FH Westküste
Wirtschaft und Technik

HAW HAMBURG

Masterarbeit

Morad Haidar

Fallstudien zur Anwendbarkeit von Deep-Learning Modellen auf Microcontrollern

Fachhochschule Westküste
Fachbereich Technik

Fachhochschule Westküste - University of Applied Sciences
Faculty of Engineering

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Hamburg University of Applied Sciences
Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Morad Haidar

Fallstudien zur Anwendbarkeit von Deep-Learning Modellen auf Microcontrollern

Masterarbeit eingereicht im Rahmen der Masterprüfung
im gemeinsamen Studiengang Mikroelektronische Systeme
am Fachbereich Technik
der Fachhochschule Westküste
und
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Klaus Jünemann, HAW Hamburg
Zweitgutachter: Prof. Dr.-Ing. habil. Michael Berger, FH Westküste

Abgegeben am 10.12.2020

Morad Haidar

Thema der Bachelorthesis

Fallstudien zur Anwendbarkeit von Deep-Learning Modellen auf Microcontrollern

Stichworte

Tiefe neuronale Netze, Mikrocontroller, Spracherkennung, Bilderkennung, Transfer-Learning, Data Augmentation, Spektrogramm, Python, Tensorflow, Keras.

Kurzzusammenfassung

Im Rahmen dieser Arbeit wird untersucht, inwiefern Deep-Learning-Modelle in verschiedenen Anwendungsgebieten wie Sprach- und Bilderkennung auf kleinen Microcontrollern implementierbar sind. Dabei werden mehrere Modelle und Beispiele erstellt und auf einem passenden Microcontroller ausgeführt, wobei verschiedene Techniken wie Transfer-Learning eingesetzt werden. Ferner wird anschließend eine per Sprache gesteuerte Lampe gebaut, die mehrere Sprachbefehle verstehen kann, ohne dass eine Internet-Verbindung benötigt wird.

Morad Haidar

Title of the paper

Case studies on the applicability of deep learning models on microcontrollers

Keywords

Deep-Learning, microcontroller, speech recognition, image recognition, transfer-learning, data augmentation, spectrogram, python, Tensorflow, Keras.

Abstract

This work investigates the extent to which deep-learning models can be implemented on microcontrollers in various application areas such as speech and image recognition. Several models and examples are created and executed on a suitable microcontroller. Furthermore, the knowledge acquired is then used to build a voice-controlled lamp with various functions without an internet connection.

Inhaltverzeichnis

Kapitel 1 - Einleitung

1.1 Hintergrund

In den 1980er Jahren wurde zusammen mit den Entwicklungen und Fortschritten in der Soft- und Hardware versucht, KI-Maschinen und Expertensysteme zu entwickeln, die In der Industrie viele Anwendungen finden könnten. Die Prognosen der Industrie konnten aber nicht eingehalten. Dies Verursachte einen Rückgang von Investitionen und Erwartungen (KI-Winter). Denn es hat sich herausstellte, dass diese Systeme, die auf Wissensbanken basieren, erfordern ein Verständnis aller Abhängigkeiten und Zusammenhänge der Daten, um diese in formale Regeln zu überführen. Dieses Problem wurde durch maschinelles Lernen gelöst. Hierbei lernt das Modell anhand der Trainingsdaten selbstständig die verborgenen Zusammenhänge zu erkennen, die Menschen nicht verstehen oder berücksichtigen können. In den ersten ML-Modellen wurden die Modelle auf Merkmale trainiert, die aus den Eingabedaten oder Trainingsdaten extrahiert worden sind. Die Extraktion dieser Merkmale wurde mit herkömmlicher Programmierung gewonnen. Diese Trainingsmethode weist ein Problem auf. Denn die Programmierer, die Programme zur Extraktion von den Merkmalen schreiben, konnten die verborgenen Zusammenhänge nicht berücksichtigen. Dieses Problem wurde gelöst, indem eine spezielle Struktur entworfen wurde, die direkt aus den Eingabedaten selbst Merkmale extrahieren und lernen können. Diese Strukturen sind die neuronalen Netze oder KNNs (künstliche neuronale Netze). Die ersten KNNs wurden ausgehend von der Informationsarchitektur des menschlichen und tierischen Gehirns als lineare Modelle entwickelt. Ein Beispiel dafür ist die McCulloch-Pitts-Zelle, die 1943 entwickelt wurde und in Abbildung 1-1 dargestellt ist. [1]

Diese Linearen Strukturen wurden zuerst abgelehnt und ausgeschlossen, weil sie die logischen Funktionen wie XOR nicht lösen konnten. Diese und solche Probleme benötigten ein nichtlineares Modell.

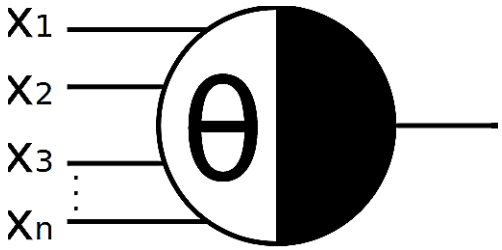


Abbildung 1-1: Die McCulloch-Pitts-Zelle [2]

In den letzten Jahren haben sich die für komplexere Modelle benötigte leistungsstarke Hard -und Software und für das Trainieren erforderlichen Datensätze entwickelt. Das hat ermöglicht, mehrlagige Strukturen, die nicht linear sind und nichtlineare Probleme wie XOR-Problem lösen konnten, zu realisieren. Diese Strukturen haben seit 2009 große Erfolge bei Klassifizierungs- und Mustererkennungsproblemen erzielt. Diese aus vielen Schichten bestehenden neuronalen Netzen werden als Deep-Learning bezeichnet. [3]

1.2 Aktueller Stand des maschinellen Lernens

Die Leistungsfähige Hardware und die Verfügbarkeit riesiger Mengen von Daten haben ermöglicht, komplexe ML-Modelle in den verschiedensten Anwendungsgebieten zu realisieren. In der jetzigen Zeit gibt es kaum eine Branche, in der ML keine Anwendung findet. Eine der erfolgreichsten und weitverbreitetsten Anwendungen des maschinellen Lernens ist die Objekterkennungsanwendung, die in der Lage ist, Objekte auf Bildern zu erkennen. Solche Modelle sind in dem autonomen Fahren einsetzbar und möglicherweise die Mobilität der Zukunft prägen würden. Weitere wichtige Anwendungen des maschinellen Lernens sind die Verarbeitung natürlicher Sprache und die Spracherkennung. Diese Anwendungen wie die digitalen Übersetzer oder per Sprache interaktive Systeme haben sich in den letzten wenigen Jahren sehr verbreitet und sind jetzt fast auf jedem Handy zu finden.

In der Finanzwelt spielt zurzeit maschinelles Lernen eine einschneidende Rolle bei dem Vorhersagen von Aktienpreisen, die Komplexitäten aufweisen und viele Zusammenhänge haben, die nicht von Menschen verständlich sind. Es gibt noch komplexere Probleme, die normalerweise nicht vorherzusagen sind wie beispielsweise das Vorhersagen des menschlichen Verhaltens beim Einkaufen.

Einige Handelsfirmen, die auch in diesem Gebiet tätig sind, haben erstaunliche Erfolge erzielt, wie zum Beispiel das Vorhersagen von der Schwangerschaft anhand des Verhaltens von schwangeren Frauen beim Einkaufen, bevor die Frau selbst gewusst hat, schwanger zu sein.

Maschinelles Lernen ist nicht ganz harmlos. Denn manche Regierungen haben bereits angefangen, durch ML-Anwendungen die Menschen zu überwachen und zu zensieren, was eine Einschränkung der Freiheit und Menschenrechte in diesen Ländern bedeutet. Beispielsweise wurde in China eine Anwendung zur Erkennung von Verbrechern anhand von Gesichtsbildern entwickelt. Solche Anwendungen könnten sehr gefährlich sein und könnten dazu führen bestimmte Menschengruppen zu Unrecht zu stigmatisieren oder zu verdächtigen.

1.3 Ziele der Arbeit

Maschine-Learning-Algorithmen sind in meisten Fällen rechenintensiv und benötigen einen großen Speicher, besonders beim Trainieren, was bisher bedeutet hat, dass diese komplexen Modelle nur von Firmen wie Google oder Amazon, die über riesige Rechenzentren und Big-Data verfügen, umgesetzt werden konnten. Deshalb gehören bis jetzt alle ML-Anwendungen wie Sprachassistenten diesen Firmen. Das hat aber viele Nachteile wie später erläutert wird.

In den letzten wenigen Jahren ist eine neue ML-Trend ins Leben gerufen, die sich damit beschäftigt, ML-Modelle auf kleinen günstigen Mikrocontrollern zu implementieren. Solche Implementierung der ML-Modelle ist für uns in dieser Arbeit vom großen Interesse, da viele Anwendungen und Vorteile mit sich bringen könnte.

In dieser Arbeit soll untersucht werden, inwieweit sich tiefe ML-Modelle auf kleine günstige SOC-Boards und verschiedene Mikrocontroller-Architekturen übertragen lassen. Gefordert ist dabei die Erstellung von ML-Modellen aus verschiedenen Anwendungsgebieten.

Als erstes werden bereits erstellte Modelle auf den vorhandenen Mikrocontrollern ausgeführt. Dies soll dazu dienen, die verschiedenen Mikrocontrollerboards zu vergleichen und einen Überblick darüber zu geben, in welchem Ausmaß ML-Modelle auf Mikrocontrollern Implementierbar sind.

Anschließend ist gefordert die Erstellung eines Objekterkennungsmodells und Spracherkennungsmodells und der für das Training benötigten Datensätze. Die erstellten Modelle sollen dann auf einem geeigneten Mikrocontroller ausgeführt werden.

Rekurrente Netze sind wichtige Strukturen, die zur Erkennung von zeitlich codierten oder sequenziellen Daten eingesetzt werden können. Daher ist die Implementierung von diesen Netzen auf dem Mikrocontroller viele Vorteile mitbringen und Anwendungen finden kann. Deswegen muss im Laufe dieser Arbeit nach möglichem Einsatz dieser neuronalen Netze zur Erkennung von einfachen Wörtern auf dem Mikrocontroller gesucht werden.

Transfer-Learning ist eine sehr wichtige Technik, die die Übertragung eines vortrainierten Netzes auf eine neue Aufgabe ermöglicht. Diese Technik wird üblicherweise eingesetzt, wenn das Modell sehr groß und rechenintensiv ist, wobei die Trainingszeit Tage oder Wochen dauern könnte. Daher wird in der Arbeit gefordert, nach einem möglichen Einsatz dieser Technik auf dem Mikrocontroller zu suchen.

Anhand von den erworbenen Kenntnissen im Bereich Spracherkennung soll ein per Sprache gesteuerte Lampe aufgebaut werden. Hierfür soll, wenn möglich, ein LSTM-Netz zur Erkennung von Sprachbefehlen eingesetzt werden. Falls sich LSTM-Netze zu diesem Zweck nicht einsetzen lassen, sollte das Ziel durch Kopplung mehrerer Mikrocontrollerboards unter Verwendung von Faltungsnetzen erreicht werden. Diese Anwendung ist der Hauptteil dieser Arbeit. Was dabei wichtig ist, dass die Erkennung der Sprachbefehle auf dem Mikrocontroller ohne Internetverbindung stattfindet.

1.4 Überblick über die Arbeit

Als nächstes werden Im Kapitel 2 die wichtigsten ML-Strukturen, die in dieser Arbeit eingesetzt werden, und ihre Funktionsweise im Allgemeinen vorgestellt. Anschließend werden die Vor- und Nachteile von ML-Algorithmen auf Mikrocontrollern untersucht und die Hard- und Software, die für diese Arbeit benötigt werden, aufgelistet.

ImKapitel 3 werden einfache ML-Anwendungen auf verschiedenen Mikrocontroller-boards ausgeführt und Messungen durchgeführt. Dies soll dazu dienen, einen Überblick darüber zu geben, wie schnell ein Modell auf dem Mikrocontroller ausgeführt werden kann, wie groß das Modell sein darf, und welches Board am besten passt.

Anschließend werden Im Kapitel 4 zwei Modelle im Bereich Spracherkennung und Computervision erstellt, trainiert und auf den Mikrocontroller übertragen. Dabei wird auf die kleinsten Einzelheiten eingegangen.

Das Kapitel 5 beinhaltet eine Einführung in die theoretische Funktionsweise der rekurrenten Netze. Nach dieser Einführung wird ein Spracherkennungsmodell unter Verwendung von LSTM-Netzen erstellt, trainiert und auf dem Mikrocontroller ebenfalls ausgeführt.

Das Kapitel 6 befasst sich mit Transfer-Learning im Allgemeinen. Nach einer Einführung in diese wichtige Technik, wird nach einem passenden Modell gesucht und auf dem Mikrocontroller ausgeführt.

Als Hauptanwendung dieser Arbeit wird imKapitel 7 eine per Sprache gesteuerte Lampe als ein Beispiel für ein per Sprache interaktives System entworfen. Für diese Anwendung werden viele Bibliotheken und Protokolle eingesetzt und viele Eigenschaften der Mikrocontrollerboards zunutze gemacht. Hierbei sollen die Farbe und Stärke sowie die Stromversorgung einer RGB-Lampe per Sprache gesteuert werden. Zudem kommt eine zusätzliche Funktion hinzu, die dem Benutzer ermöglicht, sich Hilfe im Notfall über das Internet zu holen. Hierbei ist die Erkennung der Sprachbefehle keine Internetverbindung benötigt.

Im letzten Kapitel 8 wird ein Fazit über die gesamte Arbeit gezogen. Hierbei wird erläutert, inwieweit die Ziele der Arbeit erreicht wurden und inwiefern jede verwendete Technik auf dem Mikrocontroller implementierbar ist.

Kapitel 2 - Deep Learning auf dem Mikrocontroller

In diesem Kapitel wird auf die wichtigsten Konzepte des maschinellen Lernens, und was sich dabei von den herkömmlichen Programmierungsmethoden unterscheidet, eingegangen. Zudem werden die wichtigsten ML-Strukturen vorgestellt, die im Laufe der Arbeit verwendet werden.

Da es sich bei dieser Arbeit um ML-Algorithmen auf dem Mikrocontroller handelt, werden die wichtigsten Hindernisse und Schwierigkeit bei der Implementierung erklärt und nach möglichen Lösungen gesucht. Außerdem werden für die Arbeit Soft -und Hardware benötigt, die später in diesem Kapitel aufgelistet und kürz erklärt sind. Das wichtigste Teil dieses Kapitels ist das Flussdiagramm (Workflow), das die Arbeitsfolge Schritt für Schritt veranschaulicht.

2.1 Machine- Learning und Deep-Learning

Im Jahr 2006 wurde zum ersten Mal ein neuronales Netz, das in der Lage war, handgeschriebene Ziffern mit der Genauigkeit >98% zu erkennen, vorgestellt. [4] Zu dieser Zeit war dieses Verfahren ausgeschlossen und als unmöglich angesehen. Dieser Erfolg hat bewiesen, dass Deep Learning Verfahren nicht nur möglich war, sondern auch viele Anwendungen finden kann. Heute hat das maschinelle Lernen ganze Industriebranchen wie Spracherkennung, Sprachübersetzer, Suchmaschinen und vieles mehr dominiert. [4]

Das Erstellen eines maschinellen Lernprogramms unterscheidet sich vom üblichen Verfahren, wobei ein Code geschrieben wird. Bei der herkömmlichen Programmierung entwirft ein Programmierer einen Algorithmus, der eine Eingabe übernimmt, verschiedene Operationen anwendet, und ein Ergebnis zurückgibt. Diese Operationen werden vom Programmierer über Codezeilen explizit implementiert. [4] Um etwas vorherzusagen muss der Programmierer in diesem Fall alle Abhängigkeiten und Daten und worauf sie hinweisen, verstehen. In vielen Fällen kann es aber schwierig sein, alle Kombinationen und Faktoren zu erkennen, die einen bestimmten Zustand repräsentieren. Bei Maschine-Learning gibt der Programmierer Daten in einen speziellen Algorithmus ein und lässt diesen Algorithmus die Regeln erkennen und aus früheren Erfahrungen lernen. Dies bedeutet, dass Modelle erstellt werden, die

Vorhersagen treffen können, ohne die gesamte Komplexität verstehen zu müssen. Als Beispiel dazu kann der Spamfilter-Anwendung in [4] die Funktionsweise beider Methoden deutlich machen.

Mit der herkömmlichen Programmierung wird folgendes gemacht:

1. Es wird überlegt, wie der Spam typischerweise aussieht, welche Begriffe sich häufig in der Betreffzeile oder im Text der E-Mail wiederholen oder auffällig sind.
2. Dann wird ein Algorithmus geschrieben, der Merkmale basierend auf diesen Auffälligkeiten aus der E-Mail extrahiert. Sobald bestimmte Kriterien erfüllt werden, werden E-Malis als Spam klassifiziert.
3. Diese zwei Schritten werden wiederholt und das Programm wird überarbeitet, bis es gut genug erkennt, welche E-Mails als Spam klassifizieren muss. [4]

Es ist hier zu merken, dass es keine leichte Aufgabe ist, eine große Anzahl von Regeln zu programmieren. Außerdem gibt es viel komplexere Aufgaben, bei denen es keine klaren und gut verstandenen Merkmale und Regeln gibt, und komplexere Abhängigkeiten aufweisen, wie beispielweise das Vorhersagen von Börsen oder Menschenverhalten beim Einkaufen. Die Abbildung 2-1 zeigt die herkömmliche Vorgehensweise bei der Lösung solcher Probleme, die in [4] beschrieben wird.

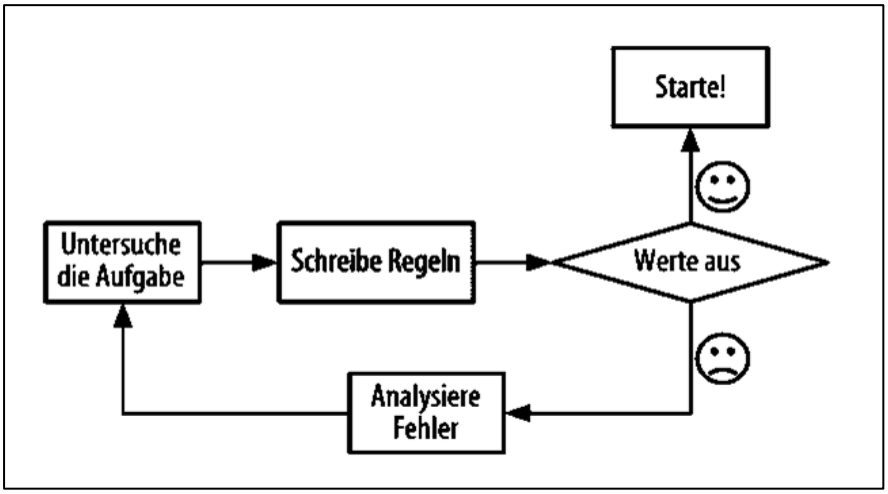


Abbildung 2-1: herkömmliche Vorgehensweise [4]

Bei Maschine-Learning-Algorithmen lernt der Algorithmus aus früheren Erfahrungen und vorhandenen Daten durch das Trainieren, wie er Emails als Spam markieren muss. Diese Vorgehensweise vereinfacht das Programm und erfordert kein genaues Verständnis von den Regeln und die Ergebnisse sind in vielen Fällen besser. Die

Abbildung 2-2 veranschaulicht wie ein Maschine-Learning-Algorithmus aus Erfahrungen und vorhanden Daten (Datensatz) lernen kann.

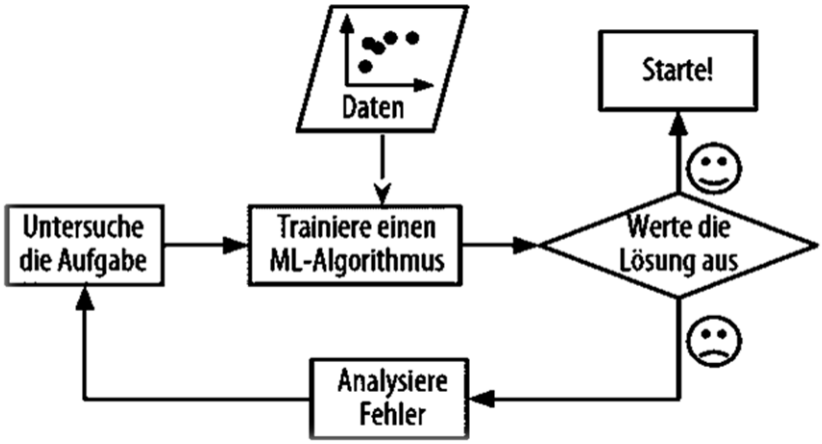


Abbildung 2-2: Die Vorgehensweise beim Trainieren von ML-Modellen [4]

Die Vorgehensweise beim Trainieren von ML-Modellen bleibt aber eine schlechte Nachahmung des menschlichen oder auch tierischen Gehirns. Denn das menschliche Gehirn lernt viel schneller und viel flexibler als solche Algorithmen. Beispielsweise sieht der Mensch nur einmal eine Katze und lernt aus dieser einzigen Erfahrung oder aus einer kleinen Zahl an Erfahrungen, wie die Katze aussieht. Beim Maschine-Learning benötigt der Algorithmus hingegen eine große Anzahl von gespeicherten Erfahrungen zum Training. Deswegen gab es viele Forschungen und Ideen, die untersuchten, wie der Algorithmus schneller lernen und verallgemeinern kann. Solche Techniken wie One-shot, Few-Shot oder n-Shot werden von Wissenschaftlern entwickelt, um die zum Trainieren erforderliche Datenmenge stark zu reduzieren. [5]

2.1.1 Maschinelles Lernen, künstliche Intelligenz und Deep Learning

Es gibt in Forschungsgebiet des maschinellen Lernens drei Begriffe, die häufig verwechselt werden. Diese Begriffe sind: maschinelles Lernen, künstliche Intelligenz und Deep Learning. Die folgende Abbildung 2-3 verdeutlicht den Unterschied zwischen den drei Gebieten. [6]

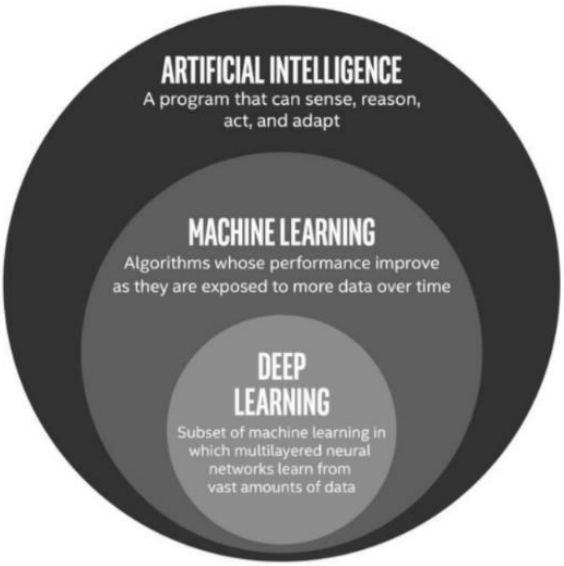


Abbildung 2-3: KI, ML und Deep Learning [6]

Künstliche Intelligenz umfasst im Allgemeinen die Systeme, die wie Menschen handeln, denken, rational handeln, oder rational denken. [7] Maschinelles Lernen ist dabei ein Teilgebiet der künstlichen Intelligenz. Systeme des maschinellen Lernens können eigenständig aus Erfahrungen und Daten lernen und basierend auf diesen früheren Erfahrungen ohne direkte Programmierung Entscheidungen treffen. Dabei Ist Deep Learning ein Teilgebiet des maschinellen Lernens und umfasst die Maschine-Learning-Systeme, die menschliche Lernverhalten mittels großer Datenmengen nachahmt, und aus zahlreichen Schichten besteht, wobei das Hauptelement des Netzes das Neuron ist. [6]

In dieser Arbeit werden verschiedene Deep-Learning-Strukturen verwendet. Daher werden diese Strukturen in den folgenden Abschnitten beschrieben und ihre theoretische Funktionsweise und Grundlagen verdeutlicht.

2.1.2 **Die wichtigsten Deep-Learning-Strukturen**

2.1.2.1 **Vollständig verbundene tiefe Netzwerke (FC-Netze)**

Diese Art neuronaler Netze ist weit verbreitet und wurde für zahlreiche Anwendungen wie Klassifizierungs- und Regressionsaufgaben eingesetzt. Der Hauptvorteil dieser Netzwerke besteht darin, dass sie Strukturunabhängig sind. das bedeutet, dass es egal ist, wie die Eingabe aussieht oder woraus sie besteht (Bilder oder zahlen oder was anders).

Ein vollständig verbundenes neuronales Netzwerk besteht aus einer Reihe vollständig verbundener Schichten, wobei eine vollständig verbundene Schicht eine Funktion von $\mathbb{R}^m$ zu $\mathbb{R}^n$ ist. [8] Dabei hängt Jeder der Ausgänge von jedem Eingang ab. In Abbildung 2-4 wird ein vollständig verbundenes Netzwerk dargestellt. Mathematisch wird die Beziehung zwischen der Eingabe und Ausgabe wie folgt berechnet:

$$Y= \begin{matrix} \sigma\left(\mathrm{w}_{1,1}\ \mathrm{x}_1 \ldots + \cdots + \cdots\ w_{1,m}\ x_1\right) \\ . \\ . \\ \sigma\left(\mathrm{w}_{\mathrm{n},1}\ \mathrm{x}_1 \ldots + \cdots + \cdots\ w_{n,m}\ x_m\right) \end{matrix} \qquad (2\text{-}1)$$

Hier wird nicht auf die mathematische Beschreibung des Netzes eingegangen. Es ist aber wichtig zu wissen, dass **σ** eine nichtlineare Funktion (Sigmoid-Funktion) ist und wird als Aktivierungsfunktion bezeichnet. Wi,j sind dabei die Gewichtungen (oder trainierbare Parameter), die während des Trainierens angepasst werden.

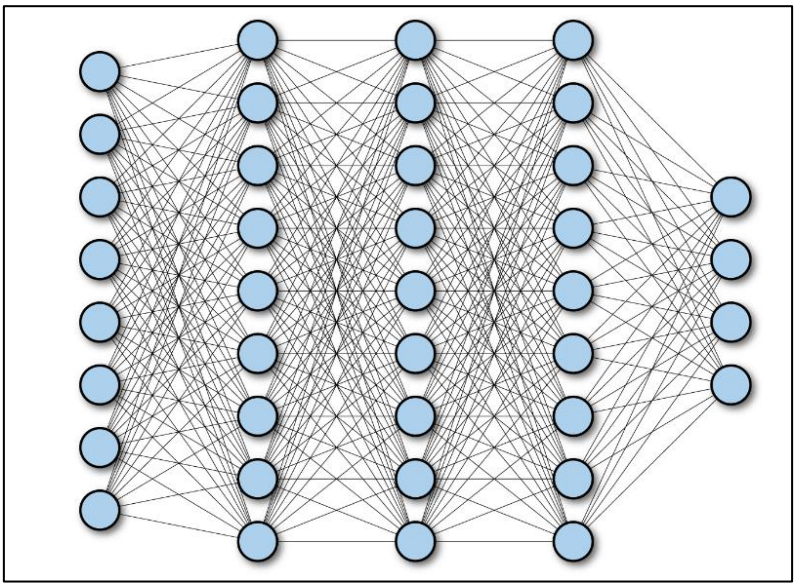


Abbildung 2-4: Vollständig verbundene Netzwerke [8]

2.1.2.2 Faltungsnetze CNN

Die Funktionsweise dieser neuronalen Netze ahmt die Funktionsweise des Sehens und der Interpretation von Bildern im menschlichen und tierischen Gehirn nach.

Wichtige Experimente und Studien zeigten, dass viele Neuronen im visuellen Cortex des Gehirns einen kleinen lokalen Wahrnehmungsbereich haben, also nur auf visuelle

Stimulationen in einem begrenzten Bereich des gesehenen Bildes reagieren. Die Neuronen, die unterschiedlichen Wahrnehmungsbereiche des Bildes abdecken, können einander überlappen und die Wahrnehmung des gesamten Bildes realisieren. Beispielsweise wurde in diesen Experimenten nachgewiesen, dass einige Neuronen nur horizontale Linien in einem bestimmten Wahrnehmungs-bereich wahrnehmen und darauf reagieren können, andere nur auf Linien mit anderer Ausrichtung. [4]

Andere Neuronen in höheren Ebenen reagieren auf komplexere Muster, die aus einfachen Mustern zusammengesetzt sind. In anderen Worten wird das betrachtete Bild im Gehirn in kleine Wahrnehmungsfelder unterteilt. Gleichzeitig Reagieren Neuronen nur auf bestimmte Formen in diesen Wahrnehmungsfelder. Das bedeutet, dass das Gehirn das Bild in kleine einfachere Bestandteile zerlegt. Dasselbe Prinzip hat sogenannte Faltungsnetzwerke inspiriert, die in vielen ML-Anwendungen und Erkennungsmodellen verwendet werden. Abbildung 2-5 zeigt das Prinzip von Faltungsnetzen. [4]

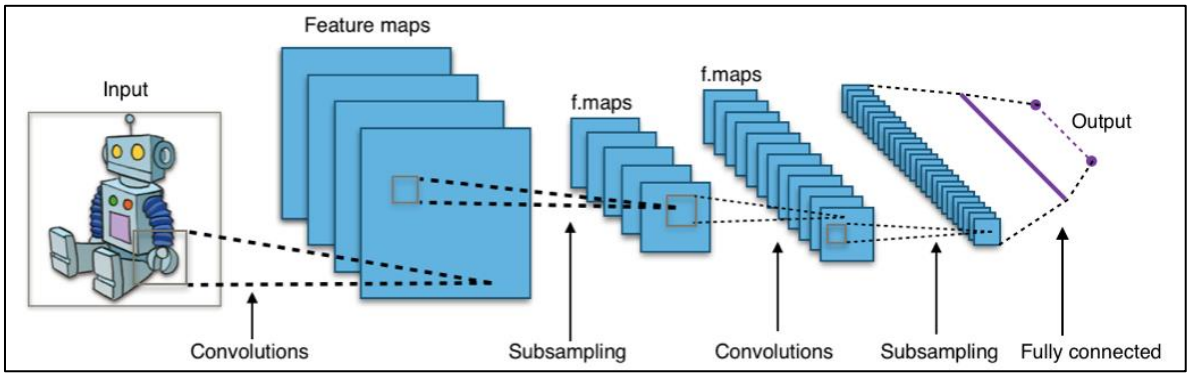


Abbildung 2-5: Die Funktionsweise der Faltungsnetze CNN [9]

Das Erste CNN-Schicht ist nicht mit jedem Pixel der Eingabebild verbunden, sondern nur mit Neuronen innerhalb eines Wahrnehmungsbereichs. Dabei konzentriert das Netz nur auf die Merkmale in diesem Kleinen Feld. Auf diese Weise sind Neuronen von der zweiten Schicht nur mit Neuronen in einem kleinen Rechteck der vorigen Schicht verbunden. Diese Hierarchie ermöglicht dem Netz, die Merkmale und Strukturen aus dem Bild zu extrahieren. [4]

Jedes Neuron in einer Bestimmten Schicht ist mit der Ausgabe aller in dem Wahrnehmungsfeld befindlichen Neuronen der vorigen Schicht verbunden. Im Wahrnehmungsfeld geschieht eine mathematische Operation oder sogenannte

Faltung zwischen dem Ausgang der vorigen Schicht und einem Filter (oder Kernel). Dieser Filter hebt Bestimmte Merkmale oder Linien in dem vorigen Bild hervor und unterdrückt andere Einzelheiten des Bildes. Ein Beispiel dafür ist der horizontale oder vertikale Sobel-Filter (Kantendetektor). Dieser Filter hebt die horizontalen oder vertikalen Kanten eines Bildes hervor und unterdrückt andere Bereiche des Bildes. Abbildung 2-6 zeigt einen horizontalen Sobel-Filter.

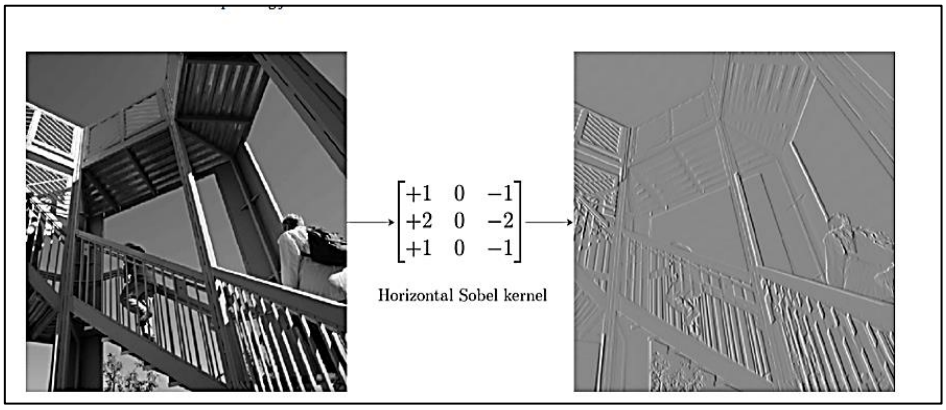


Abbildung 2-6: Die Anwendung eines horizontalen Sobel-Filters auf ein Bild [10]

Die Anwendung des Filters auf dem gesamten Bild muss ein zweidimensionales Bild mit gleicher Höhe und Breite ergeben. Um dieses Ergebnis zu erhalten, muss um die Eingabebilder herum mit Nullen ergänzt werden. Dieses Verfahren nennt man Zero-Padding (Abbildung 2-7).[1] Es gibt eine weitere Möglichkeit, bei der die Bildelemente, die am Rand der Eingabe liegen, ignoriert werden. In diesem Fall kann auf Zero-Padding verzichtet werden.

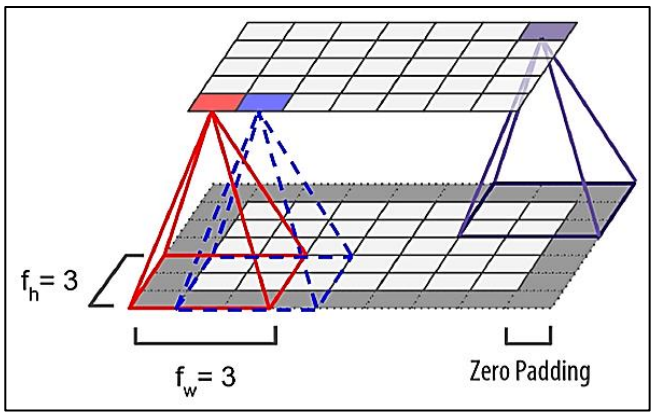


Abbildung 2-7: Zero-Padding [4]

[1] Wenn der Filter auf den äußersten Pixel des Eingangs platziert ist, deckt ein Teil des Filters keine Bildelemente ab. Deshalb platzieren wir Nullen um das Eingabebild herum.

Manchmal werden Faltungsschichten mit unterschiedlichen Ausgabegrößen benötigt, wie beispielsweise wenn die Ausgabe der ersten Schicht die Größe M **x** N hat, und die Eingabe der zweiten Schicht die Größe M/2 x N hat, dann muss ein weiteres Verfahren angewendet werden. Dieses Verfahren heißt (Stride-Verfahren oder auf Deutsch Schrittweite). Dabei wird der Abstand zwischen zwei aufeinanderfolgenden Wahrnehmungsfeldern vergrößert. [4]

Angenommen, die Größe der ersten Schicht ist 10x10, und die Eingabe der zweiten Schicht hat die Größe 5x10, dann können diese zwei Schichten miteinander verbunden werden, Indem dem Parameter Stride (Schrittweite) der Wert 2x1 zugewiesen wird. Die Abbildung 2-8 verdeutlicht dieses Verfahren und die Schrittweite des Kernels (Wahrnehmungsfeld).

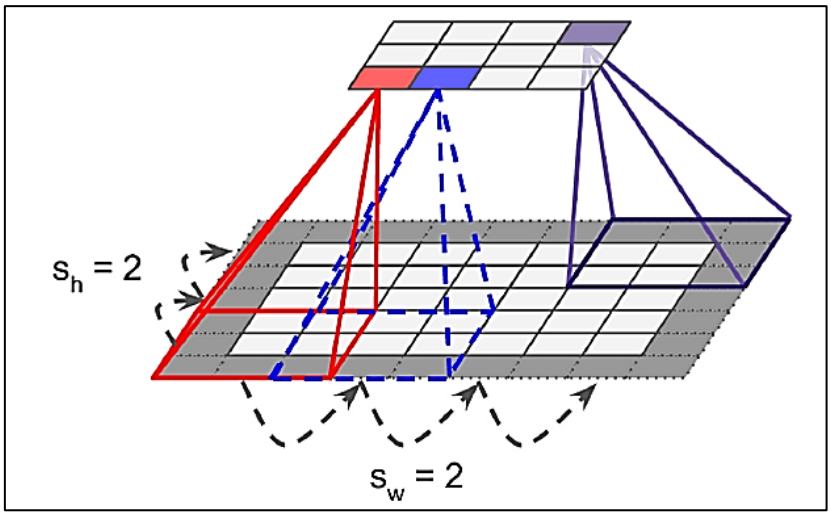


Abbildung 2-8: Die Schrittweite des Faltungsfilters [4]

Die CNNs werden sehr häufig für Objekterkennung oder Klassifizierung zweidimensionaler Daten eingesetzt. Ebenfalls werden diese Strukturen im Laufe der Arbeit häufig für Spracherkennungsaufgaben sowie die Klassifizierung von Bildern verwendet.

Da in dieser Arbeit diese neuronalen Netze auf dem Microcontroller implementiert werden müssen, wo der Speicher knapp ist und die Belastung des Microprozessors von großer Bedeutung ist, werden nachfolgend einige Nachteile der CNNs aufgelistet.

1. Faltungsnetze haben eine große Zahl an Hyperparameter, die man während der Modellerstellung einstellen muss. Dieser Prozess ist zeitintensiv und

aufwändig. Dabei werden durch Experimentieren die perfekten Werte dieser Hyperparameter gefunden. Diese Hyperparameter sind die Zahl der Filter, die Höhe und Breite des Filters, die Schrittweite (oder Stride) und das Zero-Padding-Verfahren.

2. Die CNNs sind sehr rechenintensiv und benötigen einen großen Speicher. Das macht die Implementierung auf dem Microcontroller schwierig. Beispielsweise hat ein CNN-Layer mit 200 Filter der Größe 5x5 und Eingabebilder mit der Auflösung 96x96 ungefähr (5x5+1) x200 = 5200 trainierbaren Parameter. Jeder der 200 Feature Maps enthält 96x96 Neuronen. Jeder dieser Neuronen ist wiederum mit 5x5 Gewichtungen verbunden. Das ergibt 200 x 25 x 96 x 96 = 46 Millionen Gleitkommamultiplikationen. Die Ausgabe der Schicht wird normalerweise als Float32 dargestellt. Das ergibt 200x96x96x32 = 59 Millionen Bits (7.5 Mbyte). Das ist nur für einen Datenpunkt. Während des Trainierens werden mehrere Datenpunkte (zum Beispiel 100 oder sogenannte Batch von Daten) in das Netz geladen, was bedeutet, dass 750 MByte Speicher für ein Layer benötigt wird. Während der Inferenz (wenn eine Vorhersage getroffen werden muss) benötigt das Netz 7.5 MByte Speicher. Dieser Speichergröße überteigt den normallerweise auf dem Mikrocontroller zur Verfügung stehenden Speicher, wo die Speichergröße einige 100 K Bytes beträgt. [4]

Aus den bisher diskutierten Eigenschaften der Faltungsnetze kann eine Schlussfolgerung gezogen werden, dass die Faltungsnetze einen sehr großen Speicher, besonders beim Trainieren beanspruchen und sehr Rechenintensiv. Später in dieser Arbeit werden einige Techniken verwendet, um die Größe des benötigten Speichers und die Anzahl der Rechenoperationen zu reduzieren, damit diese mächtige Art der neuronalen Netze auf dem Microcontroller implementiert werden kann. Trotzdem bleibt der Speicherverbrauch und die Auslastung der CPUs bei CNNs geringer und effizienter im Vergleich mit FC-Netzen, was die Faltungsnetze in vielen Anwendungsfällen bevorzugt macht.

2.1.2.3 Pooling Layer

Die Pooling-Schicht kommt häufig nach einem CNN-Layer zum Einsatz. Der Zweck dieser Schicht ist die Verkleinerung der Ausgabedimensionen, was dazu führt, die Rechenlast, den Speicherbedarf und den Energieverbrauch stark zu verringern. In

Abbildung 2-9 ist zu sehen, wie die Ausgabe von dieser Schicht berechnet wird. Die Pooling-Schicht hat ähnliche Parameter wie CNNs aber keine Trainierbare Gewichtungen. Diese Parameter sind Stride (Schrittweite), Filterhöhe, Filterbreite, und Zero-Padding.

Es ist hier erwähnenswert, dass es zwei Arten von Pooling-Layers gibt, Average- oder Mean-Pooling und Max-Pooling. Hierbei ist die Maxpooling-Schicht häufiger verwendet.[2]

Wie es der Abbildung 2-9 entnommen werden kann, wird nur die größte Wert in jedem Max-Pooling-Filter an die nächste Schicht übergeben und alle übrigen Eingaben werden verworfen. Die Größe der Ausgabe dieser Schicht ist dabei viel kleiner als die Größe der Eingabe, da nur die relevantesten Werte aus der vorigen Schicht an die Nächste Schicht weitergeleitet werden.

Der Name von Mean-pooling ist selbsterklärend. Diese Pooling-Schicht ist ein Bisschen komplexer als die Max-Pooling-Schicht und belastet den Prozessor mit größerem Rechenaufwand. Denn In dieser Schicht wird den Mittelwert aller in dem Filter befindlichen Bildelementen berechnet und an die nächste Schicht weitergegeben. In Abbildung 2-9 wird die Ausgabe der Max-Pooling-Schicht mit der Ausgabe der Mean-Pooling-Schicht verglichen.

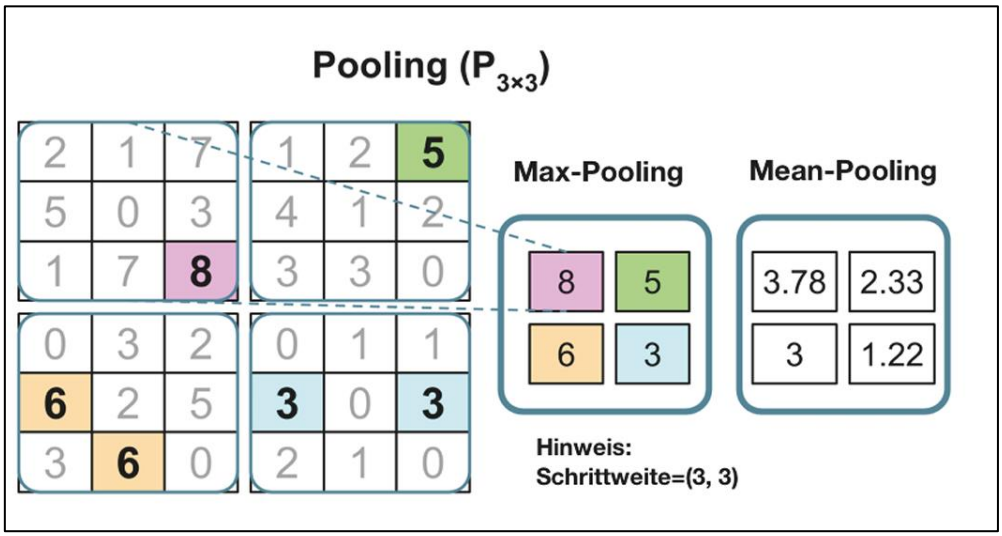


Abbildung 2-9: Anwendung von Pooling-Filter auf das Eingabebild [11]

[2] Die Mean-Pooling-Schicht ist auf dem Mikrocontroller noch nicht unterstützt. Dies ist der Fall, wenn das verwendete Modell quantisiert ist (int8/uint8) .

Jetzt stellt sich die Frage, warum sind die Pooling-Schichten in einem CNN-Netz so nützlich? Die Pooling-Schicht bringt mit sich viele Vorteile, die nachfolgend aufgelistet sind:

1. Das Hinzufügen der Polling-Schicht direkt am Ausgang der CNN-Schicht verhindert sogenannte Over-Fitting oder Überanpassung.
2. Die Verwendung von einer Polling-Schicht verringert stark die Belastung der CPU. Dieser Einfluss ist von großer Bedeutung, da das neuronale Netz auf dem Microcontroller implementiert werden muss, wo der Programspeicher und die Rechenfähigkeit sehr begrenzt sind.
3. Die Pooling-Schicht verbessert die Allgemeinbarkeit des Modells.
4. Da diese Schicht die unwichtigen und kleinen Änderungen der Pixelwerte vernachlässigen, wird die Extraktion der Merkmale weniger anfällig für das Rauschen. [11]

2.1.2.4 Die Regularisierung

Regularisierung ist eine wichtige Technik, die nur beim Trainieren des Modells aktiviert wird und dient der Vermeidung der Überanpassung (Over-fitting) während des Trainings. Hierbei werden alle Netzgewichte zum Lernen gezwungen, indem ein vordefinierter bestimmter zufälliger Anteil der Gewichte während des Trainings abgeschaltet wird. Der Einsatz dieser Technik in einem Modell führt dazu, die Allgemeinbarkeit des Modells zu verbessern. Bei der Inferenz des Modells wird es keine Regularisierung geben, was bedeutet, dass alle Netzgewichte an den Vorhersagen beteiligen werden. In dieser Arbeit wird später ein Dropout-Schicht als Regularisierung-Schicht verwendet, wobei Dropout ein effizientes Verfahren zur Regularisierung ist.

2.1.2.5 Die Aktivierungsfunktionen

Die Aktivierungsfunktion wird verwendet, um die Ausgabe eines Netzes oder einer Schicht zu berechnen. Es gibt im Allgemeinen lineare und nichtlineare Aktivierungsfunktionen, die in einem neuronalen Netz zu finden sind.

Die Meistverwendeten Aktivierungsfunktionen sind :

1. Sigmoid-Funktion. Die Ausgabe dieser Funktion liegt zwischen Null und Eins. Daher wird diese Funktion insbesondere verwendet, wenn die Ausgabe oder die Vorhersage des Netzes als Wahrscheinlichkeiten dargestellt werden müssen. Der Grund dafür ist, dass die Wahrscheinlichkeiten ebenso Werte zwischen Null und Eins annehmen. [12]
2. Tanh oder hyperbolic tangent Aktivierungsfunktion. Dabei liegt die Ausgabe zwischen –1 und 1.
3. RELU-Funktion oder auch Gleichrichter-Funktion. Diese Funktion wird am häufigsten verwendet und wird später in dieser Arbeit ebenso häufig benützt. Dabei werden alle negativen Werte der Eingabe in Null umgesetzt. Die Abbildung 2-10 zeigt diese drei Aktivierungsfunktionen.

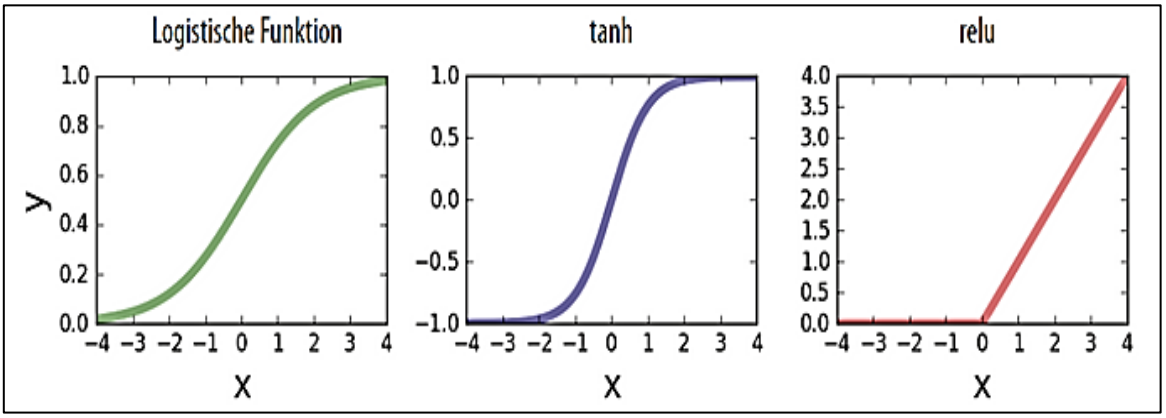


Abbildung 2-10:Sigmoid-Funktion(Links),Tanh-Funktion(Mitte) und Relu-Funktion (rechts). [12]

4. Softmax-Funktion oder normalisierte Exponentialfunktion. Diese Funktion ist sehr wichtig, wenn die Ausgabe des Netzes mehrere Labels enthält und wird normalerweise auf die Ausgabe letzter Schicht in einem Modell angewandt (Abbildung 2-11). Die Ausgabe dieser Funktion ist die Wahrscheinlichkeits-verteilung auf alle Labels, wodurch die Summe aller Labels-Wahrscheinlichkeiten 1 ist. Diese Funktion wird mit den folgenden Gleichungen beschrieben. [13]

$$\boldsymbol{\sigma}: \boldsymbol{R^k} \rightarrow \left\{ \boldsymbol{z} \in \boldsymbol{R^k} \mid \boldsymbol{z_i} \geq \boldsymbol{0}, \sum_{\boldsymbol{i=1}}^{\boldsymbol{k}} \boldsymbol{z_i} = \boldsymbol{1} \right\} \qquad (2\text{-}2)$$

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}} \qquad \text{für } i = 1 \dots k \qquad (2\text{-}3)$$

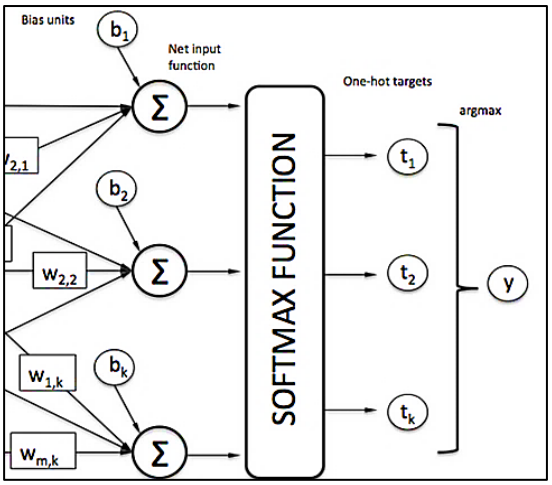


Abbildung 2-11: Die Anwendung der Softmax-Funktion auf die Ausgabe der letzten Schicht. [13]

2.1.2.6 Rekurrente neuronale Netze RNN

Diese Netze sind eine der wichtigsten Strukturen im Deep-Learning und wird darauf später in Kapitel 5 eingegangen.

2.2 Relevanz von Microcontrollern

Mikrokontroller sind in der Regel kleine Rechensysteme, die in ein elektronisches System eingebettet werden können. Diese kleinen Rechner sind in der Lage kleine Programme und Rechenaufgaben, die kleinen Speicher und begrenzten Rechenaufwand erfordern, auszuführen. Die größten Vorteile der Mikrokontroller sind hierbei der sehr geringe Energieverbrauch und die niedrigen Kosten. Daher ist es von großem Interesse, wenn Maschine-Learning-Algorithmen auf dem Mikrokontrollern ausgeführt werden könnten. Diese winzigen intelligenten Systeme könnten in vielen Geräten des Alltags inklusive Haushaltgeräte und Internet of Things-(IoT) Anwendungen integriert werden, wodurch die Interaktion zwischen dem Benutzer und der Maschine erleichtert wird.

Cloud-basierte Systeme existieren bereits und sind in vielen Anwendungen zu finden. Trotzdem bleibt das Interesse an smarte Anwendungen auf dem Mikrokontroller unverändert groß, und zwar aus folgenden Gründen:

1. Die bereits existierenden Systeme verlangen eine Internetverbindung mit großer Geschwindigkeit und Bandbreite. Denn das intelligente System liegt nicht in dem Gerät, sondern in der Cloud (Abbildung 2-12).
2. Es besteht lange Zeitverzögerung und Latenzen, weil die Verarbeitung von den Informationen nicht auf dem Gerät, sondern in Cloud stattfindet. Dies ist zu merken, wenn beispielsweise einen Sprachassistenten wie Alexa, Siri, Google, Cortana und Bixby verwendet wird.
3. In manchen Fällen möchte man zum Beispiel ein Objekterkennungssystem in einem abgelegenen Ort (im Wald oder In der Wüste), wo keine Internetverbindung zur Verfügung steht, einsetzen. Dort könnten solche Systeme für die Überwachung von Gebieten oder Tieren eingesetzt werden.
4. Aus Sicherhitgründen kaufen viele Menschen keine Produkte wie Amazon-Echo, die immer mithört, wobei die Informationen das Gerät verlassen. Deshalb könnte die Implementierung auf dem Mikrocontroller zum Schutz der Privatsphäre beitragen und viele Befürworter und Kunden gewinnen.

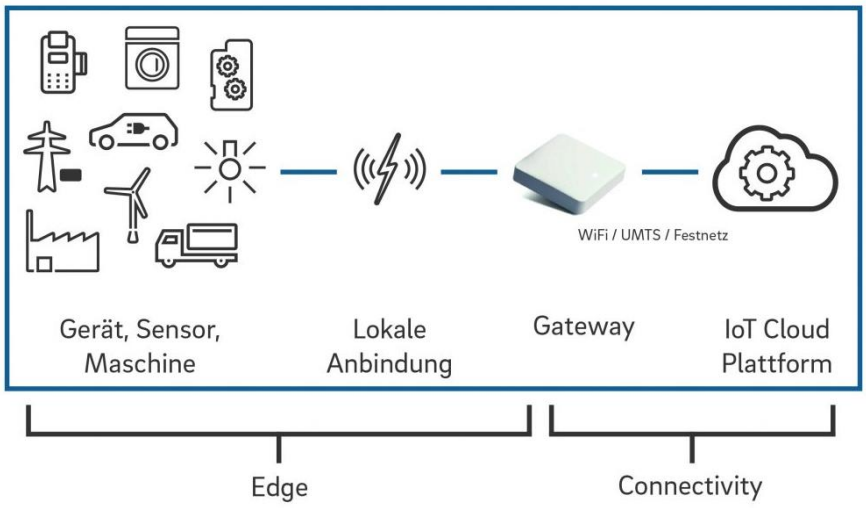


Abbildung 2-12: Der Datenfluss bei IoT-Anwendungen [14]

2.2.1 Herausforderungen bei der Implementierung auf Mikrokontrollern

Wie es bisher bekannt worden ist, ist die Implementierung der Maschine-Learninig-Algorithmen auf Mikrocontrollern sehr wichtig, aber leider nicht so einfach. Denn Mikrocontroller sind kleine Rechensysteme, die über kleinen Programmspeicher (meistens kleiner als 100 KByte) und können begrenzte Anzahl von arithmetischen

Operationen per Sekunde (weniger als 10 Millionen arithmetische Operationen per Sekunde) abarbeiten. Gleichzeitig sind Maschine-Learning-Algorithmen sehr Rechenintensiv und benötigen großen Speicher. Außerdem haben Mikrokontroller kein Betriebssystem oder File-System, das auf normalen Rechnern zu finden ist. Diese Herausforderungen machen die Implementierung auf Mikrocontrollern schwierig.

Da diese Branche des maschinellen Lernens noch am Anfang ist, gibt es viele Probleme, die mit der verwendeten API zu tun haben (In unserem Fall Tensorflow für Mikrokontroller). Diese und weitere Probleme können wie folgt zusammengefast werden:

1. Unterstützung für nur eine begrenzte Teilmenge der Tensorflow-Operationen. Beispielsweise sind die RNN-Netze noch nicht unterstützt.
2. Unterstützung für eine begrenzte Teilmenge der Mikrocontroller, Wobei ML-Modelle nur auf einigen Hardware-Plattformen ausführbar sind.
3. Low-Level-C++-API, die eine manuelle Speicherverwaltung erfordert. [15]
4. Eine große Zahl an Abhängigkeiten und mehrere Programmiersprachen, mit denen man arbeiten muss. Beispielsweise muss man das Modell in Python trainieren und es in ein in C++ geschriebenes Programm integrieren.
5. On-Device-Training wird nicht unterstützt. Man muss das Model auf einem externen Rechner trainieren dann das trainierte Modell auf den Mikrocontroller übertragen. Das führt zu Problemen bei der Aufnahme der Informationen mit den Sensoren auf dem Board. Denn die mit Mikrocontrollersensoren aufgenommenen Daten unterscheiden sich in vielen Fällen von den Daten, die in das Modell beim Trainieren eingespeist worden sind. [15]
6. Tensorflow-API ändert sich ständig und ist noch nicht stabil. Das bedeutet, dass einige Operationen in den neuen Versionen veraltet sind und andere hinzugefügt werden.
7. Das Trainieren von ML-Modellen und die Verarbeitung von Daten erfordern einige Packages und Bibliotheken, die nur auf Linux-Systemen installiert werden können.[3] Deshalb empfehlt es sich, ein auf Linux basiertes System zu benutzen.

[3] Das Build-Management-Tool Make wird benötigt. Die Version, die auf Windows installiert werden kann, ist aber alt.

8. Jeder Mikrocontroller hat einen eigenen Entwicklung-Framework. Dies führt dazu, dass für jede Plattform einen C-Programm geschrieben werden muss.[4]

Die Abbildung 2-13 zeigt ein Beispiel für solche Probleme, die bei Implementierung auftauchen könnten. Hierbei wird das Modell mit zwei Tensorflow-Versionen Tensorflow 2.3 und Tensorflow 2.2 quantisiert. Obwohl der Code derselbe war und dieselbe Quantisierungsart verwendet wird, ist das Ergebnis nicht identisch. Der Unterschied liegt daran, dass beim ersten TFlite-Modell die Quantisierung des Ein- und Ausgangs innerhalb des Modells stattfindet. Das bedeutet, dass Quantize- und Dequantize-Schichten auf dem Mikrocontroller benötigt werden. Hingegen wird die Quantisierung des Ein- und Ausgangs beim zweiten TFlite-Modell (rechts) bereits während der Konvertierung angewendet.

Das Entdecken von solchen Problemen könnte zeitaufwändig sein. Dies wird dadurch verstärkt, dass es keine explizite Erwähnung einiger Probleme in der Literatur gibt oder auf der Webseite von Tensorflow.

[4] Tensorflow-Operationen ändern sich nicht auf den verschiedenen Mikrocontrollern. Was sich von einem Mikrocontroller zu einem anderen ändert, ist der Code zur Aufnahme der Sensordaten.

