Hochschule
Bonn-Rhein-Sieg
University
of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Thesis

Studiengang Informatik

Verkehrslärmüberwachung durch KI-gestützte akustische Fahrzeugidentifikation

von

Ameur Khemissi

MatrikelNr: 9039774

Erstprüfer	Prof. Dr. Michael Rademacher
Zweitprüfer	M.Sc. Oliver Lanzerath
Betreuer	B.Sc. Marcel Aniol
	Accso - Accelerated Solutions GmbH
Eingereicht am	16.07.2025

Name: Ameur Khemissi
Adresse: ***zensiert***

Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbst angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Unterschrift

Sankt Augustin, den

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich während meiner Bachelorarbeit unterstützt haben.

Ein großes Dankeschön geht an **Accso – Accelerated Solutions GmbH** dafür, dass ich meine Abschlussarbeit im Unternehmen schreiben durfte. Die spannenden Einblicke in die Praxis und die Unterstützung bei allen Fragen haben mir viel gebracht. Besonders danke ich auch für die Übernahme der Hardwarekosten – das hat die Umsetzung meines Projekts deutlich erleichtert.

Vielen Dank an **Marcel Aniol** für die Betreuung im Unternehmen sowie an **Axel Burghof** für die organisatorische Unterstützung. Ich schätze besonders das konstruktive Feedback und die hilfreichen Anregungen, die mir im Verlauf der Arbeit sehr weitergeholfen haben.

Ebenso bedanke ich mich bei **Prof. Dr. Michael Rademacher** und **M.Sc. Oliver Lanzerath** für die engagierte Begleitung meiner Arbeit von Seiten der Hochschule. Auch hier haben mir die Rückmeldungen sehr geholfen, meine Ideen gezielt weiterzuentwickeln.

Nicht zuletzt danke ich meiner Familie und meinen Freunden für die Unterstützung, die Motivation und das Verständnis – nicht nur während dieser Arbeit, sondern während des gesamten Studiums.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kurzfassung

Diese Arbeit präsentiert die Entwicklung und Evaluierung eines ressourcenschonenden, datenschutzfreundlichen Systems zur akustischen Erkennung und Klassifikation von Fahrzeugen im urbanen Raum. Ziel ist es, Verkehrsgeräusche in Echtzeit aufzunehmen, zwischen Verkehr und keinem Verkehr zu unterscheiden sowie Fahrzeuge in vier Klassen (PKW, Bus, LKW, Motorrad) zu erkennen und zu zählen. Grundlage bildet der öffentlich verfügbare IDMT-Datensatz, auf dessen Basis drei Modellarchitekturen (CNN1D, CNN2D, CNN-LSTM) miteinander verglichen werden. Das CNN2D-Modell erzielte dabei die besten Ergebnisse und wurde im weiteren Verlauf verwendet. Ergänzend wurde gezeigt, dass Datenaugmentierung eine signifikante Verbesserung der Klassifikationsleistung bei unterrepräsentierten Klassen bewirken kann.

Durch Quantisierung konnte das Modell auf eine einsatzfähige Größe für den Raspberry Pi Zero 2 reduziert werden, ohne die Klassifikationsleistung zu beeinträchtigen. In Tests mit real aufgenommenen Audiodaten – die vom Trainingsdatensatz abweichten – erzielte das System zunächst eine moderate Genauigkeit von 56 %. Durch ein gezieltes Finetuning mit zusätzlichen und augmentierten Aufnahmen konnte diese auf 81 % gesteigert werden. Besonders zuverlässig arbeitete das System bei der Unterscheidung zwischen Verkehr und keinem Verkehr sowie bei den Klassen PKW, Bus und Motorrad. Die kontinuierlichen Tests unter realistischen Verkehrsbedingungen zeigten eine stabile Systemleistung, niedrige CPU- und RAM-Auslastung sowie eine problemlose Stromversorgung über eine Powerbank. Einschränkungen zeigten sich insbesondere bei der Erkennung von LKWs und bei der Datensicherung, da Daten nur lokal gespeichert wurden.

Die Ergebnisse zeigen, dass ein leichtgewichtiges, akustisches Verkehrserkennungssystem auch unter realen Bedingungen zuverlässig funktionieren kann und damit einen Beitrag zur datenschutzfreundlichen Verkehrsanalyse leisten kann.

1 Einleitung

1.1 Motivation und Problemstellung

In den letzten Jahren ziehen immer mehr Menschen in die Städte ein. Im Jahr 2023 lebten etwa 77,77 % der deutschen Bevölkerung in städtischen Gebieten. Der Anteil lag im Jahr 2000 noch bei 74,97 %. Prognosen gehen davon aus, dass dieser Anteil bis 2050 auf über 84 % steigen wird [38][7]. Diese fortschreitende Urbanisierung stellt vor allem infrastrukturelle Herausforderungen dar. Die zunehmende Anzahl an Fahrzeugen führt regelmäßig zur Überlastung der Innenstädte und zu Staus. In einer Umfrage von Ben Impey bezeichneten über 48 % der Befragten dies als das größte Problem. Zeitverlust ist jedoch nicht die einzige Nebenwirkung, sondern auch Luftverschmutzung und Lärm (Abbildung 1).

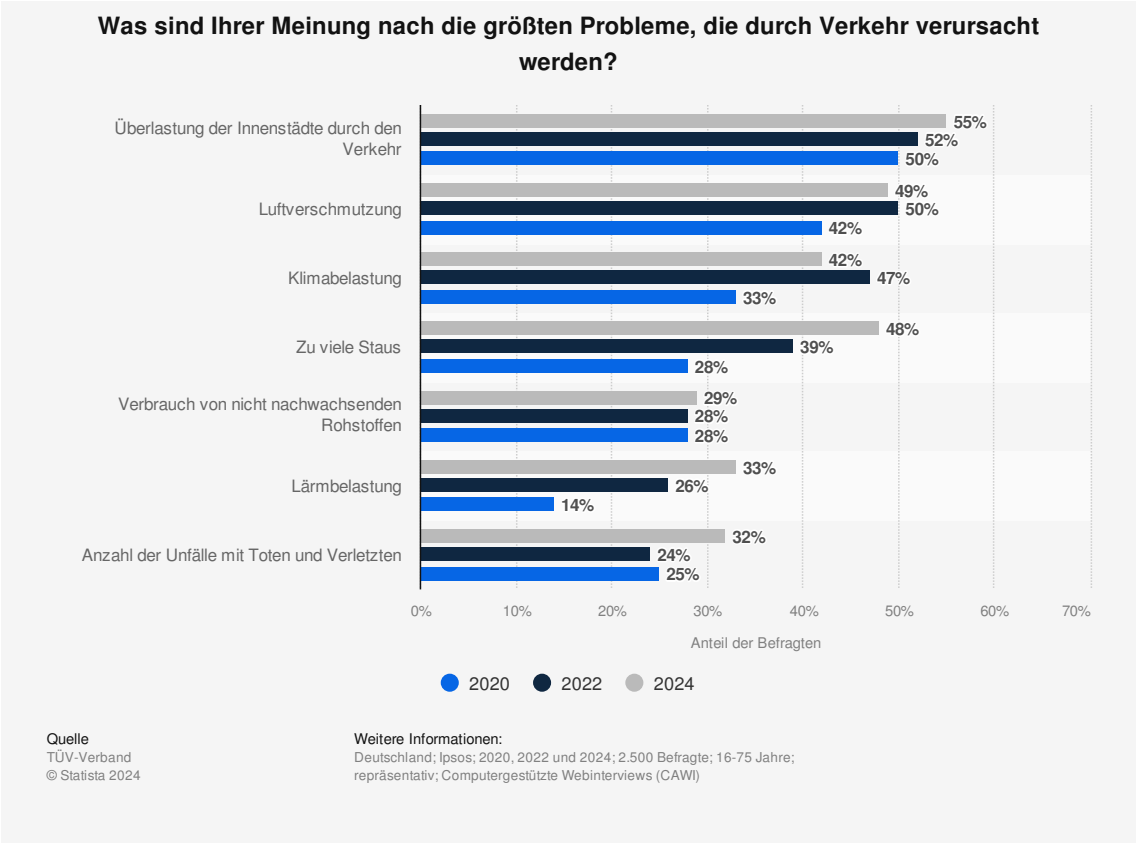

Abbildung 1: Umfrage zu den größten Problemen von Verkehr [36]

Lärm und Luftverschmutzung gehören zu den größten Umweltstressoren. Sie haben negative Auswirkungen auf die Gesundheit und können unter anderem Atemwegserkrankungen, Herz-Kreislauf-Erkrankungen, Schlafstörungen und Stress verursachen. Etwa 20 % der europäischen Bevölkerung sind laut Schätzungen der Weltgesundheitsorganisation (WHO) gesundheitsschädlichen Lärmpegeln ausgesetzt [39]. In Deutschland übersteigt die Anzahl der betroffenen Menschen die 16 Millionen [31].

Bei der Bekämpfung von Luftverschmutzung haben die EU-Richtlinien zwar Fortschritte gezeigt, jedoch bleibt die Lärmbelastung weiterhin ein Problem. Das Ziel der EU-Kommission war, die Anzahl der durch Verkehrslärm betroffenen Menschen bis 2030 um

30 % zu senken. EU-Prüfer halten es für unrealistisch, da die Lärmdaten in den meisten EU-Mitgliedstaaten unregelmäßig oder verzögert erfasst wurden [50][53][56].

Diese Arbeit präsentiert ein Verkehrsüberwachungssystem, das die Erhebung der Daten erleichtert und zugänglicher macht. Das System soll anhand von künstlicher Intelligenz die Motoren- und Reifengeräusche analysieren, um Fahrzeuge zu klassifizieren und zu zählen. Im Gegensatz zu visuell basierten KI-Ansätzen ist diese Methode kostengünstiger, energieeffizienter und bleibt unter erschwerten Umweltbedingungen wie Dunkelheit zuverlässig [19]. Außerdem stellen Bild- und Videoaufnahmen aufgrund der strengen Datenschutzvorgaben in Europa und Deutschland eine rechtliche Hürde dar. Durch den Einsatz von Signalverarbeitungstechniken und modernen Technologien lässt sich das Datenschutzproblem bei dem akustikbasierten Ansatz teilweise beseitigen.

Die Entwicklung von einem System zur Klassifizierung und Zählung von Fahrzeugen ist sowohl eine spannende technische Herausforderung als auch ein wichtiger Schritt für die Verbesserung der Lebensqualität in urbanen Gebieten. Die erfassten Daten können Ländern und Städten insbesondere dabei helfen, überlastete Straßen und Lärmquellen zu identifizieren, gezielte Maßnahmen zur Lärmminderung zu ergreifen und ihre zukünftige Infrastrukturplanung zu optimieren.

1.2 Ziel und Vorgehensweise

Aus der Problemstellung ergibt sich die folgende Forschungsfrage: Wie kann ein kostengünstiges System entwickelt werden, das Fahrzeuge anhand akustischer Signale zuverlässig klassifiziert und zählt? Um diese Frage zu beantworten, werden in dieser Arbeit folgende Teilfragen untersucht:

- Welche rechtlichen Probleme bringt dieses System mit sich, und wie wird damit umgegangen, um den Schutz personenbezogener Daten zu gewährleisten?
- Mit welchen modernen Technologien kann ein solches System entwickelt werden?
- Inwiefern liefert das System zuverlässige Ergebnisse bei realistischen Verkehrsszenarien?

Das Ziel dieser Arbeit besteht darin, ein vollständiges System, bestehend aus Hardware und Software, zu entwickeln und aufzubauen. Das System soll Verkehrslärmsignale in Echtzeit erfassen und in vier Fahrzeugklassen – PKW, LKW, Bus und Motorrad – klassifizieren. Darüber hinaus soll es in der Lage sein, zwischen Verkehr und keinem Verkehr zu unterscheiden und entsprechende Ereignisse zu zählen. Dabei sollen so wenig personenbezogene Daten wie möglich verarbeitet werden. Außerdem soll das System eine kompakte Bauweise aufweisen, um für den Straßeneinsatz geeignet zu sein.

1.3 Aufbau der Arbeit

Diese Arbeit besteht aus 7 Kapiteln. Das erste Kapitel, ,,Einleitung", führt den Leser in die Herausforderungen der Urbanisierung und des überlasteten Verkehrsaufkommens ein und gibt einen Überblick über moderne Lösungsansätze zur Bekämpfung der Lärmbelastung.

In Kapitel 2 werden zentrale Begriffe und notwendige Grundlagen erläutert, die dem Leser ermöglichen, sich ein besseres Verständnis der Arbeit zu verschaffen. Dazu gehören Kon-

zepte aus den Bereichen Datenschutz, Mikrocontroller, Verarbeitung von Audiosignalen, künstliche Intelligenz und maschinelles Lernen.

Im darauf folgenden Kapitel werden relevante wissenschaftliche Arbeiten vorgestellt, die als Grundlage für diese Arbeit dienen. Dabei wird ein Überblick über bestehende Ansätze im Bereich der akustischen Klassifikation verschaffen und deren Methoden und Ergebnisse erläutert.

In Kapitel 4 wird das Konzept des Systems sowie die zugrunde liegende Vorgehensweise beschrieben. Dabei werden die wesentlichen Komponenten sowie die grundlegenden Designentscheidungen vorgestellt, die zur Umsetzung des Gesamtsystems geführt haben. Ziel dieses Kapitels ist es, ein umfassendes Verständnis für die Architektur und die funktionalen Abläufe zu vermitteln.

Die Kapitel 5 und 6 bilden den zentralen Teil dieser Arbeit. In Kapitel 5 erfolgt die Konkretisierung des beschriebenen Konzepts durch die technische Umsetzung des Systems, einschließlich der Implementierung der Audioverarbeitung und des Klassifikationsmodells.

In Kapitel 6 wird das System dann im echten Umfeld untersucht. Dabei werden verschiedene Tests durchgeführt, um herauszufinden, wie gut das System in der Praxis funktioniert, wo es noch Schwächen zeigt und welche Verbesserungen möglich wären.

Im abschließenden Kapitel wird das durch diese Arbeit erworbene Wissen zusammengefasst und ein Ausblick auf zukünftige Arbeiten gegeben.

2 Grundlagen

In diesem Kapitel werden die zentralen Grundlagen erläutert, die für das Verständnis dieser Arbeit notwendig sind. Der Fokus liegt dabei insbesondere auf der Einführung und Erläuterung von Konzepten und Fachbegriffen aus den Bereichen Datenschutz, eingebettete Systeme, Signalverarbeitung sowie künstliche Intelligenz. Das Ziel des Kapitels ist es, dem Leser ausreichend Hintergrundwissen zu vermitteln, um die Ansätze und Technologien, die in den folgenden Kapiteln vorgestellt werden, nachzuvollziehen.

2.1 Datenschutz

Datenschutz bezeichnet den Schutz natürlicher Personen sowohl bei automatisierter als auch bei nicht automatisierter Verarbeitung von personenbezogenen Daten. Das sind alle Informationen, die zur direkten (z.B. Vor- und Nachname) oder indirekten (z.B. Standortdaten) Identifizierung einer natürlichen Person verwendet werden können. Die rechtlichen Grundlagen hierfür sind durch die Datenschutz-Grundverordnung (DSGVO) und das Bundesdatenschutzgesetz (BDSG) bestimmt. Diese Regelungen zielen darauf ab, den Schutz der Grundrechte und Grundfreiheiten zu gewährleisten [4].

Biometrie ist laut ISO und IEC die automatisierte Erkennung von Personen basierend auf ihren biologischen und verhaltensbezogenen Merkmalen. Ein Anwendungsbereich davon ist die Sprachbiometrie (engl. Speaker Recognition). Es handelt sich dabei um die Erkennung von Personen durch die Eigenschaften der Stimme [9].

Da sich ein Teil dieser Thesis mit Tonaufnahmen in öffentlichen Räumen, etwa auf Straßen, befasst, können unbeabsichtigt Stimmen und Gespräche erfasst werden. Somit unterliegt die Arbeit den Regelungen der DSGVO.

Die folgende Tabelle stellt die für diese Arbeit relevanten Anforderungen aus der DSGVO dar:

Referenz	**Inhalt**
Art. 5	- Zweck der Verarbeitung - Datenminimierung - Speicherbegrenzung - Integrität und Vertraulichkeit
Art. 6	- Einwilligung von betroffenen Personen - Rechtmäßigkeit der Verarbeitung
Art. 13	- Speicherdauer
Art. 32	- Sicherheit der Verarbeitung

Tabelle 1: Datenschutztabelle basierend auf der DSGVO

2.2 Eingebettete Systeme

Ein eingebettetes System (engl. embedded System) ist eine Kombination aus Hardware und Software, die speziell für eine bestimmte Anwendung entwickelt und optimiert wurde. Der Zweck ist ein möglichst geringer Energieverbrauch, eine kompakte Bauweise, ein kleiner Programmcode sowie eine kurze Reaktionszeit [30]. Durch den Einsatz eines eingebetteten Systems wird ein Teil der Zielsetzung, die in Kapitel 1.2 erläutert wurde, erfüllt.

2.2.1 Kategorien und Anwendungsbereiche

Basierend auf den Hardwareeigenschaften lassen sich eingebettete Systeme in verschiedene Kategorien unterteilen [37]. Die erste Kategorie umfasst die sogenannten Field Programmable Gate Arrays (FPGAs). Das sind Schaltkreise, die sich auf Gatterebene durch Hardware Description Language (HDL) beschreiben lassen. Sie bieten die höchste Flexibilität an und werden häufig in sicherheitskritischen Anwendungen eingesetzt, wie Flugsteuerungssystemen. Im weiteren Verlauf der Arbeit werden FPGAs nicht mehr berücksichtigt, da sie deutlich kompliziert und aufwendig sind.

