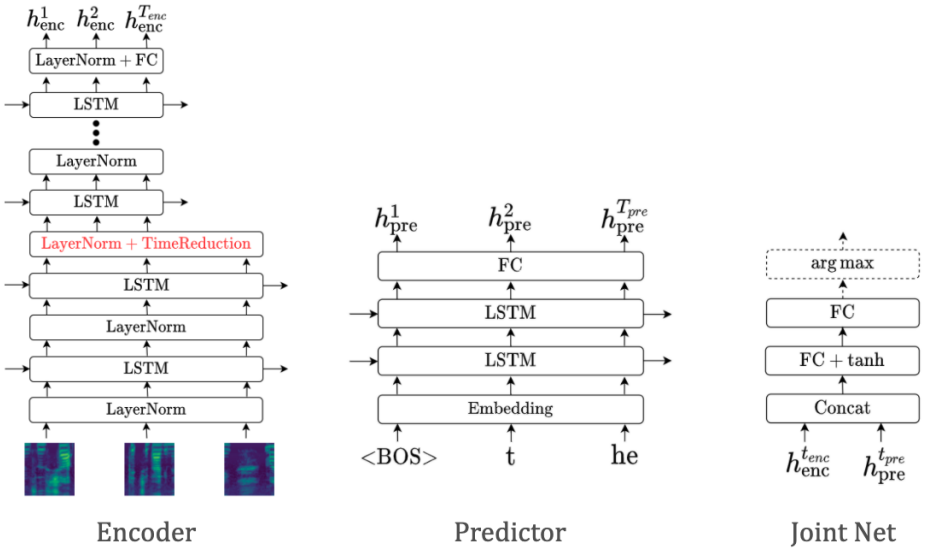

Abbildung 4.1: Modellaufbau der RNN-T Komponenten.
Quelle: https://github.com/theblackcat102/edgedict

Im Rahmen dieser Arbeit werden drei Varianten (Model 1 / 2 / 3) mit unterschlichen Hyperparameter analysiert:

Tabelle 4.2: Hyperparameter für Model 1, 2, 3.

Hyperparamter	**Model 1**	**Model 2**	**Model 3**
Anzahl LSTM-Schichten (*Encoder*)	4	6	8
Dimension LSTM-Schicht (*Encoder*)	512	768	512
Anzahl LSTM-Schichten (*Pred. Network*)	2	2	2
Dimension LSTM-Schicht (*Pred. Network*)	512	512	512
Dimension des *Join Net*	320	512	320
Anzahl der lernbaren Parameter	12.779.376	37.405.744	21.180.272

4.4 Optimierung für Embedded Systems

In 4.3 ist Model 1 die kleinste Variante von allen drei und besteht aus ca. 12.779.376 Parametern. Die Modelgröße in serialisierter Form erfasst ca. 48MiB (Übereinstimmend mit einer Approximierung, da 32bit Floatingwerte als Parameter verwendet werden).

Eine EdgeTPU (z.B. ein Coral USB-Stick) kann als Referenz eines Embedded Systems genommen werden, da sie bestimme mathematische Operationen beschleunigt ausführen kann. Dazu wird eine Post-Quantisierte Variante des Models benötigt. Eine Quantisierung ermöglicht eine Reduktion des Speicherbedarfs (Modelgröße) und das Erhöhen der Performance (Latenz der Ausführung des Modells) des Modells. Ein TensorFlow-Model nutzt intern typischerweise 32-bit Floatingwerte. Eine Quantisierung erreicht eine Approximierung der 32-bit in 16-bit Floatingwerten. Diese Konversation ist jedoch nicht Verlustfrei und kann zu Präzisionsverlust führen. Eine Konversation zu Integer Datentypen (8-bit Integers) ist auch möglich, und wird im Falll einer EdgeTPU erfordert. Modelgröße und Inferenzzeit/Latenz werden bei einer 8-bit Quantisierung weiter reduziert, da generell Integeroperationen performanter gegenüber Floatingoperationen sind. Die Post-Quantisierung erfolgt in der Evaluierung mit der TensorFlow-Lite Software.

