

ING

Fakultät
Ingenieurwissenschaften

HTWK

Hochschule für Technik,
Wirtschaft und Kultur Leipzig

Bachelorarbeit

Thema: Evaluierung von Beschleuniger-Komponenten auf einem eingebetteten System

vorgelegt von: Thijs Joris Bakker

Studiengang: ELEKTROTECHNIK UND INFORMATIONSTECHNIK

Studienprofil: ELEKTRONISCHE SCHALTUNGSTECHNIK UND SIGNALVERARBEITUNG

verantwortlicher Hochschullehrer: Prof. Dr. Gerold Bausch

betrieblicher Betreuer: Herr Vincent Weiss

Ausgabetermin: 06. Dezember 2024

Abgabetermin: 28.02.2025

Leipzig, 27. Februar 2025

Aufgabenbeschreibung Bachelorarbeit

„Evaluation von Beschleuniger-Komponenten auf einem eingebetteten System“

für Herrn Joris Bakker

06. Dezember 2024

Machine-Learning-Anwendungen finden zunehmend ihren Weg in eingebettete Systeme, um Sensordaten bereits am Ort der Messung bzw. Datenaufnahme zu verarbeiten und nur auf bestimmte Ereignisse Reaktionen auszuführen. Dadurch lassen sich zum einen datenschutzrechtliche Aspekte verbessern, da Rohdaten weder gespeichert noch übertragen werden müssen, zum anderen sind keine Breitbandverbindungen zur Übertragung von Rohdaten z.B. an Cloud-Systeme notwendig, was zu geringerem Energiebedarf führt.

Für eine effiziente Verarbeitung der Daten auf eingebetteten Systemen stehen in modernen Mikrocontrollern wie dem Max78000 spezielle Hardwarefunktionen zur Verfügung. Diese sind in ihrer Leistungsfähigkeit selten vergleichbar mit klassischer ML-Hardware und erfordern häufig auch einen anderen Workflow bei der Erstellung von Programmen.

Ziel der Bachelorarbeit ist es, einen Klassifikator für akustische Szenen zu trainieren und auf einem Max78000 Evaluationsboard lauffähig zu implementieren. Die Performance des Max78000 unter Verwendung des CNN-Beschleunigers soll gegenüber einer herkömmlichen Implementation in Python evaluiert werden. Aussagen zu Inferenzzeiten und Energieverbrauch sind zu treffen.

Aufgaben:

1. Implementierung eines Szenenklassifikators auf Basis eines 1D-CNN in Tensorflow/Keras, alternativ Pytorch 2
2. Training und Evaluation des Netzes auf Basis des TUT Acoustic Scenes 2017-Datensatzes für eine Auswahl von 5 bis 10 Klassen. Möglichkeiten der Datenreduktion zur Beschleunigung der Inferenzphase und Speicherreduktion sind zu berücksichtigen.
3. Übertragung des trainierten Netzes auf den CNN-Hardwarebeschleuniger des Max78000
4. Evaluation von Energieverbrauch und Ausführungszeit des trainierten Netzes auf dem Beschleuniger, sowie der erreichten Performance gegenüber der Referenz.
5. Dokumentation des Workflows sowie der erzielten Ergebnisse.

Betreuer: M.Sc. Vincent Weiß

Prof. Dr.-Ing. Gerold Bausch
Betreuer Hochschulprofessor

Abstract

In this work, a convolutional neural network was trained and implemented on an embedded system (MAX78000-FTHR) with an AI accelerator to evaluate its performance in terms of inference time and energy consumption. The neural network is designed to classify an audio signal from different acoustic environments, such as a forest path or an office. For the implementation, a MAX78000-specific workflow was used, enabling training in a Python environment with PyTorch, followed by a conversion into C code. The neural network was trained using the TUT Acoustic Scenes 2017 dataset and achieved an accuracy of 70% after quantization, which decreased to 57% due to the audio preprocessing of the deployed model. The inference time is comparable to that of a GPU, while the energy consumption is significantly lower. The results demonstrate that the implementation of AI algorithms on embedded systems can be a useful approach, as they offer energy-efficient processing while maintaining similar performance.

Zusammenfassung

In dieser Arbeit wurde ein faltendes neuronales Netz trainiert und auf einem eingebetteten System (MAX78000-FTHR) mit KI-Beschleuniger implementiert, um diesen im Hinblick auf Performance, Inferenz-Zeit und Energieverbrauch zu evaluieren. Das neuronale Netz soll dabei ein Audiosignal einer Geräuschkulisse, wie einem Waldweg oder einem Stadtzentrum, zuordnen. Für die Implementierung des neuronalen Netzes wurde ein MAX78000-spezifischer Workflow genutzt, der ein Training in einer Python-Umgebung mit PyTorch sowie eine anschließende Umwandlung in C-Code ermöglicht. Das neuronale Netz wurde mit Hilfe des Datensatzes TUT-Acoustic-Scenes-2017 trainiert und erreichte nach der Quantisierung eine Genauigkeit von 70%, wobei die Genauigkeit durch die Audiorverarbeitung des implementierten Modells auf 57% gesenkt wird. Die Inferenzzeit ist vergleichbar mit der einer GPU, während der Energieverbrauch dabei wesentlich geringer ausfällt. Die Arbeit zeigt, dass die Implementierung von KI-Algorithmen sinnvoll ist, da sie energiesparender sind und das Potenzial haben, eine ähnliche Performance zu erreichen.

Inhaltsverzeichnis

Abkürzungsverzeichnis	VIII
Abbildungsverzeichnis	IX
Tabellenverzeichnis	X
Symbolverzeichnis	XI
1 Einleitung	1
1.1 Motivation	1
1.2 Zielsetzung	2
1.3 Inhaltlicher Überblick	2
2 Stand der Technik	3
2.1 Edge-AI	3
2.1.1 STM32 N6	4
2.1.2 Jetson Orin Nano Super	5
2.1.3 MAX78000	6
2.1.4 Vergleich	6
2.2 Acoustic Scene Detection	8
3 Grundlagen	9
3.1 Neuronale Netze	9
3.1.1 Einfaches Neuronales Netz	9
3.1.2 Faltende neuronale Netze	10
3.1.3 Zusätzliche Operationen	12
3.1.4 Hyperparameter	15
3.2 Mel Frequency Cepstral Coefficients	17
3.3 CNN Beschleuniger	19

4 Umsetzung	21
4.1 Workflow	21
4.2 Datensatz	23
4.2.1 Datenverteilung	23
4.2.2 Ortsabhängigkeit	23
4.2.3 Anpassung des Datensatzes	24
4.2.4 Splits	24
4.3 Audio Preprocessing	24
4.3.1 Abtastfrequenz	25
4.3.2 MFCC	26
4.4 Training	26
4.4.1 Basismodell	26
4.4.2 Limitierungen des MAX78000	27
4.4.3 Hyperparameter Optimierung	28
4.4.4 Quantization Aware Training (QAT)	32
4.5 Quantization	33
4.5.1 Evaluation	33
4.6 Ai8x-Synthese	34
4.6.1 YAML-Network-Description	34
4.7 Implementierung auf eingebettetem System	36
4.7.1 Programmablauf	36
4.8 MAX78000	38
4.8.1 Audio Codec	38
4.8.2 CMSIS DSP Library	39
4.8.3 Daten für CNN laden	41
4.9 Metriken für Evaluierung	41
4.9.1 Performance des neuronalen Netzes	41
4.9.2 Inference-Time Messung	42
4.9.3 Energieverbrauch	43
5 Ergebnisse und Auswertung	45
5.1 Genauigkeit des Models	45
5.2 Inferenz-Zeit	47
5.3 Energieverbrauch	48
5.4 Auswertung	50
6 Ausblick	52

7 Anhang	53
7.1 Nutzung von Generativer KI	53
7.2 Bestimmung der Inferenzzeit der GPU	54
7.3 Messungen der Berechnungszeiten MAX78000	56
Literaturverzeichnis	58

Abkürzungsverzeichnis

ML maschinelles Lernen

KI künstliche Intelligenz

GPU Graphics Processing Unit

CNN Convolutional Neural Network

MFCC Mel-Frequency-Cepstral-Coefficients

DFT Discrete Fourier Transform

FFT Fast Fourier Transform

ReLU Rectified Linear Unit

GPU Graphics Processing Unit

ASC Acoustic Scene Detection

SED Sound Event Detection

NPU Neural Processing Unit

PMIC Power Management Integrated Circuit

DMA Direct Memory Access

MAC Multiply-Accumulate

DCT Discrete Cosine Transform

SED Sound Event Detection

QAT Quantization Aware Trainingg

Abbildungsverzeichnis

2.1	STM32 Nucleo-144 mit STM-N6 MCU	4
2.2	Jetson Orin Nano Super Developement Kit	5
2.3	MAX78000 FTHR	6
3.1	Bestandteile eines neuronalen Netzes	9
3.2	Aufbau eines faltenden neuronalen Netzes	10
3.3	2-Dimensionale Faltung	11
3.4	Visualisierung von 2D Max-Pooling	13
3.5	Veranschaulichung der Lernrate	16
3.6	Implementierung der Mel-Frequency-Cepstral-Coefficients (MFCC)[14]	17
3.7	Mel-Skalierung	18
3.8	MFCC-Beispiel	19
3.9	CNN-Beschleuniger	20
4.1	Workflow	22
4.2	Durschnittliches FFT-Spektrum	25
4.3	Trainingsergebnisse	32
4.4	Programmablaufplan	36
4.5	Datenladen für CNN-Beschleuniger	41
4.6	Confusion Matrix Beispiel	42
4.7	Messaufbau	44
5.1	Stromverbrauch während der Inferenz	49
7.1	Messung der Verarbeitungszeit eines MFCC-Fensters	56
7.2	Messung der Berechnungszeit der load_input Funktion	56
7.3	Messung der Berechnungszeit des CNNs	57

Tabellenverzeichnis

2.1	Vergleich der vorgestellten Systeme	7
4.1	Aufbau des Convolutional Neural Network (CNN) als Basismodell	27
4.2	Netzwerk Architektur V6	29
4.3	Trainingsergebnisse der Netzwerkarchitektur V6	29
4.4	Netzwerkarchitektur V5	30
4.5	Trainingsergebnisse der V5-Netzwerkarchitektur	31
4.6	Angepasstes Modell V5_1	31
4.7	Autogenerierte Funktionen	34
4.8	Parameter Audio Codec	38
4.9	Vergleich verschiedener MFCC-Implementierungen	40
5.1	Confusion-Matrix nach Quantisierung (normalisiert)	45
5.2	Confusion-Matrix des Endergebnis	46
5.3	Performance-Vergleich	47
5.4	Berechnungszeiten von MAX78000 und GPU	48
5.5	Energieverbrauch während der Inferenz	49

Symbolverzeichnis

1 Einleitung

1.1 Motivation

Mit den wachsenden Möglichkeiten, maschinelles Lernen (ML) -Algorithmen und künstliche Intelligenz (KI) in den Alltag zu integrieren, besteht ein großes Interesse an Edge-Computing. Viele Anwendungen auf Basis von KI befinden sich auf Servern, auf die über eine Cloud zugegriffen werden kann. Die zu verarbeitenden Daten werden über das Internet an die Cloud-Server gesendet. Diese Methodik birgt mehrere Nachteile:

1. Es ist eine Internetverbindung zur Cloud notwendig, die es möglich macht, die Rohdaten zu übertragen, was abhängig von der geographischen Lage schwierig sein kann.
2. Die Rechenzentren, in welchen die Daten verarbeitet werden, sind häufig in anderen Ländern, wo andere Gesetze zum Datenschutz gelten, was für regionale Unternehmen ein Problem darstellen kann, da sie nicht die Gesetze zum Datenschutz für die Endverbraucher gewährleisten können.
3. Die Graphics-Processing-Units (GPU) oder Tensor-Processing-Units (TPU), die in den Rechenzentren benutzt werden, verbrauchen viel Energie [1].

Daher ist es sinnvoll, die Daten direkt an der Datenquelle zu verarbeiten, weshalb ML-Anwendungen zunehmend auf eingebetteten Systemen realisiert werden. Dieses Gebiet nennt sich Edge-AI. Mögliche Anwendungsbereiche von Edge-AI sind Robotik, intelligente Kameras, industrielle Automatisierung und medizinische Diagnostik. Eines der eingebetteten Systeme, welches für ML-Anwendungen entwickelt wurde, ist der MAX78000, der in dieser Arbeit benutzt wird.

1.2 Zielsetzung

Ziel der Arbeit ist es, auf dem MAX78000 eine Geräuschkulissen-Erkennung zu implementieren. Mit Geräuschkulissen-Erkennung ist die Zuordnung einer Aufnahme zu einer Geräuschkulisse wie Park, Bibliothek oder Waldweg gemeint. Diese Identifizierung erfolgt durch eine KI, die trainiert werden muss. Der MAX78000 besitzt einen CNN-Beschleuniger, eine spezielle Hardware, die für die Berechnung von neuronalen Netzen entwickelt wurde. Die Geräuschkulissen-Erkennung wird auf dem Beschleuniger implementiert. Dazu soll der MAX78000 mit einer herkömmlichen Implementierung in Python verglichen werden. Die Vergleichskriterien sind:

1. Inference-Zeit
2. Energieverbrauch während der Datenverarbeitung
3. Genauigkeit der Geräuschkulissen-Erkennung

1.3 Inhaltlicher Überblick

Inhaltlich wird in Kapitel 2 „Stand der Technik“ ein Ist-Zustand erörtert, der als Basis für die Arbeit dient. Anschließend werden die für die Arbeit relevanten theoretischen Grundlagen erklärt. Im Hauptteil wird auf den Workflow eingegangen, der für die Implementierung eines CNN auf dem MAX78000 notwendig ist. Dazu gehören alle Schritte des Trainings des Neuronalen Netzes in PyTorch in der Programmiersprache Python sowie der Umsetzung auf dem eingebetteten System in der Programmiersprache C. Die Ergebnisse der Evaluation werden in Kapitel 5 ausgewertet. Zuletzt wird die Arbeit zusammengefasst und mögliche Verbesserungen werden erläutert.

Alle genutzten Dateien, Skripte und Programmcodes sind in dem Github-Repository ASC-MAX78000_FTHR [2] hinterlegt.

2 Stand der Technik

2.1 Edge-AI

Die Landschaft der künstlichen Intelligenz und im Speziellen der Edge-AI-Anwendungen entwickelt sich kontinuierlich. Ein wachsendes Interesse an Edge-AI-Lösungen sorgt für die Entwicklung von eingebetteten Systemen für KI-Anwendungen[3]. Hauptanwendungen von Edge-AI sind derzeit :

- Smart-Home [4]
- Robotics [4]
- Spracherkennung [4]
- Acoustic Event-Detection [4]
- Autonome Systeme [4]

Die aktuellen eingebetteten Systeme, die für diese Arbeit relevant sind, werden hier vorgestellt.

2.1.1 STM32 N6

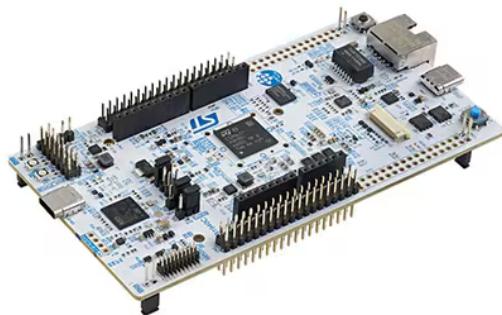


Abbildung 2.1: STM32 Nucleo-144 mit STM-N6 MCU
[5]

Das neueste Edge-KI-System im Portfolio von STMicroelectronics (STM) ist der STM32-N6, eine leistungsstarke Mikrocontroller-Plattform, die speziell für KI- und Bildverarbeitungsanwendungen im Embedded-Bereich entwickelt wurde. Der STM32-N6 kombiniert eine Neural Processing Unit (NPU) mit einem Arm Cortex-M55 Prozessor und integriert eine Bildverarbeitungseinheit (Image Signal Processor, ISP) zur Echtzeitverarbeitung von Sensordaten.

Die NPU, ein von STMicroelectronics entwickelter ST-Neural-Art-Accelerator, ist für die effiziente Ausführung von KI-Algorithmen optimiert und erreicht eine maximale Rechenleistung von 0,6 Tera-Operationen pro Sekunde (TOPS) bei einer Energieeffizienz von 3 TOPS/W [6]. Ergänzt wird diese Architektur durch den Arm Cortex-M55, der mit einer maximalen Taktrate von 800 MHz betrieben wird und für allgemeine Steuerungs- und Verarbeitungsaufgaben zuständig ist.

Zusätzlich verfügt der STM32-N6 über einen integrierten Image Signal Processor (ISP), der für die Echtzeitverarbeitung von Bild- und Sensordaten optimiert ist. Diese Einheit ermöglicht die Verarbeitung von Kamerabildern mit einer Auflösung von bis zu 5 Megapixeln bei 30 FPS und unterstützt Bildverbesserungen, Rauschreduktion sowie H.264-Hardware-Encodierung und JPEG-Kompression. Damit eignet sich der STM32-N6 insbesondere für Computer-Vision-Anwendungen, einschließlich Objekterkennung, automatisierte Überwachung und Edge-basierte Bildverarbeitung durch KI.

2.1.2 Jetson Orin Nano Super

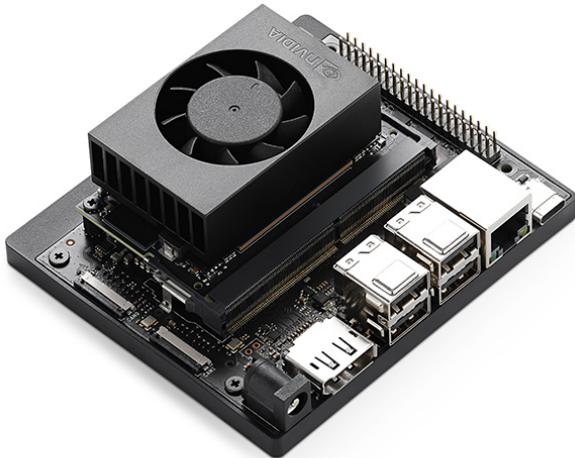


Abbildung 2.2: Jetson Orin Nano Super Developement Kit
[7]

Der Jetson Orin Nano Super stellt ebenfalls eine leistungsstarke Plattform für Edge-AI-Anwendungen dar. Diese Plattform wurde speziell für rechenintensive Anwendungen im Bereich der KI entwickelt und ist in der Lage, komplexe KI-Modelle, darunter Large Language Models (LLMs), effizient auszuführen.