Die nächste Kategorie ist die Familie der Mikrocontroller (MCUs). Dabei handelt es sich um kleine Chips, die ohne zusätzliche Hardwarekomponente betriebsfähig sind (Abbildung 2). Sie verfügen über eine begrenzte Rechenleistung und Speicherkapazität, sind jedoch kostengünstig, energieeffizient und lassen sich in High-Level Programmiersprachen wie C/C++ programmieren. Typische Einsatzbereiche hierfür sind Haushaltgeräte und Internet of Things (IoT). Mikrocontroller werden in der Regel mit weiteren Modulen wie WLAN/Bluetooth erweitert. Diese werden häufig auch System on Chip (SoC) genannt.
Beispiel: Arduino, ESP32, STM32

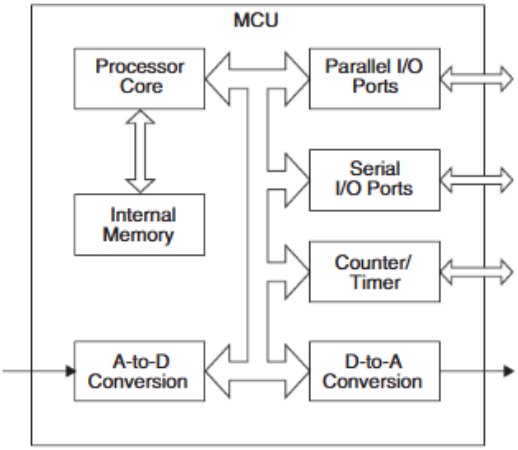

Abbildung 2: Mikrocontroller und interne Funktionseinheiten [30]

Die letzte Kategorie umfasst die Mikroprozessoren. Sie sind leistungsfähiger als Mikrocontroller, jedoch teurer und energieintensiver. Erforderliche Funktionseinheiten wie Speicher und Peripherie müssen separat angebunden werden. Mikroprozessorbasierte Systeme werden häufig als Single Board Computers (SBCs) genannt und verwenden typischerweise ein vollständiges Betriessystem, meist auf der Basis von Linux. SBCs können sowohl als einfache Steuerungssysteme (ähnlich zu MCUs) als auch für rechenintensivere Aufgaben (z.B. Objekterkennung mit KI) eingesetzt werden.
Beispiel: Raspberry Pi Zero, Raspberry Pi 5

2.2.2 Komponenten

Die Hauptkomponenten eines typischen eingebetteten Systems sind auf Abbildung 2 zu erkennen. Die zentrale Einheit dabei ist der Prozessorkern (engl. *Processor core*), der für die Ausführung von Befehelen und Steuerung des Programmflusses zuständig ist. Außerdem verfügt es über einen schnellen Speicher (RAM), wo ausgeführte Programme und Daten temporär gespeichert werden. ROM ist ein langsamer Speicher, der dauerhaft Daten wie

Firmware speichert. Parallele und Serielle I/O sind Schnittstellen, die zur Eingabe (z.B. Sensoren) und Ausgabe (z.B. Display) von Daten dienen. Bei analogen Ein- und Ausgaben (Spannungen) kommen die A/D bzw. D/A Converter zum Einsatz (kurz Analog-Digital Converters (ADCs)/Digital-Analog Converters (DACs)). Sie wandeln analoge Signale in digitale Signale um und umgekehrt. Ein Counter/Timer misst Zeitintervalle, zählt Ereignisse und sorgt für die Steuerung von zeitabhängigen Abläufen (z.B Blinkfrequenz einer LED) [37].

Moderne eingebettete Systeme verfügen zudem über einen Interrupt-Controller. Dieser ist dafür verantwortlich, den normalen Programmablauf zu unterbrechen, um auf interne oder externe Ereignisse zu reagieren. In Kombination mit dem Sleep-Modus lässt sich der Energieverbrauch erheblich reduzieren [34].

2.3 Signalverarbeitung

Ein Signal ist ein Medium zur Übertragung von Informationen von einer Quelle zu einem Empfänger. Es übermittelt beispielsweise Befehle in Fernbedienungssystemen sowie Daten, wie Sprache oder Bilder über Netzwerke. Die Signalverarbeitung umfasst Operationen und arithmetische Berechnungen, die auf das Signal angewendet werden. Das Ziel davon ist es, Rauschen zu minimieren, unerwünschte Verzerrungen zu korrigieren und eine verbesserte Darstellung des Signals zu erzeugen [28][45].

Signalverarbeitung ist ein wesentlicher Bestandteil der akustischen Klassifizierung von Fahrzeugen, da sie Rohsignale aus Lärmsensoren analysierbar macht.

2.3.1 Signalformen und Digitalisierungsprozess

Generell können Signale in 3 Formen auftreten [23][18]. Die erste Form ist das zeitkontinuierliche Signal (engl. *continuous-time Signal*), das zu jedem beliebigen Zeitpunkt einen Wert hat. Diese werden häufig auch als analoge Signale bezeichnet. Abbildung 3A zeigt ein Beispiel eines solchen Signals, welches durch die folgende Funktion beschrieben wird:

$$f(t) = 5\sin(2\pi \cdot 100 \cdot t) + \cos(2\pi \cdot 400 \cdot t) + 0.5\sin(2\pi \cdot 800 \cdot t) \tag{1}$$

In der Praxis ist es nicht möglich, zeitkontinuierliche Signale in ihrer originalform aufzuzeichnen und auf einem digitalen Gerät zu speichern. Stattdessen wird das Signal in kurzen und festen Zeitabständen abgetastet (engl. *Sampling*). Dadurch entsteht ein zeitdiskretes Signal (engl. *discrete-time Signal*), wie Abbildung 3B veranschaulicht. Um Informationsverluste zu vermeiden, muss die Abtastrate mindestens doppelt so hoch sein wie die höchste Frequenz im analogen Signal (Nyquist–Shannon Sampling Theorem).

Ein zeitdiskretes Signal besitzt weiterhin kontinuierliche Werte auf der y-Achse. Aufgrund der begrenzten Bitanzahl des ADCs kann es jedoch in digitalen Systemen nur eine endliche Anzahl an Werten annehmen. Daher müssen die abgetasteten Werte quantisiert werden. Das bedeutet, die Amplitudenwerte werden auf die nächstliegende darstellbare Zahl gerundet (3C). Abschließend werden die quantisierten Werte in Bits codiert. Somit wird ein digitales Signal produziert, das sich mithilfe digitaler Systeme weiterverarbeiten lässt.

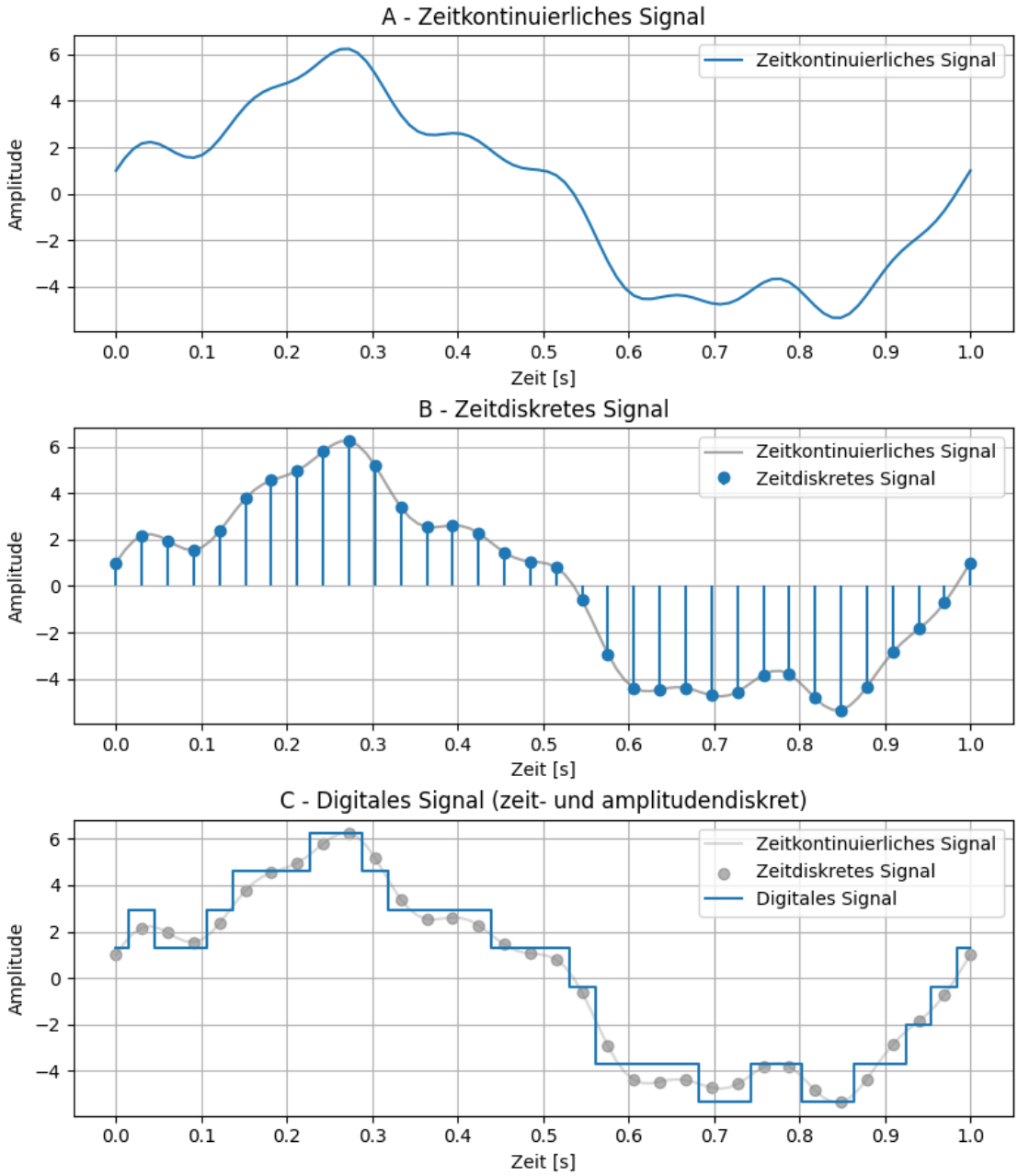

Abbildung 3: Beispiele von Signalformen [eigene Darstellung, Anhang 1]

Die Diagramme in Abbildung 3 wurden mit Python erstellt und können anhand des Beispielcodes in Anhang 1 reproduziert werden.

2.3.2 Die Fourier-Transformation

Die Fourier-Transformation (FT) ist ein leistungsstarkes mathematisches Werkzeug, das dazu dient, Signale vom Zeitbereich in den Frequenzbereich zu überführen. Dadurch lassen sich Frequenzkomponenten von komplexen Signalen leichter erkennen und vearbeiten. In der Akustik und Lärmanalyse ermöglicht die FT die Identifizierung von Lärmquellen [42].Abbildung 4 stellt das Frequenzspektrum des Signals dar, das in Gleichung 1 im vorherigen Abschnitt beschrieben wurde. In der Gleichung lassen sich die drei Frequenzen (100,

400 und 800 Hz) leicht erkennen. In der Praxis sind Signale jedoch deutlich komplexer, und eine explizite Formel liegt meist nicht vor.

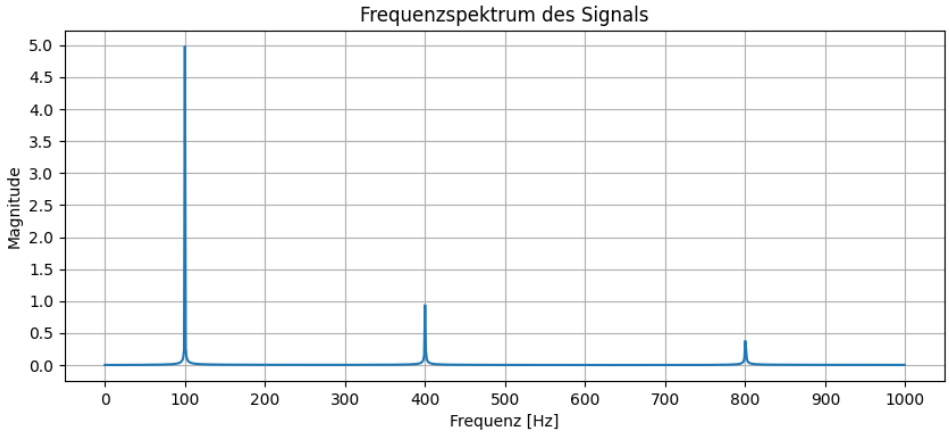

Abbildung 4: Frequenzspektrum des Signals mit der Gleichung (1) [eigene Darstellung, Anhang 2]

Die Transformation eines diskreten (oder digitalen) Signals in den Frequenzbereich erfolgt mithilfe der folgenden mathematischen Formel:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi}{N}kn} \tag{2}$$

wobei:

- X[k]: komplexer Wert der Frequenzkomponente bei Index k
- k: Frequenzindex
- N: Anzahl der diskreten Abtastwerten
- x[n]: Werte des diskreten Signals im Zeitbereich
- j: imaginäre Einheit mit $j^2 = -1$

2.3.3 Beispielanwendungen im Frequenzbereich

Filterung

Die Filterung von Signalen ermöglicht die Verstärkung oder Unterdrückung von bestimmten Frequenzkomponenten. Die am häufigsten eingesetzten Filter in der Signalverarbeitung sind Bandpass-, Tiefpass- (engl. lowpass) und Hochpassfilter (engl. highpass). Während Tiefpassfilter niedrige Frequenzen passieren lassen und hohe Frequenzen blockieren, verhalten sich Hochpassfilter umgekehrt. Bandpassfilter hingegen kombinieren die Eigenschaften von Tiefpass- und Hochpassfiltern und lassen nur Frequenzen in einem bestimmten Bereich durch [42]. Die folgende Abbildung zeigt ein Beispiel aus der Telekommunikation, in dem Sprachsignale beim Telefonieren gefiltert werden.

- (a): Gesamte Bandbreite der Sprachsignale von 80 Hz bis 7 kHz.

- (b): Gefilterte Bandbreite von 300 Hz bis 3,5 kHz.

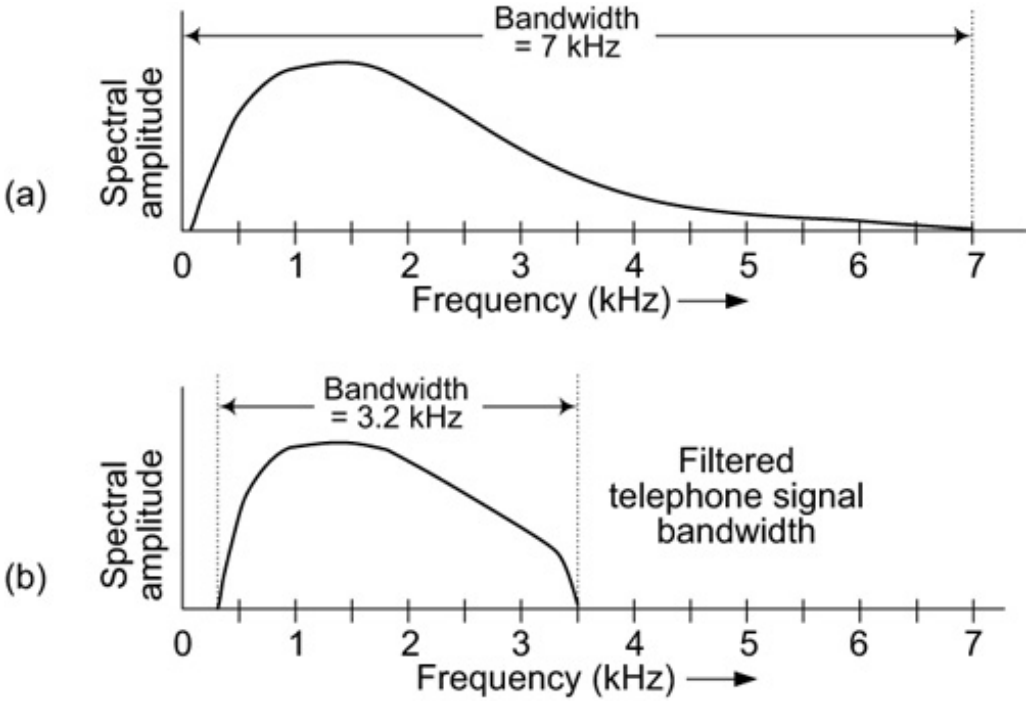

Abbildung 5: Beispiel der Filterung eines Telefonats [1]

Merkmalextraktion

Die Merkmalextraktion (engl. *feature extraction*) ist eine Technik zur Reduzierung der Dimensionalität von Rohdaten und dem Extrahieren von verborgenen relevanten Informationen. Dies reduziert erheblich den Aufwand der Weiterverarbeitung von Daten. Eins der wichtigsten Merkmale sind die Mel-Features Cepstral Coefficients (MFCCs). Um die MFCCs zu berechnen, werden zuerst Spektrogramme erstellt. Diese entstehen mithilfe der Kurzzeit-Fourier Transformation (engl. *Short-Time Fourier Transformation*, STFT). Der Algorithmus unterteilt das Signal in kleine Zeitfenster (engl. *framing*), typischerweise zwischen 20 und 30 ms mit einer Überlappung von 10 ms, und multipliziert jeden Abschnitt mit einer Fensterfunktion (engl. *windowing*), um Unstetigkeiten an Rändern zu vermeiden. Als nächstes wird für jedes Zeitfenster anhand der Fourier-Transformation das Frequenzspektrum konstruiert. Im nächsten Schritt wird das Spektrogramm in die sogenannt Mel-Skala überführt. Das ist eine logarithmische Skala, die die menschliche Wahrnehmung von Ton abbildet. Als letztes wird die *discrete cosine tranformation* (DCT) auf die Spektrogramme angewendet, um die MFCCs zu berechnen [14][21].

2.4 Künstliche Intelligenz

Künstliche Intelligenz (engl. Artificial Intelligence) ist eine Technologie, mit der die kognitiven Fähigkeiten von Menschen bei Computern und Maschinen simuliert werden können. Beispielsweise sind KI-Systeme in der Lage, Objekte zu erkennen, Sprache zu verstehen und eigenständig zu handeln. Ein zentraler Bestandteil der KI ist das maschinelle Lernen (engl. Machine Learning, ML). Das ist der Prozess, bei dem Algorithmen aus Daten lernen, um Vorhersagen oder Entscheidungen zu treffen [29].

In dieser Arbeit soll eine KI-gestüzte Lösung entwickelt werden. Die Grundlagen sollen dabei helfen, das Verständnis der Arbeit und der Funktionsweise des Systems zu erleichtern.

2.4.1 Paradigmen des maschinellen Lernens

Grundsätzlich gibt es drei Paradigmen von ML : überwachtes Lernen (engl. *supervised learning*), unüberwachtes Lernen (engl. *unsupervised learning*) und bestärkendes Lernen (engl. *reinforcement learning*) [22][43]. Bei *supervised learning* werden einem Algorithmus annotierte Datensätze bereitgestellt, die eine eindeutige Beziehung zwischen Ein- und Ausgabe aufweisen. Ziel ist es, dem Modell beizubringen, eine Ausgabe (Vorhersage) für eine unbekannte Eingabe zu erzeugen. Ein Anwendungsbereich hierfür ist die Klassifizierung von Fahrzeuggeräuschen. Das Modell wird mit verschiedenen Fahrzeuggeräuschen trainiert und lernt dabei den Unterschied. Anschließend kann es ein neues Geräusch einer Klasse zuordnen, z.B. PKW.

Bei *unsupervised learning* handelt es sich um eine selbstständige Gruppierung von Daten innerhalb eines nicht-annotierten Datensatzes. Dieser Ansatz wird häufig zur Anomalieerkennung eingesetzt. Beispielsweise wird das Modell mit verschiedenen Fahrzeuggeräuschen trainiert und versucht normale Geräusche und abweichende Geräusche in unterschiedliche Cluster zu unterteilen.