4.5 Trainingsverfahren

Das Tranieren eines RNN-Transducer ist nicht trivial und benötigt signifikante Rechenresourcen, da während des Trainings mit schon kleinen Eingabe/Ausgabedimensionen und Labelanzahl große Tensoren benötigt werden und somit viel RAM-Speicher benötigt.

Für die Evaluierung der rechenintensiven Kostenfunktion eines RNN-T wird daher eine experimentelle Open-Source RNN-T Implemetierung [9] verwendet, die auf der GPU ausgeführt werden kann, welches die hohe Parallilität der GPU ausnutzt.

Die ICC (Informatik Compute Cloud) der HAW-Hamburg, ist ein Kubernetes Rechencluster, welches als Trainingsplatform für das Trainieren verwendet wird. Zum Zeitpunkt der Evaluierung wird eine NVIDIA V100 GPU (mit 16GB RAM) Resource aus dem Rechencluster verwendet mit ca. 200GB Blockspeicher, damit alle benötigten Daten lokal verfügbar sind.

Als Optimizer wird Stochastic Gradient Descent (SGD) (*stochastic gradient (with momentum) descent*) mit folgenden Parametern verwendet:

- Lernrate: 0.0001
- Momentum: 0.9

Die verwendete Trainingsbatchgröße liegt bei 32, da auch eine höhere Batchgröße wäre mit den verfügbaren Rechenresource nicht möglich wäre.

5 Evaluation

Es werden zunächst die Kriterien der Evaluation definiert, welche als Grundlage zur Bewertung der Ergebnisse verwendet werden. Als nächstes werden die Ergebnisse des Trainings vorgestellt. Darüber hinaus werden die Ergebnisse diskutiert und Future Work Elemente geschildert.

5.1 Kriterien

Die primäre Metrik für die Evaluation des RNN-T Modells als KWS ist die *word error rate* (WER), welches im Bereich der automatischen Spracherkennung meist verwendet wird.

Die generelle Schwierigkeit ein KWS System auszuwerten ist, dass die Ausgabesequenz im Vergleich zur Referenzsequenz unterschiechlich lang sein kann. Daher basiert die *word error rate* auf der Levenshtein-Distanz auf Basis von Wörtereinheiten.

Die *word error rate* ist wie folgt definiert:

$$WER = \frac{S + I + R}{N} \tag{5.1}$$

S steht für die Anzahl der Wortsubstitutionen im Bezug zur Referenz. I steht für die Anzahl der Worteinfügungen. R steht für die Anzahl der Entfernung von Wörtern. N steht für die gesamte Wortanzahl der Referenzsequenz.

Im Beispiel 5.1 sind insgesamt 8 Wortsubstitutionen, 2 Worteinfügungen und eine Wortentfernung zu erkennen. Daher beträgt die $WER = \frac{8+2+1}{29} \approx 0.379 = 37.9\%$.

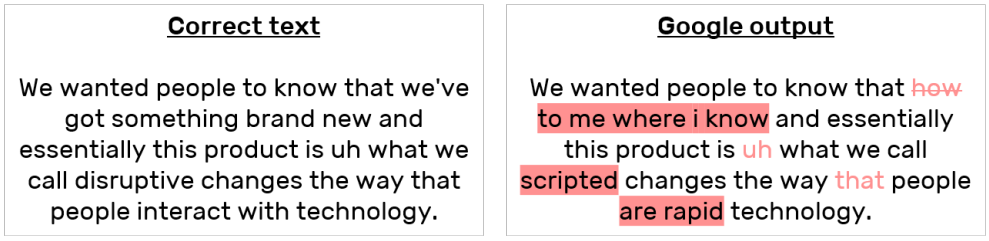

Abbildung 5.1: Links ist die Referenzsequenz zu sehen. Rechts die Vergleichssequenz. **Quelle:** rev.ai - how to calculate wer

5.2 Ergebnisse

Alle drei Modelle 4.3 wurden wie in 4.5 beschrieben auf der ICC trainiert. Die Dauer mit den verfügbaren Resourcen hat pro Modell ca. drei ganze Tage beansprucht (über 62h), bis keine signifikanten Änderungen mehr zu sehen waren und eine mögliche Konvergenz zu erwarten ist. Bei allen drei Modellen wurden bis zur 100ten Epoche trainiert, mit jeweils 6601 Batches.