Der Jetson Orin Nano basiert auf einem Linux-Betriebssystem und unterstützt die Nutzung externer SSD-Speicher zur Erweiterung der Speicherkapazität. Zudem verfügt das System über 8 Gigabyte Arbeitsspeicher, wodurch es eine hohe Rechenleistung für datenintensive Anwendungen bereitstellt.

In Bezug auf die Leistungsfähigkeit weist der Jetson Orin Nano Super eine Rechenkapazität auf, die mit einem Desktop-PC vergleichbar ist. Gleichzeitig zeichnet sich das System durch einen deutlich reduzierten Energieverbrauch von 7 bis 25 Watt sowie eine kompakte Bauweise aus.

2.1.3 MAX78000

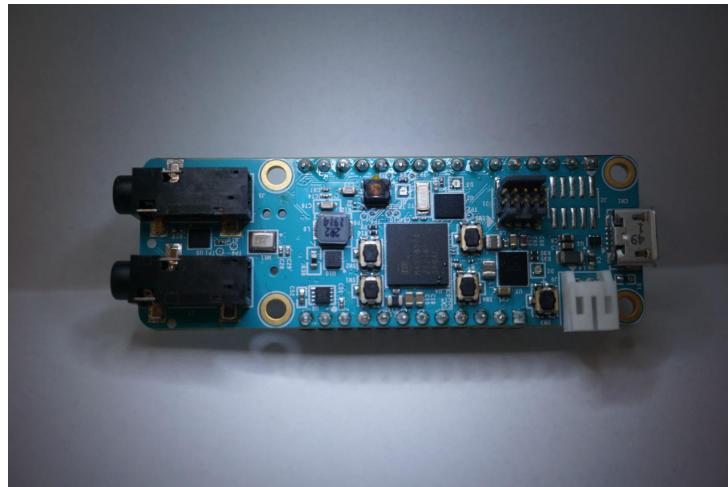


Abbildung 2.3: MAX78000 FTHR

Das in dieser Arbeit verwendete eingebettete System ist der MAX78000, welcher von Analog Devices für Low-Power KI-Anwendungen entwickelt wurde. Ein besonderes Merkmal ist seine Energieeffizienz. Der MAX78000 ist für die batteriebetriebene Anwendung optimiert, weshalb er ein Power Management Integrated Circuit (PMIC) besitzt, welcher den Betrieb mit Akkus und deren Ladung ermöglicht. Der MAX78000 besteht aus einem CNN-Beschleuniger als NPU und einem ARM-Cortex-M4 Prozessor. Der CNN-Beschleuniger besitzt einen Speicher von 442 Kbyte für Gewichte und unterstützt Netzwerkarchitekturen mit bis zu 64 Ebenen. Der ARM-Cortex-M4 Prozessor kann mit bis zu 100 MHz getaktet werden und übernimmt die restlichen Verarbeitungsaufgaben, die in der Anwendung notwendig sind.

Der Nachfolger des MAX78000, der MAX78002, ist bereits auf dem Markt verfügbar. Dieser verfügt über einen dreifach erhöhten Speicher für Gewichtsdaten (1,3 MByte) sowie eine höhere Taktfrequenz von 120 MHz, während der Stromverbrauch nur geringfügig erhöht ist. Da der MAX78002 in dieser Arbeit nicht verwendet wird und hinsichtlich seiner Eigenschaften im Bereich Edge-AI mit dem MAX78000 vergleichbar ist, wird er im folgenden Vergleich nicht weiter berücksichtigt

2.1.4 Vergleich

In diesem Abschnitt werden die relevantesten Parameter der zuvor vorgestellten eingebetteten Systeme verglichen. Durch die Zielstellung und Anwendung eines ML-Algorithmus

entstehen Anforderungen wie Genauigkeit, Latenz, Energieverbrauch, benötigter Speicher für ML-Algorithmus, Preis und Anbindung zu Peripherie.

Die folgende Tabelle 2.1 zeigt die wichtigsten Parameter auf.

	MAX78000	STM32-N6	Jetson Orin Nano Super
Prozessor	ARM-Cortex-M4	ARM-Cortex M55	ARM-Cortex A78AE
Taktfrequenz	100 Mhz	800 Mhz	1984 Mhz
CNN Speicher	442 KB	4,2 MB	erweiterbar
Operationen/s	nicht angegeben	600 GOPS	67 TOPS
Gewichtstypen	1,2,4,8 Bit	nicht angegeben	All
Energieverbrauch	25 - 50 mW	bis 200mW	7 - 25 W
Kosten (Dev. Kit)	210,11€	174,66€	249,99€

Tabelle 2.1: Vergleich der vorgestellten Systeme

Im Vergleich der spezifischen Eigenschaften der untersuchten Geräte lassen sich deren potenzielle Anwendungsbereiche klar ableiten. Der Jetson Orin Nano Super bietet im Vergleich die höchste Rechenleistung und Flexibilität, da er in der Lage ist, umfangreiche KI-Modelle mit mehreren Milliarden Parametern durch erweiterbare Speicher wie SSDs zu implementieren. Diese Leistungsfähigkeit und Komplexität gehen jedoch mit einem signifikant höheren Stromverbrauch einher, wodurch dieses System vor allem für leistungshungrige Anwendungen geeignet ist, bei denen Energieeffizienz keine Priorität darstellt.

Der STM32-N6 ermöglicht durch seinen 4,2 MB Speicher die Implementierung komplexerer Modelle. Obwohl die Speichermöglichkeiten des STM32-N6 limitiert sind, lassen sich die meisten KI-Modelle auf dem Mikrocontroller umsetzen, und das bei einem stark verringerten Stromverbrauch.

Der MAX78000 ist aufgrund seines geringen Stromverbrauchs besonders für Anwendungen geeignet, bei denen eine hohe Energieeffizienz erforderlich ist. Allerdings ist die Größe der realisierbaren neuronalen Netzwerke durch die begrenzte Speicherkapazität des Systems eingeschränkt.

Für die Implementierung eines Acoustic Scene Detection Systems ist der MAX78000 aufgrund seiner Energieeffizienz und der ausreichenden Kapazität für kleinere KI-Modelle gut geeignet. Der MAX78000 bietet genug Speicher für die Parameter der in dieser Arbeit genutzten Modelle und eignet sich für portable Anwendungen.

2.2 Acoustic Scene Detection

Das Ziel der Acoustic Scene Detection ist die Zuordnung von Audiomaterial zu verschiedenen Szenen. Als Eingangssignal wird eine Aufnahme von einem spezifischen zu identifizierenden Ort verwendet. Die Hintergrundgeräusche der Aufnahme enthalten Informationen zu dem Ort der Aufnahme und sollen von der Acoustic Scene Detection identifiziert werden. Die Herausforderung bei dieser Aufgabe besteht darin, dass in einer Geräuschkulisse viele Ereignisse stattfinden, die in verschiedenen Frequenzbereichen zu finden sind, und nicht präsent sind. Menschen können die Hintergrundgeräusche sehr gut zuordnen, für KI ähneln sie eher einem Rauschen. In dieser Thematik gibt es auch die Sound Event Detection (SED), welche sich mit der Erkennung von spezifischen Geräuschen befasst, wie zum Beispiel einem Pistolenschuss oder einer Türklingel.

Während in dieser Arbeit der Datensatz TUT-Acoustic-Scenes-2017 genutzt wird, sind neuere und größere öffentlich zugängliche Datensätze vorhanden, wie zum Beispiel TAU Urban Acoustic Scenes 2022 Mobile Development-Dataset [8], welches für aktuelle Arbeiten verwendet wird. Dieser Datensatz beinhaltet 64 Stunden Audiomaterial, welches zu 10 verschiedenen akustischen Szenen zugeordnet ist. Das Audiomaterial wurde in 12 verschiedenen europäischen Städten mit 4 verschiedenen Aufnahmegeräten aufgenommen.

Die DCASE Community (Detection and Classification of Acoustic Scenes and Events) veranstaltet jährlich einen Wettbewerb, bei dem KI-Algorithmen der Acoustic Scene Detection eingereicht werden können und die besten Ergebnisse vorgestellt werden. Dieser Wettbewerb spiegelt den Stand der Technik wider, da die Ziele an den Stand der Technik angepasst werden. Aufgaben des diesjährigen Wettbewerbs, (DCASE2025 Challenge[9]) sind unter anderem eine Acoustic-Scene-Detection mit einer geringen Komplexität und Modelle zur Lokalisierung von Acoustic Events in Videomaterial mit Stereo-Audio. Die aktuellen Modelle verwenden Ensemble-CNNs, um akustische Szenen zu erkennen.

Auch wenn die Implementierung der Acoustic Scene Detection einem akademischen Zweck dient, gibt es verschiedene Anwendungsmöglichkeiten für Acoustic Scene Detection, wie zum Beispiel die Anpassung von Hörgeräten an die Umgebung. Acoustic Event Detection kann zum Beispiel auch verwendet werden, um Fehlfunktionen von Geräten durch Geräusche zu erkennen.

3 Grundlagen

3.1 Neuronale Netze

3.1.1 Einfaches Neuronales Netz

Neuronale Netze sind ein ML-Konzept, welches dem menschlichen Gehirn nachempfunden ist. In neuronalen Netzen sind viele Neuronen durch gewichtete Verbindungen miteinander verknüpft. Die Verbindungen sind gerichtet, weshalb auch von einem „Feedforward-Netzwerk“ gesprochen wird. In Abbildung 3.1 ist ein einfaches neuronales Netz dargestellt.

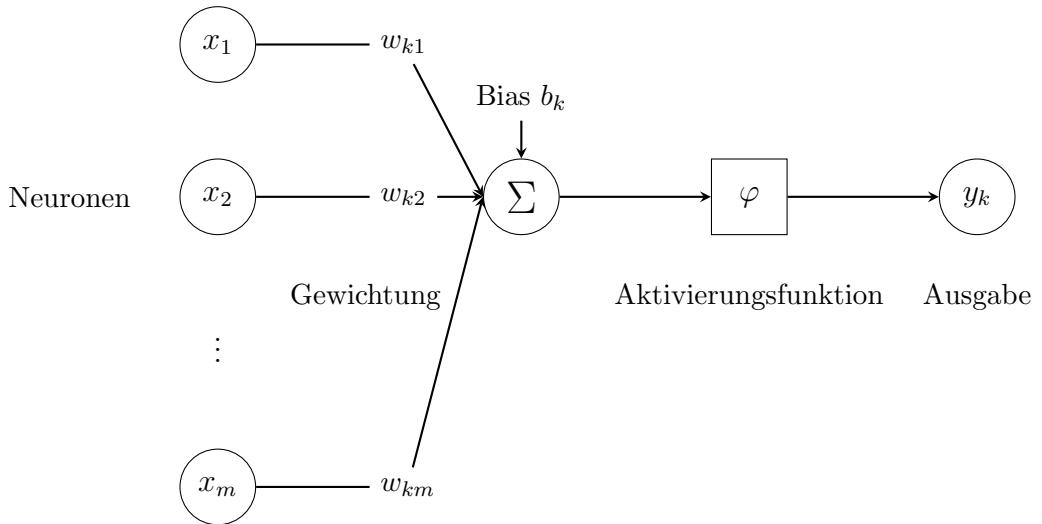


Abbildung 3.1: Bestandteile eines neuronalen Netzes

Alle Eingangssignale werden mit den dazugehörigen Gewichten multipliziert und anschließend mit dem Bias, einer Konstanten, summiert. Anschließend wird auf die Summe eine Aktivierungsfunktion angewendet. Eine typische Aktivierungsfunktion ist Rectified Linear Unit (ReLU) mit folgender Formel:

$$f(x) = \begin{cases} x & \text{wenn } x \geq 0 \\ 0 & \text{wenn } x < 0 \end{cases} [10] \quad (3.1)$$

Durch die Aktivierungsfunktion wird eine nichtlineare Operation in das Netz mit eingebbracht, was neuronale Netze zu nichtlinearen Systemen macht. Ohne die Aktivierungsfunktion wäre ein neuronales Netz eine einfache Matrixmultiplikation.

Neuronale Netze werden im supervised learning mit Hilfe von Datensätzen trainiert. Die Gewichte und Bias sind der Hauptbestandteil des Trainings, da diese verändert werden, um das neuronale Netz auf den Datensatz anzupassen. Bei dem Training wird ein Sample des Datensatzes geladen und der Output des Datensatzes mit dem gewünschten Wert verglichen. Dabei wird der Fehler des Netzes berechnet. Die Gewichte und der Bias werden abhängig von dem Einfluss auf den Ausgangsfehler verändert, um den Fehler zu minimieren. Eine detaillierte Erläuterung zu Backpropagation und Gradient Descent befindet sich im Buch Deep Learning von Ian Goodfellow, Yoshua Bengio und Aaron Courville, Kapitel 6.5 [11].

3.1.2 Faltende neuronale Netze

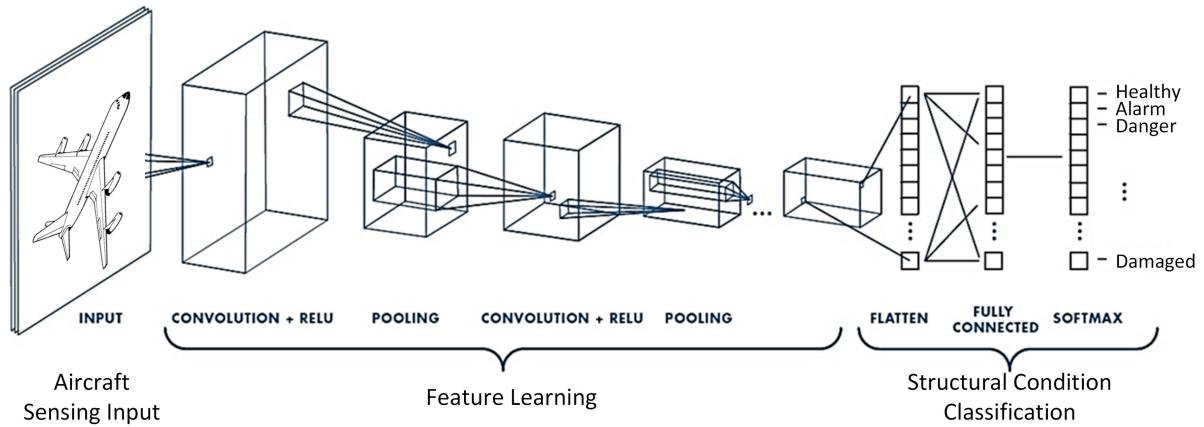


Abbildung 3.2: Aufbau eines faltenden neuronalen Netzes

[12]

Faltende Neuronale Netze oder auch CNN gehören zu der Gruppe der neuronalen Netze. Der Hauptunterschied zu anderen neuronalen Netzen besteht in der Faltungsoperation. Alle anderen Operationen sind identisch. Die diskrete Faltungsoperation kann durch fol-

gende Formel dargestellt werden:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)[11] \quad (3.2)$$

In dem Fall der faltenden Neuronalen Netze werden die Neuronen nicht mit allen Neuronen der darauf folgenden Schicht verbunden, sondern mit einem Kernel gefaltet. In vielen ML-Bibliotheken wird die Cross-Correlation anstatt der Faltung verwendet und trotzdem Faltung genannt. Die Formel ist:

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n)[11] \quad (3.3)$$

Zu beachten ist, dass in der Regel 2-dimensionale Informationen verarbeitet werden. Das Ergebnis der Faltung nennt man "Feature Map". Die Feature Map ist ein 2-dimensionaler Vektor, den man auch als Bild der abstrahierten Informationen interpretieren kann. Durch die Faltungsoperation ist die Anzahl der Gewichte geringer als bei gewöhnlichen neuronalen Netzen. Ein Pixel in der Feature Map ist nicht mit allen Pixeln der Schicht zuvor verbunden, sondern mit einem Pixel und den benachbarten Pixeln, abhängig von der Kernel-Größe. Dadurch wird die Anzahl der veränderlichen Parameter geringer. Auch die Größe des Ausgangsvektors verringert sich, da der Kernel nicht über den Rand des Bildes hinausgeht. Die folgende Abbildung veranschaulicht die Operation.

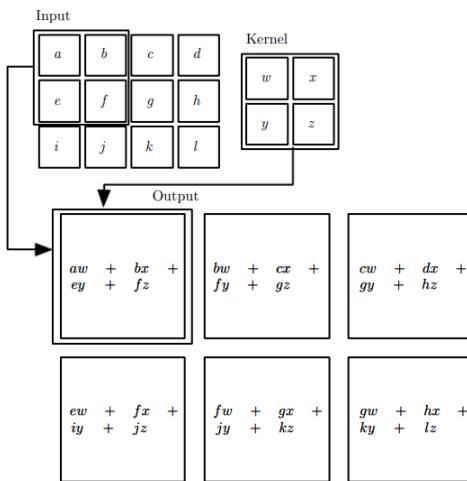


Abbildung 3.3: 2-Dimensionale Faltung
[11]

In dieser Arbeit wird die eindimensionale Faltung verwendet. Der einzige Unterschied ist jener, dass ein eindimensionaler Kernel verwendet wird. Für Bilderkennung ist dieser Ansatz zwar eher untypisch, aber ein Spektrogramm ist kein typisches Bild. Auf der X-Achse befinden sich Zeitinformationen und auf der Y-Achse die Frequenzinformationen. Durch eine eindimensionale Faltung werden die Informationen reihenweise entlang der X-Achse gefaltet und Merkmale extrahiert.

3.1.3 Zusätzliche Operationen

Faltende neuronale Netze verwenden in der Regel nicht nur Faltungsschichten für die Klassifizierung, sondern auch andere Operationen, die entweder der Optimierung des Trainings oder der Merkmalsextraktion dienen. Die in dieser Arbeit genutzten Operationen werden im Folgenden erläutert.