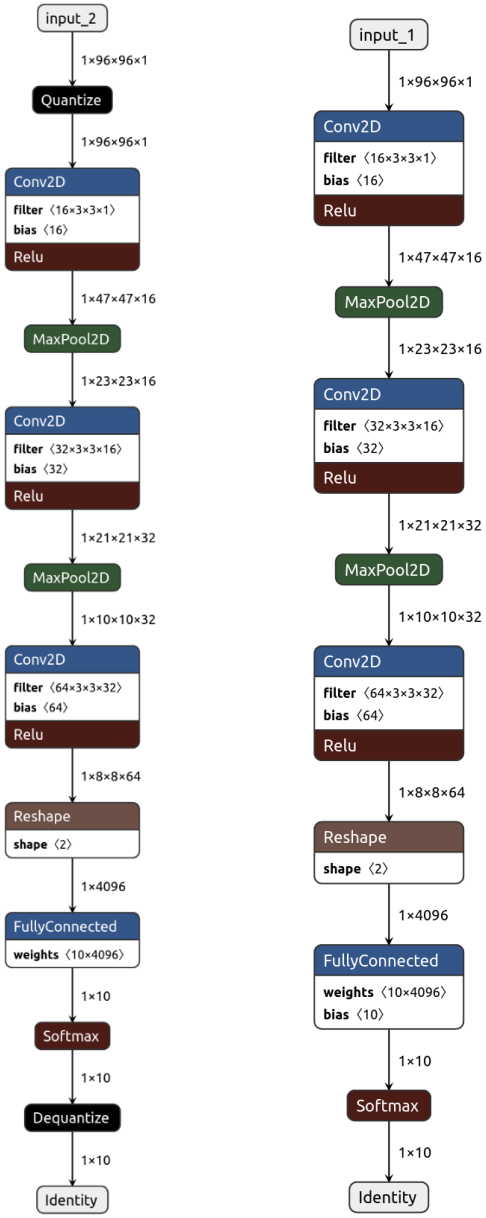


Abbildung 2-13: Ein Modell wurde mit zwei unterschiedlichen Tensorflow-Versionen konvertiert.

2.3 Die Hardware

Für das Training und Ausführung der Python-Skripte wird ein Rechner benötigt, der die Programmierumgebung sein wird. Dieser Rechner muss kein leistungsstarker Rechner sein, weil die anspruchsvollsten Aufgaben, die nicht auf einem normalen Rechner zu lösen sind, können in der Cloud trainiert werden. Für das Trainieren der Spracherkennungsmodelle wird ein Rechner mit mindestens 8 GByte -RAM benötigt. Für das Trainieren der Objekterkennungsmodelle ist ein größerer Arbeitsspeicher etwa 13 GByte erforderlich.

Das Modell wird anschließend auf das eingebettete Board übertragen. Daher werden mehrere USB-Anschlüsse und mehrere Adapter benötigt, je nachdem auf welches Board das Modell übertragen werden muss.

Im Laufe der Arbeit werden mehrere ML-Modelle in verschiedenen Anwendungsgebieten auf den verschiedenen Mikrocontrollerboards ausgeführt. Anschließend wird das Hauptprojekt, das mehrere Module benötigt, entworfen. Hierfür werden mehrere Hardware-Komponenten und Boards benötigt, die nachfolgend vorgestellt sind.

2.3.1 **ESP-Eye**

Dieses mächtige Board (Abbildung 2-14) ist häufig in dieser Arbeit für Spracherkennungsanwendungen eingesetzt. Die wesentlichen Eigenschaften dieses Boards sind:

1. Der Microprozessor auf dem Board ist der mächtige Chip ESP32(Dual-core Tensilica LX6 Mikroprozessor).
2. Die maximale Taktfrequenz beträgt 240 MHz.
3. Integrierte Kamera OV2640 mit der Auflösung 2 MPixel.
4. Integriertes Mikrophon
5. Braucht kein Adapter (hat einen Micro-USB Anschluss).
6. Das Board hat keine GPIO-Pins nach außen (Ein-/Ausgang).
7. Integrierter 802.11 b/g/n WiFi Transceiver
8. Integriertes Dual-Mode-Bluetooth (BLE und Classic)
9. Hat keinen Beschleunigungssensor

Abbildung 2-14: ESP-EYE

2.3.2 **ESP-Camera**

Dieses Board (Abbildung 2-15) hat die folgenden Eigenschaften:

1. Der Microprozessor auf dem Board ist ebenso der mächtige Chip ESP32(Dual-core Tensilica LX6 Mikroprozessor).
2. Integrierte Kamera OV2640 mit der Auflösung 2 MPixel.
3. Das Board hat GPIO-Pins nach außen.
4. MicroSD Card Slot auf dem Board.
5. Bis zu 240MHz Taktfrequenz.
6. Integrierter 802.11 b/g/n WiFi Transceiver.
7. Integriertes Dual-mode Bluetooth (BLE und Classic).
8. Braucht einen Adapter wie FTDI 232(hat keinen USB-Anschluss).

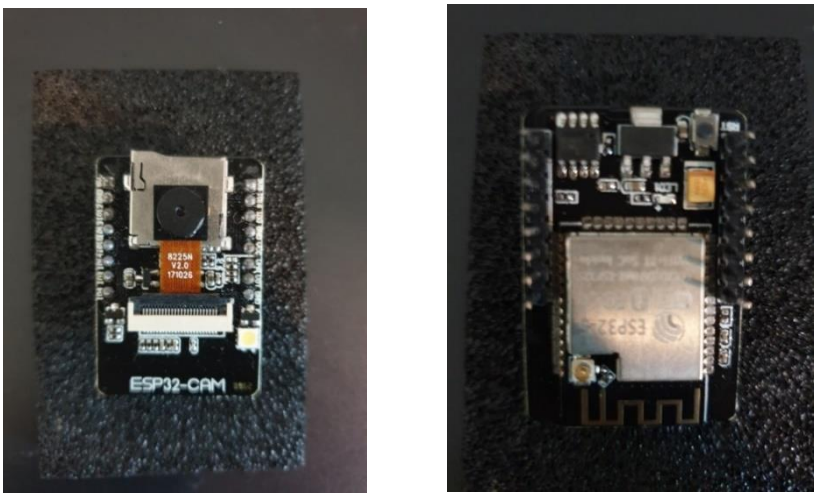


Abbildung 2-15 : ESP-Camera (AI-Thinker)

2.3.3 **NodeMCU ESP-32 Entwicklungsboard**

Dieses Board (Abbildung 2-16) wird nur im Hauptprojekt zur Steuerung einer Lampe eingesetzt und hat die folgenden Eigenschaften:

1. Der Microprozessor auf dem Board ist der mächtige Chip ESP32(Dual-core Tensilica LX6 Mikroprozessor).
2. Bis zu 240MHz Taktfrequenz
3. Braucht keinen Adapter (hat einen Micro-USB Anschluss).
4. Das Board hat GPIO-Pins nach außen (Ein-/Ausgang).
5. Integrierter 802.11 b/g/n WiFi Transceiver
6. Integriertes Dual-mode-Bluetooth (BLE und Classic)

Abbildung 2-16: ESP-32 NodeMCU

2.3.4 **Arduino Nano sense ble**

Dieses kompakte Board enthält zahlreiche Sensoren, die für viele Projekte eingesetzt werden können und wird mithilfe von Arduino-IDE programmiert. es verfügt über Beschleunigung-, Temperatur-, Farb-, Feuchtigkeit-, und Lichtsensor. Zudem verfügt über ein Mikrophon, das für die Spracherkennungsanwendungen erforderlich ist, GPIO-Pins nach außen und kann über eine Bluetooth-Verbindung mit der Außenwelt kommunizieren. Der Mikroprozessor auf dem Board ist der leistungsstarke ARM Cortex-M4 mit 64 MHz als Taktfrequenz. Ein Micro USB-Buchse zur Programmierung des Boards ist außerdem vorhanden und benötigt damit keinen Programmierer. Abbildung 2-17 zeigt das Board Arduino Nano BLE Sense.

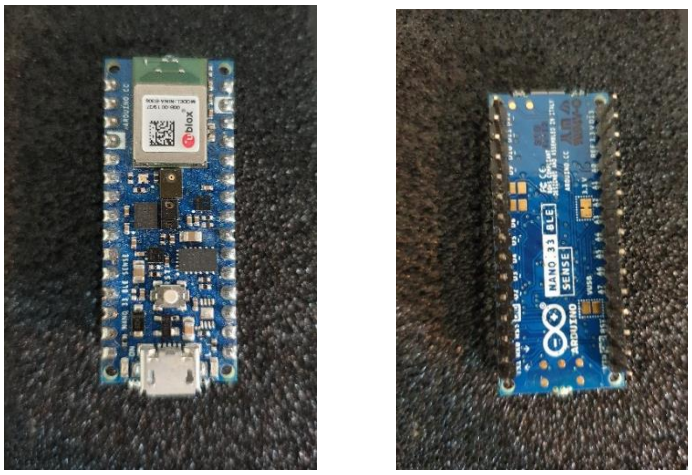


Abbildung 2-17: Arduino Nano Ble Sense

2.3.5 **Sparkfun Edge Apollo3-Blue**

Dieses Board (Abbildung 2-18) wurde mit Unterstützung von Tensorflow hergestellt, wie auf der Rückseite des Boards zu sehen ist. Das Board enthält ein Mikrophon und Beschleunigungssensor, die für ML-Anwendungen nötig sind. Außerdem hat es eine Schnittstelle für die Kamera HM01B0 Himax. Der Prozessor auf dem Board ist ebenfalls ARM Cortex M4 und die maximale Taktfrequenz beträgt 96 MHz. Der Stromverbrauch ist dabei sehr gering (etwa 6uA/MHz). Auf der Rückseite des Boards befindet sich ein Batteriehalter, wobei eine Batteriezelle das Board für bis zu 10 Tage betreiben kann.

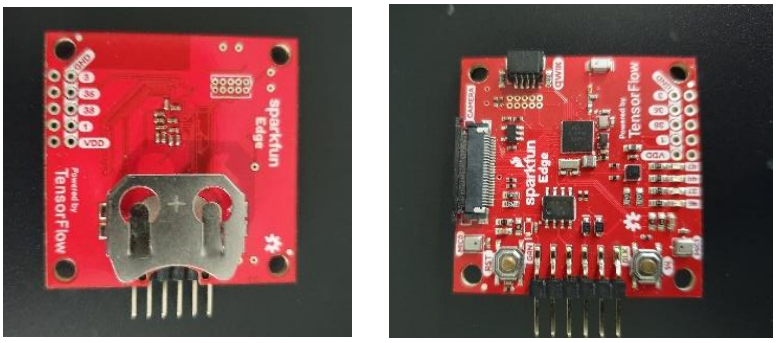


Abbildung 2-18: Sparkfun Edge

2.3.6 **Der Adapter FTDI232**

Dieser Adapter (Abbildung 2-19) ist ein USB zu URT Umwandler, und wird zur Programmierung einiger ESP-Boards sowie das Sparkfun-Edge-Board benötigt.

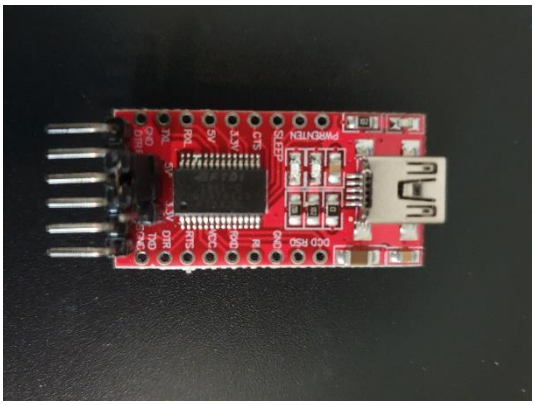


Abbildung 2-19: FTDI232 Board

2.3.7 **Die Kamera: HM01B0 Himax**

Für die Objekterkennungsanwendungen auf dem Spark-Fun-Board ist diese Kamera erforderlich.

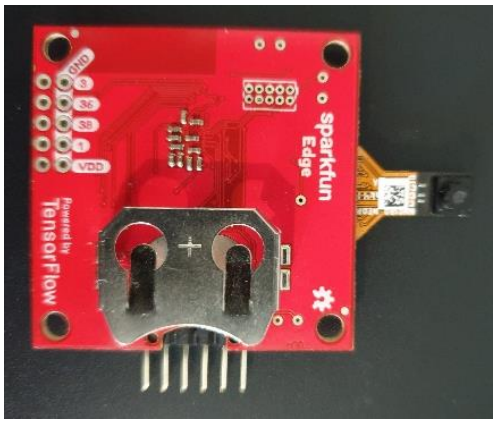


Abbildung 2-20:Sparkfun-Edge mit der Kamera

2.4 Die Software

Wie erwähnt, wird für die Projekte ein Computer und ein installiertes Operating System benötigt (Beispielsweise UBUNTU 20.04.1 LTS). Für das Trainieren von Modellen wird ein Python-Code-Editor verwendet. Dieser könnte beispielsweise Spyder oder Notizbuch wie Jupyter oder auch Google-Colab sein. Als C++-Code-Editor kann VS-Code installiert werden. Im Laufe der Arbeit sind zudem zahlreiche Packages zu installieren. Daher lohnt es sich, Conda zu benutzen, damit die Packages und die Versionen besser verwaltet werden können.

Bei den Projekten in dieser Arbeit geht es um Maschine Learning unter Verwendung von Tensorflow API und Keras API. Daher müssen die beiden APIs installiert werden. Hier ist zu beachten, dass mehrere Versionen von Tensorflow benötigt werden. Deshalb ist es sehr nützlich, wenn die Packages in Environments isoliert werden.

Je nachdem welches Board programmiert werden muss, ist eine Development Framework für dieses oder jenes Board erforderlich. Für Arduino Boards wird Arduino IDE verwendet, während für ESP-Boards ESP-IDF Framework benötigt wird.

2.5 Der Arbeitsablauf

Nachfolgend wird der Arbeitsfluss Schritt für Schritt erläutert und auf die wesentlichen Prozesse bei der Implementierung auf Mikrocontrollern eingegangen. Den Arbeitsablauf zeigt die Abbildung 2-22 .

2.5.1 Das Problem analysieren

Als erstes muss in diesem Schritt untersucht werden, was das Modell tun muss und welche Daten behandelt werden müssen. Dabei muss überlegt werden, welche Merkmale aus den Daten extrahiert werden können, und versuchen abzuschätzen, welche Größe der Daten, die durch das Modell geführt werden müssen, am besten passt. [16]

Damit das ML-Modell zur Implementierung auf einem bestimmten Microkontroller taugt, lohnt es sich hier die Eigenschaften der Hardware zu untersuchen, beispielsweise welche Auflösung die Kamera hat, und wie hoch die Taktfrequenz des Prozessors ist.

2.5.2 Daten sammeln

In diesem Arbeitsschritt werden die Daten, die die Basis des Datensatzes darstellen, gesammelt. Hierfür wird zuerst nach Datensätzen gesucht, die bereits existieren. Wenn kein passender Datensatz zur Verfügung steht, dann muss ein Datensatz erstellet werden. Dieser Prozess ist nicht immer einwandfrei und manchmal zeitaufwändig. Denn Hunderte oder manchmal Tausende von Datenpunkten jeder Klasse müssen gesammelt werden. Dabei werden die Trainingsdaten aufgenommen oder in anderen Quellen gesucht. Dabei ist es sehr hilfreich, einige Techniken, wie die Erzeugung von synthetischen Daten, einzusetzen.

Es ist manchmal sehr schwierig einzuschätzen, wie viele Trainingsdaten benötigt werden, bis das Netz die Merkmale erlernen kann. Denn dies hängt von vielen Faktoren wie Beispielsweise der Ähnlichkeit zwischen den Klassen und der Komplexität der Daten ab. Der Datensatz, auf den das Modell trainiert werden soll, muss alle möglichen Daten, die am Eingang des Modells auftauchen können enthalten. [16] Außerdem wird in dieser Arbeit ein überwachter Lernalgorithmus verwendet, was bedeutet, dass den Daten Labels zugeordnet werden müssen.

2.5.3 Daten verarbeiten

In diesem Schritt müssen die Daten verarbeitet werden, damit sie sich für Eingabe des Modells passen können. Dieser Schritt kann eine oder mehrere der folgenden Vorverarbeitungsprozesse umfassen:

1. Augmentation von Daten. Dieser Prozess kann die Trainingsdatenmenge vergrößern und damit dazu führen, die Accuracy-Rate zu verbessern
2. Normalisieren von Daten. Dieser Schritt ist sehr wichtig, wenn die Werte der Datenpunkte im großen Wertebereich schwingen. Außerdem ist es nachgewiesen, dass die Accuracy-Rate verbessert wird, wenn die Werte, auf die das Modell trainiert wird, zwischen null und eins oder im kleinen Wertebereich liegen. [16]

2.5.4 Merkmale extrahieren

Je nachdem mit Welchen Daten gearbeitet wird, muss nach Merkmalen gesucht werden, die die Klassen am besten voneinander unterscheiden. Dieser Schritt hat den größten Einfluss auf die Performance und Accuracy des Modells. Dabei können die Merkmale entweder während des Trainings oder vor dem Training extrahiert werden. Der Unterschied zwischen den zwei Fällen liegt daran, dass im ersten Fall die Operationen zum Berechnen der Merkmale bei jedem Training wiederholt werden muss. Im zweiten Fall muss dieser Prozess hingegen nur ein Mal durchgeführt werden. Jede der zwei Methoden hat Vorteile und Nachteile. Die Vorteile im ersten Fall ist die Einfachheit, wobei keine zusätzlichen Programme erforderlich sind. Im zweiten Fall muss hingegen ein Programm für das Bilden und Speichern des Merkmalsdatensatzes geschrieben werden.

Die Implementierung von Deep-Learning-Modellen auf dem Mikrocontroller ist in vielen Fällen aufwändig. Das führt dazu, dass hochwahrscheinlich viele Modelle experimentiert werden müssen, bevor ein gutes Ergebnis erreicht wird, was eine Zeitverschwendung bedeutet, wenn bei jedem Training die Merkmale berechnet werden müssen. Aus diesem Grund empfehlt es sich einen Datensatz von den Merkmalen zu erstellen, der für das Trainieren von Modellen jedes Mal verwendet werden kann.

2.5.5 **Das Modell entwerfen**

Es gibt keine klaren Regeln, wie das Modell aussehen muss. Man kann aber durch frühere Erfahrungen und Experimentieren einschätzen, wie das Modell aussehen kann. Außerdem kann man für ein bestimmtes Modell aufgrund der Daten entscheiden, welche Strukturen für das zu lösende Problem am besten funktioniert (beispielsweise ein CNN-Netz ist besser als ein FC-Netz, wenn Bilder behandelt werden müssen). Im Allgemeinen hängt die Modellarchitektur strak von den Daten ab, die in das Modell eingespeist werden müssen. In vielen Fällen ist außerdem zu berücksichtigen, auf welche Hardware das Modell übertragen werden muss. Denn die Hardware bestimmt, wie groß das Modell sein darf und wie schnell das Modell ausgeführt werden kann.

2.5.6 **Das Modell trainieren**

Das Trainieren ist ein Prozess, wodurch das Modell erlernt, die richtigen Vorhersagen zu treffen. Beim Trainieren werden die Gewichte des Modells schrittweise angepasst, bis das Modell die größte Accuracy-Rate aufweist. Bei diesem Prozess werden die Trainingsdaten in das Modell eingespeist und der Ausgang des Modells wird berechnet. Anhand von den berechneten Ausgaben und den richtigen Labels werden dann die Gewichtungen entsprechend angepasst.

Die Initialisierung der Gewichte hat hierbei einen großen Einfluss auf das Ergebnis. Die Methode der Initialisierung kann aber beliebig ausgesucht werden. Es gibt viele Recherchen und Theorien über dieses Thema. Die beste Methode zur Initialisierung eines Modells, die in dieser Arbeit verwendet wird, heißt Xavier- oder Glorot. [11]

Es besteht auch die Möglichkeit, den Gewichtungen zufällige Werte zuzuweisen.[5]

Während des Trainierens kann das Training durch zwei Leistungsmetriken beobachtet werden. Diese Metriken sind Loss-Funktion und Accuracy-Rate (Verlust und Genauigkeit). Loss ist die Straffunktion, mit deren Ableitungen und der Learning-Rate, die neuen Werte der Gewichtungen berechnet werden (Gradientenabstiegsverfahren). [11] Die Abbildung 2-21 zeigt die Straffunktion und deren Ableitung.

[5] Es wurde bereits festgestellt, dass eine gleichmäßige zufällige oder normalverteilte Initialisierung der Gewichtungen, häufig zu schlechten Ergebnissen führt. [11]

Die Accuracy-Rate ist der Anteil der richtig vorhergesagten Ergebnisse zu der Gesamtzahl der eingespeisten Datenpunkte. Das bedeutet, dass die Loss-Funktion während des Trainierens minimiert wird, während die Accuracy-Rate steigt.
Das Trainieren dauert an, bis das Netz aufhört zu lernen (Globales Minimum). Ein perfektes Modell hat die Loss=0 und Accuracy-Rate =100% , was bei den meisten Modellen unmöglich ist. Im Allgemeinen müssen die Loss-Funktion kleinstmöglich und Accuracy-Rate größtmöglich sein.

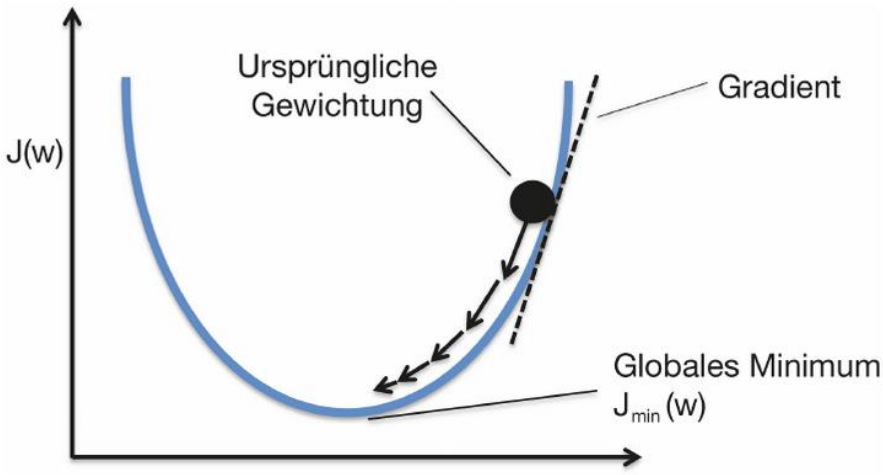


Abbildung 2-21: Die Straffunktion [11]

Die Daten werden in Stapeln oder Batches in das Modell eingespeist und das Trainieren erfolgt in Epochen (oder Iterationen). Die Größe des Batchs und die Anzahl der Epochen müssen vor dem Trainieren festgelegt werden.
Um zu beobachten, ob eine Überanpassung (Overfitting) geschieht, muss die Genauigkeit des Modells mit Daten, die das Modell während des Trainierens nicht gesehen hat, bewerten. Deshalb wird die Datensammlung in zwei Teildatensätze Validation und Training aufgeteilt. Hierbei muss beachtet werden, dass die Validationsdatenmenge klein sein muss, da diese Daten möglichst für das Training verwendet werden müssen. [11]
Es bietet sich an, das Netz mit mehreren Lernraten in mehreren Trainingsphasen zu trainieren, weil das Modell lernt schneller am Anfang des Trainings, während es mit kleinen Schritten am Ende lernt. Daher werden größere Werte für die Lernrate ausgewählt, dann wird sie alle ein Paar Epochen verkleinert. Hier muss erwähnt werden, dass Es möglich ist, das Training mit adaptiver Lernrate durchzuführen.

2.5.7 **Die Ergebnisse analysieren**

Nachdem das Training beendet wird, ist es nicht selten, dass man mit der Accuracy-Rate nicht zufrieden ist. In diesem Fall muss das Modell überarbeitet und nochmal trainiert werden oder muss eine neue Architektur gefunden werden. Außerdem ist es sehr wahrscheinlich, dass ein gutes Modell nicht auf dem Mikrocontroller funktioniert, obwohl die Accuracy-Rate groß genug ist. In diesem Fall muss ebenfalls ein neues Modell erstellt und Trainiert werden.[6]

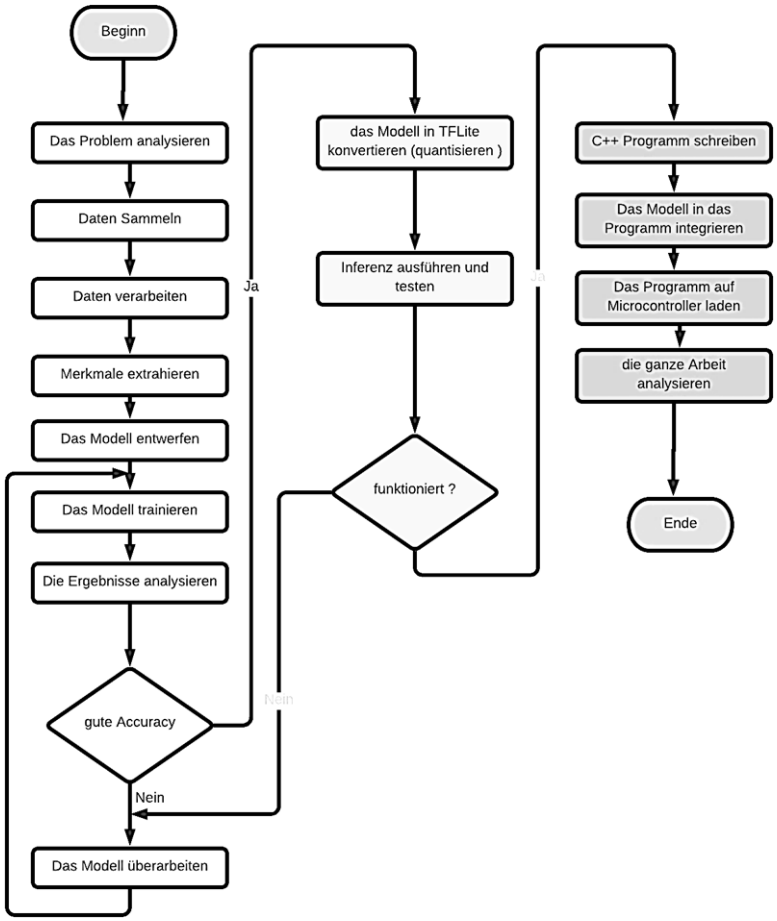


Abbildung 2-22: Der Arbeitsablauf

2.5.8 **Das Modell in Tensorflow-Lite konvertieren**

Während des Trainings werden die Gewichtungen schrittweise mit kleinen Werten angepasst. Dies bedeutet, dass die Unterschiede zwischen den Gewichtungen sehr

[6] Die Freiheitgrad bei Auswahl der Anzahl der Schichten und der Größe des Modells ist sehr klein.

klein sein können und mit großer Auflösung dargestellt werden müssen. Daher müssen diese Gewichtungen als Float32-Zahlen dargestellt werden.

Bei der Inferenz des trainierten Modells werden die Gewichtungen normalerweise ebenfalls als Float32 dargestellt, was bedeutet, dass jeder Parameter vier Bytes Speicher benötigt. Hier tritt das Speicherproblem auf, wo kein großer Speicher zur Verfügung steht und die Rechenfähigkeit sehr begrenzt ist. Der Speicherbedarf lässt sich in diesem Fall minimieren, indem diese Gewichtungen mit kleineren Datentypen wie INT8 beispielsweise dargestellt werden. Dieser Prozess wird als Quantisierung genannt und ist ein der wichtigsten Verfahren, das die Implementierung der neuronalen Netze auf Mikrocontrollern und mobilen Geräten ermöglicht hat.

Die Gewichte eines Modells unterscheiden sich leicht voneinander und sind innerhalb eines kleinen Bereichs verteilt (beispielsweise von –3 bis 6). Die Größe des Modells lässt sich in diesem Fall durch die Quantisierung verkleinern, indem die maximale und minimale Werte der Gewichtungen ermittelt und dann den Gewichtungen neue Werte mit kleinerer Auflösung (8-bit) zwischen den maximalen und minimalen Werten zugewiesen werden.

Angenommen, die Gewichtungen variieren zwischen –5.4 und 4.5, und sie müssen als Uint8-Zahlen dargestellt werden. In diesem Fall wird den Wert –5.4 durch den Wert 0 (Uint8) und den Wert 4.5 durch den Wert 256 (Uint8) ersetzt. Die Werte zwischen –5.4 und 4.5 werden zu den Werten zwischen 0 bis 225 quantisiert, was bedeutet, dass einige Informationen verloren gehen werden. Die Abbildung 2-23 veranschaulicht die Quantisierung [17].

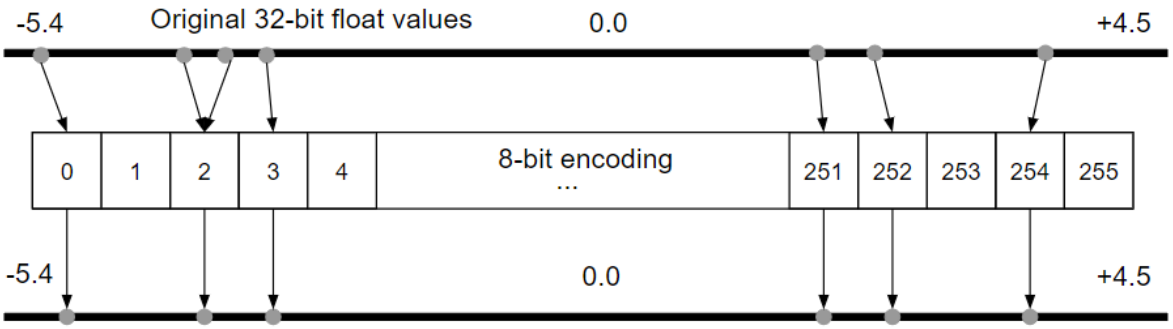


Abbildung 2-23: Quantisierung in TFlite [17]

Ein weiterer Grund für den Einsatz der Quantisierung besteht darin, den Rechenaufwand zu reduzieren, der den Prozessor während der Inferenz belastet, indem diese Berechnungen vollständig mit 8-Bit-Operationen ausgeführt werden. [18]

Das Abrufen von 8-Bit-Werten erfordert nur 25% des benötigten Speicherplatzes im Vergleich mit 32 bei float32 im normalen Fall, sodass Engpässe beim RAM-Zugriff vermieden werden, und der gesparte Speicher für den Rest des Programms verwendet werden kann.

Wenn die Berechnungen mit 8 Bit-Zahlen durchgeführt werden, werden die Modelle schneller mit geringerem Stromverbrauch ausgeführt, was bei mobilen Systemen besonders wichtig ist. Es gibt außerdem eingebettete Systeme, die nicht effizient mit Gleitkomma-Berechnungen sind. In diesem Fall können quantisierte Modelle auf solchen eingebetteten Systemen implementiert werden. [18]

Das quantisierte Modell, wie später im Laufe der Arbeit nachgewiesen wird, weist in den meisten Fällen eine geringere Accuracy-Rate verglichen mit dem Float32-Modell auf. Der Grund dafür ist, dass einige Informationen durch die Quantisierung verloren gehen. Die Abbildung 2-24 und Abbildung 2-25 verdeutlichen den Quantisierungsprozess.

Es gibt in Tensorflow die folgenden Quantisierungsmöglichkeiten:

1. Unsigned integer 8-bit (uint8)
2. Signed integer 8-bit (int8)
3. Float 16-bit (Float16)

Wenn die Quantisierung Float16 angewendet wird, ist das quantisierte Modell halb so groß wie das originale Modell.[7] Diese Quantisierungsmethode wird aber meistens zum Training verwendet werden.

Bei der Quantisierung muss der Konverter ebenfalls die maximalen und minimalen Werte wissen, die am Eingang des Netzes auftreten können. Dies ist wichtig, weil die Eingabewerte ebenfalls quantisiert werden müssen. Deshalb werden dem Konverter

[7] Float16 auf dem Mikrocontroller ist nicht unterstützt, und wird manchmal beim Training verwendet.

während der Quantisierung einige Datenpunkte übermittelt, wie später im Laufe der Arbeit zu sehen ist.

Bei der Berechnung der Ausgabe erzeugt das Modell ebenfalls Zahlen (Wahrscheinlichkeiten) mit ähnlicher Auflösung. Hier wird die Ausgabe wieder in Float32-Zahlen umgewandelt. Dies passiert durch Dequantize-Schicht, wie Abbildung 2-25 und die Abbildung 2-13 zeigen.

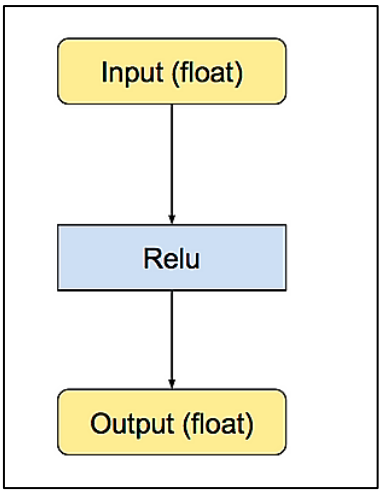


Abbildung 2-24: Das Modell ohne Quantisierung [18]

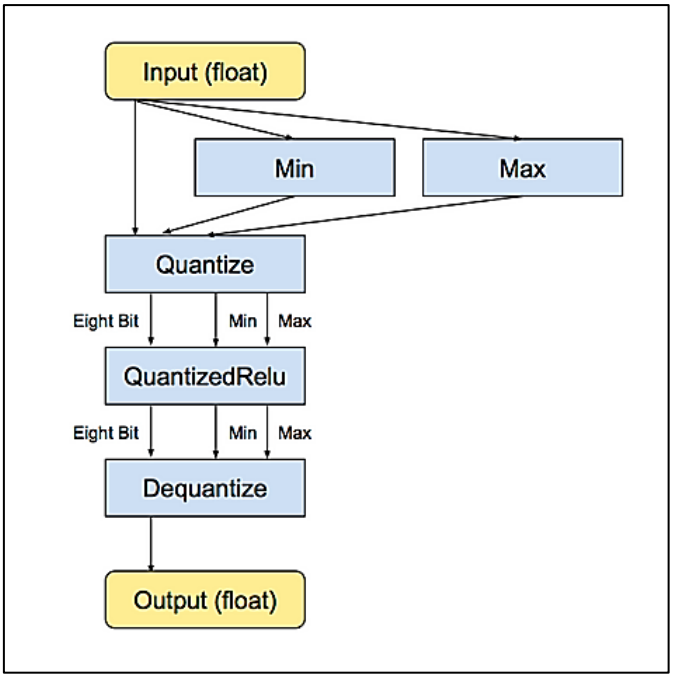


Abbildung 2-25: Die Quantisierung des Netzes [18]

Es ist hier erwähnenswert, dass es zwei Tensorflow-Konverter gibt, die zwei Quantisierungsmethoden verfolgen, wie in Abbildung 2-13 zu sehen ist. Hierbei muss

die Eingabe des Modells während der Inferenz auf dem Mikrocontroller mit unterschiedlichen Datentypen dargestellt werden. Bei Quantisierung des Spracherkennungsmodells wird später im Laufe der Arbeit der Konverter von Tensorflow 2.2 verwendet, wobei das Spektrogramm als Float32-Zahlen dargestellt werden müssen. Bei Objektserkennungsmodellen in Kapitel 4 und Kapitel 6 wird hingegen der Konverter von Tensorflow-nightly verwendet, wobei die Eingabe (die Bilder) als Int8-Zahlen dargestellt werden müssen.

2.5.9 **Ausführung des Modells**

In diesem Schritt wird das quantisierte Modell auf dem Mikrocontroller ausgeführt. Das ist wichtig, weil es keine Garantie dafür gibt, dass das Modell auf dem Mikrocontroller funktionieren wird. Daher kann man ein kleines Programm für den Mikrocontroller schreiben, in dem nur ein Interpreter[8] definiert wird, um nur zu sehen, ob das Modell richtig funktioniert und keine Fehlermeldungen auftreten werden.

Der Grund dafür, warum das Modell auf dem Mikrocontroller nicht funktionieren würde, könnte die Größe des Modells sein, oder vielleicht werden in dem Modell Operationen verwendet, die noch nicht unterstützt sind. In diesem Fall muss ein neues Modell erstellt werden.

2.5.10 **C-Programm für den Mikrocontroller schreiben**

Nachdem ein auf dem Mikrocontroller funktionsfähiges Modell erstellt, trainiert und konvertiert worden ist, muss ein Programm in C für den Mikrocontroller geschrieben werden. Dieser Schritt ist nicht immer einfach. Denn man muss überlegen, wie die Informationen mit der Kamera oder den Sensoren aufgenommen werden und wie die Merkmale aus den aufgenommenen Daten extrahiert werden sollen, um diese Merkmale in das Modell eingespeist zu werden. In dieser Arbeit wird beispielsweise sogenannte Windowing [16] der Audio-Daten angewendet, um sekundenlange Audiodaten zu bilden.

[8] Sehe Kapitel 4 für mehr Informationen über den Interpreter

2.5.11 Das Modell in C-Programm integrieren

Der Mikrocontroller hat kein Filesystem und kein Betriebsystem. Dies führt dazu, dass der Mikrocontroller keine TFLite-Modelle (Modell_name.tflite) verstehen oder lesen kann. Daher muss das Modell in lesbare Form (Modell_name.c) konvertiert werden. Dies kann in Linux durch den **xxd**-Befehl gemacht werden[9]. Dieser Befehl erzeugt einen hexadezimalen oder binären Dump einer Datei. Abbildung 2-26 zeigt ein Beispiel für ein Modell, das auf dem Mikrocontroller ausgeführt werden kann. Dieses Modell wird als eindimensionales C-Array hexadezimaler Werte in dem C-Programm definiert.

2.5.12 Das Programm auf den Mikrocontroller laden

Jetzt kann das Programm auf den Mikrocontroller übertragen und ausgeführt werden. Dafür wird eine Entwicklung-Hardware wie Arduino-IDE für Arduino Mikrocontroller und ESP-IDF für ESP-Mikrokontroller benötigt.

Es ist möglich, dass das Modell nicht richtig klassifiziert, obwohl alles richtig gemacht worden ist, weil die Daten, die während der Inferenz durch das Modell geführt werden, mit den Sensoren auf dem Mikrocontrollerboard erzeugt werden. Diese Daten können sich von den Daten, die während des Trainierens verwendet wurden, unterscheiden. Um dieses Problem zu beheben, können passende Verarbeitungsprozesse auf die Mikrocontrollerdaten angewendet werden, bevor diese Daten in das Modell eingespeist werden.

[9] In Windows gibt es ein ähnliches Programm, das Tensorflow-Modell in ein C-Array umwandeln kann. Dieses Programm heißt HxD.

```
1.     unsigned char __home_murad_Masterarbeit_Aufgaben_tflite[] = {
2.
3.    0xf4, 0x43, 0x00, 0x00, 0xd0, 0x43, 0x00, 0x00, 0xac, 0x43, 0x00, 0x00,
4.    0x98, 0x2b, 0x00, 0x00, 0x04, 0x29, 0x00, 0x00, 0xf0, 0x20, 0x00, 0x00,
5.    0xdc, 0x00, 0x00, 0x00, 0xd0, 0x00, 0x00, 0x00, 0xbc, 0x00, 0x00, 0x00,
6.    0xa8, 0x00, 0x00, 0x00, 0x94, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00,
7.    0x6c, 0x00, 0x00, 0x00, 0x58, 0x00, 0x00, 0x00, 0x44, 0x00, 0x00, 0x00,
8.    0x30, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x46, 0xba, 0xff, 0xff,
9.    0x04, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x31, 0x2e, 0x35, 0x2e,
10.   0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
11.   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
12.   0x94, 0xa6, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
13.   0x00, 0x00, 0x00, 0x00, 0xa4, 0xa6, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00,
14.   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xb4, 0xa6, 0xff, 0xff,
15.   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
16.   0xc4, 0xa6, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
17.   0x00, 0x00, 0x00, 0x00, 0xd4, 0xa6, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00,
18.   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xe4, 0xa6, 0xff, 0xff,
19.   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
20.   0xf4, 0xa6, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
21.   0x00, 0x00, 0x00, 0x00, 0x04, 0xa7, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00,
22.   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x14, 0xa7, 0xff, 0xff,
23.   0x00, 0x00, 0x00, 0x00, 0xf6, 0xba, 0xff, 0xff, 0x04, 0x00, 0x00, 0x00,
24.   0xd1, 0xfe, 0x12, 0x07, 0xfa, 0xfa, 0x10, 0x0d, 0xf9, 0xe2, 0xf2, 0x11,
25.   0xf1, 0x04, 0xc6, 0xda, 0xe6, 0xf2, 0xc5, 0xe3, 0xf2, 0x28, 0xf0, 0xd6,
26.   0xde, 0xf9, 0xc8, 0xbe, 0x25, 0xda, 0x18, 0x0c, 0x1a, 0x07, 0xf0, 0x18,
27.   0xe7, 0xe7, 0xcc, 0x14, 0xf0, 0x17, 0x28, 0x0f, 0xdb, 0x14, 0x10, 0x1d,
28.   0xfb, 0x10, 0x07, 0x0d, 0xf7, 0xe3, 0xfb, 0xf5, 0xf5, 0xfe, 0x03, 0xff,
29.   0x06, 0xc3, 0x06, 0x24, 0x15, 0x17, 0x14, 0x1c, 0x13, 0x0e, 0x00, 0x0f,
30.   0x03, 0xfa, 0x04, 0x19, 0x00, 0x12, 0xf7, 0xbe, 0x03, 0x00, 0xd3, 0x2d,
31.   0x07, 0x28, 0x19, 0x16, 0xf0, 0x07, 0xc8, 0xe3, 0x05, 0x1b, 0xe9, 0x00,
32.   0x0e, 0xde, 0xd0, 0xe8, 0xc6, 0xe5, 0x07, 0xe5, 0xe2, 0xfa, 0x0b, 0x0c,
33.   0xf3, 0xe7, 0x0b, 0x0e, 0x0a, 0x2d, 0xfa, 0xd8, 0xf0, 0xfc, 0xe7, 0xf2,
34.   0xda, 0x0c, 0xd6, 0x0c, 0x11, 0xac, 0x16, 0x18, 0x0e, 0xbe, 0xdb, 0x0e,
35.   0x1a, 0x05, 0xce, 0xf6, 0x14, 0xe3, 0xfe, 0x2e, 0x0f, 0xef, 0x0f, 0x10,
36.   0x34, 0xf6, 0x06, 0x1e, 0x22, 0x2d, 0x1b, 0x44, 0x15, 0x10, 0x19, 0x1b,
37.   0xb6, 0x37, 0xb4, 0xc5, 0xdb, 0x04, 0xc8, 0xed, 0x06, 0xef, 0x37, 0x2e,
38.   0xd7, 0xf4, 0xbc, 0xee, 0xea, 0x1f, 0x10, 0xfb, 0xe2, 0xed, 0xbb, 0xd2,
39.   0xe6, 0xc0, 0x05, 0xfd, 0xf0, 0x1e, 0xd4, 0x0e, 0x0d, 0xc8, 0xfe, 0xe8,
40.   0x11, 0x81, 0xe4, 0xf1, 0x1c, 0xf5, 0xcd, 0xdf, 0xfd, 0xda, 0x3a, 0x24,
41.   0x21, 0x09, 0xef, 0x30, 0xba, 0xc3, 0xfb, 0x0b, 0x19, 0xe7, 0x04, 0xfa,
42.   0x23, 0xe2, 0xfe, 0x09, 0xde, 0xf6, 0xee, 0x2b, 0xea, 0xcd, 0x0a, 0x1c,
43.   0x16, 0xdc, 0xf5, 0x1c, 0x12, 0xba, 0x2c, 0x01, 0xdc, 0xe3, 0xe7, 0x0c,
44.   0xe4, 0xf8, 0xd1, 0xe1, 0x1b, 0xdf, 0x02, 0xee, 0xfe, 0x02, 0xfa, 0xf3,
45.   0x0b, 0x01, 0xee, 0x04, 0xe7, 0xf0, 0xd0, 0xed, 0xe7, 0xc3, 0xe8, 0x14,
46.   0x16, 0xaa, 0x9b, 0xea, 0xd0, 0xcd, 0xec, 0xe1, 0x0f, 0xe5, 0xbb, 0xff,
47.   0x36, 0x20, 0x0b, 0xfe, 0xd0, 0x5d, 0x0c, 0xdf, 0x0a, 0x00, 0x0d, 0xbb,
48.   0x2a, 0x38, 0x00, 0x0f, 0x03, 0x64, 0x20, 0x14, 0xd4, 0xa1, 0xe8, 0x09,
49.   0x1e, 0x2a, 0x46, 0xaa, 0xeb, 0x12, 0xea, 0x1a, 0x3c, 0xb7, 0x25, 0x10,
50.   0xfe, 0x04, 0xe5, 0x12, 0x44, 0x0d, 0xe3, 0x1d, 0xdf, 0xf2, 0xf5, 0x0b,
51.   0x36, 0x19, 0xf3, 0xd9, 0xe2, 0x21, 0xf9, 0x18, 0xcf, 0x05, 0x0f, 0xf9,
52.    ..  ..      ..    ..   ..     ..   ..    ..    ..
53. unsigned int __home_murad_Masterarbeit_Aufgaben_tflite_len = 6000
54.
```

Abbildung 2-26: Ein Beispiel für Modell-Arr

Kapitel 3 - Einfache Beispielanwendungen und Überblick über die Hardware

In diesem Kapitel werden einige Beispiele aus verschiedenen Anwendungsgebieten auf den verschiedenen Mikrokontrollern implementiert, wobei einige Messungen durchgeführt werden. Es wird aber nicht auf alle Einzelheiten eingegangen. Dies muss dazu dienen, zeitliches Verständnis der ML-Modelle auf dem Mikrocontroller zu geben. Außerdem wird hier zwischen den verschiedenen Boards verglichen, um herauszufinden, welche Boards mehr oder weniger zur Implementierung von ML-Modellen passen. In Tabelle 3-1 sind die ausführbaren Anwendungen auf den verschiedenen Boards aufgelistet.

Tabelle 3-1: Anwendungsbeispiele.

	ESP-EYE	ESP-Camera	Arduino-Nano	Sparkfun-Edge
Hello World	Ja	Ja	Ja	Ja
Magic Wand	Nein (Hat keinen Sensor)	Nein (Hat keinen Sensor)	Ja	Ja
Person Detection	Ja	Ja	Nein	Ja
Micro speech	Ja	Nein (hat kein Mikrophon)	Ja	Ja

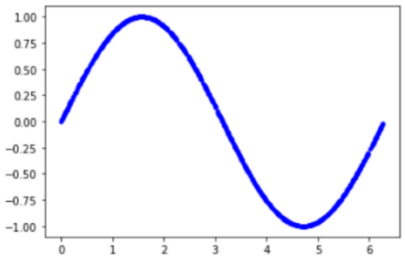
Diese Anwendungsbeispiele befinden sich in Tensorflow-Repository auf GitHub. Man Kann diese Repository klonen, die Projekte auf dem Computer generieren, auf den Mikrocontroller laden und dann schließlich ausführen. Die Terminalbefehle dazu befinden sich im Anhang A.

3.1 Hello World

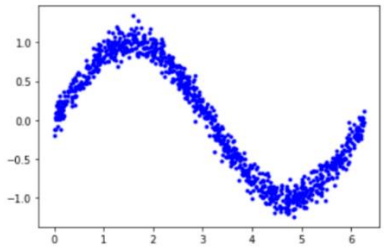
Dieses Anwendungsbeispiel zeigt auf möglichst einfache Weise, wie das ML-Modell auf dem Mikrocontroller ausgeführt wird. Es macht deutlich wie ein Modell in Tensorflow erstellt und konvertiert wird, und beleuchtet die Programmbestandteile zur Integration und Ausführung eines Modells auf dem Mikrocontroller.

Das neuronale Netz wird hierbei mit generiertem Datensatz trainiert, um ein Sinus-Signal zu erzeugen, wobei die Trainingsdaten eine ganze Periode des Sinus-Signals abdecken (Abbildung 3-1). Der erzeugte Datensatz wird dann in drei separate Unterdatensätze nach dem Zufallsprinzip aufgeteilt: trainingsdatenmenge, Testdatenmenge und Validationsdatenmenge. Die Abbildung 3-1-c verdeutlicht die Aufteilung des Datensatzes.

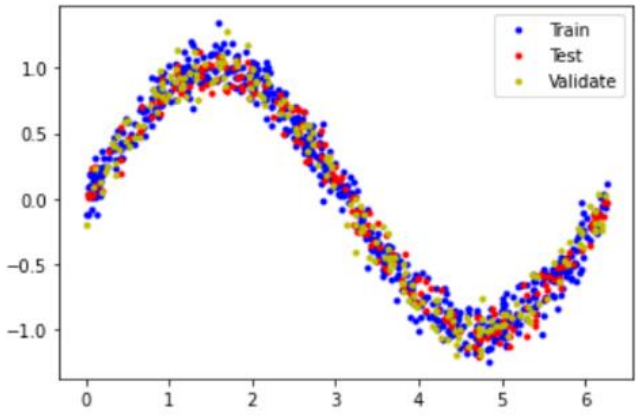
Auf dem Mikrocontroller werden Werte generiert und durch das Modell geführt, dann erzeugt das Modell anhand der erlernten Informationen die entsprechenden Amplituden (Ausgang = sin (Eingang). Der Eingang und Ausgang sind dabei ein einziger Wert (1x1). Bei größeren Modellen können aber die Dimensionen des Eingangs sowie des Ausgangs viel größer sein.



(a) Die generierten daten



(b) Daten mit zufälligem Noise



(c)

Abbildung 3-1: Aufteilung des Datensatzes

Das Modell, das mit diesen Daten trainiert wurde, besteht aus drei FC-Schichten (Fully Connected) und wurde mit 1000 Datenpunkten trainiert. Die Abbildung 3-2 zeigt die Loss-Funktion während des Trainierens.

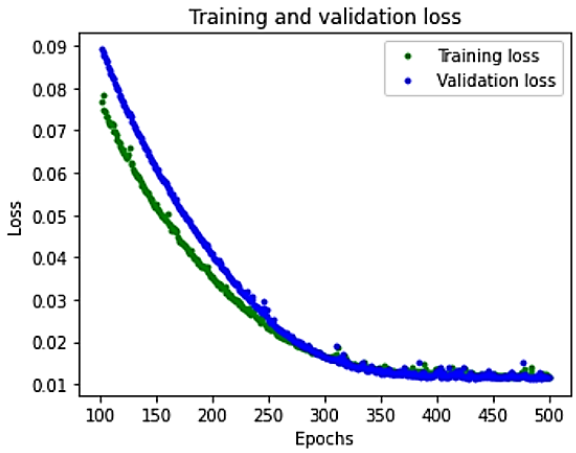


Abbildung 3-2: Loss-Funktion

Nachdem der Trainingsvorgang abgeschlossen ist, wird das Modell quantisiert. Dieses Verfahren ermöglicht, die Größe des Modells zu minimieren (Tabelle 3-2). Die Abbildung 3-3 zeigt das Modell mit und ohne Quantisierung.

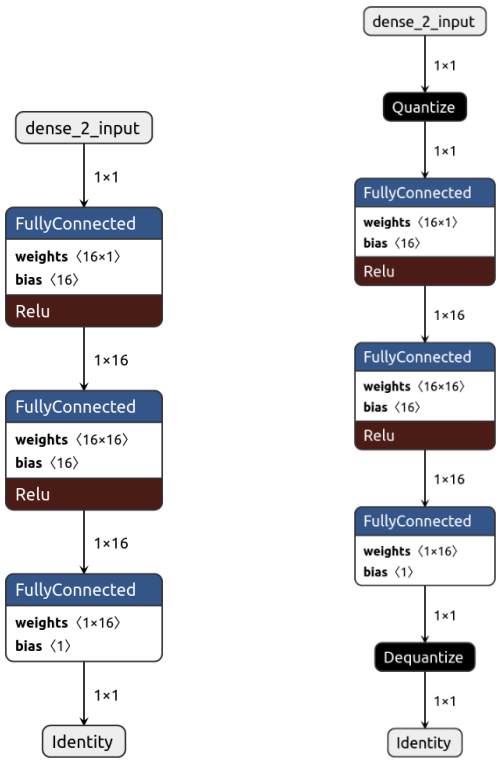


Abbildung 3-3: Das Modell mit und ohne Quantisierung

Tabelle 3-2: Die Größe des Hello-World-Modells

Größe des Modells		
Das Modell	Ohne Quantisierung	2736 Bytes
	Mit Quantisierung (INT8)	2512 Bytes

Hier muss darauf hingewiesen werden, dass der Unterschied zwischen den beiden Modellen gering ist und nur 224 Bytes beträgt. Der Grund dafür ist, dass dieses Modell sehr klein ist und der größte Anteil der Modellgröße keine quantisierbaren Parameter(Gewichtungen) sind, sondern Informationen über das Modellgraf. Bei größeren Modellen wird die Größe des Modells um 75% durch die Quantisierung gesenkt.

Das C-Programm, das auf dem Mikrocontroller ausgeführt wird, wird durch Terminalbefehle generiert. Im Anhang A befinden sich die Terminalbefehle zum Generieren des C-Codes auf den verschiedenen Mikrocontrollerboards.

3.1.1 **Hallo-World auf Arduino Nano ble sense**

Nachdem das Programm auf den Arduino-Mikrocontroller geladen wird (sehe Anhang A), wird das Modell ausgeführt. bei jeder Inferenz erzeugt der Mikrocontroller einen Wert, wobei die Werte innerhalb einer Zeitspanne zusammen ein Sinus-Signal bilden. Die Inferenzzeit des gesamten Programms beträgt 336 µs, wobei diese Zeit auch mathematische Berechnungen außerhalb des ML-Modells umfasst, die normalerweise viel kleiner als die Inferenzzeit des Modells ist. Die Abbildung 3-4 zeigt das generierte Sinus-Signal.

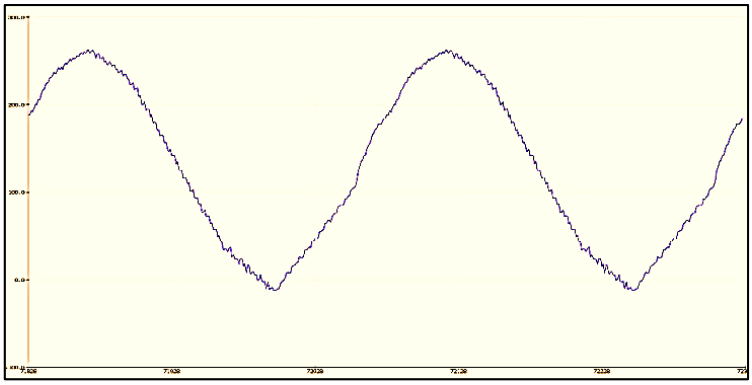


Abbildung 3-4: Das erzeugte Sinus-Signal

3.1.2 **Hello-World auf dem ESP-Board**

Der Prozessor auf diesem Board ist mächtig im Vergleich mit dem Arduino-Board, wobei die Taktfrequenz 240 MHz beträgt. Die beiden Boards ESP-EYE und ESP-Camera haben denselben Prozessor. Deshalb reicht es aus, das Programm nur auf ESP-Camera auszuführen.

Wie erwähnt, hat das ESP-CAM-Board keinen USB-Schnittstelle. Daher wird ein Adapter (ftdi232) benötigt, um das Board zu programmieren. Die Abbildung 3-5 zeigt das Board mit dem Adapter. Dabei muss beachtet werden, dass der Pin IO0 mit Ground kurzgeschlossen wird, bevor das Programm auf das Board geladen werden kann. Die Inferenzzeit des Models beträgt nur 110 µs. Der Grund dafür ist, dass das ESP-Board viel schneller als Arduino-Board ist, was bedeutet, dass mehr Werte pro Sekunde erzeugt werden.

Der Stromverbrauch bei diesem Board ist größer im Vergleich mit Arduino-Board und beträgt 80 mA beim normalen Betrieb (ohne Wi-Fi).

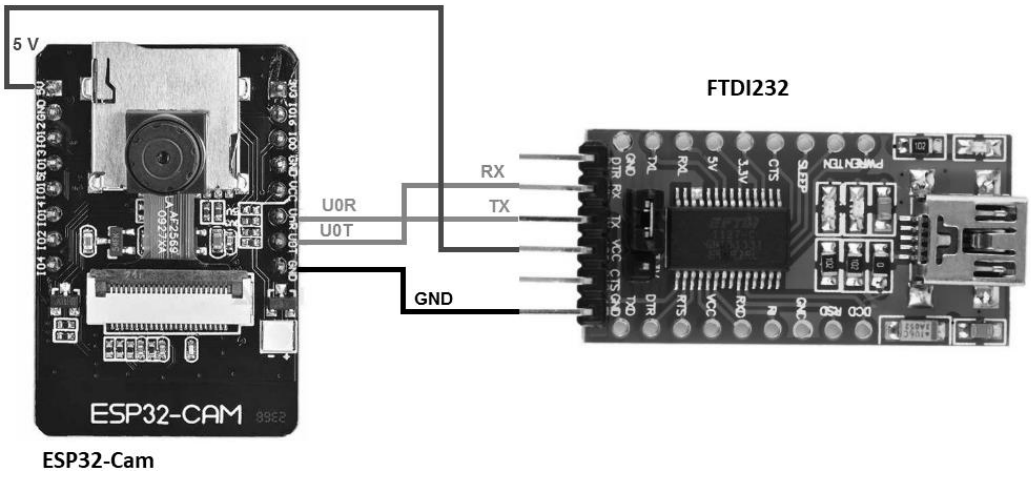


Abbildung 3-5: ESP-CAM mit ftdi232

3.1.3 **Hello-World auf Sparkfun Edge**

Bei der Implementierung des Modells auf dem Sparkfun-Edge-Board wird das sinusförmige Signal ebenfalls erzeugt (Abbildung 3-6). Die Inferenzzeit auf dem Board beträgt 720µs[10], wobei die Taktfrequenz beim normalen Betrieb 48MHz beträgt. Der größte Vorteil dieses Boards ist der sehr geringe Stromverbrauch etwa 1.6mA.

[10] Die Inferenzzeit wird beim normalen Betrieb gemessen, wobei die Taktfrequenz 48 MHz beträgt. Das Board kann aber im Burst-Mode mit 96MHz betrieben werden.

Die Tensorflow-Modelle können viel schneller auf dem Sparkfun-Board (sowie auf dem Arduino Board) mit dem optimierten Kernel ausgeführt werden, der die CMSIS-NN-Bibliothek von Arm verwendet. Im Anhang A befindet sich der Terminalbefehl zum Generieren des Hallo-Beispiels unter Verwendung von CMSIS-NN-Bibliothek. In diesem Fall wird die Inferenzzeit von 720 auf 450 reduziert. Diese große Verbesserung der Inferenzzeit auf dem Sparkfun-Board kann bei rechenintensiveren und größeren Modellen vom großen Vorteil sein.