Model 1

Während des Trainings von Model 1, ist ein stetiger Lernvorgang in 5.2 und 5.3 zu erkennen.

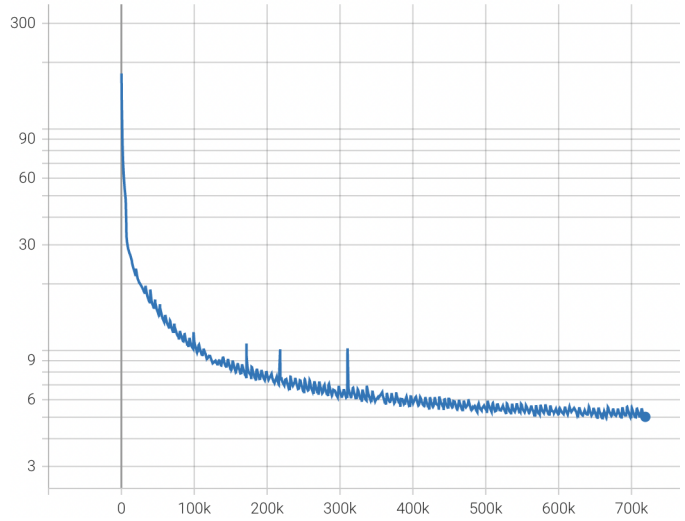

Abbildung 5.2: Verlauf der Lossfunktion (Trainingsdaten) bei Model 1. In logarithmischer Darstellung.
Quelle: Paul Nguyen

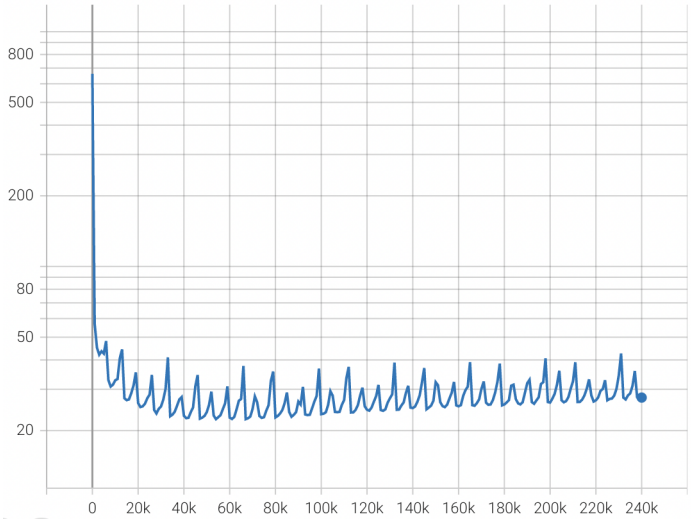

Abbildung 5.3: Verlauf der Lossfunktion (Evaluierungsdaten) bei Model1 . In logarithmischer Darstellung.
Quelle: Paul Nguyen

In 5.4 ist zu sehen, dass der durchschnittliche WER-Wert (mit der “besten” Epoche), evaluiert auf den Evaluationsdaten, ca. $WER \approx 0.35 = 35\%$ beträgt.