Batch Normalization

Durch eine gleichzeitige Aktualisierung der Gewichte können Effekte auftreten, die entweder zu einem sehr langsamen oder divergierenden Training führen [11]. Die Batch Normalization dient der Optimierung und Stabilisierung des Trainings, indem alle Eingänge der Schicht innerhalb eines Trainingsbatches \mathbf{H} zu einem gemeinsamen Mittelwert μ und der Standardabweichung des Trainingsbatches σ normalisiert werden. Der mathematische Zusammenhang ist :

$$\mathbf{H}' = \frac{\mathbf{H} - \mu}{\sigma} [11] \quad (3.4)$$

Der Effekt der Batch Normalization in verschiedenen Schichten besteht darin, dass die statistischen Eigenschaften der Schichten während des Trainings konstant bleiben und die Gewichte des CNN nicht extrem schwanken.

Pooling

Die Pooling-Operation (engl. „to pool“ = zusammenfassen) fasst Teile der Eingangsschicht zusammen. Es werden Ausschnitte der Eingangsschicht analysiert und abhängig von der spezifischen Pooling-Operation zusammengefasst. Grundsätzlich gibt es zwei Pooling-Operationen, zum einen Average Pooling und zum anderen das in dieser Arbeit verwendete Max Pooling. Dort wird nur der höchste in einem Ausschnitt vorkommende Wert im Ausgang gespeichert, während alle anderen Werte nicht mit aufgenommen werden. In der folgenden Abbildung 3.4 ist die zweidimensionale Max-Pooling-Operation dargestellt:

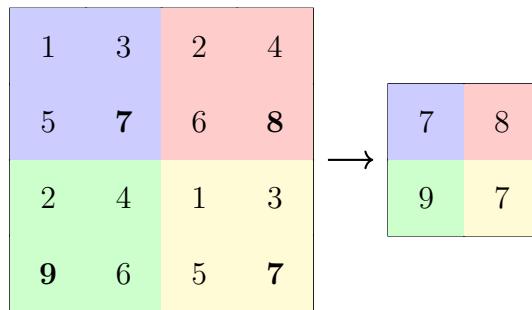


Abbildung 3.4: Visualisierung von 2D Max-Pooling

Obwohl sich in diesem Fall die Größe der Schicht halbiert hat, wird die Erkennung robust. Während durch Faltungsschichten Merkmale extrahiert werden, sollen diese durch Pooling-Layer verallgemeinert werden. Kleine Verschiebungen in der Eingangsschicht wirken sich nicht stark auf den Ausgang aus, da die Max-Pooling-Operation sich durch eine Veränderung des höchsten Wertes und nicht durch dessen Position verändert. Die Kombination aus Faltung und Pooling ist ein guter Ausgangspunkt für gute neuronale Netze.

Dropout

Faltende neuronale Netze neigen zu Overfitting. Overfitting beschreibt die Überanpassung der Parameter eines Modells an den Trainingsdatensatz. In diesem Fall erkennt das CNN die Trainingsdaten sehr gut, indem es sie – umgangssprachlich gesagt – „auswendig lernt“. Jedoch ist das Netzwerk dann nicht in der Lage, neue, unbekannte Daten mit der gleichen Genauigkeit zu klassifizieren, da es keine generalisierbaren Muster abstrahiert.

Es gibt verschiedene Ursachen für Overfitting, wie z. B. eine zu hohe Modellkomplexität, ein Mangel an Trainingsdaten oder eine unzureichende Regularisierung. Ebenso gibt es

eine Vielzahl von Methoden, um Overfitting zu verhindern. Eine dieser Methoden ist die Verwendung von Dropout.

Beim Einsatz von Dropout wird jedes Neuron einer Schicht mit einer vorher festgelegten Wahrscheinlichkeit deaktiviert. Diese Deaktivierung erfolgt während einer Trainingsepoch zufällig. Dadurch wird in jeder Epoche ein anderer Teil des neuronalen Netzwerks trainiert. Diese zufällige Deaktivierung reduziert die Abhängigkeit des Netzwerks von spezifischen Neuronen und erschwert das Auswendiglernen der Trainingsdaten, was die Robustheit erhöht und die Anfälligkeit für Überanpassung verringert. Dropout wird nur während des Trainings aktiviert und reduziert nicht die Menge der aktivierten Neuronen während der Anwendung.

Fully Connected Layer

Wie in Abbildung 3.2 zu sehen ist, werden die extrahierten Merkmale durch ein Fully Connected Layer weiterverarbeitet. Für die Verarbeitung wird der Merkmalsvektor in einen eindimensionalen Vektor umgeformt. Jedes Neuron wird anschließend mit jedem Neuron der nächsten Schicht verknüpft, wodurch ein Fully Connected Layer entsteht. Der Output des Fully Connected Layers ist ein eindimensionaler Vektor, welcher die finalen Klassen beinhaltet. Dementsprechend ist jedes extrahierte Merkmal mit einer Gewichtung ins Ergebnis mit einzbezogen. Die Klasse, die den höchsten Wert besitzt, stellt die Prediction des neuronalen Netzes dar.

Softmax Aktivierung

Der letzte Schritt in neuronalen Netzen bei Klassifikationsaufgaben ist die Softmax-Aktivierung, die die Rohwerte der Ausgabeschicht in Wahrscheinlichkeiten umwandelt. Diese Wahrscheinlichkeiten stellen die Confidence-Werte der Vorhersage für jede Klasse dar. Die Softmax-Funktion, die auf den Vektor $\mathbf{z} = [z_1, z_2, \dots, z_K]$ angewandt wird, ist definiert als:

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^K \exp(z_j)} [11] \quad (3.5)$$

Die Softmax-Aktivierung transformiert den Eingabevektor in einen Wahrscheinlichkeitsvektor, bei dem alle Werte im Bereich [0,1] liegen und sich zu 1 summieren. Dies ermöglicht eine intuitive Interpretation der Werte, da sie die Wahrscheinlichkeit angeben, mit der die Eingabe zu den jeweiligen Klassen gehört.

3.1.4 Hyperparameter

In KI-Algorithmen gibt es viele Parameter, die Einfluss auf den Algorithmus haben. Dabei wird zwischen Parametern und Hyperparametern unterschieden. Mit Parametern sind jene gemeint, die durch das Training des Algorithmus verändert werden, wie zum Beispiel Gewichte in neuronalen Netzen. Hyperparameter haben einen Einfluss auf das Training an sich und müssen vor dem Training eines Netzes festgelegt werden. Die Hyperparameter gilt es zu optimieren, damit das Training die bestmöglichen Ergebnisse erzielt. Grundsätzlich gibt es zwei Methoden zur Optimierung der Hyperparameter. Die eine Methode ist ein manuelles Verfahren, bei dem eine Vielzahl an Trainingsdurchläufen durchgeführt und nach jedem Durchlauf die Hyperparameter manuell angepasst werden. Die Anpassung der Hyperparameter basiert auf Erfahrungswerten und sogenannten Best-Practice-Regeln, da der Einfluss von Hyperparametern nicht immer vorhersehbar ist. Die zweite Methode zur Optimierung der Hyperparameter ist automatisch. Die Hyperparameter werden von einem Algorithmus, wie Random Search, verändert und durch viele Trainingsiterationen angepasst und optimiert. Diese Algorithmen können die Hyperparameter-Optimierung erleichtern, haben aber einen erhöhten Rechenaufwand und besitzen selbst auch Hyperparameter, die sich auf die Optimierung auswirken. Die manuelle Hyperparameter-Optimierung ist erfolgsbringend, wenn eine Person ein gutes Verständnis für die Einflüsse von Hyperparametern hat und es einen guten Startpunkt durch vergleichbare Arbeiten gibt [11].

Die in dieser Arbeit veränderten Hyperparameter sind:

Preprocessing	Netzwerk-Architektur	Batch-Größe	Lernrate
---------------	----------------------	-------------	----------

Die Vorverarbeitung (engl. Preprocessing) der Informationen, wie zum Beispiel Abtastfrequenz oder die Fenstergröße (siehe 3.2), kann sich stark auf das Training auswirken. Für die Optimierung ist es notwendig, die Informationen zu charakterisieren und einzuschätzen, welche Informationsmenge für das Ziel ausreichend ist und in welcher Form diese dem neuronalen Netz gegeben werden sollen. In dieser Arbeit könnte zum Beispiel das Audiosignal direkt ohne Transformation dem CNN übergeben werden, was erhebliche Auswirkungen hat.

Die Netzwerkarchitektur ist ebenfalls ein Hyperparameter, den es zu optimieren gilt. Wenn das Netz nicht in der Lage ist, an den Trainingsdaten zu lernen, ist die Architektur und die dazugehörige Menge der Parameter möglicherweise zu klein. Falls das Trainingsmodell überangepasst (engl. Overfitting) ist, kann eine Reduktion der Parameter helfen. Außerdem können zusätzliche Methoden, wie zum Beispiel Dropout oder Batch Normalization, angewendet werden, die ebenfalls die Trainingsergebnisse verbessern.

Die Batch-Größe ist die Menge der Beispiele, die aus einem Datensatz für eine Trainingsepoch genutzt werden. Während große Batch-Größen zu Overfitting neigen und eine größere CPU/GPU Auslastung mit sich bringen, ist das Training stabiler und die Gradientenberechnung stabiler. Kleinere Batch-Größen fügen dem Training eine Rauschcharakteristik durch schwankende Gradienten hinzu, können aber Overfitting reduzieren.

Die Lernrate bestimmt die Steilheit, mit der die Gewichte angepasst werden. Das Ziel ist es, durch die Anpassung der Gewichte das globale Minimum der Fehlerfunktion zu erreichen. Höhere Lernraten können sich eignen, um lokale Minima der Gradienten zu überwinden, um eine bessere Performance zu erreichen, können aber auch die Konvergenz des Trainings verhindern. Geringere Lernraten haben die umgekehrten Eigenschaften. Sie konvergieren schneller, können aber lokale Minima nicht überwinden. Die folgende Abbildung 3.5 veranschaulicht diesen Effekt.

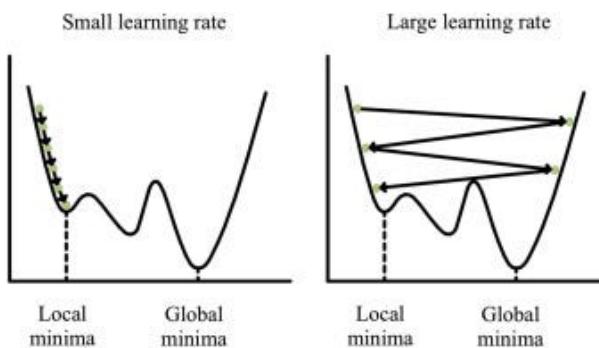


Abbildung 3.5: Veranschaulichung der Lernrate
[13]

Die Anpassung eines Hyperparameters erfolgt im Zusammenspiel mit den anderen Hyperparametern. Zum Beispiel kann eine kleine Batch-Größe gepaart mit einer sehr kleinen Lernrate zu einer Verbesserung des Trainings führen.

3.2 Mel Frequency Cepstral Coefficients

Für viele ML-Anwendungen, vor allem in der Spracherkennung, wird das zu verarbeitende Audiosignal von einer Zeitfolge in ein Spektrogramm transformiert. Obwohl es möglich ist, das reine Audiosignal als Input für den ML-Algorithmus zu verwenden, gibt es Gründe, weshalb eine Transformation in den spektralen Bildbereich sinnvoll ist. Zum einen besitzt das reine Audiosignal eine Menge von Informationen. Die Informationen beinhalten die Veränderung der Amplitude über der Zeit, besitzen aber keine direkten Informationen über das Frequenzverhalten des Signals. Menschen sind in der Lage, durch ihr Gehör das Frequenzverhalten wahrzunehmen, was eine große Rolle bei der Identifizierung von Lauten spielt. Beispielsweise ist bei der Spracherkennung nicht die Tonhöhe, sondern die Veränderung der Formanten relevant. Diese Art von Information lässt sich in dem spektralen Bildbereich besser erkennen.

Faltende neuronale Netze sind vor allem gut in der Erkennung von Bildern, weshalb es sinnvoll sein kann, das Audiosignal zu transformieren. Eine besondere Form der Spektrogramme sind Mel Frequency Cepstral Coefficients (MFCC). Die Abbildung 3.6 zeigt die Schritte für die Implementierung der MFCC. Das Audiosignal ist sowohl zeit- als auch wertdiskret.

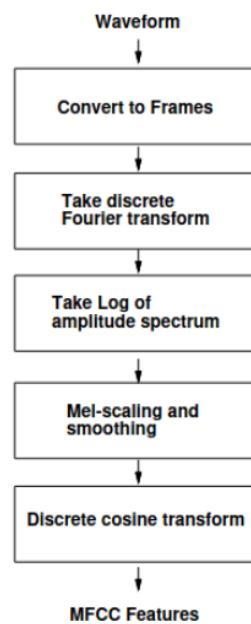


Abbildung 3.6: Implementierung der MFCC[14]

Der erste Schritt ist die Segmentierung des Audiosignals mithilfe einer Fensterfunktion, wie des "Hanning Windows". Die Größe des Fensters hängt von der Anwendung ab. Ein größeres Fenster sorgt für eine bessere Frequenzauflösung, aber für eine schlechtere Zeitauflösung. Bei einem kleineren Fenster ist es umgekehrt.

Im zweiten Schritt wird eine Discrete Fourier Transform (DFT) auf jedes Segment angewendet. Praktisch ist die Fast Fourier Transform (FFT) anstelle der DFT besser geeignet, da die Berechnungszeit kürzer ist [15]. Die Formel der DFT ist:

$$X(k) = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} [15] \quad (3.6)$$

Das Signal wird anschließend logarithmiert, während die Phaseninformationen verworfen werden. Der nächste Schritt ist die Skalierung des Frequenzbereichs auf die Mel-Frequenzen. Die Mel-Skalierung wie in Abbildung 3.7 gezeigt, ist dem menschlichen Gehör angepasst. Menschen nehmen Frequenzen über 1 kHz nicht linear wahr, sondern logarithmisch [14].

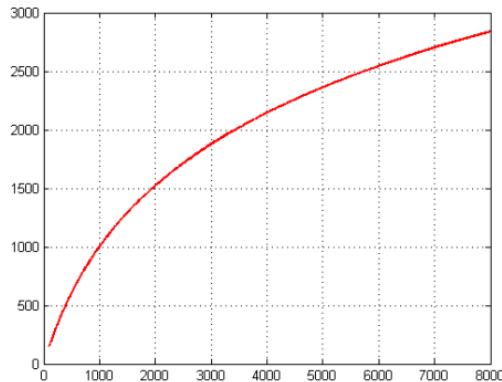


Abbildung 3.7: Mel-Skalierung
[16]

Um das Spektrum zu glätten, werden die spektralen Komponenten über Mel-Binning gemittelt. Das bedeutet, dass nahe beieinanderliegende Frequenzen in Gruppen zusammengefasst werden, was ein „geglättetes“ Spektrum ergibt. Diese Gruppen nennt man Bins. Da viele Informationen im Frequenzspektrum stark miteinander korrelieren, sind viele Informationen redundant. Um die Informationen zu dekorrelieren, wird der Mel-Vektor durch eine Discrete Cosine Transform (DCT) transformiert. Die Menge der Informationen

wird dadurch ebenfalls reduziert. Der Output der MFCC ist in der folgenden Abbildung erkennbar.

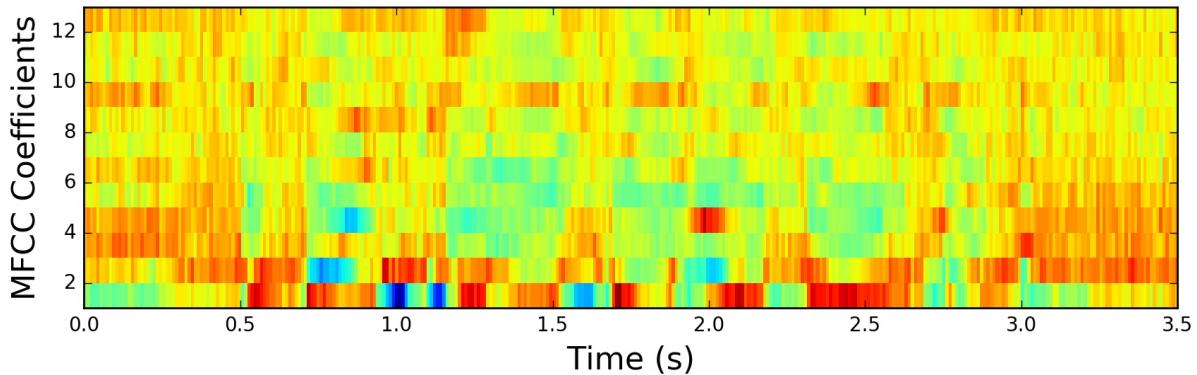


Abbildung 3.8: MFCC-Beispiel
[17]

3.3 CNN Beschleuniger

Für die Berechnung neuronaler Netze fallen zahlreiche Multiply-Accumulate (MAC)-Operationen an. Bei der Betrachtung eines einfachen Neurons wie in Abbildung 3.1 wird deutlich, dass zur Bestimmung seines Ausgangswerts mehrere Eingangssignale zunächst mit den jeweiligen Gewichten multipliziert und anschließend aufsummiert werden. Aus diesem Grund sind MAC-Operationen für die Berechnungen in neuronalen Netzen unverzichtbar. Bei einem Single-Core-Prozessor, wie der auf dem MAX78000 vorhandene ARM-Cortex-M4, werden solche Operationen sequentiell durchgeführt, weshalb der MAX78000, wie in Abschnitt 2.1.3 schon erwähnt, einen Hardware CNN-Beschleuniger besitzt. Der CNN-Beschleuniger besteht aus 64 Prozessoren, die parallel mit bis zu 50 MHz arbeiten und die Aufgabe haben, die Berechnung des neuronalen Netzes durchzuführen. Die 64 Prozessoren sind in 4 Quadranten aufgeteilt und jeder Quadrant ist in jeweils 4 Gruppen aufgeteilt. In jeder Gruppe befinden sich dementsprechend 4 Prozessoren, die sich einen gemeinsamen Speicher von 32KB für Gewichte teilen. Insgesamt besitzt der CNN-Beschleuniger 512KB SRAM. Die folgende Abbildung 3.9 verdeutlicht die Struktur eines einzelnen Quadranten.