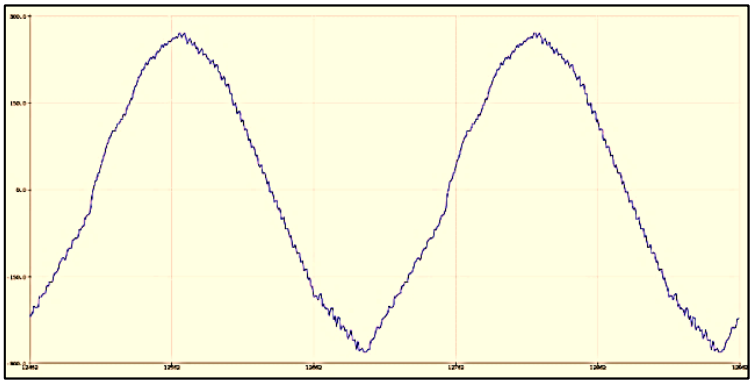


Abbildung 3-6:Das durch das Sparkfun-Board generierte Signal

Die Tabelle 3-3 enthält die zeitlichen Angaben, die bei der Implementierung auf dem Sparkfun und anderen Boards entnommen worden sind.

Tabelle 3-3:Die Inferenzzeit auf den verschieden Mikrocontrollerboards

	Zeit in µs	Taktfrequenz In MHz
Arduino Nano (CMSIS-NN)	274	64
ESP-camera	110	240
ESP-Eye	110	240
Sparkfun-Edge	720	48
Sparkfun-Edge (CMSIS-NN)	450	48

3.2 Micro Speech

In den letzten Jahren haben sich digitale Assistenten rasant verbreitet. Diese Assistenten ermöglichen einen sofortigen Zugriff auf Informationen, ohne dass ein Bildschirm oder eine Tastatur erforderlich sein muss. Sie sind überall in intelligenten

Lautsprechern, Computern und anderen Geräten integriert. Diese Assistenten wie Google kann man wecken, indem man ein Schlüsselwort sagt. Das bedeutet, dass diese Assistenten immer rund um die Uhr auf unsere Stimme hören müssen. Die Spracherkennung und die Generierung von Antworten erfolgen normalerweise auf dem Server oder Rechenzentrum, wobei ein Audio-Stream an den Server gesendet wird. Aus Sicht des Datenschutzes ist es eine Verletzung, alle Audio-Daten an den Server zu senden. Außerdem benötigt das einen enormen Datenverkehr und schnelle Internet Verbindungen zwischen Millionen von Benutzern und dem Server. Das führt zu einem enormen Stromverbrauch und zur Belastung des Funknetzes. Außerdem gibt es immer bei der Verwendung der Assistenten eine große Zeitverzögerung, bis der Benutzer eine Antwort bekommt. Aus diesen Gründen wurde in den letzten wenigen Jahren nach einem alternativen Assistenten gesucht, der keine Internetverbindung benötigt und keine Daten zu einem Rechenzentrum oder Server sendet. [16] Dieses Ziel wird in diesem Beispiel und Später in größerem Ausmaß realisiert.

Die Micro-Speech-Anwendung ist ein kleines Spracherkennungsmodell, das auf den Datensatz Speech-Command trainiert wurde und die zwei Wörter YES und NO erkennen kann. Außerdem kann das Modell die Hintergrundgeräusche und unbekannte Wörter unterscheiden. Der Trainingsdatensatz enthält eine Sekunde lange Audiodaten von 32 englischen Wörtern, die durch Hunderten von Personen mit unterschiedlichen Akzenten und Stimmen aufgenommen wurden.
In diesem Anwendungsbeispiel hört das Mikrophon die Umgebung ständig ab und liefert die aufgenommenen Daten an das Erkennungsmodell, wo erkannt wird ob das gesprochene Wort einer bekannten Klasse zuzuordnen ist. Dabei wird das Spektrogramm als Merkmal aus den Audiodaten extrahiert.
Das Modell besteht aus 4 Schichten (Abbildung 3-7). Die Erste ist eine Reshape-Schicht, die die Eingabe in neue Dimensionen umformt. Anschließend wird das Spektrogramm in eine CNN-Schicht geführt. Diese Schicht lernt während des Trainierens, wie das zweidimensionale Spektrogramm jeder Klasse aussieht (später wird auf die Einzelheiten eingegangen).

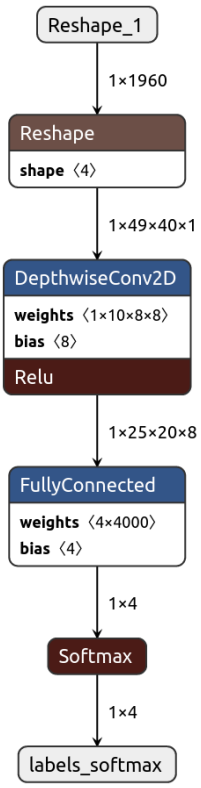


Abbildung 3-7: Das Micro-Speech-Modell

Zur Klassifizierung der Ausgabe der CNN-Schicht wird ein FC-Schicht verwendet, die so viele Neuronen enthält wie die Anzahl der Klassen (in diesem Fall vier Klassen). Das Modell ist viel größer als Hello-World-Beispiel ungefähr 64 KByte. Die Tabelle 3-4 zeigt auf, wie sich die Größe des Modells mit der Quantisierung ändert. Bei diesem Beispiel ist der Einfluss der Quantisierung auf die Größe des Modells deutlich, weil das Modell viel größer als das Hello-World-Modell ist und viel mehr guantisierbare Parameter(Gewichtungen) enthält.

Tabelle 3-4: Die Größe des Modells mit und ohne Quantisierung

	Modell Größe in Bytes
Ohne Quantisierung (Float32)	64000
Mit Quantisierung(int8)	18208

Die Quantisierung des Modells verringert die Modellgröße um fast 75%, damit das Modell auf dem Mikrocontroller ohne Speicherprobleme ausgeführt werden kann.

Wie im Hello-World-Beispiel muss das C-Programm für die verschieden Plattformen generiert werden, bevor es auf das Mikrocontrollerboard geladen werden kann. Im

Anhang A befindet sich die zur Generierung des C-Programms benötigten Terminalbefehle.

Nachdem das Programm auf den Mikrocontroller übertragen wurde, wurden die Inferenzzeiten gemessen und in die Tabelle 3-5 eingetragen. Dabei wird zwischen zwei Zeitangaben unterschieden. Die Erste stellt die zur Extraktion benötigte Zeit dar, während bei der Zweite um die für die Ausführung des Modells benötigte Zeit geht.

Tabelle 3-5: Die Zeitlichen Messungen auf den verschiedenen Plattformen

Das Board	**Das gesamte Programm**	**Spektrogramm-Extraktion**	**Das Modell**	**CPU TAKTFREQUENZ In MHz**
ESP-EYE	98000 **µs**	2000 **µs**	96000 **µs**	240
Arduino Nano ble sense ohne optimiertem Kernel	583000 **µs**	116000 **µs**	467 000 **µs**	64
Arduino Nano ble sense mit optimiertem Kernel (Cmsis-NN)	82245 **µs**	19502 **µs**	62745 **µs**	64
Sparkfun-Edge	388000 **µs**	77331 **µs**	313000 **µs**	96
Sparkfun-Edge Mit optimiertem Kernel (Cmsis-NN)	60000 **µs**	14000 **µs**	41800 **µs**	96

Für bessere Erkennung muss das Modell schnellstmöglich ausgeführt werden, damit kleinstmögliche Details zwischen den Inferenzen verlorengehen. Daher wird ausgehend von der Tabelle 3-5 erwartet, dass die Erkennung auf dem Sparkfun-Board am besten ist. Das Arduino-Board sowie das Sparkfun-Board haben im Vergleich mit ESP-Board eine geringere Taktfrequenz und die Inferenzzeit ist infolgedessen sehr groß (fast eine Sekunde) wenn die CMSIS-NN-Bibliothek nicht verwendet wird.

Glücklicherweise kann die Inferenzzeit auf diesen zwei Boards mit dem optimierten Kernel und der Cmsis-NN-Bibliothek um den Faktor 9 verringert werden, damit die Inferenzzeit auf den genannten Boards sogar geringer als die Inferenzzeit auf dem ESP-Board wird.

3.3 Person Detection

Dieses Beispiel befindet sich ebenfalls im Tensorflow-Repository und kann heruntergeladen werden. Die Befehle Zur Generierung des Beispielscodes sind ebenfalls im Anhang A zu finden. Da dieses Anwendungsbeispiel eine Kamera benötigt, wird es nur auf dem ESP-EYE-Board sowie dem Sparkfun-Board implementiert.

Dieses Modell ist ein Erkennungsmodell, das erkennen kann, ob es sich um Person bei dem aufgenommenen Bild handelt. Dieses Modell ist viel größer im Vergleich mit den anderen Beispielen und wurde mit dem Datensatz COCO Trainiert.

Dieses Modell basiert auf MobileNet-Architekturen, auf die später im Kapitel 6 ausführlich eingegangen wird. Die erste Version (v1) dieser Architekturen ist die kleinste und kann unter bestimmten Bedingungen auf dem Mikrocontroller ausgeführt werden. Die Bilder, die durch die Kamera aufgenommen werden, werden in Graustufenbilder umgewandelt, bevor sie in das Modell eingespeist werden.

Die Mobilenet-Architektur besteht aus 14 CNN-Schichten und einer FC-Schicht zur Klassifizierung. Diese Architektur ist in Tabelle 3-6 gezeigt. Hier ist zu erwarten, dass die Inferenzzeit des Modells auf dem Mikrocontroller viel größer ist. In Tabelle 3-7 sind die Inferenzzeiten auf dem ESP-EYE-Board bei unterschiedlichen Taktfrequenzen sowie auf dem Sparkfun-Board aufgelistet.

Der COCO-Datensatz (steht für Common Objects in Context), auf den das Modell trainiert wurde, ist ein vom Microsoft erstelltem Datensatz und wurde zur Segmentierung von Bildern und Erkennung von Objekten verwendet. [16]

Tabelle 3-6: MobilNet-Architektur [19]

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Die Größe des quantisierten Modells beträgt 236072 Bytes (sehr groß im Vergleich mit 18 K Byte für das Spracherkennungsmodell).

Da die Inferenzzeit auf dem Mikrocontroller sehr groß ist, wird davon ausgegangen, dass solche Objekterkennungsmodelle praktisch sind, wenn die erforderliche Anzahl der Erkennungen per Sekunde sehr gering sein soll. Für Echtzeitanwendungen wird ein Leistungsfähigerer Mikrocontroller benötigt, der das Modell mehrmals pro Sekunde ausführen kann (sehe Kapitel 8).

Tabelle 3-7: Inferenzzeit des Person-Detection-Modells

Das Board	**Zeit in Millisekunden**	**CPU-Taktfrequenz**
ESP-EYE/Cam	4718 (fast 5 Sekunden)	240 MHz
ESP-EYE/Cam	7337	160 MHz
ESP-EYE/Cam	14332	80 MHz
Sparkfun-Edge	3400 Mit optimierten Kerneln (CMSIS-NN)	96 MHz (Burst Mode)

Das Board Sparkfun-Edge kann mit größerer Taktfrequenz (96 MHz) betrieben werden, Indem man die Burst Mode aktiviert, während das ESP-Board mit beliebigen Taktfrequenzen zwischen 10 und 240 MHz betrieben werden kann.

Es ist hier zu beachten, dass der Energieverbrauch des Boards mit der Taktfrequenz abhängt. Daher empfehlt es sich die Taktfrequenz klein halten, wenn das Modell sehr klein ist oder wenn die Inferenzzeit groß sein darf.
Wie bei den bisherigen Beispielen befindet sich die Terminalbefehle zur Generierung vom C-Programm auf den verschiedenen Plattformen im Anhang A.
Schließlich ist es zu erwähnen, dass das Sparkfun-Board die beste Leistung aufgewiesen hat, obwohl die Taktfrequenz des ESP-Boards viel größer ist. Dies ist der Fall, wenn die CMSIS-NN-Bibliothek verwendet wird.

3.4 Magic Wand

Dieses Beispiel befindet sich im Tensorflow-Repository auf GitHub und kann geklont werden. Im Anhang A sind der Terminalbefehle zur Generierung des C-Programms zu finden.
Für dieses Beispiel wird eine Beschleunigungssensor benötigt. Daher kann die Anwendung nur auf Arduino- und Sparkfun-Board, die über solchen Sensor verfügen, ausgeführt werden.
Die mit Beschleunigungssensor aufgenommenen Messdaten auf drei Achsen X,Y, und Z, tragen Details über die Bewegungen des Sensors, wobei jede Bewegung ein eigenes Signalmuster generiert, worauf ein neuronales Netz trainiert werden kann, um die Bewegungen zu klassifizieren. Beispielsweise beschreibt das Sensorsignal, das in Abbildung 3-8 zu sehen ist, eine gehende Person. Ferner haben die Wissenschaftler solche Messdaten dazu verwendet, die Personen anhand von diesen Daten zu identifizieren. Hierfür wird die Tatsache zunutze gemacht, dass sich jeder Mensch in eigener Art und Weise bewegt. Es wird sogar versucht, das Alter, das Geschlecht und weitere Daten vorherzusagen, wobei diese Parameter die Bewegung des Menschen stark beeinflussen.
In diesem Anwendungsbeispiel wird das Netz auf drei unterschiedliche Gesten trainiert, die in Abbildung 3-9 zu sehen sind, wobei die Trainingsdaten durch ein Sparkfun-Board aufgenommen worden sind. Jeder Trainingsdatenpunkt enthält dreiachsige Beschleunigungsdaten von mehr als fünf Sekunden Sensordaten, wobei das Sparkfun-Board ungefähr 25 Messungen pro Sekunde liefert. Das bedeutet, dass Jeder Trainingsdatenpunkt die Größe 128x3 hat. [16]

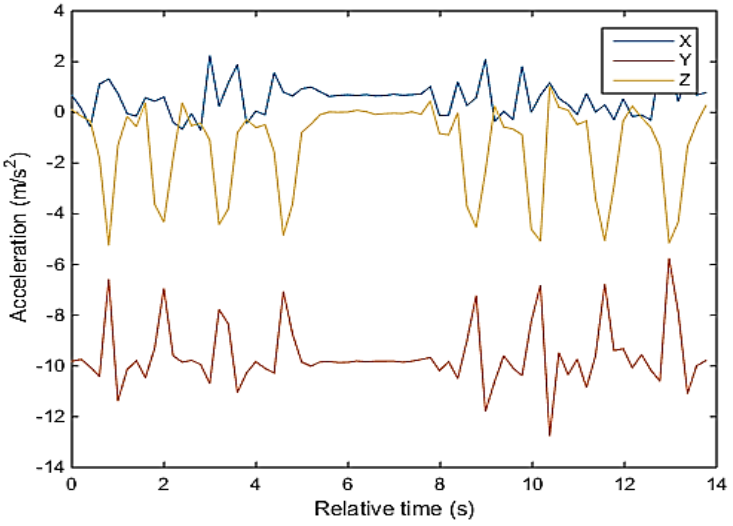


Abbildung 3-8: Messdaten eines Beschleunigungssensors [20]

Abbildung 3-9: Die Trainingsgesten

Das trainierte Netz, dessen Struktur in Abbildung 3-10 dargestellt wird, besteht hauptsächlich aus CNN-Schichten zur Extraktion von Merkmalen und zwei FC-Schichten zur Klassifizierung der Gesten. Die letzte FC-Schicht ist von einer Softmax-Aktivierungsfunktion zur Erzeugung von Wahrscheinlichkeiten gefolgt. Die Ausgabe dieser Funktion enthält 4 Klassen, wobei 3 davon die drei Gesten darstellen, während die vierte Klasse eine unbekannte Bewegung oder Geste darstellt.

Bei der Inferenz auf dem Mikrocontroller empfängt die Eingabeschicht des Netzes vierdimensionale Eingabe(hat vier Dimensionen) mit der Größe (1,128 ,3,1) und liefert für jede Eingabe eine Vorhersage. Daher ist ein Buffer notwendig, der die Messdaten von ungefähr 5 Sekunden vorläufig speichert. Die Inferenzzeit bestimmt außerdem, wie schnell die Bewegung sein muss, um erkannt zu werden. Die Anzahl der pro Sekunde gelieferten Messungen unterscheidet sich dabei, je nachdem auf welchem Board das Modell ausgeführt wird. [16]

Hier wird auch die Quantisierung des Netzes eingesetzt, wodurch der Rechenaufwand und der Speicherbedarf stark minimiert wird. Der Tabelle 3-8 zeigt die Größe des Modells mit und ohne Quantisierung.

Tabelle 3-8: Die Größe des Magic-Wand-Modells

	Größe des Modells in Bytes
Mit Quantisierung (Uint8)	9700
Ohne Quantisierung (Float32)	19616

In diesem Beispiel wird die Windowing-Technik angewendet, wobei das Zeitfenster 5 Sekunden von Beschleunigungsdaten umfasst. Die Tabelle 3-9 zeigt, wie schnell das Modell auf dem jeweiligen Board ausgeführt wird.

Tabelle 3-9: Inferenzzeit des Magic-Wand-Modells

Das Board	**Inferenzzeit des Modells**	**Ausführungszeit des gesamten Programm**	**Taktfrequenz in MHz**
Arduino Nano ble sense	144836 **µs**	176544 **µs**	64
Sparkfun Edge	35000 **µs**	40000 **µs**	96 (Burst Mode)

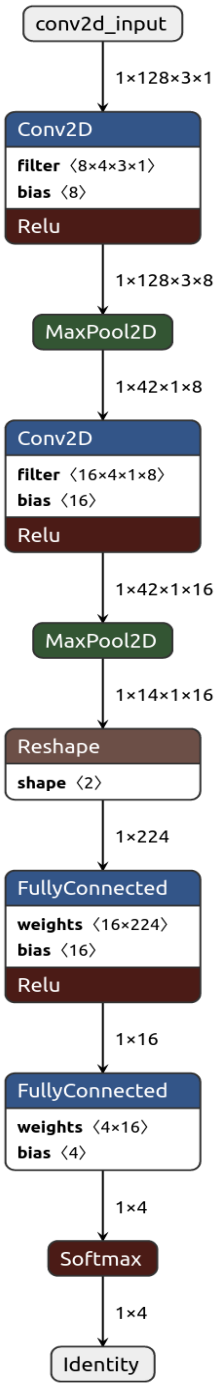


Abbildung 3-10: Die Netzarchitektur des Magic-Wand-Modells

3.5 Zusammenfassung

Das ESP-Board hat den schnellsten CPU mit maximaler Taktfrequenz von 240 MHz, und somit ist es ein interessantes Board für ML-Anwendungen. Außerdem bietet dieses Board die Möglichkeit, die Taktfrequenz flexibel zwischen 10 und 240 MHz einzustellen, je nachdem, welche Anwendung man ausführen möchte. Bei Verwendung von maximaler Frequenz (240 MHz) ist der Energieverbrach maximal und

das Board wird in diesem Fall stark erhitzt. Ein Vorteil von diesem Board ist die vielen Kommunikationsmöglichkeiten wie Bluetooth und Wi-Fi, die später im Kapitel 7 verwendet werden, um verschiedene Boards miteinander zu vernetzen. Der Preis dieses Boards unterscheidet sich zwischen wenigen Euros bis zu mehr als 40 Euro Je nachdem, welches Board mit welchen Peripherien und Sensoren benötigt wird.

Das Board Arduino nano ble sense verfügt über Vielzahl an Sensoren. Daher kann man dieses Board in vielen ML-Anwendungen einsetzen. Der ARM-CPU auf dem Board hat maximale Taktfrequenz von 64 MHz und bietet die Möglichkeit, ML-Modelle unter Verwendung von CMSIS-NN Bibliotheken auszuführen. Diese macht die Inferenz auf dem Board viel schneller und effizienter. Außerdem kommt die Möglichkeit, das Board mit der Umgebung über Bluetooth-Verbindung zu vernetzen. Der Stromverbrauch ist bei diesem Board gering und ist somit geeignet für mobile Anwendungen, wo der Stromverbrauch von entscheidender Bedeutung ist. Außerdem stehen GPIOs zur Verfügung, um das Board mit anderen Modulen zu verbinden.

Das Board Sparkfun-Edge ist ein sehr interessantes Board und wurde von Tensorflow für ML-Anwendungen entwickelt. Dieses Board verfügt über einen ARM-Prozessor M4, der mit zwei Taktfrequenzen 48 MHz und 96MHz (Burst Mode) betrieben werden kann. Dieses Board hat eine sehr gute Leistung aufgewiesen. Besonders interessant ist, wie beim Arduino-Board, die Möglichkeit, die Modelle mit optimiertem Kernel unter Verwendung von CMSIS-NN Bibliotheken auszuführen. Es hat zudem eine Vielzahl an Sensoren und kann somit für viele ML-Anwendungen eingesetzt werden. Dieses Board hat einen sehr geringen Stromverbrauch und passt somit am besten für mobile Anwendungen, wobei eine Batteriezelle das Board für bis zu 10 Tage betreiben kann. Es ist hier zu beachten, dass die Größe des TFlite-Modells (in Bytes) keine klaren Informationen über das benötigte RAM auf dem Mikrocontroller liefert , was bedeutet, dass das für das Modell benötigte Ram durch Experimentieren gefunden werden kann.

Kapitel 4 **- Erstellung von Erkennungsmodellen**

In den letzten Kapiteln wurde untersucht, welche Probleme auftauchen könnten, wenn das Modell auf den Mikrocontroller übertragen werden muss. In diesem Spezialfall wird kein Netz auf dem Mikrocontroller trainiert sondern auf einem externen Rechner, was Schwierigkeiten und Herausforderungen bereiten könnte. Außerdem kommen die Probleme hinzu, die man im allgemeinen Fall bei Erstellung eines ML-Modells haben könnte, besonders wenn keine Ressourcen zur Verfügung stehen, die beim Training von ML-Systemen benötigt werden, wie zum Beispiel die Datensätze. In diesem Fall muss man selbst darum kümmern, Trainingsdaten zu sammeln. Dies ist aber nicht ganz einfach, und zwar aus folgenden Gründen:

1. Eine große Zahl an Datenproben benötigt das Trainieren eines Netzes. Die Größe dieser Datenmenge hängt aber von der Komplexität der Daten und den Ähnlichkeiten zwischen den Klassen ab.
2. Bei Daten wie Audio, Fotos oder Video müssen die Trainingsdatenpunkten von verschiedenen Quellen und Personen stammen (nicht selten von tausenden Quellen). Dies ist Sehr wichtig für die Verallgemeinerung des Netzes. [11]
3. Man muss die bestmögliche Netzarchitektur durch Versuchen und Erfahrungen finden. Gleichzeitig beeinflusst viele Faktoren und einstellbare Parameter die Arccuracy-Rate des Netzes.

Trotz dieser Schwierigkeiten wird in diesem Kapitel, ein ganzes Spracherkennungs-modell entworfen und auf erstellten Datensatz trainiert. Dieses Modell soll eine kleine Gruppe von deutschen Wörtern erkennen kann und wird anschließend auf den Mikrocontroller übertragen. Dies bedeutet, dass auch ein C-Programm für den Mikrocontroller geschrieben werden muss. Der in Abbildung 2-22 dargestellte Arbeitsablauf verdeutlicht die Arbeitsschritte und kann dabei helfen, ein funktionsfähiges Modell zu erstellen. Anschließend wird ein Objekterkennungsmodell erstellt, das handgeschriebene Zahlen erkennen kann und sich auf den Mikrocontroller übertragen lässt.

[11] Netzverallgemeinerung bedeutet, dass das Netz die Daten von allen Quellen richtig klassifiziert, obwohl es diese Daten während des Trainierens nicht gesehen hat.

4.1 Spracherkennung deutscher Wörter

4.1.1 Anwendungsarchitektur auf dem Mikrocontroller

Das Mikrophon auf dem Mikrocontroller nimmt die Audiodaten auf und generiert einen kontinuierlichen Audiostream, der unter anderem die Informationen enthält, die erkannt werden müssen. Dieser unendliche Fluss von Audiodaten kann eigentlich ohne Vorverarbeitungsprozesse nicht weiterverarbeitet werden. Daher wird eine Technik benötigt, die diesen Datenfluss in kleinere Audiobruchteile zerlegt, damit eine begrenzte Menge von Audiodaten separat verarbeitet und in das Modell eingespeist werden können. Diese Technik ist als Windowing der Audiodaten bekannt. [16]

Das Fenster (Window) hat eine bestimmte Breite oder in anderen Worten umfasst eine bestimmte Menge von Audiodaten, die in dem Zeitfenster auftreten. Je nachdem, welche Menge von Informationen in das Modell eingespeist werden müssen, wird die Breite des Fensters bestimmt. Aus diesen Audiodaten wird ein Merkmal extrahiert, das die Sprachinformationen gut ausdrücken kann. Dieses Merkmal heißt Spektrogramm, das durch das Modell geführt wird, um ein Muster zu erkennen, das eine Bestimmte Klasse darstellt.

Das Modell erzeugt bei der Ausführung Wahrscheinlichkeiten, wie jede Klasse im Spektrogramm dargestellt ist. Anhand dieser Wahrscheinlichkeiten wird eine Entscheidung getroffen und ein Code ausgeführt (zum Beispiel das Anzeigen von Informationen auf dem Bildschirm). Dieser Prozess wird mehrere Male pro Sekunde wiederholt. Abbildung 4-1 zeigt eine Vorstellung darüber, wie das Programm auf dem Mikrocontroller aussehen könnte.

Die Ausführung des Modells sowie die Extraktion des Spektrogramms befinden sich in einer unendlichen Schleife. Währenddessen werden die Audiodaten andauernd mit dem Mikrophon aufgenommen und daraus Spektrogramme extrahiert. Wenn ein bekanntes Wort in das Fenster auftritt, wird das Modell entsprechend eine Vorhersage treffen.[12]

[12] Auf mehr Einzelheiten wird später in diesem Kapitel eingegangen.

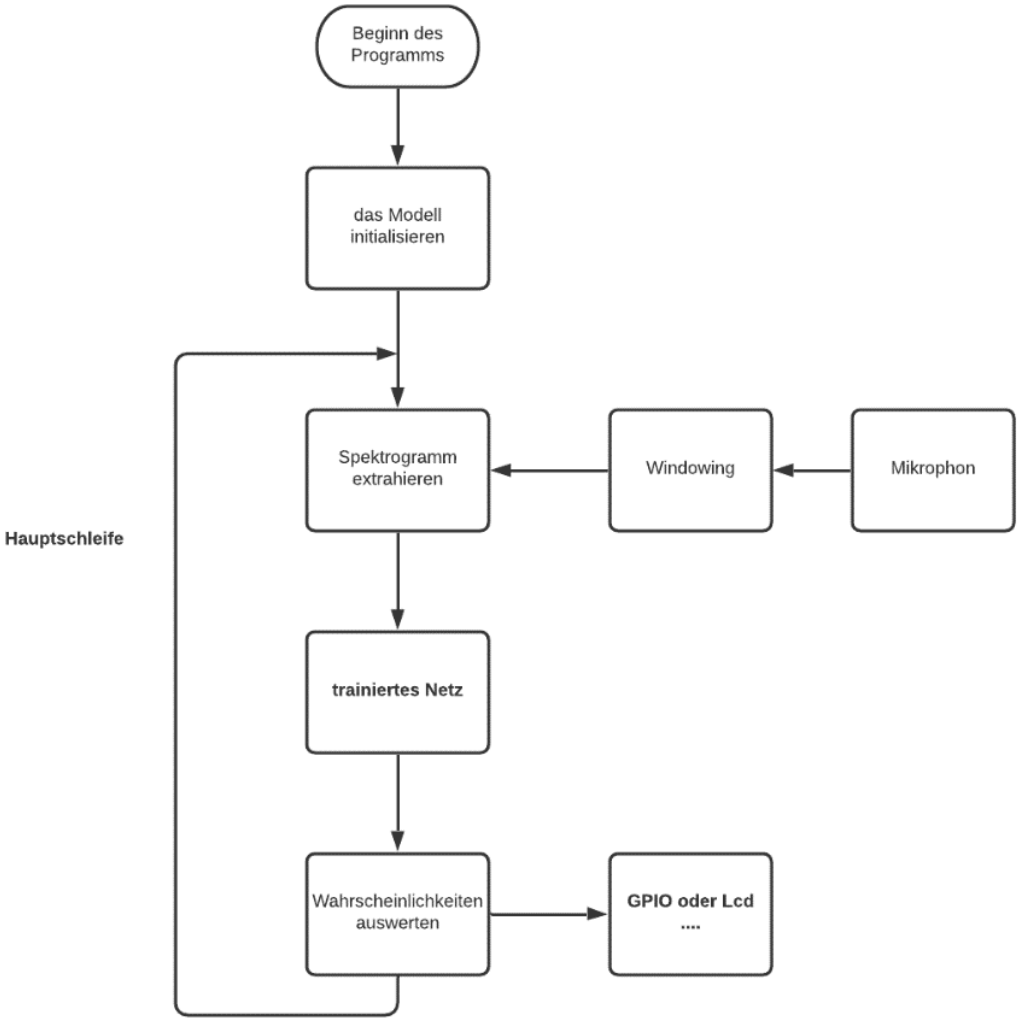


Abbildung 4-1: Flussdiagramm des Programms auf dem Mikrocontroller

4.1.2 **Spektrogramm Extraktion und Windowing von Audiodaten**

Das Mikrophon nimmt die analogen Audiosignale auf, die sich in der Luft verbreiten und wandelt sie in elektrisches Signal um. Diese elektrischen Signale sind ebenso analog. Da die Informationen auf dem Mikrocontroller nur digital dargestellt werden, werden diese Signale abgetastet, quantisiert und in digitale Form durch ADC umgewandelt. Die Abtastrate des Signals ist die Anzahl der abgetasteten Werte in einer Sekunde, während die Anzahl des Quantisierungsebenen die Auflösung darstellt. Größere Anzahl der Quantisierungsebenen oder größere Abtastrate bedeutet größere Qualität des Audiosignals, aber auch größere Datenmenge. In dieser Anwendung wird die Abtastrate 16000 ausgewählt. Die Abtastungen werden gleichzeitig mit 16 Bits-Zahlen repräsentiert. Das bedeutet 2^{16} Quantisierungsebene (oder in anderen Worten

die Auflösung beträgt 16 Bits). Die Abbildung 4-2 verdeutlicht die Abtastung des Signals.

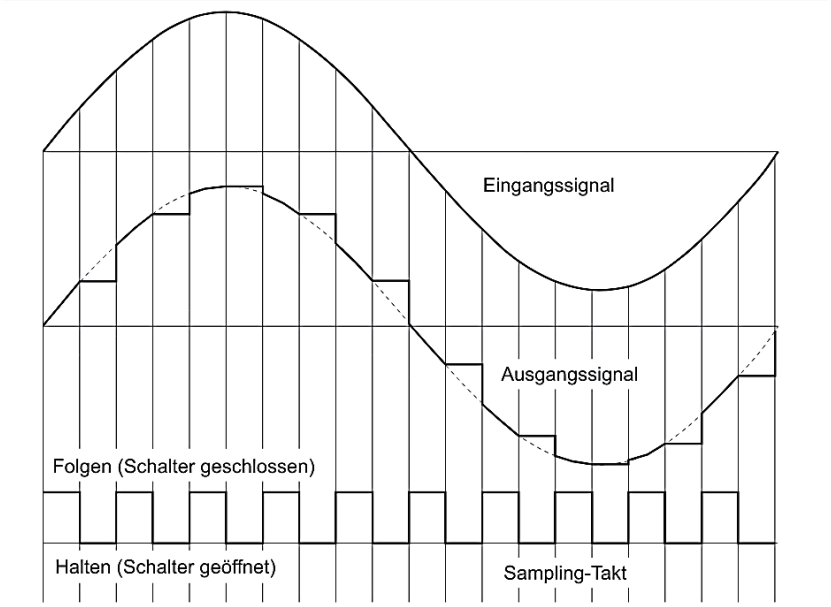


Abbildung 4-2: Signalabtastung [21]

Die Abtastrate muss double so groß wie die Signalfrequenz. Daher beträgt die größte Frequenz des Signals, die berücksichtigt wird, 8 KHz. Alle größeren Frequenzen werden hingegen vernachlässigt.

In jeder Inferenz werden Abtastungen genommen, die in das Zeitfenster hineinpassen. Die Breite dieses Fensters (Window) hängt von der Länge des Audioausschnitts ab, der erkannt werden muss. In diesem Fall wird angenommen, dass die zu erkennenden Wörter maximal eine Sekunde lang sind, und somit muss die Breite des Fensters ebenso eine Sekunde lang sein. Die Abbildung 4-3 zeigt, wie das Spektrogramm berechnet wird.

Für die Erzeugung der Spektrogramme, wird MFCC verwendet. [16] Dieser ist ein Ansatz zur Erzeugung von Spektrogrammen und wird von Google für Spracherkennungsanwendungen eingesetzt. Hierbei ist das Spektrogramm von einer Sekunde Audio ein zweidimensionales Array mit 49 Zeilen und 40 Spalten, wobei jede Zeile die Fast-Fourier-Transformation (FFT) von 30 Millisekunden Audiodaten darstellt.

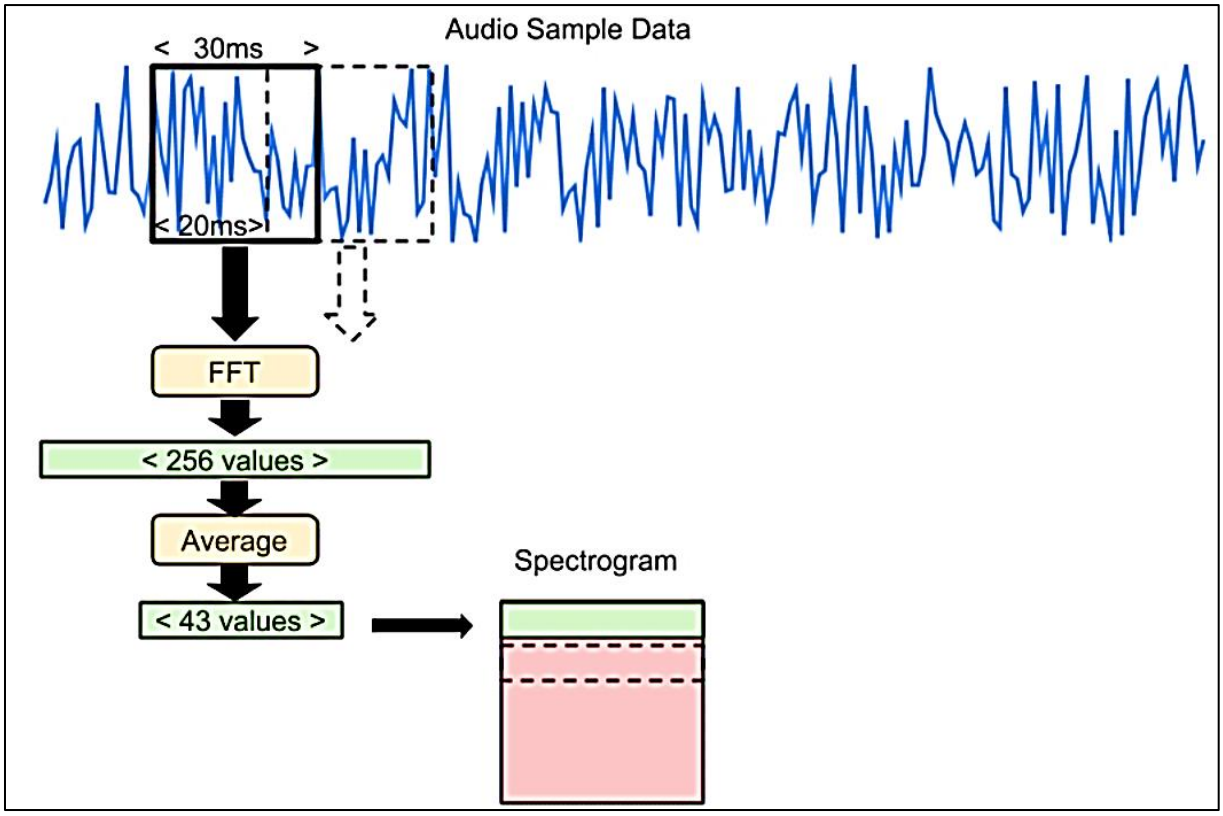


Abbildung 4-3: Spektrogramm-Extraktion [22]

Die Anwendung der FFT-Transformation auf 30 Millisekunden Audiodaten ergibt N/2 Werte, die der Energieverteilung auf N/2 Frequenzbereichen entspricht, wobei N die Anzahl der Abtastungen ist. [16] In unserem Fall enthält die 30 Millisekunde 480 Abtastung.

$$N = (30/1000) * 16000 = 480 \text{ Samples}$$

Daher erzeugt die Anwendung von FFT auf diese N-Abtastungen 240 komplexe Werte, die die Energieverteilung auf 240 Frequenzen darstellen. Diese Komplexen Werte setzen sich aus zwei Teilen zusammen, realem Teil und imaginärem Teil. Was hier bei der Berechnung der FFT von Interesse ist, sind die Amplituden der Frequenzen. Daher wird der absolute Wert von diesen Komplexen Zahlen berechnet (Amplitude = $\sqrt{R^2 + I^2}$).

Um diese 240 Werte durch Zahlen zu repräsentieren, werden 8 Bits benötigt, mit denen 2^8 =256 Werte dargestellt werden können, wobei die Werte zwischen 240 und 256 (16 Werte) auf null gesetzt werden. Diese 256 Werte für jede 30 Millisekunde der

Audiodaten sind zu viel, deswegen muss die Anzahl der Werte verkleinert werden[13], indem die 256 Werte in Sechsergruppen unterteilt werden. Anschließend wird der Durchschnitt von den Sechs Werten berechnet. Dieses Verfahren reduziert die Anzahl der Werte von 256 auf 43. [16]

Der Rechenaufwand (in Operationen) der FFT-Transformation kann berechnet werden, und wird wie folgt angegeben:

1. $\frac{N}{2} \cdot log_2(N)$ Multiplikationsoperationen
2. $N \,.\, log_2(N)$ Additionen

und insgesamt $3 \cdot \frac{N}{2} \cdot log_2(N)$ (4-1)

Wobei N die Anzahl der Audioabtastungen ist.

Beispielsweise beträgt der Rechenaufwand der 30 Millisekunden Audiodaten 6412 Operationen, wobei die Berechnung des Rechenaufwands in diesem von großer Bedeutung ist, da die Rechenfähigkeit des Mikrocontrollers begrenzt ist.

In Abbildung 4-3 ist zu sehen, dass ein 30ms langes Fenster jedes Mal 20 Millisekunden nach vorne verschoben wird, sodass in einer Sekunde 49 Slices entstehen, die die gesamte Sekunde abdecken. Hier ist zu merken, dass sich die 30ms langen Fenster überlappen und die Überlappung 10ms beträgt. Dies erzeugt ein zweidimensionales Array mit 49 Zeilen und 40 Spalten und insgesamt 40*49 =1960 Spektrogramm-Element. Die Abbildung 4-4 zeigt das Spektrogramm von vier unterschiedlichen Klassen. Es ist zu sehen, dass das Spektrogramm einer Klasse sehr ähnlich ist. Hier wird auf dieser Tatsache aufgebaut, um ein Spracherkennungsmodell zu erstellen. [22]

Für das Trainieren des Netzes wird ein ähnliches Spektrogramm verwendet ,das aus einem Sekunde langen Audio extrahiert worden ist, wobei der verwendete Datensatz Tausenden von Audioproben enthält.

[13] Die Größe des Spektrogramms hat einen großen Einfluss auf die Größe des Modells.

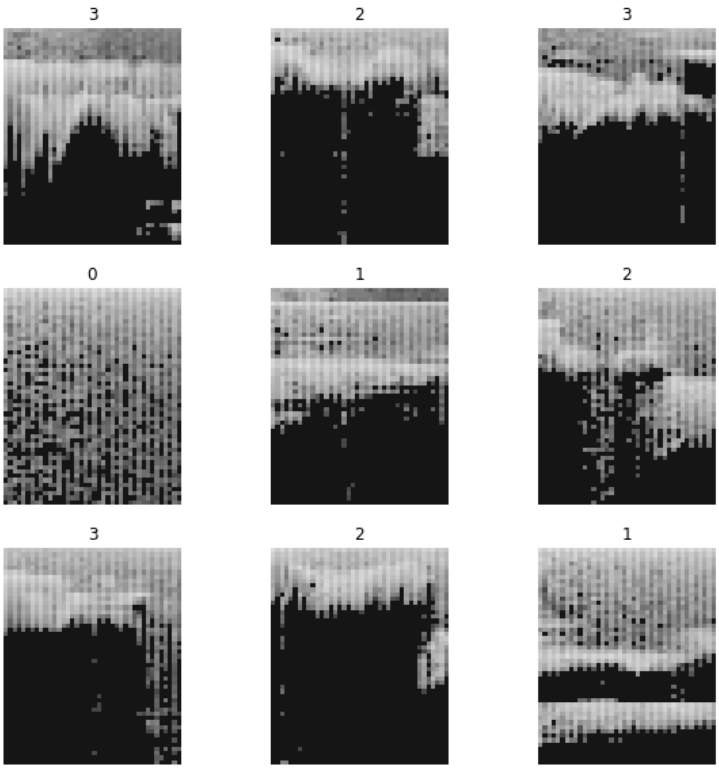


Abbildung 4-4: Das Spektrogramm unterschiedlicher Klassen

Es gibt hierfür zwei Möglichkeiten, entweder wird das Spektrogramm während des Trainierens extrahiert, oder wird es vorerst berechnet und als Trainingsdatensatz gespeichert.

In solchen Projekten wird erwartet, dass viele Experimente durchgeführt werden müssen, bis ein Modell gefunden wird, das alle Anforderungen erfüllt und auf dem Mikrocontroller funktioniert. Da die Extraktion des Spektrogramms den Trainingsvorgang viel verlangsamt, bietet es sich an, Datensatz aus Spektrogrammen zu erstellen und auf der Festplatte zu speichern. Wenn ein Netz trainiert werden muss, wird einfach dieser Datensatz aus der Festplatte geladen.

Zu diesem Zweck wurden zwei Funktionen in Python geschrieben, die erste für das Extrahieren und Speichern des Spektrogramms, während die zweite für das Landen des Spektrogramms aus der Festplatte ist. Die Funktion spektrogramm_save extrahiert hierbei das Spektrogramm und speichert es in 3 JSON-Dateien, Trainingsdatenmenge, Validationsdatenmenge und Testdatenmenge. Der folgende Listing 4-1 zeigt die Parameterliste dieser Funktion.

Listing 4-1: Die Parameterliste der spektrogarmm_save Funktion

```
1. spektrogram_save ( data_dir , # Pfad zur Audiodaten
2.     save_d ,
3.     # Der Pfad zum Speichungsort auf der Festplatte
4.     wortliste='yes,no',
5.     # Die wörter, von denen das Spektrogramm extrahiert wird
6.     kommentar ='spektrogramm' ,
7.     # Der Name des Folders, wo wir das Spektrogramm  speichern
8.     abtastrate = 16000 ,
9.     #Abtastrate des Audiosignals
10.       zeit_fenster= 1000 ,
11.       #Die Lände des Audiodateien
12.       windowing_fenster= 30.0 ,
13.       # die Breite des Fensters
14.       zeit_verschiebung= 20 ,
15.       # die Größe des Verschiebungschritt
16.       spektrogramm_breite= 40 ,
17.       #die Breite des Spektrogramms
18.       validierung_anteil= 5 ,
19.       #Die prozentige Anteil des Validierungsdatensatz
20.       testing_anteil= 5,
21.       #die Prozentige Anzeil des Testingsdatensatz
22.       Ruhe_anteil= 25 ,
23.       #Anzahl der Silence Klasse
24.       unbekannt_anteil = 25 ,
25.       #Anzahl der Unknown Klasse
26.       time_shift_ms=100,
27.       hintegrund_frequenz = 0.8 ,
28.       #die Frequenz des Noise-Signals
29.       noise_amplitude = 0.1 ,
30.       #die Amplitude des noise-Signals
31.       speech_commands_url=
32.
   'https://storage.googleapis.com/download.tensorflow.org/data/speech_commands_v0.02.tar.gz'
33.       )
34.
```

Die Funktion verwendet mehrere Bibliotheken und Klassen. Die Wichtigsten sind:

- Json (JavaScript Object Notation). Diese Bibliothek ermöglicht, die strukturierten Daten sehr kompakt zu speichern und kann in Python problemlos ohne zusätzliche Installationen importiert werden.
- Die Klassen models und input_data aus Tensorflow-Repository auf GitHub. Daher muss Tensorflow geklont werden und dann den Pfad zu diesen Klassen auf der Festplatte erweitert werden, bevor die Funktion Spektrogram_save verwendet werden kann. Außerdem ist es zu beachten, dass diese Klassen nur mit Tensorflow < = 2.2 funktionieren.

Wenn das Modell trainiert werden muss, müssen die Json-Dateien aus der Festplatte importiert und entschlüsselt werden, damit die Spektrogramme wieder in ein Numpy-Array umgewandelt werden können. Diese Arrays enthalten dann die Spektrogramme aller Trainingsdaten. Zu diesem Zweck wird die Funktion spektrogram_load verwendet. Der folgende Programmausschnitt in Listing 4-2 zeigt die Parameterliste der Funktion.

Listing 4-2:Die Parameterliste der Spektrogramm_load Funktion

```
1. def spektrogram_load ( save_dir=
2.                       '/home/murad/spektrogram_dataset' ,
3.                        dims='2d' ,
4.                        one_hot =True,
5.                        classes_num = 4    ):
```

Um das Spektrogramm zu importieren, muss der Funktion der Pfad zum gespeicherten Spektrogramm auf der Festplatte übermittelt werden. Außerdem kann die Form des Spektrogramms entweder als 2d oder als 4d ausgewählt werden, wobei 2d bedeutet, dass das Spektrogramm die Form (1960,1) haben wird, während 4d bedeutet, dass das Spektrogramm die Form (1,49,40,1) hat. In dieser Arbeit wird die Form (1960 ,1) verwendet und durch einen Reshape-Schicht innerhalb des Modells umgeformt werden. Zudem muss man der Funktion die Klassenanzahl übergeben.

Die Verwendung dieser Zwei Funktionen beschleunigt den Trainingsprozess und fasst viele Schritte zusammen. Allerdings muss man darauf achten, dass einige Packages und Bibliotheken installiert werden müssen.

4.1.3 **Die Erstellung des Datensatzes**

Ein großes Problem besteht darin, dass kein deutschsprachiger Datensatz zur Verfügung steht. Dies bedeutet, dass ein Datensatz erstellt werden muss. Dieser Schritt ist ein der schwierigsten Schritten auf dem Weg zu einem trainierten Netz. Denn man muss Hunderten oder Manchmal Tausenden von Datenproben sammeln. Dieser Schritt ist in Großunternehmen wie Google oder Amazon eine einfache Aufgabe, weil sie über die Big-Data, die durch Milliarden von Benutzern erzeugt wird, verfügen. In diesem Projekt müssen mindestens ein Tausend Audio-Probe pro Klasse verfügbar sein, sodass eine gute Accuracy-Rate beim Trainieren erzielt werden kann. Zudem müssen die Audioproben idealerweise von unterschiedlichen Personen stammen, damit das Modell die verschiedenen Aktzenten und Tonen lernen kann. Diese Probleme werden in dieser Arbeit überwältigt, Indem die folgenden Techniken verwendet werden:

1. Audio aufnehmen. Zu diesem Zweck wurde ein Programm in Python geschrieben, das ermöglicht, Audiodaten mit beliebiger Länge und Auflösung aufzunehmen.
2. Augmentation auf die Datenproben anwenden. Hierfür wurde ein Programm in Python geschrieben, das mehr als 8 Arten von Audio-Augmentation auf den Audio-Datensatz anwendet und die verarbeiteten Daten speichert. Dieser Schritt hilft sehr dabei, die Allgemeinbarkeit des Netzes und die Accuracy-Rate zu verbessern.
3. Synthetische Audiodaten erzeugen. Eine große Zahl an Audioproben mit verschiedenen Stimmen können künstlich durch Synthese erzeugt werden. Durch diesen Schritt lernt das Modell zu verallgemeinern und Stimmen mit weiblichen und männlichen Tonen zu erkennen. Dies ist sehr wichtig, da die aufgenommenen Audioproben nur von einer Person stammen. Ein in Python geschriebenes Programm wird ermöglichen, synthetische Audioproben zu generieren und Vorverarbeitungsprozesse auf die erzeugten Datenproben anzuwenden.

Die Abbildung 4-5 zeigt die Zusammensetzung des erstellten Datensatzes, der Aufnahmen und synthetische Audiodaten enthält.

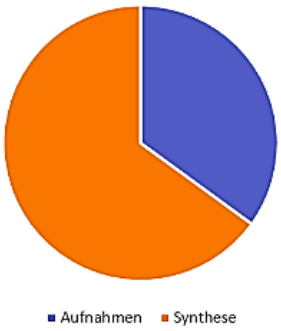


Abbildung 4-5: Die Zusammensetzung des Datensatzes

4.1.4 Augmentation von Audiodaten

Wie erwähnt, werden in dieser Arbeit verschiedene Arten von Augmentation-Methoden verwendet und wurde dafür ein Programm in Python geschrieben, das diese Augmentation-Methoden auf die Datenpunkten des Datensatzes anwendet und die resultierende Daten speichert.

Augmentation von Daten ist eine Erweiterung der Daten durch kleine realistische Änderungen oder Hinzufügen von zusätzlichen Signalen, sodass die erzeugten Daten als neue Daten betrachtet werden können. Das bedeutet, dass durch Augmentation die Größe des Datensatzes erhöht werden kann. Die Augmentation-Methoden, die verwendet werden, sind vielfältig und hängen von der Datennatur und der Form der Daten ab. Für die Audiodaten können die Augmentation-Methoden entweder direkt auf das Audio oder auf das Spektrogramm angewendet werden. Nachfolgend werden einige der in dieser Arbeit angewendeten Augmentation-Methoden aufgelistet und kurz beschrieben.

4.1.4.1 Shelving Filter

Hochpassfilter und Tiefpassfilter werden normallerweise verwendet, um unerwünschte Signale über oder unter einer festgelegten Frequenz zu entfernen. Die Shelving-Filter werden hingegen benutzt, um die Verstärkung der Signale unter oder über einer festgelegten Frequenz zu erhöhen oder zu verringern. Diese Arten von Filtern werden in Verstärkergeräten verwendet, um Effekte wie Bass oder Treble zu realisieren. Es gibt dabei zwei Shelving-Filter, Highselving-Filter Lowshelving-Filter.

4.1.4.2 Bandpass-Filter

Dieser Filter lässt nur die Frequenzen innerhalb eins Frequenzbandes passieren und dämpft die Frequenzen außerhalb dieses Frequenzbandes.

4.1.4.3 Zeitverschiebung (Shift)

Wie in Abbildung 4-6 gezeigt ist, wird das Audio-Signal nach rechts oder nach links verschoben, wobei die Audio-Daten zufällig verschoben werden.

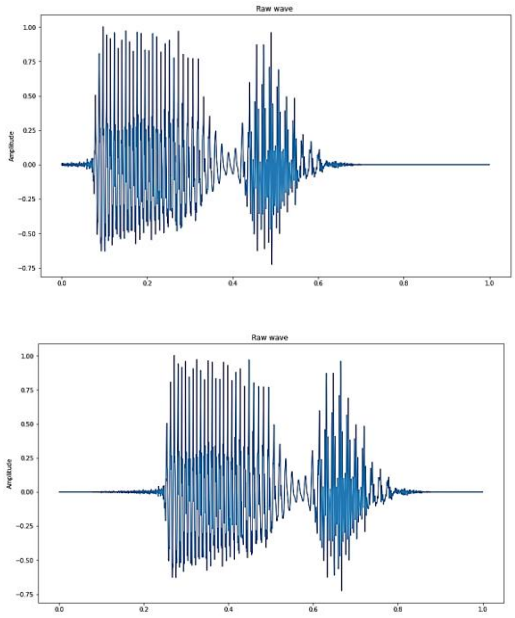


Abbildung 4-6: Zeitverschiebung

4.1.4.4 Sprachrate (Speed)

Bei dieser Art von Augmentation werden die Audiodaten beschleunigt oder verlangsamt. Dies ist wichtig, weil die Wörter mit unterschiedlichen Geschwindigkeiten gesprochen werden können. Die Abbildung 4-7 zeigt den Einfluss dieser Methode auf das Audiosignal.

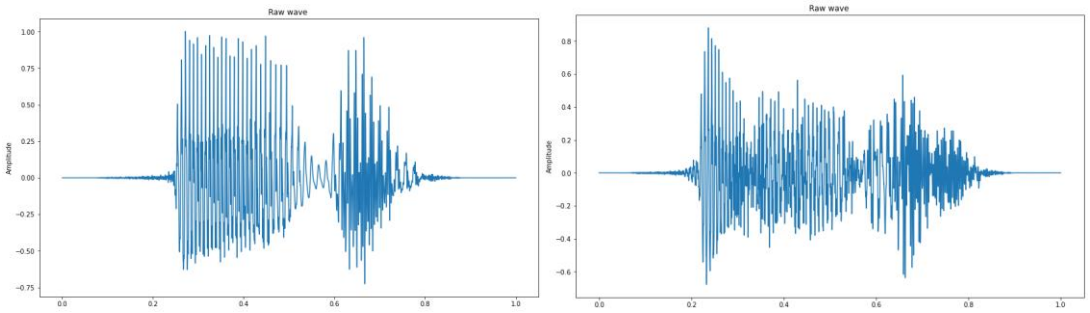


Abbildung 4-7: Änderung der Audiogeschwindigkeit.

4.1.4.5 Noise hinzufügen

Dabei wird ein zufälliges Geräusch generiert und dem Audiosignal hinzugefügt. Dies ist bedeutsam, weil das Audiosignal weißes Rauschen (Wight Noise), Geräusche aus der Umgebung und durch elektronische Bauelemente erzeugte Signale enthält. Die Abbildung 4-8 verdeutlicht das Hinzufügen von nach dem Zufallsprinzip generiertem Rauschen.

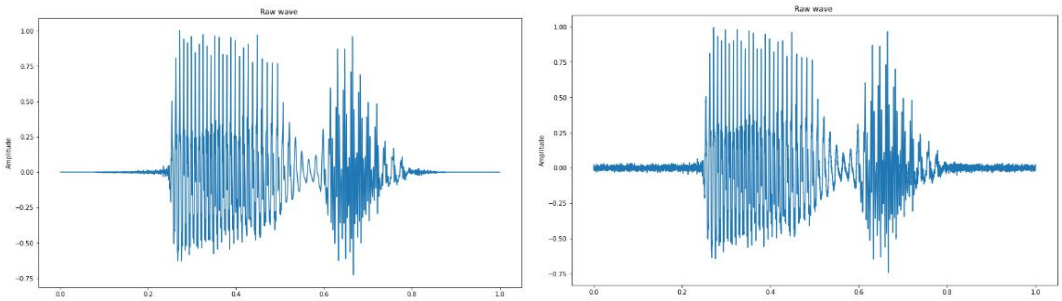


Abbildung 4-8: Das Hinzufügen von Noise-Signalen

4.1.4.6 Amplitude ändern

Da die natürlichen Signale mit unterschiedlichen Amplituden aufgenommen werden, wird die Amplitude des Signals in diesem Fall zufällig verstärkt oder gedämpft, sodass die Audioproben nach der Anwendung dieser Methode unterschiedliche Amplituden aufweisen.

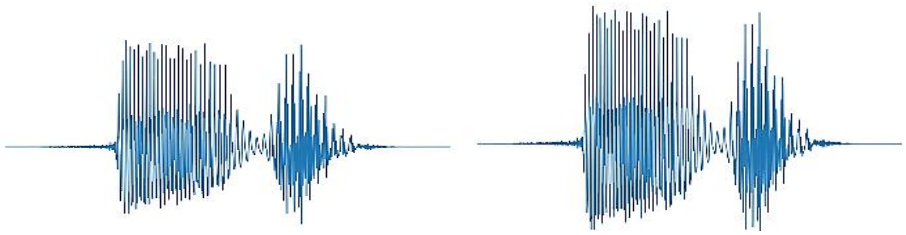


Abbildung 4-9: Augmentation durch die Änderung der Amplituden

4.1.4.7 Pitch ändern

Pitch ist die Tonhöhenänderung eines Audiosignals. Die Abbildung 4-10 zeigt die Änderung von Pitch eines Audiosignals.

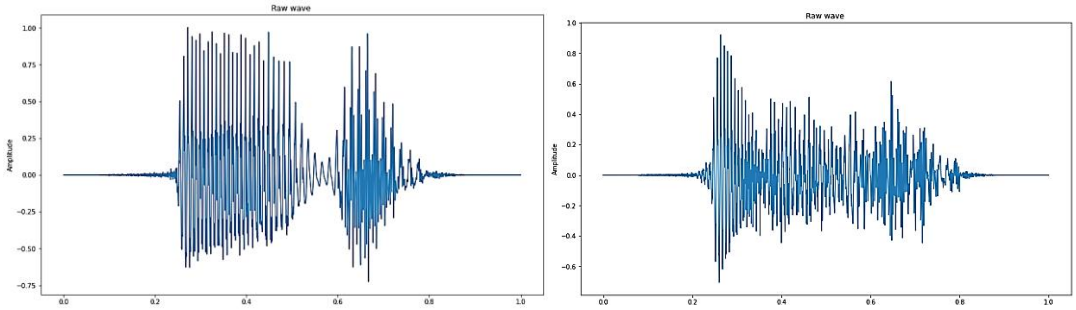


Abbildung 4-10: Die Pitch-Änderung

4.1.4.8 Reverb erzeugen

Normalerweise befindet sich das Mikrophon in einem Raum, wo die Schallwellen an Wänden und Gegenständen reflektiert werden. Diese reflektierten Signale werden dann nach einer Verzögerungszeit eintreffen und mit dem direkten Signal summiert. Dieser Effekt wird durch Erzeugung von Reverb realisiert.