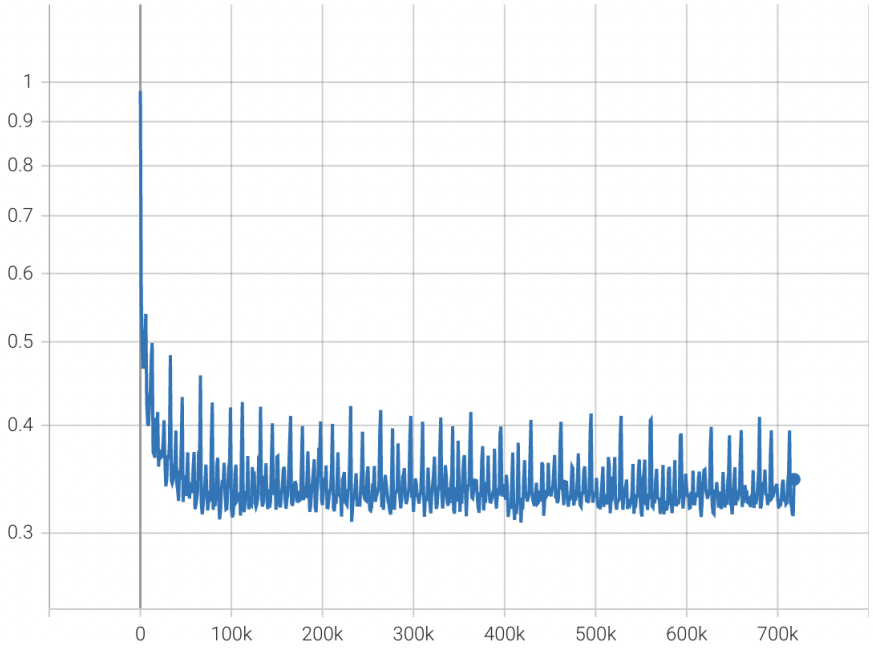

Abbildung 5.4: Verlauf von WER (Evaluierungsdaten) bei Model 1. In logarithmischer Darstellung.
Quelle: Paul Nguyen

Model 2

Während des Trainings von Model 2, ist ein stetiger Lernvorgang in 5.5 und 5.6 zu erkennen.

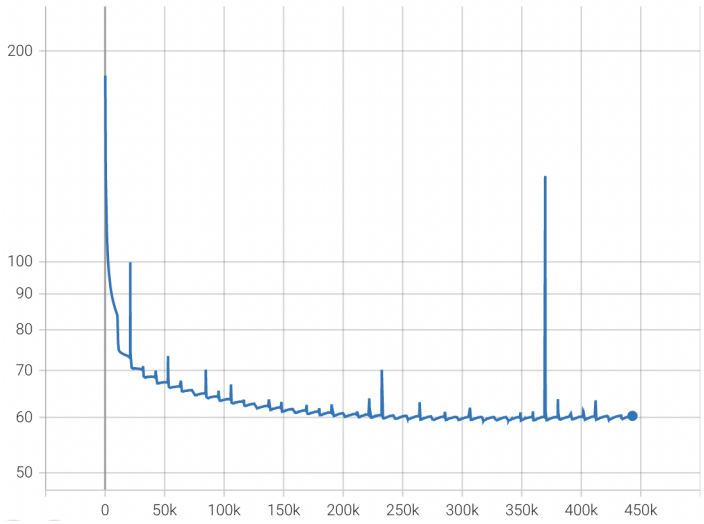

Abbildung 5.5: Verlauf der Lossfunktion (Trainingsdaten) bei Model 2. In logarithmischer Darstellung.
Quelle: Paul Nguyen

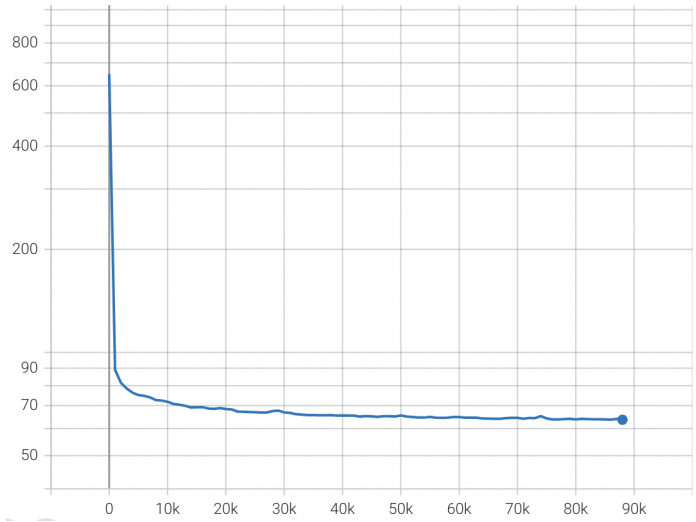

Abbildung 5.6: Verlauf der Lossfunktion (Evaluierungsdaten) bei Model 2. In logarithmischer Darstellung.
Quelle: Paul Nguyen

In 5.7 ist zu sehen, dass der durchschnittliche WER-Wert (mit der "besten" Epoche), evaluiert auf den Evaluationsdaten, ca. $WER \approx 0.8 = 80\%$ beträgt. Jedoch sind bei Model 2 sehr starke Schwankungen bei der WER-Evaluierung zu erkennen.