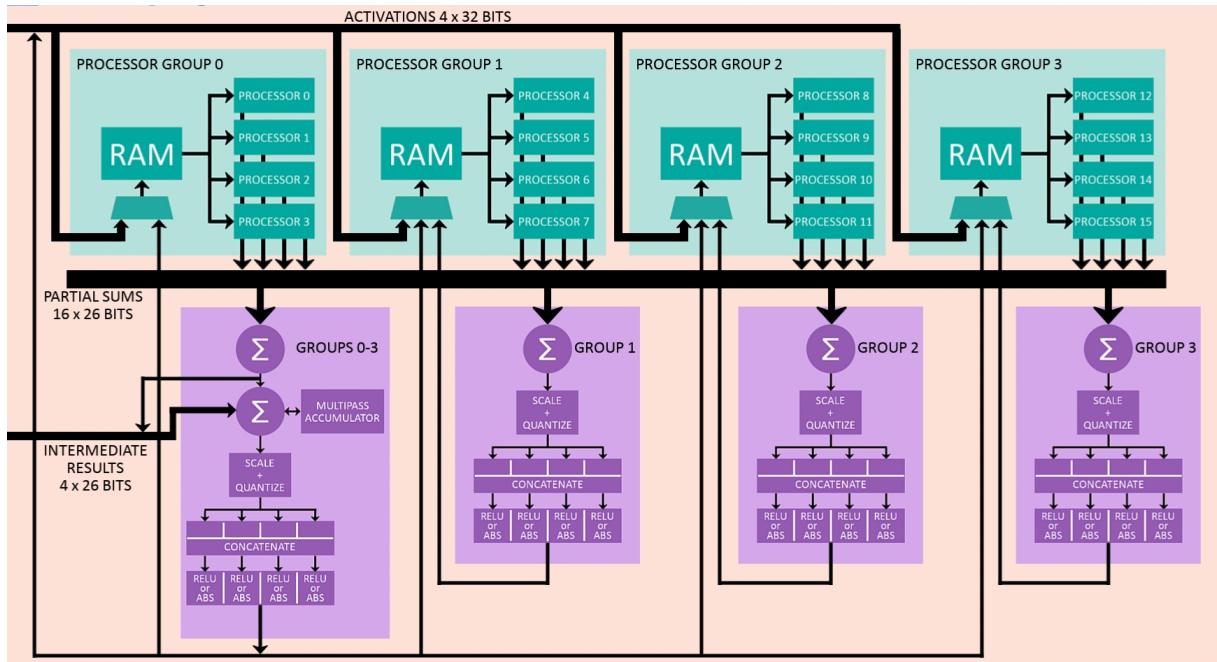


Abbildung 3.9: CNN-Beschleuniger

[18]

Jede Gruppe von Prozessoren hat eigene Blöcke für Summierung und Addition, kann aber auch mit den anderen Prozessor-Gruppen interagieren. Für eine detaillierte Beschreibung eignet sich ein Vorstellungsvideo des CNN-Beschleunigers [18] in Kombination mit dem Kapitel 27 des User-Guides [19]. Um eine manuelle Zuordnung der Gewichte zu den Prozessoren und die manuelle Konfiguration des CNN-Beschleunigers zu vermeiden, wird von Analog Devices ein Tool zur Generierung der gewünschten Konfiguration zur Verfügung gestellt. Dieses Tool macht es möglich, ein in Python trainiertes Modell in C-Code zu überführen. In Abschnitt 4.6 wird weiter auf dieses Tool eingegangen.

4 Umsetzung

4.1 Workflow

Eingebettete Systeme für ML-Anwendungen besitzen einen besonderen Workflow, der dokumentiert werden muss. In diesem Abschnitt werden die Arbeitsschritte für die Implementierung des neuronalen Netzes auf dem MAX78000 erläutert. Der Workflow lässt sich in drei wesentliche Arbeitsschritte unterteilen:

1. Training des neuronalen Netzes
2. Umwandlung des trainierten CNN in C-Code
3. Implementierung des C-Codes

Der MAX78000 ist ein eingebettetes System für Low-Power-Anwendungen. Das Training von neuronalen Netzen benötigt viel Energie und Rechenleistung, weshalb dieser Schritt von einem Desktoprechner durchgeführt wird. Dort kann die Graphics Processing Unit (GPU) verwendet werden, um rechenintensive Matrixoperationen zu übernehmen. Das Training des neuronalen Netzes wird in einer Python-Umgebung durchgeführt. Ein ausgewählter Datensatz ist die Grundlage des Trainings. Während im Datenblatt die Unterstützung für TensorFlow/Keras-Bibliotheken angegeben wird, weist die Dokumentation [20] des CNN-Trainings darauf hin, dass der Support für TensorFlow veraltet ist. Daher wird in dieser Arbeit eine spezielle PyTorch-Bibliothek verwendet. Der Hersteller des MAX78000 hat einen eigenen Workflow für die Implementierung von neuronalen Netzen bereitgestellt, der in dieser Arbeit teilweise genutzt wurde. Der Hauptunterschied besteht darin, dass der Trainingsabschnitt durch einen Alternativweg ersetzt wurde. Der Alternativweg wurde von Prof. Dr. Matthias Rosenthal in einem Blog-Beitrag der Zürcher Hochschule für Angewandte Wissenschaften (ZHAW) mit dazugehörigem

GitHub-Repository [21], der ZHAW erläutert. Dieser Workflow ermöglicht den Trainingsablauf in einem Jupyter-Notebook. Jupyter-Notebook ist eine webbasierte Entwicklungsumgebung für u.A. Python, die in der DataScience und ML-Community häufig benutzt wird, da einzelne Zeilen Code verändert und ausgeführt werden können, ohne das ganze Programm neu ausführen zu müssen. Der nächste Arbeitsschritt ist die Quantisierung des CNN und die Umwandlung des trainierten CNNs zu einem für den MAX78000 zugeschnittenen C-Code. Das Tool für die Umwandlung heißt ai8x-synthesis [22], welches ein skriptbasiertes Python-Tool von „Analog Devices“ ist. Der generierte C-Code umfasst Funktionen zur Initialisierung und Konfiguration des CNN-Beschleunigers zur Zuweisung der Gewichte an die Prozessoren des Beschleunigers sowie eine spezielle Funktion zum Entladen der Ausgabe (unload-Funktion) des CNN-Beschleunigers. Die Anbindung der Peripheriegeräte, wie beispielsweise des Audio-Codecs oder des internen Mikrofons, fällt nicht in den Funktionsumfang des Tools. In Abbildung 4.1 ist der komplette Workflow visualisiert.

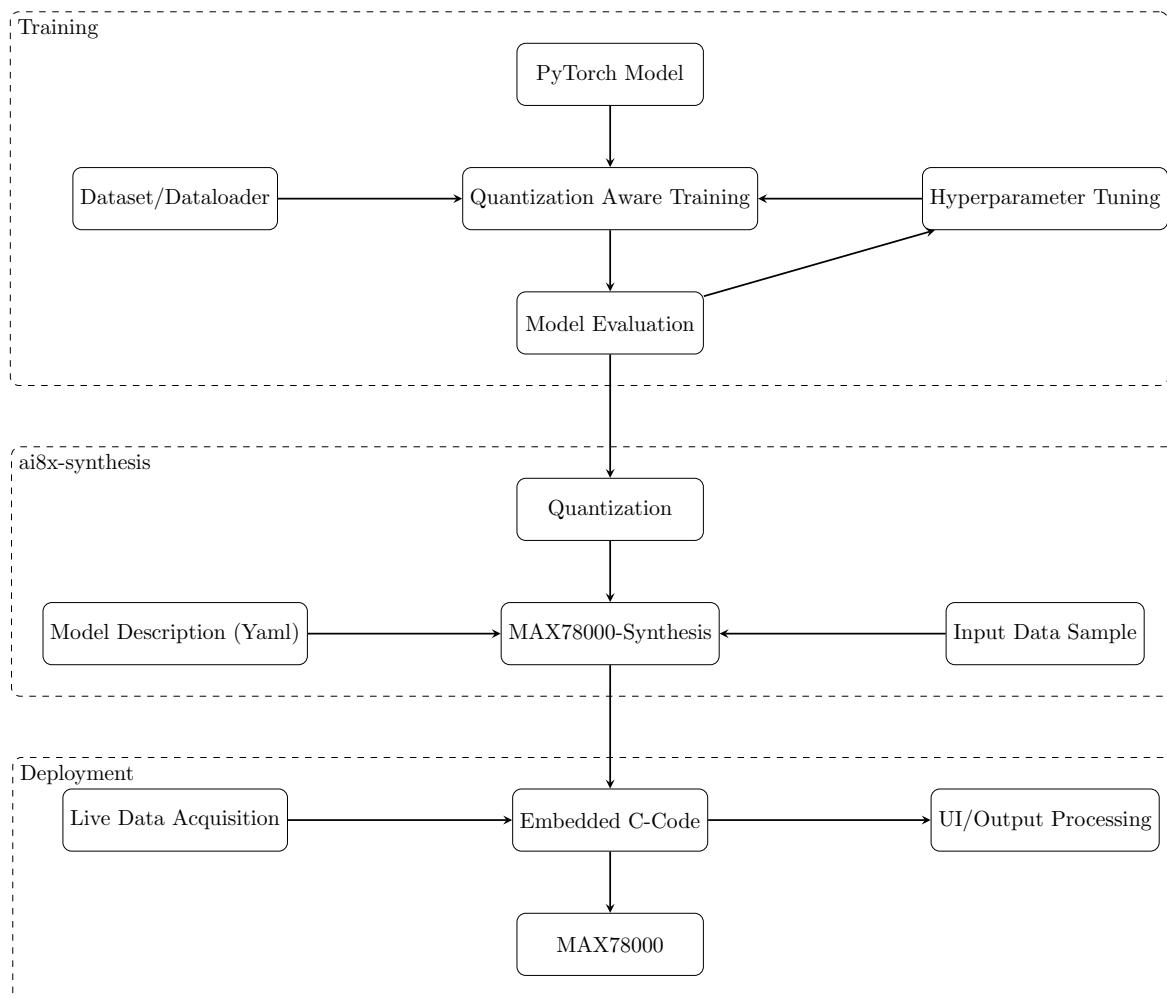


Abbildung 4.1: Workflow

4.2 Datensatz

Das neuronale Netz wird mit dem Datensatz TUT-Acoustic-Scenes 2017 [23] trainiert. Der Datensatz wurde von der Technischen Universität Tampere in Finnland erstellt und beinhaltet 52 Minuten Audio zu 15 verschiedenen Geräuschkulissen-Klassen. Die Klassen sind:

Bus	Cafe	Car	City center	Forest path
Grocery store	Home	Beach	Library	Metro station
Office	Residential area	Train	Tram	Urban park

Die Aufnahmen wurden mit einer Abtastrate von 44,1 kHz und einer Auflösung von 24 Bit aufgenommen. Die Aufnahmen sind in 10-Sekunden-Segmente aufgeteilt, die dem Machine-Learning-Algorithmus als Eingang dienen.

4.2.1 Datenverteilung

Die Menge der Daten pro Klasse ist gleich. Es muss keine Datenaugmentierung durchgeführt werden, um einen ausgeglichenen Datensatz zu benutzen.

Class	Sample size
Cafe	312
Car	312
City center	312
Forest path	312
home	312
office	312
park	312

4.2.2 Ortsabhängigkeit

Geräuschkulissen derselben Klasse aus verschiedenen Orten weisen häufig andere Charakteristiken auf, wie zum Beispiel U-Bahn-Töne, die beim Schließen der Türen andere

Töne spielen. Da dieser Datensatz an einem spezifischen Ort aufgenommen wurde, ist die Generalisierung des Machine-Learning-Algorithmus nur bedingt möglich.

4.2.3 Anpassung des Datensatzes

Damit eine Implementierung auf dem MAX78000 möglich ist, wurden für das Training nur 7 Klassen berücksichtigt. Der Vorteil ist, dass durch die verringerte Komplexität eine kleinere Netzstruktur für die Klassifizierung gewählt werden kann. Durch die verringerte Anzahl der Klassen ist der Datensatz insgesamt kleiner. Die ausgewählten Klassen sind:

1. Cafe	2. Car	3. City center	4. Forest path	5. Home	6. Office	7. Park
---------	--------	----------------	----------------	---------	-----------	---------

Damit der Datensatz als ein TensorFlow Dataset verwendet werden kann, muss der Datensatz auf 16-Bit-Float quantisiert werden. In PyTorch ist dies nicht notwendig.

4.2.4 Splits

Der Datensatz wurde nach Machine-Learning Best-Practices in Trainings-, Validierungs- und Testdaten aufgeteilt. Die Aufteilung ist wie folgt: Training/Testing/Validation Split:

	Verteilung	Samples
Training	0,7	1526
Validation	0,2	434
Testing	0,1	224

Die Trainingsdaten dienen der Anpassung der Gewichte und werden im Training genutzt. Die Validierungsdaten dienen der Anpassung der Hyperparameter, während die Testdaten für die abschließende Evaluation nach dem Training verwendet werden.

4.3 Audio Preprocessing

Für die Klassifizierung von Tonspuren, sei es Spracherkennung, Acoustic Scene Detection (ASC) oder SED, ist es wie in 3.2typisch, mit der rohen Tonspur ein Spektrogramm zu berechnen und dieses als Bild-Eingang für neuronale Netze zu benutzen. Die in dieser Arbeit genutzten Vorverarbeitungsschritte sind:

1. Abtastrate anpassen
2. Stereo to mono - Transformation
3. Mel-Spectrogram berechnen
4. Normalisieren

4.3.1 Abtastfrequenz

Die Abtastrate des Datensatzes ist 44,1 kHz. Für die Analyse des Datensatzes wurde ein durchschnittliches FFT-Spektrum des gesamten Datensatzes berechnet (siehe 4.2). Das durchschnittliche Spektrum bietet keine Anhaltspunkte zu der Relevanz der Frequenzinformationen, aber einen Überblick darüber, welche Spektralanteile überhaupt im Datensatz enthalten sind. Der Begriff der Relevanz ist auf die Wichtigkeit der Informationen zur Erkennung der Acoustic-Scene bezogen. Da oberhalb von 8 kHz keine große Informationsmenge vorhanden ist, wird die Abtastrate auf 16 kHz festgelegt. Um die Relevanz der Informationen im Frequenzbereich beurteilen zu können, wird während des Trainings die Abtastrate experimentell verändert. Im Fall einer ähnlichen Performance bei verringriger Abtastfrequenz kann davon ausgegangen werden, dass die verloren gegangenen Informationen nicht relevant sind.

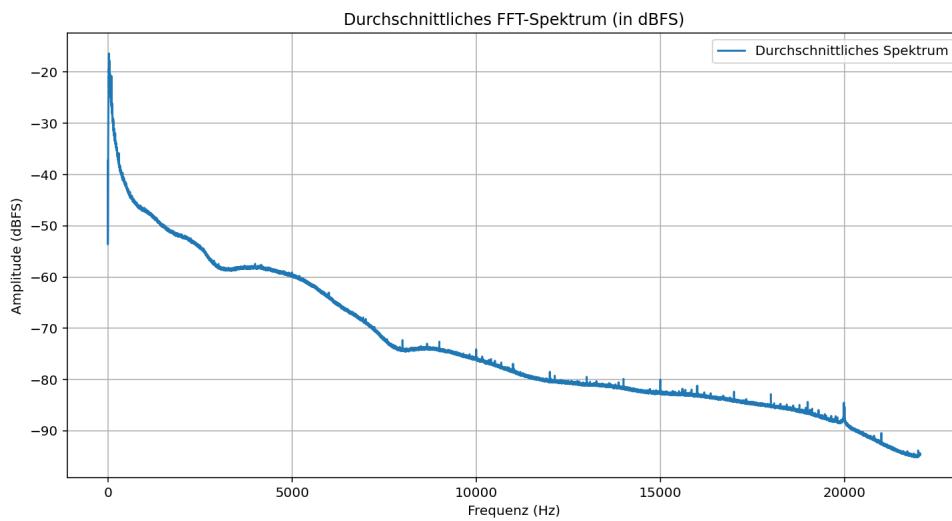


Abbildung 4.2: Durchschnittliches FFT-Spektrum

4.3.2 MFCC

Die Transformation des Audiosignals in den spektralen Bildbereich erfolgt durch die Berechnung der MFCC. Es werden folgende Parameter festgelegt:

```
window size : 0,92ms  
hop length : 0,46ms  
mfcc_outputs : 40  
n_mels : 40
```

Die Größe des MFCC Output-Vektors wurde in einer anderen Arbeit genutzt und übernommen. Mit größeren Werten wie 128 lernt das neuronale Netz den ganzen Datensatz auswendig, was nicht erwünscht ist. In Spracherkennungsanwendungen werden häufig Fenstergrößen von 20 - 40 ms und eine Sprunglänge von 10 ms gewählt, da 20 ms ausreichen, um Vokale aufzunehmen. Da in dieser Anwendung keine Spracherkennung erfolgt, wurde eine Fenstergröße von 92 ms für eine bessere Frequenzauflösung gewählt.

Während des Trainings wird die MFCC-Funktion der Tensorflow-Bibliothek verwendet. Die Implementierung der MFCC-Berechnung auf dem MAX78000 ist in 4.8.2 weiter beschrieben.

4.4 Training

4.4.1 Basismodell

Das in dieser Arbeit eingesetzte neuronale Netzwerk basiert auf der Verwendung von eindimensionalen Faltungen. Die Verwendung der eindimensionalen Faltung ist zwar im Kontext von Bildverarbeitung untypisch, kann aber in der Merkmalsextraktion von MFCCs sinnvoll sein. Die 1D-Faltung extrahiert auf der X-Achse Informationen zu der zeitlichen Veränderung der einzelnen Mel-Frequenzen. Als Ausgangspunkt wurde ein Netzwerk gewählt, das bereits in einem ähnlichen Kontext angewendet wurde [24]. Die Netzwerkkaratur ist in der Abbildung 4.1 visualisiert.