Reinforcement learning bezeichnet den Lernprozess durch kontinuierliche Interaktion mit der Umgebung. Das Modell versucht durch das Ausprobieren von verschiedenen Möglichkeiten, eine optimale Verhaltensstragie zu entwicklen. Zum Beispiel lernt ein intelligentes System Schach zu spielen, in dem es Millionen Spiele gegen sich selbst spielt.

Im weiteren Verlauf der Arbeit werden nur Algorithmen des überwachten Lernen betrachtet, da die restlichen Paradigmen für das Ziel dieser Arbeit nicht relevant sind.

2.4.2 Supervised Learning - Modellansätze

Lineare Modelle

Das sind eine Kategorie von Algorithmen, die häufig für Regressionsanalyse und Kurvenanpassungen verwendet werden. Ziel ist es, optimale Parameter einer linearen mathematischen Funktion zu bestimmen, die die Trainingsdaten möglichst gut beschreiben (Abbildung 6) [43]. Zum Beispiel trifft das Modell anhand der Wohnfläche und des Orts eine Vorhersage über den Hauspreis.

Entscheidungsbäume

Entscheidungsbäume (engl. *Decision Trees*) verwenden eine baumartige Struktur für den Entscheidungsprozess. Sie bestehen aus einer Wurzel (engl. root), inneren Knoten (engl. inner nodes) und Blattknoten (engl. leaf nodes). Die Trainingsdaten werden während des Trainingsprozesses anhand ausgewählter Attribute aufgeteilt. Ziel ist es, möglichst homogene (reine) Teilmengen zu erzeugen [43]. Die folgende Abbildung 7 repräsentiert einen einfachen Entscheidungsbaum mit 2 Zielklassen, der entscheidet, ob ein Auto gekauft werden soll.

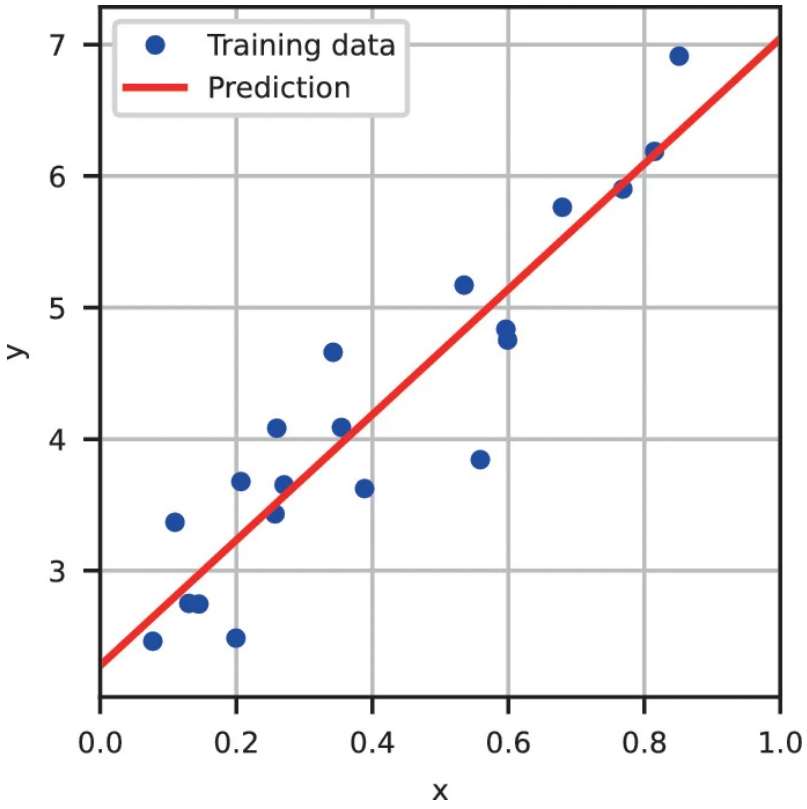

Abbildung 6: Ziel eines einfachen linearen Modell [25]

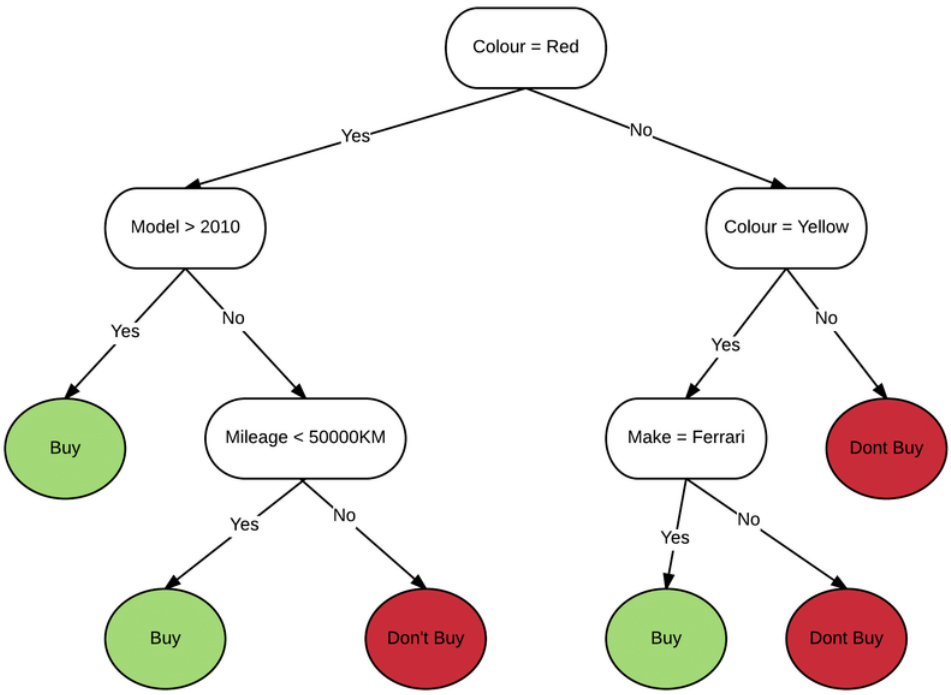

Abbildung 7: Entscheidungsbaum für den Kauf eines Autos [3]

Künstliche Neuronale Netze

Künstliche neuronale Netze (engl. *Artificial Neural Networks*, ANNs) sind Modelle, die vom biologischen Gehirn und den Nervenzellen inspiriert wurden. Ziel ist es, komplexe Muster und Zusammenhänge in Daten zu erkennen. Ein typisches ANN besteht aus einer Eingabeschicht, verborgenen Schichten (engl. *Hidden Layers*) und einer Ausgabeschicht. Das einfachste Modell eines ANNs ist das Perzeptron (Abbildung 8), das von Frank Rosenblatt 1958 vorgestellt und 1960 als Computerprogramm implementiert wurde [43][5].

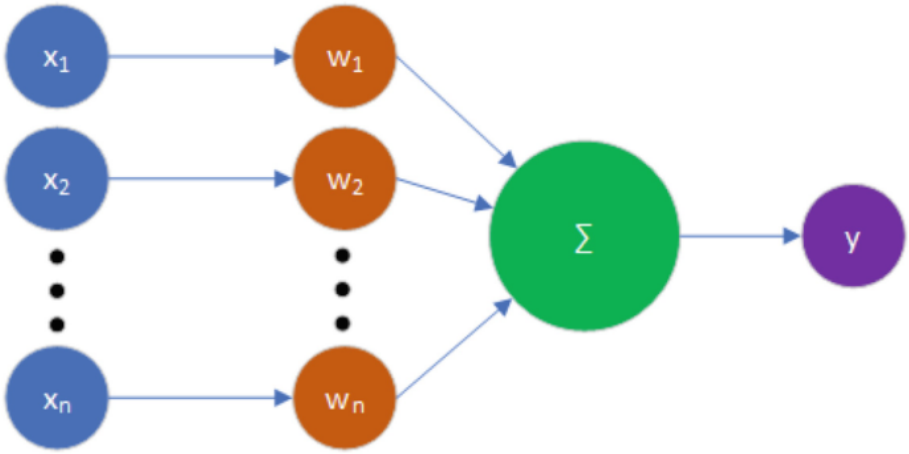

Abbildung 8: Perzeptron [16]

Um komplexere Zusammenhänge abzubilden, wird das Perzeptron um mehrere Schichten erweitert. Dieses Modell wird als MLP (*Multilayer Perceptron*) bezeichnet und bildet die Grundlage vieler moderner Deep-Learning-Modelle [2].

2.4.3 Deep Learning

Deep Learning ist ein Teilgebiet des maschinellen Lernens, das auf künstlichen neuronalen Netzen mit mehreren Schichten basiert. Typische Modelle sind Convolutional Neural Networks (CNNs) und Recurrent Neural Networks (RNNs) [16].

CNNs verfügen über eine Faltungsschicht (engl. *convolution layer*), die aus einer Serie von zweidimensionalen Filtern besteht. Die Filter werden über die ganze Eingabe geschoben und lernen dabei charakteristische Merkmale. Dadurch entsteht eine dreidimensionale Ausgabe (Stapel von *Feature Maps*). Die Ausgabe wird als nächstes durch die ReLU-Aktivierungsfunktion[1] ($f(x) = max(0, x)$) transformiert, indem alle negativen Werte auf 0 gesetzt werden. Um die Rechenlast zu senken, reduziert die Pooling-Schicht die Dimension der fließenden Daten. Die wichtigsten gewonnenen Informationen werden dabei erhalten. Die letzte Schicht des Netzwerks ist die sogenannte Fully Connected Layer. Sie ähnelt der Struktur eines MLP und dient dazu, die finale Vorhersage zu generieren. CNNs eignen sich besonders gut für statische Daten, die sich zeitlich nicht verändern, wie etwa Bilder oder Spektrogramme [16][43].

[1] Aktivierungsfunktionen führen Nicht-Linearität in das Modell ein, um komplexere Muster zu erfassen. Ohne Aktivierungsfunktionen wären ANNs nicht anders als ein lineares Modell.

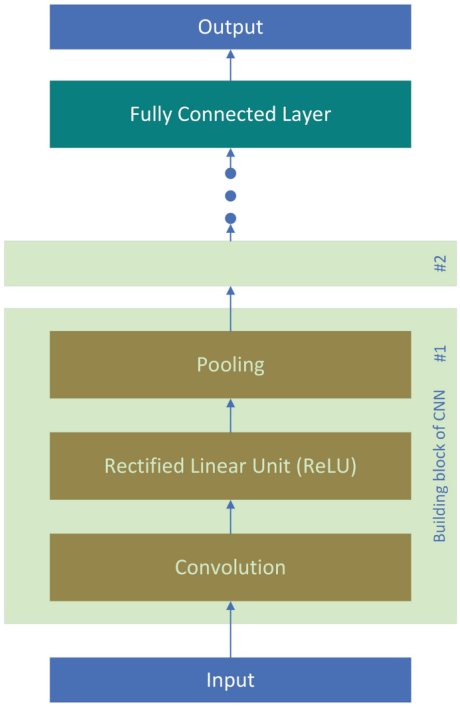

Abbildung 9: CNNs Architektur [16]

Im Gegensatz zu CNNs sind RNNs speziell dafür konzipiert, sequentielle oder zeitabhängige Daten (z.B. Sprache) zu verarbeiten. RNNs verfügen über einen zeitabhängigen Zustand, der in jedem Zeitschritt anhand folgender Funktion aktualisiert wird:

$$H_t = f_H(U \cdot X_t + W \cdot H_{t-1})$$

wobei:

- f_H: Aktivierungsfunktion
- U: Gewichtsmatrix für die aktuelle Eingabe X_t
- W: Gewichtsmatrix für den vorherigen Zustand H_{t-1}

Trotz ihrer Stärke bei der Erkennung komplexer Muster bringen Deep-Learning Modelle einige Herausforderungen mit sich. Die zwei Hauptursachen für schlechte Leistung sind das *Overfitting* und das *Underfitting* Problem. Overfitting tritt auf, wenn ein Modell die Trainigsdaten zu genau lernt und dadurch unzuverlässig auf neue Daten generalisiert. Underfitting hingegen entsteht, wenn ein Modell zu simpel ist und selbst Muster in Trainigsdaten nicht ausreichend erlernt. Zur Verhinderung dieser Probleme stehen verschiedene Ansätze zur Verfügung, darunter die sogennante *Cross-Validation*. Dabei wird ein Datensatz in k Teilmengen unterteilt. Anschließend wird das Modell über mehrere Epochen[2] mehrfach mit $k-1$ Teilmengen trainiert und mit einer der verbleibenden Teilmenge getestet [43]. Ein weiteters Problem kann auftreten, wenn während der *Backpropagation*[3] die Gewichte der ersten Schichten kaum oder nicht mehr aktualisiert werden. Dieses Phönomen wird als *vanishing gradients* bezeichnet. Eine potenzielle Lösung zu diesem Problem sind die Long Short-Term Memorys (LSTMs)-RNNs. Darüber hinaus bieten LSTM-Netzwerke viele Vorteile und werden häufig in modernen RNN-Anwendungen verwendet [16].

[2] vollständiger Durchlauf durch den gesamten Datensatz
[3] Die Fehler werden rückwärts propagiert, um die Gewichte der früheren Schichten anzupassen

2.4.4 Evaluierung von KI-Modellen

Die Bewertung von Modellen erfolgt anhand einer Konfusionsmatrix (engl. confusion matrix) [11]. Bei einer binären Klassifikation besitzt diese Matrix die Dimension 2x2 (Tabelle 2).

Tatsächlich \ Vorhergesagt	Positiv	Negativ
Positiv	True Positive (TP)	False Negative (FN)
Negativ	False Positive (FP)	True Negative (TN)

Tabelle 2: Konfusionsmatrix für binäre Klassifikation

- True Positive (TP): Probe ist positiv, Vorhersage ist positiv
- False Positive (FP): Probe ist negativ, Vorhersage ist positiv
- True Negative (TN): Probe ist negativ, Vorhersage ist negativ
- False Negative (FN): Probe ist positiv, Vorhersage ist negativ

Aus der vorherigen Matrix können folgende Metriken abgeleitet werden:

- **Genauigkeit** (engl. *accuracy*) $= \frac{TP+TN}{TP+FP+FN+TN}$
  Gibt an, wie viele Vorhersagen insgesamt korrekt waren.
- **Präzision** (engl. *precision*) $= \frac{TP}{TP+FP}$
  Gibt an, wie viele der als positiv vorhergesagten Fälle tatsächlich positiv waren.
- **Sensitivität** (engl. *recall* oder *sensitivity*) $= \frac{TP}{TP+FN}$
  Zeigt, wie gut das Modell tatsächliche positive Fälle erkennt (Trefferquote).
- **Spezifität** (engl. *specificity*) $= \frac{TN}{TN+FP}$
  Gibt an, wie gut das Modell tatsächliche negative Fälle korrekt erkennt (Vermeidung von Fehlalarmen).
- **F1-Score** $= 2 \cdot \frac{\text{Präzision} \cdot \text{Recall}}{\text{Präzision} + \text{Recall}}$
  Harmonisches Mittel aus Präzision und Sensitivität, nützlich bei nicht einheitlicher Verteilung der Klassen.

2.4.5 TinyML

KI-Modelle benötigen in der Regel eine hohe Rechenleistung und einen hohen Energieverbrauch. Aus diesem Grund werden sie typischerweise auf leistungsstarken (Cloud)-Servern ausgeführt. Eingebettete Systeme wurden traditionell nur dazu genutzt, Sensordaten zu erfassen und an die Server weiterzuleiten, wo anschließend die Inferenz durchgeführt wird.

TinyML ist ein Teilgebiet des maschinellen Lernens, das sich mit der Ausführung von KI-Modellen auf eingebetetten Systemen beschäftigt. Durch spezielle Optimierungs- und Komprimierungstechniken lassen sich Inferenzen direkt auf einem ressourcebschränkten Gerät ausführen, ganz ohne Verbindung zu einem Cloud-Server. Dadurch werden Latenzzeiten erheblich reduziert und die Privatsphäre verbessert [20].

3 Stand der Forschung und technischer Hintergrund

Nachdem das Grundlagenwissen im vorherigen Kapitel vermittelt wurde, lässt sich nun den Stand der Forschung größtenteils nachvollziehen. Dieses Kapitel behandelt Ansätze sowohl zur allgemeinen Klassifizierung von akustischen Signalen, als auch im speziellen zur Klassifizierung von Fahrzeuggeräuschen. Die wissenschaftlichen Arbeiten werden je nach Schwerpunkt in 2 Kategorien unterteilt: Studien mit Fokus auf Architekturen von KI-Modellen und Studien mit Fokus auf Optimierung der KI-Modelle für den Einsatz auf eingebetteten Systemen.

3.1 Schwerpunkt - Deep Learning und Architekturen

Der Artikel von Michele Valenti et al. aus dem Jahr 2017 beschreibt einen Ansatz für die Klassifizierung akustischer Signale mithilfe von CNNs [6]. Das Modell wurde mittels Mel-Spektrogrammen trainiert. Die vorgeschlagene Trainingsmethode besteht aus 2 Phasen, ,,Non-Full Training" und ,,Full Training". In der ersten Phase werden die Daten in 2 Teilmengen aufgeteilt, 80 % für das Training und 20 % für die Validierung. Zeigt sich nach 100 Epochen keine Verbesserung der Validierungsleistung , wird das Modell in der zweiten Phase mit dem gesamten Datensatz erneut trainiert. Diese Methode ist besonders hilfreich, wenn nur ein kleiner Datensatz zur Verfügung steht. Die besten Ergebnisse des Systems wurden bei dem DCASE2016-Datensatz erreicht. Das System erzielte dabei eine Genauigkeit von 86,2 %, was 6 % mehr als die des Basissystems ist.

Die Studie von Yuxi Luo et al. aus dem Jahr 2021 stellt einen hybriden Ansatz vor [13]. Die Methode kombiniert CNNs und RNNs in einer sogenannten Sound-Convolutional Recurrent Neural Networks (S-CRNN). Die vorgeschlagene Architektur besteht aus drei CNN-Schichten, gefolgt von einer RNN-Schicht. Zwei Varianten wurden getestet: S-CRNN1, die Gated Recurrent Unit (GRU) als RNN-Schicht verwendet, und S-CRNN2, die auf LSTM basiert. Beide wurden mit dem originalen Modell S-CNN verglichen, das nur auf CNNs basiert. Die Autoren haben sich für Mel-Spektrogramme als Eingabemerkmale entschieden. CRNN2 erzielte einen F1-Score von 28,5 %, leicht besser als S-CRNN1 (28,4 %), während S-CNN nur 7,5 % erreichte. Dies deutet darauf hin, dass die Kombination unterschiedlicher Modelle die Leistung verbessern kann.