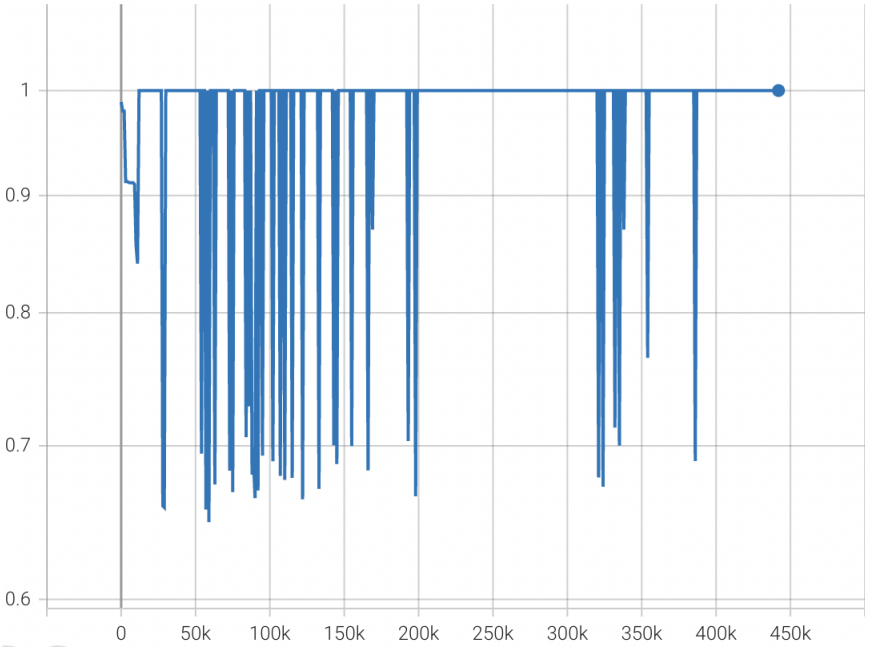

Abbildung 5.7: Verlauf von WER (Evaluierungsdaten) bei Model 2. In logarithmischer Darstellung.
Quelle: Paul Nguyen

Model 3

Während des Trainings von Model 3 ist ein stetiger Lernvorgang in 5.8 und 5.9 zu erkennen.

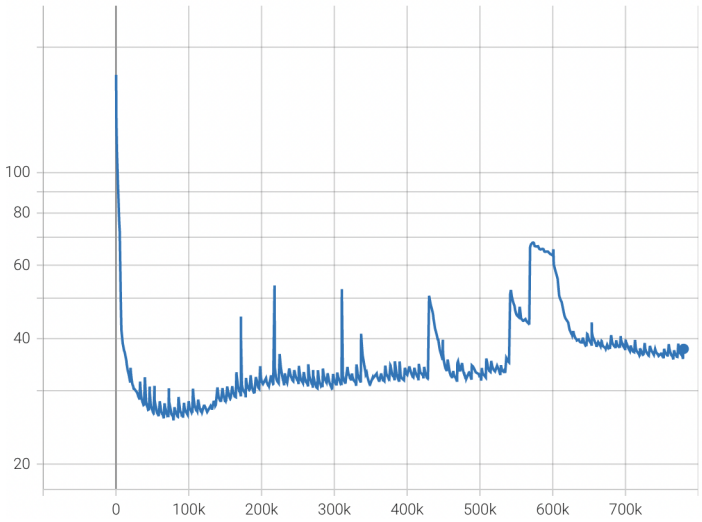

Abbildung 5.8: Verlauf der Lossfunktion (Trainingsdaten) bei Model 3. In logarithmischer Darstellung.
Quelle: Paul Nguyen