Das neuronale Netzwerk besteht aus drei Schichten, die der Merkmalsextraktion dienen. Jede Schicht umfasst eine 1D-Faltung mit einer Filtergröße von 5, gefolgt von einer Batch

Layer Type	Parameter
1D-Convolution + BN+ReLU	output_channels= 64 kernel_size = 5
MaxPooling	Pooling = 3
1D-Convolution + BN+ReLU	output_channels= 128 kernel_size = 5
MaxPooling	Pooling = 3
1D-Convolution + BN+ReLU	output_channels= 256 kernel_size = 5
Dropout	Dropout_rate = 0.5
Fully Connected Layer	Output = 512
Flatten+ Softmax	Output = 15

Tabelle 4.1: Aufbau des CNN als Basismodell
[24]

Abkürzungen: BN= Batch Normalization

Normalization und der ReLu-Aktivierungsfunktion. Zwischen den Faltungsschichten wird eine Max-Pooling-Operation mit einer Pooling-Größe von 3 durchgeführt, um die Dimensionen der Merkmale zu reduzieren und relevante Informationen zu extrahieren.

Nach Abschluss der Merkmalsextraktion wird eine Dropout-Schicht mit einer Dropout-Rate von 0,5 eingefügt, um Überanpassung (Overfitting) während des Trainings zu vermeiden. Anschließend folgen zwei vollständig verbundene Schichten (Fully Connected Layers), die die extrahierten Merkmale verwenden, um eine Klassenvorhersage zu treffen. Die Wahrscheinlichkeiten der Klassenvorhersagen werden durch die Softmax-Funktion berechnet, welche die Klassenzuordnung normalisiert und die Unsicherheit der Vorhersage quantifiziert.

4.4.2 Limitierungen des MAX78000

Der MAX78000 ist durch Hardware-Einschränkungen limitiert. Die für diese Arbeit relevanten Einschränkungen werden hier betrachtet. Diese Einschränkungen geben Vorgaben zur Architektur des neuronalen Netzes. Die maximale Anzahl der Layer eines neuronalen Netzes beträgt 64. Die Zahl der Parameter des Netzes kann bei einer 8-Bit-Auflösung 442 000 nicht überschreiten. Außerdem ist die maximale Input-Dimension 1024 x 1024. Dies ist ein Grund, weshalb die oben dargestellte Netzwerkarchitektur nicht verwendet werden kann, da bei der Flatten-Operation ein zu großer Vektor entsteht. Für 1D-Faltung

können Kernel-Größen zwischen 1 und 9 verwendet werden. Eine Verwendung von Batch-Normalization ist nur in Kombination mit der Benutzung von Bias möglich.

4.4.3 Hyperparameter Optimierung

Vor dem Training werden Kriterien, eine Zielstellung zur Performance und die genutzten Metriken festgelegt, damit diese während des Trainings betrachtet und evaluiert werden können. In dieser Arbeit wird die Genauigkeit des Modells als Metrik für die Performance verwendet. Es gibt KI-Anwendungen, bei denen die Vermeidung von bestimmten fehlerhaften Klassifizierungen von Bedeutung ist, was in dieser Aufgabenstellung nicht der Fall ist, weshalb die Genauigkeit ausreicht. Ein Beispiel wäre die Identifizierung von einem fehlerhaften Gerät, bei der es besser ist, zu oft einen möglichen Fehler zu erkennen, als zu selten.

Die Hyperparameter werden durch manuelle Anpassung optimiert, da bereits ein vielversprechender Ausgangspunkt durch die Arbeit von Venkatesh Duppada und Sushant Hiray [24] mit zufriedenstellenden Ergebnissen vorlag.

Während des Trainings werden verschiedene CNN-Architekturen unter verschiedenen Hyperparametern getestet und verglichen. Dabei wird die Performance des CNN für die Trainingsdaten, Validierungsdaten und Testdaten festgehalten. In diesem Abschnitt werden die für die Arbeit relevanten Trainingsergebnisse dargestellt.

Netzwerkarchitektur: V6

Die folgende Tabelle 4.2 zeigt die CNN-Architektur V6. Die Architektur ähnelt dem zuvor beschriebenen Basismodell mit kleinen Modifikationen. Alle einzelnen Operationen sind zusammengefasst und durch eine Funktion des Trainingstools `ai8x.py` ersetzt. Zusätzlich wird eine Schicht mit Average Pooling, 1D-Faltung, Batch Normalization und einer ReLu Aktivierung hinzugefügt, um die Größendimension zu verringern, da diese sonst die Limitierungen des MAX78000 überschreiten.

Layer Type	Parameter
1D-Convolution + BN+ReLU	output_channels = 64 kernel_size = 5 padding = 1
MaxPooling + 1D-Convolution + BN+ReLU	output_channels = 128 kernel_size = 3 padding = 0
MaxPooling + 1D-Convolution + BN+ReLU	output_channels = 256 kernel_size = 5 padding = 1
AvgPooling + 1D-Convolution + BN+ReLU	output_channels = 32 kernel_size = 3 padding = 1
Dropout	Dropout rate = 0.5
Flatten + Fully Connected Layer	Output channels = 7

Tabelle 4.2: Netzwerk Architektur V6

Abkürzungen : BN = Batch Normalization, Avg = Average, ReLU = Rectified Linear Unit

Model	V6_base	V6_1	V6_2	V6_3	V6_4
Sample Rate	16Khz	16Khz	14Khz	14Khz	14Khz
Learning rate	1	0,001	0,001	0,01	0,01
Weight Decay	0	0,0005	0,0005	0,0005	0,0005
Epochen	200	50	50	40	40
QAT-Epochen	50	50	50	50	50
Optimizer	Adadelta	Adam	Adam	Adam	Adam
Dropout	0,5	0,5	0,5	0,5	0,65
Batch Size	256	128	128	32	32
Accuracy Training	97.84%	99.41%	98.62%	97.64%	98.10%
Accuracy Validation	78.11%	79.49%	71.20%	72.81%	69.35%
Accuracy Testing	67.41%	64.29%	69.64%	76.79%	68.30%

Tabelle 4.3: Trainingsergebnisse der Netzwerkarchitektur V6

Die Tabelle 4.3 zeigt die wichtigsten Trainingsergebnisse, die eine Validation Accuracy von über 60 Prozent erzielt haben.

Die Hyperparameter des trainierten Basismodells aus Abschnitt 4.4.1 werden als Ausgangspunkt für diese Netzwerkarchitektur V6_Base verwendet. Das Training des Basismodells zeigt, dass 200 Epochen nicht notwendig sind, da das Training vorher konvergiert, weshalb das Training in den darauf folgenden Versuchen weniger Epochen durchläuft. Wie in der Tabelle 4.3 zu sehen ist, wird das CNN überangepasst (engl. Overfitting) und kann die Validierungsdaten nicht so gut klassifizieren wie die Trainingsdaten. Teilweise wird das

Netz durch eine hohe Lernrate nicht trainiert, weshalb in der darauf folgenden Trainingsiteration statt einer hohen Batchsize und einer hohen Lernrate eine kleinere Batch-Size und eine kleine Lernrate verwendet werden, was ähnliche Ergebnisse liefert. Die Verringerung der Abtastfrequenz löst das Problem der Überanpassung nicht und verringert stattdessen die Validierungsgenauigkeit. Eine kleinere Batch-Size erzielt keine Verbesserung. Die Veränderung des Optimizers hat sich als sinnvoll für niedrigere Lernraten erwiesen. Aus den Trainingsiterationen wird Modell V6_1 für eine Implementierung auf dem MAX78000 in Betracht gezogen, da dies die beste Validierungsgenauigkeit erzielt.

Netzwerkarchitektur: V5

Ein weiteres Netzwerk, welches trainiert und dessen Hyperparameter angepasst werden, ist das CNN-V5, welches in der folgenden Abbildung 4.4 visualisiert ist.

Layer Type	Parameters
1D - Convolution + ReLu	output_channels = 64 kernel_size = 5 padding = 0
1D- Convolution + ReLu	output_channels = 128 kernel_size = 3 padding = 0
MaxPooling + 1D Convolution + Relu	output_channels = 256 kernel_size = 5 padding = 1
AvgPooling +1D Convolution + Relu	output_channels = 64 kernel_size = 3 padding = 1
MaxPooling+ 1D Convolution + ReLu	output_channels = 32 kernel_size = 6 padding = 0
Dropout	Dropout rate = 0.2
Flatten + Fully Connected Layer	Output_channels = 7

Tabelle 4.4: Netzwerkarchitektur V5

Abkürzungen : Avg = Average, ReLU = Rectified Linear Unit

Im Gegensatz zum Netzwerk V6 kommt diese Netzwerkarchitektur ohne Batch-Normalization aus. Die ersten beiden Schichten bestehen aus einer reinen 1D-Convolution mit ReLU-Aktivierung. In den nachfolgenden Schichten werden Pooling-Operationen eingesetzt, die eine verbesserte Generalisierung unterstützen sollen. Zusätzliche Schichten dienen der er-

weiteren Merkmalsextraktion, während gleichzeitig die Dimensionen reduziert werden, um eine Implementierung auf dem MAX78000 zu ermöglichen. Die folgende Tabelle 4.5 beinhaltet die Trainingsergebnisse abhängig von ausgewählten Hyperparametern.

Model	V5_base	V5_1	V5_2	V5_3	V5_4
Sample Rate	16Khz	16Khz	11Khz	16Khz	16Khz
Learning rate	1	0,0004	0,0002	0,01	0,01
Weight Decay	0	0,0005	0,0005	0,0005	0,0005
Epochen	200	50	40	40	40
QAT-Epochen	50	50	50	50	50
Optimizer	Adadelta	Adam	Adam	Adam	Adam
Dropout	0,2	0,2	0,2	0,2	0,2
Batch Size	256	32	32	32	128
Accuracy Training	87.02%	97.71%	96.00%	94.50%	98.23%
Accuracy Validation	70.51%	75.58%	63.82%	66.59%	74.42%
Accuracy Testing	62.05%	70.09%	63.39%	69.20%	66.96%

Tabelle 4.5: Trainingsergebnisse der V5-Netzwerkarchitektur

Mit Ausnahme der Dropout-Rate werden die Hyperparameter des trainierten Basismodells aus Abschnitt 4.4.1 als Ausgangspunkt für diese Netzwerkarchitektur V5_Base verwendet. Das Training konvergiert bereits vor 200 Epochen, weshalb in den darauffolgenden Trainingsdurchläufen die Anzahl der Epochen reduziert wird. Zwar fällt die Überanpassung – erkennbar an der Differenz zwischen Trainings- und Validierungsgenauigkeit – geringer aus, jedoch verringert sich auch die allgemeine Validierungsgenauigkeit. In dem darauf folgenden Training V5_1 wird eine verringerte Lernrate, mit einer verringerten Batch-Size verwendet. Die Genauigkeit ist dadurch bei allen Metriken höher. Der Versuch V5_2 zeigt, dass eine niedrigere Abtastfrequenz die Performance beeinträchtigt. Eine mittlere Lernrate, ohne die Batch-Größe zu erhöhen, ist wie in Versuch V5_3 ebenfalls nicht erfolgreich. Versuch V5_4 zeigt, dass die Veränderung der Lernrate in Kombination mit einer Veränderung der Batch-Größe bessere Ergebnisse erzielt. Die Hyperparameter aus Versuch V5_1 werden verwendet, um ein Modell für diese Arbeit zu trainieren und weiter zu verwenden. Das V5_1 wird für bessere Kompatibilität mit der Hardware mit einer window-size von 1024 und einer hop-length von 512 neu trainiert. Das Training liefert folgende Ergebnisse:

	V5_1
Accuracy Training	98.23%
Accuracy Validation	83.64%
Accuracy Testing	70.09%

Tabelle 4.6: Angepasstes Modell V5_1

Die allgemeinen Erkenntnisse zum Training werden in Abschnitt 5.4 aufgegriffen.

In der folgenden Abbildung 4.3 ist die Confusion Matrix der Netzwerkarchitektur V5_1 zu erkennen. Die Confusion Matrix zeigt die Prediction der Klassen im Vergleich zur tatsächlichen Klasse zu einem Datensatz.

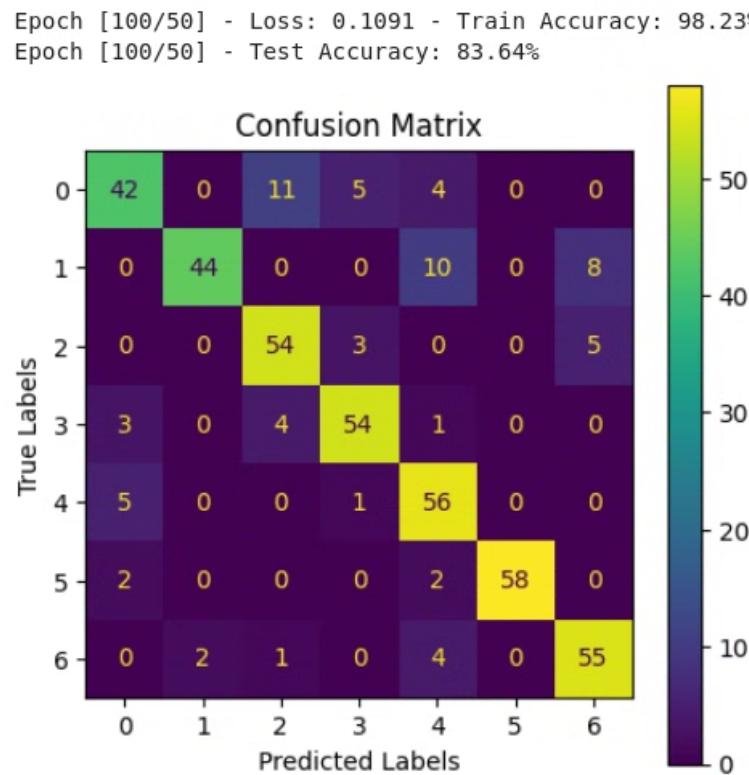


Abbildung 4.3: Trainingsergebnisse

Die Confusion-Matrix bezieht sich auf die Ergebnisse des Validierungsdatensatzes. Das Overfitting ist im Vergleich zu den anderen Trainingsergebnissen geringer. Diese Ergebnisse sind ausreichend, um dieses neuronale Netz für das eingebettete System zu verwenden.

4.4.4 Quantization Aware Training (QAT)

Die Implementierung des CNN auf der Hardware erfordert eine Quantisierung von 32-Bit-Float-Werten auf 8-Bit-Integer-Werte. Es ist ebenfalls möglich, eine 1,2- und 4-Bit-Quantisierung durchzuführen. Dabei werden alle trainierten Parameter (Gewichte, Bias)

quantisiert, was zu einer signifikant schlechteren Performance führen kann, da sie während der Optimierung unreguliert verändert werden. Um diesem Problem entgegenzuwirken, wird das Quantization Aware Trainingg (QAT) eingesetzt. Während des Quantization-Aware-Trainings werden die Parameter so angepasst, dass der durch die Quantisierung entstehende Fehler minimiert wird.

Die Implementierung des QAT ist in dem ai8x-training Tool vorhanden und wurde aus dem MNIST-Beispielprogramm [25] übernommen.

4.5 Quantization

Das CNN wird nach Abschluss des Trainings in einer Checkpoint-Datei gespeichert. Diese Datei enthält Informationen zu den genutzten Operationen, den verschiedenen Schichten und Gewichten. Für die Quantisierung des CNN wird das Github-Repository „ai8x-synthesis“ [22] verwendet. Das neuronale Netz besitzt 267.687 Parameter, was etwa 268 Kilobyte entspricht. Diese Gewichte belegen etwa 60% des Verfügbaren Speichers des CNN-Beschleunigers.

4.5.1 Evaluation

Die Bewertung eines trainierten Modells erfolgt routinemäßig an zwei Stellen im Entwicklungsworkflow: nach dem Training und nach der Quantisierung. Die Evaluierung nutzt Testdaten, also Daten, die während des Trainings nicht verwendet wurden, um die Leistungsfähigkeit des neuronalen Netzwerks zu messen. Die Evaluierung nach der Quantisierung dient dazu, die Auswirkungen der quantisierten Gewichte auf die Netzwerkleistung zu analysieren.