Der Artikel von Qu et al. aus dem Jahr 2022 stellt einen innovativen Multi-Channel Multi-Input Ansatz vor, der auf einer Kombination aus 3D-CNNs und SE-ResNets basiert [17]. Die 3D-CNNs können sowohl räumliche als auch zeitliche Abhängigkeiten erfassen. Parallel dazu kommt ein ResNet, das mit einem Attention-Mechanismus ,,Squeeze and Excitation (SE)" ausgestattet wurde. Dies dient dazu, die relevantesten Merkmale des Signals zu erlernen. Die Eingabe besteht aus einem Stapel verschiedener Signaldarstellungen: Frequenzspektren, Mel-Spektrogramme, Log-Mel-Spektrogramme und Chromatogramme. Dies ermöglichte es dem Modell, mehrere Aspekte des Audiosignals gleichzeitig zu verarbeiten. Die Ausgaben von dem 3D-CNN und SE-ResNet werden anschließend kombiniert. Für die Verbesserung der Leistung wurde zusätzlich die Datenaugmentierung genutzt. Das System erreichte bei dem Evaluierungsdatensatz von DCASE 2019 eine durchschnittliche Genauigkeit von 86,4 % und übertraf andere State-of-the-Art Modelle.

Der Konferenzbeitrag von Lam Pham et al. aus dem Jahr 2022 präsentiert ein Framework mit geringer Komplexität. Zuerst werden die Audiodaten in 3 unterschiedliche Spektrogramme transformiert: Mel-, Grammatone- und CQT-Spektrogramme (Constant Q Transformation). Anschließend wird jedes Spektrogramm einem eigenen CNN-Modell zugeführt.

Im letzten Schritt werden die Vorhersagen kombiniert, um eine finale Klassifizierung zu erzeugen. Die Ergebnisse des Modells übertrafen das DCASE-Basissystem um 19 %. Es erreichte eine Genauigkeit von 66,7 % und belegte den 6. Platz im Wettbewerb.

Ein Open-Benchmark Datensatz unter dem Namen ,,IDMT-Traffic" wurde 2021 vom Jakob Abeßer et al. veröffentlicht [10]. Er umfasst Aufzeichnungen von vier verschiedenen Fahrzeugtypen: PKW, LKW, Bus und Motorrad. Die Aufzeichnungen wurden sowohl mit hochwertigen Mikrofonen als auch mit Mikrofonen mittlerer Qualität aufgenommen. Zudem wurden in dieser Arbeit mehrere bestehende Modelle (VGGNet, ResNet und SqueezeNet) getestet und verglichen. VGGNet erzielte dabei die besten Ergebnisse. Basierend auf diesen Ergebnissen haben die Autoren des Artikels ,,Improved Vehicle Sub-type Classification for Acoustic Traffic Monitoring" im Jahr 2023 gezeigt, dass eine einfache optimisierte CNN Architektur State-of-the-Art Modelle übertreffen kann [19]. Das Modell besteht aus 4 CNN-Schichten, einer Flatten-Schicht und 3 Fully Connected Layers. Die Optimisierung in dieser Arbeit wurde an den Daten durchgeführt. Anhand der Root Mean Square Errors (RMSEs)-Formel und der Datenaugmentierung wurden schlechte Proben entfernt und neue Proben generiert. Außerdem wurde das Modell mit drei verschiedenen Merkmalen untersucht: MFCCs, GFCCs und Mel-Spektrogramme. Das auf MFCCs basierte Modell erreichte mit 98.95 % die höchste Genauigkeit. Die F1-Scores übertrafen die Ergebnisse von VGGNet. In der LKW-Klasse wurde eine Verbesserung von 42 % erzielt.

Der Artikel von Ahmed Ihsan Yassin et al. aus dem Jahr 2023 untersucht die akustische Erkennung von Fahrzeugen (PKW, LKW, Motorrad, Kein_Verkehr) [27]. Die Autoren haben ein kostengünstiges System vorgestellt, das auf einem LSTM-Netzwerk (Long Short-Term Memory) basiert. LSTM-Netzwerke sind eine Weiterentwicklung des RNNs. Sie bieten eine Lösung zum verschwindenden Gradient Problem. Bei klassischen RNNs führen langfristige Abhängigkeiten dazu, dass die Gewichte kaum noch aktualisiert werden [41]. In dieser Studie wurden die MFCCs als Eingabemerkmale verwendet. Die Autoren haben sich für die ersten 13 Koeffizienten entschieden, die die wichtigsten Eigenschaften des Signals repräsentieren. Das System erreichte eine Genauigkeit von über 90 % in der Unterscheidung zwischen PKW und Kein_Verkehr, zeigte jedoch eine deutlich schlechtere Leistung bei LKWs und Motorrädern. Je nach LSTM-Parametern (z.B. Anzahl an verborgenen Schichten) lag die gesamte Genauigkeit zwischen 82% und 86,2 %.

Ähnlich zum IDMT-Traffic Datensatz haben Muhammed Asif und seine Kollegen im Jahr 2022 einen Datensatz veröffentlicht, der zur Unterscheidung zwischen Notfallsirenen und normalem Verkehr dient [15]. Die Daten stammen aus verschiedenen Quellen: Kameras, Online Quellen sowie aus gezielt eingesetzten Sirenen zur aktiven Datenerhebung. Um die Qualität und Einheitlichkeit der Audioaufnahmen zu gewährleisten, wurden verzerrte Proben entfernt und die verbleibenden auf eine Samplingrate von 44 kHz normiert. Anschließed wurden unter anderem die MFCCs sowie weitere Merkmale extrahiert und zur Verfügung gestellt. Auf Basis dieser Merkmale wurde ein Modell trainiert, das lediglich aus MLPs besteht. Das System erreichte 97 % Genauigkeit auf dem Validierungsdatensatz. Auf Grundlage dieser Arbeit haben Ahsan Shabbir et al. im Jahr 2024 einen Ensemble-Deep-Learning Ansatz vorgeschlagen [35]. Das System setzt sich aus MLPs, DNNs und LSTMs zusammen. Zuerst werden Die MLPs und DNNs trainiert und validiert. Deren Ausgabe wird im nächsten Schritt kombiniert und als Eingabe an das LSTM-Netzwerk weitergegeben, das für die finale Klassifizierung zuständig ist. Anhand dieser Architektur konnte eine Genauigkeit 98,46 % erzielt werden.

3.2 Schwerpunkt - TinyML und Komprimierung

Der Konferenzbeitrag von Pietro Montino und Danilo Pau aus dem Jahr 2019 beschäftigt sich mit der Entwickung eines Prototyps zur Verkehrsüberwachung in städtischen Gebieten [8]. Dabei wurde ein CRNN-basiertes Modell mit Tensorflow Keras trainiert, um drei Klassen zu erkennen: *approaching vehicle*, *leaving vehicle* und *None*. Als Merkmalseingaben wurde MFCCs eingesetzt. Die Autoren verwendeten das Sensortile development kit von STMicroelectronics, das unter anderem mit dem STM32L476JG-Mikrocontroller und einem MEMS-Mikrofon ausgestattet ist. Zur Optimierung und Quantisierung des KI-Modells wurde das Tool STM32Cube.AI genutzt, welches den Integrationsprozess auf STM-Mikrocontrollern erleichtert. Die Daten waren jeweils 0,5 Sekunden lang und wurden mit einer Abtastrate von 12 kHz aufgezeichnet. Das System erreichte eine Genauigkeit von 97.5 % auf dem Validierungsdatensatz. Zudem wurde das System in Städten getestet und zeigte sich als robust. Die genauen Ergebnisse der Feldtests wurden jedoch nicht angegeben.

Für die Klassifizierung von Umgebungsgeräuschen in einer lauten Umgebung präsentierten Steven Wyatt und seine Kollegen 2021 ein kostengünstiges System , das auf einem komprimierten BERT-Modell basiert (BERT-based tiny transformer) [12]. Die Quanitiserung des Modells erfolgte mithilfe von Tensorflow Lite. Das Ziel dieser Arbeit war, ein robustes System zu entwickeln, dass unter realen Umweltbedingungen zuverlässige Ergebnisse liefert. Hierfür wurden Trainigsdatensätze, die mit einer Abstastrate von 44 kHz vorlagen, mit verschiedenen Lärmgeräuschen erweitert. Die Geräusche wurden mit einem preisgünstigen Mikrofon (KISEER 2 Pcs USB 2.0 Mini microphone) aufgezeichnet, das an einem Raspberry Pi Zero angeschlossen war. Schließlich wurden die Tests mit den geräuschvollen Daten direkt auf dem Pi Zero durchgeführt, das mit einem 1 GHz Single-Core Prozessor und 512MB RAM ausgesttet ist. Die Ergbenisse zeigten eine höhere Genauigkeit im Vergleich zum Modell, das ohne Geräusche trainiert wurde.

Im Konferenzbeitrag von Bhattarapong Somwong et al. im Jahr 2023 basiert das System zur Klassifizierung von Waldgeräuschen und zur Gefahrerkennung auf dem leistungsstarken Arduino Mikrocontroller Portenta H7 [26]. Dieses Entwicklungsboard verfügt über einem Dual-Core Prozessor (ARM-Cortex M4 und M7) und wurde speziell für industrielle IoT-Anwendungen und Echtzeitsysteme konzipiert. Die Audioaufzeichnungen erfolgten mithilfe von Portenta H7 Vision Shield, das mit mehreren Sensoren sowie zwei Mikrofonen ausgestattet ist. Für die Klassifizierung entschieden sich die Autoren für ein CNN basiertes Modell in Kombination mit Mel-Spektrogrammen. Die Modellentwicklung, Merkmalextraktion und das Deployment erfolgten über Edge-Impulse, eine Platform, die das Training und Bereitstellung bestehender KI-Modelle ermöglicht. Die Gefahrgeräusche wie Ketten-, Handsäge oder Schüsse sind schwer reproduzierbar. Aus diesem Grund wurden zwei Python-Skripte entwicklt: eines zum Abspielen der Audiodaten und eines zur Erfassung der Ergebnisse. Das System erzielte bei einigen Klassen gute Erkennungsgenauigkeiten, während es bei anderen deutlich schlechtere Ergebnisse lieferte.

Der Konferenzbeitrag von Hyungtae Park et al. aus dem Jahr 2024 präsentiert einen Ansatz Zur Anwendung von Deep Learning mit Spektrogrammen auf ressourcebeschränkten Geräten [33]. Das KI-Modell wurde zunächst auf einem leistungsfähigen Rechner mit geeigneten Audiodaten trainiert. Anschließend erfolgte eine Komprimierung mit Tensorflow Lite, um die Einschränkungen des Mikrocontrollers einzuhalten. Um die Konsistenz der Eingabedaten sicherzustellen, wurden die Vorverarbeitungs- und Merkmalextraktionsmethoden ebenfalls in ein TFLite-Modell überführt. Die Inferenz auf dem Mikrocontroller erfolgte mithifle von Tensorflow Lite Micro (TFLM). Die Experimente wurden mit dem STM32H747I-Discovery Board durchgeführt, das über einen Dual-Core-Prozessor verfügt

(ARM-Cortex M4 und M7). Schließlich wurden die Testdaten über eine SD-Karte bereitgestellt und die Ergebnisse auf einem LCD-Display angezeigt. Der durchnittliche Konfidenzwert für 4 Klassen lag bei 90 %.

Ein weiterer Konferenzbeitrag von Zhaolan Huang et al. aus dem Jahr 2024 befasst sich mit der Klassifizierung von Vogelgesängen [32]. Ziel ist es, mit möglichst wenig Energie und Speicherkapazität spezifische Vogelarten zu erkennen. Verschiedene Modelle wie CNN und SqueezeNet wurden mit Zeitreihendaten und Mel-Spektrogrammen getestet und evaluiert. Vor der Merkmalextraktion wurden die Audiodaten von einer Abtastrate von 48 kHz auf 16 kHz heruntergesampelt. Zur Optimierung der Modelle wurden Techniek wie Quantisierung und Partial Convolution eingesetzt. Die Tests wurden mit dem nRF52840-Mikrocontroller durchgeführt. Das Mikrofonmodell wurde jedoch nicht explizit erwähnt. Der Testdatensatz wurde aus realen Feldaufnahmen und öffentlichen Vogelstimmen-Datenbanken zusammengestellt. Die Ergebnisse zeigen, dass TinyML-Modelle bei geringem Energieverbrauch eine Genauigkeit von 99 % erzielen können.

3.3 Schlussfolgerung

Auf Basis der untersuchten Arbeiten lässt sich feststellen, dass die Mehrheit der Ansätze auf den CNNs sowie RNNs, insbesondere LSTMs-Netzen, basieren. Dies liegt daran, dass CNNs besonders gute Leistungen bei der Erfassung räumlicher Abhängigkeiten zeigen, während RNNs bei der Erfassung zeitlicher Abhängkeiten ebenfalls gute Ergebnisse liefern. Mel-Frequency Cepstral Coefficientss (MFCCs) wurden in nahezu allen Arbeiten als primäre Merkmale verwendet. Sie sind leichtgewichtig und sehr flexibel, da die Anzahl der Koeffizienten frei wählbar ist. Einige Studien haben gezeigt, dass bereits die ersten 13 Koeffizienten zur Klassifizierung von Audiosignalen ausreichend sind. In den Studien, die sich mit der Optimierung von KI-Modellen befasst haben, kam häufig Tensorflow Lite zum Einsatz. Das ist eine speziell optimierte Version des Open-Source Frameworks Tensorflow, die für den Einsatz auf ressourcebeschränkten Geräten entwickelt wurde.

Allerdings wurden die Systeme bei der Mehrheit der Studien auf begrenzten Datensätzen getestet und evaluiert, die vermutlich Szenarien aus der realen Welt nicht vollständig abbilden. Zu einer ähnlichen Schlussfolgerung kamen auch Pau Gairí et al. in ihrem Review-Artikel aus dem Jahr 2025 [40]. Darüber hinaus bleibt der Aspekt des Datenschutzes unberücksichtigt.

In dieser Arbeit wird ein ähnliches System zur Klassifizierung von Fahrzeuggeräuschen entwickelt, das sich an bestehenden Ansätze orientiert. Der Fokus liegt dabei auf der Umsetzung auf einem ressourcebeschränkten System. Dabei werden insbesondere die Herausforderungen adressiert, die in den untersuchten Arbeiten vernachlässigt wurden.

4 Konzeption und Planung

In diesem Kapitel wird mithilfe des erworbenen Wissens aus den vorherigen Kapiteln ein Konzept zur akustischen Klassifizierung von Fahrzeugen entwickelt. Ziel dieses Kapitels ist es, ein System zu entwerfen, dass zuverlässig Fahrzeuggeräusche in Echtzeit aufnimmt, klassifiziert und zählt. Die Klassen hierbei sind PKW, LKW, Bus und Motorrad. Das System soll außerdem möglichst kompakt sein, um für den Straßeneinsatz geeignet zu sein und so wenig personenbezogene Daten wie möglich verarbeiten.

4.1 Vorgehen

Die Entwicklung des Systems erfordert einen klar definierten und systematischen Umsetzungsplan. Abbildung 10 stellt die Schritte vor, die zur Realisierung dieses Systems nach dem Prinzip der agilen Softwareentwicklung durchgeführt werden.

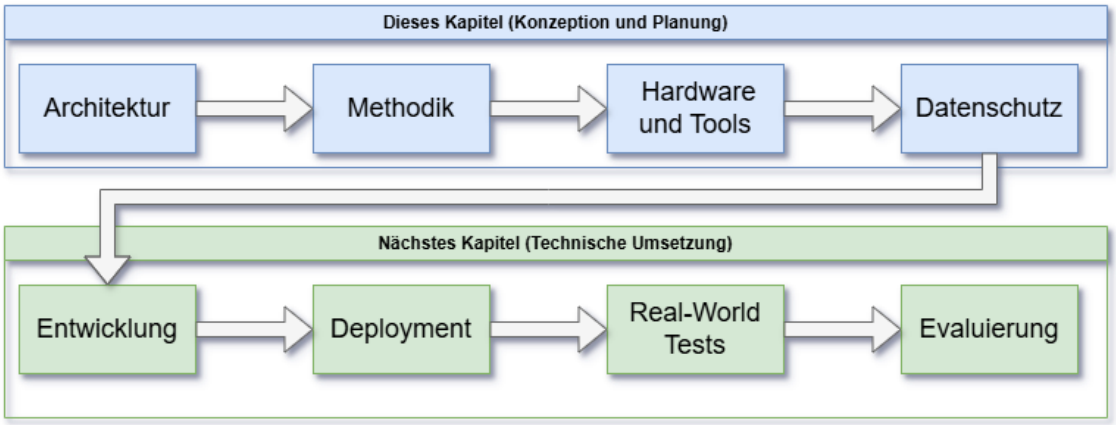

Abbildung 10: Umsetzungsplan zur Reaslisierung des Systems

Der erste Schritt besteht darin, sich für eine Architektur des Gesamtsystems zu entscheiden. Zur Auswahl stehen zwei Optionen zur Verfügung: Cloud-basierte und Edge-basierte. Bei der Cloud-basierten Architektur findet die Verarbeitung und Analayse der Daten hauptsächlich auf einem zentralen (Cloud)-Server statt. Hingegen wird bei einer Edge-basierten Architektur die vollständige Verarbeitung und Analyse direkt auf dem Edge-Device durchgeführt (Abbildung 11).

Basierend auf dieser Architektur wird die Methodik vorgestellt. Dabei werden die wissenschaftlichen Grundlagen des Vorgehens erläutert sowie die Gründe für die Auswahl bestimmter Methoden dargelegt.

Anschließend werden die Hardware und Tools vorgestellt. Diese werden so ausgewählt, dass sie bestmöglich zum Ziel dieser Arbeit passen.

Die Entwicklungs- und Deploymentphase bilden den Programmierteil. Der Fokus hierbei liegt auf der konkreten Implementierung der Software und darauf, wie sie auf dem eingebetteten System zum Laufen gebracht wird.

Schließlich soll das System im realen Umfeld getestet und anhand spezifischer Metriken evaluiert und bewertet werden.