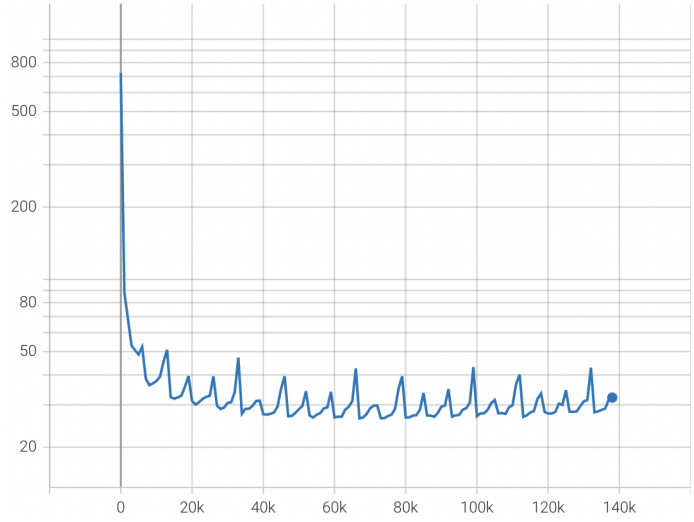

Abbildung 5.9: Verlauf der Lossfunktion (Evaluierungsdaten) bei Model 3. In logarithmischer Darstellung.
Quelle: Paul Nguyen

In 5.10 ist zu sehen, dass der durchschnittliche WER-Wert (mit der "besten" Epoche), evaluiert auf den Evaluationsdaten, ca. $WER \approx 0.5 = 50\%$ beträgt.

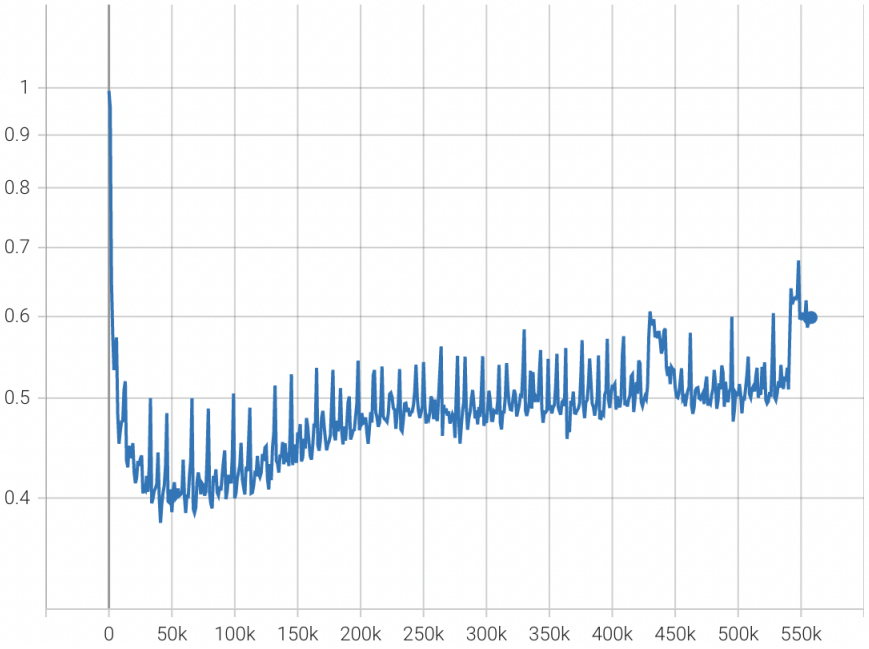

Abbildung 5.10: Verlauf von WER (Evaluierungsdaten) bei Model 3. In logarithmischer Darstellung.
Quelle: Paul Nguyen

Post-Quantisierungs Performance

Die Post-Quatisierung über TensorFlow-Lite produziert funktionale Model Varianten, die 8-bit Integer quantisiert sind. Die folgende Tabelle stellt die Größen und Inferenzzeiten im Vergleich zur Referenz dar:

Tabelle 5.1: Post-Quantisierungs Ergebnisse

Model Variante	**Größe des Modells**	**Größe des TF-Lite Modells**	**Inferenzzeit (CPU)**	**Inferenzzeit (EdgeTPU)**
Model 1	**49 MiB**	**15 MiB**	**220ms**	**20ms**
Model 2	143 MiB	37 MiB	471ms	65ms
Model 3	81 MiB	22 MiB	301ms	42ms

5.3 Diskussion

Die Trainingsergebnisse von Model 1, 2 und 3 zeigen dass die Modelle schließlich zu einem Lernstillstand konvergieren. Das kleinste Model unter den Drei, ist Model 1, welches auch den kleinsten WER mit ca. 35% besitzt. Jedoch, schneidet ein KWS, bei so einer Rate relativ schlecht ab, da sich die Ausgabe, welches aus Sprachlauten besteht, bei WER-Raten von über 10%, nicht trivial rekonstruieren lässt.