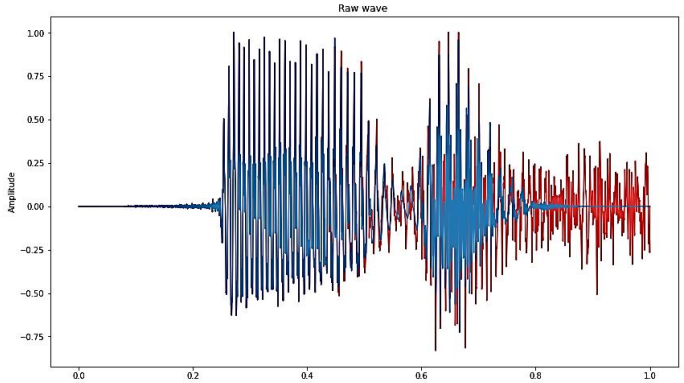


Abbildung 4-11: Veranschaulichung des Reverb-Effekts

4.1.4.9 High und Low Pass Filter

Bei Hochpassfilter werden die niedrigeren Frequenzen unterdrückt und die Hochfrequenzen durchgelassen. Die Abbildung 4-12 zeigt den Einfluss des HP-Filters auf das Signal. Ein Low Pass Filter unterdrückt hingegen die hohen Frequenzen und lässt die niedrigeren Frequenzen durch.

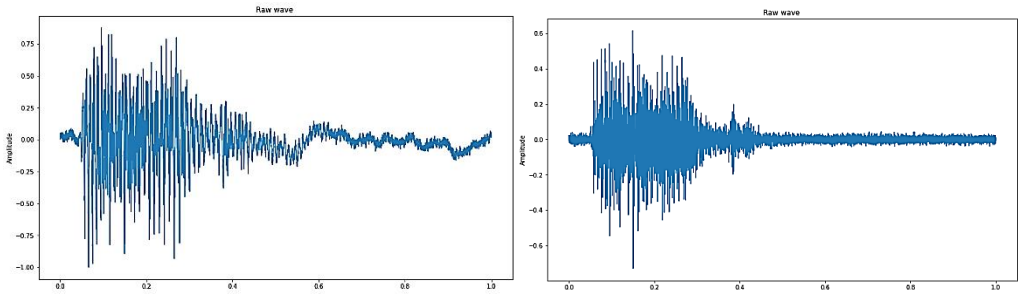


Abbildung 4-12: Der Einfluss des HP-Filters (Rechts).

4.1.4.10 Tremolo

Dieser Effekt wird normallerweise durch elektronische Komponenten generiert und ähnelt teilweise dem Flanging-Effekt. Dieser Effekt wird manchmal als Unterwasser-Effekt bezeichnet.

4.1.4.11 Die Skalierung der Daten

Die Skalierung von Daten wird normallerweise an die Trainingsdaten angewendet (in diesem Fall das Spektrogramm), bevor diese Daten in das neuronale Netz während des Trainings eingespeist werden. Dieser Schritt ist wichtig, weil neuronale Netze besser abschneiden, wenn sie auf Daten trainiert werden, deren Werte zwischen 0 und 1 variieren.

4.1.4.12 Normalisierung von Daten

Die Datenpunkte, die für das Training des Netzes verwendet werden, unterscheiden sich normallerweise voneinander und liegen in unterschiedlichen Wertebereichen. Dies verschlechtert die Trainingsergebnisse. Daher lohnt es sich die Audiodaten zu normalisieren. Die Normalisierung bedeutet hier, dass die Amplitude der Audiodaten so vergrößert oder verkleinert werden, dass sie in einem gemeinsamen Amplitudenbereich liegen.

4.1.4.13 Data Augmentation mit Python

Um die Anwendung von Data Augmentation an viele Audiodateien zu erleichtern, wurde ein Programm in Python geschrieben, das verschiedene Augmentation Methoden an einen Datensatz anwendet. Dieses Programm enthält eine Klasse namens augmentation. Der Folgende Listing 4-3 verdeutlicht die Verwendung dieser Klasse. Bevor diese Klasse im Programm instanziiert wird, werden der Klasse der

Pfad zu dem Datensatz auf der Festplatte und der Ordner, wo die Daten gespeichert werden müssen, übermittelt.

Der Datensatz enthält Unterordner, in denen Wav-Dateien gespeichert sind. Jeder dieser Unterordner hat den Namen eines Wortes (Klasse) und enthält die Trainingsdaten von diesem Wort. Um zu bestimmen, welche Data Augmentation auf welche Wörter angewendet werden müssen, werden die Klassennamen zwischen zwei ‘’ Zeichen geschrieben und mit Komma getrennt (Listing 4-3).

Listing 4-3:Die Instanz der Klasse zur Anwendung von Augmentation-Methoden

```
1. import augmentaion
2. from augmentaion import augmentation
3. save_dir
   ='/home/murad/Masterarbeit_Aufgaben/Spracherkennung/data_
   set'
4. ww= 'aus,ein'
5. aug = augmentation( dataset_path = None ,
6.                     save_path = save_dir ,
7.                     tar= False,
8.                     samples_ = 500)
9.    aug.crop( words_list = ww)
10.   aug.amplitude( words_list = ww)
11.   aug.mask(words_list = ww)
12.   aug.shift ( words_list = ww )
13.   aug.vtlp(words_list = ww )
14.   aug.speed(words_list = ww )
15.   aug.pitch ( words_list = ww )
16.   aug.stretch ( words_list = ww )
17.   aug.add_noise(words_list=ww)
18.   aug.Random_augmentaton(words_list =ww)
19.   aug.random_aug_2( words_list = ww,samples_zahl=1000  )
20.
```

Im Listing 4-3 ist beispielsweise zu sehen, dass die Klassen Aus und Ein ausgesucht worden sind. Die verarbeiteten Daten werden in einem Unterordner gespeichert, dessen Name auf die Klasse (das Wort) und die Augmentation-Methode hindeutet.

Das Programm benötigt einige Bibliotheken zur Verarbeitung von Audiodaten, die wichtigsten sind:

- Librosa ist eine Python-Bibliothek für Verarbeitung von Audiosignalen. Sie ermöglicht die Anwendung von Filtern und anderen Operationen wie Dezimation und Interpolation an die Audiosignale.
- Soundfile ist eine weitere Audio-Bibliothek, die das Schreiben, Lesen und Konvertieren von Audio-Dateien ermöglicht.
- Nlpaug ist eine Audio-Augmentation-Bibliothek. Sie bietet einige Augmentation-Methoden wie Zeitverschiebung und Pitch-Änderung.
- Audiomentations ist eine weitere Bibliothek für Audio-Augmentation. Sie bietet interessante Augmentation-Methoden.
- Pysndfx eine Interessante Audio-Bibliothek. Sie ermöglicht das Hinzufügen von Audio-Effekten wie Phaser und Shelving-Filter.

Es ist außerdem zu beachten, dass weitere Packages und Python-Bibliotheken installiert werden müssen.

Wie erwähnt, ist die Augmentation in dieser Arbeit bedeutsam. Denn das Netz wird ohne Anwendung von Vorverarbeitungen eine gute Accuracy-Rate haben, aber auf dem Mikrocontroller schlecht funktionieren. Der Grund dafür ist, dass das Netz ohne die Anwendung von Augmentation und Vorverarbeitungen nicht lernen kann, zu verallgemeinern(weil die Audiodaten nur von einer Person stammen). Außerdem unterscheiden sich die Trainingsdaten von den Daten, die mit dem Mikrophon auf dem Board während der Inferenz aufgenommen werden.

Es gibt auch Augmentation-Methoden, die auf das Spektrogramm angewendet werden können. Diese Methoden werden in dieser Arbeit(Außer Skalierung)nicht benutzt. Es ist zudem zu beachten, dass die Audiodateien Wave (.wav) sein müssen. Die Abbildung 4-13 veranschaulicht den Einfluss der Data Augmentation auf das Spektrogramm derselben Audiodatei.

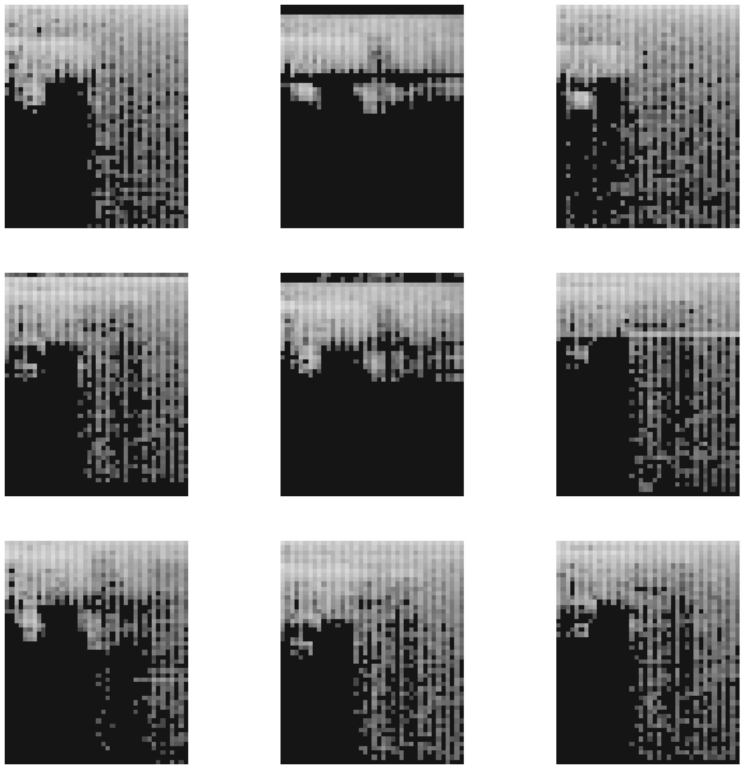


Abbildung 4-13: Der Einfluss der Augmentation auf das Spektrogramm

4.1.5 **Erzeugung synthetischer Audiodaten**

Beim Trainiernieren von KI-Modellen stellt der zum Trainieren erforderliche Datensatz einen Stolperstein dar. Deshalb würde nach alternativen gesucht, die die Erzeugung von Trainingsdaten vereinfacht. Eine dieser Alternativen ist die Verwendung von synthetischen Daten, die besonders gut in Objekterkennungsmodellen abgeschnitten haben. Es handelt sich dabei um künstliche Repräsentation von natürlichen Daten. Diese Daten werden von speziellen Algorithmen generiert, die in der Lage sind, natürliche Daten nachzuahmen. Diese Methoden werden in Bereichen wie Objekterkennung und die Analyse von Videomaterial eingesetzt. Diese Methode hat sich als erfolgreiche Strategie bewiesen und wird häufiger für Objekterkennungs-anwendungen eingesetzt.

Zur Erzeugung Synthetischer Audiodaten wird in dieser Arbeit ein mächtiges Werkzeug von Google eingesetzt, das unter der Bezeichnung Text to Speech bekannt ist**.** Dieser Dienst ist ein Bestandteil der Google Cloud Plattform. Hierbei werden geschriebene Texte in gesprochene Wörter mit zehn unterschiedlichen Stimmen und Tonen synthetisiert.

Für die Erzeugung von synthetischen Audiodaten wurde die Funktion audio_erzeuger in Python geschrieben. Diese Funktion generiert die erwünschten Daten mit zufälligen Eigenschaften durch Anwendung von Vorverarbeitungen, sodass möglichst unterschiedliche Stimmen und Tone erzeugt werden können. Um diese synthetischen Daten natürlicher zu machen, werden anschließend auf diese Daten unterschiedliche Augmentation-Methoden angewendet. Der Listing 4-4 veranschaulicht, wie die Funktion audio_erzeuger in einem Programm aufgerufen wird. Für mehr Informationen kann man die Code auf den Datenträger einsehen.

Der Parameter w_words bestimmt die erwünschen Wörter, die erzeugt werden müssen. Hierbei werden diese erwünschten Wörter zwischen zwei Anführungszeichen getrennt mit Komma angegeben.

Der Parameter augmentation_arten bestimmt die Augmentation-Methoden, die auf die erzeugten Daten angewendet werden müssen. Hierfür stehen mehrere Augmentation-Methoden zur Verfügung. Die erzeugten Daten werden dann in einem vom Benutzer angelegten Ordner gespeichert. Der Speicherort auf der Festplatte wird dabei durch den Parameter save_dir bestimmt.

Die Anzahl der generierten Datenpunkten werden durch den Parameter gewuenschte_samples festgelegt. Der Parameter separat kann entweder den Wert False oder True annehmen. Mit True werden alle Audiodaten mit verschieden Augmentation-Methoden getrennt in separaten Ordnern gespeichert, während mit False die verarbeiteten Daten in einem einzigen Ordner gespeichert werden.

Es muss beachtet werden, dass Google Text to Speech nur für eine bestimmte Zahl an erzeugten Daten kostenfrei zur Verfügung steht. Außerdem muss ein Google-Konto eingerichtet werden und die erforderlichen Packages und Bibliotheken müssen installiert werden, bevor dieser Dienst und diese Funktion verwendet werden können.

Der Listing 4-4 verdeutlicht, wie diese Funktion in einem Programm aufgerufen werden kann.

Listing 4-4: Der Aufruf von audio_erzeuger

```
save_dir =
'/home/murad/Masterarbeit_Aufgaben/Spracherkennung/data_set'
audio_erzeuger ( w_words ='aus',
                 lang='de' ,
                 augmentation_arten=
                 'shift,vtlp,noise,amplitude',
                 save_dir=save_dir ,
                 anfang = 0 ,
                 gewuenschte_samples= 100,
                 separat =False )
```

4.1.6 **Aufnehmen von Audiodaten**

Wie erwähnt, müssen für das Training reale Audiodaten aufgenommen werden, da es keinen Datensatz gibt, der erwünschen deutschen Wörter bietet. Die Aufnahme der Audiodaten findet in diesem Fall durch eine Funktion statt, das in Python geschrieben worden ist. Durch diese Funktion können beliebig Audiodateien der Format Wav aufgezeichnet werden. Der folgende Listing 4-5 zeigt die Parameterliste der Funktion und veranschaulicht, wie diese Funktion eingesetzt werden kann.

Listing 4-5: Die Parameterliste der Funktion aufnehmer

```
save_dir=
'/murad/Masterarbeit_Aufgaben/Spracherkennung/dataset'
folder_name = 'meine_recordings_2'
words_list = 'aus'
samples_anzahl = 200
dauer = 1
anfang = 200
aufnehmer ( save_dir = save_dir ,
            words_list = words_list
             ,samples_anzahl=samples_anzahl,
              dauer = dauer ,
              folder_name = folder_name ,
              anfang = anfang  )

```

Der Speicherort , wo die Aufnahmen gespeichert werden müssen, wird der Funktion durch den Parameter save_dir übergeben. Wie bei der Erzeugung von synthetischen Daten werden hier auch die erwünschten Wörter zwischen zwei Anführungszeichen getrennt mit Komma angegeben. Dies dient dazu, dass das Programm automatisch einen Ordner für jedes Wort anlegt und die Aufnahmen speichert. Ein wichtiger Parameter ist der Parameter Dauer, der die Dauer der Audioaufnahmen bestimmt (in diesem Fall 1 Sekunde). Der Parameter samples_anzahl bestimmt die Anzahl der aufzunehmenden Audioproben, während der Parameter anfang bestimmt, ab welcher Zahl die Aufnahmen nummeriert werden müssen.

Wenn die Funktion aufgerufen wird, werden automatisch Aufnahmen gemacht. Bei jeder Aufnahme gibt das Programm dem Benutzer einen Hinweis, dass jetzt das Wort ausgesprochen werden muss, um aufgenommen zu werden. Wenn die Aufnahmezeit abläuft (in diesem Fall eine Sekunde) zeigt das Programm das Spektrogramm an und spielt das gerade aufgenommene Audio ab, um sichergestellt zu werden, dass die Aufnahme akzeptable Qualität hat. Anschließend fragt das Programm den Benutzer, ob die Aufnahme gespeichert oder verworfen werden soll. Dieses Programm beschleunigt sehr das Aufnehmen von Audiodaten.

Es muss hierbei beachtet werden, dass einige Bibliotheken installiert werden müssen, bevor die Funktion verwendet werden kann.

4.1.7 **Trainieren des Modells**

Nachdem der Datensatz erstellt worden ist, wird nach einer Architektur gesucht, die alle Anforderungen erfüllt und funktionsfähig auf dem Mikrocontroller ist. Hierfür gibt es zwei Möglichkeiten, entweder wird eine neue Architektur entworfen, oder wird eine bereits existierende Architektur verwendet, die eine gute Accuracy-Rate aufgewiesen hat.

Die Implementierung neuronaler Netze auf dem Mikrocontroller ist ein Spezialfall, wobei die Architektur zeitliche Anforderungen erfüllen muss. Daher ist es nicht selten, dass eine Architektur für jedes Klassifizierungsproblem gefunden werden muss.

Nachfolgend wird ein Spracherkennungsmodell gefunden, das auf dem Mikrocontroller funktioniert und gute Accuracy-Rate aufweist. Außerdem wird untersucht, wie sich die

Quantisierung auf die Accuracy-Rate auswirkt und wie schnell das Modell auf dem Mikrocontroller ausgeführt werden kann.

4.1.7.1 Architektur des Modells

Die beste mögliche Modellarchitektur kann durch Versuchen und Experimentieren gefunden werden. Für die zweidimensionalen Daten wie Bilder oder auch in diesem Fall Spektrogramme passen die CNN-Netze am besten. Außerdem muss berücksichtigt werden, dass nicht alle Strukturen unterstützt sind und das Modell kleinstmöglich sein muss. Zu diesem Zweck wurden zahlreiche Netze trainiert, verglichen und auf dem Mikrocontroller ausgeführt. Die beste Architektur, die diese Bedingungen erfüllt, hat hauptsächlich drei CNN-Schichten für das Erlernen von Merkmalen, und eine FC-Schicht gefolgt von Softmax-Schicht für die Klassifizierung. Die Abbildung 4-14 zeigt Visualisierung dieser Architektur unter Verwendung von Tensorboard.

Jeder Faltungsschicht hat Mehrere Filter, und jeder Filter ist ein rechteckiges Fenster mit einer Breite und einer Höhe, das über die Eingabe verschoben wird. Der Ausgabe der Schicht stellt die Übereinstimmung zwischen dem Filter und der Eingabe an jedem Punkt des Eingabebilds dar. Jeder Filter hat ein Muster, das während des Trainierens gesucht wird. Wenn die Eingabe eine Ähnlichkeit zu diesem Muster aufweist, dann wird ein höher Wert in den entsprechenden Teil des Ausgabebilds geschrieben. Jede Faltungsschicht erzeugt nicht nur ein Ausgabebild, sondern N, wobei N die Anzahl der Filter ist. Zum Beispiel erzeugt die erste Faltungsschicht in diesem Modell 8 Ausgabebilder, die Kanäle genannt werden. Die Größe jedes Ausgabebild hat dieselbe Größe wie das Eingabebild, weil die Schrittweite (stride) auf 1 gesetzt wurde.

Die Eingabe des Netzes hat die Form (1,1960) und wird durch eine Reshape-Schicht in die Form (1,49,40,1) umgewandelt. Diese sind die Dimensionen des Spektrogramms.

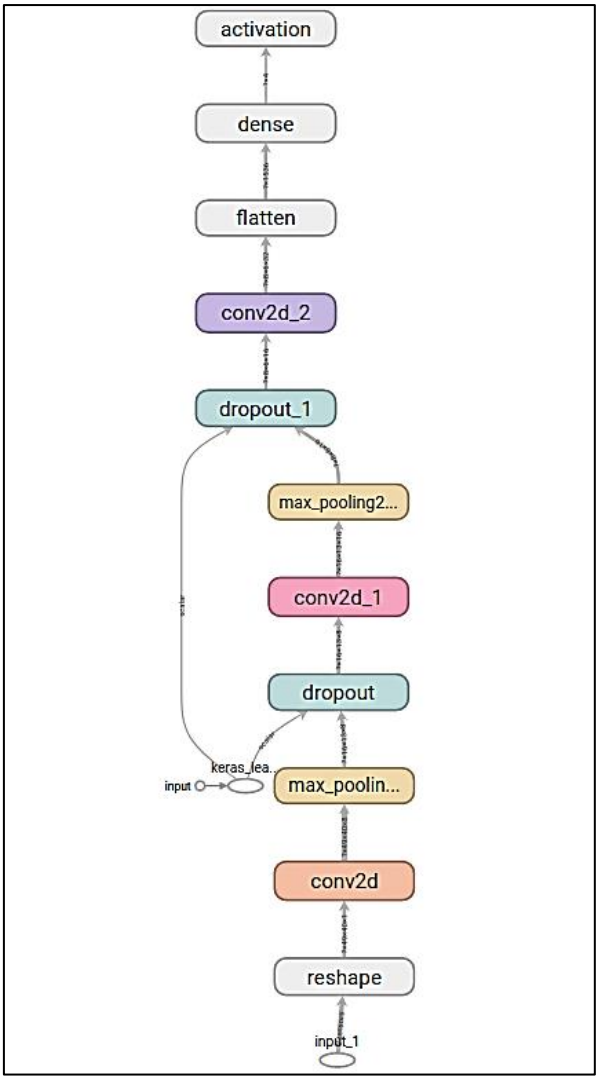


Abbildung 4-14: Eine mit Tensorboard erzeugte Visualisierung des Modells

Zwischen den Faltungsschichten befinden sich Max-Pooling-Schichten, die die Größe der vorigen Ausgabe verringert und somit den Rechenaufwand und den Speicherbedarf, wie bereits in Kapitel 2erwähnt wurde. Außerdem es ist zu sehen, dass Regularisierung-Schichten (Dropout) verwendet werden, die der Vermeidung der Überanpassung und der Erhöhung der Lernbarkeit des Netzes dienen. Diese Schichten werden nur beim Trainieren aktiviert, während bei der Ausführung abgeschaltet werden. Die letzte Schicht hat 4 Ausgänge, die die vier Klassen darstellen. Die 4 Klassen sind dabei Ruhe, unbekanntes Wort, Ein und Aus.

Ruhe bedeutet, dass es nur Hintergrundgeräusche gibt, während unbekanntes Wort alle Wörter und Stimmen darstellt, die das Netz nicht kennt. Ein und Aus sind hierbei die Wörter, die das Netz erkennen soll. Die Abbildung 4-15 verdeutlicht die Funktionsweise der Faltung und zeigt die Ausgabe der drei Faltungsschichten.

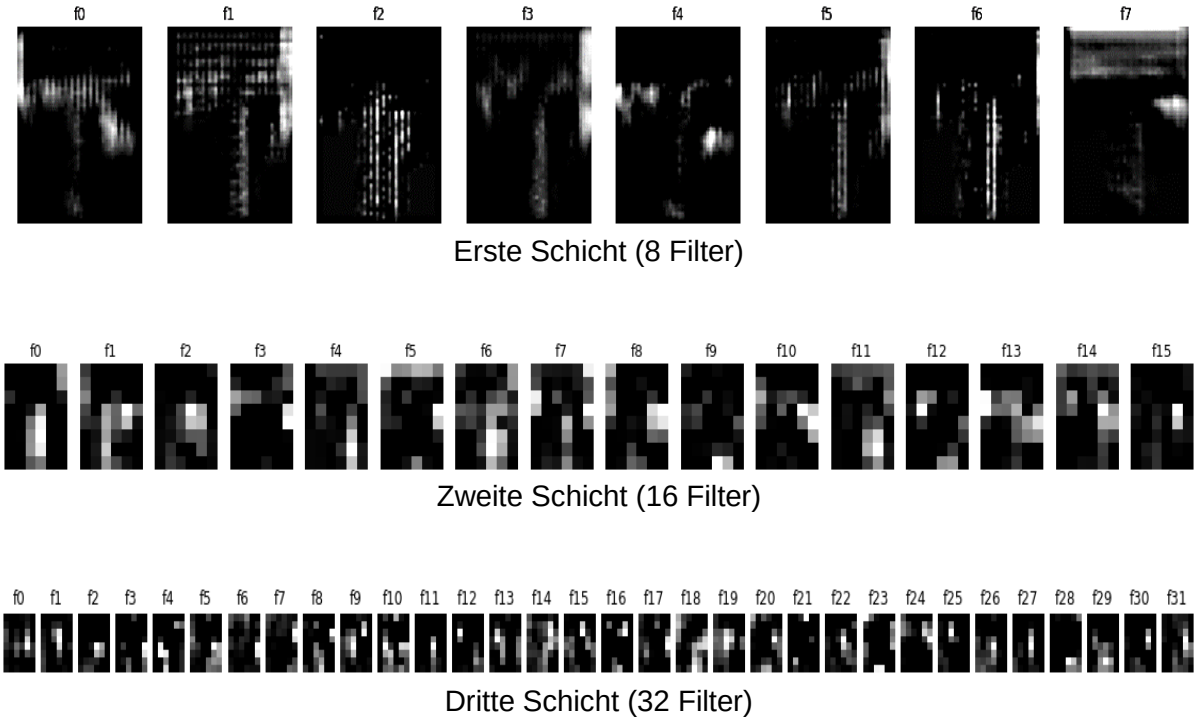


Abbildung 4-15: Die Ausgabe der Faltungsschichten

Die Faltungsschichten sind von einer Flatten-Schicht gefolgt. Diese Schicht wandelt die Ausgabe der letzten Faltungsschicht in ein eindimensionales Array, das sich als Eingabe der FC-Schicht eignet, wobei die FC-Schicht die Ausgabe der Faltungsschichten klassifiziert. Die Softmax-Schicht am Ausgang des Netzes erzeugt eine Wahrscheinlichkeitsverteilung auf alle Labels (Kapitel 2) und trägt dazu bei, den Unterschied zwischen dem höchsten Wert des Ausgangs und den konkurrierenden Klassen zu erhöhen.

4.1.7.2 Trainingsvorgang

Bis jetzt wurden der Datensatz und das Modell erstellt. Jetzt muss das Netz kompiliert und trainiert werden. Davor müssen die Verlustfunktion (Loss), die Optimierungsfunktion und die Lernrate bestimmt werden. Als Verlustfunktion wurde hier für den Trainingsprozess die ‘MSE’ (mittlere quadratische Abweichung) ausgewählt.

Adam-Optimierung kann anstelle von Gradientenabstiegsverfahren als Optimierungsmethode verwendet werden, um die Gewichtungen des Netzes

anzupassen. Diese Optimierungsmethode ist nicht Rechenintensiv und passt gut für große Datensätze. Sie verwendet eine adaptive Lernrate bei Default, kann aber als vordefinierter Wert eingegeben werden, wobei Adam die Standardoptimierungs-methode in Tensorflow ist.

Das Netz lernt am Anfang mit großen Schritten, deshalb muss die Lernrate größer sein. Am Ende des Trainierens wird hingegen das Lernen langsamer mit kleinen Schritten. Daher muss die Lernrate klein sein. Hierbei wurde das Netz mit 3 unterschiedlichen Lernraten trainiert. Die Abbildung 4-16 zeigt die drei Lernphasen.

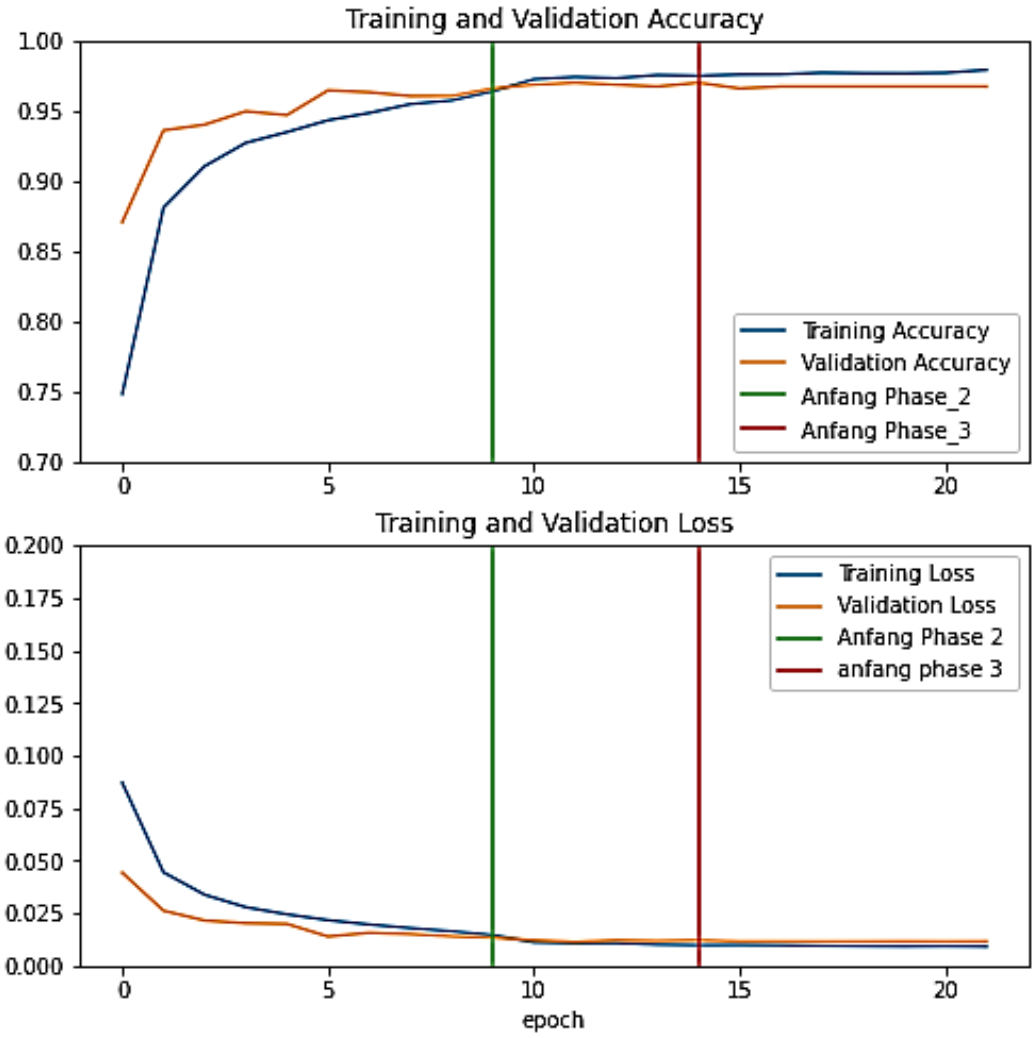


Abbildung 4-16: Training des Netzes in drei Phasen

4.1.7.3 Tensorflow-Lite-Modell

Tensorflow-Lite ist ein Teil der Tensorflow API und wurde zur Ausführung der ML-Modelle auf mobilen und IoT-Geräten entwickelt. Zum Beispiel können ML-Modelle auf Android-Geräten implementiert werden, wobei nur ein Teil der Tensorflow-operationen

unterstützt sind. Von dieser Bibliothek wurde eine noch kleinere Bibliothek namens Tensorflow-Lite for Microcontroller abgeleitet. Diese Bibliothek wurde zur Implementierung von ML-Modellen auf Mikrocontrollern entwickelt und unterstützt nur einen kleinen Teil der Tensorflow-Lite Operationen und Strukturen (Anhang B).

Nachdem das Training des Modells beendet wird, wird das Modell quantisiert, wobei Die Quantisierung durch einen Konverter stattfindet. Die Abbildung 4-17 veranschaulicht die Funktionsweise des Konverters.

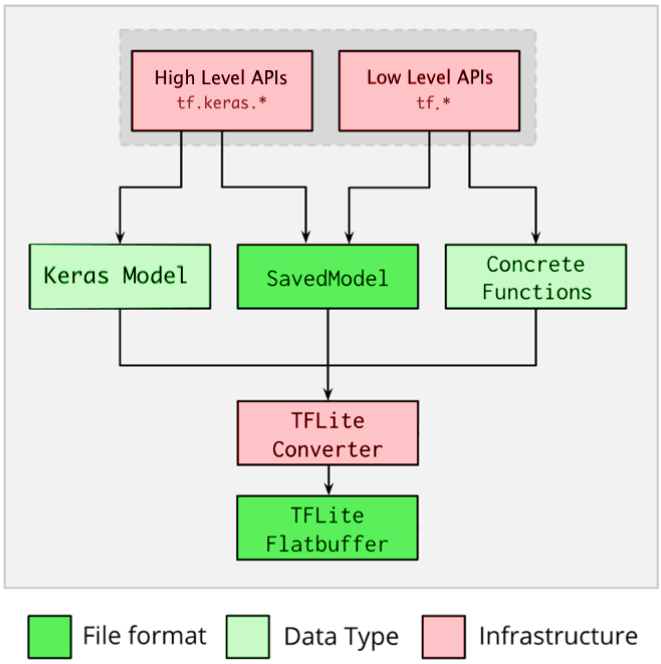


Abbildung 4-17: Die Erzeugung von TFLite-Modellen [23]

Ein in Tensorflow gespeichertes Modell kann unterschiedliche Formen annehmen, Je nachdem welche Dateiform benötigt wird und in welcher TF-API-Ebene oder (Level) das Modell geschrieben wurde. Hier kann man zwischen zwei unterschiedlichen Ebenen der Tensorflow-API unterscheiden, Low Level und High Level. Mit höheren Ebenen (Keras) kann man Modelle erstellen und trainieren, ohne dass auf der Details zu konzentrieren. Hingegen hat die Low Level eine niedrige Abstraktionseben und wird von Entwicklern verwendet, um neue Strukturen beispielsweise zu untersuchen, wobei der Datenfluss im Modellgraf mit allen Tensoren vollständig entworfen wird. Durch den Konverter können diese Formen des Tensorflow-Modells in sogenannte TFLite-Flatbuffer umgewandelt werden. Diese Datei enthält alle Informationen über den Modellgraph und die Gewichtungen des Modells, die zur Ausführung des Modells

notwendig sind. Durch den Konverter kann zudem die Quantisierungsart ausgewählt werden, wobei es drei Quantisierungsarten gibt (Kapitel 2).

Wie vorher erwähnt wurde, ist die Suche nach einem passenden Modell für die Implementierung auf dem Mikrokontroller sehr mühsam und zeitintensiv. Zudem gibt es in Tensorflow viele Quantisierung- und Konverter-Typen, die möglicherweise einsetzbar sind. Deswegen wurde für die Konvertierung eine in Python geschriebene Funktion verwendet, die TFLite-Modelle unter Verwendung von verschiedenen Konverter-Arten erzeugen kann(Listing 4-6).

Listing 4-6:Der Konverter

```
1. gen_shape = [1,1960]
2. generator_variable= train_data_shuffled
3.
4. def dataset_gen():
5.     for i in range(100):
6.         data= generator_variable[i]
7.         flattened_data = np.array(data.flatten(),
8.         dtype=np.float32).reshape(gen_shape )
9.         yield [flattened_data]
10.
11.   tflite_model_path =\
12.   '/home/murad/Masterarbeit_Aufgaben/Spracherkennung/mode
   l_konvertiert'
13.
14.   model_konverter ( model=model_class.model   ,
15.                     save_dir =tflite_model_path   , \
16.                     konverter_type =2,
17.                     generator =
18.                     dataset_gen , \
19.                     kommentar ='SimpleRNN_',
20.                     quantization_type ='int8' )
```

Angenommen, ein trainiertes Modell (model) muss in TFLite konvertiert werden. Dafür müssen der Funktion die erforderlichen Parameter übergeben werden. Dadurch erzeugt die Funktion das TFLite_modell und speichert es in einem vordefinierten

Ordner auf der Festplatte. Der Parameter Konverter_type gibt an, welche Konverter-Type verwendet werden muss und kann die Werte 1 bis 6 annehmen, wobei jeder Wert einen Konverter-Typ darstellt. Die Tabelle 4-1 zeigt die Konverter-Arten.

Tabelle 4-1: Mögliche Konverter-Typen

Konverter_type	Quantisierungsart
1	Erzeugt ein Float32-Modell.
2	Erzeugt ein Int8/uint8-Modell.
3	Erzeugt auch int8/uint8-Modell.
4	Erzeugt ein Float16-Modell.
5	Ein anderer Weg zur Erzeugung von int8/uint8
6	Eine dritte Methode für int8/uint8-Modelle

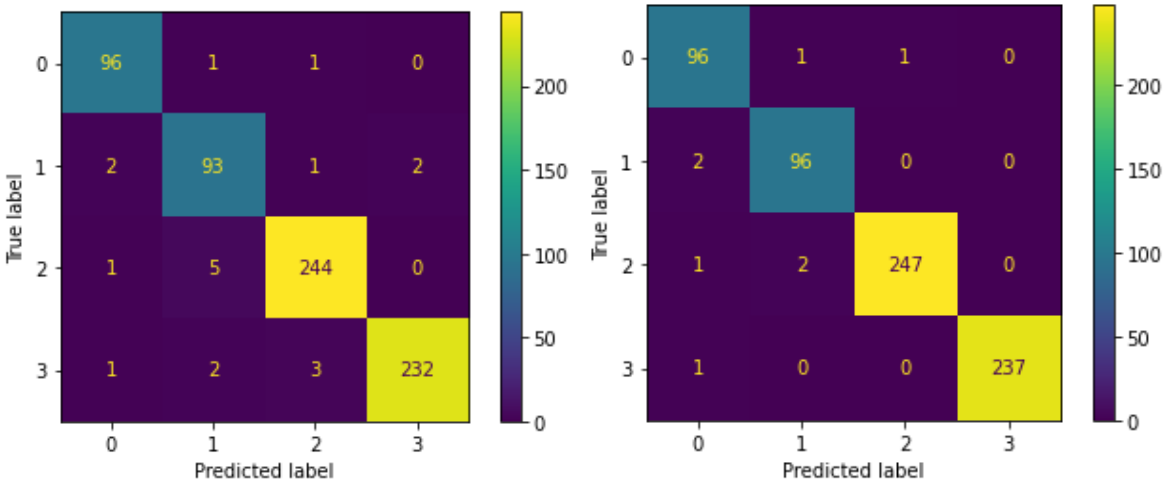
Für die Spracherkennungsanwendungen in dieser Arbeit werden die trainierten Modelle mit dem ersten und zweiten Konverter konvertiert. Ein wichtiger Parameter ist dabei der generator. Dieser Parameter wird benötigt, wenn ein int8 beziehungsweise uint8 Modell erzeugt werden muss. Dieser Generator übermittelt dem Konverter die maximalen und minimalen Werte, die in der Eingabe auftreten können, indem der Generator Dataset_Generator aufgerufen wird. Der Inhalt des Generators wird anschließend dem Konverter übergeben. Dieser Generator muss mindestens 100 Datenpunkte von Eingabedaten enthalten. Der Parameter quantization_type bestimmt welche Quantisierungstechnik (int8/uint8) verwendet wird.

Nachdem das Modell trainiert worden ist, wird es mit unterschiedlichen Konverter-Typen konvertiert, um den Einfluss der Quantisierung auf die Größe und die Accuracy-Rate des Modells zu untersuchen (Tabelle 4-2).

Tabelle 4-2: Der Einfluss der Quantisierung auf die Accuracy-Rate und die Modellgröße

Quantisierungsart	Die Modellgröße	Accuracy-Rate	Anzahl der Gewichtungen
Ohne Quantisierung (Flot32)	71372 Bytes	98.830409%	17084
Uint8/Int8	23408 Bytes	97.222222%	17084
Float 16 Bit	37000 Bytes	98.830409 %	17084

Durch die Quantisierung werden die Gewichtungen mit kleinerer Auflösung dargestellt, was zu einer kleinen Verschlechterung der Accuracy-Rate führt. Was die Modellgröße angeht, wird sie wie erwartet viel kleiner mit der Quantisierung. Die Quantisierung mit Float16-Zahlen weist hierbei keine Beeinträchtigung der Accuracy-Rate und eine Verringerung der Modellgröße um fast 50% auf. Die Abbildung 4-18 zeigt die Konfusionsmatrix beider Modelle.



(a) Int8-Modell (b) Float32-Modell

Abbildung 4-18: Die Konfusion-Matrix

Der Konfusion-Matrix kann entnommen werden, dass einige Testdaten nach der Quantisierung falsch klassifiziert werden. Dies ist bei den Klassen 3,2 und 1 zu sehen. Die Abbildung 4-19 zeigt eine Visualisierung des quantisierten Modells unter Verwendung von Netron.

4.1.7.4 Das Modell in ein C-Array konvertieren

Mikrocontroller haben normalerweise kein File-System (sehe 2.5.11) . Das bedeutet, dass das Modell in eine Form konvertiert werden muss, die in ein Mikrocontroller-programm integriert werden kann. Diese Form ist ein C-Array (sehe Abbildung 2-26).

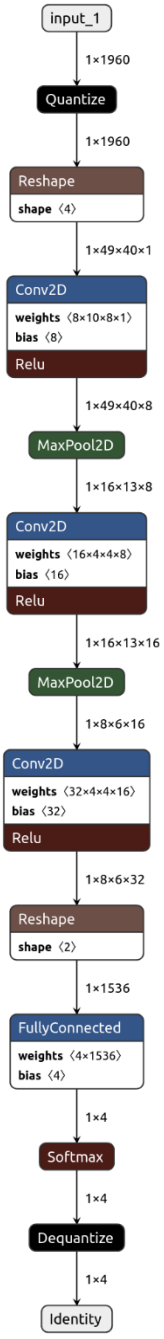


Abbildung 4-19: Das quantisierte Modell.

4.1.8 **Ausführung des Modells auf dem Mikrocontroller**

Bis jetzt wurde das Modell entworfen, trainiert und in Tensorflow-Lite konvertiert. Jetzt muss es auf dem Mikrocontroller ausgeführt werden. Dieser Schritt erfordert das Schreiben von einem Programm für den Mikrocontroller, in das das Modell integriert werden kann. Dieses Programm muss in C geschrieben werden, und ändert sich, je nachdem auf welchem Mikrocontroller implementiert werden muss.

Da die ESP-Boards leistungsfähig sind und über Kommunikationsmöglichkeiten verfügen, die in dem späteren Projekt zunutze gemacht werden können, empfehlt es sich das Modell auf diesem Board zu implementieren. Das Programm für ESP-Boards hat eigene Struktur und muss Mithilfe von ESP-IDF (ESP IoT Development Framework) geschrieben und kompiliert werden. ESP-IDF ist eine Sammlung von Bibliotheken und Werkzeugen, die alles beinhaltet, was bei Programmierung von ESP-Boards benötigt wird. [24] Die Ermittlung der zeitlichen Angaben und Inferenzzeiten auf dem Mikrocontroller (Tabelle 4-3) veranschaulicht den Einfluss der Quantisierung auf den Rechenaufwand. Zudem ist es besonders wichtig zu ermitteln, wie groß der Speicherbedarf während der Inferenz ist.

Tabelle 4-3 : Die Inferenzzeiten und der Speicherbedarf auf dem ESP-EYE

Das Modell	Modellgröße	Inferenzzeit Millisekunde	Benötigter Arbeitsspeicher (RAM)	CPU-Taktfrequenz
Quantisiertes Modell Uint8/Int8	23408 Bytes	305	25 K Byte	240 MHz
		500.6	25 K Byte	160 MHz
		966	25 K Byte	80 MHz
Ohne Quantisierung (Float32)	71372 Bytes	301	80 K Byte	240 MHz
		600	80 K Byte	160 MHz
		905	80 K Byte	80 MHz

Die Ergebnisse in Tabelle 4-3 zeigen eine große Verringerung des benötigten Arbeitsspeichers für das quantisierte Modell. Das ist erwartet, weil das quantisierte Modell Int8- oder Uint8-Zahlen verwendet, während die Gewichtungen beim nichtquantisierten Modell mit Float32-Zahlen dargestellt werden. Was nicht erwartet ist, dass die Inferenzzeit beim quantisierten Modell ähnelt der des nichtquantisierten Modells beim 240 MHz, ist kleiner bei 160MHz und ist größer beim 80MHz. Was hingegen erwartet wird, dass das quantisierte Modell schneller sein müsste, weil alle Operationen ebenfalls Integrer 8 Bit oder Unsigned integer 8 Bit sind, was bedeutet, dass die Operationen schneller ausgeführt werden müssten. Der Grund hierfür könnte sein, dass es beim quantisierten Modell zusätzliche Schichten (Quantize und Dequantize) für Umwandlung der Werte am Ein- und Ausgang verwendet werden (Abbildung 4-19).

Es ist hier zu beachten, dass das Modell mit Tensorflow 2.2 konvertiert worden ist. In späteren Tensorflow-Versionen wird eine weitere Konvertierungsmethode verwendet,

wobei die Quantisierung der Aus/Eingänge während der Konvertierung stattfindet, was bedeutet, dass das konvertierte Modell in diesem Fall keine Quantize und Dequantize Schichten beinhaltet (sehe Abbildung 2-13).

4.1.9 **Das C-Programm**

Das Programm auf dem Mikrocontroller wird in C geschrieben, wobei es mehrere Komponenten und Verarbeitungseinheiten enthält (Abbildung 4-1). Diese Verarbeitungseinheiten und Komponenten umfassen:

- Die Aufnahme von Audiosignalen.
- Extraktion von Spektrogrammen
- Die Ausführung des Erkennungsmodells
- Irgendwas machen (auf LCD etwas anzeigen, ein Gerät Steuern oder etwas zum Computer per serielle Port senden).

Diese Funktionen werden mehrere Male pro Sekunde aufgerufen. Die benötigte Zeit für die Ausführung hängt aber von der Taktfrequenz des Mikrocontrollers und dem Modell selbst ab. Deshalb unterscheidet sich die Performanz, je nachdem auf welchem Board das Modell ausgeführt wird.

Wie bereits besprochen wurde, ist das Fensters eine Sekunde lang und wird über das Audiostream in Schrittweiten verschoben. Diese Schrittweite entspricht eigentlich der Inferenzzeit des Programms. Das bedeutet auch, dass mehr als ein Fenster das Wort oder Teile des Worts umfassen werden. Die Abbildung 4-20 zeigt das gleitende Fenster über das Audiosignal. Hier taucht ein Problem auf. Denn es gibt Wörter, die mehrere Silben haben, wobei eine oder mehrere der Silben einem Wort ähneln könnten, auf die das Modell trainiert wurde. In diesem Fall wird das Modell mit hoher Wahrscheinlichkeit die Silbe einer Klasse zuordnen, was eine Falsche Vorhersage darstellt.

Angenommen, das Modell wird auf die Wörter ‘Aus’ und ‘Ausgang’ trainiert, und das Fenster die erste Silbe des Worts Ausgang einlest. In diesem Fall wird das Modell eine hohe Wahrscheinlichkeit geben, dass das gesprochene Wort ein ‘Aus’ ist und somit erzeugt es eine falsche Vorhersage . Hier kann davon ausgegangen werden, dass sich man nicht auf eine einzige Vorhersage verlassen kann. Um dieses Problem zu beseitigen, wird stattdessen der Durschnitt der Wahrscheinlichkeiten (Vorhersagen)

von mehreren aufeinanderfolgenden Fenster berechnet. Jetzt wird dasselbe Szenario mit Berechnung von mehreren Vorhersagen wiederholt.

Das erste Zeitfenster positioniert sich gegenüber der Silbe ‘Aus’ von dem Wort ‘Ausgang’ und das Modell wird damit das Wort als ‘Aus’ klassifizieren. Jetz nach einer bestimmten Zeitspanne (Inferenzzeit) wird sich das Zeitfenster bewegen und das Wort ‘Ausgang’ vollständig einlesen. Jetzt wird das Modell eine hohe Wahrscheinlichkeit geben, dass das erkannte Wort ‘Ausgang’ ist. Das wird mit dem dritten oder vielleicht vierten Fenster wiederholt. Nach jeder Vorhersage wird der Durchschnitt der vergangenen Wahrscheinlichkeiten berechnet. Der höchste Durchschnitt wird in diesem Fall auf das Wort ‘Ausgang’ deuten. [16]

Wenn der verwendete Prozessor Leistungsstark ist, wird sich das Zeitfenster in kleinen Schritten bewegen. Dies ergibt, dass mehr Fenster das Wort oder Teile des Wortes umgeben werden und somit werden mehr Finster für die Berechnung des Durchschnitts genommen. Im Gegensatz, wenn der Prozessor nicht leistungsfähig oder das Modell rechenintensiv ist, wird das Fenster mit größerem Zeitschritten springen. In diesem Fall wird der Durchschnitt von wenigen Vorhersagen berechnet.

Die Ausgabe des Modells ist der Ausgang der Softmax-Schicht, wobei die Summe aller Wahrscheinlichkeiten eins beträgt. Die Wahrscheinlichkeiten sowie der Durchschnitt werden als Int8-Zahlen dargestellt. Als akzeptabel wird das Ergebnis angesehen, wenn es einen Schwellenwert überschreitet. Anderenfalls wird dieses Ergebnis verworfen. Der beste Schwellenwert unterscheidet sich von einem Modell zu einem anderen und wird durch Experimentieren eingestellt. Ebenso wird die geeignete Anzahl der Ergebnisse, von denen der Durchschnitt berechnet wird, durch Experimentieren gefunden. Je größer der Schwellenwert ist, desto sicherer ist die Erkennung (wenige Fehler bei der Erkennung). Mit größerem Schwellenwert lässt die Sensitivität aber nach.

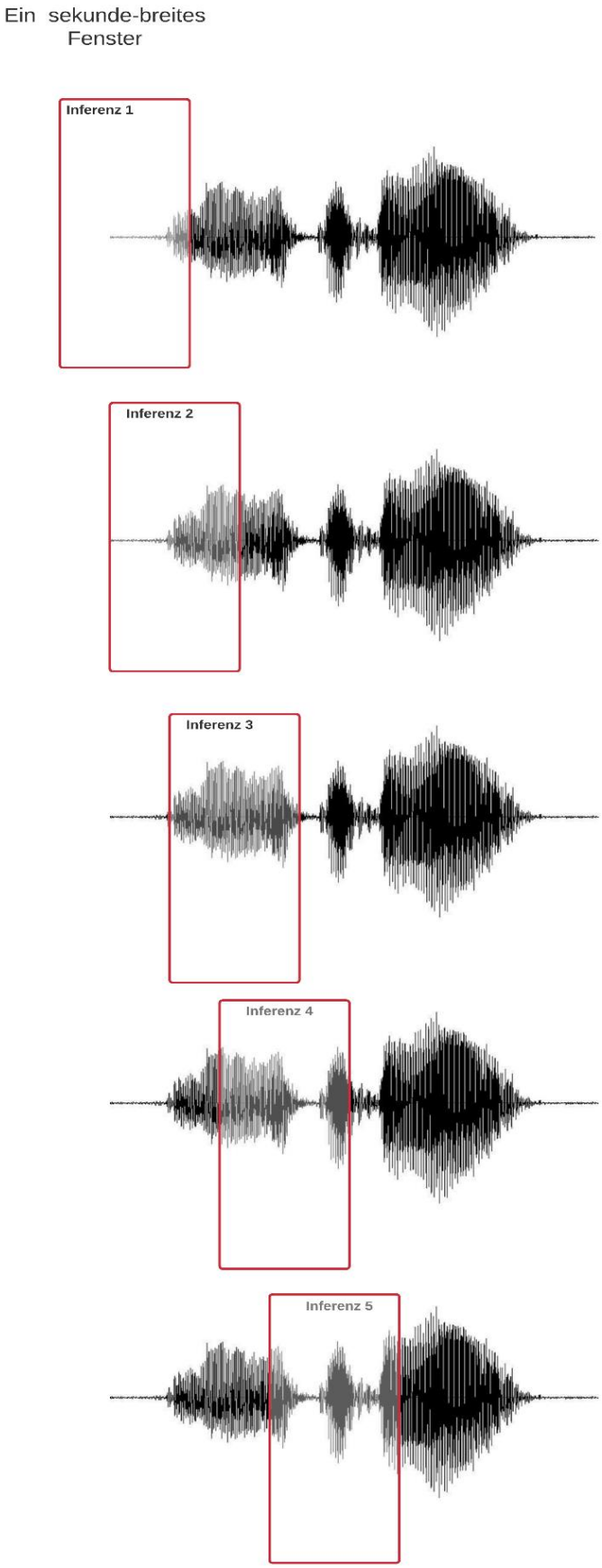


Abbildung 4-20: Gleitendes Fenster über das Audiosignal

Während der Ausführung des Modells, dauert im Hintergrund des Programms die Aufnahme des Audiosignals an. Die innerhalb der Inferenz aufgenommene Audiodaten werden in einen Ring-Buffer vorläufig bis die nächste Inferenz verschoben, wobei Die Datenübertragung über einen I2S-Bus stattfindet. Dieser Bus wurde von Philips zur Übertragung von Audiodaten entwickelt. Die Abbildung 4-21 veranschaulicht die Funktionsweise des Ring-Buffers.

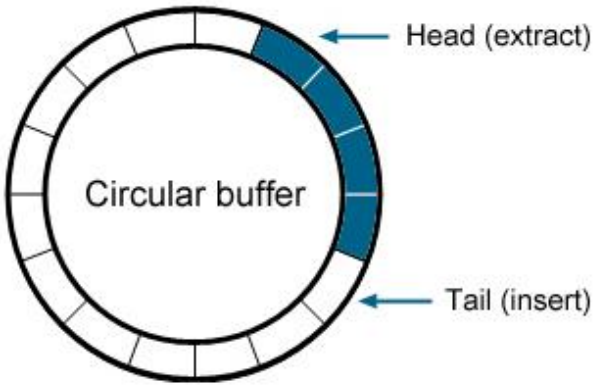


Abbildung 4-21: Der Ring-Buffer. [25]

Der Ring-Buffer ist ein flüchtiger Zwischenspeicher mit zwei Zeiger Tail und Head. Wenn Ein Wert geschrieben wird, wird der Tail-Zeiger verschoben und die verfügbaren Speicherplätze werden eine Zelle weniger. Im Gegensatz, wenn ein Wert gelesen wird, wird der Head-Zeiger um eine Adresse in dieselbe Richtung verschoben und ein Speicherplatz wird befreit.

In diesem Projekt beträgt die Abtastrate 16000 Hz und die Auflösung 16-Bits. Der Abbildung 4-20 kann entnommen werden, dass ein Teil der Abtastungen, die in einem Zeitfenster liegen, werden in dem nächsten Fenster wiederholt. In anderem Worten wird das Spektrogramm für dieses Teil der Abtastungen mehr als einmal berechnet. Dies stellt eine unnötige Erhöhung der Anzahl der arithmetischen Operationen zur Berechnung des Spektrogramms dar. Eine bessere Möglichkeit wäre, wenn das Spektrogramm nur für die neu eintreffenden Abtastungen berechnet und mit dem bereits berechneten Anteil des vorherigen Fensters zusammengefügt wird, um ein vollständiges Spektrogramm für das neue Fenster zu bilden. Die Abbildung 4-22 veranschaulicht das Berechnen des Spektrogramms.

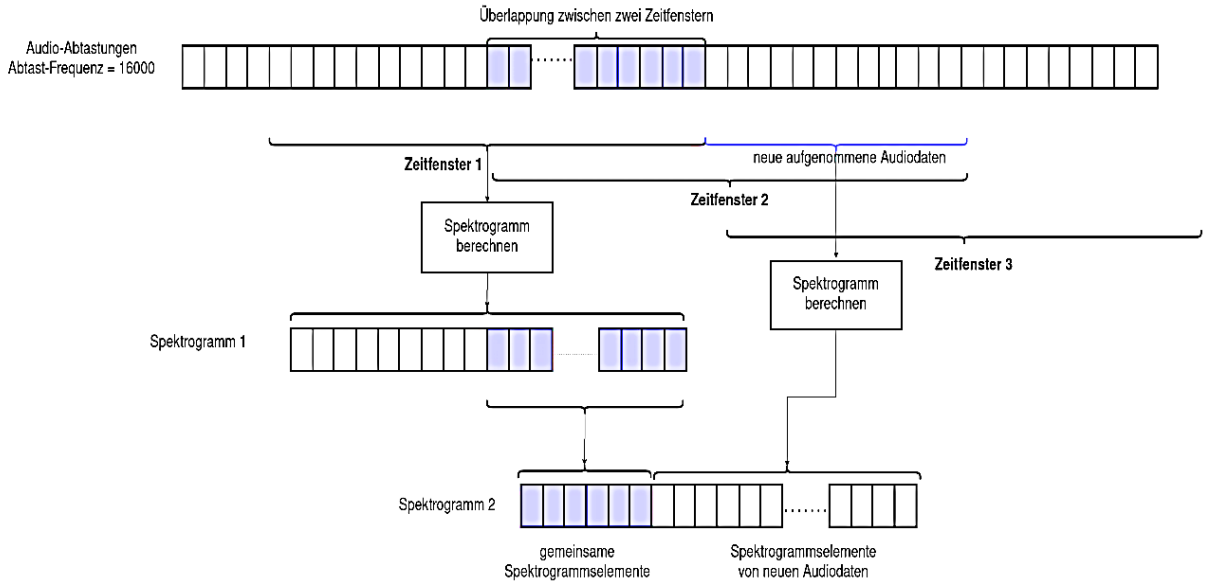


Abbildung 4-22: Berechnen der Spektrogramme

Die verschiedenen Boards, die im Kapitel 2 vorgestellt wurden, unterscheiden sich voneinander dadurch, welche Datenbussysteme zur Aufnahme von Audiodaten verwenden, wie sie mit der Außenwelt kommunizieren, und welche I/O-Möglichkeiten zur Verfügung stehen. Daher gibt es einige Bibliotheken für dieses oder jenes Board, die in dem Programmcode inkludiert werden müssen.

Es muss hier auch erwähnt werden, dass die Implementierung von Tensorflow-Operationen, die Definition des Modells innerhalb des Programms, und die Ausführung des Modells bleiben für die verschiedenen Boards unverändert.

In diesem Kapitel wird dieses Modell auf dem ESP-EYE-Board implementiert, weil es eine gute Leistung aufgewiesen hat und über ein Mikrophon verfügt. Außerdem werden die Eigenschaften und die Kommunikationsmöglichkeiten bei diesem Board benutzt, um die vollständige Arbeit zur Steuerung eins Geräts Kapitel 7) zu realisieren. Nachfolgend wird auf die Hauptteile des Programms auf dem Mikrocontroller eingegangen, wobei die Funktionen und Bibliotheken vorgestellt werden.

Die Main Funktion befindet sich in der Datei (main.cc) und hat den Namen tensorflow_main_funktion, wie in Listing 4-7 zu sehen ist.