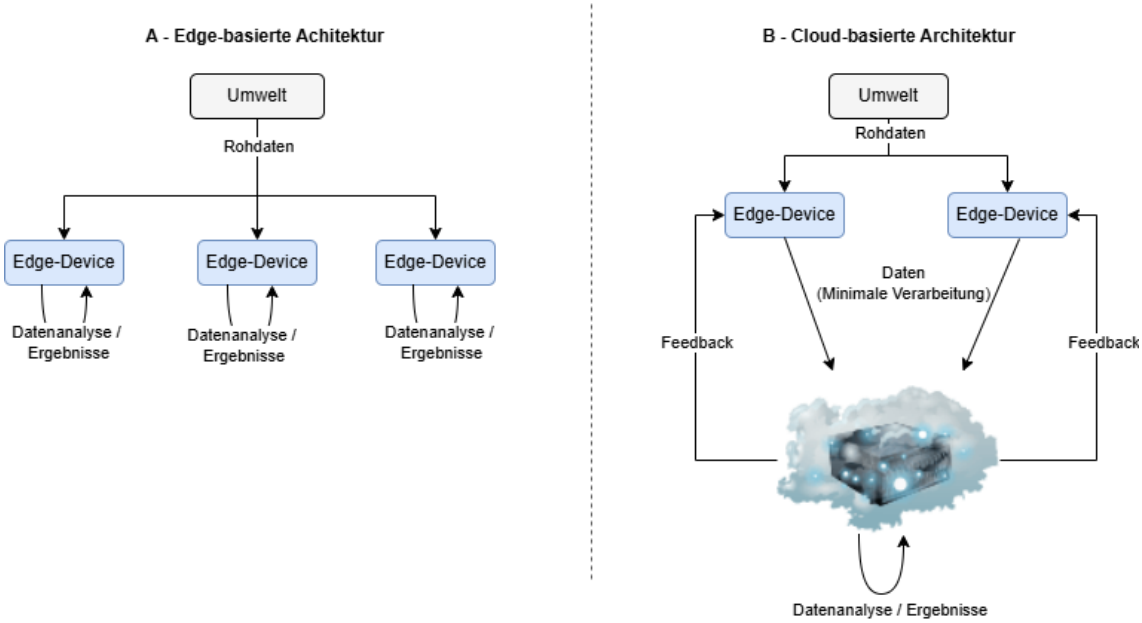

Abbildung 11: Edge- vs Cloud-basierte Architektur

In dieser Arbeit werden Edge-Devices und eingebettete Systeme gleichbehandelt, auch wenn sie technisch nicht vollständig identisch sind.

4.2 Architektur des Zielsystems und Datenfluss

Das System interagiert mit seiner Umgebung mittels eines Lärmsensors, der an einen MCU- oder MPU-basierten Entwicklungsboard angeschlossen ist (siehe Kapitel 2.2.1). Der Lärmsensor dient dazu, Schallwellen zu erfassen, die hauptsächlich durch Motoren- und Reifengeräuschen von PKWs, LKWs, Bussen und Motorrädern entstehen. Die Schallwellen lösen Schwingungen der Membran des Mikrofons aus, wodurch Spannungen in Form eines elektrischen kontinuierlichen Signal produziert wird [46]. Die Signale werden anschließend mittels eines Analog-Digital Converters (ADC) digitalisiert (siehe Kapitel 2.3.1). Zu Beginn war noch offen, ob ein analoger oder ein digitaler Lärmsensor zum Einsatz kommen würde. Dies hat den Einfluss darauf, an welcher Stelle die Digitalisierung des Signals im System stattfindet, und ob zusätzlich ein externer ADC erforderlich ist.

Nach der Digitalisierung werden die gewonnenen Signale durch die Verarbeitungsmodule analysiert. Ziel dieser Module ist es, mit minimalen Ressourcen effizient und möglichst genaue Vorhersagen zu treffen. Die Signale werden zunächst vorverarbeitet, um eine einheitliche Eingabe sicherzustellen. Die Inferenz Module, in denen das KI-Modell implementiert ist, analysieren die Eingabe und liefern eine entsprechende Vorhersage zurück. In dieser Arbeit werden die Inferenzmodule mithilfe von TinyML optimiert und komprimiert, damit sie lokal und effizient auf einem ressourcebeschränkten Gerät laufen. Ein zentraler Vorteil von TinyML ist die Reduzierung der Kosten. Da die Daten direkt auf dem Gerät verarbeitet werden, sind zusätzliche LPWAN-Module nicht notwendig. Darüber hinaus führt dies dazu, die Latenzzeiten zu verbessern und die Privatsphäre zu schützen, da keine sensiblen Daten an externe Server übertragen werden müssen. Es folgt zudem daraus, dass keine Implementierung von Kommunkationsprotokollen und Sicherheitsmaßnahmen für die Übertragung der Daten notwendig ist.

Schließlich werden die Ergebnisse und Metadaten wie Timestamps anhand der Ausgabemodule in einer Datenbank gespeichert, typischerweise auf einem externen Speicher, und über Visualisierungskomponenten (Display, LEDs, ...) veranschaulicht. Optional können die Daten in Form von Diagrammen dargestellt werden.

Um den Betrieb des Systems zu gewährleisten, ist eine Stromversorgung notwendig.

Die folgende Abbildung stellt die Architektur des Zielsystems dar sowie den Datenfluss außerhalb und innerhalb des Systems:

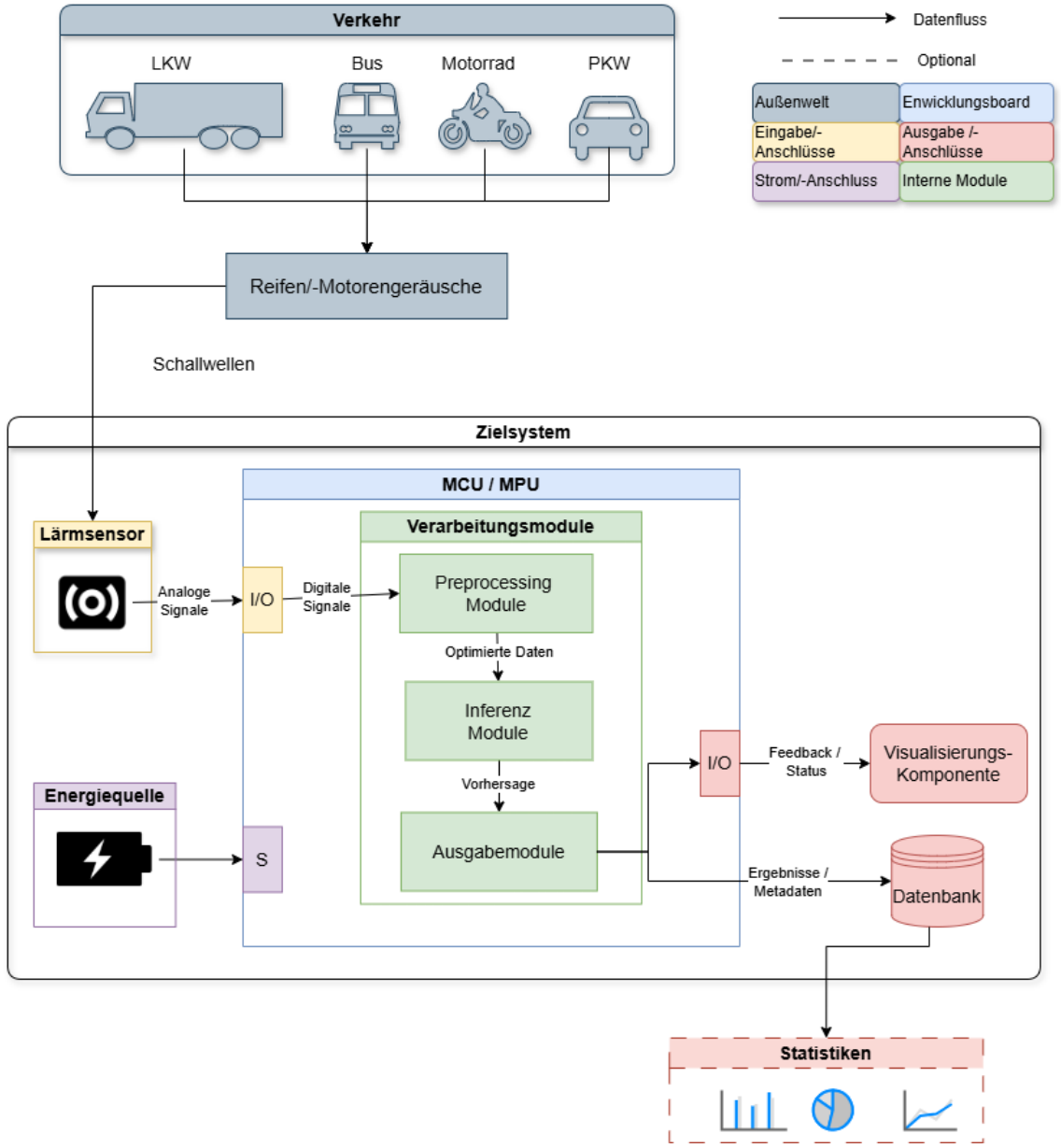

Abbildung 12: Architektur des Zielsystems und Datenfluss

4.3 Methodik

Dieses Unterkapitel erläutert verschiedene Methoden zur Erstellung eines leistungsfähigen und zuverlässigen KI-Modells. Die untenstehende Abbildung zeigt die dazugehörige Umsetzungspipeline.

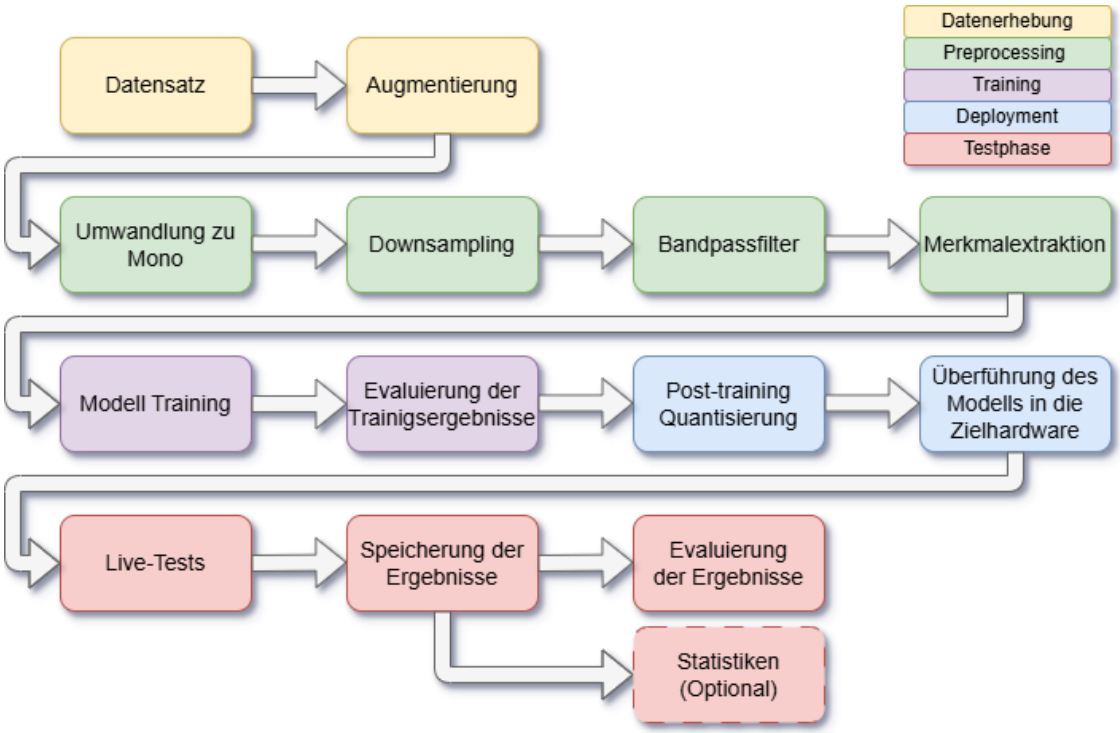

Abbildung 13: Umsetzungspipeline

4.3.1 Datenerhebung

Eine der größten Herausforderungen in KI-Projekten stellt die Datenerhebung dar. Dabei werden über einen langeren Zeitraum hochwertige Daten gesammelt und annotiert. Für die Realisierung des Ziels dieser Arbeit werden ausreichende Aufnahmen von verschiedenen Fahrzeugentypen benötigt, um ein KI-Modell zur akustischen Klassifizierung zu trainieren.

Die verfügbaren annotierten Datensätze im Internet reduzieren den Aufwand für die Datenerhebung und ermöglichen die Fokussierung auf die technische Aufgaben. Aus diesem Grund wird das KI-Modell in dieser Ausarbeitung auf Grundlage des Open-Source IDMT-Datensatzes trainiert [10]. Der Datensatz umfasst 17,506 zwei Sekunden lang Aufnahmen von PKW-, LKW-, Motorrad-, Bus- und Hintergrundgeräuschen, die sowhol mit hochwertigen als auch mit mittelwertigen Mikrofonen und einer Sampling-Rate von 48 kHz aufgezeichnet wurden.

Fahrzeutyp	**Probenanzahl**
PKW	7804
LKW	1022
Bus	106
Motorrad	430
Hintergrundgeräusche	8144
	17,506

Tabelle 3: IDMT-Datensatz - Klassenverteilung

Wie Tabelle 3 zeigt, ist die Klassenverteilung im IDMT-Datensatz nicht gleichmäßig. Mithilfe verschiedener Datenaugmentierungstechniken soll eine möglichst ausgeglichene Verteilung der Klassen erreicht werden.

Zum Testen und Evaluieren des Modells werden Audiodaten in Echtzeit aufgezeichnet und klassifiziert.

4.3.2 Preprocessing

Um eine konsistente Eingabe beim Training und Testen sicherzustellen, sollen die Rohdaten zunächst vorgearbeitet werden.

Wie in der Umsetzungspipeline dargestellt (Abbildung 13), erfolgt zunächst eine Umwandlung von Stereo- zu Mono-Audio. Dies erhöht die Kompatibilität mit verschiedenen Mikrofontypen und reduziert gleichzeitig den Speicherbedarf, da nur ein Kanal statt zwei gespeichert wird.

In der Klassifizierung von akustischen Signalen ist in der Regel eine niedrige Abtastrate ausreichend (siehe Kapitel Stand der Forschung und technischer Hintergrund). In dieser Arbeit werden sowohl die Trainingsproben als auch die Echtzeitaufnahmen auf 16 kHz heruntergesamplt, um den Speicherplatz, Energie und Rechenressources effizienter zu nutzen.

Bei einer Distanz von 10 Metern erstreckt sich das Frequenzspektrum des Straßenverkehrslärms von etwa 10 Hz bis 8 kHz, mit einem Peak im tieffrequenten Bereich bei 50–100 Hz, verursacht durch Motorgeräusche, sowie einem weiteren Peak im mittleren Frequenzbereich bei etwa 1 kHz, verursacht von Reifengeräuschen [49]. Um Störgeräusche zu reduzieren und den Fokus auf relevante Signalmerkmale zu richten, wird ein Bandpassfilter im Bereich von 20 Hz bis 5 kHz eingesetzt.

Der letzte Schritt der Vorverarbeitung ist die Merkmalextraktion. Wie im vorherigen Kapitel gezeigt wurde, sind die MFCCs flexible und leichtgewichtige Merkmale. Es werden lediglich die ersten 13 Koeffizienten extrahiert, da in mehreren Studien nachgewiesen wurde, dass diese für die Klassifikation akustischer Signale ausreichend sind.

4.3.3 Modelltraining und Deployment

In dieser Arbeit wird ein einfaches CNN-Modell eingesetzt. Das Modell besteht hauptsächlich aus drei 2D-CNN-Schichten, die für das Lernen der Merkmale zuständig sind, und drei Dense-Schichten, die das Mapping der Merkmale zu den Klassen übernehmen. Zudem erfolgt ein Vergleich der Leistungsfähigkeit des Modells mit zwei weiteren Modellarchitekturen.

Der Datensatz wird in drei Mengen aufgeteilt: 70 % Training, 15 % Validierung und 15 % Test. Die Trennung ermöglicht es, eine erste objektive Aussage über die Generalisierungsfähigkeit des Modells zu treffen.

Nach jeder Faltungsschicht (engl. *convolution layer*) wird eine Batch-Normalisierung eingesetzt. Diese Technik verbessert die Trainingsgeschwindigkeit und Anpassung der Gewichte während der Backpropagation [43][47].

Zur Reduktion von Overfitting werden während des Trainings zufällig Neuronen deaktiviert. Diese Technik wird Dropout genannt und hilft dabei, redundante Merkmale zu vermeiden. Darüber hinaus wird nach einer bestimmten Anzahl an Epochen ein Early

Stopping erzwungen. Dieser beendet das Training automatisch, sobald sich die Validierungsleistung nicht mehr verbessert [43].

Das Modell soll anschließend mit einer Konfusionsmatrix evaluiert und mit Arbeiten, die den gleichen Datensatz verwendet haben, verglichen werden.

Um Rechenanforderungen zu verringern erfolgt im letzten Schritt eine Quantisierung des Modells. Dies bedeutet, dass die Genauigkeit des ursprünglichen Modells von 32 Bit floating-points auf 8 Bit Integer Werte reduziert wird, ohne dabei die Leistung zu beeinträchtigen [20]. Dieses Modell soll ebenfalls getestet und mit dem ursprünglichen Modell verglichen werden. Schließlich wird das quantisierte Modell in die Zielhardware überführt.

4.3.4 Teststrategie und Evaluierungsmetriken

Um das finale System objektiv evaluieren zu können, wird eine geeignete Teststrategie entwickelt. Die Tests finden vorzugsweise auf mittel- bis wenigbefahrenen Straßen unter möglichst optimalen Wetterbedingungen statt (z.B. mäßige Temperaturen, geringer Wind, kein Regen). Die Teststrategie ist im folgenden Aktivitätsdiagramm dargestellt:

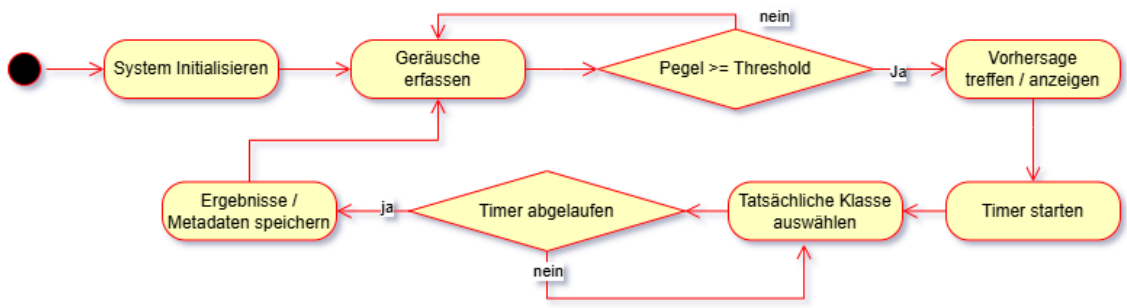

Abbildung 14: Aktivitätsdiagramm - Teststragie

Das System soll in Echtzeit Straßengeräusche oberhalb eines definierten Lautstärkepegels erfassen, eine Klassifizierung vornehmen und das Ergebnis auf einem Display anzeigen. Der Benutzer kann anschließend über eine Taste die tatsächliche Klasse bestätigen. Nach Ablauf eines Timers speichert das System die vorhergesagte Klasse, die vom Benutzer angegebene Klasse sowie relevante Metadaten (z.B. Zeitstempel, Lautstärke) in einer Datenbank.

Abschließend werden die aufgezeichneten Daten hinsichtlich Gesamtgenauigkeit, Precision, Recall und F1-Score ausgewertet. Zusätzlich werden der Stromverbrauch, die Vorverarbeitungs- und Inferenzzeiten des Systems analysiert, um auch die Effizienz der Umsetzung zu bewerten.