Das größte Model 3 schneidet schlecht ab, bei den ungesehende Evaluierungdaten. Ein Grund dafür ist das *overfitting* Problem. Das Model enthält zu viele Parameter, was dazu führt, dass keine Generalisierung über die Trainingsdaten erfolgt. Daher schneiden Modell 1 und 2 besser ab, denn anhand des WER-Wertes ist erkennbar, dass dort eine Generalisierung stattgefunden hat.

Die schlechte Performance von Model 1 und 2 können folgende Gründe haben:

Das Trainingsdaten können, trotz der Gleichverteilung der Sprachlaute (wie in 4.2 beschrieben), zu wenig Keywort relevante Informationen enthalten, so dass wir das Netzwerk nicht auf die Schlüsselwörter, sondern auf den sämtlichen Sprachlautebereich generalisieren (d.h. das Model bildet eher ein generelles Spracherkennungssystem ab). Folglich, kann die Transformation der Wörter aus den Transkripten könnte suboptimal gewesen sein.

Die generelle Dimension des Models könnte auch eine signifikante Rolle spielen. Größere Modelle haben den Vorteil mehr Informationen zu lernen, jedoch spielt hier das *overfitting* Problem eine große Rolle. Das *overfitting* Problem kann nur mit weiteren geeigneten Trainingsdaten gelöst werden.

Das Erweitern der Trainingsdaten hat im Folge weitere Konsequenzen bzgl. der Trainingsperformance. Mehr Trainingsdaten erforden mehr Rechenresourcen, welche im Rahmen diese Arbeit nur limitierend vorhanden sind.

Die Problematik kann auch bereits auf dem fundamentalen Level mit der Modelarchitekturauswahl zusammenhängen. [6] beschreibt das Trainieren eines RNN-Transducers mit Datensätzen, die um vielfaches größer sind als den evaluierten Datensatz von Mozilla. Dies könnte auf eine Characteristik eines RNN-Tranducers hinweisen, dass Trainingsdaten im großen Umfang vorhanden sein muss, um optimale Generalierung zu erreichen.

Die Embedded-System Anforderungen konnten jedoch erfüllt werden. Die Post-Quantisierungs Ergebnisse in 5.1 zeigen gute Inferenzzeiten (d.h. gute Latenzzeiten) und können mit Voraussetzung einer EdgeTPU im Embedded Systems Kontext verwendet werden.

6 Fazit und Future Work

6.1 Fazit

Wir haben erfolgreich ein RNN-Transducer Model für ein limitiertes KWS entwickelt, implementiert, trainiert und evaluiert. Das Trainiern erfolgte auf der Informatik Compute Cloud (ICC) (Rechencluster des HAW-HAmburg). Die erste evaluierte Variante erreicht ein WER von 35%, welches aber nicht für das definierte KWS ausreicht. Die Ergebnisse haben gezeit, dass der Trainingsaufwand eines RNN-Transducers, im Bezug zur Dauer und Rechenresourcen, sehr hoch ist (Die gesamte Dauer der Trainingsphase von allen drei Modellvarianten umfasste drei bis fünf Tage). Die Dimensionen des Modells und die verfügbaren Trainingsdaten zeigen eine sehr hohe WER-Sensitivität und starke Abhängigkeit. Die diskutierten Ergebnisse zeigen, dass die RNN-T Modellarchitektur sich nicht einfach auf ein KWS anpassen lässt. Es wurden trozdessen Informationen gesammelt bgzl. der Anwendbarkeit eines RNN-Transducer im Bereich eines KWS.

6.2 Future Work

Die sub-optimalen Ergebnisse geben dennoch Raum für Verbesserungen. Folgende Bereiche könnten Folgearbeiten beinhalten:

Es können kleinere Datensätze zusammen mit einem Hyperparameter-reduziertem RNN-Transducer verwendet werden. Um die entgültige Brauchbarkeit eines RNN-Transducers mit kleineren Datensätzen festzustellen. Die Aufbereitung der Daten könnte anders erfolgen, so dass vollständige Wörter verwendet werden und trotzdessen keine *class imbalance* auftritt.