Listing 4-7: Die Main Function

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <sys/time.h>
5. #include "esp_system.h"
6. #include "freertos/FreeRTOS.h"
7. #include "freertos/task.h"
8. #include "Funktionen.h"
9.
10. int tensorflow_main_funktion(int argc, char* argv[]) {

11.    setup();
12.    while (true) {
13.      loop();
14.    }
15. }
16.
17. extern "C" void app_main(void ) {
18.
19.    xTaskCreate((TaskFunction_t)&tensorflow_main_funktion,
   "tensorflow", 32 * 1024, NULL, 8, NULL);
20.    vTaskDelete(NULL);
21.
22.    }
```

Die Programmierung des ESP-Boards mit ESP-IDF ist ein bisschen schwieriger, da der Code eine geringere Abstraktionsebene hat im Vergleich mit Arduino-IDE zum Beispiel. Zudem muss man Kenntnisse über die Verwendung von Tasks und FreeRTOS-Bibliothek haben, wobei Diese Bibliothek ein Echtzeitbetriebssystem für eingebettete Systeme ist.

In der tensorflow_main_funktion werden zwei Funktionen Setup () und loop () aufgerufen. Die Funktion Setup () wird nur einmal ausgeführt, während die Funktion loop () innerhalb der Programmschleife liegt und wird unendlich ausgeführt. Dieselben Funktionen sind normallerweise in einem Arduino-Programm zu sehen.

Was hier wichtig ist, dass die tensorflow_main_funktion als Task mit einer bestimmten Priorität auszuführen ist. Die Definition der zwei Funktionen sind in der Datei namens Funktionen.cc zu finden. In dieser Datei werden zuerst die Bibliotheken

inkludiert, die für die Modellausführung nötig sind. Die Folgende Listing 4-8 zeigt diese Bibliotheken.

Listing 4-8: Die verwendeten Bibliotheken

```
1. #include "Funktionen.h"
2. #include "model.h"
3. #include "tensorflow/lite/micro/kernels/micro_ops.h"
4. #include "tensorflow/lite/micro/micro_error_reporter.h"
5. #include "tensorflow/lite/micro/micro_interpreter.h"
6. #include "tensorflow/lite/schema/schema_generated.h"
7. #include "tensorflow/lite/version.h"
8. #include "tensorflow/lite/micro/all_ops_resolver.h"
9. #include "audio_verarbeitung.h"
```

- Die Datei Funktionen.h enthält die Deklaration der bereits erwähnten Funktionen setup() und loop().
- In der Datei model.h wird das Modell als ein C-Array deklariert. Dieses C-Array wird durch den Terminal-Befehl 'xxd' generiert und enthält alle Informationen über den Modellgraph und die Modellgewichtungen.
- Die Datei all_operation_resolver.h deklariert eine Klasse, die ermöglicht, die für die Ausführung des Modells benötigten Operationen zu importieren.
- Micro_error_reporter.h ist eine Klasse zur Protokollierung und Ausgabe von Fehlernachrichten und hilft beim Debuggen.
- Micro_interpreter.h ist der Tensorflow für Mikrocontroller Interpreter, der das Modell ausführt.
- Shema_generated.h, Das Schema, das die Struktur des Modells in der Datei model.h definiert und es verständlich macht.
- Die Datei version.h enthält die Versionsnummer des Schemas, durch die überprüft werden kann, ob das Modell mit derselben Version definiert wurde. [26]
- Die Datei audio_verarbeitung.h deklariert einige Klassen und Funktionen zur Audioaufnahme und Erzeugung von Spektrogrammen.

Als nächstes wird auf die wesentlichen Codezeilen eingegangen, die zur Ausführung des Modells nötig sind. Die folgenden Codezeilen befinden sich in der Datei Funktionen.cc.

In der setup() Funktion(Listing 4-9) werden alle nötigen Klassen initialisiert, die in den oben genannten Bibliotheken und Dateien enthalten sind.

Listing 4-9:Die setup() Funktion

```
1. namespace {
2. tflite::ErrorReporter* error_reporter = nullptr;
3. const tflite::Model* model = nullptr;
4. tflite::MicroInterpreter* interpreter = nullptr;
5. TfLiteTensor* model_input = nullptr;
6. int32_t previous_time = 0;
7. constexpr int kTensorArenaSize = 80* 1024;
8. uint8_t tensor_arena[kTensorArenaSize];
9. int8_t feature_buffer[feature_elements_anzahl ];
10.   float * model_input_buffer = nullptr;
11.   uint8_t output_uint8[anzahl_der_klassen] ;
12.   }
13.   feature_class  feature_class_instanz ( feature_buffer  ) ;
14.   auswertung_der_results auswertung_der_results_instanz;
15.
16.   void setup() {
17.
18.   static tflite::MicroErrorReporter micro_error_reporter;
19.         error_reporter = & micro_error_reporter;
20.
21.   model = tflite::GetModel(g_model);
22.
23.     if (model->version() != TFLITE_SCHEMA_VERSION) {
24.        TF_LITE_REPORT_ERROR  (error_reporter,
25.        "die Schema-version  %d "
26.        " und die unterstuetzte Version %d
27.         sind nicht gleich",
28.         model->version(),
29.         TFLITE_SCHEMA_VERSION);
30.          return;}
```

```
31.
32.    tflite::AllOpsResolver resolver;
33.    static tflite::MicroInterpreter static_interpreter(
34.                          model, resolver, tensor_arena,
35.                          kTensorArenaSize, error_reporter);
36.                          interpreter = &static_interpreter;
37.   TfLiteStatus allocate_status =
38.                         interpreter->AllocateTensors();
39.                         if (allocate_status != kTfLiteOk) {
40.                         TF_LITE_REPORT_ERROR(error_reporter,
41.                        "AllocateTensors() fehlgeschlagen");
42.                          return; }
43.   model_input = interpreter->input(0);
44.    if ((model_input->dims->size != 2) ||
45.                  (model_input->dims->data[0] != 1) ||
46.                  (model_input->dims->data[1] !=
47.                  (slices_in_einer_sekunde *
48.                   spektrogram_elements_per_slice)) ||
49.                  (model_input->type != kTfLiteFloat32)) {
50.                   TF_LITE_REPORT_ERROR(error_reporter,
51.                  "Bad input tensor parameters in model");
52.                                                 return;}
53.   model_input_buffer = model_input->data.f;
54.    }
```

In den zwei Zeilen 13,14 werden zwei Klassen instanziiert, die zur Extraktion von Spektrogrammen notwendig sind. Die Definition von diesen Klassen befindet sich in der Datei audio_verarbeitung.cc, wobei feature_buffer ein Zeiger ist, in dem das Spektrogramm gespeichert wird.

```
13.feature_class  feature_class_instanz ( feature_buffer  ) ;
14.auswertung_der_results auswertung_der_results_instanz;
```

Die Klasse MicroErrorReporter wird in der setup() Funktion instanziiert. Sie wird verwendet, um Fehlernachrichten auszugeben. Die Tensorflow-Klassen und–Funktionen verwenden ebenfalls der Error-Reporter, um auf Fehler, die auftreten könnten, aufmerksam zu machen. Wie in den folgenden Zeilen zu sehen ist, wird ein Zeiger (error_reporter) definiert, der auf die Klasse MicroErrorReporter verweist.

```
18.static tflite::MicroErrorReporter micro_error_reporter;
19.       error_reporter = & micro_error_reporter;
```

Anschließend wird in den nachfolgenden Zeilen das in model.h definierte Modell-Array an die Methode GetMode() übergeben. Diese Methode gibt einen Zeiger zurück, der auf das Modell verweist, dann wird überprüft, ob die Bibliothekversion, die verwendet wird, mit der Modellversion übereinstimmt. Falls die Versionsnummer nicht stimmt, wird der Error-Reporter eine Fehlermeldung ausgeben.

```
1. model = tflite::GetModel(g_model);
2.
3.   if (model->version() != TFLITE_SCHEMA_VERSION) {
4.                TF_LITE_REPORT_ERROR(error_reporter,
5.                            "die Schema-version  %d "
6.                            " und die unterstuetzte Version %d
7.                              sind nicht gleich",
8.                               model->version(),
9.                              TFLITE_SCHEMA_VERSION);
10.                               return;}
11.
12.   tflite::AllOpsResolver resolver;
13.    static tflite::MicroInterpreter static_interpreter(
14.                            model, resolver, tensor_arena,
15.                          kTensorArenaSize, error_reporter);
16.                             interpreter = &static_interpreter;
```

AllOpsResolver ist eine Klasse, die dem Micro-Interpreter ermöglicht zu wissen, welche Operationen im Modell verwendet werden. Diese Klasse kann alle Operationen und Strukturen erkennen, die in einem Modell verwendet sind. In der zweiten Zeile wird der Microinterpreter instanziiert, der das Modell ausführen wird. Wie zu sehen ist, nimmt der Interpreter die folgenden Parameter:

- model: Zeiger auf das Modell
- resolver: Instanz von all_ops_resolver.
- tensor_arena: ist ein Array, das als Container für das Modell bezeichnet werden kann. Die folgenden Zeilen zeigen die Definition dieses Arrays.

```
7. constexpr int kTensorArenaSize = 80* 1024;
8. uint8_t tensor_arena[kTensorArenaSize];
```

Dieses Array wird alle Tensoren des Modellgraphs enthalten, wobei ihr Größe von dem Modell abhängt. In diesem Fall wird 80 KByte Speicher für das Modell reserviert.

```
37. TfLiteStatus allocate_status =
38.                     interpreter->AllocateTensors();
39.                     if (allocate_status != kTfLiteOk) {
40.                     TF_LITE_REPORT_ERROR(error_reporter,
41.                   "AllocateTensors() fehlgeschlagen");
42.                      return; }
```

Die Methode AllocateTensor() läuft das Modell durch, und weist jedem Tensor des Modells den benötigten Speicherplatz aus der tensor_arena zu, wobei die Tensoren eine spezielle Art der Datendarstellung in Tensorflow wie Arrays sind (daher kommt der Name Tensor Flow). Jetzt wird der Eingang-Tensor des Modells definiert, wobei der Datentype des Tensors in diesem Fall Float32 ist.[14]

```
55.  mo model_input = interpreter->input(0);
```

Die bisherigen Anweisungen befinden sich in der setup() Funktion und werden nur für die Initialisierung des Modells aufgerufen. Nachfolgend wird auf die Funktion Loop() eingegangen. Diese Funktion wird in einer unendlichen Schleife aufgerufen. Die folgenden Codezeilen beziehen sich auf Tensorflow-API. Für mehr Einzelheiten, wie das Audiosignal aufgenommen und wie das Spektrogramm extrahiert wird, kann der Quellcode auf dem Datenträger eingesehen werden. Der Listing 4-10 zeigt diese Funktion.

Listing 4-10: Die Funktion loop()

```
1. void loop() {
2. const int32_t current_time= LatestAudioTimestamp();
3.  int how_many_new_slices = 0;
4.  TfLiteStatus  state_ =
5.             feature_class_instanz.feature_extraktion
6.             (  error_reporter ,current_time , previous_time,
7.              & how_many_new_slices   ) ;
```

[14] Der Datentype des Tensors kann Float32, uint8 oder int8 sein, je nachdem welchen Datentyp das quantisierte Modell als Eingabe akzeptiert.