4.6 Ai8x-Synthese

Nach der Quantisierung des CNN erfolgt die Ai8x-Synthese [22]. Das quantisierte Modell wird in einen C-Code gewandelt, der die Gewichte den Prozessoren des CNN-Beschleunigers zuordnet und die Architektur des Modells auf dem Beschleuniger festlegt. Für diese Synthese werden folgende Bausteine benötigt:

- YAML-Network-Description-File
- Quantisierte Checkpoint-Datei
- Numpy-Array für Input-Beispiel

Der C-Code beinhaltet automatisch generierte Funktionen, die für die Inferenz verwendet werden:

cnn_init()
cnn_load_weights()
cnn_load_bias()
cnn_configure()
load_input()
cnn_start()
cnn_unload()
cnn_enable()

Tabelle 4.7: Autogenerierte Funktionen

4.6.1 YAML-Network-Description

Die Checkpoint-Datei beinhaltet die Gewichte der verschiedenen Schichten, aber nicht die umgesetzten Operationen. Die YAML-Network-Description beschreibt die Architektur des CNN unter Berücksichtigung der Implementierung auf Hardware-Ebene. Im Folgenden wird der Aufbau der YAML-Network-Description wiedergegeben.

```

1 arch: Custom1DCNNForImagev5_1
2 dataset: tut2017_40216
3
4 # Define layer parameters in order of the layer sequence
5 layers:
6   - pad: 0
7     activate: ReLU
8     out_offset: 0x2000
9     processors: 0x000000ffffffffffff
10    data_format: HWC
11    operation: Conv1d
12    kernel_size: 5
13    name: conv1
14   - pad: 0
15     activate: ReLU
16     out_offset: 0x0000
17     processors: 0xffffffffffffffff
18     operation: Conv1d
19     kernel_size: 3

```

Das YAML-Network-Description-Beispiel zeigt die ersten beiden Schichten des V5-CNN. Zu Beginn wird die Architektur (arch) betitelt. Die Architektur und der Name des Modells in der Checkpoint-Datei müssen identisch sein. Das Dataset verweist auf den Numpy-Array, der als Input-Beispiel dient. Der Numpy-Array muss sich in dem Unterordner **tests** des Github-Repositories ai8x-synthesis [22] befinden. In diesem Fall ist der Numpy ein generiertes MFCC eines Samples aus dem Validierungsdatensatz mit der Dimension 40x216. Anschließend werden die Schichten der Architektur definiert. Die genutzten Funktionen werden aufgeführt, wie zum Beispiel Padding, Aktivierung und Kernel Size. Zusätzlich wird die Zahl der Prozessoren des CNN-Beschleunigers festgelegt, die für die Berechnung aktiviert werden. Die Menge der aktiven Prozessoren der Schicht ergibt sich aus der Zahl der Input-Channels. Jedes Bit der 16-Bit hexadezimalen Zahl entspricht einer Gruppe von Prozessoren. Für 40 Input-Channels werden also 10 Gruppen aktiviert und der gesetzte Wert ist: 0x000000ffffffffffff. Bei Input-Channels größer als 64 werden die Prozessoren mehrmals verwendet, um die Operationen der Schicht durchzuführen. Die genutzten Prozessoren entsprechen dem größten gemeinsamen Teiler, welcher zu dem nächstgrößeren Vielfachen von 4 aufgerundet wird. Beispiele dazu befinden sich im YAML

Quickstart-Guide von Analog Devices [26].

Der Parameter `out_offset` gibt den relativen Offset im Datenspeicher an, an den die Ausgabedaten geschrieben werden. Der Eingang jeder Schicht wird vom `out_offset` der vorherigen Schicht übernommen. Um zu vermeiden, dass eine Eingabe überschrieben wird, die noch nicht verarbeitet wurde, sollte zwischen `out_offset = 0` und der Hälfte des Speichers (0x4000) oder weniger in aufeinanderfolgenden Schichten gewechselt werden (Ping-Pong-Technik)[26].

4.7 Implementierung auf eingebettetem System

Für eine funktionsfähige Acoustic-Scene-Detection müssen Audiodaten aufgenommen und verarbeitet werden. Das trainierte neuronale Netz muss in ein Programm eingebettet werden. Die folgenden Abschnitte erklären das Programm und dessen einzelnen Komponenten.

4.7.1 Programmablauf

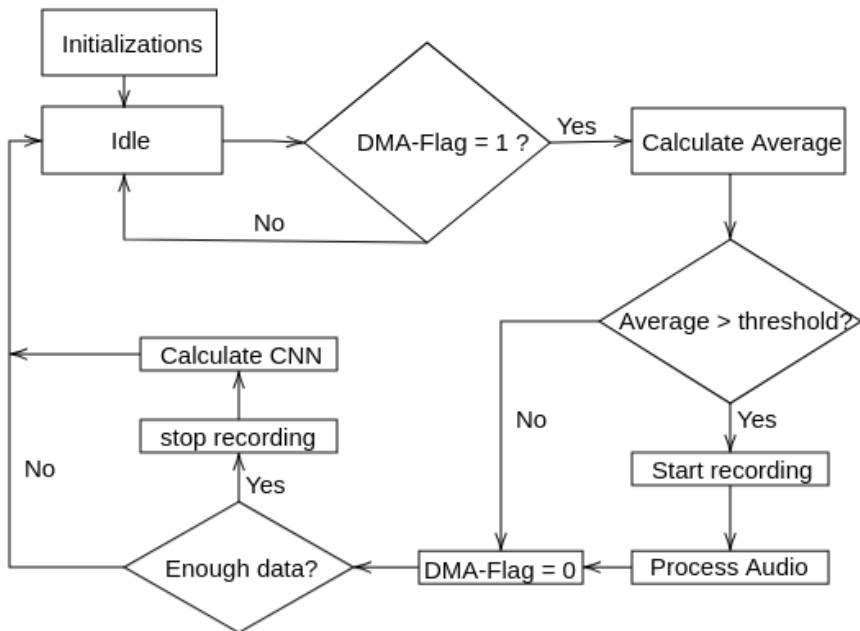


Abbildung 4.4: Programmablaufplan

Der Programmablauf beginnt mit der Initialisierung der relevanten Hardwarekomponenten. Dazu gehören die Einrichtung des Systemtaktes sowie die Initialisierung der Kommunikationsschnittstellen, darunter die I2C-Schnittstelle zur Kommunikation mit dem Audio-Codec MAX9867, die Codec-Initialisierung zur Aktivierung des Aufnahmepfads und der Verstärkungspegel sowie die I2S-Schnittstelle zur Audioübertragung. Zudem erfolgt die Initialisierung des DMA-Mechanismus, um einen effizienten und asynchronen Datentransfer von der I2S-Schnittstelle zum Prozessor zu ermöglichen.

Nach der Initialisierung werden Audiodaten kontinuierlich über den MAX9867-Codec aufgenommen und über DMA an den Hauptspeicher weitergeleitet. Dabei kommt ein DMA-Callback-Mechanismus zum Einsatz, der ein Flag setzt, sobald ein Transfer abgeschlossen ist. Sobald ein Datenpaket vorhanden ist, wird der durchschnittliche Pegel des Datenpakets berechnet. Falls der Pegel einen Schwellwert überschreitet, werden die darauf folgenden Daten für 10 Sekunden kontinuierlich verarbeitet. Die Verarbeitung startet mit einem Resampling zu 16 kHz, da der Audio Codec mit 48 kHz aufnimmt. Anschließend werden die Daten von Stereo zu Mono umgewandelt. Die verarbeiteten Audiodaten werden schrittweise in einen Ringpuffer geschrieben, um eine kontinuierliche Aufnahme zu gewährleisten. Dabei werden fortlaufend Mel-Frequency Cepstral Coefficients (MFCCs) berechnet und zur Merkmalsrepräsentation des Audiosignals zusammengeführt, da der Speicher nicht für 160.000 16-Bit-Samples ausreicht. Da die Daten kontinuierlich verarbeitet werden, während neue Daten transportiert werden, muss für die Verarbeitungszeit T_v folgende Bedingung erfüllt sein:

$$T_v < \frac{B}{f_s} \quad (4.1)$$

Das bedeutet, dass die Verarbeitungszeit T_v die zeitliche Dauer eines Puffers B nicht überschreiten darf.

Sobald ausreichend Daten verarbeitet wurden, wird der Feature-Vektor für die Klassifikation vorbereitet. Nach der Verarbeitung einer vordefinierten Menge von Audiodaten wird die weitere Datenerfassung gestoppt. Der erzeugte Merkmalsvektor wird dann als Eingabe in ein Convolutional Neural Network (CNN) eingespeist und das Ergebnis ausgegeben.

Nach Abschluss der Klassifikation wird der Prozess neu gestartet, indem die relevanten Variablen zurückgesetzt werden. Dadurch wird eine kontinuierliche akustische Szenenerkennung ermöglicht.

4.8 MAX78000

4.8.1 Audio Codec

Die Benutzung des Audio Codecs basiert im Hinblick auf die Funktionsweise auf dem Programmbeispiel `I2S_DMA_Target`, welches von Analog Devices zur Verfügung gestellt wurde. Das Programm wurde angepasst, so dass die Audiodaten per I²S-Protokoll über Direct Memory Access (DMA) an den Prozessor gesendet und keine Daten an den Audio Codec zurückgeleitet werden. Durch die Stromversorgung per USB entstehen Störgeräusche im Audiosignal. Um einen erhöhten Signal-zu-Rauschabstand zu erreichen, werden folgende Parameter für den Codec festgelegt.

Parameter	Value
ADC Level	-3, -3
Line in Gain	0, 0

Tabelle 4.8: Parameter Audio Codec

Der MAX78000 kommuniziert über I²C mit dem Audio Codec für die Initialisierung und die Einstellung der Parameter. Während der Initialisierung des Audio Codecs bleibt das Programm hängen, da ein Timing-Problem besteht. Um dieses Problem zu umgehen, muss in der Header-Datei des Audio Codecs die Funktion `reg_write()` angepasst werden. Das Hinzufügen einer Verzögerung von 8μs genügt, um dieses Problem zu beheben. Der folgende Code zeigt die genutzte Funktion.

```

1 #include "mxc.h"
2 static int reg_write(uint8_t reg, uint8_t val)
3 {
4     uint8_t buf[2] = { reg, val };
5     i2c_req.tx_buf = buf;
6     i2c_req.tx_len = sizeof(buf);
7     i2c_req.rx_len = 0;
8     int result = MXC_I2C_MasterTransaction(&i2c_req);
9     MXC_Delay(8); // Verzögerung um auf MasterTransaction zu warten
10
11    return result;
12 }
```

4.8.2 CMSIS DSP Library

Die CMSIS-DSP-Bibliothek bietet Funktionen zur Implementierung der MFCC in verschiedenen Datentypen. Die Funktionen unterscheiden sich durch das Datenformat, welches genutzt werden soll. Die Python-Bibliothek `cmsidsp` bietet die Möglichkeit, die MFCC's in Python zu berechnen, um diese dann zu vergleichen. Die in dieser Arbeit genutzte Funktion ist für das Datenformat `float32` bestimmt, obwohl die Audiodaten mit einer Auflösung von 16-Bit-Integer-Werten aufgenommen werden. Der Grund dafür ist weiter unten beschrieben. Die Initialisierung des MFCC-Algorithmus erfolgt mit der Funktion `arm_mfcc_init_512_f32()`. Die Filterkoeffizienten des Hanning-Fensters und der Discrete-Cosine-Transformation können mit einem Python-Skript generiert werden. Das Skript ist unter <https://github.com/ARM-software/CMSIS-DSP/tree/main/Scripts> zu finden.

Für die Benutzung der CMSIS-DSP Bibliothek ist eine Aktivierung der Bibliothek notwendig. Dafür muss in der Datei `project.mk` folgendes hinzugefügt werden:

```
LIB_CMSIS_DSP = 1
```

Vergleich verschiedener MFCC-Algorithmen

Ein Beitrag im Internet[27] liefert einen Hinweis dazu, dass möglicherweise verschiedene Implementierungen des MFCC in den Bibliotheken verwendet werden. Dieser Vergleich ist relevant für diese Arbeit, weshalb die Werte noch einmal verglichen werden. Für ein funktionierendes CNN müssen bei dem Training dieselben Operationen durchgeführt werden, da sonst die Performance verringert wird. Die folgende Tabelle 4.9 zeigt die Outputs an, die bei demselben Input-Signal berechnet wurden.

Vergleich MFCC	[0]	[1]	[2]	[3]	[4]	[5]	[6]
MAX78000_f32	54	-1	-2	-6	-3	2	5
MAX78000_q15	-7060	1248	338	-320	-278	245	615
Tensorflow	-47	1	-5	-7	-4	2	5
Python CMSIS_f32	-47	1	-5	-7	-4	2	5
Torchaudio	-218	16	-20	-34	-10	24	32
Librosa	-282	22	-22	-30	-16	8	20

Tabelle 4.9: Vergleich verschiedener MFCC-Implementierungen

Anmerkung: Auf ganzzahlige Werte für Übersichtlichkeit gerundet

Der Vergleich des Outputs von verschiedenen MFCC-Algorithmen aus verschiedenen Bibliotheken zeigt, dass diese sich teilweise unterscheiden und in der Form ähnlich sind, sich aber nicht durch lineare Operationen zueinander überführen lassen. Die geprüften Operationen sind additive Verschiebung, Skalierung und Normalisierung. Der CMSIS-MFCC-Algorithmus auf dem MAX78000 im float 32-Datenformat gibt ähnliche Outputs wie Tensorflow und die CMSIS Python-Bibliothek. Die Q15-Implementierung gibt einen ganz anderen Output aus als die in anderen Python-Implementierungen.

Ableitungen für diese Arbeit

Wie schon erwähnt, ist es wichtig, während der Inferenz dieselben Operationen für die Vorverarbeitung durchzuführen, wie während des Trainings, weshalb die `float32` Implementierung gewählt wird. Diese ist einer Python-Implementierung am ähnlichsten, obwohl grundsätzlich die Q15-Berechnung aufgrund der Audioauflösung und dem geringeren Berechnungsaufwand besser geeignet wäre. Zusätzlich wird das neuronale Netz neu trainiert, da die vorherige Implementierung mithilfe der Torchaudio-Bibliothek durchgeführt wurde, welche keine vergleichbaren Ergebnisse aufweist. Die genutzte Python-Bibliothek ist Tensorflow, da die `cmsisdsp` library nicht mit der Python-Konfiguration kompatibel ist, die während des Trainings genutzt wird.

4.8.3 Daten für CNN laden

Wie in 3.3 beschrieben sind die 64 Prozessoren des CNN-Beschleunigers in Gruppen segmentiert. Die Input-Daten des CNN müssen den Gruppen zugeordnet werden. Die Abbildung 4.5 visualisiert diesen Prozess.

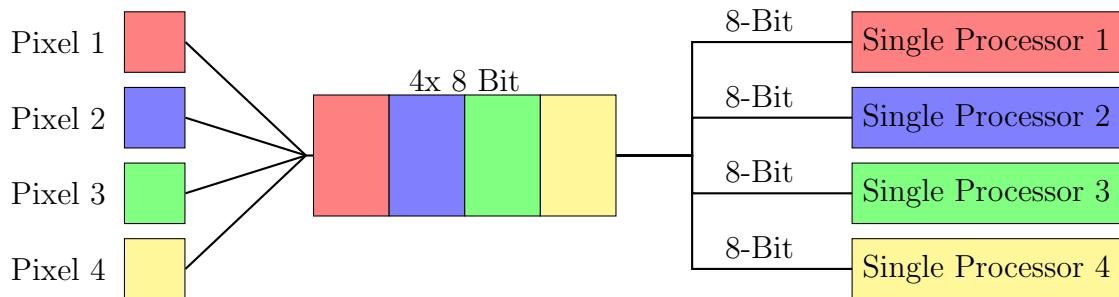


Abbildung 4.5: Datenladen für CNN-Beschleuniger

Vier Pixel mit jeweils 8-Bit-Integer-Werten werden in einer 32-Bit-Variable gespeichert und an die Adresse der Gruppe übergeben. Dies geschieht für alle Gruppen, die in der ersten Schicht des CNN genutzt werden. Da in dieser Arbeit die Eingangsschicht **40x311** Pixel besitzt, werden 10 Gruppen (40 Prozessoren) aktiviert und jeder Gruppe 216 32-Bit-Werte zugeordnet.

4.9 Metriken für Evaluierung

4.9.1 Performance des neuronalen Netzes

Um die Performance des neuronalen Netzes zu bestimmen, wird ein kleiner Teil des Datensatzes, welcher nicht im Training verwendet wurde, als Testdatensatz verwendet. Das neuronale Netz trifft Predictions, die anschließend in einer Confusion-Matrix dargestellt werden. Die Ergebnisse enthalten Informationen darüber, wie gut das CNN jede einzelne Klasse vorhersagen kann und wie gut die Vorhersage im Gesamtkontext ist. Außerdem lässt sich erkennen, zwischen welchen Klassen das neuronale Netz unter Umständen nicht gut unterscheiden kann.

Actual	Predicted						
	0	1	2	3	4	5	6
0	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	A_{17}
1	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}	A_{27}
2	A_{31}	A_{32}	A_{33}	A_{34}	A_{35}	A_{36}	A_{37}
3	A_{41}	A_{42}	A_{43}	A_{44}	A_{45}	A_{46}	A_{47}
4	A_{51}	A_{52}	A_{53}	A_{54}	A_{55}	A_{56}	A_{57}
5	A_{61}	A_{62}	A_{63}	A_{64}	A_{65}	A_{66}	A_{67}
6	A_{71}	A_{72}	A_{73}	A_{74}	A_{75}	A_{76}	A_{77}
	Pred 0	Pred 1	Pred 2	Pred 3	Pred 4	Pred 5	Pred 6

Abbildung 4.6: Confusion Matrix Beispiel

4.9.2 Inference-Time Messung

Inference-Time MAX78000

Inference bezeichnet im Machine Learning den Prozess, bei dem ein bereits trainiertes Modell auf neue, unbekannte Daten angewendet wird, um eine Vorhersage, Klassifikation oder andere Entscheidung zu treffen. Die Inference-Time ist dementsprechend die Zeit, die für die Inference benötigt wird. Um die zeitliche Performance des CNN-Beschleunigers, also die Inferenz-Zeit zu erfassen, wird ein Pin des MAX78000 vor dem Beginn der Inferenz umgeschaltet und nach der Inferenz wieder zurückgesetzt. Der Zustand des Pins und die vergangene Zeit zwischen dem Zustandswechsel lassen sich mit einem Oszilloskop messen. Im ML-Kontext wird von Vorhersagen (englisch: Predictions) gesprochen, obwohl damit nicht zwingend Aussagen über die Zukunft gemeint sind. Das Hauptaugenmerk liegt auf der Zeit, die der CNN-Beschleuniger für die Berechnung des CNN benötigt, da dieser evaluiert werden soll. Die Berechnungszeiten für das Preprocessing der Daten und der Datentransport werden aber für eine ganzheitliche Betrachtung ebenfalls mit ausgewertet. Das Oszilloskop sollte eine Zeitauflösung von 100 ns besitzen, um Messwerte im Mikrosekunden-Bereich präzise aufnehmen zu können.