4.4 Auswahl von Hardware und Tools

4.4.1 Hardware

Für die akustische Fahrzeugklassifizierung ist die Auswahl passender Hardware entscheidend, insbesondere im Hinblick auf Audioverarbeitung, Rechenleistung und Energieeffizienz. In diesem Abschnitt werden der Raspberry Pi Zero 2 W und der ESP32-C3 hin-

sichtlich ihrer Eignung für diesen Anwendungsfall verglichen. Ziel ist es, ihre Stärken und Schwächen für den Einsatz in Edge-basierten Klassifikationssystemen zu analysieren.

Die betrachteten Entwicklungsboards und Vergleichskriterien sind in der folgenden Tabelle zusammegefasst:

Kriterium	**ESP32-C3**	**Pi Zero 2 W**
Preis	ca. 5-10 €	ca. 15-20 €
Prozessor	RISC-V Single-Core @160 MHz	Quad-Core Arm Cortex-A53 @1 GHz
RAM	400 KB SRAM + bis 512 KB PSRAM extern	512 MB LPDDR2
Speicher	4MB, erweiterbar	microSD-Karte
Audiosupport	I2S	I2S / USB
Konnektivität	WLAN / Bluetooth	WLAN / Bluetooth
Stromverbrauch	niedrig	höher
Größe	ca. 27 x 18 mm	65 x 30 mm
Programmiersprachen	C, C++, MicroPython, Arduino IDE	Alle
ML-Support	TensorFlow Lite Micro	TensorFlow Lite

Tabelle 4: Vergleich von ESP32-C3 und Raspberry Pi Zero 2 W

Der ESP32-C3 ist deutlich günstiger und zeichnet sich durch einen sehr niedrigen Energieverbrauch aus, was ihn ideal für stromsparende Anwendungen macht. Im Gegensatz dazu bietet der Raspberry Pi Zero 2 W mehr Rechenleistung, was komplexere Berechnungen (z.B. Merkmalextraktion) ermöglicht. Außerdem unterstützt der Pi Zero 2 W eine größere Vielfalt an Programmiersprachen. Die Anbindung von Mikrofonen ist beim Pi Zero ebenfalls flexibler, da er USB- und I2S-Audioeingänge unterstützt. Insgesamt eignet sich der ESP32-C3 für minimalistische, energieeffiziente Projekte, während der Pi Zero 2 W für anspruchsvollere und vielseitigere Anwendungen besser geeignet ist. Die weiteren Entscheidungen und Implementierungen dieser Arbeit basieren daher auf dem Raspberry Pi Zero 2 W.

Als Mikrofon wurde Respeaker 2 mics hat ausgewählt, das auf Basis des stromsparenden Stereo-codec WM8960 entwickelt wurde. Zusätzlich zu den 2 Mikrofonen befinden sich auf dem Board eine Taste und drei programmierbare RGB-LEDs. Es verfügt außerdem über eine I2C Grove Schnittstelle, die die Anbindung eines OLED-Displays vereinfacht.

Für die Tests wird das System mit einer 5000 mAh Powerbank betrieben. Im Dauerbetrieb kann die Stromversorgung durch ein Netzteil oder eine andere geeignete Spannungsquelle ersetzt werden.

Die Kosten der verwendeten Komponenten sind in der folgenden Tabelle zusammengefasst. Die Preisangaben basieren auf dem Online-Shop Berrybase [48] und dienen zur Orientierung.

Komponente	**Preis (€)**
Pi Zero 2 W (mit GPIO Header)	20,90
Respeaker 2 mics hat	14,20
Speicherkarte 32 GB	4,60
OLED-Display + Grove Adapter	8,50
Powerbank 5000 mAh	10
	≈ 60

Tabelle 5: Kosten der Hardware

Für die Einrichtung und Programmierung des Pi Zero können zusätzliche Kosten entstehen (z.B. Mini-HDMI-Adapter, Micro-USB-Hub etc.).

4.4.2 Software Tools

Laut Github sind Python und C++ zwei der beliebsten Programmiersprachen für Maschinelles Lernen [44]. Python zeichnet sich durch seine einfache Syntax aus, die das schnelle Entwickeln und Testen von Programmen ermöglicht. Außerdem ist Python im Vergleich zu C++ leichter zu lernen. C++ bietet hingegen mehr Kontrolle über die Hardware und zeichnet sich durch höhere Ausführungsgeschwindigkeit aus. Ein weiterer Vorteil von Python sind die zahlreichen Bibliotheken für Datenverarbeitung und Audioanalyse (Librosa, pandas, numpy, scipy etc.), die diese Sprache für diesen Anwendungsfall ideal macht.

Das Erstellen und Trainieren des Modells erfolgt mithilfe der Open-Source-Bibliothek Keras, die eine benutzerfreundliche High-Level-API zur Entwicklung neuronaler Netze bereitstellt. Für die Optimierung und Quantisierung des Modells wird TensorFlow Lite eingesetzt. Zusätzlich kommt die Bibliothek scikit-learn zum Einsatz, die durch zahlreiche integrierte Funktionen eine effiziente und unkomplizierte Evaluierung des Modells ermöglicht.

4.5 Einfluss der Entscheidungen auf Datenschutz

Wie in den vorherigen Kapiteln erwähnt, unterliegt diese Arbeit den DSGVO-Regelungen, da Tonaufzeichnungen in öffentlichen Bereichen personenbezogene Daten (Stimmen von Personen) enthalten können. In diesem Abschnitt werden die in Tabelle 1 aufgeführten Datenschutzaspekte näher betrachtet:

- **Zweck und Rechtmäßigkeit der Verarbeitung:** Der Hauptzweck der Verarbeitung besteht in der Klassifizierung von Fahrzeugen anhand akustischer Signale. Dabei können unbeabsichtigt auch personenbezogene Daten erfasst werden, etwa als Hintergrundgeräusche. Die Verarbeitung erfolgt ausschließlich zu Forschungszwecken.
- **Datenminimierung:** Eine Aufnahme erfolgt nur dann, wenn ein definierter Lautstärkepegel überschritten wird. Dadurch wird die Datenverarbeitung auf relevante akustische Ereignisse beschränkt und stark reduziert.
- **Speicherbegrenzung und Speicherdauer:** Die Extraktion der MFCCs erfolgt direkt im Arbeitsspeicher (RAM). Die Audiodaten bleiben dort lediglich bis zur

Durchführung der Inferenz. Ein konkreter Zeitraum kann derzeit nicht exakt festgelegt werden, sollte jedoch fünf Sekunden nicht überschreiten. Nach der Verarbeitung werden die Daten unverzüglich und unwiederbringlich gelöscht.

- **Integrität, Vertraulichkeit und Sicherheit der Verarbeitung:** Durch die Edge-basierte Systemarchitektur erfolgt die gesamte Verarbeitung lokal auf dem Gerät. Ein Zugriff auf Rohdaten durch Dritte ist ausgeschlossen.
- **Einwilligung betroffener Personen:** In öffentlichen Räumen ist es in der Regel nicht möglich, die Einwilligung aller betroffenen Personen einzuholen. Zuständige Behörden können jedoch mit Schildern auf Tonaufzeichnungen aufmerskam machen.

Damit erfüllt das System die wesentlichen Anforderungen der DSGVO.

5 Technische Umsetzung

In diesem Kapitel werden die im vorherigen Abschnitt beschriebenen Methoden konkret implementiert. Der Fokus liegt auf der Entwicklung einer Python-Pipeline, die sowohl auf High-End-Rechnern als auch auf dem Raspberry Pi Zero 2 möglichst kompatibel ist. Zudem wird darauf geachtet, möglichst wenig Zufälligkeit einzuführen, um die Reproduzierbarkeit der Ergebnisse sicherzustellen. Um Speicherplatz auf dem Pi Zero 2 zu sparen und zu vermeiden, dass unnötige Module installiert werden müssen, wurden zwei separate Git-Repositories angelegt: eines für das Training und die Quantisierung des Modells, und eines für die Inferenz auf dem Pi. Dadurch bleibt das Pi-Repository klein und schlank, während das Trainings-Repository alle notwendigen Tools für Entwicklung und Experimente enthält. Der vollständige Quellcode ist verfügbar unter
`https://gitlab.accso.de/khemissi/acoustic-vehicle-classification` und
`https://gitlab.accso.de/khemissi/acoustic-vehicle-classification-pi-zero`

5.1 Datenaugmentierung

Audio Augmentation ist eine Methode zur künstlichen Erweiterung eines Audiodatensatzes, um die Robustheit und Generalisierungsfähigkeit von Machine-Learning-Modellen zu verbessern [19][24]. Besonders im Bereich der akustischen Klassifizierung erweist sich diese Technik als hilfreich, um Modelle auf realistische Szenarien vorzubereiten, in denen Aufnahmen beispielsweise verrauscht, verzerrt oder in unterschiedlicher Lautstärke vorliegen können.

Zu den angewendeten Verfahren zählt unter anderem das Hinzufügen von Hintergrundgeräuschen, wie etwa Stimmen oder Wind. Diese Geräusche wurden mit dem Zielmikrofon (ReSpeaker 2 Mics hat) aufgenommen und anschließend auf Trainingsdaten angewendet, um realitätsnahe Störungen zu simulieren. Ein weiteres Verfahren ist das sogenannte Time Stretching, bei dem die Geschwindigkeit eines Audiosignals verändert wird, ohne die Tonhöhe zu beeinflussen. Dadurch lassen sich unterschiedliche Abspielgeschwindigkeiten oder leichte zeitliche Verzerrungen, etwa durch Bewegung, nachbilden. Zudem wird durch Pitch Shifting die Tonhöhe variiert, ohne die Dauer des Audios zu verändern. Dies hilft dabei, natürliche Schwankungen in den Fahrzeuggeräuschen realitätsnah abzubilden. Ergänzend kommt eine Verstärkungsanpassung (Gain) zum Einsatz, bei der die Lautstärke des Signals verändert wird, um Unterschiede in der Mikrofonposition oder der Entfernung zur Schallquelle nachzustellen.

Um diese Annahme überprüfen zu können, wurden 20% der ursprünglichen, nicht augmentierten Daten (Tabelle 3) mithilfe der Funktion *train_test_split* der Bibliothek *scikit-learn* [55] herausgenommen. Diese Teilmenge wurde bewusst nicht verändert, um eine saubere Referenzbasis zu schaffen und die Qualität der Trainingsdaten sicherzustellen. Dadurch lässt sich besser beurteilen, welchen Einfluss die Augmentation auf das Modellverhalten hat. Der Prozess wird in Abbildung 15 visualisiert.

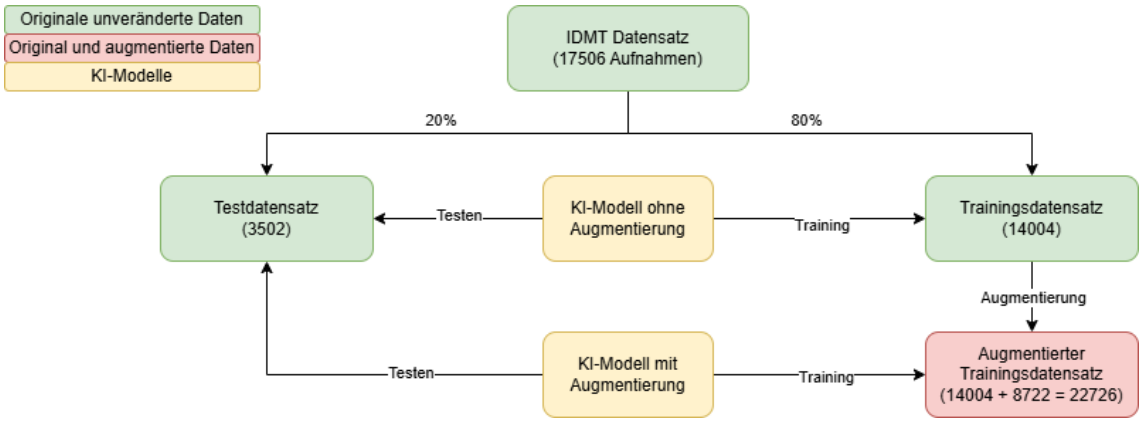

Abbildung 15: Visualisierung des Training/Test-Aufteilungsprozesses

Anschließend wurden die unterrepräsentierten Klassen (Bus, LKW und Motorrad) jeweils siebenfach mithilfe der Bibliothek *audiomentations* [51] augmentiert. Dabei kamen die Techniken sowohl einzeln als auch in Kombination zum Einsatz. Tabelle 6 stellt die Ergebnisse der Augmentierung dar.

Fahrzeugtyp	**Training(Original)**	**Augmentiert**	**Training (Gesamt)**	**Test**
PKW	6243	0	6243	1561
LKW	817	5719 (×7)	6536	205
Bus	85	595 (×7)	680	21
Motorrad	344	2408 (×7)	2.752	86
Kein Verkehr	6515	0	6515	1629
	14004	8722	22726	3502

Tabelle 6: Klassenverteilung der Daten vor und nach der Augmentierung

5.2 Vorverarbeitung und Merkmalextraktion

Die Audiodaten werden zunächst mit der Bibliothek *scipy* in ein einheitliches Format überführt, indem alle Signale auf Mono konvertiert und auf eine Abtastrate von 16 kHz heruntergerechnet werden. Dieser Schritt stellt sicher, dass alle Eingangsdaten konsistent verarbeitet werden können.

Anschließend erfolgt eine Bandpassfilterung des Audiosignals im Frequenzbereich von 20 Hz bis 5000 Hz. Hierfür wird ein Bandpassfilter ebenfalls mit *scipy* implementiert, um unerwünschte Frequenzanteile außerhalb dieses Bereichs zu entfernen. Dies dient der Rauschunterdrückung und der Fokussierung auf relevante Fahrzeuggeräusche.

Die primäre Merkmale für die Klassifikation in dieser Arbeit sind die MFCCs. Dabei werden pro Zeitfenster 13 Koeffizienten extrahiert, die das akustische Signal kompakt beschreiben. Diese Fenster sind jeweils 25 Millisekunden lang und werden alle 10 Millisekunden über das Signal geschoben, sodass man eine zeitlich aufgelöste Beschreibung erhält. Die Anzahl der resultierenden Zeitfenster lässt sich mit folgender Formel berechnen:

$$N = \left\lfloor \frac{2{,}0 - 0{,}025}{0{,}010} \right\rfloor + 1 = \boxed{198}$$

Um sicherzustellen, dass alle Aufnahmen eine einheitliche Anzahl an Zeitfenstern aufweisen, wurde die Anzahl auf 180 Zeitfenster reduziert. Damit es bei der Frequenzanalyse keine abrupten Übergänge am Fensterrand gibt, wird jedes Zeitfenster noch mit einer Hanning-Funktion geglättet. Die komplette Berechnung erfolgte mit der Bibliothek *python_speech_features* [54].

Zur Umwandlung in das Frequenzspektrum werden 128 Mel-Filterbänke verwendet. Die resultierenden Daten werden anschließend mithilfe einer logarithmischen Skalierung und einer diskreten Cosinus-Transformation auf 13 Hauptkoeffizienten reduziert. Am Ende ergibt sich daraus eine Merkmalsmatrix der Form (13×180), wobei 13 die Anzahl der Koeffizienten und 180 die Anzahl der Zeitfenster darstellt.

Um die Wiederverwendbarkeit beim Trainieren des Modells zu erleichtern, wurde die gesamte Pipeline sowohl auf den Trainings- als auch auf den Testdatensatz angewendet und die extrahierten Merkmale anschließend mithilfe von *numpy* als komprimierte NPZ-Dateien gespeichert.

5.3 Training und Quantisierung des Modells

Im Rahmen des Trainings wurden drei verschiedene Modellarchitekturen miteinander verglichen: ein eindimensionales CNN-Modell, das auf dem Ansatz von Mohd Ashhad et al. [19] basiert, ein zweidimensionales CNN-Modell, das aus dem 1D-CNN inspiriert wurde, sowie ein hybrides CNN-LSTM-Modell (Abbildung 16), ähnlich zu dem in der Arbeit von L. Yuxi et al. [13] vorgestellten Modell.

Ziel des Vergleichs war es, die Modelle hinsichtlich ihrer Genauigkeit, Modellgröße, Trainingsdauer und Inferenzzeit zu bewerten. Die Inferenzzeit wurde berechnet, indem die extrahierten Merkmale der Testdaten **einzeln** dem Modell übergeben und die durchschnittliche Latenzzeit pro Durchlauf ermittelt wurden. Die präsentierten Ergebnisse wurden auf einem Rechner mit einem Intel Core i7 Prozessor (Modell i7-1165G7, 3,6 GHz) erzielt. Bei Verwendung anderer Hardware können die Trainings- und Inferenzzeiten sowie die Performance variieren.

Alle Modelle wurden ausschließlich mit dem Original-Datensatz (ohne Augmentierung) trainiert und anschließend auf demselben Testdatensatz evaluiert, um faire Vergleichsbedingungen zu gewährleisten. Zudem wurden sie über maximal 50 Epochen mit einem Early-Stopping-Mechanismus trainiert: Das Training wurde automatisch abgebrochen, wenn sich die Validierungsleistung über zehn aufeinanderfolgende Epochen um weniger als 0,1% verbesserte. Die Implementierung und Evaluierung der Modelle erfolgten mit *Keras* [52] und *scikit-learn*.

Das 2D-CNN erzielte nach 32 Trainings-Epochen die höchste Genauigkeit von 93,38 % und schnelleste Inferenzzeit (45 ms), benötigte jedoch den größten Speicherplatz. Das 1D-CNN überzeugte durch die kürzeste Trainingsdauer (4,5 Minuten), konnte allerdings die Fahrzeugklasse „LKW“ nicht erkennen und erreichte insgesamt die geringste Klassifikationsleistung. Eine längere Trainingszeit könnte jedoch diese Ergebnisse verbessern. Das CNN-LSTM-Modell war mit lediglich 1,3 MB am speichereffizientesten, wies jedoch die längste Trainings- und Inferenzzeit auf. In Bezug auf die Klassenleistungen (F1-Scores) zeigte das 2D-CNN über allen Klassen hinaus deutlich bessere Ergebnisse gegenüber den anderen Modellen. Die vollständigen Ergebnisse sind in Tabelle 7 dargestellt.