```
8.  if ( state_ != kTfLiteOk ) { TF_LITE_REPORT_ERROR
9.          ( error_reporter ,"es gibt einen  fehler in der
10.   funktion feature-extraktion  " ) ;
11.                   return ; }
12.
13. previous_time = current_time;
14. if (how_many_new_slices == 0) { return; }
15.
16.
17. for (int i = 0; i < feature_elements_anzahl ; i++) {
18.     model_input_buffer[i] =
19.    ((float)feature_buffer[i]  +128)*1/256 ;}
20.
21.
22. TfLiteStatus invoke_status = interpreter->Invoke();
23.                   if (invoke_status != kTfLiteOk) {
24.                   TF_LITE_REPORT_ERROR(error_reporter,
25.                   "Invoke failed");
26.                    return;}
27.
28. TfLiteTensor* output =interpreter->output(0);
29.
30. for ( int i = 0 ; i<anzahl_der_klassen ; i++ ) {
31.   output_uint8[i] = (uint8_t ) 255 *output->data.f[i] ;  }
32.
33. auswertung_der_results_instanz.
34.                            result_verschieben(output_uin8)  ;
35.
36. auswertung_der_results_instanz.result_auswerten() ;
37.
38.   if ( auswertung_der_results_instanz.ist_result_gefunden =true ) {
39.     TF_LITE_REPORT_ERROR ( error_reporter , "score = %d " ,
40.        auswertung_der_results_instanz.score ) ;
41.
42.   TF_LITE_REPORT_ERROR ( error_reporter ,
43. KLASSEN_LABELS_ARRAY[auswertung_der_results_instanz.
44.                      vorhersage_result ] ) ;
45.
46. if ( auswertung_der_results_instanz.vorhersage_result !=&&
47.        auswertung_der_results_instanz.vorhersage_result  != 1 ) {
48.             auswertung_der_results_instanz.behaelter_zero() ;}
49.           auswertung_der_results_instanz.ist_result_gefunden= false
    ; }
50.  }
```

In einer for-Schleife wird der Eingang-Tensor des Modells mit dem Spektrogramm gefüllt (Zeilen 17, 18 und 19), dann wird In den darauffolgenden Zeilen(22) das Modell ausgeführt, um die Ausgabe(Wahrscheinlichkeiten) zu berechnen. Gleichzeitig wird überprüft, ob die Ausführung ohne Fehler stattgefunden hat(Zeilen 23 bis 26), wobei

TfLiteStatus eine Tensorflow-Datentype ist, der zwei Werte annehmen kann, entweder kTfLiteOk oder kTfLiteError.

Der Ausgang des Modells ist ebenso ein Tensorflow-Tensor(TfLiteTensor), der definiert werden muss (Zeile 28). In diesem Fall erzeugt das Modell Wahrscheinlichkeiten, die als Float32 Zahlen dargestellt sind. Diese Wahrscheinlichkeiten werden in Zeilen 30 und 31 in Uint8-Zahlen (0 bis 255) umgewandelt und in einem Array für eine Weiterverwendung gespeichert. In dieser Anwendung werden die vorhergesagten Ergebnisse per serielle Port an den Computer gesendet, damit sie auf dem Bildschirm angezeigt werden können. Neben der Vorhersage wird der Score angezeigt, der die Höhe der Wahrscheinlichkeit darstellt. In Abbildung 4-23 sind die Programmdateien dargestellt.

Hier muss erwähnt werden, dass die Extraktion des Spektrogramms mithilfe einer Tensorflow-Funktion berechnet wird, die für Micro-Speech-Beispiel eingesetzt wurde und in der Datei micro_feature_generator.cc definiert ist. Hierbei muss zudem beachtet werden, dass Google einen speziellen Algorithmus zur Extraktion von Spektrogrammen verwendet, der nicht veröffentlicht wurde. [16]

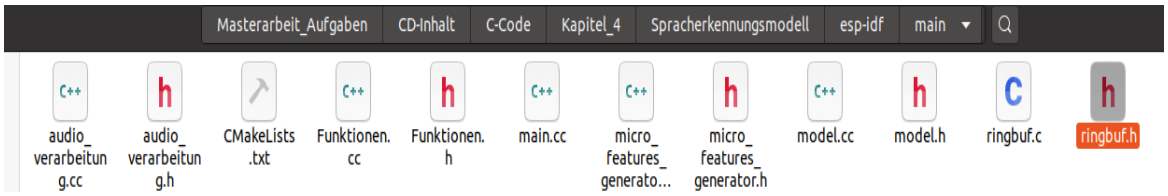


Abbildung 4-23: Die Programmdateien

4.2 Erkennung handschriftlicher Zahlen

In den folgenden Abschnitten wird ein Objekterkennungsmodell erstellt, das die handgeschriebenen Zahlen (0 bis 9) erkennt und auf einem Mikrocontroller ausgeführt werden kann. Dabei wird nach einer passenden Architektur gesucht , die Anschließend trainiert wird und sich auf den Mikrocontroller übertragen lässt. Als Trainingsdatensatz wird dafür Mnist-Datensatz verwendet, allerdings mit der Anwendung von verschiedenen Augmentation-Methoden.

Diese Anwendung soll einen Überblick darüber geben, wie groß ein Objekterkennungsmodell sein darf und wie schnell es ausgeführt werden kann, da solche Modelle normalerweise viel größer als das Spracherkennungsmodell sein könnten. Hier muss darauf hingewiesen werden, das die Größe des Modells sowie die Größe des benötigten Datensatzes stark von der Komplexität und der Ähnlichkeit der Daten abhängt. Das Mikrocontrollerboard soll in diesem Fall über eine Kamera verfügen, mit der Fotos aufgenommen werden. Es gibt dafür mehrere Möglichkeiten wie das ESP-Cam beispielsweise, das für diese Anwendung eingesetzt wird.

4.2.1 **MNIST-Datensatz**

Dieser Datensatz gehört zu den meistuntersuchten Datensätzen und wurde für viele Studien und Anwendungen in Anwendungsgebieten wie Bildverarbeitung und Deep-Learning eingesetzt. Die im Datensatz enthaltenen Daten stammen von 250 verschieden Studenten und Mitarbeitern und wurde durch die amerikanische international Institute of Standards and Technology erstellt. [11]

Der Mnist-Datensatz ist ein öffentlich verfügbarer Datensatz von handschriftlichen Ziffern und enthält 60000 Datenpunkte für das Training sowie 10000 Datenpunkte für die Validation. Die Bilder sind dabei Graustufenbilder mit der Auflösung 28x28, wie in Abbildung 4-24 ersichtlich ist.

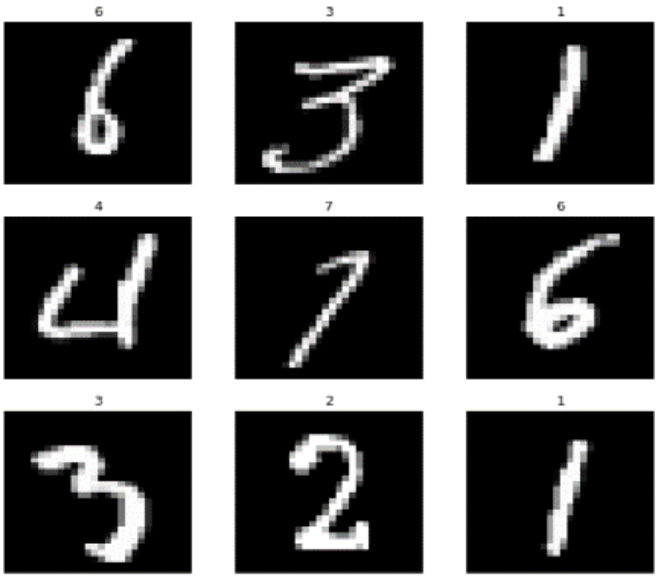


Abbildung 4-24: Mnist-Datensatz

4.2.2 **Augmentation von Bildern**

Ausgehend von dem Spracherkennungsmodell stellte es sich heraus, dass Augmentation von Daten ein wichtiger Schritt zur Verbesserung der Allgemeinbarkeit und die Accuracy-Rate eines Modells ist. Dies ist besonders wichtig, wenn sich die Daten, die während der Inferenz dem Netz zugeführt werden, von den Daten, die beim Trainieren verwendet worden sind, unterscheiden, oder wenn die Daten künstlich erzeugt wurden. Zum Beispiel hat Augmentation im Spracherkennungsmodell die synthetischen Daten, die keine Geräusche enthalten und einen bestimmten Ton oder wenige Tone haben, natürlicher gemacht. In Computervision-Anwendungen wird dasselbe Prinzip benutzt und die Augmentation eingesetzt, um bessere Ergebnisse zu erzielen.

Wie erwähnt, sind die im Datensatz enthaltenen Bilder Graustufenbilder mit der Auflösung 28x28. Zugleich müssen die in das Netz eingespeisten Bilder während der Inferenz dieselben Eigenschaften haben, die Trainingsbilder aufweisen. Dies führt dazu, das die mit der Kamera auf dem Mikrocontrollerboard aufgenommenen Bilder ebenfalls Graustufenbilder sein müssen. Was die Auflösung der Bilder angeht wird nachfolgend behandelt.

Auf dem Mikrocontrollerboard (ESP-Cam) können die Eigenschaften der Bilder je nach Anwendung eingestellt werden. Die Tabelle 4-4 zeigt die möglichen Bildauflösungen.

Tabelle 4-4: Auflösungsmöglichkeiten der Kamera auf dem ESP-CAM.

Frame-Size	Auflösung
FRAMESIZE_96X96	96x96
FRAMESIZE_QQVGA	160x120
FRAMESIZE_QQVGA2	128x160
FRAMESIZE_QCIF	176x144
FRAMESIZE_HQVGA	240x176
FRAMESIZE_QVGA	320x240
FRAMESIZE_CIF,	400x296
FRAMESIZE_VGA	640x480
FRAMESIZE_SVGA	800x600
FRAMESIZE_XGA	1024x768
FRAMESIZE_SXGA	1280x1024
FRAMESIZE_UXGA	1600x1200

Aus Erfahrungen wurde festgestellt, dass die Größe des Modells stark von der Größe der Eingabe abhängt. Gleichzeitig muss das Bild für gute Erkennung im Allgemeinen

eine akzeptable Auflösung haben. Das hängt von dem Problem ab, das gelöst werden muss. In diesem Fall ist die minimale Auflösung 96x96 mehr als genug, da nur auf einem Papier geschriebene Zahlen erkannt werden müssen und die Bilder keine große Komplexität oder viele Details aufweisen.

Es gibt hier zwei Möglichkeiten, entweder werden die Verfügbaren 28x28-Trainingsbilder vergrößert, um schließlich 96x96 Trainingsbilder zu erzeugen (das wird als Interpolation von Bildern bezeichnet), oder werden die Aufgenommenen 96x96 Bilder verkleinert, sodass die 96x96 Bilder in 28x28 Bilder umgewandelt werden(das ist als Dezimation von Bildern bekannt). In dieser Anwendung wird das Netz auf Bilder mit der Größe 96x96 verwendet, was bedeutet, dass einige Vorverarbeitungs-methoden auf den Datensatz anzuwenden sind.

Nachdem die Bildergröße angepasst wird, müssen Augmentation-Methoden auf die Trainingsbilder angewendet werden. Hierbei ist zu merken, dass die Mnist-Daten Graustufenbilder sind, allerdings mit sehr schmalem Histogramm. Das bedeutet, dass die Graustufenbilder nur im schmalen Wertebereich variieren (fast Binärbilder). Außerdem sind die Zahlen nur vor schwarzem Hintergrund ohne Rauschen dargestellt. Diese Eigenschaften der Trainingsbilder ähneln den Bildern nicht, die durch die Kamera aufgenommen werden. Daher werden einerseits einige Augmentation-Methoden auf die Trainingsbilder angewendet, andererseits werden für die mit der Kamera aufgenommenen Bilder Vorverarbeitungen eingesetzt. Die Augmentation-Methoden sind:

- **Zufällige Änderung der Bilderkontrast.** Mit Kontrast wird der Unterschied zwischen den hellen und dunklen Bereichen des Bildes gemeint. Die Mnist-Bilder haben eigentlich einen hohen Kontrast. Daher muss der Kontrast zufällig verringert.
- **Zufällige Änderung der Helligkeit.** Hierbei wird die Helligkeit der Bilder mit zufälligen Werten geändert.
- **Rauschen hinzufügen.** Dieser Schritt ist wichtig, um die Bilder natürlicher zu machen, indem die Bilder verrauscht werden.
- **Bildverschiebung.** Dabei werden die Bilder auch zufällig verschoben, sodass die Zahlen zufällig innerhalb des Bildrahmen platziert werden.

- **Die Größe der Zahlen.** Da die Größe der Zahlen auf den Bildern fast konstant ist, löhnt es sich, die Größe der Zahlen innerhalb des Bildrahmen zu ändern.
- **Normalisierung der Bilder.** Diese Augmentation-Methode wurde bereits beim Trainieren des Spracherkennungsmodells besprochen. Bei Bildern gilt dasselbe Prinzip.
- **Skalierung von Bildern.** Bei den originalen Mnist-Bildern ändern sich die Pixelwerte zwischen 0 und 255 (uint8). Beim Training empfehlt es sich, die Pixelwerte der Bilder so zu skalieren, dass sie zwischen 0 und 1 liegen. Dies trägt zur Verbesserung der Accuracy-Rate bei.

Die folgende Abbildung 4-25 zeigt die Bilder mit der Anwendung von den erwähnten Augmentation-Methoden.

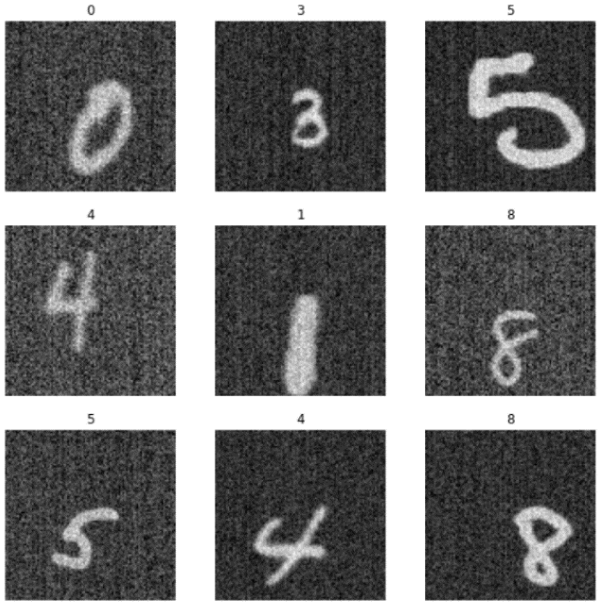


Abbildung 4-25: 96x96 Bilder mit Anwendung von Data Augmentation.

4.2.3 **Der Modellentwurf**

Nachdem die Trainingsdaten verarbeitet worden sind, muss ein Modell gefunden werden, die für die Erkennung handgeschriebener Zahlen geeignet ist. Dieser Schritt ist nicht immer problemlos. Denn es gibt keine Regeln, wie ein Netz entworfen werden kann. Auf der Suche nach geeignetem Modell kann man Experimente durchführen und verschiedene Architekturen probieren, bis man ein geeignetes Modell findet. Einige Schritte, die die Suche erleichtern könnten, werden nachfolgend aufgelistet:

- Es muss überlegt werden, welche Struktur besser für das zu lösende Problem geeignet ist. Für die Erkennung von Objekten empfiehlt es sich, die Faltungsnetze zu verwenden. Denn diese Netze schneiden bei solchen Problemen besser als die FC-Netze oder rekurrente Netze ab.
- Um die Überanpassung zu vermeiden, kann die Regularisierung-Technik auf bestimmte Weise eingesetzt werden. Die beste Art der Regularisierung ist hierbei eine Droup-out-Schicht.
- Die Anzahl der Schichten können geändert werden, bis das beste Ergebnis erreicht wird.
- Der Einfluss der Schrittweite (bei CNN- und Pooling-Schichten) auf die Accuracy-Rate kann durch Experimentieren beobachtet werden, bis ein passender Wert für diesen Parameter gefunden wird.
- Die Pooling-Schichten können zum Einsatz kommen, um die Größe des Modells sowie den Rechenaufwand zu verringern. Diese Schichten haben auch einen positiven Einfluss auf die Allgemeinbarkeit des Netzes und helfen dabei, die Überanpassung zu vermeiden. Diese Schicht wird normallerweise nach der Faltungsschicht platziert.
- Das Faltungsnetz besteht aus zwei Teilen. Das erste Teil enthält die Faltungsschichten und dient dem Erlernen von Merkmalen, Während das zweite Teil aus FC-Schichten besteht und der Klassifizierung dient. Die Anzahl der FC-Schichten können eine oder zwei Schichten sein. Es ist hier aber zu beachten, dass FC-Schichten sehr rechenintensiv sind und eine große Zahl an solchen Schichten vermieden werden soll.
- Als Aktivierungsfunktion haben die RELU und ELU gute Ergebnisse aufgewiesen. In der letzten FC-Schicht wird empfohlen, Softmax als Aktivierungsfunktion zu verwenden.
- Beim Trainieren ist der Adam-Optimizer bevorzugt. Hierfür kann entweder eine adaptive Lernrate verwendet werden, oder der Lernprozess wird in mehrere Phasen aufgeteilt, wobei in jeder Phase eine Lernrate beginnend mit großen Werten ausgewählt wird.

Durch Experimentieren kann eine Aarchitektur gefunden werden, die die Anforderungen für die Implementierung auf dem Mikrocontroller erfüllt. Die Abbildung 4-26 zeigt die Modellstruktur.

Das Modell hat hauptsächlich 4 Faltungsschichten und zwei FC-Schichten. Es ist aber ein relativ großes Modell (Tabelle 4-5) im Vergleich mit dem Spracherkennungsmodell.

Tabelle 4-5: Die Modellgröße und die Accuracy-Rate

Das Modell	Modellgröße	Accuracy-Rate
Foat32-Modell	316672 Byte	97.42 %
Uint8/int8-Modell	88512 Byte	96.555556 %

Jetzt muss das quantisierte Modell auf das Mikrocontrollerboard übertragen und ausgeführt werden. Das verwendete Board ist in diesem Fall das ESP-Cam-Board (sehe ESP-Camera in Kapitel 2). Auch hier empfehlt es sich die Inferenzzeit des Modells auf dem Mikrocontroller zu messen. Die folgende Tabelle (Tabelle 4-6) zeigt die Inferenzzeit des quantisierten Modells während der Ausführung auf dem Mikrocontroller bei unterschiedlichen Taktfrequenzen.

Tabelle 4-6: zeigt die Inferenzzeit auf dem Mikrocontroller

Das Modell	Inferenzzeit auf dem ESP-Cam-Board	Taktfrequenz	Benötigte RAM
Int8/uint8-Modell	1700 Millisekunden	240 MHz	80 kByte
	2500 Millisekunden	160 MHz	80 kByte
	4900 Millisekunden	80 MHz	80 kByte

Es ist in Tabelle 4-5 ersichtlich, dass die Accuracy-Rate nach der Quantisierung ein wenig nachlässt, und das ist erwartet, da die Gewichtungen des Modells nach der Quantisierung mit geringerer Auflösung dargestellt werden. Trotzdem kann dieses Verfahren nicht vermieden werden, weil das Modell auf Mikrocontrollern ausführt werden muss.

Die Tabelle Tabelle 4-6 liefert Informationen darüber, wie schnell die Ausführung des Modells auf dem Mikrocontroller bei unterschiedlichen Taktfrequenzen sein wird. Dabei ist der Speicherbedarf des Modells von entscheidender Bedeutung. In diesem Modell ist die Ausführungszeit groß im Vergleich mit dem Spracherkennungsmodell. Das ist darauf zurückzuführen, dass dieses Modell größere Eingabedaten verarbeitet

und mehr Schichten mit größerer Anzahl der Filter in den Faltungsschichten hat, was eine große Anzahl von arithmetischen Operationen bedeutet. Abbildung 4-27 zeigt eine Visualisierung der Accuracy-Rate und Loss-Funktion während des Trainings unter Verwendung von Tensorboard, wobei die Trainingskurve in rot und die Validations-kurve in blau ist.

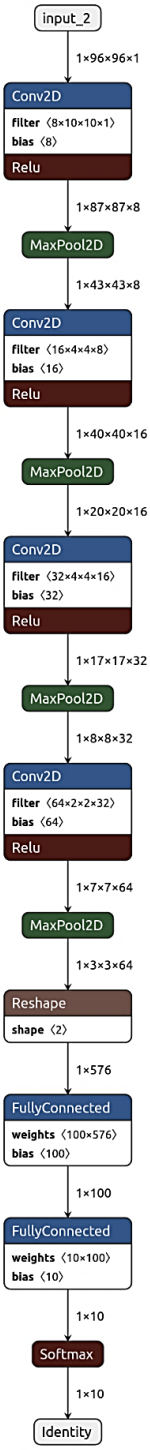


Abbildung 4-26: Das Erkennungsmodell

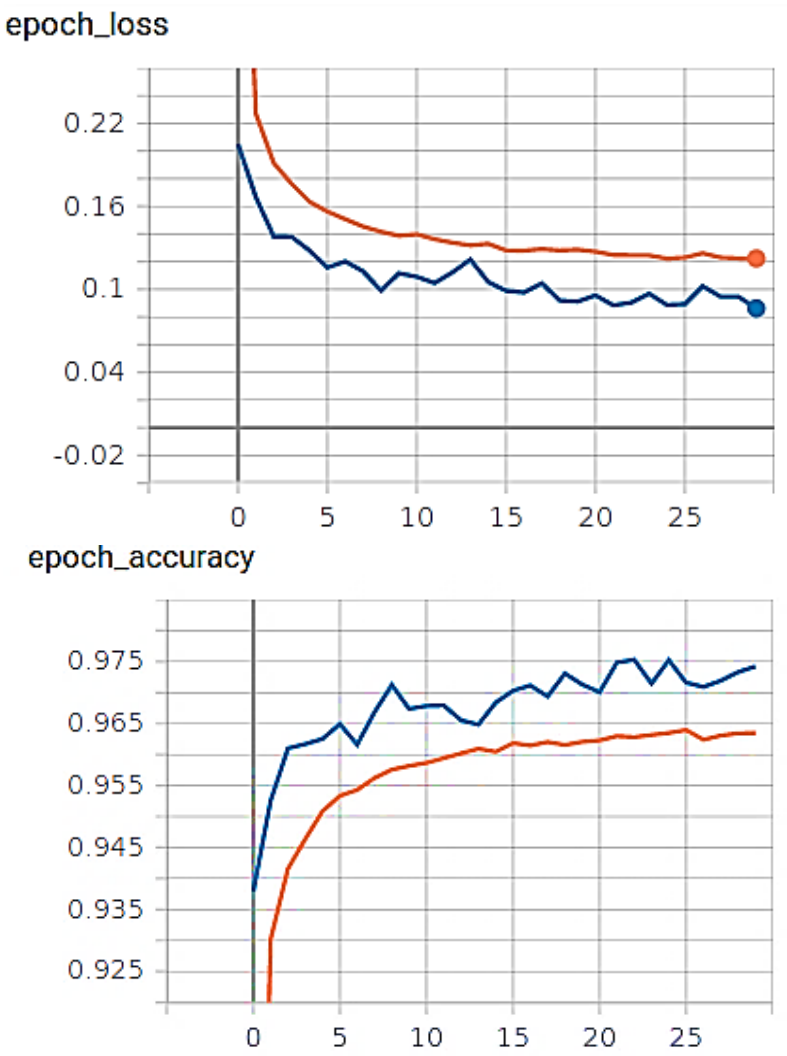


Abbildung 4-27: Visualisierung der Validation und Training mit Tensorboard

Die Abbildung 4-27 zeigt, dass es einen Unterschied zwischen den zwei Kurven gibt, wobei die Validation-Accuracy höher ist. Das kann darauf zurückgeführt werden, dass beim Training ein Anteil der Gewichtungen durch die Regularisierung abgeschaltet wird. Bei der Validation werden hingegen alle Gewichtungen an der Berechnung der Ausgabe beteiligen.

Um besser den Einfluss der Quantisierung auf die Accuracy-Rate zu analysieren und für bessere Bewertung der Klassifikation durch das Netz, empfehlt es sich die Konfusionsmatrix mit und ohne Anwendung der Quantisierung zu generieren (Abbildung 4-28).

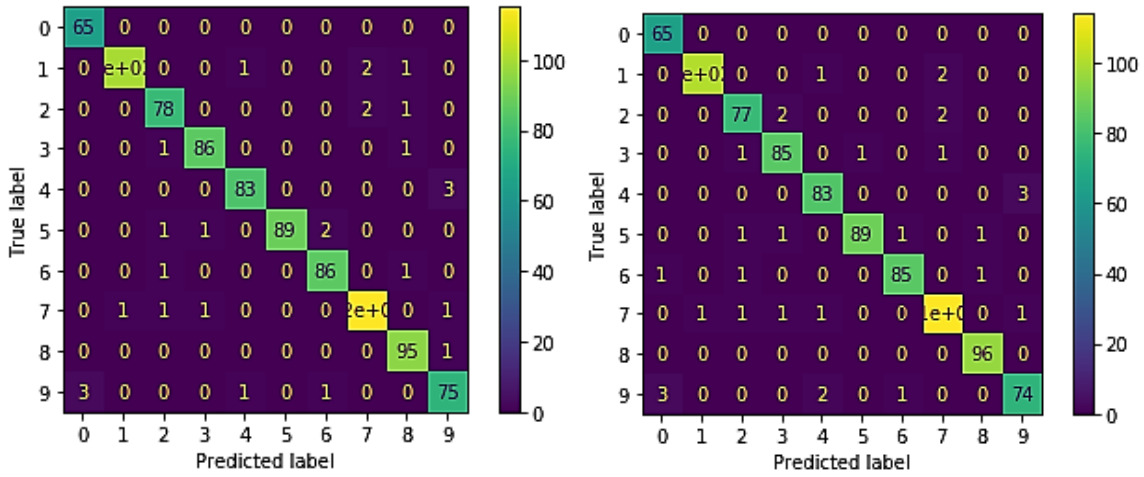


Abbildung 4-28: Die Konfusionsmatrix , Float32 (links) und Int8 (rechts)

Ausgehend von der Abbildung 4-28 kann erkannt werden, dass einige richtig klassifizierten Datenpunkte in andere Klassen nach der Quantisierung wandern, was den Erwartungen entsprecht. Was dabei bemerkenswert ist, dass die Quantisierung manchmal zur Verbesserung der richtig klassifizierten Datenpunkten führt, wie bei Klasse 8 der Fall ist. Dies ist darauf zurückzuführen, dass die Änderungen durch die Quantisierung zufälligerweise positive Verbesserung der Klassifizierung verursachen könnten. Später im Kapitel 7 wird die Accuracy-Rate nach der Quantisierung sogar erhöht. Das bedeutet aber nicht, dass das quantisierte Modell bessere Genauigkeit aufweist.

Die Abbildung 4-29 verdeutlicht die Funktionsweise der Faltungsschichten. Hierfür wurde eine in Python geschriebene Funktion eingesetzt, die den Ausgang der einzelnen Schichten visualisieren kann. Dadurch kann beispielsweise die Extraktion der Merkmale besser begriffen werden, wobei die Ausgabe der ersten Schicht 8 Kanäle hat, während die Zweite 16 Filter auf die Eingabe anwendet.

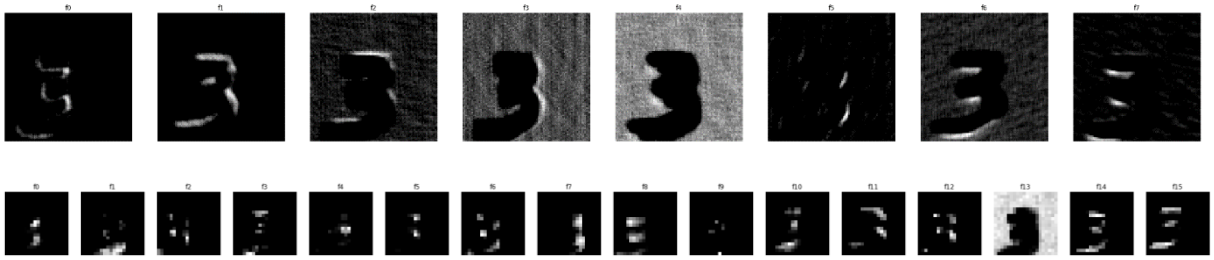


Abbildung 4-29: Visualisierung der Ausgabe von den Faltungsschichten.

4.2.4 **CPU, GPU und TPU zum Trainieren von ML-Modellen**

Das Training von Deep-Learning-Modellen erfordert normallerweise spezielle Hardware-Ressourcen und leistungsfähige Rechner, die viele arithmetischen Operationen pro Sekunde erledigen können. Diese arithmetischen Berechnungen werden Bei normaler Programmierung sequenziell ausgeführt. Hingegen werden die Operationen und Berechnungen der Deep-Learning-Modelle parallel ausgeführt, die meistens beim Trainieren Matrizen-Operationen sind. Die Abbildung 4-30 veranschaulicht, wie ein einziges Neuron trainiert wird.

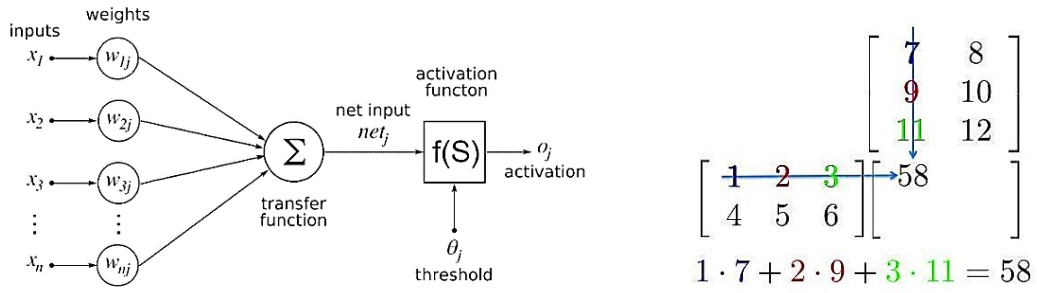


Abbildung 4-30: Berechnung der Ausgabe während des Trainierens [27]

Bei komplexen neuronalen Netzen werden manchmal beim Trainieren Millionen solcher Berechnungen pro Sekunde erledigt, was einen sehr großen Arbeitsspeicher und leistungsstarke Prozessoren erfordert.

Für lange Zeit wurden zum Trainieren neuronaler Netze CPUs (Central Processing Unit) eingesetzt, die im Allgemeinen für Abarbeitung sequenzieller Aufgaben konzipiert worden sind und keine parallelen Aufgaben erledigen können. Dies bedeutet, dass das Training nur langsam machbar ist. Trotz der Verbesserung der Parallelität bei modernen CPUs durch Verwendung mehrerer Kerne, bleibt das Training langsam und mühsam.

Zur Beschleunigung des Trainingsprozesses werden zurzeit GPUs (Graphics Processing Unit) häufig für das Training eingesetzt. Diese Recheneinheiten haben einen großen Vorteil im Vergleich mit CPUs. Denn sie wurden zur Berechnung von Tausenden von Pixelelementen gleichzeitig entwickelt und verfügen über Hunderten von einfachen Kernen oder Recheneinheiten, die parallel arbeiten. Aus diesen Gründen können die GPUs den Trainingsprozess rasant beschleunigen, besonders

wenn das Modell zur Verarbeitung von Videodaten oder Erkennung von Objekten trainiert wird. [27]

In diesem Modell zur Zahlenerkennung enthält der Datensatz 70000 Bilder mit der Auflösung 96x96. Dazu kommen die Augmentation-Daten, die so groß sind. Insgesamt sind die Trainingsdaten ungefähr 150000 Bilder. Das Trainieren des Modells kann auf einem normalen Rechner ohne Verwendung von GPU einen Tag dauern, auch wenn die Rechner leistungsfähig ist.[15] Die folgende Tabelle zeigt uns die benötigte Zeit für den Trainingsprozess auf einem GPU und einem CPU.

Tabelle 4-7: Die benötigte Trainingszeit auf einem GPU und CPU

Recheneinheit	Benötigte Trainingszeit für unser Modell in Minuten	Vorhandener Arbeitsspeicher
CPU	780	13 GByte
GPU (NVIDIA TESLA T-4)	140	13 GByte

TPU steht für Tensor Processing Unit. Es handelt sich dabei um einen Chip, der von Google zur Verwendung mit den optimierten Tensorflow-Bibliotheken und ML-Anwendungen entwickelt worden ist. Diese Recheneinheiten kommen bereits bei Google-Anwendungen wie Street-View zum Einsatz und werden für externe Kunden über Google-Cloud zur Verfügung gestellt.

TPUs beschleunigen deutlich das Training um den Faktor 15 im Vergleich mit CPUs und GPUs mit geringerem Energieverbrauch und werden hauptsächlich für die internen Arbeiten eingesetzt. [28]

Zum Trainieren des Modells wurde das Modell mithilfe von Google-Diensten trainiert, da es sehr schwierig war, solch eines Modells auf einem normalen Rechner zu trainieren. Google bietet hierbei gute kostenlose oder auch kostenpflichtige Dienste. Ein der Dienste ist Google-Colab (Colaboratory), ein Notizbuch, das kostenlose Verwendung von CPUs, GPUs und TPUs (wenn das Modell Kompatibel ist) in begrenzten Ausmaßen zur Verfügung stellt. Für kostenloses Trainieren eines Modells

[15] Der benötigte Arbeitsspeicher muss in diesem Fall 12 GByte sein.

werden beispielsweise 13 G Byte RAM und GPUs wie NVIDIA Tesla T-4 für maximale Laufzeit von 9 Stunden angeboten.

Für längere Laufzeiten, größere RAMs oder leistungsfähigere GPUs wird dieser Dienst kostenpflichtig. In Google-Colab können Python-Codes geschrieben und ausgeführt und viele vorinstallierte Packages und Bibliotheken wie Pandas und sklearn verwendet werden, was das Trainieren von ML-Modellen sehr erleichtert.

4.2.5 Die Implementierung auf dem Mikrocontroller

Nachdem das Modell trainiert und in Tensorflow-Lite konvertiert wurde, muss für die Implementierung auf dem Mikrocontroller ein C-Programm geschrieben werden. Das Verwendete Board kann dabei unterschiedlich sein, wobei die Tensorflow-Anweisungen zur Ausführung des Modells Plattformunabhängig sind. Was sich bei der Verwendung eins Boards unterscheidet, sind die Kamera und ihre Konfigurationen und vielleicht der Bussystem zwischen der Kamera und dem Mikrocontroller. Daher muss der Code für ein anderes Board angepasst werden.

Wie erwähnt, wird das Modell auf dem ESP-Cam-Board implementiert, da dieses Board über einen mächtigen Prozessor und eine Kamera verfügt. [16] Außerdem hat dieses Board GPIOs, die möglich machen, die Vorhersage auf einer kleinen 7-Segments-Anzeige anzuzeigen. Die Folgende Abbildung 4-31 zeigt die Implementierung, wobei die Vorhersage mit der vor der Kamera geschriebenen Zahl übereinstimmt.

Die Kamera wird dabei so konfiguriert, dass sie Graustufenbilder mit der Auflösung 96x96 und Pixelwerte als uint8 aufnimmt. Das aufgenommene Bild wird in einem Zeiger oder Buffer gespeichert, um rechtzeitig an das Modell geliefert zu werden. Was hier zu beachten ist, dass der Datentyp des aufgenommenen Bilds mit dem Eingang des Modells übereinstimmen muss. In diesem Fall akzeptiert das Modell die Int8-Werte als Eingabe, was bedeutet, dass die Bilder vom Datentyp uint8 in int8 umgewandelt werden müssen, bevor sie in das Modell eingespeist werden können.

[16] Das Board ESP-CAM hat keinen USB-Anschluss, was bedeutet, dass einen Programmierer wie beispielsweise ftd232 benötigt wird.

Die Integration des Modells in das C-Programm erfolgt wie beim Spracherkennungs-modell, wobei das C-Array generiert und im Programm deklariert werden muss.

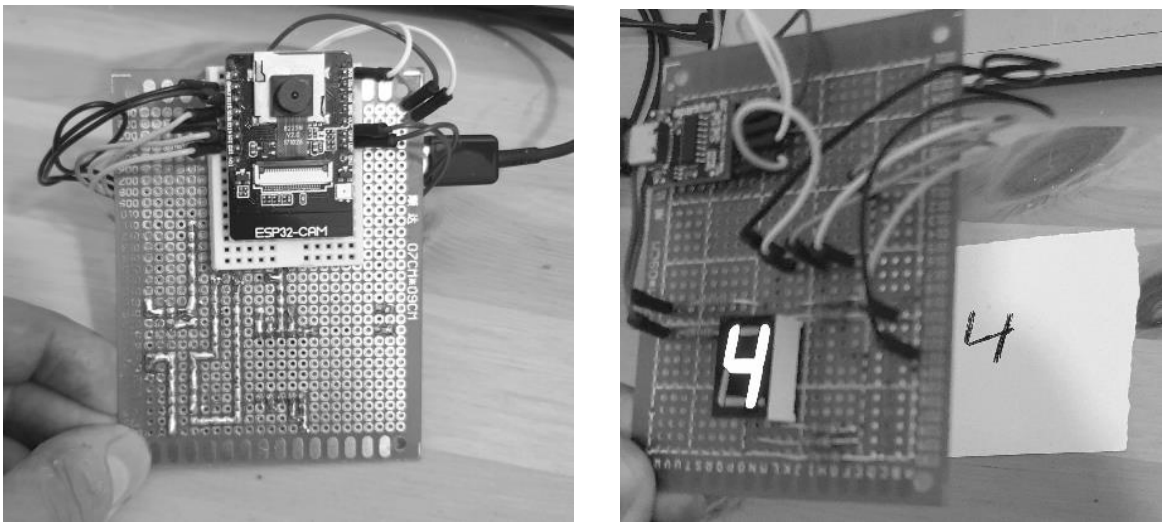


Abbildung 4-31: Die Implementierung auf dem ESP-Camera

Kapitel 5 - Long Short Term Memory (LSTM)

Als erstes wurde Im Kapitel 4 ein Spracherkennungsmodell, das Zwei Wörter erkennt, unter Verwendung von Faltungsnetzen erstellt und trainiert. In diesem Kapital werden weitere wichtige ML-Strukturen untersucht, und nach einem möglichen Einsatz der LSTM-Kategorie auf dem Mikrocontroller gesucht. Anschließend wird zwischen dem CNN-Modell und LSTM-Modell verglichen und einige Messungen durchgeführt. Diese mächtige Deep-Learning-Kategorie eignet sich besonders gut für sequenzielle Daten oder Sequenzen, die zeitlich verbunden sind. Beispiele für sequentielle Daten sind die Verarbeitung natürlicher Sprache (Texte), Videodaten, Audiodaten und die Sequenzen von Genen. [12] Eine Interessante Anwendung dieser Strukturen ist das Verstehen natürlicher Sprache NLU(beispielsweise die automatische Beantwortung von Fragen).

5.1 Einführung in rekurrente neuronale Netze

Die sequenziellen Daten sind eine Kette von Elementen oder Informationen, die sich mit der Zeit entwickeln. Jedes Element dieser Kette bringt ein Teil der Informationen ein und entwickelt somit den Zustand des RNN-Netzes. [12] Das ist der Grundidee von RNN-Netzen. Die Abbildung 5-1 veranschaulicht die Funktionsweise dieser Struktur, wobei das Netz eine Art Schleife verwendet.

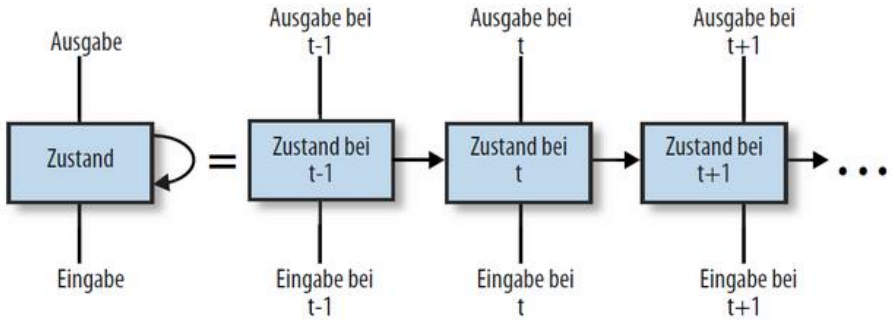


Abbildung 5-1: RNN-Netze [12]

In einem Zeitpunkt t übernimmt das Netz die Eingabe x(t) und aktualisiert den Zustandsvektor. Jetzt wenn die folgende Eingabe x(t+1) zu dem Zeitpunkt t+1 ankommt, wird die Ausgabe h nicht nur von x(t+1) abhängen, sondern auch von den vorherigen Eingaben (x(t)). In anderen Worten hat das Netz ein Gedächtnis, in dem

die vorherigen Informationen gespeichert werden, um die nachfolgende Eingabe basierend darauf vorherzusagen.

Das Problem bei den RNNs besteht darin, dass ihr Gedächtnis kurz ist. Wenn die Lücke zwischen den Informationen, und dem Zeitpunkt, an dem diese Informationen benötigt werden, sehr groß ist, kann sich das Netz an diese Informationen nicht mehr erinnern und die nächste Eingabe infolgedessen nicht richtig vorhersagen. Ein typisches RNN-Netz hat grundsätzlich nur eine Schicht zum Beispiel eine Tanh-Schicht. In Abbildung 5-2 ist die innere Struktur des RNN-Netzes dargestellt.

Bei kleineren Kontexten, wenn nur die letzte Eingabe benötigt wird, kann diese Netze die Aufgaben problemlos lösen. Das Problem der langfristigen Beziehungen wird durch eine spezielle RNN-Struktur namens LSTM gelöst. Diese Struktur kann die Aufgaben lösen, wo das Netz langfristige Abhängigkeiten erlernen muss.

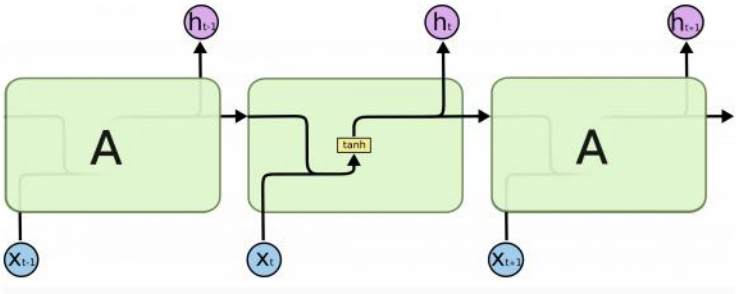


Abbildung 5-2: RNN-Netze mit Tanh-Schicht [29]

5.2 LSTM-Netze

Diese Netze haben langfristiges Gedächtnis, und können somit die Informationen für größere Zeitperioden speichern. Die Abbildung 5-3 verdeutlicht die Funktionsweise dieses Netzes.

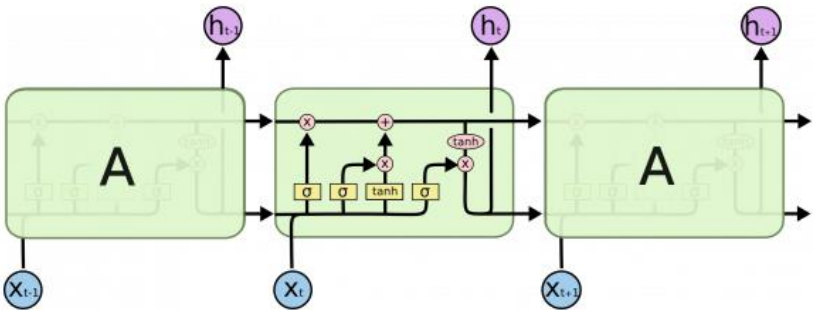


Abbildung 5-3: Das LSTM-Netz [29]

Die obere Linie, die durch eine LSTM-Zelle durchgeht, ist der Zellzustand. Dieser ist ein Vektor, der die vergangenen benötigten Eingaben speichert und an die nächste Zelle liefert. Der Zellzustand unterliegt in einer Zelle kleine Änderungen, wobei von ihm unnötige Informationen entfernt und ihm andere hinzugefügt werden. Das geschieht sorgfältig durch vier Schichten oder Gates (gelbe Kästchen), wie es in Abbildung 5-3 erkennbar ist. Dabei stellt jede schwarze Linie einen Vektor dar, während die kleinen Kreisen punktweise Operationen wie das Addieren und Multiplizieren von Vektoren sind. Es gibt außerdem Verzweigungen und Vereinigungen von Vektoren, wobei eine Verzweigung die Vervielfältigung von einem Vektor darstellt. Ein Gate ist ein Tor, der die Informationen nach Bedürfnissen entweder durchlässt oder sperrt. Die Bestandteile dieser Tore sind eine Sigmoid-Schicht eines neuronalen Netzwerkes und die punktweise Multiplikationsoperation.

Wie im Kapitel 2 erklärt wurde, kann die Ausgabe der Sigmoid-Funktion einen Wert zwischen null und eins annehmen. Dieser Wert bestimmt wie viel Prozentanteil der Informationen durchgelassen werden müssen. Der Wert Eins bedeutet ein vollständiges Durchlassen der Informationen, während eine Null das Verwerfen von Informationen bedeutet. Herbei werden zwischen drei Arten dieser Schichten unterschieden:

- Die Verlustschicht oder Vergessen-Schicht (Forget-Schicht). Hier wird eine Entscheidung getroffen, welche Informationen von dem Zellzustand entfernt und welche Informationen durchgelassen werden müssen. Dies wird durch die erste punktweise Multiplikation mit der Ausgabe der Segmoidschicht erreicht. Wenn die Ausgabe der Sigmoidschicht Eins ist, werden die Informationen vollständig gespeichert, während null bedeutet, dass die Informationen vollständig aus dem Zellzustand ausgeworfen werden müssen. Die Abbildung 5-4 veranschaulicht die Funktionsweise und die mathematische Beschreibung dieser Schicht wird in der folgenden Gleichung angegeben:

$$f_t = \sigma\left(w_f \cdot [h_{t-1}, x_t] + b_f\right) \qquad (5\text{-}1)$$

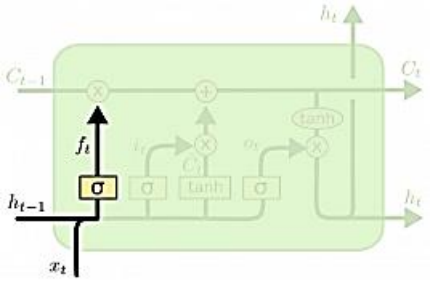


Abbildung 5-4:Die Verlustschicht des LSTM-Netzes [29]

- Die Speicherschicht. In dieser Schicht wird entschieden, welche neuen Informationen dem Zellzustand hinzugefügt werden müssen. Dieser Vorgang wird in zwei Schritten erledigt. Im ersten Schritt entscheidet eine Sigmoidschicht, welche Werte aktualisiert werden müssen, während Im zweiten Schritt eine Tahn-Schicht einen Vektor neuer Werte erstellt, die dem Zellzustand hinzugefügt werden. Die Ausgänge der Tahn-Schicht und Sigmoidschicht werden dann multipliziert und für die Aktualisierung des Zellzustands verwendet. Die folgenden Gleichungen bezeichnen das mathematische Prinzip dieser Schicht und Abbildung 5-5 veranschaulicht die Funktionsweise.

$$i_t = \sigma(w_i \cdot [h_{t-1}, x_t] + b_i) \qquad (5\text{-}2)$$

$$\tilde{c}_t = tanh(W_c . [h_{t-1}, x_t] + b_c) \qquad (5\text{-}3)$$

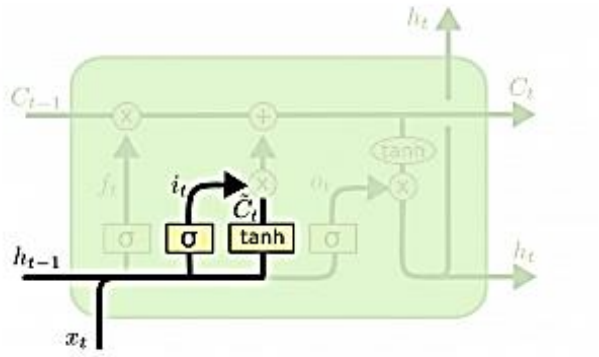


Abbildung 5-5: Die Speicherschicht [29]

- Neuer Zustand Schicht. In dieser Schicht werden die Informationen im Zellzustand gefiltert, um zu bestimmen, welche Informationen für die Ausgabe bleiben und am Ausgang erhalten werden. Hierfür werden zwei Schichten verwendet, eine Sigmoidschicht und eine Tanh-Schicht. Die Tahn-Schicht

skaliert hierbei die Werte des Zellzustands zwischen (-1 und 1), während die Sigmoidschicht bestimmt, welche Daten wiedergegeben werden müssen. Die Abbildung 5-6 verdeutlicht diesen Prozess und die folgenden Gleichungen liefern die mathematische Beschreibung. [29]

$$o_t = \sigma(w_0[h_{t-1}, x_t] + b_0) \quad (5\text{-}4)$$

$$h_t = o_t * tanh(C_t) \quad (5\text{-}5)$$

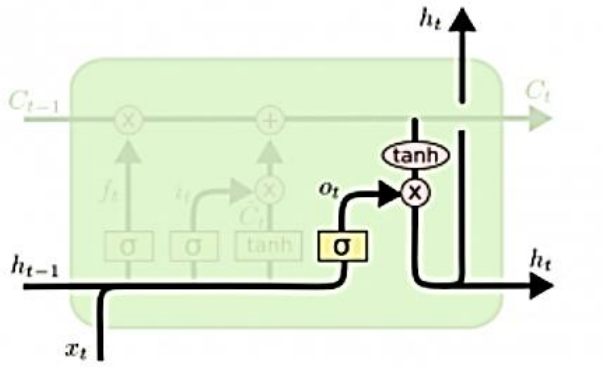


Abbildung 5-6: Änderung des Zellzustands [29]

5.3 LSTM Netze auf dem Mikrocontroller

Tensorflow-API unterstützt Vielzahl von RNN-Strukturen wie RNN, GRU, LSTM und ConvLstm. Was in dieser Arbeit vom Interesse ist, ist die Ausführung eines Spracherkennungsmodells unter Verwendung von LSTM-Netzen auf dem Mikrocontroller. Dies ist zurzeit eine schwierige Aufgabe, weil Tensorflow for Mikrocontroller keine RNN-Netze auf dem Mikrocontroller unterstützt. Es gibt trotzdem eine Lösung für dieses Problem.

Eine LSTM-Schicht verwendet eine Schleife(for-Schleife) zur Implementierung der Zeitschritten einer Sequenz. Diese Schleife ist anscheinend noch nicht unterstützt in Mikrocontrolleranwendungen, daher muss sie von dem Modell entfernt werden, um eine mögliche Implementierung auf dem Mikrocontroller zu ermöglichen. Tensorflow bietet hierfür die Möglichkeit, LSTM-Netze zu entfalten oder abzurollen wie in Abbildung 5-1 zu sehen ist. Diese Methode beschleunigt deutlich die Inferenz auf dem Mikrocontroller auf Kosten der Größe des Modells. Daher muss die Anzahl der Zeitschritten kleinstmöglich sein, um ein möglichst kleines Modell zu erhalten, das auf dem Mikrocontroller funktionsfähig ist. Das funktioniert eigentlich, weil eine LSTM-

Schicht letztendlich aus Komponenten wie Relu ,Tanh , und FC-Schichten besteht, die auf dem Mikrocontroller unterstützt sind.

5.4 Spektrogramm als Eingabe des LSTM-Netzs

LSTM-Netze oder auch RNN-Netze können als Klassifikatoren eingesetzt werden, indem die zweidimensionale Eingabe in Zeitschritte aufgeteilt werden, und somit kann das Spektrogramm oder auch die Bilder in ein LSTM-Netz eingespeist werden. Die Größe des Modells wird in diesem Fall von den Dimensionen des Spektrogramms abhängen. Außerdem hängt die Größe des Modells stark von der Anzahl der Zeitschritten der Sequenz ab. Zu diesem Zweck werden die Zeilen des Spektrogramms als Zeitschritten und die Spalten als Größe des Eingangs betrachtet. Die Abbildung 5-7 verdeutlicht das Prinzip.

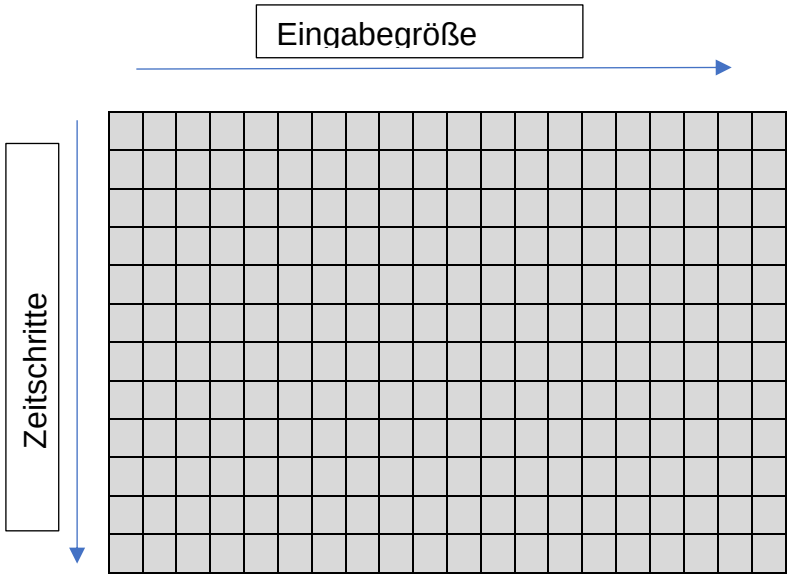


Abbildung 5-7: Aufteilung des Spektrogramms in Zeitschritte

Die Anzahl der Zeitschritte hat einen größeren Einfluss auf die Größe des Modelles als die Größe der Eingabe. Daher ist es wichtig zu erfahren, wie groß das Modell bei verschiedenen Größen des Spektrogramms sein kann und welches Modell auf dem Mikrocontroller funktioniert. Dafür wurden LSTM-Modelle erstellt und auf den ESP-Mikrocontroller übertragen. Die Tabelle 5-1 zeigt, wie sich die Größe des Modells und die Accuyracy-Rate mit den Dimensionen und der Anzahl der LSTM-Einheiten ändert.[17]

[17] Im Kapitel 4 wurde ein Spektrogramm mit der Größe (49 ,40) als Eingabe des Faltungsnetzes verwendet.

Tabelle 5-1:Die Größe des LSTM-Netzes bei unterschiedlichen Spektrogrammen

Das Modell	Spektrogramm	LSTM-Units	Die Größe des Modells (Float32)	Die Größe des Modells (Int8)	Accuracy-Rate	Zustand Auf dem ESP-32
Modell 1	49x40	32	153508 Bytes	Bytes 80000	93.66 %	Funktioniert Nicht
Modell 2	49x40	64	224804 Bytes	203216 Bytes	94.80 %	Funktioniert Nicht
Modell 3	20x98	32	115924 Bytes	90160 Bytes	85.60 %	Funktioniert
Modell 4	20x98	64	216916 Bytes	115232 Bytes	88.94 %	Funktioniert

Es kann der Tabelle 5-1 entnommen werden, dass die Dimensionen des Spektrogramms eine entscheidende Rolle bei der Bestimmung der Modellgröße spielen, wobei 49 in ersten zwei Modellen und 20 in den letzten zwei Modellen die Anzahl der Zeitschritte darstellen. Die Accuracy-Rate ist viel besser bei größerer Anzahl der Zeitschritte aber das Modell funktioniert nicht. Die Abbildung 5-8 zeigt das Spektrogramm 20x98.

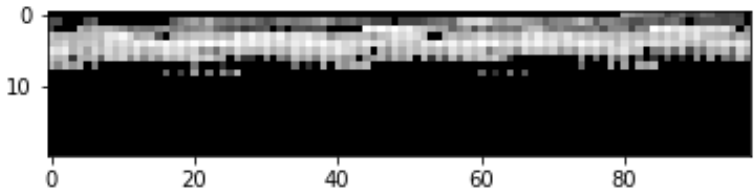


Abbildung 5-8:Das Spektrogramm 20x98

Da die Modelle bei dem Spektrogramm 49x40 auf dem Mikrocontroller nicht funktionieren, muss das Spektrogramm 20x98 verwendet werden, es sei denn, die größeren Modelle werden auf leistungsfähigeren Boards implementiert, wo der zur

Verfügung stehende Arbeitsspeicher größer ist.[18] Die Anzahl der LSTM-Units beeinflusst ebenfalls die Größe des Modells und die Inferenzzeit auf dem Board.

5.5 Trainieren des Netzes

Das Modell besteht aus einer LSTM-Schicht mit 64 Einheiten, einer FC-Schicht, einer Regularisierung-Schicht und einer Softmax-Schicht. Dieses Modell muss so trainiert werden, dass es die Wörter ein und aus erkennen kann. Außerdem kann das Modell die zwei Klassen Ruhe und unbekanntes Wort unterscheiden, und somit gibt es insgesamt 4 Klassen. Abbildung 5-9 zeigt eine Visualisierung des Modells unter Verwendung von Tensorboard. Die Datenproben, mit denen das Netz trainiert wird, sind dieselben Trainingsdaten, die im Kapitel 4 beim Training vom Faltungsnetz verwendet worden sind. Dadurch lässt sich das LSTM-Netz mit dem CNN-Netz vergleichen. Nachfolgend ist in Tabelle 5-2 die Modellgröße dargestellt.

Tabelle 5-2: Die Modellgröße

Das Modell	Größe des Modells
Float32	214740
Int8	113280

Tensorboard ist das Visualisierungs-Toolkit von Tensorflow, mit dem sich das detaillierte Graph des Modells, die skalaren Kurven und Histogramme der Gewichtungen visualisieren lassen. In Abbildung 5-10 werden die Metriken Loss und Accuracy während des Trainierens angezeigt (die Trainingskurve in rot und Validierungskurve in blau).

Es ist in Abbildung 5-10 ersichtlich, dass Überanpassung (Overfitting) vorliegt, weil das Netz die Trainingsdaten gut klassifiziert, wobei die Training-Accuracy ungefähr 98.5 beträgt (rote Kurve). Hingegen betrug die Validation-Accuracy ungefähr 94.3%, wobei das Netz auf neue Daten (Validationsdaten) schlecht verallgemeinert. Dieses Ergebnis lässt sich durch Erhöhung der Regularisierung verbessern, wobei das Netz eine noch bessere Accuracy-Rate aufweist. Noch bessere Ergebnisse können durch Hinzufügen von einer weiteren LSTM-Schicht erzielt werden, aber das Modell

[18] Im Kapitel 8 wurde ein sehr interessantes Board vorgeschlagen, das vor Kurzem erschien. Dieses Board verfügt über viel größeren Arbeitsspeicher und leistungsfähigeren Prozessor.

Funktioniert leider nicht auf dem Mikrocontroller in diesem Fall. Die Abbildung 5-11 veranschaulicht die Verbesserung der Ergebnisse durch die Erhöhung der Regularisierung.

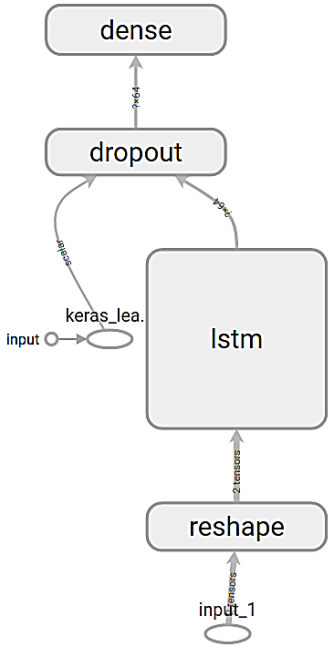


Abbildung 5-9: Visualisierung des LSTM-Modells mit Tensorboard

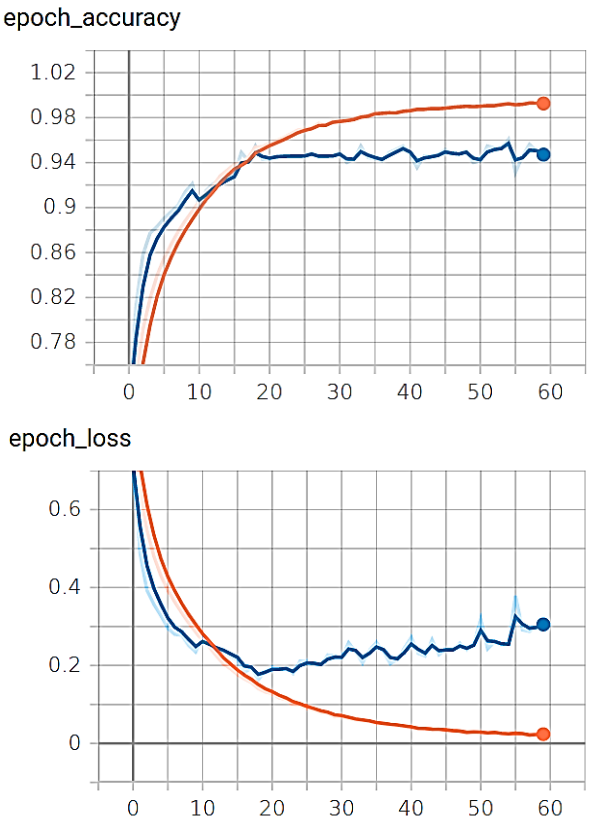


Abbildung 5-10: Visualisierung der Loss und Accuracy

Nachdem das Modell trainiert wurde, muss es auf den Mikrocontroller übertragen werden. Anschließend werden einige zeitlichen Messungen auf dem Mikrocontroller durchgeführt. Die Tabelle 5-3 zeigt die Inferenzzeit auf dem ESP-EYE-Board und die Accuracy-Rate der verschiedenen Modelle.[19]

Tabelle 5-3: Zeitliche Messungen auf dem ESP-EYE

Das Modell	Größe des Models	Inferenzzeit	Accuracy
LSTM (32 Units) Float32	115924 Bytes	100 ms	92.11%
LSTM (32 Units) Int8	90160 Bytes	92 ms	90.643275 %
LSTM (64 Units) Float32	214740 Bytes	200 ms	95.18%
LSTM (64 Units) Int8	113280 Bytes	197 ms	94.298246 %

In Tabelle 5-3 ist ersichtlich, dass die Größe beim ersten Modell um 21% und beim zweiten Modell um 52% durch die Quantisierung verringert wird. Diese Verkleinerung der Modellgröße ist klein im Vergleich mit 75% beim CNN-Netzen. Der Grund dafür ist, dass der Anteil der trainierbaren Parameter(Gewichtungen) beim LSTM ist kleiner[20]. Dies ist in Hello World-Beispiel im Kapitel 3 zu sehen.

5.6 Vergleich mit dem CNN-Modell

Das LSTM-Modell hat eine geringere Accuracy-Rate im Vergleich mit dem CNN-Modell im Kapitel 4 aufgewiesen. Hierbei muss beachtet werden, dass das LSTM-Modell mit umgeformtem Spektrogramm trainiert wurde. Diese Umformung des Spektrogramms hat sich auf die Accuracy-Rate negativ ausgewirkt, wie es in Tabelle 5-1 erkennbar ist. Trotzdem schneiden die Faltungsnetze bei solchen Klassifizierungsaufgaben besser als die LSTM-Netze ab. Außerdem ist die Inferenzzeit des LSTM-Modells auf dem Mikrocontroller kleiner als die Inferenzzeit des CNN-Netzes. Dies kann auf die Anzahl der arithmetischen Operationen zurückgeführt werden, die pro

[19] Die in Tabelle 5-1 und 5-3 gezeigten Accuracy-Raten unterscheiden sich voneinander, weil das Modell mit unterschiedlichen Datensätzen trainiert worden ist.

[20] Mit der Größe des TFlite-Modells werden nicht nur die Gewichtungen des Modells ausgedrückt, sondern auch der Modellgraph.

Sekunde abgearbeitet werden müssen, was bedeutet, dass bei einem größeren Modell (mehr als eine LSTM-Schicht) die Inferenzzeit viel größer sein könnte.

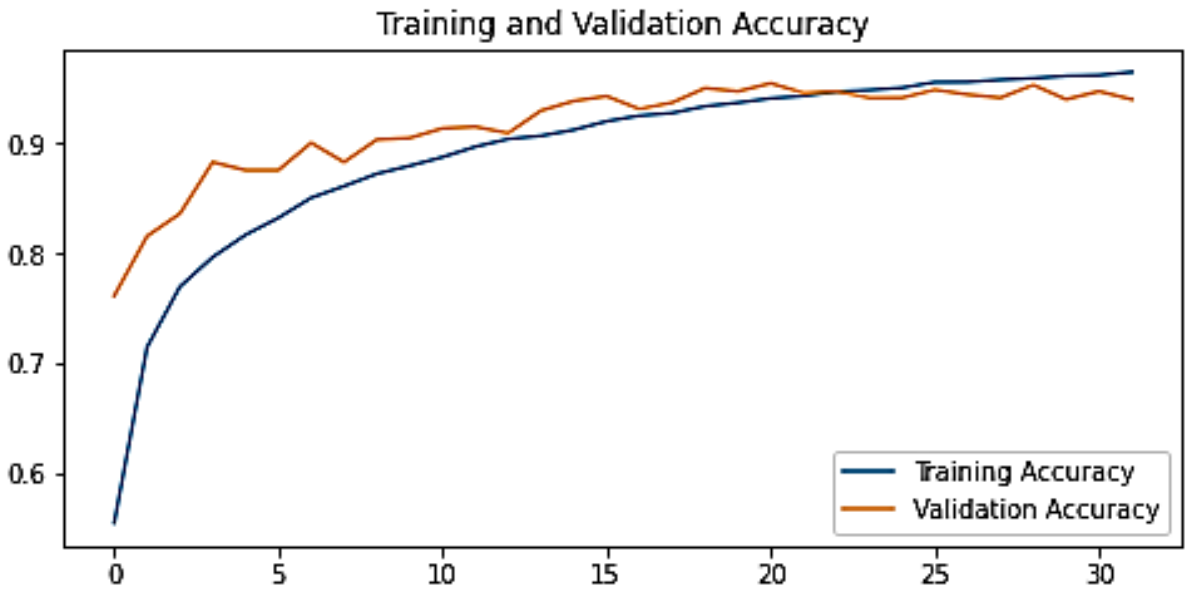


Abbildung 5-11: Accuracy-Rate nach Erhöhung der Regularisierung

Kapitel 6 - Transfer-Learning auf dem Mikrocontroller

Die bis jetzt untersuchten Erkennungsmodelle sind klein im Vergleich mit größeren Modellen und benötigten Hardware-Ressourcen, die in einem Personalrechner vorhanden sein könnten.

Bei komplexeren Erkennungsproblemen werden die Machine-Learning-Modelle viel größer (bis zu 1GByte) und rechenintensiver, sodass das Training Tage oder Wochen andauern könnte, auch wenn die verwendeten Rechner Leistungsfähig sind. Solche Modelle werden normallerweise in Großunternehmen wie Google trainiert, die über Leistungsstarke Rechenzentren und Big-Dat verfügen. Einige dieser vortrainierten Architekturen werden als fertig trainierte Modelle zur Verfügung gestellt. In diesem Kapitel wird eine Technik, die Transfer-Learning genannt wird, vorgestellt und nach möglicher Implementierung auf dem Mikrocontroller gesucht. Dabei wird ein vortrainiertes Netz verwendet, um ein Bildklassifizierungsmodell zu erstellen, das die Bilder von drei Klassen (Hintergrund , Hund , Katze) erkennen und unterscheiden kann. Anschließend wird das Modell auf das ESP-Board übertragen, und somit wird das größte Modell implementiert, das auf den verfügbaren Boards ausgeführt werden kann.

6.1 Grundlagen und Anwendungsfälle

Unter der Bezeichnung Transfer-Learning versteht man das Übertragen der Gewichtungen eins bereits trainierten Modells auf ein neues Netz für neue Aufgabe. Dabei werden die vortrainierten Schichten entweder fixiert und nur die Klassifizierungsschichten am Ausgang des Netzes nachtrainiert, oder werden alle Schichten für die neue Aufgabe weiter trainiert. Die vortrainierten Schichten sind normallerweise die Extraktionsschichten oder Faltungsschichten in einem CNN-Netz, während die FC-Schichten Klassifizierungsschichten sind.

Der Einsatz dieser Technik hat einen großen Vorteil, wenn das zu lösende Problem sehr groß oder das verwendete Modell sehr tief ist. In diesem Fall wird das Trainieren des Modells sehr rechenintensiv und zeitaufwendig. Solche Modelle sind normallerweise als Objektserkennungsmodelle zu finden.

Der Vorteil dieser Technik besteht darin, dass die Extraktionsschichten eines vortrainierten Modells bereits gelernt haben, Merkmale zu extrahieren und Objekte zu unterscheiden, wobei diese Schichten für neue Aufgaben wiederverwendet werden können. Diese Technik kommt außerdem zum Einsatz, wenn das Netz auf völlig neue Aufgabe vollständig trainiert wird. Hierbei werden die erlernten Gewichtungen zum Initialisieren des Modells verwendet (keine zufällig ausgewählte Gewichtungswerte). Dies wird zu einer großen Verkürzung der Trainingszeit führen.

Es gibt eigentlich nicht nur eine Transfer-Learning-Strategie, sondern mehrere, die je nach Anwendung eingesetzt werden können. Die verwendete Strategie hängt dabei von zwei Faktoren ab:

- Die Größe des verfügbaren Datensatzes.
- Die Ähnlichkeit zwischen den neuen und alten Klassen. [30]

Diese Zwei Faktoren bestimmen, wie das Modell auf eine neue Aufgabe transferiert wird. Hierbei werden vier Anwendungsfälle unterschieden:

- In dem ersten Anwendungsfall weisen die zu erkennenden Objekten ähnliche Strukturen auf, die das Netz bereits erkennen kann. Gleichzeitig stehen nur kleine Trainingsdatenmenge zur Verfügung. Hier wird die Klassifizierungsschicht (normallerweise Fully-Connected) durch eine neue ersetzt und mit zufälligen Werten initialisiert, während die Extraktionsschichten (Faltungsschichten) komplett beibehalten werden. Anschließend wird die Ausgangschicht mit dem kleinen Datensatz darauf trainiert, die Ausgabe der vortrainierten Schichten richtig zu klassifizieren. Dabei durchlaufen die Daten die beibehaltenen und fixierten Schichten, die Strukturen und Objekte erkennen können, ohne die Gewichtungen anzupassen, bevor diese Daten in die hinzugefügten Schichten eingespeist werden. Der Trainingsaufwand dieser Strategie ist sehr gering und die Ergebnisse sind gut im Vergleich mit den erreichbaren Ergebnissen, wenn das Netz komplett neu auf diesen kleinen Datensatz trainiert würde.
- Wenn der zur Verfügung stehende Datensatz klein ist und die neuen Klassen unterscheiden sich von dem erlernten Klassen, bringt ein einfaches Ersetzen der Fully-Connected-Schichten keinen Erfolg, weil die vorigen Faltungs-schichten die zuerkennenden Objekte nicht abstrahieren können. Trotzdem

kann Transfer-Learning sehr nützlich sein. Die ersten Faltungsschichten des vortrainierten Modells lernen einfache Strukturen, auch die nicht zu den zu erkennenden Klassen gehören, zu extrahieren. Hingegen können die späteren Faltungsschichten des vortrainierten Netzes mit höheren Abstraktionen bestimmte Strukturen und Objekte abstrahieren. Diese Schichten tragen zur Erkennung neuer Klassen und Objekte nicht bei. Daher werden sie von dem Netz abgeschnitten. Die Fully-Connected-Schicht wird dann direkt mit dem beibehaltenen Schichten des vortrainierten Modells verbunden und auf den verfügbaren Datensatz trainiert. Die Anzahl der Neuronen in den FC-Schichten entspricht in diesem Fall der Anzahl der zu erkennenden Klassen. Auf diese Weise können die anhand von Millionen Bildern gelernten Merkmale für neue Aufgaben benutzt werden.

- Wenn ein großer Datensatz zur Verfügung steht und die Klassen ähnlich sind, wird das vortrainierte Modell als Basis-Modell verwendet und die Anzahl der Ausgänge wird der Anzahl der neuen Klassen entsprechend angepasst. Die hinzugefügte Schicht wird dann mit zufälligen Gewichtungen initialisiert und trainiert. Beim Training wird in diesem Fall das Basis-Modell nicht fixiert, sondern mit der neuen Ausgangschicht trainiert. Da das Basis-Modell die Objekte erkennen kann, wird die Trainingszeit dadurch verkürzt. Die Accuracy-Rate, die das gesamte Modell erreichen kann, ist mindestens so groß, als würde das Modell komplett neu trainiert.
- Wenn sich die neuen Daten von den erlernten Daten unterscheiden und ein großer Datensatz zur Verfügung steht, wird In diesem Fall das vortrainierte Modell als Basis-Modell verwendet und die Ausgangschicht durch eine andere ersetzt, dann wird das gesamte Modell trainiert. Auch wenn die Klassen unterschiedlich sind, lässt sich die Trainingszeit mit dieser Strategie verkürzen. [30]

6.2 Vortrainierte Modelle in Tensorflow

Tensorflow bietet einige vortrainierten Architekturen und Netze, die direkt in einem Projekt oder als Basis-Modell für neue Anwendungen verwendet werden können. Diese Modelle erstrecken sich über Computervision, Sprachübersetzer, Speech to Text Anwendungen und vieles mehr. Sie sind sehr tief und groß und benötigen einen

rechenfähigen Prozessor und großen Speicher. In Tabelle 6-1 sind die wichtigsten Netze mit der jeweiligen Accuracy-Rate, Parameteranzahl und Modelltiefe aufgelistet.

Für die Implementierung auf dem Mikrocontroller muss nach einem Modell gesucht werden, das auf dem ESP-Board funktionieren kann und sich als Basis-Modell unter Verwendung von Transfer-Learning einsetzen lässt. Verglichen mit den in dieser Arbeit erstellten Architekturen, sind diese Modelle viel tiefer und größer, wie der Tabelle 6-1 zu entnehmen ist. Die Verwendung dieser Modelle in einem Projekt bringt mit sich einige Vorteile, die dazu beitragen könnten, die Modellgröße und die Parameteranzahl zu minimieren, damit sie auf einem eingebetteten System ausgeführt werden können. Diese Vorteile sind nachfolgend ausgelistet:

- Einige Objekterkennungsmodelle wurden auf bestimmte Bildgröße trainiert, es ist aber möglich, die Größe des Eingangs beliebig anzupassen. Dies hat einen großen Einfluss auf die Größe des Modells, die Anzahl der Gewichtungen, und infolgedessen auf den benötigten Arbeitsspeicher.
- Mobilenet-Netze haben einen in unserem Fall entscheidenden Vorteil, der möglich macht, die Modellgröße auf das Kleinstmögliche zu minimieren, wobei diese Strukturen einen Parameter haben, der ALPHA oder die Breite des Modells heißt. Durch diesen Parameter lässt sich die Anzahl der Faltungsfilter in jeder Schicht des vortrainierten Modells ändern, was dazu führt, die Größe des Modells zu ändern.
- Die vortrainierten Modelle kann man direkt verwenden oder auf neue Aufgaben transferieren.
- Die Aktivierungsfunktion der Ausgangschicht kann beliebig bei einer direkten Verwendung geändert werden.
- Die Architektur kann mit zufälligen Gewichtungen initialisiert und vom Anfang an auf neuen Datensatz trainiert werden.

Tabelle 6-1: Die wichtigsten vortrainierten Modelle in Tensorflow [31]

Das Modell	Größe	Top-1 Accuracy	Top-5 Accuracy	Trainierbare Parameter	Modelltiefe
Xception	88 MB	0.790	0.945	22,910,480	126

Das Modell	Größe	Top-1 Accuracy	Top-5 Accuracy	Trainierbare Parameter	Modelltiefe
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-

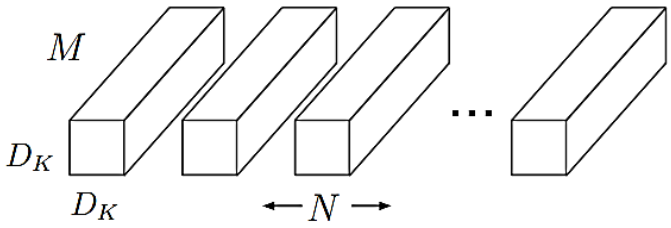
Das Modell	Größe	Top-1 Accuracy	Top-5 Accuracy	Trainierbare Parameter	Modelltiefe
EfficientNetB3	48 MB	-	-	12,320,535	-
EfficientNetB4	75 MB	-	-	19,466,823	-
EfficientNetB5	118 MB	-	-	30,562,527	-
EfficientNetB6	166 MB	-	-	43,265,143	-
EfficientNetB7	256 MB	-	-	66,658,687	-

6.3 Die Mobilenet-Architekturen

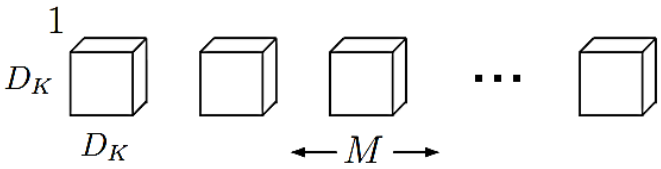
Wie der Tabelle 6-1 entnommen werden kann, sind die Mobilenet-Architekturen von besonderem Interesse, weil sie die Einzigen, die akzeptable Größe für eine mögliche Implementierung auf dem Mikrocontroller haben. Nachfolgend wird diese Architektur vorgestellt und untersucht

Diese Architekturen basieren auf einer speziellen Art der Faltung sogenannte Depthwise Separable Convolution. Diese Faltungsmethode ist eine faktorisierte Faltung, welche die herkömmliche Faltung in zwei Schichten erledigt ,eine separate Schicht für die Kombinierung von Merkmalen und eine separate Schicht für die Filterung.

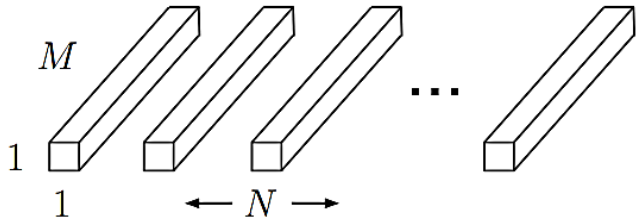
In Mobilenet-Architekturen werden auf jeden Eingangskanal ein einziges Filter angewendet. Im nächsten Schritt wendet die Pointwise-Faltung eine 1x1-Faltung an, um die Ausgänge zu kombinieren. In der herkömmlichen Faltung finden die Filterung und die Kombinierung hingegen in einer einzigen Schicht statt. Diese Faktorisierung der Faltung reduziert drastisch die Größe des Modells und die Anzahl der arithmetischen Operationen. Die Abbildung 6-1 veranschaulicht den Unterschied zwischen der herkömmlichen Faltung und der Depthwise separable Convolution.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) Pointwise Convolution

Abbildung 6-1: Depthwise separable Convolution im Vergleich mit der Standerdfaltung [19]

Eine Standerd-Faltungsschicht, wie die im Spracherkennungsmodell verwendet worden ist, nimmt als Eingabe ein Feature-Map mit der Größe D_f x D_f x M, Wobei D_f die Höhe und die Breite des Eingangskanals (ein Bild beispielsweise), während M die Anzahl der Eingangskanäle ist (bei Farbbildern 3). Jede Faltungsschicht produziert in diesem Fall eine Feature-Map mit der Größe D_f x D_f x M x N, wobei N die Anzahl der Faltungsfilter oder Kernel. Hier wird die Rechenaufwand dieser Schicht wie folgt berechnet:

$$\boldsymbol{D_k \cdot D_k \cdot M \cdot N \cdot D_F \cdot D_F} \qquad (6\text{-}1)$$

Wobei D_k x D_k die Filterdimensionen sind. Der Rechenaufwand hängt in diesem Fall multiplikativ von der Eingangsdimension D_F, der Anzahl der Eingangskanäle M, der Anzahl der Faltungsfilter N und der Größe des Faltungsfilters D_k ab.

Bei Depthwise-Faltung ist die Anzahl der Filter N=1. Dies führt zur großen Reduzierung der Anzahl der arithmetischen Operationen im Vergleich mit Standard-Faltung, wobei der Rechenaufwand wie folgt angegeben wird:

$$D_k \cdot D_k \cdot M \cdot D_F \cdot D_F \qquad (6\text{-}2)$$

Allerdings wendet diese Schicht nur die Filterung auf die Eingabe an, ohne die Kanäle zu kombinieren. Daher wird eine zusätzliche Schicht, die die Ausgabe der Depthwise-Schicht kombiniert und Feature-Maps produziert. Der totale Rechenaufwand ist diesem Fall die Summe der benötigten Rechenoperationen der zwei Schichten und wird wie folgt angegeben:

$$D_k \cdot D_k \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \qquad (6\text{-}3)$$

Die Verringerung der Anzahl der Rechenoperationen kann berechnet werden, indem der Rechenaufwand der Standerdfaltungsschicht durch den Rechenaufwand der Depthwise Separable Schicht dividiert wird, und wird wie folgt angegeben:

$$\frac{D_k \cdot D_k \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_k \cdot D_k \cdot M \cdot N . D_F \cdot D_F}$$

$$= \frac{\mathbf{1}}{\boldsymbol{N}} + \frac{\mathbf{1}}{\boldsymbol{D_k^2}} \qquad (6\text{-}4)$$

Anmerkung: in den bisherigen Berechnungen wurde angenommen, dass die Dimensionen der Faltungsfilter gleich sind (quadratförmig). Ebenfalls wurde eine ähnliche Annahme für die Eingabe gemacht.

Das Mobilenet-Netz basiert auf diese Faltungsstrukturen, die geringeren Rechenaufwand und Speicherbedarf während der Ausführung benötigen. Die Architektur dieses Netz besteht aus 28 Depthwise- und Pointwise-Faltungsschichten. Jeder Faltungsschicht folgen ein Batchnorm und Reul-Aktivierungsfunktion. Die letzte Schicht der Architektur ist eine FC-Schicht und stellt die Klassifizierungsschicht dar, gefolgt von einer Softmax-Aktivierungsfunktion. In Tabelle 6-2 ist das Mobilenet-Netz anschaulich beschrieben. [21] [19]

[21] Im Kapitel 3 wurde erwähnt, das Modilenet-Architektur aus 14 Schichten besteht. Hierbei ist gemeint, dass jede zwei Schichten (Depthwise und Pointwise) eine Faltungsschicht bilden.

Tabelle 6-2: Die Mobilenet-Architektur [19]

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

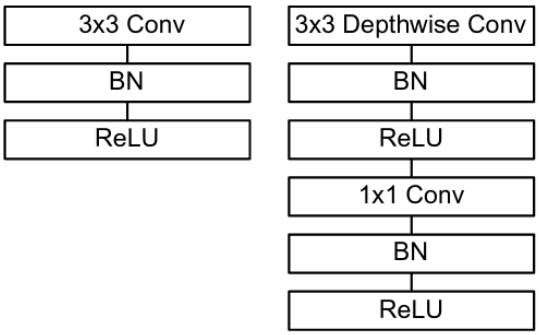


Abbildung 6-2: links eine Standardfaltungsschicht und rechts eine Depthwise und Pointwise Faltungsschichten [19]

Mobilnet-Architektur wurde ursprünglich für Klassifizierung von Farbbildern mit der Auflösung 224 x 224 erstellt. Daher hat die Eingabe der ersten Schicht die Form 224 x224 x 3, wobei 3 die Anzahl der Farbkanäle ist. Der Trainingsdatensatz, der beim Trainieren des Netzes verwendet wurde, ist ImageNet-Datensatz, wobei es mit 1001 Klassen trainiert wurde(Anhang C). Deswegen wird für die Ausgabe eine FC-Schicht mit 1001 Neuronen benötigt. Der größte Rechenaufwand während der Ausführung (fast 95%) entsteht in den 1 x 1 Pointwise-Faltungsschichten, wo sich 94.86% der arithmetischen Operationen befinden. Die folgende Tabelle 6-3 veranschaulicht die Anzahl der arithmetischen Operationen und die Anzahl der Parameter (Gewichtungen) jeder Schichtart.

Tabelle 6-3: : Die Anzahl der Parameter und arithmetischen Operationen jeder Schicht [19]

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

Die Latenzzeit und der Speicherbedarf der Mobilenet-Architekturen sind sehr niedrig im Vergleich mit den anderen vorhandenen Architekturen. In manchen Fällen sind Trotzdem der Rechenaufwand und die Latenzzeit sehr groß, wenn die Mobilenet-Architektur auf einem eingebetteten System wie einem Mikrocontroller ausgeführt werden muss. Hier hat diese Architektur, wie erwähnt, einen sehr wichtigen Parameter, der Alpha (oder Width Multiplier) genannt wird. Dieser Parameter bestimmt die Anzahl der Eingangskanäle (die Eingangstiefe) und die Anzahl der Ausgangskanäle (Ausgangstiefe). Unter Berücksichtigung von Alpha wird der Rechenaufwand wie folgt angegeben:

$$D_k \cdot D_k \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F \qquad (6\text{-}5)$$

Hierbei sind M und N die Anzahl der ursprünglichen Eingabekanäle und Filter-Kernel, wobei Alpha die Multiplikationsfaktor und einen Wert zwischen 0 und 1 annehmen kann.[22] Mit niedrigen Alpha-Werten werden der Rechenaufwand aber auch die Accuracy-Rate deutlich verringert. Die Tabelle 6-4 zeigt auf, wie sich die Accuracy-Rate sowie der Rechenaufwand mit Alpha ändern.

Tabelle 6-4: Die Accuracy-Rate bei verschiedenen Werten von Alpha [19]

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Es gibt einen weiteren Faktor, der ebenfalls dabei hilft, die Größe des Modells und den Rechenaufwand zu minimieren. Dieser Parameter ist die Auflösungsfaktor oder auf Englisch Resolution Multiplier ρ , der ebenfalls einen Wert zwischen 0 und 1 annimmt.

[22] Eigentlich kann Alpha nur einen der Werte 0.25 ,0.50 , 0.75 und 1 annehmen

Die Latenzzeit lässt sich mithilfe dieses Parameter noch mehr verkürzen, wobei der Rechenaufwand unter Berücksichtigung von α und ρ zusammen wie folgt berechnet werden kann :

$$D_k \cdot D_k \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho \qquad (6\text{-}6)$$

Die folgende Tabelle 6-5 verdeutlicht, wie die Anzahl der trainierbaren Parameter sowie die Accuracy-Rate mit der Auflösung minimiert werden .

Tabelle 6-5: Die arithmetischen Operationen und Accuracy-Rate bei unterschiedlichen Auflösungswerten [19]

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

6.4 ImageNet-Datensatz

Dieser Datensatz enthält mehr als 14 Millionen von Farbbildern in 20000 Kategorien und kann für Forschungsprojekte eingesetzt werden, wobei jede Kategorie aus Hunderten von Bildern besteht. Nachfolgend werden die Farbbilder von drei Klassen dieses Datensatzes für das Nachtrainieren vom Mobilenet unter Verwendung von Transfer-Learning eingesetzt.

6.5 Transfer-Learning zur Erkennung von Hunden und Katzen

Als nächstes werden die bereits erworbenen theoretischen Kenntnisse zur Erstellung eines Erkennungsmodells zunutze gemacht, das erkennen kann, ob sich eine Katze oder ein Hund auf dem mit der Kamera aufgenommenen Bild befindet. Dabei werden die erlernten Gewichtungen auf neue Aufgabe übertragen. Anschließend wird das Modell auf einem passenden Mikrokontroller ausgeführt.

6.5.1 Vorbereitung vom Basis-Modell

Wie bereits erwähnt wurde, wurde das Mobilenet-Modell auf die Bilder von 1001 unterschiedlichen Klassen trainiert. Unter diesen Klassen sind Katzen und Hunden zu finden. Dieses vortrainierte Modell soll jetzt eine neue Aufgabe übernehmen.

Das Basis-Modell ist ein von dem vortrainierten Modell abgeleitetes Netz, das nur aus den Extraktionsschichten (Faltungsschichten) besteht, wobei die Klassifikations-schichten(FC-Schichten) abgeschnitten werden.

Als erstes muss das Basis-Modell importiert werden. Dafür müssen die Größe der Eingabebilder (Auflösung) und der Alpah-Parameter festgelegt werden. Durch Experimentieren wurde festgestellt, dass das einzige Mobilenet-Modell, das auf dem Mikrocontroller funktioniert, ist das Modell Mobilenet v1 (Version 1) mit Alpha 0.25 und Bildauflösung 96x96x3. Hier muss beachtet werden, dass die Größe des Modells auch von der Anzahl der zu klassifizierenden Klassen abhängt.

Wie eben erwähnt wurde, wird das vortrainierte Mobilenet-Modell v1 ohne die Klassifizierungsschicht benötigt. Daher muss die Klassifizierungsschicht von dem ursprünglichen Modell abgeschnitten werden. Die folgende Abbildung 6-3 zeigt das abgeschnittene Netzteil. Die zurückgebliebenen Faltungsschichten stellen das Basis-Modell dar. Dabei müssen die Gewichtungen des resultierenden Basis-Modells fixiert bleiben. Das bedeutet, dass diese Gewichtungen während des Trainings nicht geändert werden dürfen.

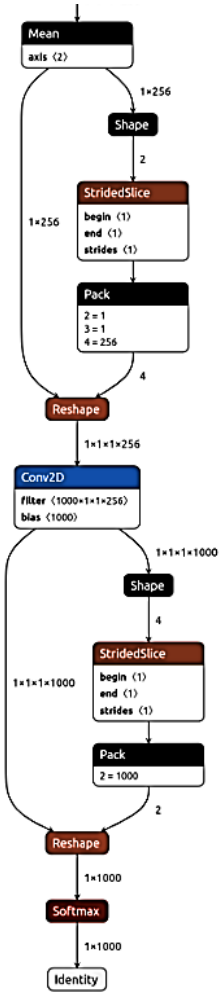


Abbildung 6-3:Die abzuschneidenden Klassifizierungsschichten

6.5.2 Hinzufügen von neuer Klassifizierungsschicht

Als nächstes muss eine eigene Klassifizierungsschicht (Abbildung 6-4) den abgeschnittenen Teil ersetzen. Diese Schicht besteht hauptsächlich aus einer FC-Schicht, einer Reshape-Shicht und einer Softmax-Schicht.

Das neue Modell soll drei Klassen unterscheiden können. Diese Klassen sind Katze, Hund und Hintergrund wenn sich weder Katze noch Hund auf dem aufgenommenen Bild befindet.

Die Eingabe dieses Netzteils hat die Größe der vorigen Faltungsschicht, daher muss die Ausgabe der letzten Faltungsschicht des Basis-Modells ermittelt werden. Andererseits entspricht die Anzahl der Ausgänge der Klassenanzahl des neuen Modells (Drei in diesem Fall). Anschließend werden die hinzugefügten Schichten mit zufälligen Werten initialisiert.

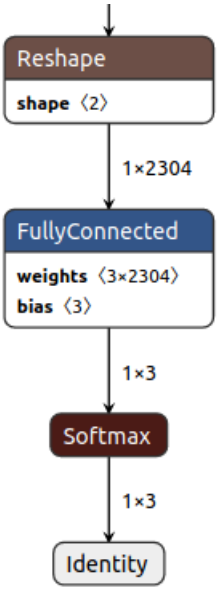


Abbildung 6-4: Die hinzugefügten Schichten.

6.5.3 Trainieren der hinzugefügten Schichten

Nachdem die neuen Schichten hinzugefügt worden sind, müssen sie trainiert werden. Hierfür werden nur eine kleine Menge von Bilderdaten benötigt, die zu den drei Klassen Hunde, Katzen und Background gehören. Die Abbildung 6-5 zeigt einige Trainingsbilder.

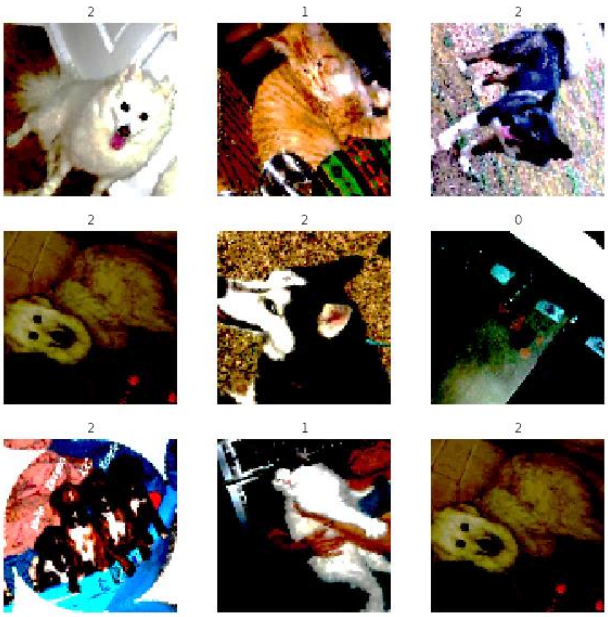


Abbildung 6-5: Trainingsbilder aus dem Trainingsdatensatz ImageNet.

Das Trainieren des Modells findet in zwei Stufen statt. Als Erstes werden nur die hinzugefügten Gewichtungen auf die Trainingsdaten trainiert. Als Nächstes müssen alle Gewichtungen inklusive die des Basis-Modells angepasst werden, wobei dieser

Schritt unter der Bezeichnung Fine-Tuning bekannt ist. Dabei muss beachtet werden, dass Fine-Tuning mit kleiner Lernrate stattfinden muss.

Bevor die Daten in das Modell eingespeist werden, müssen einige Vorverarbeitungen angewendet werden. Die hier benötigten Vorverarbeitungen sind:

- Skalierung von Daten. Weil das Basis-Modell mit bestimmten Pixelwerten trainiert worden ist, während die Pixelwerte der Trainingsbilder uint8 sind.[23]
- Rotation der Bilder. Diese Augmentation-Methode führt zur Verbesserung der Accuracy-Rate.
- Die Größe der Bilder. Weil die mit der Kamera aufgenommenen Bilder eine bestimmte Größe(96x96x3) haben , müssen auch die Trainingsbilder dieselbe Größe haben.

Nachdem das Modell trainiert worden ist, muss es auf dem Mikrocontroller ausgeführt werden. Dies erfordert, dass das Modell kleinstmöglich sein muss. Diese Anforderung kann durch die Anpassung des Alpha-Parameters erfüllt werden. Die folgende Tabelle 6-6 veranschaulicht, wie sich die Größe des Modells bei Bildgröße 96 x 96 x 3 mit Alpah ändern lässt.

Tabelle 6-6: Zeigt die Modellgröße bei verschiedenen Alpha-Werten.

Anzahl der Klassen	Auflösung der Bilder	Alpha	Modellgröße Int8/uint8	Modellgröße Float32
3	96 x 96 x3	0.25	336.6 KByte	888.3 KByte
3	96 x 96 x3	0.5	1 MByte	3.3 MByte
3	96 x 96 x3	0.75	2.140416 MByte	7.33 MByte
3	96 x 96 x3	1.	3.631264 MByte	12.914608 MByte

Das erste Modell bei Alpha= 0.25 funktioniert gut mit einer geringen Latenzzeit. Bei den größeren Modellen bei Alpha > 0.25 taucht das Speicherproblem auf, und somit können diese Modelle nicht auf dem untersuchten Mikrocontroller ausgeführt werden.

[23] Das Mobilenet-Netz wurde auf Trainingsbilder mit Pixelwerten [-1,1] trainiert.

Andererseits hängt die Accuracy-Rate stark von alpha ab, was bedeutet, dass bei Alpha=0.25 keine große Accuracy-Rate erzielt werden kann. In Tabelle 6-7 ist die Accuracy-Rate bei verschiedenen Alpha-Werten aufgelistet.

Tabelle 6-7: Accuracy-Rate bei verschiedenen Alpha-Werten.

Alpha	Accuracy-Rate (Quantisiertes Modell int8)
0.25	81 %
0.50	87 %
0.75	92.3 %
1	94 %

Während des Trainings kann man den Prozess durch die zwei Metriken Loss und Accuracy beobachten. Zum Beispiel kann man dadurch erfahren, ob Overfitting stattgefunden hat. Die Abbildung 6-6 zeigt die zwei Metriken.

Wie bereits erwähnt, findet das Training in zwei Phasen statt. Zuerst werden die hinzugefügten Schichten auf den Trainingsdatensatz trainiert(die Kurven vor der grünen Linien) , während die Gewichtungen des Basis-Modells fixiert bleiben. In der zweiten Phase werden alle Gewichtungen (das Basis-Modell und die hinzugefügten Schichten) auf den Datensatz erneut trainiert (die Kurven nach der grünen Linien).

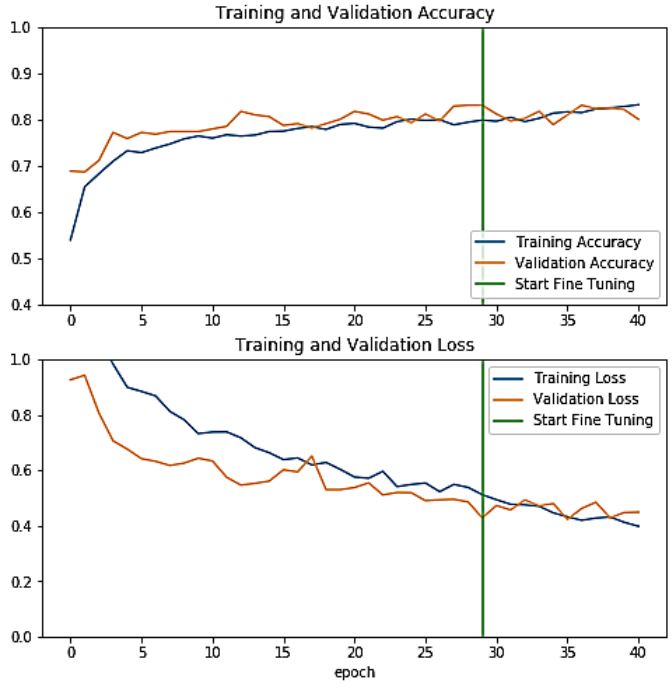


Abbildung 6-6: Loss und Accuracy während des Trainings

6.5.4 **Das C-Programm auf dem Mikrocontroller**

Wie bei den anderen Modellen, die im Laufe dieser Arbeit erstellt worden sind, wird das Modell-Array durch den xxd Befehl erzeugt. Dieses Array kann dann in das C-Programm inkludiert werden, wobei die Ausführung des Modells in einer Schleife Aufgerufen wird.

Das Modell wurde auf dem ESP-Cam-Board ausgeführt, wobei die Ausführungszeit mehr als eine Sekunde dauert. Dabei werden Farbbilder mit der Auflösung 96x96x3 mit der Kamera auf dem Board vor jeder Inferenz aufgenommen und dann in das Modell eingespeist. Die getroffene Vorhersage wird dann wie bei dem Zahlenerkennungsmodell im Kapitel 4auf einer kleinen Anzeige(7 seqments) angezeigt, wobei die Buchstaben H, C, und D auf die drei Klassen Hintergrund, Cat und Dog hindeuten (Abbildung 6-7). Hier muss erwähnt werden, dass die Kamera auf dem ESP-Cam Bilder mit Pixelwerten Uint8 erzeugt, während das Modell int8 Werte akzeptiert, was bedeutet, dass eine Konvertierung der Bilder erforderlich ist, bevor sie in das Modell eingespeist werden. Für mehr Informationen liefert der vollständige Code auf dem Datenträger mehr Einzelheiten.

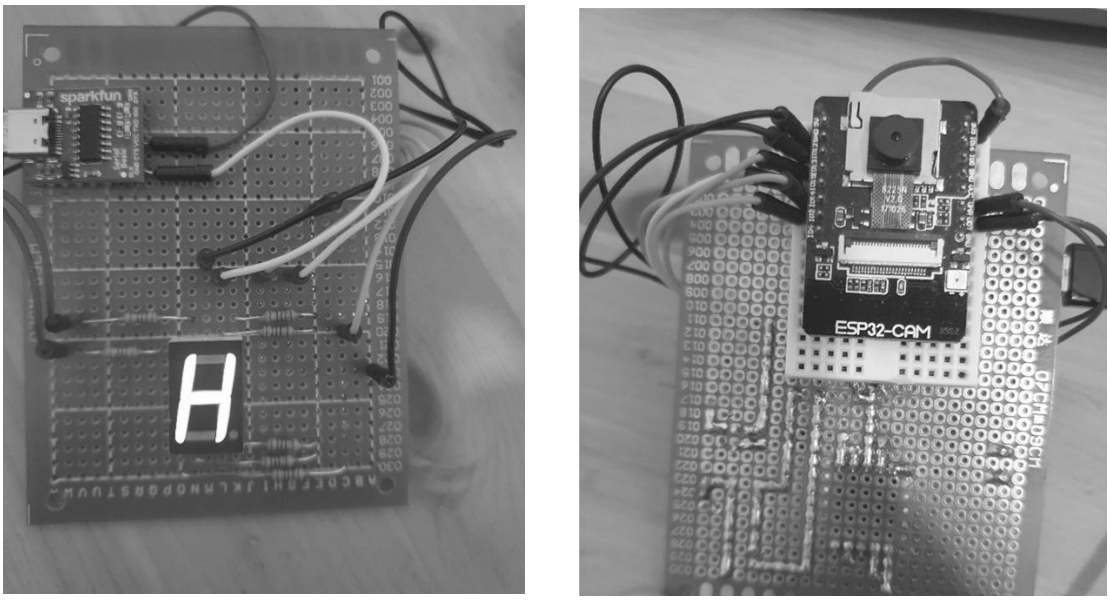


Abbildung 6-7: Die Ausführung des Modells auf dem ESP-Cam-Board

Kapitel 7 - Steuerung einer Lampe per Sprache

In Kapiteln 3, 4 und 5 wurde unter anderem ein Modell trainiert, das einfache Wörter wie ein und aus erkennen kann. Außerdem wurden die Eigenschaften verschiedener Boards, die zur Verfügung stehen, und das zeitliche Verhalten dieser Boards studiert, um zu sehen, inwieweit jedes Modell auf dem Mikrocontroller implementierbar ist. In diesem Kapitel soll ein Gerät entworfen werden, das Sprachbefehle verstehen und eine Last steuern kann. Dieses System soll ein Beispiel dafür sein, wie ein per Sprache interaktives Steuerungssystem aussehen könnte.

7.1 Der Sprachassistent

In den letzten Jahren haben sich unaufhaltsam Cloud-basierten Sprachassistenten verbreitet. Diese Sprachassistenten gehörten bis jetzt den führenden Konzernen und Herstellern, die in der Technik oder auch im Handel tätig sind, wie Amazon und Google. Diese Cloud-basierten Assistenten weisen mehrere Vor- und Nachteile auf, die in Kapitel 2 besprochen wurden. Die Vorteile sind die technischen Ressourcen wie die Riesenrechner, über die diese Firmen verfügen. Diese technischen Ressourcen sind sehr wichtig für die Verarbeitung vom riesigen Datenfluss und für das Trainieren komplexer und großer ML-Modelle. Die Nachteile dieser Sprachassistenten sind die Verzögerungszeiten, die erforderliche schnelle Internetverbindung, und vor allem die geringe Sicherheit, was eine Verletzung des Datenschutzes darstellt. Aus diesen Gründen wurde in den letzten wenigen Jahren eine neue Branche des maschinellen Lernens gegründet, die versucht, ML-Modelle darunter kleine Sprachassistenten auf kleinen SoCs zu implementieren. Diese Branche heißt TinyML. Dabei verlassen im Gegensatz zu cloud-basierten ML-Modellen keine Informationen das Gerät und werden damit viele Probleme gelöst.

Die Funktionsweise des Sprachassistenten kennt fast jeder, der mit modernen Smartphones in Berührung gekommen ist. Das Gerät hört ständig mit und wartet auf bestimmte Schlüsselwörter, Zum Beispiel Bixby bei Samsung. Auf diese Schlüsselwörter folgt den Sprachbefehl oder was gemacht werden muss, zum Beispiel Bixpy wie spät ist es.

Bei den elektrischen Geräten wie Haushaltgeräten, die per Sprache gesteuert werden müssen, müssen sie nur einen kleinen Befehlssatz verstehen, da diese Geräte nur auf bestimmte Sache spezialisiert sind. Zum Beispiel wenn eine Kaffeemaschine per Sprache gesteuert werden muss, muss sie nur wenige Sprachbefehle, wie Maschine Kaffee oder Maschine Tee, verstehen. In diesen Fällen ist es nicht praktisch, alle diese Maschinen mit dem Internet zu verbinden. Es Wäre viel besser ein integrierbares System herzustellen, das die Befehle verstehen und interpretieren kann. Nachfolgend wird ein solches System entworfen.

7.2 Auswahl der Mikrocontrollerboards

Im Kapitel 2 wurden einige Mikrocontrollerboards vorgestellt, die zur Verfügung stehen und gleichzeitig vom Tensorflow unterstützt sind. Diese Boards unterscheiden sich voneinander bezüglich der Taktfrequenz, Arbeitsspeicher, Energieverbrauch und der Kommunikation mit der Außenwelt. Besonders interessant ist das Board ESP-EYE. Es verfügt über einen mächtigen Prozessor mit zwei Kernen und Taktfrequenz bis zu 240 MHz sowie viele Kommunikationsmöglichkeiten, die ermöglichen, mehrere Boards zu vernetzen. Diese beiden Eigenschaften werden in diesem Projekt von großer Bedeutung sein, wobei insgesamt drei ESP-Boards verwendet werden. Diese Boards sind zwei ESP-EYE und ein ESP32 NodeMCU.

7.3 Der Befehlssatz

Eine Lampe soll durch Sprachbefehle gesteuert werden, wobei die Farbe des Lichtes, die Stärke des Lichtes und die Energieversorgung per Sprache eingestellt werden können. Außerdem muss beim ersten Starten ein dreistelligen Sicherheitscode (Pin-Code) eingegeben werden.[24]

Das Spracherkennungssystem muss daher auf drei Schlüsselwörter warten, um zu bestimmen welcher Befehl ausgeführt werden muss. Diese drei Wörter sind Lampe, Stärke und Farbe. Nach jedem dieser Wörter kommt die zweite Hälfte des Befehls. Die Tabelle 7-1 zeigt den Befehlssatz .

[24] Der Pin-Code wird nicht durch eine Tastatur eingegeben, sondern ebenfalls per Sprache.

Tabelle 7-1: Der Befehlssatz des Projektes.

	Der Befehl	**Die Funktionalität des Befehles**
Befehl 1	Lampe ein	Die Lampe wird eingeschaltet
Befehl 2	Lampe Aus	Die Lampe wird ausgeschaltet
Befehl 3	Stärke Up	Die Stärke des Lichtes wird erhöht
Befehl 4	Stärke Down	Die Stärke des Lichtes wird gesenkt
Befehl 5	Farbe One	Grünes Licht
Befehl 6	Farbe Two	Rotes Licht
Befehl 7	Farbe Three	Blaues Licht
Befehl 8	Farbe Four	Weißes Licht
Befehl 9	Lampe Hilfe	Wenn ein Wi-Fi Netz zur Verfügung steht, wird das Gerät im Notfall eine E-Mail an eine vordefinierte Person oder Mehrere Personen schicken.
Befehl 10	Unbekannter Befehl	Wenn ein unbekanntes Wort gesprochen wird, wird dieses Wort als unbekannte Befehl interpretiert.
Befehl 11	Ruhe	Wenn keine akustischen Signale mit dem Mikrophon aufgenommen werden, wird diese Situation als Ruhe klassifiziert.

Dieser Befehlssatz kann sich ändern, je nachdem in welches Gerät das Modell integriert werden muss. Wie es zu sehen ist, bestehen die Befehle aus deutschen Wörtern, die aus erstelltem Datensatz stammen, und englischen Wörtern, die bereits im Datensatz ‘Speech Commands’ vorhanden sind und von Tausenden Personen stammen. Hier kann man vergleichen, inwieweit die Allgemeinbarkeit des Netzes beim erstellten Datensatz unter Verwendung verschiedener Techniken erfolgreich war.

Im Notfall, wenn eine Person krank ist oder zum Beispiel im Brand steckt, kann die Hilfe über das Internet durch einen Sprachbefehl (Lampe Hilfe) angefordert werden. Das Gerät schickt in diesem Fall eine E-Mail mit einem vorbestimmten Text an eine vordefinierte Person oder sogar an mehrere Personen um Hilfe zu holen. Die Sprachbefehle an sich werden dabei ohne Internetverbindung erkannt.

7.4 Verwendung mehrerer ESP-Boards

In diesem Projekt bestehen die Sprachbefehle aus zwei Wörtern, die in einer bestimmten Reihenfolge auftauchen können. Dies erfordert normalerweise die Benutzung von einem rekurrenten Netz oder für bessere Leistung einem ConvLstm-Netz. Diese Netze sind aber von Tensorflow für Mikrocontroller gar nicht oder nur in kleinem Ausmaß unterstützt. Daher muss nach einer Alternative gesucht werden.

Um dieses Problem zu beheben, werden in diesem Projekt zwei ESP-Boards für die Erkennung der Sprachbefehle verwendet. Das erste Board erkennt dabei das erste Wort und das zweite Board erkennt das zweite Wort des Sprachbefehls. Zu diesem Zweck müssen die zwei Boards miteinander kommunizieren und Daten austauschen. Die Kommunikation kann hierbei über ein drahtlose Verbindung erfolgen. Diese Zwei Boards sind ESP-EYE-Board, das im zweiten Kapitel vorgestellt wurde, wobei es über ein Mikrophon verfügt und für die Erkennung der Sprachbefehle eingesetzt werden kann. Es hat aber keine GPIO-Pins und kann nur über ein drahtloses Netz mit der Außenwelt verbunden werden. Daher muss ein drittes ESP-Board, das eine kooperative Rolle zwischen den Boards spielt und eine direkte Verbindung mit der zusteuernden Last (in diesem Fall eine Lampe) über GPIOs herstellt. Die Abbildung 7-1 verdeutlicht die Kommunikation zwischen den drei Boards.

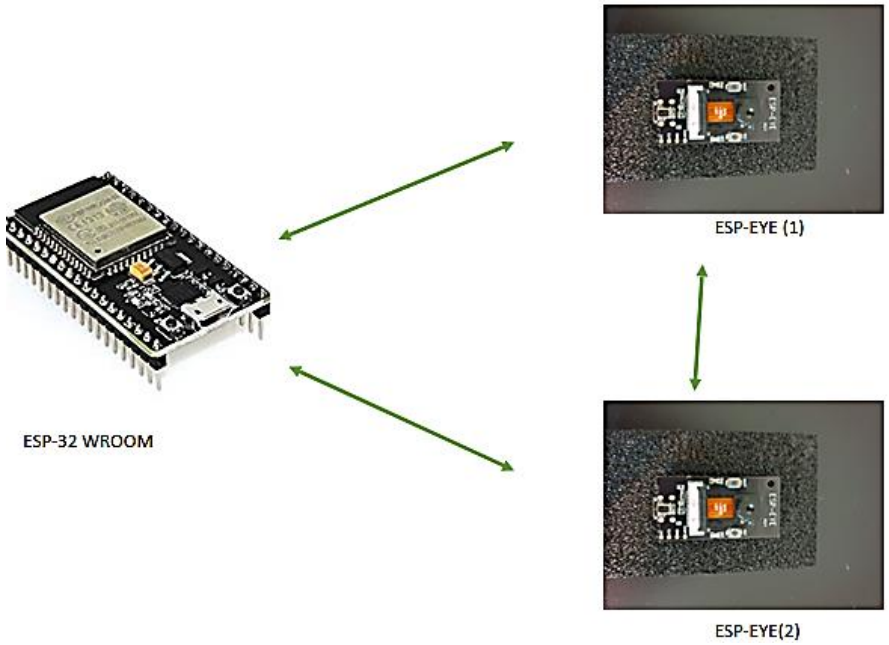


Abbildung 7-1: Die Kommunikation zwischen den ESP-Boards

7.5 ESP-NOW-Protokoll

ESP-Now ist ein von Espressif (Hersteller von Esp-32) entwickeltes Protokoll, mit dem mehrere ESP-Boards vernetzt werden können. Dieses Protokoll ist ein drahtloses Protokoll mit geringem Stromverbrauch, wobei die Verbindung zwischen den Boards in eine Richtung oder in zwei Richtungen sein kann. Zudem kann diese Verbindung zwischen beliebiger Zahl an Esp-Boards zugleich hergestellt werden. Jedes verbundene Board wird dabei als Peer bezeichnet.

Dieses Protokoll ist besonders nützlich, wenn beispielsweise Sensordaten von mehreren Boards an ein einziges Board gesendet werden oder wenn die Informationen zwischen den verschiedenen Boards ausgetauscht werden müssen, ohne dass ein Router verwendet werden muss. Die Verbindung ist dabei Peer-to-Peer. Der Sender in dieser Verbindung wird als Master und der Empfänger wird als Slave bezeichnet, wobei jedes Board Slave oder Master sein kann. Dieses Protokoll eignet sich für den Austausch kleiner Datenmengen (bis zu 250 Bytes per Frame)und wird nur auf den ESP-Boards unterstützt. Die Abbildung 7-2-a zeigt die zum Datenaustausch verwendete Data-Frame. Die Übertragung der Daten erfolgt mit einer maximalen Bitrate von 1Mbps.

MAC Header	Category Code	Organization Identifier	Random Values	Vendor Specific Content	FCS
24 bytes	1 byte	3 bytes	4 bytes	7~255 bytes	4 bytes

(a)

Element ID	Length	Organization Identifier	Type	Version	Body
1 byte	1 byte	3 bytes	1 byte	1 byte	0~250 bytes

(b)

Abbildung 7-2: Data-Frame von ESP-Now Protokoll [3]

Jedes ESP-Board hat eine Mac-Adresse, die zur Verbindung zwischen den Boards benötigt wird und eindeutig ist. Diese stellt die physische Adresse jedes Gerät dar. Ein Categorie-Code enthält die ersten drei Nummer der MAC-Adresse, während die Random-Value eine Zufällige Nummer ist, die der Sicherheit der Kommunikation dient. Vendor Specific Content enthält die zu sendenden Daten mit maximaler Größe von 250 Bytes. Die Abbildung 7-2-b zeigt den Inhalt von Vendor Spesific Content.

7.6 Programmierung des ESP-Boards

Die Programmierung des ESP-Boards erfolgt entweder in Arduino-IDE oder ESP-IDF. Arduino-IDE ermöglicht dem Programmierer eine einfachere und schnellere Programmierung des Boards und die Verwendung von den bekannten Funktionen der Arduino-IDE. Außerdem kann man unterschiedliche Mikrocontrollerboards in Arduino-IDE programmieren, ohne dass mehrere Programme und IDEs installiert werden müssen. Bevor die Arduino-IDE für die Programmierung von ESP-Boards eingesetzt werden kann, muss der Arduino-Core für ESP-Boards installiert werden.

ESP-IDF (Espressif IoT Development Framework) ist die offizielle Entwicklungs-framework für ESP-Boards, wobei die Programmierung komplexer ist und eine niedrigere Abstraktionsebene als die Arduino-IDE hat. Die Software kann entweder manuell, mit VC-Studio oder auch mit Eclipse heruntergeladen werden.

In diesem Projekt wird das Board ESP-32 NodeMCU in Arduino-IDE programmiert, während das Programm für das Board ESP-EYE mithilfe von ESP-IDF geschrieben wird. Die Abbildung 7-3 verdeutlicht, wie die Anwendungen für das ESP-Board unter Verwendung von ESP-IDF entwickelt werden.

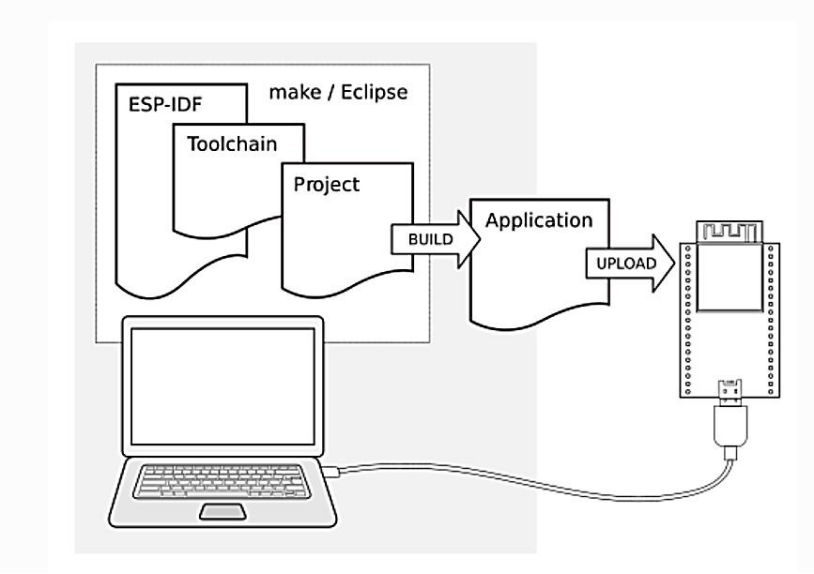


Abbildung 7-3: Entwicklung der Anwendung in ESP-IDF [24]

Für die Programmierung eines ESP-Boards wird das Board über einen USB-Kabel dem Computer angeschlossen. Einige Boards wie das ESP-EYE haben eine USB-Buchse, andere ESP-Boards benötigen einen Adapter oder Programmierer, wie es bei ESP-Cam der Fall ist. Das Betriebssystem auf dem Computer kann Linux, Windows oder Mac sein.

Bevor das Board programmiert werden kann, wird folgende Software benötigt (manuelle Installierung):

- Toolchain zur Kompilierung des Codes.
- CMake und Ninja zum Erstellen einer vollständigen Anwendung(Build Tool).
- ESP-IDF enthält die API (Softwarebibliotheken und Quellcode). [24]

7.7 Der Systementwurf

Bisher wurden die grundliegenden Ideen und Konzepte des Spracherkennungs-systems und die dafür benötigten Soft- und Hardware vorgestellt. Als nächstes werden die erworbenen Kenntnisse eingesetzt, um dieses Steuerungssystem zu realisieren.

Wie besprochen, soll das Steuerungssystem dem Benutzer ermöglichen, die Lichtfarbe, Lichtstärke und die Energieversorgung einer RGB-Lampe per Sprachbefehle zu steuern, wobei diese Sprachbefehle aus zwei Wörtern bestehen. Die Abbildung 7-4 zeigt ein prinzipielles Blockdiagramm einer per Sprache gesteuerte Lampe.

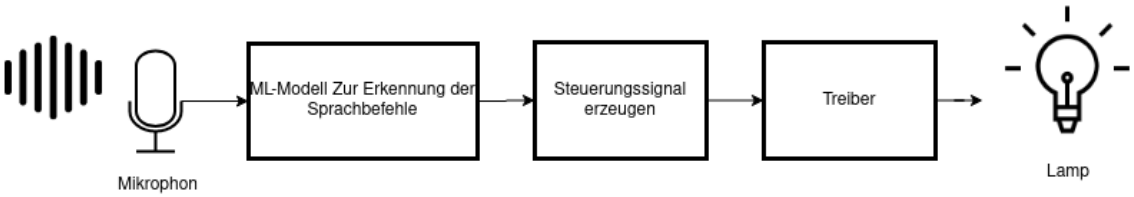


Abbildung 7-4: Prinzip eines Steuerungssystems.

Ein intelligentes Steuerungssystem, das Sprachbefehle erkennen und ein Gerät oder in diesem Fall eine Lampe steuern kann, enthält einen Sensor (Mikrophon), das die physikalischen Angaben in elektrische Signale umwandelt. Um diese Signale digital verarbeiten zu können, werden sie durch ADC abgetastet und als digitale Werte dargestellt. Aus diesen Daten werden dann Merkmale extrahiert, die ein ML-Modell erkennen kann. Diese Merkmale sind in einem Spracherkennungsmodell das Spektrogramm (sehe Kapitel 4), wobei das ML-Modell den gesprochenen Befehl erkennt. Anhand dieser erkannten Befehle werden Steuerungssignale erzeugt, die das Gerät oder die Lampe steuern, und vom Mikrocontroller erzeugt werden. Diese Signale haben kleine Leistung und können im Allgemeinfall die Last, die viel höhere Leistung erfordert, nicht direkt mit Strom versorgen. Deshalb werden vor der Last Treiber eingesetzt. Dieser Treiber hat im Allgemeinen einen sehr großen Eingangsimpedanz und einen kleinen Ausgangsimpedanz. In anderen Worten stellt dieser Treiber keine große oder sogar vernachlässigbare Last für den Mikrocontroller dar und kann gleichzeitig die Last (in diesem Fall eine Lampe) mit ausreichender Leistung versorgen.

Bei dem in Abbildung 7-4 beschriebenen Steuerungssystem müssen die verschiedenen Komponenten in der Nähe von dem zusteuernden Gerät und gleichzeitig innerhalb der Reichweite der Schallwellen liegen. In vielen Fällen trennt aber zwischen dem Sensor (das Mikrophon) und dem zu steuernden Gerät (Lampe oder Heizungsanlage ...) einen großen Abstand. Daher wäre es ideal, wenn das System so entworfen wird, dass die Bestandteile des Systems in zwei Teilsysteme aufgeteilt werden, wobei ein Teilsystem tragbar ist und das Mikrophon sowie das Erkennungsmodell enthält. Hierfür wird eine drahtlose Verbindung zwischen den beiden Teilen des Systems benötigt. Die Abbildung 7-5 veranschaulicht die Funktionsweise des Systems in diesem Fall.

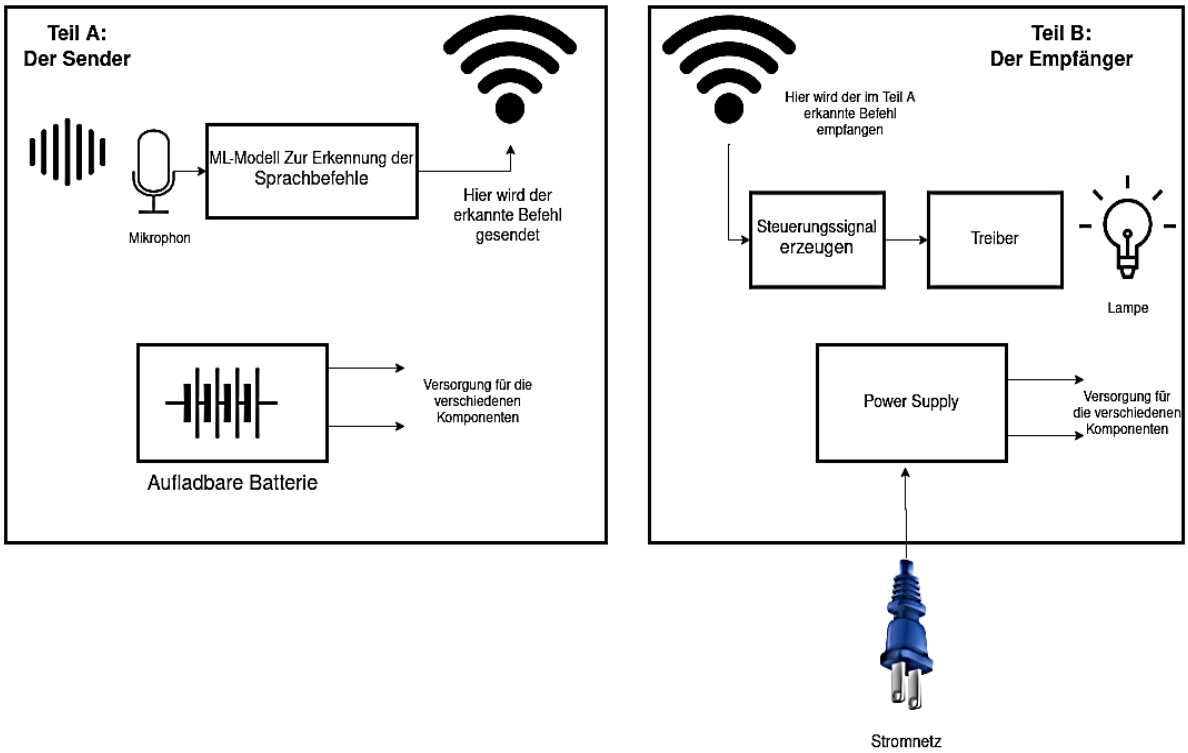


Abbildung 7-5: Die Kommunikation zwischen Teil-A und Teil-B

Was in Abbildung 7-5 mit Sender und Empfänger gemeint ist, ist nicht der Datenverkehr, sondern die Richtung der erkannten Sprachbefehle. Denn es gibt in beiden Richtungen einen Austausch der Signale. Der Teil-A ist hierbei tragbar und muss mit Strom versorgt werden. Die Stromquelle kann in diesem Fall eine aufladbare Batterie (Lithium-Batterie) oder irgendeine tragbare Stromquelle sein. Nachfolgend wird auf die Einzelteile eingegangen.

7.8 Der Sender (Teil A)

in Abbildung 7-5 ist ersichtlich, dass die Erkennung der Sprachbefehle im Teil A stattfindet, wo sich die Zwei ESP-EYE-Boards befinden. Hier werden aus den im Abschnitt 7.4 erwähnten Gründen zwei Maschine-Lernen-Modelle auf zwei ESP-EYE-Boards trainiert. Das erste Modell erkennt das erste Wort (das Schlüsselwort) und das zweite Modell erkennt das zweite Wort des Befehles. Hier stellt sich die Frage, warum müssen die Sprachbefehle aus zwei Wörtern bestehen? Die intuitive Antwort ist, dass das System in diesem Fall einzelne Wörter aus dem Sprachfluss in der Umgebung als befehle interpretieren könnte, die nicht unbedingt Sprachbefehle darstellen.

Für die Kommunikation zwischen dem Sender und dem Empfänger wird das im Abschnitt 7.5 beschriebene ESP-NOW-Protokoll verwendet. Diese Kommunikations-methode ist für den Austausch kleiner Datenmengen effizient und sicher und bietet Datenübertragung in Echtzeit, was in diesem Steuerungssystem notwendig ist. Außerdem reicht der Umfang der Datenübertragung bis zu 200 Meter, was bedeutet, dass der Benutzer den tragbaren Teil innerhalb einer großen Wohnung beispielsweise mitnehmen kann.

Die Versorgung mit Strom für den Sender wird durch eine aufladbare Lithium-Batterie (Abbildung 7-6) gewährleistet, die mithilfe eines Lademoduls aufgeladen wird. Wenn das Lademodul dem Stromnetz über einen Adapter angeschlossen wird, geht es automatisch in den Zustand ‘Aufladen’ über. Wenn das Lademodul hingegen von dem Stromnetz entfernt wird, wird die benötigte Energie an die beiden ESP-EYE-Boards weiter geliefert. Das Board ESP-EYE hat eigentlich keine PINs. Daher erfolgt die Stromversorgung über den auf dem ESP-EYE-Board befindlichen USB-Buchse.

Auf dem ersten Board wird erkannt, ob das gesprochene Wort der Ersten Klassengruppe gehört. Diese Gruppe enthält die Klassen: Ruhe, Unbekanntes Wort, Lampe, Stärke und Farbe(Tabelle 7-2). Andererseits wird auf dem zweiten Board erkannt, ob ein der Wörter Ruhe, unbekanntes Wort, Ein, Aus, Up, Down, One, Two, Three, Four und Hilfe gesprochen wurde (Tabelle 7-3).

Die Abbildung 7-6 zeigt die Bestandteile des Senders. Hierbei werden die Spracherkennungsmodelle auf den zwei Boards in der Programmschleife ausgeführt, wobei in jedem Durchgang der Schleife das Spektrogramm aus den aufgenommenen Audiodaten extrahiert und durch das Modell geführt wird. Was sich hier von dem Spracherkennungsmodell im Kapitel 4unterscheidet, ist, dass die Boards in diesem Fall miteinander und mit dem Empfänger kommunizieren müssen, wobei die erkannten Sprachbefehle an Teil-B gesendet werden und wird nachfolgend behandelt.

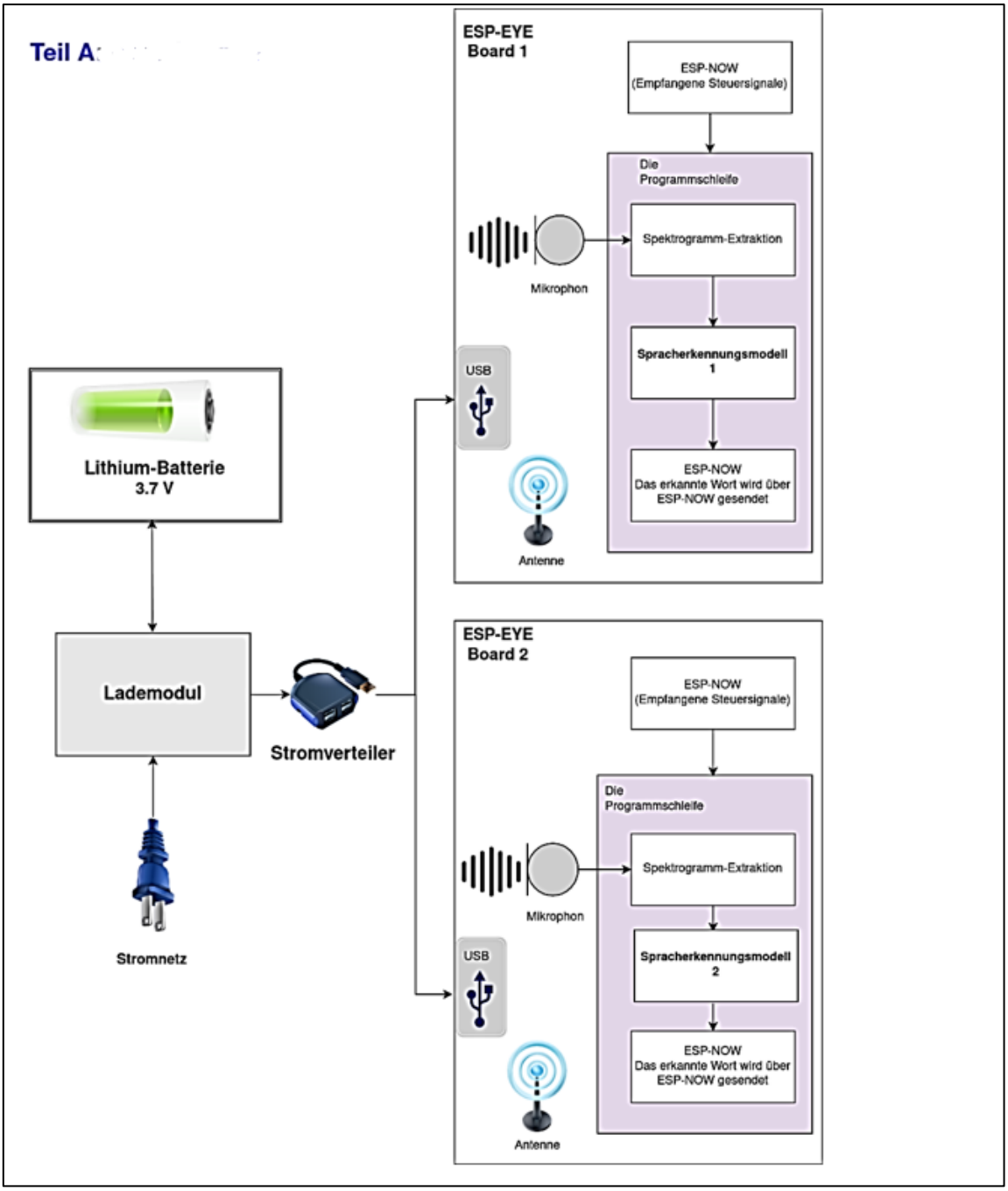


Abbildung 7-6: Der Teil-A (der Sender)

7.8.1 ESP-EYE-Boards zur Erkennung der Sprachbefehle

Dieses Board wurde für KI- und IoT-Anwendungen entwickelt und verfügt über Mikroprozessor des Typs Tensilica LX6 mit zwei Kernen und einen 8MByte PSRAM. Außerdem befindet sich auf dem Board einen 4MByte Flash-Speicher und ein Mikrophon zur Audioaufnahme. Zur Kommunikation mit der Außenwelt hat das Board Wi-Fi- und Bluetooth-Komponenten.

Die Abbildung 7-7 zeigt die auf dem ESP-EYE-Board befindlichen Komponenten. es ist zu sehen, dass das Board über 3 On-Board-Buttons oder Tasten verfügt. Diese Tasten sind eine Reset-Taste, eine Boot-Taste und eine programmierbare Taste für die Bildaufnahme beispielsweise.

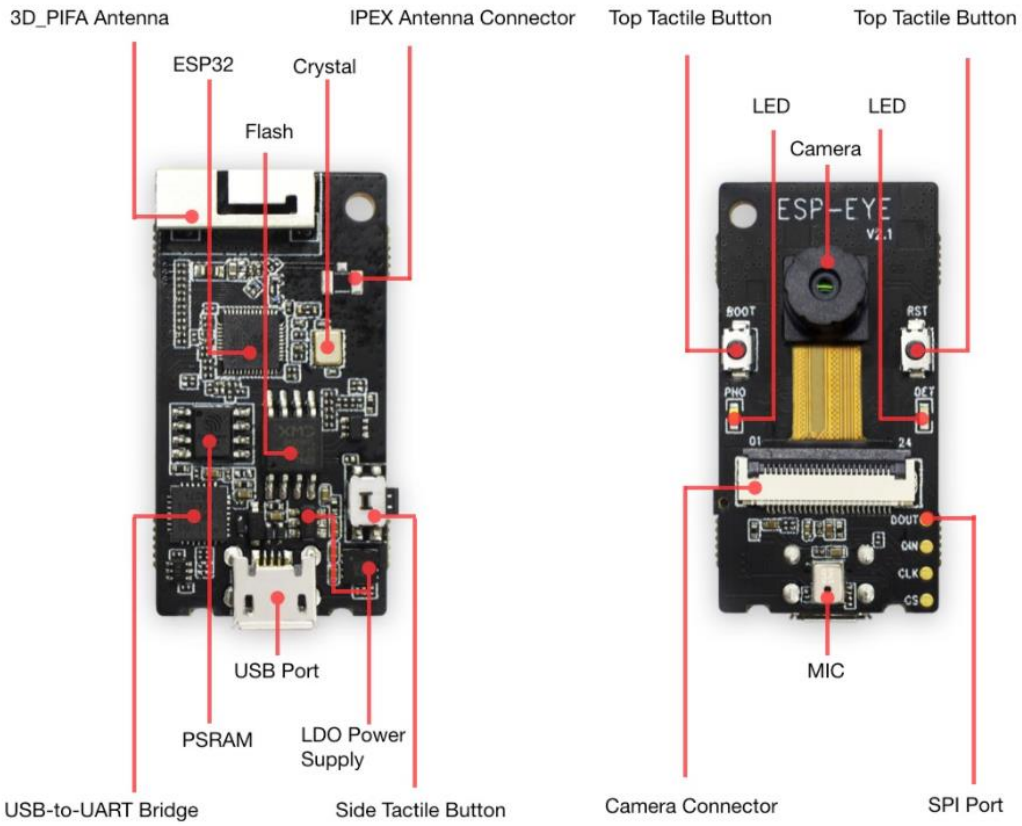


Abbildung 7-7:ESP-EYE-Board und die darauf befindlichen Komponenten [32]

7.8.2 Die Lithium-Batterie

Die im Teilsystem-A verwendete Batterie ist die TS 18650 Lithium-Batterie. Diese Zylindrische Zelle setzt sich aus mehreren Zellen zusammen und hat gute Eigenschaften. Die Zellenkapazität gemessen in Milliamperestunden beträgt 3500mAh und die maximale Entladungsdauer liegt bei 30A. Aufgrund ihrer grundsätzlich hohen Energiedichte, minimaler Selbstentladung und geringem Memory-Effekt gehört diese Zelle zu den wichtigsten Akkumulatoren für tragbare elektronische Geräte. Beim schnellen Aufladen liegt die Temperatur zwischen 5 und 45 C. Solche Zellen werden in vielen tragbaren elektronischen Geräten wie Laptops eingesetzt. Abbildung 7-8 zeigt die TS 18650 Zelle.

Abbildung 7-8: Die Lithium-Zelle TS 18650.

7.8.3 Das Gehäuse für Teil-A

Für das Teilsystem A (den Sender), wird ein passendes Gehäuse benötigt, das bequem tragbar ist. Beim Aufladen der Batterie und Betrieb von ESP-Boards wird Wärme erzeugt. Daher ist es wichtig, dass ein möglichst belüftetes Gehäuse verwendet wird. Außerdem müssen natürlich alle Komponenten und Module in es hineinpassen. Das fertige Gehäuse, das diese Anforderungen erfüllt, ist auf Amazon zu finden. Die folgende Abbildung 7-9 zeigt das Gehäuse für Teilsystem A und Abbildung 7-10 zeigt die im Gehäuse enthaltenen Komponenten.

Abbildung 7-9: Das Gehäuse für Teilsystem A.

Für das Aufladen der Batterie wurde ein USB-Buchse eingebaut. Das eingebaute Led zeigt dabei den Zustand der Batterie und des Geräts an, wobei es mit drei Lichtfarben drei Zustände signalisiert. Diese Zustände sind Ladezustand, Betrieb und Leer (wenn die Batterie leer ist). Außerdem kann das Gerät durch einen eingebauten Schalter aus- und eingeschaltet werden, damit die Energie der Batterie gespart wird, wenn das Gerät beispielweise nicht benutzt wird.

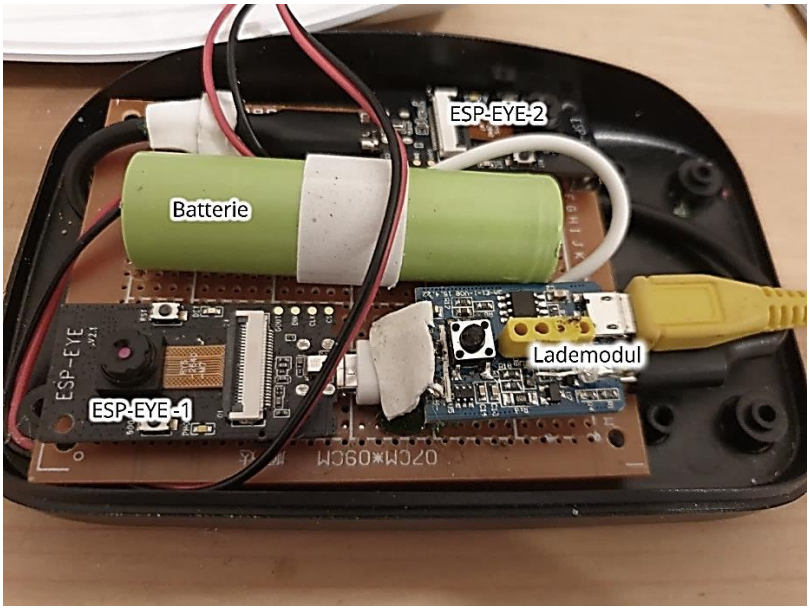


Abbildung 7-10: Im Teil-A enthaltenen Komponenten

7.9 Erstellung und Training der Modelle

Wie bereits erwähnt, muss das erste Modell eine Gruppe und das zweite Modell auf dem zweiten ESP-EYE-Board eine andere gruppe von Wörtern erkennen. Diese zwei Klassengruppen sind in Tabelle 7-2 und

Tabelle 7-3 anschaulich dargestellt.

Tabelle 7-2: Die Wörter in Klassengruppe 1

Klassengruppe 1				
Klasse 1	Klasse 2	Klasse 3	Klasse 4	Klasse 5
Ruhe	Unbekannt	Lampe	Stärke	Farbe

Tabelle 7-3: Die Wörter in Klassengruppe 2

Klassengruppe 2										
Klasse 1	Klasse 2	Klasse 3	Klasse 4	Klasse 5	Klasse 6	Klasse 7	Klasse 8	Klasse 9	Klasse 10	Klasse 11
Ruhe	Unbekannt	Ein	Aus	Up	Down	One	Two	Three	Four	Hilfe

Im Kapitel 4wurde ein Spracherkennungsmodell gefunden, das eine gute Accuracy-Rate aufweist und gleichzeitig gut auf dem Mikrocontroller funktioniert. Dieses Modell wird (mit Kleinen Änderungen) für die Erkennung der Klassengruppe 2 eingesetzt. Für die Erkennung von Klassengruppe 1 wird ein neues Modell erstellt, das in diesem Fall bessere Genauigkeit aufweist. In Tabelle 7-4 sind die Größe beider Erkennungs-modelle mit und ohne Quantisierung aufgelistet, während die Abbildung 7-12 die quantisierten Modelle unter Verwendung von Netron zeigt.

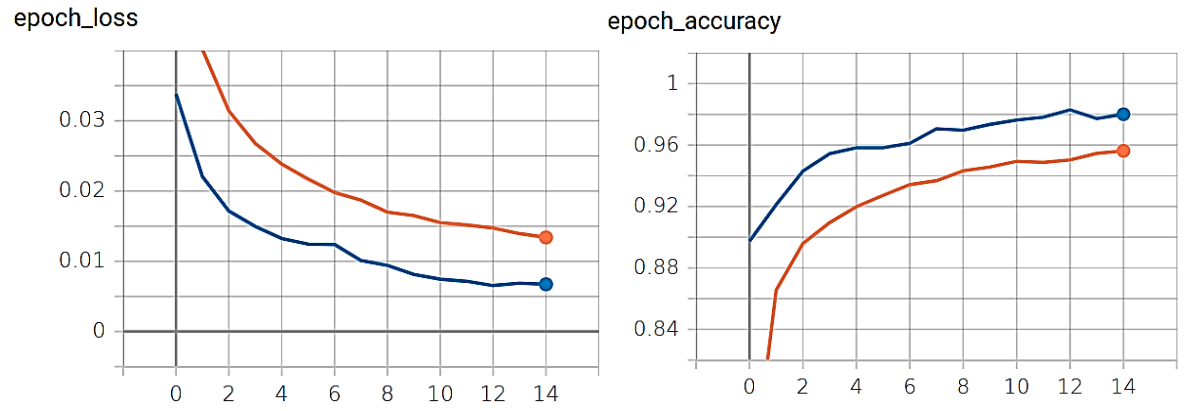
Tabelle 7-4:Die Größe der Erkennungsmodelle mit und Ohne Quantisierung.

Modell	Quantisierung	Modellgröße
Modell 1	Ohne Quantisierung(Float32)	123.2kB
	Mit Quantisierung (int8)	37.2 kB
Modell 2	Ohne Quantisierung(Float32)	114.4 kB
	Mit Quantisierung (int8)	34.2 kB

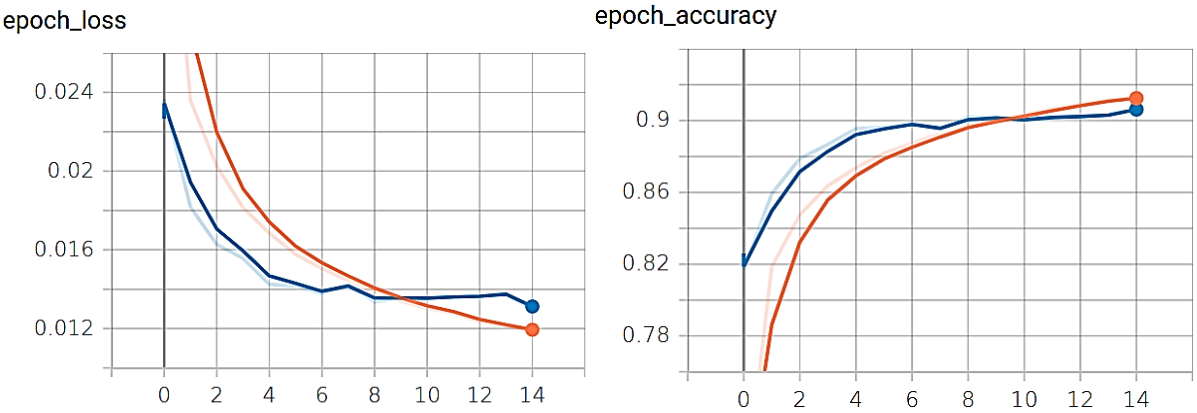
Nach wie vor wird Tensorboard verwendet, um verschiedene Metriken zu visualisieren. Die Abbildung 7-11-a und Abbildung 7-11-b zeigen die Accuaracy-Rate und Loss-Funktion beider Modelle unter Verwendung von Tensorboard. Der Tabelle 7-5 kann entnommen werden, dass sich die Modelle voneinander stark nach der Accuracy-Rate unterscheiden. Dies kann darauf zurückgeführt werden, dass der Anteil der synthetischen Daten, auf die das erste Modell-1 trainiert wurde, ist viel größer, Wobei diese synthetischen Daten viel besser klassifiziert werden können als die natürlichen Aufnahmen, die eindringenden Geräusche enthalten und sich sehr voneinander unterscheiden. Andererseits hat die Anzahl der Klassen auch einen Einfluss auf die Accuracy-Rate, wobei das erste Modell auf nur Fünf Klassen trainiert wird, während das zweite Modell Elf Klassen erkennen kann.

Tabelle 7-5: Accuracy-Rate mit und ohne Quantisierung.

Das Modell	Quantisierung	Accuracy-Rate
Modell 1	Float 32	98.575499%
	int8	98.670465%
Modell 2	Float 32	91.110278%
	Int8	91.110278%



(a) Modell 1



(b) Modell 2

Abbildung 7-11: die Accuracy und Loss während des Trainierens.

Die rote Kurve in Abbildung 7-11 ist die Trainingskurve, während die blaue Kurve die Validationskurve darstellt. Hier kann gemerkt werden, das es Unterschied zwischen den beiden Kurven gibt. Der Grund dafür ist, dass während des Trainings ein Teil der Gewichtungen durch die Regularisierung-Schichten abgeschaltet wird, während alle Gewichtungen an der Berechnung der Ausgabe bei der Validation beteiligen.

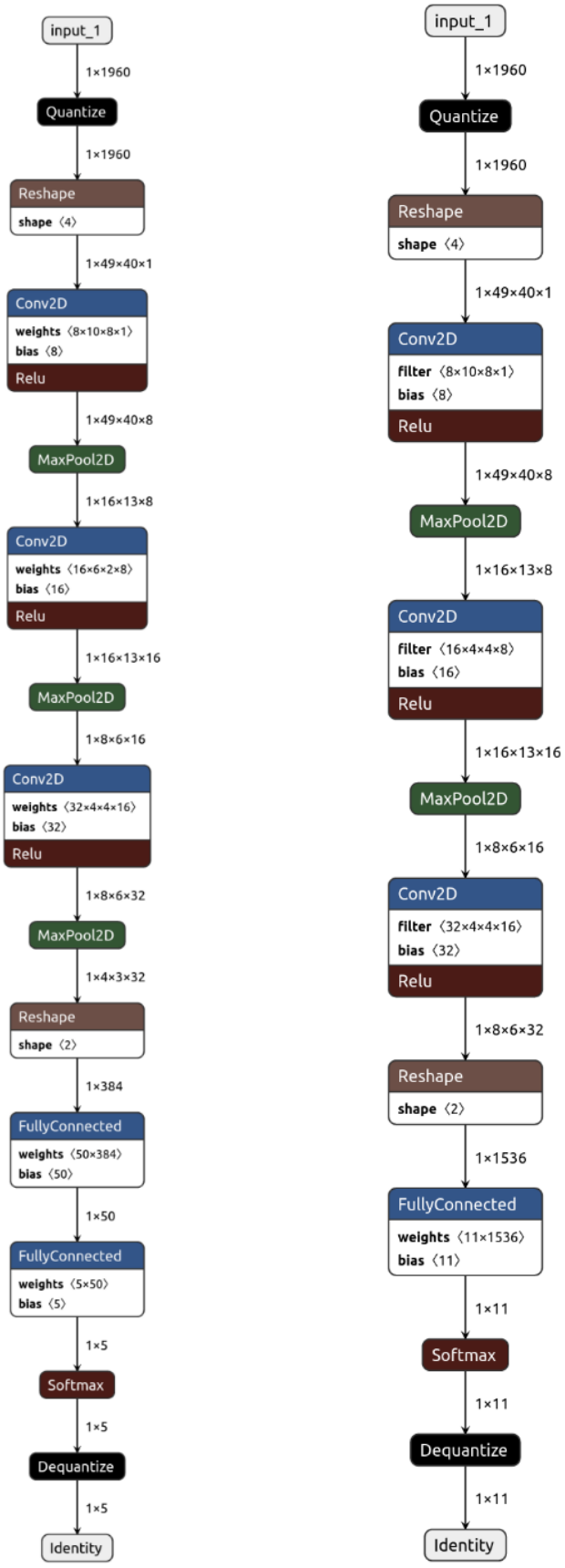
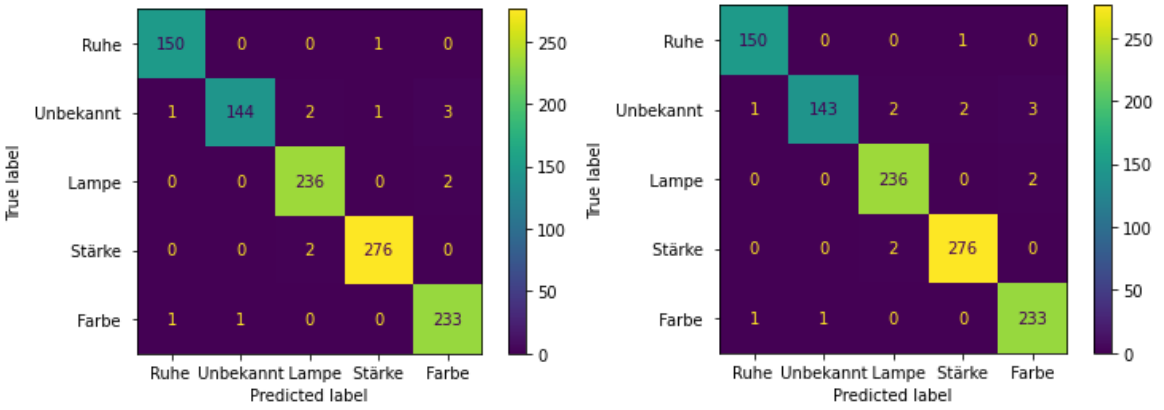


Abbildung 7-12: Die Erkennungsmodelle(Links das Modell 1 und rechts das Modell 2)

Zum Anallysieren von ML-Modellen kann die Konfusionsmatrix bedeutsame Informationen liefern und detailliert zeigen, wie viele Datenpunkte jeder Klasse richtig und falsch klassifiziert wurden. Dies ist beim Einsatz von Quantisierung besonders wichtig, weil es besagt, welche Änderungen die Quantisierung verursacht hat. Die Abbildung 7-13 und Abbildung 7-14 zeigen die Konfusionsmatrix von beiden Modellen mit und ohne Quantisierung.



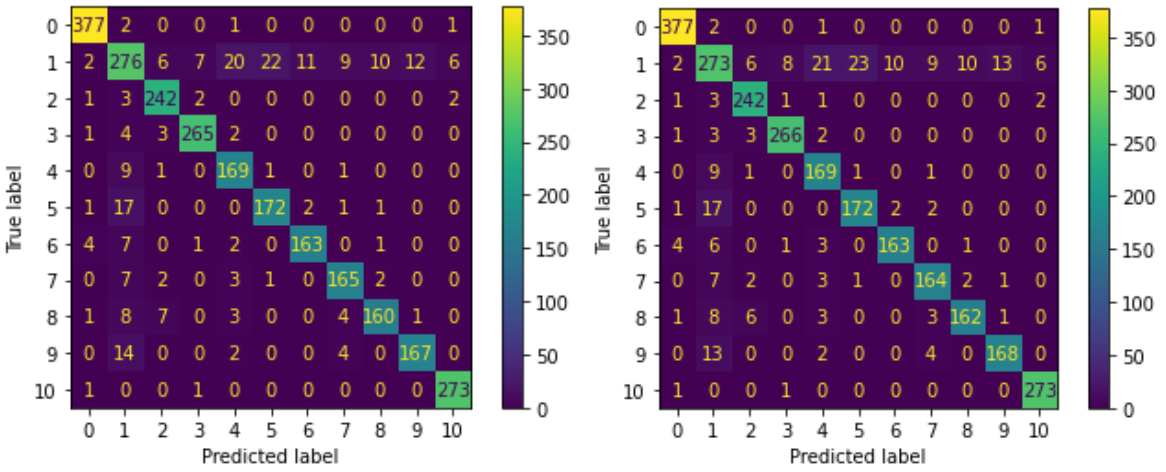
(a) int8 (Mit Quantisierung) **(b) Float32-Modell (ohne Quantisierung)**

Abbildung 7-13:Konfusionsmatrix für Modell 1

Überreichend ist hierbei die leicht erhöhte Accuracy-Rate nach der Quantisierung. Das widerspricht die logische Tatsache, dass die Modellgewichtungen nach der Quantisierung mit geringerer Auflösung dargestellt werden und damit die Genauigkeitsrate weniger wird. Das kann als Zufall betrachtet werden. Denn die Änderungen wegen der Quantisierung hatten in diesem Fall einen positiven Einfluss, sodass ein Datenpunkt (Klasse unbekannt) zufälligerweise mehr richtig klassifiziert wurde, das bedeutet aber nicht, dass das quantisierte Modell besser klassifiziert.

Der Tabelle 7-5 zu entnehmen, dass die Accuracy-Rate beim Modell 2 mit Quantisierung wie die Accuracy-Rate ohne Quantisierung bleibt, obwohl Änderungen auf der Konfusionsmatrix (Abbildung 7-14) aufgetaucht sind. Der Grund dafür ist ,dass der Nettoeffekt dieser Änderungen wegen der Quantisierung in diesem Fall nichtig bleibt. Es muss hierbei beachtet werden, dass die Ergebnisse von dem untersuchten Modell abhängen.

Nachdem die Modelle für die ESP-Boards trainiert worden sind, müssen sie in C-Arrays konvertiert und in das C-Programm integriert werden.



(a) int8 (Mit Quantisierung) **(b) Float32-Modell (ohne Quantisierung)**

Abbildung 7-14: Konfusionsmatrix für das Modell 2

Die Extraktion der Spektrogramme und das Einspeisen dieser Spektrogramme in das Modell bleiben wie im C-Programm im Kapitel 2 unverändert. Was sich hier unterscheidet, ist die Verwendung vom ESP-Now-Protokoll und die Verwaltung von dem Datenaustausch zwischen den verschiedenen Boards. Die Abbildung 7-15 gibt einen Überblick darüber, wie die beiden Modelle im Sender mit dem Empfänger oder Teil-B kommunizieren.

In Abbildung 7-15 ist zu sehen, dass die Inferenz der beiden Modelle innerhalb einer unendlichen Schleife stattfindet. Zuerst wird das erste Modell (Modell 1) ausgeführt und das Ergebnis zum Teil B (dem Empfänger) per ESP-NOW-Protokoll gesendet. Im Teil B wird geprüft, ob es sich um ein Schlüsselwort bei dem empfangenen Wort handelt. Falls Ja schickt der Empfänger ein Aktivierungssignal an das Board ESP-2. In diesem Moment wird das Modell 2 für eine bestimmte Zeit aktiviert und ausgeführt , wobei die durch dieses Modell erkannten Wörter zum Empfänger ebenfalls zurückgeschickt, um zu prüfen, ob das zweite Wort eines Befehls gesprochen wurde.

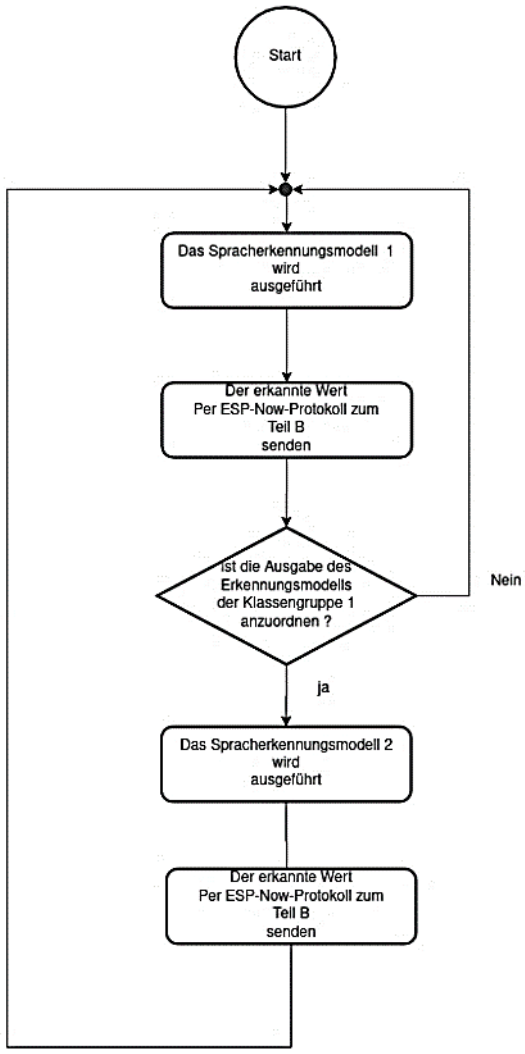


Abbildung 7-15: Die Kommunikation über ESP-NOW-Protokoll

7.10 Der Empfänger (Teil B)

Nachdem der Sender, der für die Erkennung der Befehle zuständig ist, entworfen und trainiert wurde, wird nachfolgend die Funktionsweise des Empfängers (Teil B) erläutert.

Bevor auf die einzelnen Komponenten eingegangen wird, empfehlt es sich, das grobe Blockbild des Empfängers anzusehen (Abbildung 7-16). Das ESP-Board im Teil B empfängt die im Sender erkannten Wörter über ESP-NOW-Protokoll und bildet aus diesen Wörtern Befehle. Um die Interaktion zwischen dem Gerät und dem Benutzer zu erleichtern, werden ein LCD und mehrere Leds verwendet, womit die verschiedenen Zustände des Geräts und die erkannten Sprachbefehle angezeigt werden. Zwischen der Lampe und dem Mikrocontroller wird ein Treiber geschaltet, der die Mikrocontrollersignale aufnimmt und entsprechend die Lampe mit regelmäßigen

Gleichstrom versorgt. Der Benutzer kann außerdem den Empfänger mit dem Internet verbinden, um eine Textnachricht im Notfall zu schicken, indem ein WLAN-Netz konfiguriert wird. Hier muss erwähnt werden, dass das ESP-Board über einen USB-Anschluss programmiert wird, daher wurde eine USB-Buchse eingebaut.

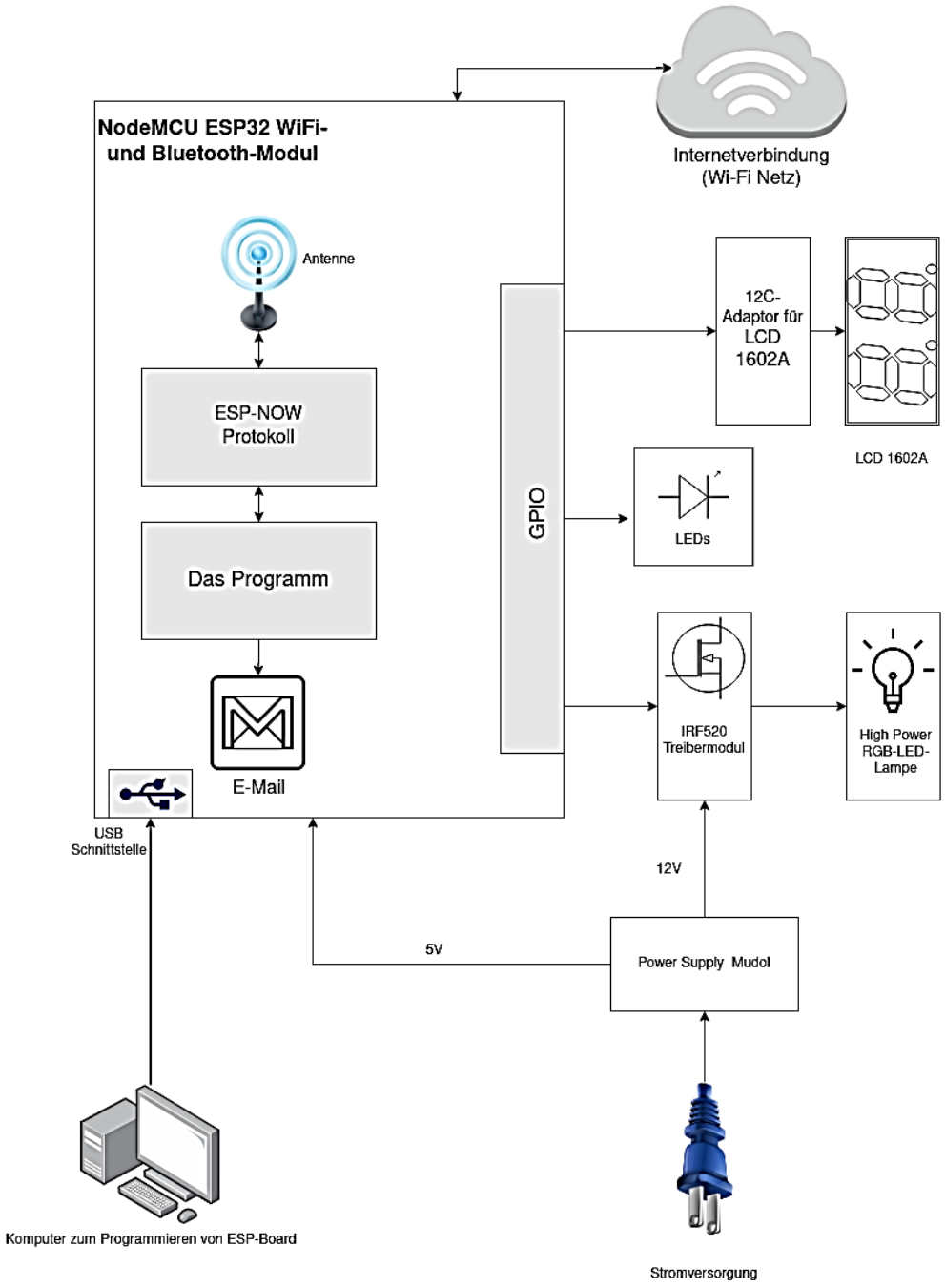


Abbildung 7-16: Blockbild des Empfängers

7.10.1 NodeMCU ESP WiFi und Bluetooth Board

Dieses Board ist ebenfalls ein auf ESP-32 basiertes mächtiges und günstiges Board, das ebenfalls über WiFi/Bluetooth Module verfügt und kann somit über eine Internetverbindung oder ESP-NOW-Verbindung mit der Außenwelt kommunizieren. Mit anderen Modulen(wie LCD und Leds) kann das Board außerdem über GPIOs verbunden werden.

Das Board wird entweder mit 5v oder 3.3v betrieben werden und verfügt über einen USB-to-UART-Konverter, sodass es direkt mit dem Rechner verbunden werden kann. Für das Empfangen und Senden von Hochfrequenzsignalen befindet sich auf dem Board eine On-Board-PCB-Antenne. Die Abbildung 7-17 zeigt die Pin-Belegung des Boards.

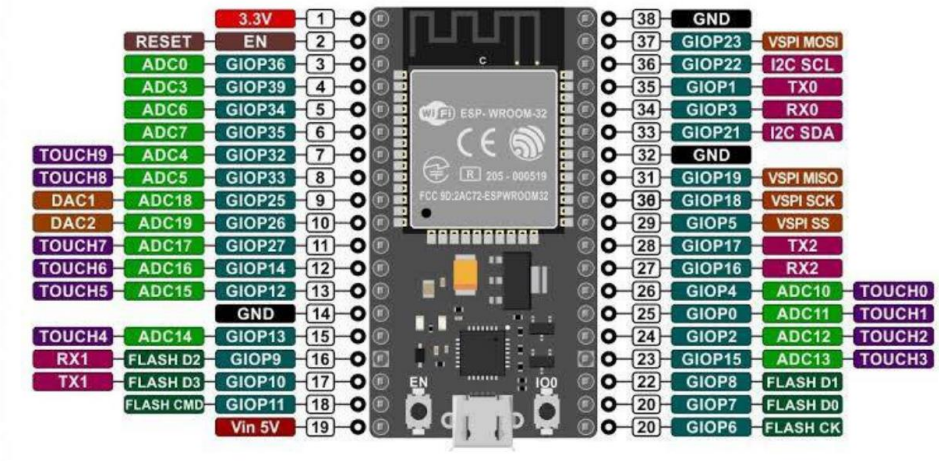


Abbildung 7-17: Das im Empfänger verwendete NodeMCU ESP32-Board

7.10.2 Das LCD 1602 und der I2C-Adapter

Dieses LCD(Abbildung 7-18) kann zwei Zeilen mit je 16 Symbolen anzeigen (Daher kommt die Nummer 1602). Es wird in den Empfänger integriert, um anzuzeigen, welcher Befehl erkannt wird und was zu tun ist. Dies erleichtert erheblich die Interaktion zwischen dem Benutzer und dem Gerät. Es wird über einen I2C-Adapter mit dem ESP-Board verbunden. Dieser Adapter ermöglicht dem ESP-Board, das LCD über den I2C-Bus anzusprechen, wobei I2C-Bus ein von Phillips konzipierter Datenbus und wird für die interne Kommunikation zwischen den Bestandteilen eines elektronischen Systems entwickelt.

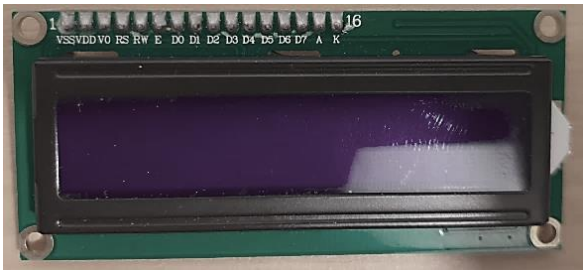


Abbildung 7-18: Das verwendete LCD 1602.

7.10.3 Der Treiber

Der Treiber ist eine Schaltung, die einen sehr großen Eingangsimpedanz und einen sehr kleinen Ausgangsimpedanz besitzen und wird eingesetzt, um die Last mit ausreichender Energie zu versorgen. Eine einfache Form des Treibers ist ein High-Power Mosfet Transistor wie beispielsweise der Transistor IRF520. Die Abbildung 7-19 zeigt die Schaltung des Treibers.

Das ESP32-Board erzeugt das Steuerungssignal, das in das Gate des Transistors eingespeist wird. Dieses schwache Signal kann den Transistor zur Sättigung oder Sperrung treiben, je nachdem welchen Wert es hat. Die Last in diesem Fall ist eine RGB-Lampe, die mit niedriger Spannung (weniger als 12v) betrieben werden kann.

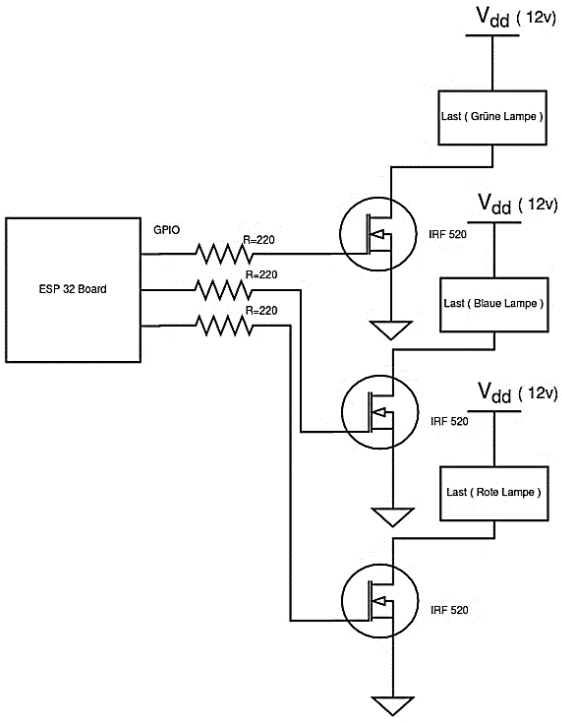


Abbildung 7-19:Die Treiberschaltung.

Wie es zu sehen ist, werden drei Mosfet-Transistoren im Empfänger benötigt, wobei jeder davon die Lampe mit einer Lichtfarbe mit Strom versorgt.

7.10.4 Das Power-Supply-Modul

Dieses Modul(Abbildung 7-20) ist ein Spannungsregler, der das ESP-Board mit regelmassiger Spannung und Strom versorgt. Der Eingang kann zwischen 6v und 12v liegen, während der Ausgang entweder 3.3v oder 5v ausgewählt werden kann. In diesem Projekt wird der Mikrocontroller mit 5v versorgt.

Abbildung 7-20: Das Power-Sypply-Modul.

7.10.5 Die RGB-Lampe

Die gesteuerte Lampe ist ein RGB Hochleistung-Led , der mit verschiedenen Farben leuchten kann, je nachdem wie die drei Farben (Grün, Rot , Blau) gemischt werden (wie viele Prozentanteil jede Farbe hat). Die Stromversorgung ist dabei ein Gleichstrom und die Leistungsaufnahme beträgt 3 W. Die folgende Abbildung 7-21 zeigt den Led.

Abbildung 7-21: Der RGB-Led (die Lampe)

Da der Led mit großer Leistung betrieben werden kann, ist zur Abkühlung auf einem Aluminiumplatine aufgelötet und in einer Schutzhülle eingebaut, wie in Abbildung 7-21 (rechts) zu sehen ist.

7.10.6 Das Gehäuse

Das Gehäuse, das für den Empfänger verwendet wird, muss die verschiedenen Komponenten enthalten darunter das LCD. Daher muss es ein Fenster mit bestimmter Größe haben. Solches Gehäuse ist auf Amazon zu finden und eignet sich perfekt für den Empfänger. Es ist aus Kunststoff Plastik Boxen und wurde als Anschlussdose für elektronische Anwendungen hergestellt. Die Abbildung 7-22 zeigt das Gehäuse.

Abbildung 7-22: Das Gehäuse für den Empfänger(Teil B).

7.10.7 Die Funktionsweise vom Empfänger

In dem Empfänger findet die Interpretation der befehle statt, die im Sender erkannt werden und wird entsprechend die RGB-Lampe gesteuert, wie bereits erwähnt. Die Abbildung 7-23 zeigt die Vorderseite des Empfängers.

Das gelbe Led 1 leuchtet, wenn ein Schlüsselwort der Klassengruppe 1(Tabelle 7-2) erkannt wird, was bedeutet, dass das Gerät auf das zweite Wort des Befehls wartet. Das rote Led 2 leuchtet, wenn Ruhe in der Umgebung herrscht, während das weiße Led 3 einen unverstandenen Befehl oder Geräusche signalisiert. Das vierte Led 4 ist ein RGB-Led, das genau den Zustand des Ausgangs annimmt und leuchtet ähnlich wie die RGB-Lampe, die gesteuert wird.

Wie es zu sehen ist, wird der Empfänger durch einen eingebauten Schalter am Gehäuse ein- und ausgeschaltet. Beim ersten Einschalten muss man einmalig einen Bin-Code sagen, bevor man das Gerät benutzen kann. Dieser Bin-Code ist dreistellige Zahl und dient der Sicherheit des Geräts.

Abbildung 7-23: Die Vorderseite des Empfängers.

Ein USB-Anschluss ist zudem erforderlich, da das Gerät mit dem Computer verbunden wird, wenn es programmiert werden muss. Die Energie wird in diesem Fall dem Gerät durch einen 12v-Anschluss zugeführt.

7.10.8 Das C-Programm

Dieser Abschnitt befasst sich mit dem C-Programm, das die erforderlichen Aufgaben erledigen kann. Das Flussdiagramm in Abbildung 7-24 gibt einen allgemeinen Überblick über die Funktionsweise des Programms. Es besteht aus einer Schleife und einem Interrupt zum Empfangen von Daten über ESP-NOW-Protokoll.

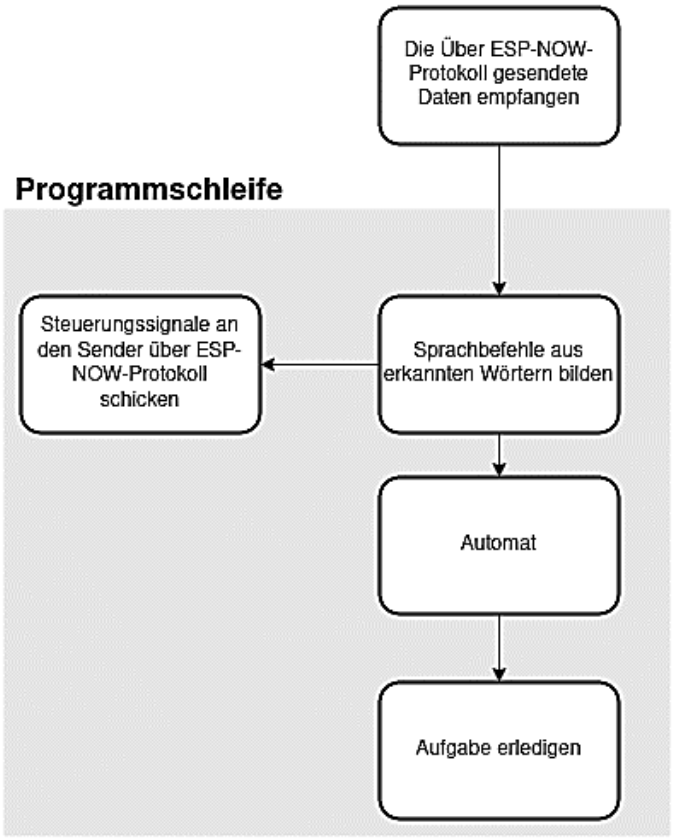


Abbildung 7-24: Flussdiagramm beschreibt das Programm auf dem Empfänger

Die Inferenzzeit des Empfängers ist hierbei viel kürzer als die des Senders, da es kein ML-Modell ausgeführt wird, das längere Zeit benötigt. Deshalb wird die Schleife des Empfängerprogramms mehrmals innerhalb einer Modellinferenz (im Sender) ausgeführt. Am Anfang der Programmschleife wird überprüft, ob Daten vom Sender angekommen sind, dann werden Anhand dieser Daten Befehle gebildet. Zu dieser Zeit wird ein Steuerungssignal erzeugt, das zum Sender gesendet wird. Zugleich werden die gebildeten Befehle einem Automaten zugeführt. Dieser Automat bestimmt, in welchem Zustand sich das Programm befindet, wobei in jedem Zustand eine Aufgabe erledigt wird. Die folgende Abbildung 7-25 zeigt den Automat.

Die Zustandsmaschine hat Neun Zustände. In jedem Zustand wird eine Funktion ausgeführt und eine Nachricht (Message) auf dem LCD angezeigt. Der folgende Code (Listing 7-1) ist ein Ausschnitt aus dem C-Programm und verdeutlicht die Befehle, die der Empfänger verstehen kann, und die Messages, die auf dem Bildschirm in jedem Zustand angezeigt werden. Diese Messages und die Befehlskombinationen kann man überarbeiten und beliebig umformen. Der Sicherheitscode oder Bin-Code, Wi-Fi Netz Konfigurationen sowie die Email-Adressen können im Programm definiert werden.

Listing 7-1: Ausschnitt aus dem C-Programm