Inference-Time Python

Die direkte Messung der Inference-Time eines neuronalen Netzes in einer Desktop-Anwendung kann nicht mit einem Oszilloskop durchgeführt werden, da keine physikalische Möglichkeit besteht, die Zeit im Mikrosekunden-Bereich präzise zu erfassen. Daher wird ein software-basierter Ansatz gewählt. Für die Messung wird eine spezialisierte Timing-Funktion innerhalb von PyTorch verwendet. Das CNN und die Eingabedaten werden auf die GPU geladen. Es werden 10000 Inferenzen aufeinanderfolgend ausgeführt und die Berechnungszeit dafür wird gemessen. Die durchschnittliche Inferenzzeit wird berechnet, indem die Gesamtzeit durch die Anzahl der ausgeführten Inferenzen geteilt wird. Wenn als Eingabedaten ein zufälliger Dummy-Input verwendet wird, erhält man die Berechnungszeit des CNN's. Bei der Benutzung des Datensatzes wird die Berechnungszeit inklusive der Datenvorverarbeitung bestimmt, da die Audiodaten noch verarbeitet werden. Durch die Differenz der beiden Inferenz-Zeiten erhält man die Berechnungszeit der Vorverarbeitung. Dieser Ansatz ermöglicht eine höhere Messgenauigkeit und reduziert statistische Messfehler. Durch die hohe Anzahl an Durchläufen werden zufällige Schwankungen minimiert und eine zuverlässige Zeitbestimmung für die Modellinferenz erreicht.

4.9.3 Energieverbrauch

Der Energieverbrauch des MAX78000 lässt sich durch die Messung des Stroms mit Hilfe einer Strommesszange während der Inference mit der Inference-Time bestimmen. Das Umschalten des Pins dient hier als Trigger, wodurch sich durch folgende Formel der Energieverbrauch bestimmen lässt.

$$E[J] = V_s \cdot I \cdot t \quad (4.2)$$

Die folgende Abbildung 4.7 zeigt den Messaufbau für die Evaluation. Das Amperemeter stellt die Strommesszange und das Voltmeter das Oszilloskop dar.

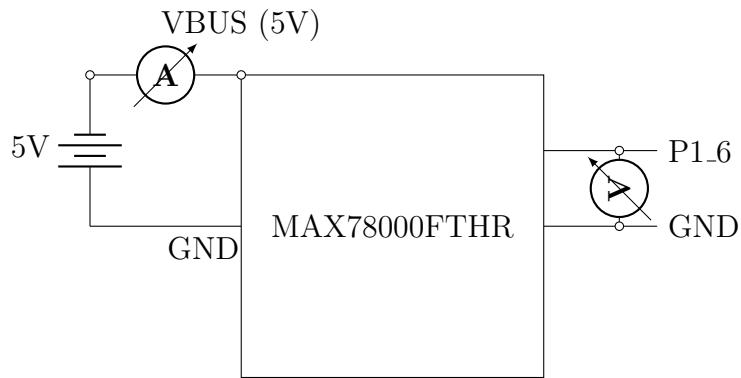


Abbildung 4.7: Messaufbau

Energieverbrauch Python

Der Energieverbrauch der Pythonanwendung lässt sich mit den zur Verfügung stehenden Mitteln nicht genau bestimmen. Die GPU, die während der Inferenz die Berechnungen durchführt, wird durch Hintergrundprozesse, wie zum Beispiel die Nutzung eines Bildschirms, belastet. Eine Näherung des Energieverbrauchs (E_i) lässt sich durch die Differenz der Leistung der GPU während der Inferenz (P_i) und der Leistung im Ruhezustand (P_r) multipliziert mit der Inference-Zeit bestimmen:

$$E_i[J] = (P_i - P_r) \cdot t_i \quad (4.3)$$

Die Leistung der GPU lässt sich im System Management Interface (SMI) ablesen. Diese lässt sich bei Nvidia GPUs im Terminal mit dem Befehl `nvidia-smi` aufrufen.

5 Ergebnisse und Auswertung

5.1 Genauigkeit des Models

Um die Genauigkeit des Modells zu bestimmen, wird der Test-Datensatz verwendet. Die Genauigkeit wird an drei Stellen ermittelt: Nach dem Training, nach der Quantisierung und während der Anwendung (Deployed Model). Das Python-Modell erreicht eine Genauigkeit von 70,09%. Um die Genauigkeit des quantisierten Modells zu bestimmen, wird der Testdatensatz verarbeitet und jedes Sample in einen Array im C-Format umgewandelt und als Header-Datei gespeichert. Diese können einem Testprojekt eingefügt werden. Da der Speicher des MAX78000 begrenzt ist, werden die Header-Dateien stückweise eingebunden. Das quantisierte Modell erreicht ebenfalls eine Genauigkeit von 70,09%. Die folgende Tabelle 5.1 zeigt die Confusion-Matrix des quantisierten Modells. Das Python-Modell liefert dieselben Ergebnisse.

	Prediction						
Actual	Cafe	Car	CC	FP	Home	Office	Park
Cafe	0.750	0.000	0.000	0.250	0.000	0.000	0.000
Car	0.000	0.688	0.000	0.000	0.156	0.000	0.156
CC	0.000	0.000	0.969	0.000	0.000	0.000	0.031
FP	0.000	0.000	0.406	0.594	0.000	0.000	0.000
Home	0.031	0.000	0.000	0.000	0.531	0.063	0.375
Office	0.000	0.000	0.000	0.000	0.313	0.688	0.000
Park	0.125	0.000	0.188	0.000	0.000	0.000	0.688

Tabelle 5.1: Confusion-Matrix nach Quantisierung (normalisiert)

Abkürzungen: CC = City Center, FP= Forest Path

Die grün markierten Werte sind die richtig erkannten akustischen Szenen. Die orange markierten Werte zeigen die Predictions, die häufig falsch waren. Die Confusion-Matrix zeigt, dass „City Center“ in 97% der Fälle richtig erkannt wird, aber den Forest Path in 41% ebenfalls als City Center klassifiziert. Außerdem scheint das CNN nicht gut zwischen Home und Office unterscheiden zu können.

Evaluierung des implementierten Modells

Zur Bestimmung der Genauigkeit des implementierten Modells wird der Audioausgang des Desktop-Computers mit dem Audioeingang des MAX78000 verbunden, sodass der Testdatensatz über den Audioausgang abgespielt und vom CNN verarbeitet werden kann. Damit der Datensatz im selben Wertebereich aufgenommen, wie auf dem Computer abgespielt wird, muss der Pegel des Datensatzes verstärkt werden. Ein Python-Programm spielt sequenziell den Datensatz ab, und der MAX78000 gibt über die serielle Schnittstelle die Ergebnisse der Inferenz aus. Der Code dazu befindet sich im Anhang 7.2. Die folgende Abbildung 5.2 zeigt die Ergebnisse.

	Prediction						
Actual	Cafe	Car	CC	FP	Home	Office	Park
Cafe	0.53	0.00	0.00	0.25	0.22	0.00	0.00
Car	0.00	0.19	0.00	0.00	0.19	0.00	0.63
CC	0.00	0.00	0.88	0.00	0.03	0.00	0.09
FP	0.00	0.00	0.19	0.75	0.06	0.00	0.00
Home	0.00	0.03	0.00	0.00	0.97	0.00	0.00
Office	0.00	0.00	0.00	0.00	1.00	0.00	0.00
Park	0.00	0.00	0.22	0.00	0.09	0.00	0.69

Tabelle 5.2: Confusion-Matrix des Endergebnis

Abkürzungen: CC = City Center, FP = Forest Path

Wie zu erkennen ist, sind die Fehler, die während des Trainings und nach der Quantisierung stärker ausgeprägt. Das neuronale Netz ist nicht in der Lage, die Klasse „Office“ zu erkennen, und erkennt stattdessen „Home“. Die Klasse „Car“ wird ebenfalls nicht gut

erkannt. Mögliche Ursachen werden in der Auswertung (5.4) aufgegriffen.

Die folgende Abbildung zeigt die Evaluierung noch einmal konzentriert.

	Python-Modell	Quantisiertes Modell	implementiertes Modell
Cafe	0,75	0,75	0,53
Car	0,69	0,69	0,19
City_center	0,97	0,97	0,88
Forest_path	0,59	0,59	0,75
home	0,53	0,53	0,97
office	0,69	0,69	0
park	0,69	0,69	0,69
Gesamt	0,70	0,70	0,57

Tabelle 5.3: Performance-Vergleich

Die Tabelle zeigt, dass die Genauigkeit des implementierten Modells um etwa 13% im Vergleich zum quantisierten Modell sinkt. Überraschenderweise zeigt das implementierte Modell eine wesentlich bessere Erkennung der Geräuschkulisse „Home“ als das Python-Modell.

5.2 Inferenz-Zeit

In diesem Abschnitt werden die erzielten Inferenz-Zeiten des MAX78000 und der Python-Implementierung mit einer Nvidia RTX 4060 Ti GPU verglichen. Die GPU besitzt 4352 Cuda-Recheneinheiten, die mit 2,31 GHz getaktet sind. Andere GPUs erzielen andere Ergebnisse. Verglichen werden die Zeiten, die der CNN-Beschleuniger für die Berechnung des CNNs benötigt. Ebenfalls wird mit aufgenommen, welche Zeit für das Verarbeiten und Laden des Inputs benötigt wird. Da der MAX78000 erst die Daten aufnehmen muss, wird nur die Berechnung des letzten MFCC-Fensters berücksichtigt, da die anderen Fenster während der Aufnahme schon verarbeitet werden. Die Zeit, die für das Laden des Inputs benötigt wird, ist bei der GPU mit im Preprocessing inbegriffen. Die Screenshots der Messungen befinden sich im Abschnitt 7.3

	MAX78000	GPU
Vorverarbeitung in ms	2,82	5,75
Load_input in ms	2,52	-
CNN Berechnung in ms	3,18	1,92
Gesamt in ms	8,52	7,68

Tabelle 5.4: Berechnungszeiten von MAX78000 und GPU

Die Messergebnisse zeigen, dass das CNN auf der GPU 1,26 ms schneller berechnet wird als auf dem CNN-Beschleuniger des MAX78000. Die Vorverarbeitungsschritte werden auf der GPU ebenfalls schneller durchgeführt.

5.3 Energieverbrauch

Um den Energieverbrauch des MAX78000 zu bestimmen, wird der Stromverbrauch im Betrieb ohne CNN-Berechnung und während der CNN-Berechnung gemessen. Der Leiter, dessen Strom gemessen wird, ist zweimal um die Strommesszange gewickelt, um den Messbereich zu erweitern und den Messfehler zu verringern. Der in der Abbildung gezeigte Strom muss also durch zwei geteilt werden.

Es ist zu erkennen, dass der Stromverbrauch während der Inferenz ansteigt. Der Pin, der als Trigger dient, wird vor dem Laden des Inputs auf „High“ und nach der CNN-Berechnung auf „Low“ gesetzt. Nach dem Laden des Inputs, also während der tatsächlichen CNN-Berechnung, steigt der Strom von durchschnittlich 40,14 mA auf 69,3 mA. Der steigende Stromverbrauch des MAX78000 ist durch die zusätzliche Nutzung des Hardware-CNN-Beschleunigers zu erklären. Die Werte liegen in einem Bereich, der mit dem Datenblatt übereinstimmt. Die folgende Tabelle 5.5 zeigt den Energieverbrauch des MAX78000 und der GPU.

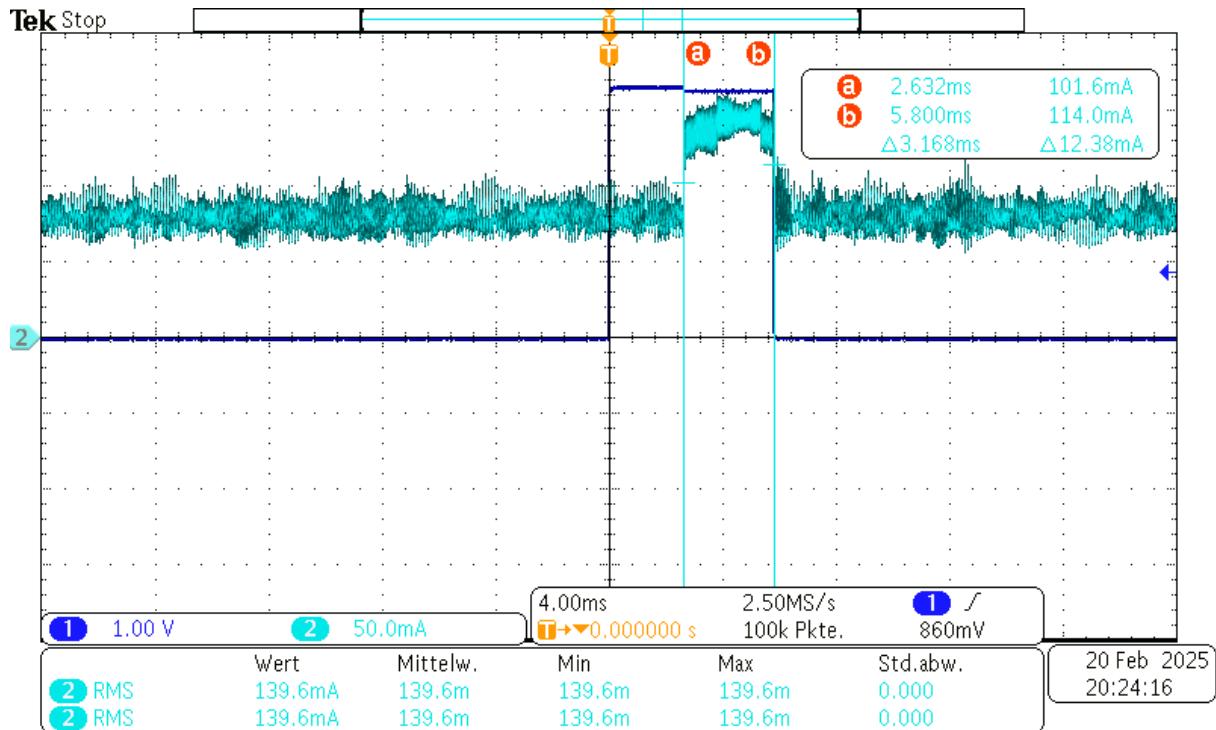


Abbildung 5.1: Stromverbrauch während der Inferenz

	MAX78000	GPU
Leistung ohne CNN Belastung	200,70 mW	17 W
Leistung bei CNN Berechnung	346,50 mW	47 W
Leistung der CNN Berechnung	145,75 mW	30 W
Inferenz-Zeit	3,168 ms	1,924 ms
Energieverbrauch der CNN-Berechnung	461,736μJ	57,72 mJ

Tabelle 5.5: Energieverbrauch während der Inferenz

Die Messungen zeigen, dass der Energieverbrauch während der CNN-Berechnung bei beiden Systemen steigt. Die GPU benötigt bei 0% Auslastung 17 Watt. Während der Inferenz steigt die Leistung auf 47 Watt bei 42% Auslastung. Trotz der geringeren Inferenz-Zeit der GPU wird durch die Inferenz circa 125 Mal mehr Energie verbraucht als durch die Inferenz des CNN-Beschleunigers.

5.4 Auswertung

Die Ergebnisse zeigen, dass die Performance des trainierten Modells in Python durch die Quantisierung nicht beeinflusst wird. Das QAT erfüllt dementsprechend seinen Zweck. Der Vergleich der Performance des quantisierten Modells und des Deployed-Modells zeigt, dass die Audio-Vorverarbeitung die Performance des CNN einschränkt. Mögliche Ursachen können sein:

1. Rauschanteile des Audio Codecs
2. Zu Große Buffer-Size
3. MFCC-Implementierung

Im Audiosignal werden Störgeräusche eingebunden, die aus der Stromversorgung stammen. Das Testbeispiel **I2S-DMA-Target** zeigt, dass das Toggeln einer LED zu einer niederfrequenten Änderung des Stromverbrauchs führt, was dem Audiosignal Artefakte hinzufügt. Dieser Effekt könnte die Genauigkeit des CNNs beeinflussen. Die Buffer-Size könnte ebenfalls die Genauigkeit beeinflussen, da die Audiodaten möglicherweise nicht direkt erkannt werden und der MAX78000 verspätet mit der Aufnahme beginnt. Die MFCC-Berechnung der CMSIS-DSP Library weist, wie in Abschnitt 4.8.2 beschrieben, Unterschiede zur Python-Implementierung auf, was zusätzlich die Performance beeinflussen kann. Basierend auf den Ergebnissen ist eine Genauigkeit von 57% nicht ausreichend für die Anwendung. Insgesamt lässt sich jedoch die Generalisierung des Modells mit der genutzten Größe des Datensatzes nicht gut einschätzen. 32 Samples pro Klasse reichen nicht aus, um statistische Aussagen zu treffen, und der Trainingsdatensatz ist ebenfalls nicht groß genug, um ein Modell sinnvoll zu trainieren. Viele ML-Anwendungen erzielen bessere Ergebnisse, wenn das System viele Daten für das Lernen nutzen kann [11]. Ein größerer Datensatz könnte helfen, Überanpassung zu verringern.

Die GPU besitzt wesentlich mehr Prozessoren (Cores) als der CNN-Beschleuniger, die auch durch CUDA effizient genutzt werden können. Trotzdem ist die Inferenz-Zeit der beiden Systeme vergleichbar. Dies könnte an dem Datentyp liegen, da für die Berechnung eines 8-Bit-integer-CNNs weniger Berechnungsschritte durchgeführt werden können, als bei einer 32-Bit-float-Architektur. Bei dem Vergleich des Energieverbrauchs wird der Nutzen des CNN-Beschleunigers sichtbar. Bei einer vergleichbaren Inferenz-Zeit ist der Energieverbrauch circa 125 mal geringer als bei der GPU. Im Betrieb verbraucht der MAX78000

ungefähr 30mA mehr Strom, während der CNN-Beschleuniger die Inferenz durchführt. Das bedeutet zwar, dass mit der Inferenz auch ein erhöhter Energieverbrauch einhergeht, dieser aber im Vergleich zur Nutzung der GPU gering ist. Insgesamt zeigen die Ergebnisse das Potenzial des MAX78000 und dessen CNN-Beschleuniger. Der CNN-Beschleuniger ist in der Lage, bei geringerem Energieverbrauch und ähnlicher Inferenz-Zeit die gleiche Genauigkeit zu erreichen wie die Implementierung in Python. Der Faktor, der die Performance des Deployed-Modells verringert, ist die Audiovorverarbeitung, die noch verbessert werden kann, aber nicht Teil der Beschleuniger-Komponente ist.