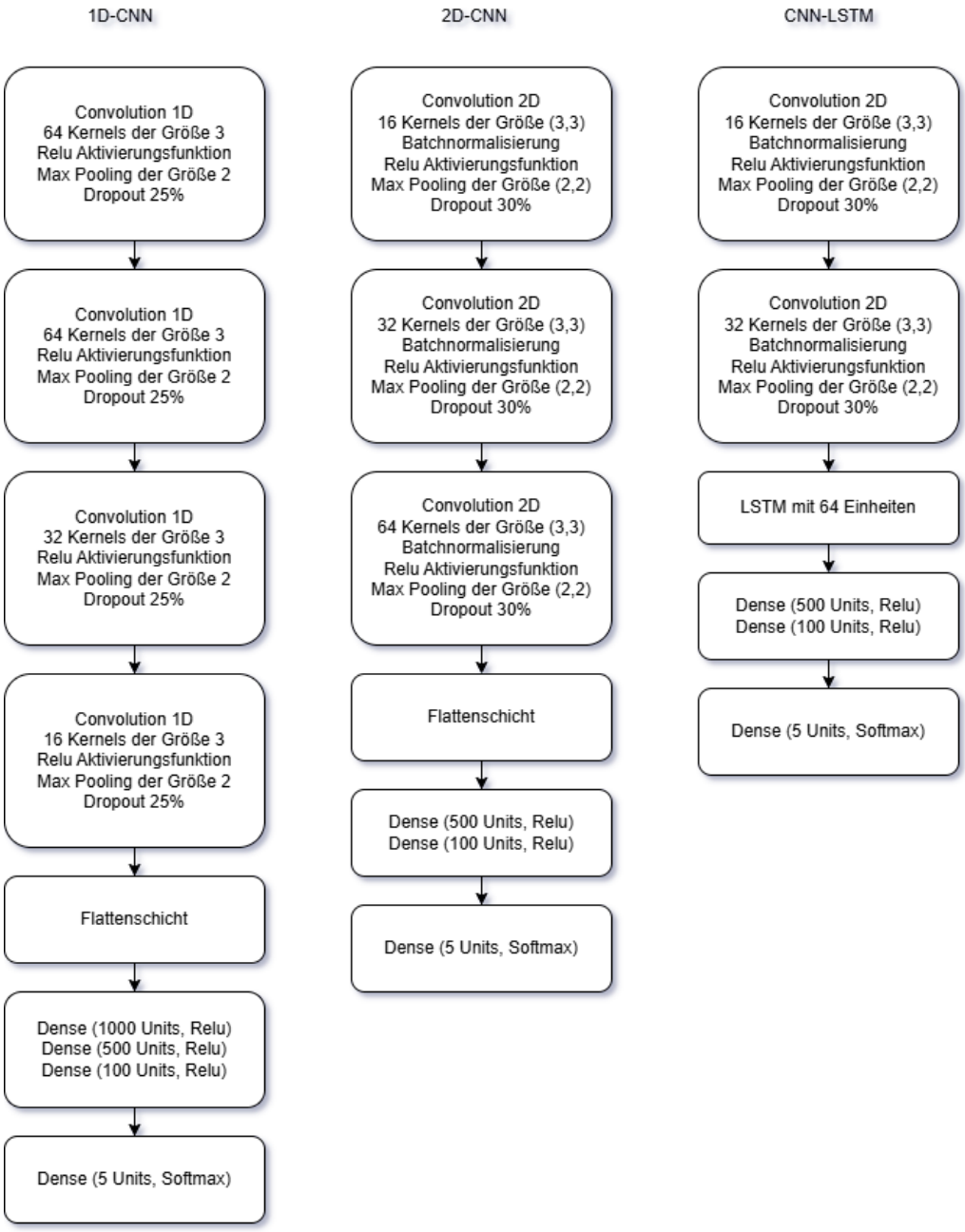

Abbildung 16: Architektur der betrachteten Modelle

Eigenschaft	**1D-CNN**	**2D-CNN**	**CNN-LSTM**
Epochen	*31*	33	45
Trainingsdauer (min)	**4,5**	10	21
Größe (MB)	8,5	9,2	**1,3**
Inferenzzeit (ms)	47	**45**	50
PKW (F1-%)	92	**93**	**93**
LKW (F1-%)	0	**39**	29
Bus (F1-%)	5	**70**	67
Motorrad (F1-%)	93	**98**	97
Kein Verkehr (F1-%)	98	**99**	**99**
Genauigkeit (%)	92,15	**93,38**	93,23

Tabelle 7: Vergleich verschiedener Modelle

Insgesamt zeigte das 2D-CNN-Modell vergleichbare Ergebnisse zu den Arbeiten [19] und [10]. Aufgrund dieser überzeugenden Leistung werden die nächsten Schritte auf Basis dieses Modells durchgeführt. Darüber hinaus wurde mit der RandomSearch-Methode von *Keras Tuner* versucht, durch automatisiertes Hyperparameter-Tuning eine leistungsfähigere Modellkonfiguration zu finden. Dieser Prozess erwies sich jedoch als sehr zeitaufwendig (ca. 4 Stunden bei 20 von rund 11.500 möglichen Kombinationen) und führte zu keiner signifikanten Leistungssteigerung.

5.3.1 Funktionsweise des ausgewählten Modells (2D-CNN)

Zu Beginn wendet das Modell eine zweidimensionale Faltungsschicht (Conv2D) mit 16 Filtern der Größe 3×3 an. Diese Filter gleiten über das MFCC-Spektrogramm und erfassen jeweils lokale Zeit-Frequenz-Muster. An jeder Position berechnet der Filter das gewichtete Skalarprodukt zwischen seinem Kern und dem entsprechenden lokalen Ausschnitt des MFCCs. So entsteht für jeden Filter eine sogenannte Merkmalskarte (Feature Map). Anschließend wird die Ausgabe mittels Batch-Normalisierung stabilisiert und durch die ReLU-Aktivierungsfunktion nichtlinear transformiert, wobei gilt:

$$\mathrm{ReLU}(x) = \max(0, x).$$

Darauf folgt eine Max-Pooling-Schicht mit einem Fenster von 2×2, welche die zeitlich-frequenzielle Auflösung reduziert und die Rechenkomplexität verringert. Eine Dropout-Schicht mit einer Rate von 30 % verhindert anschließend ein Overfitting, indem sie zufällig Verbindungen innerhalb des Netzes deaktiviert. Dieser Block wiederholt sich zweimal mit steigender Filteranzahl: Im zweiten Convolutional-Block werden 32 Filter derselben Größe verwendet, im dritten 64. Diese tieferen Schichten sind in der Lage, aus den vorher extrahierten einfachen Mustern komplexere akustische Muster zu erlernen. Nach jedem Block folgen erneut Batch-Normalisierung, ReLU, Max-Pooling und Dropout. Nach den drei Faltungsblöcken wird die resultierende dreidimensionale Ausgabe über eine Flatten-Schicht in einen Vektor umgewandelt, der die gesamte Merkmalsinformation kompakt zusammenfasst. Dieser Vektor wird anschließend durch zwei voll verbundene Schichten (Dense Layers) mit 500 bzw. 100 Neuronen weiterverarbeitet, wobei jeweils die ReLU-Aktivierung zum Einsatz kommt. Zum Schluss folgt eine Dense-Schicht mit fünf Neuronen, entsprechend der Anzahl der Zielklassen. Diese verwendet die Softmax-Funktion, um die Ausgabe in eine Wahrscheinlichkeitsverteilung über alle Klassen zu transformieren:

$$\mathrm{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \quad \text{für } i = 1, \ldots, K,$$

wobei $K = 5$ die Anzahl der Klassen ist. Die resultierende Ausgabeschicht liefert somit einen Vektor mit fünf Werten zwischen 0 und 1, die die Wahrscheinleichkeiten der einzelnen Klassen darstellen.

5.3.2 Einfluss der Datenaugmentierung

In diesem Abschnitt wird das 2D-CNN-Modell erneut mit den originalen sowie den augmentierten Daten trainiert. Die Testdaten umfassen 20 % der gesamten Originaldaten und werden in beiden Fällen während des Trainings nicht verwendet. Dadurch ist gewährleistet, dass beide Modelle auf denselben, zuvor nicht gesehenen Daten evaluiert werden, was eine genauere Aussage über den Einfluss der Datenaugmentierung erlaubt. Die folgende Konfusionsmatrix (Abbildung 17) und deren Klassifizierungsbericht (engl. *classification report*,

Tabelle 8) wurden mit *matplotlib*, *seaborn* und *scikt-learn* erstellt und beschreiben die Ergebnisse, die vor der Augmentierung erzielt wurden.

Klasse	**Precision**	**Recall**	**F1-Score**	**Proben**
Bus	0,74	0,67	0,70	21
KV	0,99	0,99	0,99	1629
LKW	0,56	0,30	0,39	205
MR	0,98	0,99	0,98	86
PKW	0,91	0,96	0,93	1561
Macro Avg	0,83	0,78	0,80	3502
Genauigkeit			0,93	3502

Tabelle 8: Klassifikationsbericht vor der Augmentierung

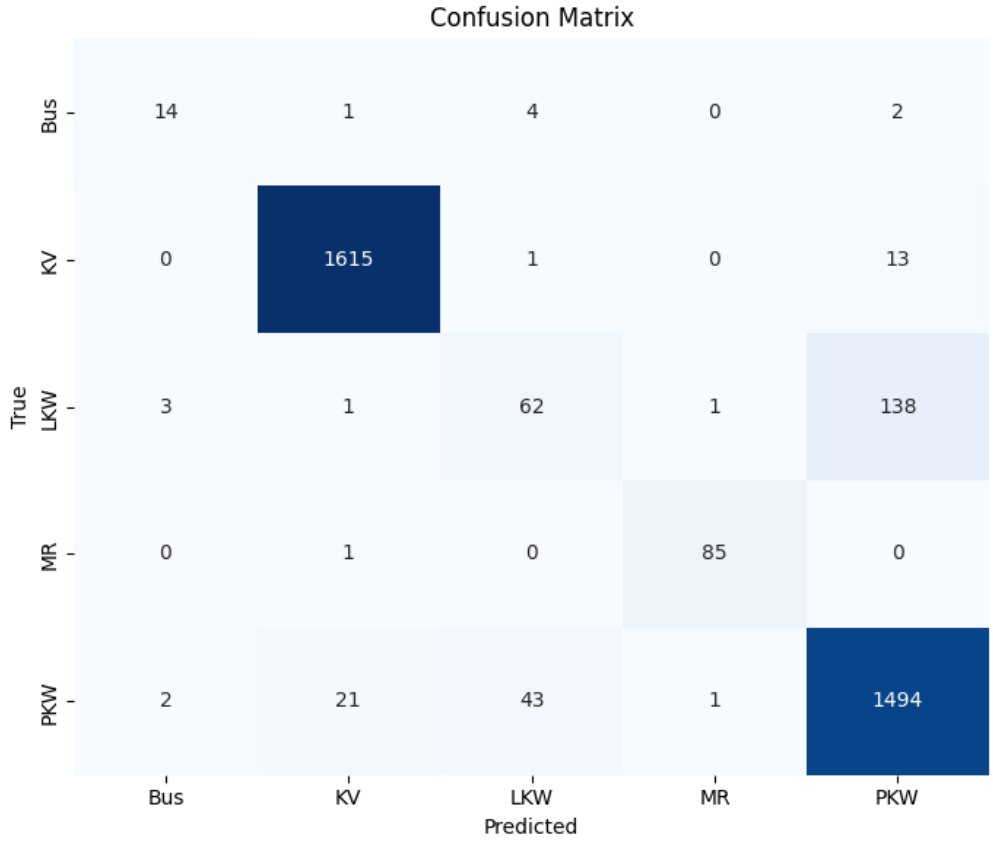

Abbildung 17: Konfusionsmatrix vor der Augmentierung

Nach der Augmentierung sehen die Ergebnisse folgendermaßen aus:

Klasse	Precision	Recall	F1-Score	Proben
Bus	0,77	0,81	0,79	21
KV	0,98	0,99	0,99	1629
LKW	0,51	0,55	0,53	205
MR	0,96	1,00	0,98	86
PKW	0,94	0,91	0,93	1561
Macro Avg	0,83	0,85	0,84	3502
Genauigkeit			0,93	3502

Tabelle 9: Klassifikationsbericht nach der Augmentierung

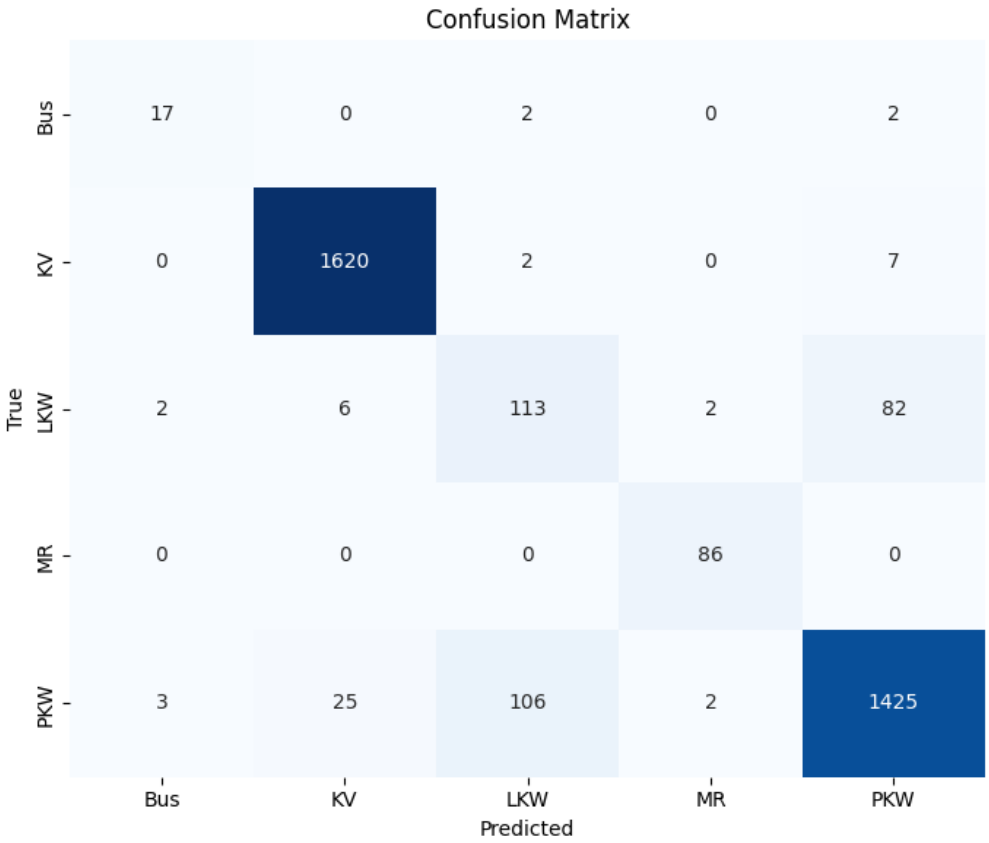

Abbildung 18: Konfusionsmatrix nach der Augmentierung

Vor der Anwendung der Datenaugmentierung zeigen Abbildung 17 und Tabelle 8 die Leistung des Modells auf den Originaldaten. Die Genauigkeit liegt bei 93 %, wobei insbesondere die Klassen KV (Kein Verkehr) und PKW mit F1-Scores von 99 bzw. 0,93 sehr gut erkannt werden. Hingegen sind die F1-Werte für die Klassen Bus und LKW mit 70 bzw. 0,39 % deutlich niedriger, was auf eine eingeschränkte Fähigkeit des Modells hinweist, diese Klassen zuverlässig zu unterscheiden.

Nach der Anwendung der Datenaugmentierung, wie in Abbildung 18 und Tabelle 9 dargestellt, zeigt sich eine deutliche Verbesserung in nahezu allen Metriken. Die Gesamtgenauigkeit bleibt mit 93 % stabil, während sich insbesondere die F1-Scores der zuvor schwächeren Klassen „Bus“ und „LKW“ deutlich erhöhen – auf 79 % bzw. 53 %. Auch die Klassen „MR“ und „PKW“ weisen weiterhin sehr hohe F1-Werte auf. Die Gesamtleistung des Systems, gemessen am makrogewichteten Durchschnitt ohne Berücksichtigung der Klassenverteilung, ist von 80 % auf 84 % gestiegen.

Die Ergebnisse zeigen jedoch auch, dass vermehrt „PKWs“ mit „LKWs“ verwechselt werden (von 43 auf 106 Fälle gestiegen). Dies lässt sich möglicherweise dadurch erklären, dass die Klasse LKW nicht eindeutig abgegrenzt ist und einige Fahrzeuge, etwa Lieferwagen wie die von DHL, trotz Zuordnung zur LKW-Klasse, ähnliche Motor und Reifengeräusche wie PKWs aufweisen.

Die bessere Leistung kommt daher, dass das Modell durch die künstliche Vergrößerung der unterrepräsentierten Klassen mehr verschiedene Varianten der Merkmale kennenlernen konnte. Insgesamt zeigt die Datenaugmentierung also einen klaren Vorteil für die Klassifikation, besonders bei Klassen, für die es weniger Trainingsdaten gibt.

5.3.3 Quantisierung des Modells und Deployment auf dem Pi Zero 2

Das trainierte Modell ist nun bereit für die Quantisierung. Dieser Schritt wurde mithilfe von *tensorflow* durchgeführt, wobei das ursprüngliche Keras-Modell in ein TFLite-Modell konvertiert wurde. Die Modellgröße konnte dabei von 9,2 MB auf 771 KB reduziert werden (eine Verringerung um den Faktor 12). Wie Tabelle 10 veranschaulicht, hat die Komprimierung die Leistung des Modells nicht beeinträchtigt. Darüber hinaus hat sich die Inferenzzeit deutlich verkürzt und liegt nun bei etwa 25 ms (vorher war 45 ms).

Klasse	**Precision**	**Recall**	**F1-Score**	**Proben**
Bus	0,81	0,81	0,81	21
KV	0,98	0,99	0,99	1629
LKW	0,50	0,56	0,53	205
MR	0,96	1,00	0,98	86
PKW	0,94	0,91	0,93	1561
Makro-Durchschnitt	0,84	0,85	0,85	3502
Genauigkeit			0,93	3502

Tabelle 10: Klassifikationsbericht des quantisierten Modells

Das Deployment des Modells und der Python-Module auf dem Pi Zero erfolgte teilweise über Git, um Versionskontrolle und einfache Updates zu gewährleisten. Für schnellere Tests wurden die Dateien manchmal auch direkt per SSH mit FileZilla übertragen, was einen schnellen und unkomplizierten Zugriff auf die Dateien ermöglichte.

Die Testergebnisse auf dem Pi Zero 2 stimmten vollständig mit den in Tabelle 10 dargestellten Resultaten überein. Überraschenderweise lag die durchschnittliche Inferenzzeit auf dem Pi bei lediglich 7 ms. Ein möglicher Grund hierfür könnten die unterschiedlichen Bibliotheken sein, die zum Laden des quantisierten Modells verwendet wurden (*tensorflow.lite* vs. *tflite-runtime*). *TFLite-runtime* ist eine leichgewichtige Bibliothek, die speziell für ressourcenbeschränkte Geräte optimiert wurde. Da diese Bibliothek jedoch auf dem Trainingsrechner nicht installiert werden konnte, ließ sich diese Annahme nicht verifizieren.

6 Testphase und Ergebnisanalyse

In diesem Kapitel wird die entwickelte Live-Teststrategie zur Evaluierung des Modells im Praxiseinsatz vorgestellt. Dabei werden zunächst die aufgetretenen Herausforderungen und Probleme bei der Umsetzung analysiert und die vorgenommenen Anpassungen beschrieben. Anschließend erfolgt eine Darstellung der Modell-Performance sowie der Leistungsfähigkeit des Raspberry Pi Zero bei der Verarbeitung realer Aufnahmen.