Die ausgewählen Hyperparameter, in dieser Arbeit, könnten nicht optimal für den Anwendungsfall gewesen sein. Unter Umständigen, wenn mehr Rechenresourcen verfügbar sind, können weitere Hyperparameterkonfigurationen evaluiert werden.

Letzlich, ist es möglich, dass ein RNN-Transducer nicht für ein KWS geeignet ist. Andere Basisarchitekturen können im Bezug eines KWS evaluiert werden.

Literaturverzeichnis

[1] Ba, Jimmy L. ; Kiros, Jamie R. ; Hinton, Geoffrey E.: *Layer Normalization*. 2016

[2] Chan, William ; Jaitly, Navdeep ; Le, Quoc V. ; Vinyals, Oriol: *Listen, Attend and Spell*. 2015

[3] Davis, S. ; Mermelstein, P.: Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28 (1980), Nr. 4, S. 357–366

[4] Fernández, Santiago ; Graves, Alex ; Schmidhuber, Jürgen: An Application of Recurrent Neural Networks to Discriminative Keyword Spotting. In: Sá, Joaquim M. de (Hrsg.) ; Alexandre, Luís A. (Hrsg.) ; Duch, Włodzisław (Hrsg.) ; Mandic, Danilo (Hrsg.): *Artificial Neural Networks – ICANN 2007*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007, S. 220–229

[5] Graves, Alex: *Sequence Transduction with Recurrent Neural Networks*. 2012

[6] He, Yanzhang ; Sainath, Tara N. ; Prabhavalkar, Rohit ; McGraw, Ian ; Alvarez, Raziel ; Zhao, Ding ; Rybach, David ; Kannan, Anjuli ; Wu, Yonghui ; Pang, Ruoming ; Liang, Qiao ; Bhatia, Deepti ; Shangguan, Yuan ; Li, Bo ; Pundak, Golan ; Sim, Khe C. ; Bagby, Tom ; Chang, Shuo-Yiin ; Rao, Kanishka ; Gruenstein, Alexander: Streaming End-to-end Speech Recognition For Mobile Devices. In: *CoRR* abs/1811.06621 (2018). – URL http://arxiv.org/abs/1811.06621

[7] Hochreiter, Sepp: Untersuchungen zu dynamischen neuronalen Netzen. (1991), 04

[8] Hochreiter, Sepp ; Schmidhuber, Jürgen: Long Short-term Memory. In: *Neural computation* 9 (1997), 12, S. 1735–80

[9] Huang, Mingkun: *warp-transducer*. https://github.com/HawkAaron/warp-transducer. 2018

[10] Jeong, Jiho ; Mondol, S I M M R. ; Kim, Yeon-Wook ; Lee, Sangmin: An Effective Learning Method for Automatic Speech Recognition in Korean CI Patients' Speech. In: *Electronics* 10 (2021), 03, S. 807

[11] Pete Warden, Google Brain T.: *Launching the Speech Commands Dataset*. 2017. – URL https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html

[12] Wang, Dong ; Wang, Xiaodong ; Lv, Shaohe: An Overview of End-to-End Automatic Speech Recognition. In: *Symmetry* 11 (2019), 08, S. 1018

[13] Wilpon, J.G. ; Rabiner, L.R. ; Lee, C.-H. ; Goldman, E.R.: Automatic recognition of keywords in unconstrained speech using hidden Markov models. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 38 (1990), Nr. 11, S. 1870–1878

[14] Zhang, Yundong ; Suda, Naveen ; Lai, Liangzhen ; Chandra, Vikas: Hello Edge: Keyword Spotting on Microcontrollers. In: *CoRR* abs/1711.07128 (2017). – URL http://arxiv.org/abs/1711.07128

A Anhang

A.1 Befehlswörter

the, to, on, no, around, left, me, off, back, right, go, loomo, start, turn, stop, drive, follow, walk, forward, yes, straight, reverse, hey, ok, turnaround, sentence.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Keyword-spotting für eingebettete Systeme auf Basis eines RNN-Transducers

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original