6 Ausblick

Es gibt verschiedene Möglichkeiten, die zu einer Verbesserung des Neuronalen Netzes führen können. In diesem Abschnitt werden die Verbesserungsmöglichkeiten als Ausblick dargestellt. Wie bereits erwähnt, kann eine Vergrößerung des Datensatzes die Generalisierungsfähigkeit des Netzes verbessern. Dazu eignet sich zum Beispiel der Datensatz „TAU Urban Acoustic Scenes 2019“ welcher mehr als 4 mal so viele Daten enthält. Außerdem ist es möglicherweise nicht notwendig, 10 Sekunden Audiosegmente für die Inferenz zu verwenden. Eine Segmentierung in 1-Sekunden-Samples würde den Datensatz um das 10-Fache vergrößern. Ein Problem, welches während des Trainings auffiel, ist das Overfitting. Eine Nutzung von Regularisierungsmethoden wie L2-Regularisierung könnte dabei helfen, das Problem zu vermindern. Das genutzte CNN musste angepasst werden, damit eine Implementierung auf dem MAX78000 möglich ist. Der Nachfolger des MAX78000 besitzt weniger Limitierungen und wäre in der Lage, bessere Neuronale Netze zu verwenden. Es ist sicherlich auch möglich, andere Netzwerkarchitekturen zu trainieren, die weniger Parameter und eine bessere Performance liefern. Da es bei der Entwicklung von neuronalen Netzen keine klaren Regeln, sondern eher „Best-Practices“ gibt, müsste mit anderen Netzen experimentiert werden. Weniger Parameter wären wünschenswert, da diese die Inferenz-Zeit verringern.

Um die gesamte Berechnungszeit zu verringern, wäre auch ein Ansatz ohne die Berechnung von MFCCs interessant. Die Berechnung der MFCC benötigt in diesem Beispiel pro Fenster etwa 1,86 ms. Das sind 65% der Gesamtdauer der Vorverarbeitung und etwa 22% der gesamten Inferenz. Der Ansatz ohne MFCC-Berechnung würde die Nutzung der Rohaudiiodaten in einem 1D-Vektor oder 2D-Vektor beinhalten.

7 Anhang

7.1 Nutzung von Generativer KI

In dieser Arbeit wurde teilweise die Generative KI „ChatGPT“ für die Generierung von Programmcode verwendet. Die Programmcodes, die davon betroffen sind, sind:

- Deployed Model (Custom1DCNNForImagev5)
- Model5_train_eval.ipynb
- train_TUT2017_v6.ipynb
- Conversion Audiodatei zu MFCC Numpy-Arrays (Transform_toNumpyarray.ipynb)
- Conversion Numpyarrays zu C header-Dateien (dataloader_mfcc.py)
- Datensatz Struktur Umwandlung (Datasetersteller.ipynb)

Die Codeerstellung erfolgte durch einen iterativen Prozess, bei dem bei Auftreten eines Fehlers der generierte Output erneut als Input in die generative KI eingespeist wurde – bis das gewünschte Ergebnis erzielt wurde. Der initiale Startbefehl dieses Verfahrens ist in den Programmcodes oben beigefügt.

7.2 Bestimmung der Inferenzzeit der GPU

```

1 import time
2 dummy_input = torch.randn(1, 40, 311).to(device)
3
4 model.eval() # Evaluationsmodus aktivieren
5 with torch.no_grad():
6     # Optional: Warm-Up-Läufe
7     for _ in range(100):
8         _ = model(dummy_input)
9
10    if torch.cuda.is_available():
11        torch.cuda.synchronize()
12
13    num_runs = 10000
14    start_time = time.time()
15    for _ in range(num_runs):
16        _ = model(dummy_input)
17    if torch.cuda.is_available():
18        torch.cuda.synchronize()
19    end_time = time.time()
20
21    total_time = end_time - start_time
22    avg_time = total_time / num_runs*1000
23    print(f"Durchschnittliche Inferenzzeit mit Dummy Input:
24        {avg_time:.6f} ms")
25
26 with torch.no_grad():
27     # Warm-Up: ein paar Batches durchlaufen, um Verzögerungen zu
28     # minimieren
29     for i, (inputs, labels) in enumerate(dataset_train):
30         inputs = inputs.to(device)
31         _ = model(inputs)
32         if i >= 5:
33             break
34
35     # Synchronisiere alle GPU-Operationen vor dem Start

```

```
34 if torch.cuda.is_available():
35     torch.cuda.synchronize()
36
37 start = time.time()
38 for inputs, labels in dataset_train:
39     inputs = inputs.to(device)
40     _ = model(inputs)
41     if torch.cuda.is_available():
42         torch.cuda.synchronize() # Sicherstellen, dass der Batch
        ↳ vollständig abgearbeitet ist
43 end = time.time()
44
45 total_time = end - start
46 average_time = total_time/1526*1000 # 1526samples im
        ↳ trainingsdatensatz, *1000 für ms zeit
47 print(f"Average Inferenzzeit mit Datensatz verarbeitung:
        ↳ {average_time:.6f} ms")
48
```

7.3 Messungen der Berechnungszeiten MAX78000

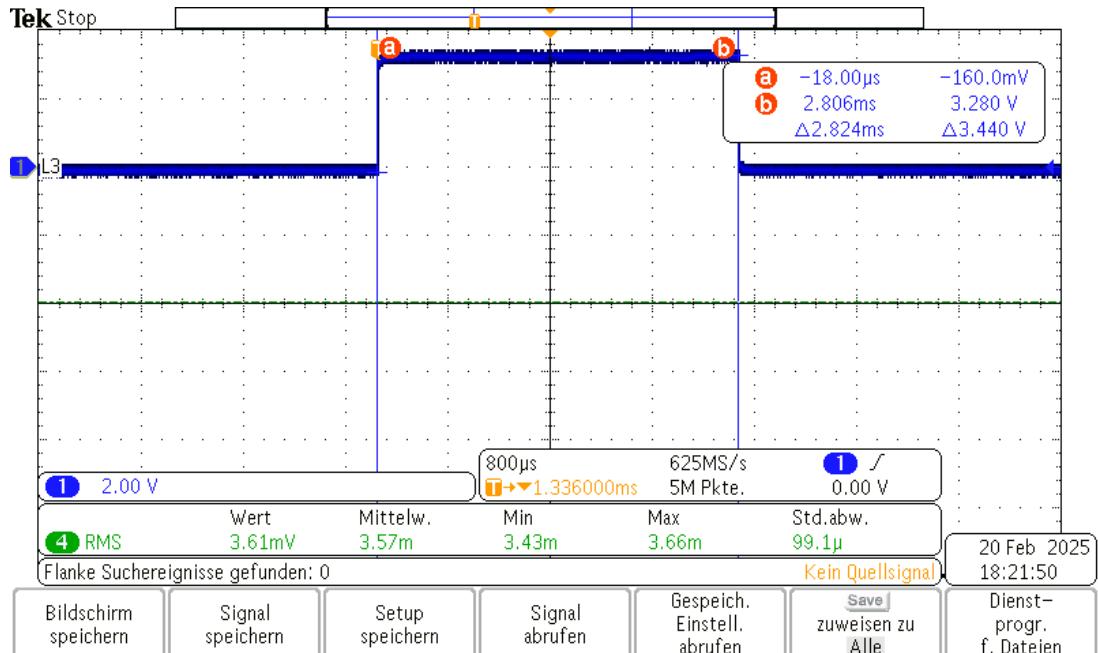


Abbildung 7.1: Messung der Verarbeitungszeit eines MFCC-Fensters

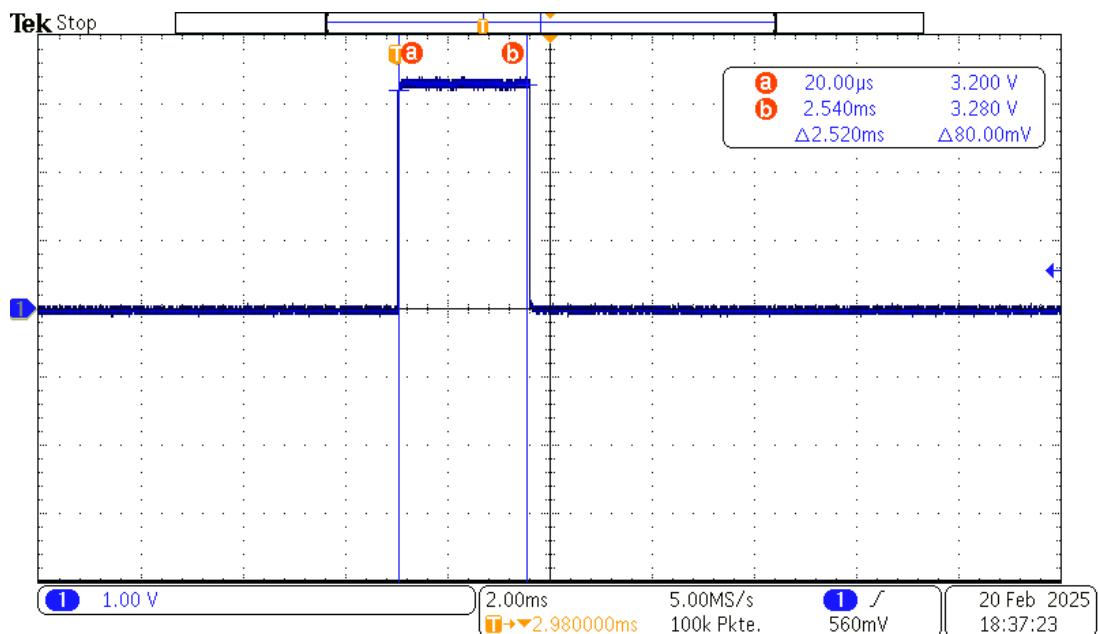


Abbildung 7.2: Messung der Berechnungszeit der load_input Funktion

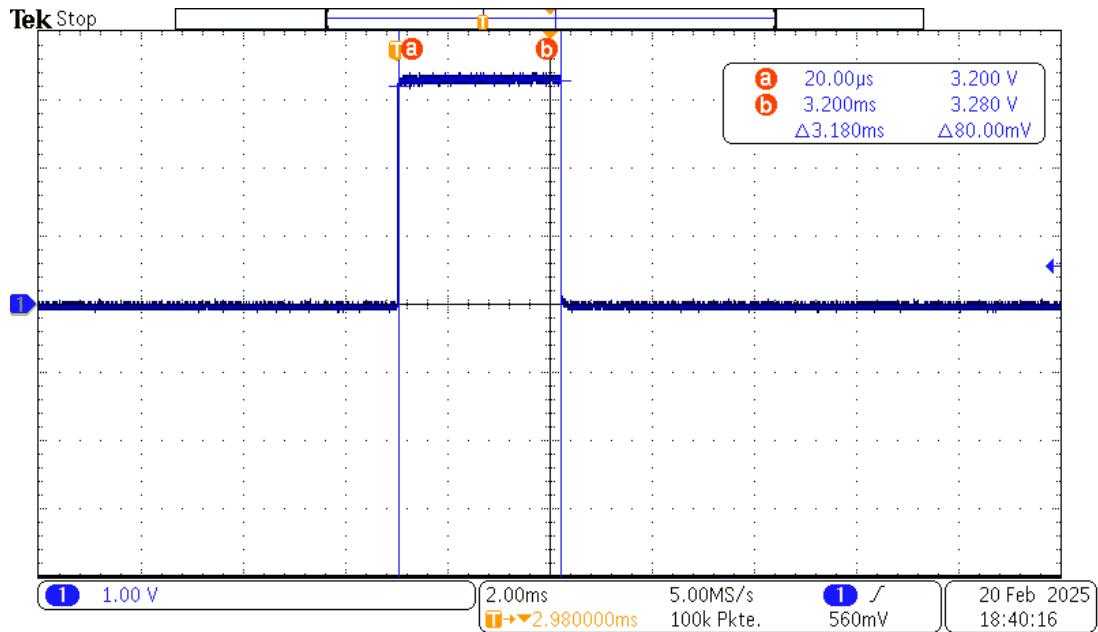


Abbildung 7.3: Messung der Berechnungszeit des CNNs

Literaturverzeichnis

- [1] Oliver Knodel. Wieviel energie braucht die cloud?, besucht am 21-01-2025.
https://tu-dresden.de/ing/informatik/ti/vlsi/ressourcen/dateien/dateien_forschung/publikationen/LNdW_Knodel.pdf?lang=de.
- [2] Joris Bakker. Asc-max78000_fthr, besucht am 25-02-2025.
https://github.com/joris-bakker/ASC-MAX78000_FTHR.
- [3] Brandon Lewis. embedded world 2024 highlights growing demand for edge ai. inside.tech, 2025.
- [4] Douglas Youvan. Ai at the edge vs. ai in the cloud: A comparative analysis of high-end and edge ai systems, 06 2024.
- [5] Nucleo-n657x0-q, besucht am 01-02-2025.
<https://www.st.com/en/evaluation-tools/nucleo-n657x0-q.html>.
- [6] STMicroelectronics. Stm32n6 high-performance mcus, 2024.
- [7] Jetson orin nano super developer kit, besucht am 01-02-2025.
<https://developer.nvidia.com/buy-jetson?product=all&location=DE>.
- [8] Tau urban acoustic scenes 2022 mobile, development dataset, besucht am 08-02-2025.
<https://zenodo.org/records/6337421>.
- [9] Dcase2025 challenge, besucht am 08-02-2025.
<https://dcase.community/challenge2025/>.
- [10] Eva Hüll. Das universelle approximationstheorem für neuronale netze, 2021.

- [11] Aaron Courville Ian Goodfellow, Yoshua Bengio. Deep learning, 2016. S. 414.
- [12] Iuliana Tabian, Hailing Fu, and Zahra Sharif Khodaei. A convolutional neural network for impact detection and characterization of complex composite structures. *Sensors*, 19(22), 2019.
- [13] Tejovk. Optimization challenges in deep learning, besucht am 08-02-2025.
<https://medium.com/@tejovk311/optimization-challenges-in-deep-learning-a4b085d529>
- [14] Beth Logan. Mel frequency cepstral coefficients for music modeling. Proc. 1st Int. Symposium Music Information Retrieval, 11 2000.
- [15] S.Palani. Discrete Time Systems and Signal Processing. Springer, 2023.
- [16] Narcís Sayols. Pattern modelling and pattern processing in image and speech signals, 07 2012.
- [17] Haytham M. Fayek. Speech processing for machine learning: Filter banks, mel-frequency cepstral coefficients (mfccs) and what's in-between, 2016.
<https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html>.
- [18] Analog Devices inc. Understanding artificial intelligence episode 5, besucht am 08-02-2025.
<https://www.analog.com/en/resources/media-center/videos/6313217809112.html>.
- [19] Analog Devices inc. Max78000 user guide, besucht am 08-02-2025.
<https://www.analog.com/media/en/technical-documentation/user-guides/max78000-user-guide.pdf>.
- [20] ai8x-training, besucht am 21-01-2025.
<https://github.com/analogdevicesinc/ai8x-training>.
- [21] Dr.Matthias Rosenthal. Max7800x-jupyter-training, besucht am 27-01-2025.
<https://github.com/InES-HPMM/MAX7800x-Jupyter-training/tree/main>.

- [22] ai8x-synthesis, besucht am 21-01-2025.
<https://github.com/analogdevicesinc/ai8x-synthesis>.
- [23] Tut acoustic scenes 2017, development dataset, besucht am 01-02-2025.
<https://zenodo.org/records/400515>.
- [24] Venkatesh Duppada and Sushant Hiray. Ensemble of deep neural networks for acoustic scene classification. CoRR, abs/1708.05826, 2017.
- [25] Matthias Rosenthal. Max7800x-jupyter-training, 2022.
<https://github.com/InES-HPMM/MAX7800x-Jupyter-training/tree/main>.
- [26] Robert Muchsel. Network description yaml files: A quick start guide, 2023.
https://github.com/analogdevicesinc/MaximAI_Documentation/blob/main/Guides/YAML%20Quickstart.md.
- [27] Andrea Ronco. Matching mfcc configuration with librosa's, 2022.
<https://github.com/ARM-software/CMSIS-DSP/issues/54>.

Muster einer Selbständigkeitserklärung bei erlaubtem Einsatz von KI-Tools:

Selbständigkeitserklärung

Name, Vorname: Bakker, Joris

Matr.-Nr.: 77923

Prüfungsnummer: E376

Prüfungsform: Hausarbeit

Ausgabedatum: 06.12.2024

Bearbeitungszeitraum: 12 Wochen

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen (einschließlich elektronischer Quellen und dem Internet) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht.

Ich versichere, dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen sind, durch Zitate als solche gekennzeichnet habe. Dies gilt auch für Textpassagen, die manuell oder mittels digitaler Übersetzungshilfen aus einer anderen Sprache übersetzt oder hin- und rückübersetzt wurden.

Ferner versichere ich, Textpassagen, bildliche Darstellungen oder anderweitige generierte Ergebnisse, die mittels Künstlicher Intelligenz verfasst oder erstellt wurden, von mir vollständig markiert wurden und die Zugriffsquelle, Datum und Eingabeparameter (Prompts) von mir vollständig angegeben wurden.

Ich versichere auch, dass eine von mir gegebenenfalls eingereichte papierförmige Version mit der digitalen Version übereinstimmt.

Unerlaubte Hilfsmittel wurden von mir nicht verwendet und auch keine unerlaubte Hilfe durch dritte Personen in Anspruch genommen.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht physisch oder elektronisch veröffentlicht.

Ich erkläre mich damit einverstanden, dass die Digitalversion dieser Arbeit zwecks Plagiatsprüfung auf die Server externer Anbieter hochgeladen werden darf. Die Plagiatsprüfung stellt keine Zurverfügungstellung für die Öffentlichkeit dar.

Eine Veröffentlichungserlaubnis ist einer besonderen Erklärung vorbehalten.

Ort, Datum

Leipzig 27.02.25

Vorname, Name
(Unterschrift)

Joris Bakker
Joris Bakker