```
1. // Die Klassennamen für das Modell 2 auf dem ESP-EYE-2
2. char* gruppe_2[] = { "Ruhe"  ,"unbekannt" , "ein" ,"aus"
   ,"up","down","one","two"  ,"three" , "four" , "hilfe"  } ;
3. // die Klassennamen für das Modell 1 auf dem ESP-EYE-1 (die
   Schlüsselwörter )
4. char* gruppe_1[] = { "Ruhe"  ,"unbekannt" , "lampe" , "staerke"
   ,"farbe"   } ;
5.
6. //die Befehlkompinationen, die das Gerät verstehen
7. const int8_t befehl_0 [2] = {0,0} ; //kein_befehl
8. const int8_t befehl_1[2] = { 2 ,2 } ; // lampe ein
9. const int8_t befehl_2[2] = { 2,3} ; // lampe aus
10.  const int8_t befehl_3 [2] = { 3,4 } ; //staerke up
11.  const int8_t befehl_4 [2] = { 3,5} ; //staerke  down
12.  const int8_t befehl_5 [2] = { 4,6} ; // farbe one
13.  const int8_t befehl_6 [2] = { 4,7 } ; //farbe two
14.  const int8_t befehl_7[2] = { 4,8 } ; //farbe three
15.  const int8_t befehl_8[2] = {4,9 } ; //farbe four
16.  const int8_t befehl_9 [2] = { 2, 10 } ; // Lampe Hilfe
17.
18.
19.  // Nachrichten zum Anzeigen auf dem Bildschirm
20.  String hallo_message  = " Hallo Benutzer " ;
21.  String  ich_message  = "hier die lampe HAW  ";
22.  String message_0 ="sagen Sie einen " ;
23.  String message_0_1 = "Sprachbefehl" ;
24.  String message_1 = " lampe  ein" ;
25.  String message_2 = " lampe  aus" ;
26.  String message_9 = "staerke  up " ;
27.  String message_10 = "staerke  down " ;
28.  String message_11 = "gruenes licht  " ;
29.  String message_12 = "rotes  licht " ;
30.  String message_13 = "blaues licht  " ;
31.  String message_14 = "weisses licht two " ;
32.  String message_15 = "Hilfe fordern " ;
33.  String message_17 = " E-mail gesendet " ;
34.  String message_19 = "Peering Fehler " ;
35.  String absender = " from Lampe HAW " ;
36.  String  Email_Inhalt= " ich stecke in einem Problem , ich
   brauche deine Hilfe.....Anschrift: " ;
37.  // hier wird der Pincode definiert und kann beliebig geändert
   werden
38.  int8_t pin_code[3] = { 1,2,4 } ;
39.  int8_t gefundene_pin_code[3] = {0} ;
```

Das Programm auf dem Mikrocontroller ist über 600 Zeilen und verwendet viele Bibliotheken und Funktionen. Daher lohnt es sich den Code für mehr Informationen einzusehen. Die Abbildung 7-26 zeigt die Bestandteile des Empfängers (Teil-B).

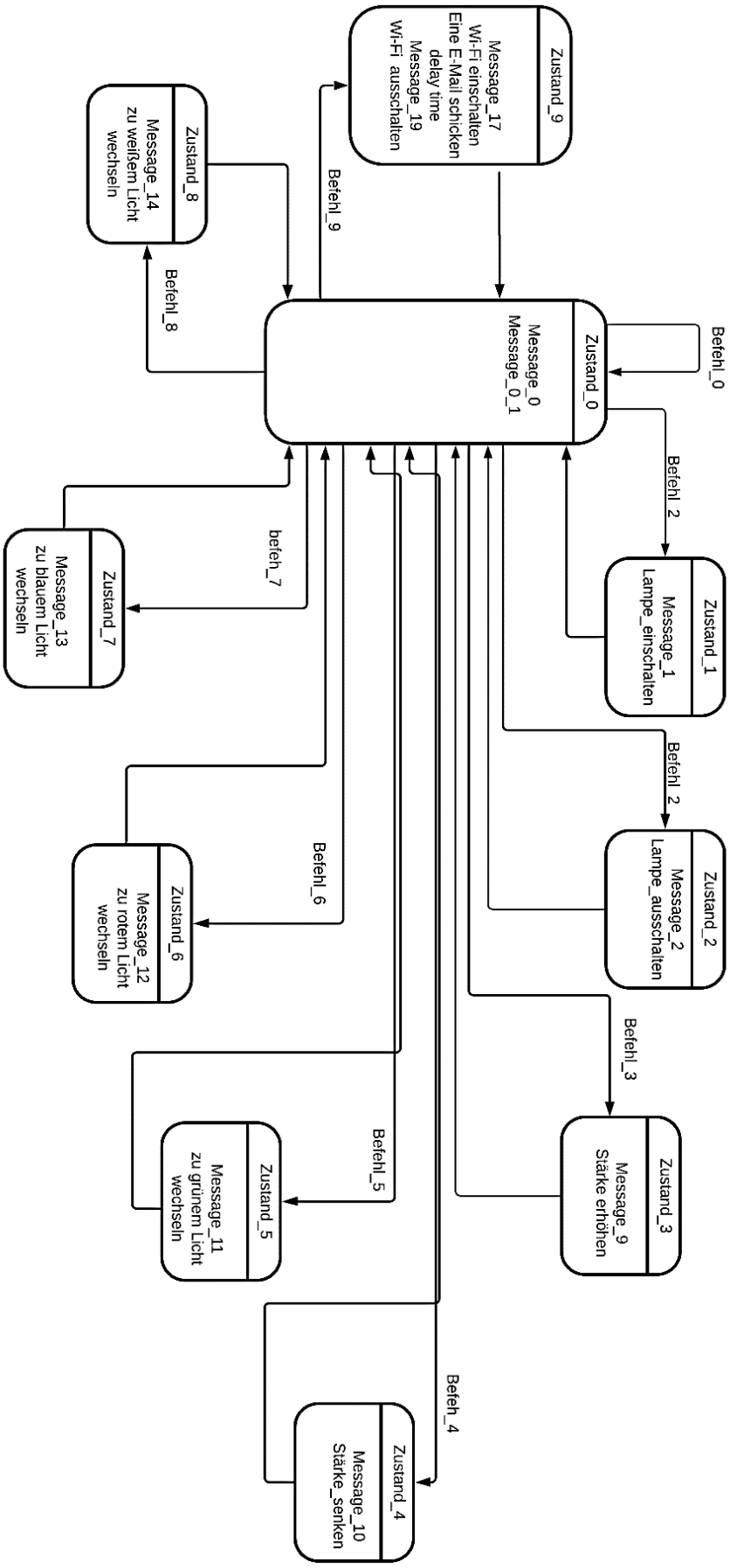


Abbildung 7-25: Die Zustandsmaschine des Empfängers

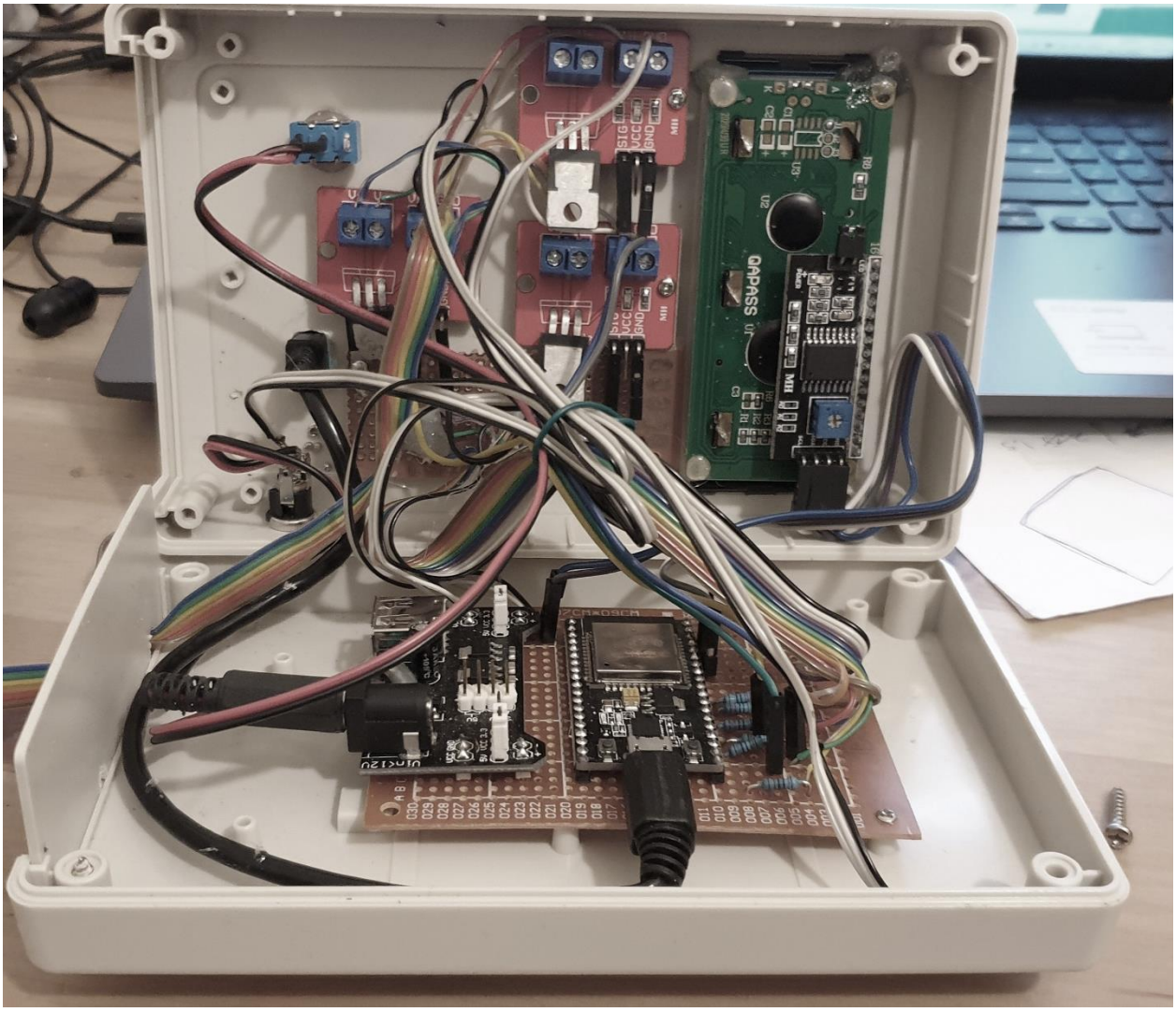


Abbildung 7-26: Die Bestandteile des Empfängers

Kapitel 8 **- Bewertung der Ergebnisse**

Im Laufe dieser Arbeit wurden mehrere Modelle in den Anwendungsbereichen Spracherkennung und Objekterkennung auf einem passenden Mikrocontroller implementiert und schließlich eine per Sprache gesteuerte Lampe aufgebaut. Ausgehend von den Ergebnissen und Messungen im Kapitel 2ergibt sich, dass das Board Spakfun-Edge ein sehr interessantes Board für ML-Anwendungen ist, wobei der Energieverbrauch im Vergleich mit anderen Boards äußerst gering ist, was bei mobilen Anwendungen von entscheidender Bedeutung ist. Außerdem ist die Inferenzzeit der ML-Modelle auf diesem Board klein, obwohl die Taktfrequenz geringer als die der ESP-Boards beispielsweise. Dies ist besonders wichtig, wenn das Modell schnellstmöglich ausgeführt werden muss (bei Echtzeitanwendungen).

Hingegen verbraucht das ESP-Board viel mehr Energie im Vergleich mit Sparkfun-Board, was dieses Board für mobile Anwendungen weniger geeignet macht. Trotzdem bleibt dieses Board eine gute Wahl zur Implementierung von ML-Modellen, da es über den rechenfähigsten Prozessor verfügt. Was bei diesen Boards besonders wichtig ist, sind die zahlreichen Kommunikationsmöglichkeiten, die die Vernetzung mehrerer ESP-Boards zum Datenaustausch in Echtzeit ermöglichen. Zudem gibt es viele günstige ESP-Varianten, die in einer bestimmten ML-Anwendung verwendet werden können. Zum Beispiel kann das sehr günstige Board ESP-32 NodeMCU mit einem externen Mikrophon-Modul anstelle der teuren ESP-EYE für die Spracherkennungs-anwendungen eingesetzt werden.

Das Arduino-Board ist ebenfalls ein interessantes Board, das über zahlreiche Sensoren verfügt und sehr geringen Stromverbrauch aufweist, was möglich macht, zahlreiche Anwendungen in den verschiedenen Anwendungsgebieten zu implementieren.

Die Objekterkennung in Echtzeit wie die Verfolgung oder Zählung von Objekten ist auf den in dieser Arbeit verwendeten Boards unmöglich, weil solche Modelle rechenintensiv sind und die Inferenzzeit hoch ist. Bei den Spracherkennungs-anwendungen war die Inferenzzeit klein und wurde das Modell mehr als 5 Male pro Sekunde ausgeführt. Bei Objekterkennung ist es nicht der Fall, weil die Eingabe viel

größer und komplexer ist, was zu einer drastischen Erhöhung des Rechenaufwands und Speicherbedarfs führt.

Transfer-Learning ist eine Wichtige Technik, die ermöglicht, vortrainierte Modelle auf eine neue Aufgabe zu transferieren. Diese Technik ist besonders wichtig, wenn das Modell sehr groß ist und sehr lange Trainingszeit benötigt. Solche Modelle sind in einem sehr begrenzten Umfang auf dem Mikrocontroller implementierbar.

Im Kapitel 7wurde ein per Sprache gesteuertes Gerät aufgebaut. Dafür wurden mehr als ein Board verwendet. Der Grund dafür war, dass bei solchen Geräten die Erkennung von sequenziellen und zeitlich verbundenen Daten erforderlich ist. Solche Erkennungssysteme benötigen eine spezielle Art der neuronalen Netze wie LSTM. Diese Netze sind für Mikrocontroller noch nur eingeschränkt unterstützt und benötigen einen großen Speicher (Kapitel 5). Dies führte dazu, dass diese Netze nur für kleine Anwendungen implementierbar sind.

Auf der Webseite von Tensorflow wurde erwähnt, dass LSTM-Netze auch für Mikrocontroller in kommenden Versionen vollständig unterstutzt werden. Dies wird ermöglichen, dass die Verwendung von mehreren Boards und daraus resultierenden Systemkomplexität (weil die Boards miteinander kommunizieren müssen) vermieden werden kann.

Die Implementierung der ML-Modelle auf dem Mikrocontroller wird sich zukünftig sehr verbreiten, weil viele Probleme dadurch gelöst werden können. Daher kann in den kommenden Jahren erwartet werden, dass diese Branche des maschinellen Lernens schnell weiterentwickelt wird, wobei neue fähigeren SoC-Boards zur Verfügung gestellt und unterstützt werden.

Das interessante Board **“ Himax WE-I Plus EVB ”** wurde in den letzten Wochen zur Verfügung gestellt und von Tensorflow unterstützt. Es ist teurer als die bis jetzt unterstützten Boards, aber hat einen sehr mächtigen Prozessor, der das Person-Detection-Beispiel nur in 40 Millisekunden ausführen kann. Das bedeutet, dass ein Objekterkennungsmodell in Echtzeit wie Beispielsweise Zählung von Fahrrädern auf diesem Board implementierbar ist. Beispielsweise hat das Board einen sehr großen Arbeitsspeicher für die Ausführung größerer Modelle, und einen mächtigen Prozessor, wodurch viel mehr Modelle unter Verwendung von Transfer-Learning zum Beispiel ausgeführt werden kann.

I Abkürzungsverzeichnis

ML	Machine Learning, Maschinelles Lernen
KI	Künstliche Intelligent
FC	Fully Connected, vollständig verbundene Netze
API	Application Programming Interface
GPIO	General Purpose Input/Output
CNN	Convolutional Neural Network
IDE	Integrated Development Environment
RNN	recurrent neural network
LSTM	Long Short Term Memory
CMSIS	Cortex Microcontroller Software Interface Standard
ADC	Analog to Digital Converter
FFT	Fast Fourier Transformation
MFCC	Mel Frequency Cepstral Coefficients
MSE	Mean Square Error
ESP-IDF	Espressif IoT Development Framework
NLP	Neuro-linguistic Programming, Verarbeitung natürlicher Sprache
NLU	Natural Language Understanding, Verstehen natürlicher Sprache
SOC	System on Chip
MAC	Media-Access-Control-Adresse
RTOS	Real-time Operating System

II Abbildungsverzeichnis

III Tabellenverzeichnis

IV Listingverzeichnis

V Literaturverzeichnis

[1] Wikipedia(o.D): *Geschichte der künstlichen Intelligenz* (Online), https://de.wikipedia.org/wiki/Geschichte_der_k%C3%BCnstlichen_Intelligenz. (Zugriff am 1 8 2020).

[2] Wikipedia(o.D): *McCulloch-Pitts-Zelle* (Online), https://de.wikipedia.org/wiki/McCulloch-Pitts-Zelle. (Zugriff am 3 8 2020).

[3] ESPRESSIF (o.D) : *Esp-Now (*Online), https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html. (Zugriff am 14 10 2020).

[4] Aurélien, Géron: *Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow*, 1. Hrsg., dpunkt.verlag, 2018.

[5] W. YAQING , Y. QUANMING , T. K. JAMES und M. N. LIONEL(2020) : *Generalizing from a Few Examples A Survey on Few-Shot*, https://arxiv.org/pdf/1904.05046.pdf.

[6] Wuttke, Laurenz (o.D): *Machine Learning vs. Deep Learning: Wo ist der Unterschied?* (Online), https://datasolut.com/machine-learning-vs-deep-learning/. (Zugriff am 12 8 2020).

[7] Zinser, Erwin ; Zugaj, Wilhelm (o.D): *Künstliche Intelligenz & Big Data* (Online) ,https://www.fh-joanneum.at/forschung/forschungszentren/big-data-and-artificial-intelligence-research-center/kuenstliche-intelligenz-big-data/. (Zugriff am 3 7 2020).

[8] Ramsundar, Bharath; Zadeh, Reza Bosagh : *TensorFlow for Deep Learning*, O'Reilly Media, 2018.

[9] BECKER, ROLAND (2019): *Convolutional neural networks –Aufbau, Funktion und Anwendungsgebiete* (Online), https://jaai.de/convolutional-neural-networks-cnn-aufbau-funktion-und-anwendungsgebiete-1691/.

[10] Shafkat, Irhum (2018) : *Intuitively Understanding Convolutions for Deep Learning* (Online), Available: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1. (Zugriff am 5 7 2020).

[11] Lorenzen, Knut : *Machine Learning mit Python und Scikit-learn und TensorFlow*, mitp verlag, 2018 .

[12] Hope , Tom ; Yehezkel, Resheff; Lieder, Itay : *Einführung in Tensorflow,* dpunkt.verlag, 2018 .

[13] RIP Tutorial (o.D) : *Softmax-Funktion* (Online), https://riptutorial.com/de/machine-learning/example/31625/softmax-funktion. (Zugriff am 17 8 2020).

[14] it-production(o.D) : *Der Weg der Daten im Industrial IoT*(Online), https://www.it-production.com/produktionsmanagement/daten-im-industrial-iot/. (Zugriff am 17 10 2020).

[15] tensorflow(o.D) : *TensorFlow Lite für Mikrocontroller* (Online), https://www.tensorflow.org/lite/microcontrollers. (Zugriff am 2 7 2020).

[16] Warden ,Pete ; Situanayake , Daniel : *TinyML: Machine Learning with TensorFlow on Arduino, and Ultra-Low Power Microcontrollers* , USA: O'Reilly Media, 2019.

[17] Stanford University (2020) : *Machine learning on embedded systems* (Online), https://docs.google.com/presentation/d/1zGm5bqGrkAepwJZ5PABiYjrIKq1pDnzafa8ZYeaFhXY/edit#slide=id.p1.

[18] Warden , Pete (2017): *What I've learned about neural network quantization* (Online), https://petewarden.com/2017/06/22/what-ive-learned-about-neural-network-quantization/. (Zugriff am 28 8 2020).

[19] M. Z. B. C. D. K. ,. W. ,. W. M. A. H. A. Andrew G. Howard (2017): *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision*, Google Inc.

[20] mathworks (o.D) : *Counting Steps by Capturing Acceleration Data from Your Mobile Device* (Online), https://de.mathworks.com/help/matlabmobile_android/ug/counting-steps-by-capturing-acceleration-data.html. (Zugriff am 28 9 2020).

[21] Stotz , Dieter : *Computergestützte Audio- und Videotechnik*, Springer, 2011.

[22] Github (o.D) : *Micro Speech Training* (Online), https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/micro/examples/micro_speech/train. (Zugriff am 25 6 2020).

[23] Tensorflow (o.D): *TensorFlow Lite converter* (Online), https://www.tensorflow.org/lite/convert. (Zugriff am 10 8 2020).

[24] EPRESSIF (o.D) : *Get Started* (Online), https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html. (Zugriff am 12 10 2020).

[25] „Ozeki Camera SDK(o.D) : *How to implement circular buffer video recording in C#* (Online), https://www.camera-sdk.com/p_6554-how-to-implement-circular-buffer-video-recording-in-c-.html. (Zugriff am 7 9 2020).

[26] Tensorflow (o.D) : *Beginnen Sie mit Mikrocontrollern* (Online), https://www.tensorflow.org/lite/microcontrollers/get_started. (Zugriff am 9 7 2020).

[27] Shaikh , Faizan (2017) : *Why are GPUs necessary for training Deep Learning models?* (Online), https://www.analyticsvidhya.com/blog/2017/05/gpus-necessary-for-deep-learning/.(Zugriff am 6 7 2020).

[28] Luber , Stefan ; Litzel, Nico (o.D) : *Was ist eine Tensor Processing Unit (TPU)?* (Online), https://www.bigdata-insider.de/was-ist-eine-tensor-processing-unit-tpu-a-750292/. (Zugriff am 22 10 2020).

[29] Olah, Christopher (2015): *Understanding LSTM nets* (Online), https://colah.github.io/posts/2015-08-Understanding-LSTMs/. (Zugriff am 8 19 2020).

[30] jaai (2019): *Transfer Learning –So können neuronale Netze voneinander lernen* (Online), https://jaai.de/transfer-learning-1739/. (Zugriff am 25 10 2020).

[31] Keras (o.D): *Keras Applications* (Online), https://keras.io/api/applications/. (Zugriff am 2 10 2020).

[32] Espressif (o.D) : *ESP-EYE Getting Started Guide* (Online), https://github.com/espressif/esp-who/blob/master/docs/en/get-started/ESP-EYE_Getting_Started_Guide.md. (Zugriff am 2 12 2020).

VI Anhangsverzeichnis

Anhang A. Die Terminalbefehle

A.1. Tensorflow Klonen

In diesem Anhang wird der Terminalcode zur Generierung der Tensorflow-Beispiele in Kapitel 3 aufgelistet. Der erste Schritt ist dabei das Klonen der Tensorflow-Repository:

```
1. git clone --depth 1 https://github.com/tensorflow/tensorflow.git
2. cd tensorflow
```

Um die Beispiele kompilieren zu können und auf den Mikrocontroller zu laden muss man einige Backages installieren. Die wichtigsten sind:

21. Arduino IDE für Arduino-Mikrocontroller
22. ESP-IDF für ESP-Mikrocontroller

Die Beispiele, die im Kapital-3 erwähnt wurden, befinden sich in Tensorflow-Repository in dem Folder:

```
tensorflow/tensorflow/lite/micro/examples/
```

A.2. Generierung der Beispiele für Arduino Nano

Die generierten Beispiele für Arduino-Nano mithilfe von den Terminalbefehlen findet man in dem folgenden Folder:

```
/tensorflow/tensorflow/lite/micro/tools/make/gen
```

In diesem Folder befindet sich ein Arduino-Folder namens **arduino_x86_64.** Dieser Folder enthält unter anderem ein ZIP-File. Dies ist ein Arduino-Library, die den Bibliotheken der Arduino-IDE hinzugefügt werden kann.

A.2.1. Hello-World

Im Terminal wird der folgende Code eingegeben:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile TARGET=arduino TAGS="" 
   generate_hello_world_arduino_library_zip
```

Für den Code mit optimiertem Kernel (CMSIS-NN Bibliothek):

```
1. make -f tensorflow/lite/micro/tools/make/Makefile TARGET=arduino TAGS=
   "cmsis-nn" generate_hello_world_arduino_library_zip
```

A.2.2. Micro Speech

Im Terminal den folgenden Code eingeben:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile TARGET=arduino TAGS=""
   generate_micro_speech_arduino_library_zip
```

Für den Code mit optimierten Kernel (CMSIS-NN Bibliothek):

```
1. make -f tensorflow/lite/micro/tools/make/Makefile TARGET=arduino
   TAGS="cmsis-nn" generate_micro_speech_arduino_library_zip
```

A.2.3. Magic Wand

Man muss Im Terminal den folgenden Code eingeben:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile TARGET=arduino TAGS=""
   generate_magic_wand_arduino_library_zip
```

Für den Code mit optimierten Kerneln (CMSIS-NN Bibliothek):

```
1. make -f tensorflow/lite/micro/tools/make/Makefile TARGET=arduino
   TAGS="cmsis-nn" generate_magic_wand_arduino_library_zip
```

A.3. Sparkfun Edge

Zur Generierung von dem Code muss man einige Backages installieren.

Zuerst muss pycrypto installiert werden

```
1. pip3 install pycrypto pyserial --user
```

Jetzt wird das Board dem Computer durch den Programmierer FTDI232 angeschlossen, weil das Board keinen eigenen USB-Anschluss hat. Die folgende Abbildung zeigt das Board mit dem Programmierer:

Abbildung A.1: Sparkfun-Board und FTDI232-Programmierer

A.3.1. Hello World

Durch den folgenden Code (schrittweise in den Terminal eingeben) kann man den Beispielcode auf dem Sparkfun-Board ausführen.

Schritt 1:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile clean
```

Schritt 2:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile TARGET=sparkfun_edge
   hello_world_bin
```

Schritt 3:

```
1. cp tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/keys_info0.py
   \tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/keys_info.py
```

Schritt 4:

```
1. python3 tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/create_cust_image_blob.py \
2. --bin tensorflow/lite/micro/tools/make/gen/sparkfun_edge_cortex-
   m4/bin/hello_world.bin \
3. --load-address 0xC000 \
4. --magic-num 0xCB \
5. -o main_nonsecure_ota \
6. --version 0x0
```

Schritt 5:

```
1.  python3 tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts /create_cust_wireupdate_blob.py \
2. --load-address 0x20000 \
3. --bin main_nonsecure_ota.bin \
4. -i 6 \
5. -o main_nonsecure_wire \
6. --options 0x1
```

Schritt 6:

```
1. export DEVICENAME=/dev/ttyUSB0
2. export BAUD_RATE=921600
```

Schritt 7:

```
1. python3 tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/uart_wired_update.py \
2. -b ${BAUD_RATE} ${DEVICENAME} \
3. -r 1 \
4. -f main_nonsecure_wire.bin \
5. -i 6
```

Anmerkung: Vor dem letzten Schritt muss man das Board dem Computer anschließen, dann die Taste-14 drücken und halten und nur einmal kurzzeitig die Taste Reset drücken, dann den Code im Schritt 7 eingeben. Während das Programm auf das Board geladen wird, muss die Taste 14 weiter gedrückt werden. Am Ende drückt man Reset nochmal, um das Programm auszuführen.

Man muss darauf achten, dass der Name des Devices und die Baudrate im Schritt 6 anders sein könnte. Das ist abhängig von dem Operating-System auf dem Computer.

A.3.2. Micro Speech

Schritt 1:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile clean
```

Schritt 2:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile  TARGET=sparkfun_edge
   micro_speech_bin
```

Schritt 3:

```
1. cp tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/keys_info0.py
   \tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/keys_info.py
```

Schritt 4:

```
1. python3 tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/create_cust_image_blob.py \
2. --bin tensorflow/lite/micro/tools/make/gen/sparkfun_edge_cortex-
   m4/bin/micro_speech.bin \
3. --load-address 0xC000 \
4. --magic-num 0xCB \
5. -o main_nonsecure_ota \
6. --version 0x0
```

Schritt 5:

```
1. python3 tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/create_cust_wireupdate_blob.py \
2. --load-address 0x20000 \
3. --bin main_nonsecure_ota.bin \
4. -i 6 \
5. -o main_nonsecure_wire \
6. --options 0x1
```

Schritt 6:

```
1. export DEVICENAME=/dev/ttyUSB0
2. export BAUD_RATE=921600
3.
```

Schritt 7:

```
1. python3 tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/uart_wired_update.py \
2. -b ${BAUD_RATE} ${DEVICENAME} \
3. -r 1 \
4. -f main_nonsecure_wire.bin \
5. -i 6
6.
```

Wenn man die Anwendung unter Verwendung von CMSIS-NN generieren möchte, muss der folgende Code im Schritt 2 eingegeben werden:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile TARGET=sparkfun_edge
   TAGS="cmsis-nn" micro_speech_bin
```

A.3.3. Person Detection

Schritt 1:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile clean
```

Schritt 2:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile TARGET=sparkfun_edge
   person_detection_bin
```

Schritt 3:

```
1. cp tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/keys_info0.py \
2. tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/keys_info.py
3.
```

Schritt 4:

```
1. python3 tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/create_cust_image_blob.py \
2. --bin tensorflow/lite/micro/tools/make/gen/sparkfun_edge_cortex-
   m4/bin/person_detection.bin \
3. --load-address 0xC000 \
4. --magic-num 0xCB \
5. -o main_nonsecure_ota \
6. --version 0x0
```

Schritt 5:

```
1. python3 tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/create_cust_wireupdate_blob.py \
2. --load-address 0x20000 \
3. --bin main_nonsecure_ota.bin \
4. -i 6 \
5. -o main_nonsecure_wire \
6. --options 0x1
```

Schritt 6:

```
1. export DEVICENAME=/dev/ttyUSB0
2. export BAUD_RATE=921600
```

Schritt 7:

```
1. python3 tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/uart_wired_update.py \
2. -b ${BAUD_RATE} ${DEVICENAME} \
3. -r 1 \
4. -f main_nonsecure_wire.bin \
5. -i 6
```

Wenn man die Anwendung unter Verwendung von CMSIS-NN generieren möchte, muss der folgende Code im Schritt 2 eingegeben werden:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile TARGET=sparkfun_edge
   TAGS="cmsis-nn" person_detection_bin
```

A.3.4. Magic Wand

Schritt 1:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile clean
```

Schritt 2:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile TARGET=sparkfun_edge
   magic_wand_bin
```

Schritt3:

```
1. cp tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/keys_info0.py \
```

```
2. tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/keys_info.py
```

Schritt4:

```
1. python3 tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/create_cust_image_blob.py \
2. --bin tensorflow/lite/micro/tools/make/gen/sparkfun_edge_cortex-
   m4/bin/magic_wand.bin \
3. --load-address 0xC000 \
4. --magic-num 0xCB \
5. -o main_nonsecure_ota \
6. --version 0x0
```

Schritt 5:

```
1. python3 tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/create_cust_wireupdate_blob.py \
2. --load-address 0x20000 \
3. --bin main_nonsecure_ota.bin \
4. -i 6 \
5. -o main_nonsecure_wire \
6. --options 0x1
```

Schritt 6:

```
1. export DEVICENAME=/dev/ttyUSB0
2. export BAUD_RATE=921600
```

Schritt 7:

```
1. python3 tensorflow/lite/micro/tools/make/downloads/AmbiqSuite-
   Rel2.2.0/tools/apollo3_scripts/uart_wired_update.py \
2. -b ${BAUD_RATE} ${DEVICENAME} \
3. -r 1 \
```

```
4. -f main_nonsecure_wire.bin \
5. -i 6
```

Wenn man die Anwendung unter Verwendung von CMSIS-NN generieren möchte, muss der folgende Code im **Schritt 2** eingegeben werden:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile TARGET=sparkfun_edge
   TAGS="cmsis-nn" magic_wand_bin
```

A.4. Generierung der Beispiele für ESP-Board

Bevor man die Beispiele auf dem Esp-Mikrocontroller ausführen kann, muss die Entwicklung-Framework ESP-IDF installiert werden. Es kann hilfreich sein die Webseite von ESP-IDF zu besuchen.
Wenn das ESP-Board keinen Mikro-USB-Anschluss hat, muss man den Adapter FTDI232 verwenden.

A.4.1. Micro Speech

Schritt 1:

```
1. . $HOME/esp/esp-idf/export.sh
```

Schritt 2:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile clean
```

Schritt 3:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile TARGET=esp
   generate_micro_speech_esp_project
```

Schritt 4:

```
1. cd tensorflow/lite/micro/tools/make/gen/esp_xtensa-
   esp32/prj/micro_speech/esp-idf
```

Schritt 5:

```
1. idf.py build
```

Schritt 6:

```
1. idf.py --port /dev/ttyUSB0 flash
```

Wenn man die Einstellungen des Mikrocontroller verändern möchte, muss man den folgenden Befehl nach Schritt 4 eingeben.

```
1. idf.py menuconfig
```

Nach der Eingabe dieses Befehles, wird das in der Abbildung A.3 gezeigte Fenster gezeigt. Dieses Fenster enthält alle Einstellungen des Mikrocontrollers.

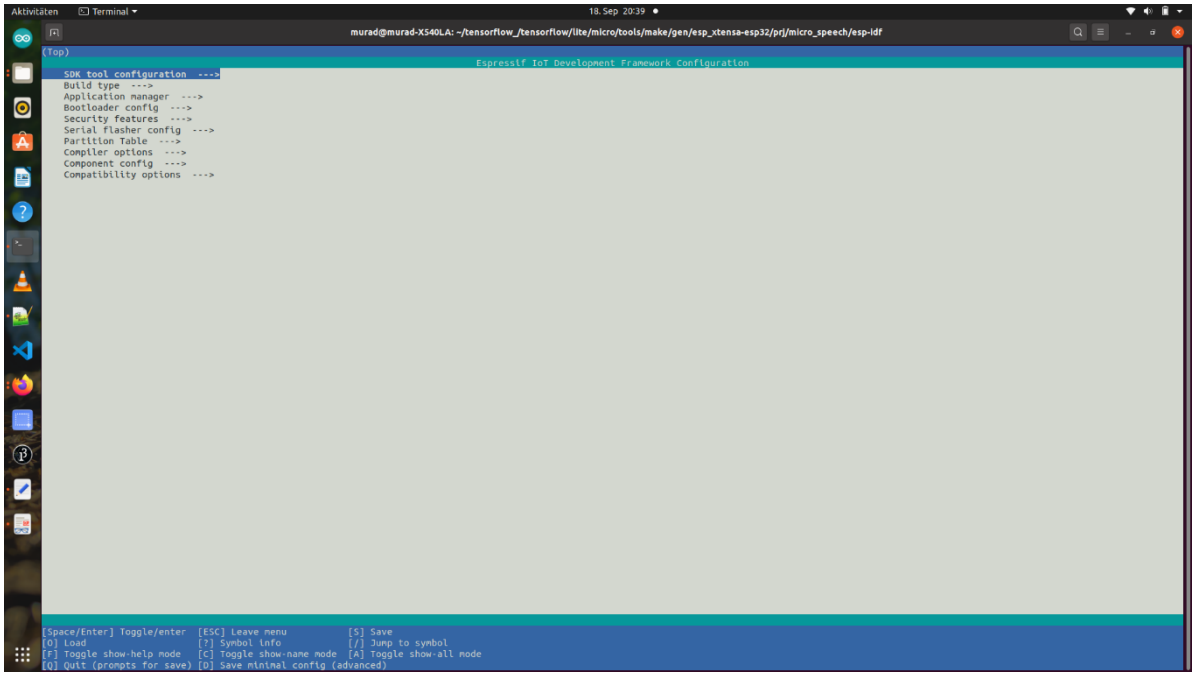


Abbildung A.2: ESP-Einstellungen

Von hier wählt man beispielsweise das Esp-Board, das programmiert werden muss und die Taktfrequenz, mit der das Board betrieben werden muss. Bei Default-Einstellungen ist das Ausgewählte Board ESP-EYE und die Taktfrequenz 160 MHZ.

A.4.2. Person Detection

Schritt 1:

```
1. . $HOME/esp/esp-idf/export.sh
```

Schritt 2:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile clean
```

Schritt 3:

```
1. make -f tensorflow/lite/micro/tools/make/Makefile TARGET=esp
   generate_person_detection_esp_project
```

Schritt 4:

```
1. cd tensorflow/lite/micro/tools/make/gen/esp_xtensa-
   esp32/prj/person_detection/esp-idf
```

Schritt 5:

```
1. git clone https://github.com/espressif/esp32-camera.git components/esp32-
   camera
```

Schritt 7:

```
1. idf.py build
```

Schritt 8:

```
1. idf.py --port /dev/ttyUSB0 flash
```

Anhang B. Die unterstützten Operationen in TFlite for Microcontroller

AddAdd()
AddArgMax()
AddArgMin();
AddAveragePool2D()
AddCeil()
AddConcatenation()
AddConv2D()
AddCos()
AddDepthwiseConv2D()
AddDequantize()
AddEqual()
AddFloor()
AddFullyConnected()
AddGreater()
AddGreaterEqual()
AddHardSwish()
AddL2Normalization()
AddLess()
AddLessEqual()
AddLog()
AddLogicalAnd()
AddLogicalNot()
AddLogicalOr()
AddLogistic()
AddNeg()
AddNotEqual()
AddPack()
AddPad()
AddPadV2()
AddPrelu()
AddQuantize()
AddReduceMax()
AddRelu()
AddRelu6()
AddReshape()
AddResizeNearestNeighbor()
AddRound()
AddRsqrt()
AddShape()
AddSin()
AddSoftmax()
AddSplit()
AddSplitV()
AddSqrt()
AddSquare()
AddStridedSlice()
AddSub()
AddSvdf()
AddMinimum()
AddTanh()
AddUnpack()
AddMul()
AddMaximum()
AddMaxPool2D()
AddMean()

Anhang C. Mobilenet V1 Labels

Background
tench
goldfish
great white shark
tiger shark
hammerhead
electric ray
stingray
cock
hen
ostrich
brambling
goldfinch
house finch
junco
indigo bunting
robin
bulbul
jay
magpie
chickadee
water ouzel
kite
bald eagle
vulture
great grey owl
European fire salamander
common newt
eft
spotted salamander
axolotl
bullfrog
tree frog
tailed frog
loggerhead
leatherback turtle
mud turtle
terrapin
box turtle
banded gecko
common iguana
American chameleon
whiptail
agama
frilled lizard
alligator lizard
Gila monster
green lizard
African chameleon
black swan
tusker
echidna
platypus
wallaby
koala
wombat
jellyfish
sea anemone
brain coral
flatworm

Angora
hamster
porcupine
fox squirrel
marmot
beaver
guinea pig
sorrel
zebra
hog
wild boar
warthog
hippopotamus
ox
water buffalo
bison
ram
bighorn
ibex
hartebeest
impala
gazelle
Arabian camel
llama
weasel
mink
polecat
black-footed ferret
otter
skunk
badger
armadillo
three-toed sloth
orangutan
gorilla
chimpanzee
gibbon
siamang
guenon
patas
baboon
macaque
langur
colobus
proboscis monkey
marmoset
capuchin
howler monkey
titi
spider monkey
squirrel monkey
Madagascar cat
indri
Indian elephant
African elephant
lesser panda
giant panda
barracouta
eel
coho
rock beauty
anemone fish
sturgeon
gar

diaper
digital clock
digital watch
dining table
dishrag
dishwasher
disk brake
dock
dogsled
dome
doormat
drilling platform
drum
drumstick
dumbbell
Dutch oven
electric fan
electric guitar
electric locomotive
entertainment center
envelope
espresso maker
face powder
feather boa
file
fireboat
fire engine
fire screen
flagpole
flute
folding chair
football helmet
forklift
fountain
fountain pen
four-poster
freight car
French horn
frying pan
fur coat
garbage truck
gasmask
gas pump
goblet
go-kart
golf ball
golfcart
gondola
gong
gown
grand piano
greenhouse
grille
grocery store
guillotine
hair slide
hair spray
half track
hammer
hamper
hand blower
hand-held computer
handkerchief

plunger
Polaroid camera
pole
police van
poncho
pool table
pop bottle
pot
potter's wheel
power drill
prayer rug
printer
prison
projectile
projector
puck
punching bag
purse
quill
quilt
racer
racket
radiator
radio
radio telescope
rain barrel
recreational vehicle
reel
reflex camera
refrigerator
remote control
restaurant
revolver
rifle
rocking chair
rotisserie
rubber eraser
rugby ball
rule
running shoe
safe
safety pin
saltshaker
sandal
sarong
sax
scabbard
scale
school bus
schooner
scoreboard
screen
screw
screwdriver
seat belt
sewing machine
shield
shoe shop
shoji
shopping basket
shopping cart
shovel
shower cap
shower curtain
ski

traffic light
book jacket
menu
plate
guacamole
consomme
hot pot
trifle
ice cream
ice lolly
French loaf
bagel
pretzel
cheeseburger
hotdog
mashed potato
head cabbage
broccoli
cauliflower
zucchini
spaghetti
squash
acorn squash
butternut squash
cucumber
artichoke
bell pepper
cardoon
mushroom
Granny Smith
strawberry
orange
lemon
fig
pineapple
banana
jackfruit
custard apple
pomegranate
hay
carbonara
chocolate sauce
dough
meat loaf
pizza
potpie
burrito
red wine
espresso
cup
eggnog
alp
bubble
cliff
coral reef
geyser
lakeside
promontory
sandbar
seashore
valley
volcano
ballplayer
groom
scuba diver

Anhang C Mobilenet V1 Labels

nematode
conch
snail
slug
sea slug
chiton
chambered nautilus
Dungeness crab
rock crab
fiddler crab
king crab
American lobster
spiny lobster
crayfish
hermit crab
isopod
white stork
black stork
spoonbill
flamingo
little blue heron
American egret
bittern
crane
limpkin
European gallinule
American coot
bustard
ruddy turnstone
red-backed sandpiper
redshank
dowitcher
oystercatcher
pelican
king penguin
albatross
grey whale
killer whale
dugong
sea lion
Chihuahua
Japanese spaniel
Maltese dog
Pekinese
Shih-Tzu
Blenheim spaniel
papillon
toy terrier
Rhodesian ridgeback
Afghan hound
basset
beagle
bloodhound
bluetick
black-and-tan coonhound
Walker hound
English foxhound
redbone
borzoi
Irish wolfhound
Italian greyhound
lionfish
puffer
abacus
abaya
academic gown
accordion
acoustic guitar
aircraft carrier
airliner
airship
altar
ambulance
amphibian
analog clock
apiary
apron
ashcan
assault rifle
backpack
bakery
balance beam
balloon
ballpoint
Band Aid
banjo
bannister
barbell
barber chair
barbershop
barn
barometer
barrel
barrow
baseball
basketball
bassinet
bassoon
bathing cap
bath towel
bathtub
beach wagon
beacon
beaker
bearskin
beer bottle
beer glass
bell cote
bib
bicycle-built-for-two
bikini
binder
binoculars
birdhouse
boathouse
bobsled
bolo tie
bonnet
bookcase
bookshop
bottlecap
bow
bow tie
brass
brassiere
breakwater
breastplate
broom
bucket
buckle
bulletproof vest
hard disc
harmonica
harp
harvester
hatchet
holster
home theater
honeycomb
hook
hoopskirt
horizontal bar
horse cart
hourglass
iPod
iron
jack-o'-lantern
jean
jeep
jersey
jigsaw puzzle
jinrikisha
joystick
kimono
knee pad
knot
lab coat
ladle
lampshade
laptop
lawn mower
lens cap
letter opener
library
lifeboat
lighter
limousine
liner
lipstick
Loafer
lotion
loudspeaker
loupe
lumbermill
magnetic compass
mailbag
mailbox
maillot
maillot
manhole cover
maraca
marimba
mask
matchstick
maypole
maze
measuring cup
medicine chest
megalith
microphone
microwave
military uniform
milk can
minibus
miniskirt
minivan
missile
mitten
mixing bowl
mobile home
Model T
ski mask
sleeping bag
slide rule
sliding door
slot
snorkel
snowmobile
snowplow
soap dispenser
soccer ball
sock
solar dish
sombrero
soup bowl
space bar
space heater
space shuttle
spatula
speedboat
spider web
spindle
sports car
spotlight
stage
steam
locomotive
steel arch bridge
steel drum
stethoscope
stole
stone wall
stopwatch
stove
strainer
streetcar
stretcher
studio couch
stupa
submarine
suit
sundial
sunglass
sunglasses
sunscreen
suspension bridge
swab
sweatshirt
swimming trunks
swing
switch
syringe
table lamp
tank
tape player
teapot
teddy
television
tennis ball
thatch
theater curtain
thimble
thresher
throne
tile roof
toaster
tobacco shop
toilet seat
torch
rapeseed
daisy
yellow lady's slipper
corn
acorn
hip
buckeye
coral fungus
agaric
gyromitra
stinkhorn
earthstar
hen-of-the-woods
bolete
ear
toilet tissue

Anhang C Mobilenet V1 Labels

whippet
Ibizan hound
Norwegian elkhound
otterhound
Saluki
Scottish deerhound
Weimaraner
Staffordshire bullterrier
American Staffordshire terrier
Bedlington terrier
Siamese cat
Egyptian cat
cougar
lynx
leopard
snow leopard
jaguar
lion
tiger
cheetah
brown bear
American black bear
ice bear
sloth bear
mongoose
meerkat
tiger beetle
ladybug
ground beetle
long-horned beetle
leaf beetle
dung beetle
rhinoceros beetle
weevil
fly
bee
ant
grasshopper
cricket
walking stick
cockroach
mantis
cicada
leafhopper
lacewing
dragonfly
damselfly
admiral
ringlet
monarch
cabbage butterfly
sulphur butterfly
lycaenid
starfish
sea urchin
sea cucumber
wood rabbit
hare

bullet train
butcher shop
cab
caldron
candle
cannon
canoe
can opener
cardigan
car mirror
carousel
carpenter's kit
carton
car wheel
cash machine
cassette
cassette player
castle
catamaran
CD player
cello
cellular telephone
chain
chainlink fence

chain mail
chain saw
chest
chiffonier
chime
china cabinet
Christmas stocking
church
cinema
cleaver
cliff dwelling
cloak
clog
cocktail shaker
coffee mug
coffeepot
coil
combination lock
computer keyboard
confectionery
container ship
convertible
corkscrew
cornet
cowboy boot
cowboy hat
cradle
crane
crash helmet
crate
crib
Crock Pot
croquet ball
crutch
cuirass
dam
desk
desktop computer
dial telephone

modem
monastery
monitor
moped
mortar
mortarboard
mosque
mosquito net
motor scooter
mountain bike
mountain tent
mouse
mousetrap
moving van
muzzle
nail
neck brace
necklace
nipple
notebook
obelisk
oboe
ocarina
odometer
oil filter
organ
oscilloscope
overskirt
oxcart
oxygen mask
packet
paddle
paddlewheel
padlock
paintbrush
pajama
palace
panpipe
paper towel
parachute
parallel bars
park bench
parking meter
passenger car
patio
pay-phone
pedestal
pencil box
pencil sharpener
perfume
Petri dish
photocopier
pick
pickelhaube
picket fence
pickup
pier
piggy bank
pill bottle
pillow
ping-pong ball
pinwheel
pirate
pitcher
plane
planetarium
plastic bag
plate rack
plow

totem pole
tow truck
toyshop
tractor
trailer truck
tray
trench coat
tricycle
trimaran
tripod
triumphal arch
trolleybus
trombone
tub
turnstile
typewriter keyboard
umbrella
unicycle
upright
vacuum
vase
vault
velvet
vending machine
vestment
viaduct
violin
volleyball
waffle iron
wall clock
wallet
wardrobe
warplane
washbasin
washer
water bottle
water jug
water tower
whiskey jug
whistle
wig
window screen
window shade
Windsor tie
wine bottle
wing
wok
wooden spoon
wool
worm fence
wreck
yawl
yurt
web site
comic book
crossword
puzzle
street sign

Anhang D. Inhalt des Datenträgers

Inhalt/Python_Code/Kapitel_4: enthält den Python-Code und alle Funktionen, die im Kapitel-4 erwähnt wurden.

Inhalt/Python_Code/Kapitel_5: enthält den Python-Code für Kapitel-5 .

Inhalt/Python_Code/Kapitel_6: enthält den Python-Code für Kapitel-6 .

Inhalt/Python_Code/Kapitel_7: enthält den Python-Code zum Training der Erkennungsmodelle im Kapitel 7.

Inhalt/C-Code/Kapitel_4: enthält die C-Projektverzeichnisse für die ESP-Boards im Kapitel-4.

Inhalt/C-Code/Kapitel_5: enthält das C-Projektverzeichnis für das Erkennungs-modell im Kapitel-5

Inhalt/C-Code/Kapitel_6: enthält das C-Projektverzeichnis für das Board ESP-Cam im Kapitel-6 .

Inhalt/C-Code/Hauptprojekt: enthält die 3 Projektverzeichnisse für die 3 ESP-Boards, die im Kapitel-7 eingesetzt wurden.

Inhalt/Verschiedenses : Dieser Ordner enthält Bilder , Modelle und weitere Daten.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____