6.1 Probleme und Erkenntnisse bei der ursprünglichen Teststrategie

Die ursprüngliche Teststrategie basierte auf der Idee, dass bei Überschreiten eines definierten Pegels automatisch eine Aufnahme von zwei Sekunden gestartet wird. Im Anschluss sollte das Modell eine Vorhersage treffen und das Ergebnis auf dem Display anzeigen. Parallel startete ein Timer, innerhalb dessen der Nutzer mittels Knopfdruck die tatsächliche Klasse eingeben konnte. Dabei wurden ausschließlich die Vorhersageergebnisse gespeichert, während die Rohdaten der Aufnahmen nicht erfasst wurden.

Diese Herangehensweise führte jedoch zu mehreren Problemen. Zum einen zeigten die Vorhersagen des Modells häufig keine plausiblen Ergebnisse, da in vielen Fällen immer nur eine Klasse mit einer Wahrscheinlichkeit von 1.0 zurückgegeben wurde. Zum anderen erschwerte das Fehlen der Rohdaten die Fehleranalyse erheblich, da die Ergebnisse nicht reproduzierbar waren und somit keine Rückschlüsse auf mögliche Ursachen gezogen werden konnten. Als potenzielle Ursache wurde eine schlechte Mikrofonqualität vermutet. Außerdem könnte das Aufnahmetiming durch die Pegelüberwachung ungenau gewesen sein, was die Datenqualität beeinträchtigte. Ein weiterer kritischer Punkt könnte die Vorverarbeitung der Audiosignale gewesen sein: Die Extraktion der MFCC-Features während der Live-Inferenz stimmte mit der Trainingspipeline nicht überein, was zu Inkonsistenzen führte.

Zur gezielten Untersuchung der oben genannten Probleme wurde die Strategie so angepasst, dass eine 2-sekündige Audioaufnahme per Knopfdruck gestartet wird. Im Anschluss kann innerhalb von 7 Sekunden nach mehrfachem Drücken die tatsächliche Fahrzeugklasse ausgewählt werden. Nach Ablauf dieses Zeitfensters speichert das System die Aufnahme im Format TIMESTAMP_KLASSE.wav.

Im Verlauf der Untersuchung konnten folgende Ursachen identifiziert und schrittweise behoben werden

1. **Hoher Hintergrundpegel durch Mikrofon-Gain:** Die Aufnahmen enthielten sehr viele Störgeräusche, da der Mikrofon-Gain mit 51 dB zu hoch eingestellt war. Durch Anpassung des Gains mit dem Tool `Alsamixer` auf 12,5 dB hat sich die Audioqualität deutlich verbessert. Obwohl dadurch das Störgeräuschlevel sank, war das Problem der unplausiblen Modellvorhersagen noch nicht vollständig beseitigt.
2. **Unterschiedliche Bibliotheken für MFCC-Extraktion:** Während beim Training die MFCCs mit der Bibliothek `librosa` berechnet wurden, kam bei der Live-Inferenz `python_speech_features` zum Einsatz. Um die Konsistenz zu erhöhen, wurde die Trainingspipeline ebenfalls auf `python_speech_features` umgestellt. Dennoch blieb das Problem weiterhin bestehen.
3. **Abweichende Signalformate beim Laden der Audiodateien:** Ein unerwarteter weiterer Grund war, dass im Training die Audiodateien mit `librosa` geladen

wurden, welches das Signal als normalisiertes `float32` im Bereich $[-1; 1]$ zurückgibt. Die Live-Inferenz verwendete jedoch Signale im `int16`-Format. Da `librosa` auf dem Pi Zero nicht installiert werden konnte, musste das Laden und Resampling mit `scipy.io` umgesetzt werden. Diese Umstellung sorgte für eine einheitliche Datenbasis und ermöglichte es, die Ergebnisse auf dem Pi Zero mit denen auf dem Trainingsrechner besser zu vergleichen.

6.2 Evaluierung des Systems anhand isolierter Aufnahmen

Nachdem die Probleme schrittweise behoben wurden, liefert das Modell nun nachvollziehbare Vorhersagen. Es wird jedoch weiterhin auf die ursprünglich geplanten automatischen Live-Tests verzichtet. Stattdessen wird die angepasste Strategie angewandt, bei der die Vorhersagen zusammen mit der tatsächlichen Eingabe und den Rohdaten gespeichert werden. Dies führt dazu, dass gleichzeitig ein wertvoller Testdatensatz mit Aufnahmen vom Zielmikrofon entsteht, welcher für weitere Analysen und Optimierungen genutzt werden kann.

Zu diesem Zweck wurden insgesamt 442 annotierte Audiobeispiele gesammelt. Der Datensatz setzt sich zusammen aus 207 PKWs, 71 Bussen, 40 Motorrädern, 23 LKWs sowie 101 Hintergrundgeräuschen. Alle Aufnahmen wurden im WAV-Format mit einer Abtastrate von 16.000 Hz, einer Bittiefe von 16 Bit (INT16) sowie in Mono aufgezeichnet. Als Aufnahmegerät kam das ReSpeaker 2-Mic Array zum Einsatz, das gleichzeitig auch auf dem Zielsystem (Raspberry Pi Zero 2) verwendet wird (Abbildung 19). Die Mikrofonposition befand sich in einem Abstand von etwa 1 bis 12 Metern zum Verkehr, wobei die meisten Aufnahmen bei Nacht durchgeführt wurden, um Fahrzeuge möglichst isoliert und ohne Überlappung erfassen zu können. Als Aufnahmeort diente die Adresse „Hermann-Wandersleb-Ring, 53121 Bonn", eine vierspurige Straße mit hohem Busaufkommen (Abbildung 20). Die Umgebungsbedingungen waren dabei realitätsnah: Es traten mäßige bis starke Windverhältnisse, Baustellenlärm, Vogelrufe sowie Geräusche raschelnder Bäume auf.

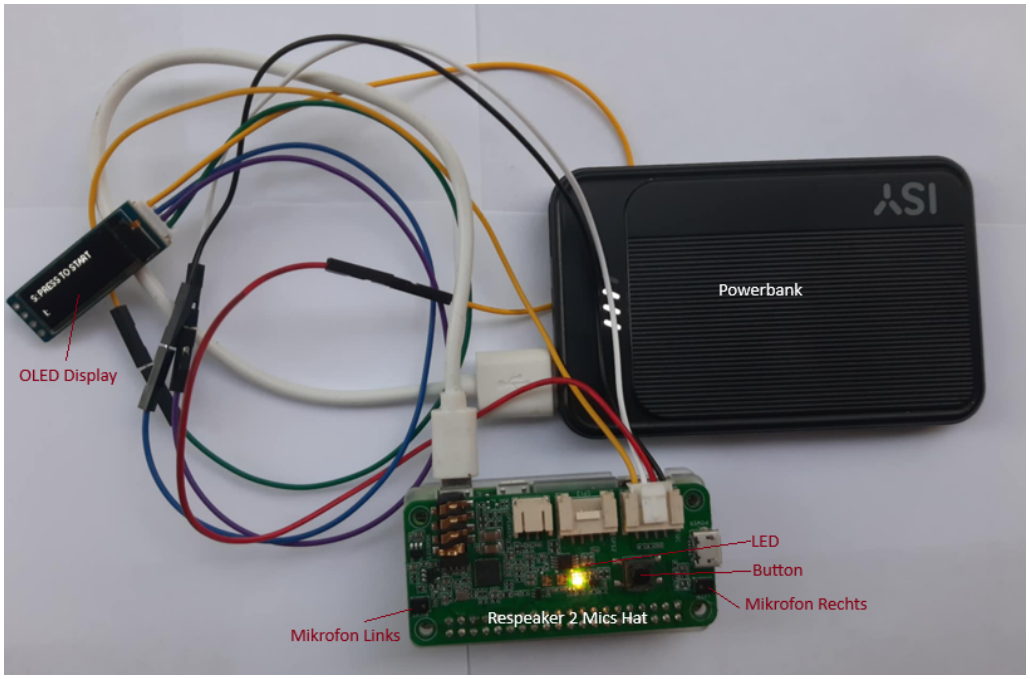

Abbildung 19: Verwendetes Setup zur Geräuschaufnahme

Zudem wurde darauf geachtet, dass keine erkennbaren Gespräche oder unbefugten Stimmen aufgezeichnet wurden. Dies stellt sicher, dass das System unter realitätsnahen Bedingungen und mit Einhaltung der Datenschutzvorgaben evaluiert werden kann.

Abbildung 20: Straßenansicht des Aufnahmeorts (Quelle: Google Street View)

Ähnlich zum Prozess in der Trainingsphase, wurden zufällig ca. 34 % der Aufnahmen zum Testen herausgenommen.

Die Ergebnisse aus Tabelle 11 und Abbildung 21 zeigen, dass das System insgesamt eine moderate Genauigkeit von 56 % erreicht. Bei genauer Betrachtung der einzelnen Klassen wird deutlich, dass das Modell für die Klassen KV und PKW vergleichsweise gute Werte für Precision und Recall erzielte. Dies deutet darauf hin, dass diese Klassen vom Modell verhältnismäßig gut erkannt werden.

Hingegen zeigt sich bei den Klassen Bus, LKW und MR erhebliche Schwierigkeiten. Insbesondere die Klasse LKW wurde vom Modell gar nicht korrekt erkannt (Precision, Recall und F1-Score von 0,00), was auf eine starke Fehlklassifikation hinweisen kann. Aus der Konfusionsmatrix lässt sich erkennen, dass Busse, LKWs und Motorräder häufig mit der PKW-Klasse verwechselt oder fälschlicherweise als Hintergrundgeräusche (KV) klassifiziert wurden.

Diese Ergebnisse spiegeln sich auch im Makro-Durchschnitt wider, der mit 0,31 beim F1-Score relativ niedrig ausfällt und verdeutlicht, dass die Leistung über alle Klassen hinweg ungleichmäßig ist.

Klasse	**Precision**	**Recall**	**F1-Score**	**Proben**
Bus	1,00	0,04	0,08	24
KV	0,44	0,91	0,60	34
LKW	0,00	0,00	0,00	8
MR	1,00	0,07	0,13	14
PKW	0,70	0,73	0,72	71
Makro-Durchschnitt	0,63	0,35	0,31	151
Genauigkeit			0,56	151

Tabelle 11: Klassifikationsbericht des Modells auf realen Aufnahmen

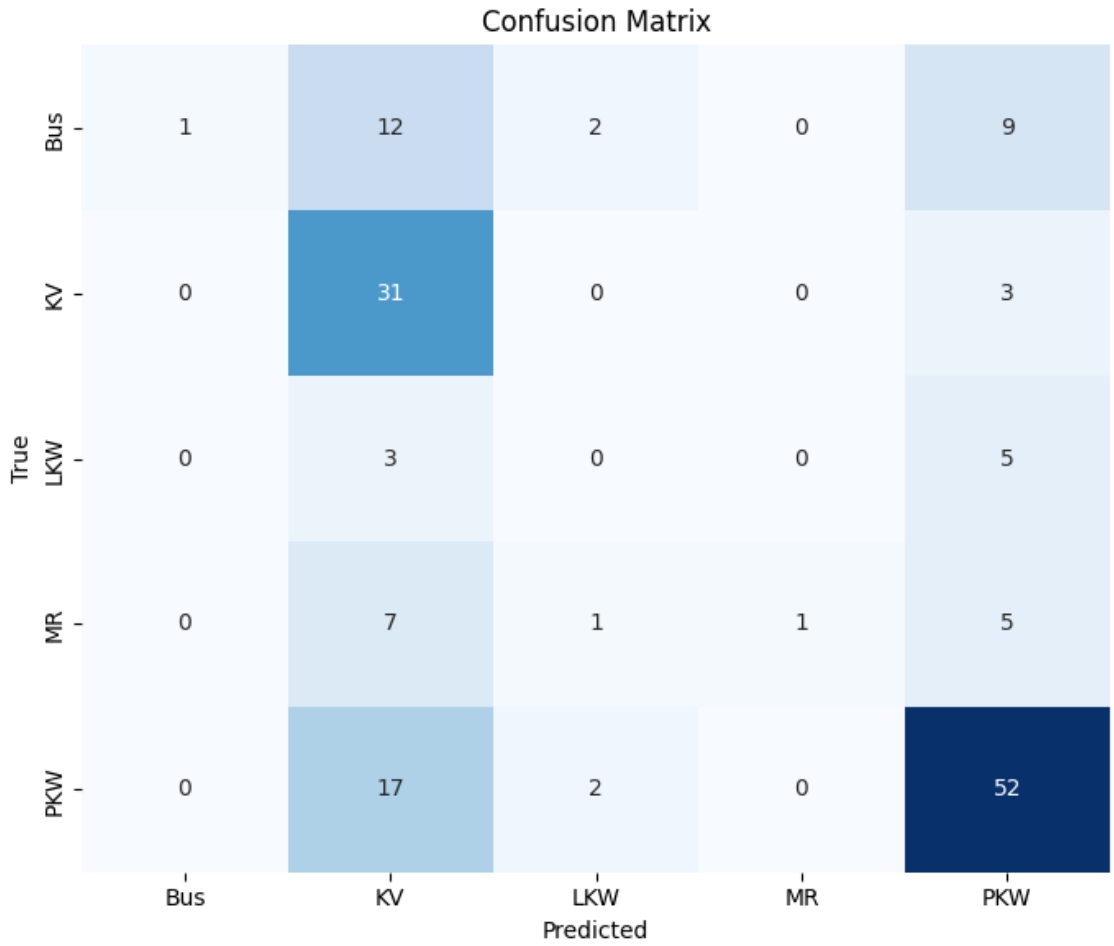

Abbildung 21: Konfusionsmatrix des Modells bei realen Aufnahmen

Zusammenfassend lässt sich sagen, dass das System vor allem bei kleineren oder seltener vertretenen Klassen Schwierigkeiten hat, diese zuverlässig zu erkennen, während häufigere Klassen wie „PKW“ deutlich besser klassifiziert werden.

Das Modell wird nun mit den verbleibenden 291 Aufnahmen, die zuvor fünffach augmentiert wurden und somit insgesamt 1.746 Aufnahmen ergeben, weitertrainiert. Ziel ist es, das Modell an das neue Mikrofon anzupassen und neu zu lernen, wie diese Aufnahmen korrekt zuzuordnen sind.

Wie Abbildung 22 zeigt, startete das Modell mit einer niedrigen Validierungsgenauigkeit von etwa 52 %. Nach circa 300 Trainings-Epochen konnte diese Genauigkeit kontinuierlich gesteigert werden und erreichte schließlich einen Höchstwert von etwa 95 %. Dieses Wachstum verdeutlicht den erfolgreichen Anpassungsprozess des Modells zu den neuen Aufnahmen.

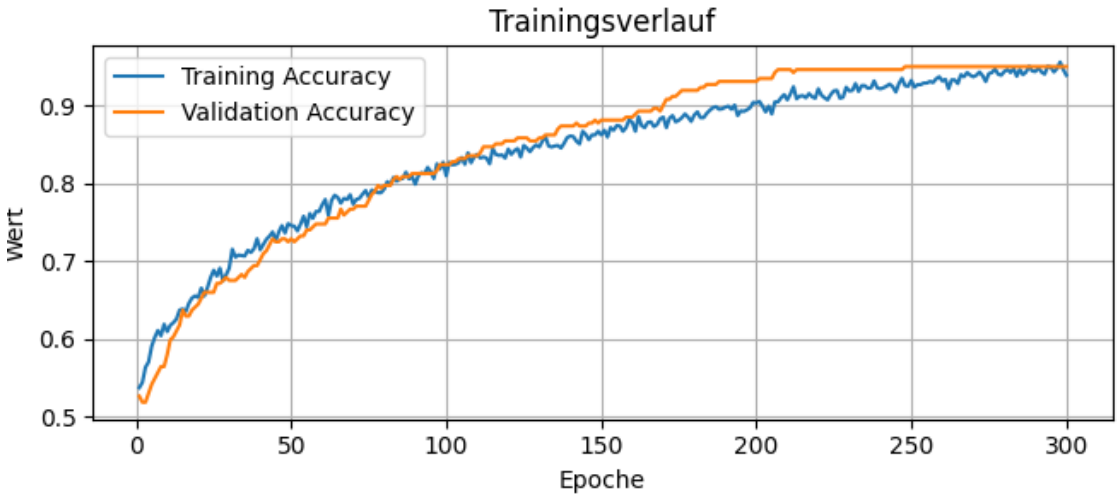

Abbildung 22: Verlauf des Weitertrainieren des Modells

Die abschließenden Testergebnisse, welche in Tabelle 12 zusammengefasst sind, zeigen eine Gesamtgenauigkeit von 81 %. Damit ist die Genauigkeit im Vergleich zu vorherigen Tests um 25 % gestiegen. Diese Verbesserung beweist, dass das Modell durch das weitere Training besser generalisiert und die spezifischen Merkmale der neuen Mikrofonaufnahmen erfolgreich erlernt hat.

Die detaillierte Betrachtung der einzelnen Klassen zeigt, dass alle Klassen außer der LKW-Klasse deutliche Leistungssteigerungen verzeichnen konnten. Besonders hervorzuheben ist der hohe F1-Score der Klasse KV (Kein Verkehr), welches darauf hinweist, dass das Modell nun in der Lage ist, zuverlässig zwischen verkehrsrelevanten und nicht verkehrsrelevanten Geräuschen zu unterscheiden. Diese Fähigkeit ist essenziell für den praktischen Einsatz, um Fehlalarme durch Hintergrundgeräusche zu minimieren.

Die geringe Performance bei der LKW-Klasse ist vermutlich auf die geringe Anzahl an Trainingsbeispielen zurückzuführen. Um diese Klasse zukünftig besser zu klassifizieren, sind nicht nur mehr Trainingsdaten erforderlich, sondern auch eine präzisere Definition, welche Fahrzeuge konkret als LKW eingestuft werden. Nur mit einer klaren Klassendefinition kann das Modell zuverlässig lernen, die LKW-Geräusche von denen anderer Fahrzeugklassen zu unterscheiden.

Insgesamt zeigt die Evaluation, dass das Modell durch das gezielte Feintraining an das neue Aufnahme-Setup angepasst wurde und nun eine deutlich verbesserte Klassifikationsleistung erzielt.

Klasse	**Precision**	**Recall**	**F1-Score**	**Proben**
Bus	0,64	0,75	0,69	24
KV	0,97	0,94	0,96	34
LKW	0,00	0,00	0,00	8
MR	0,89	0,57	0,70	14
PKW	0,89	0,90	0,90	71
Makro-Durchschnitt	0,68	0,63	0,65	151
Genauigkeit			0,81	151

Tabelle 12: Klassifikationsbericht des angepassten Modells

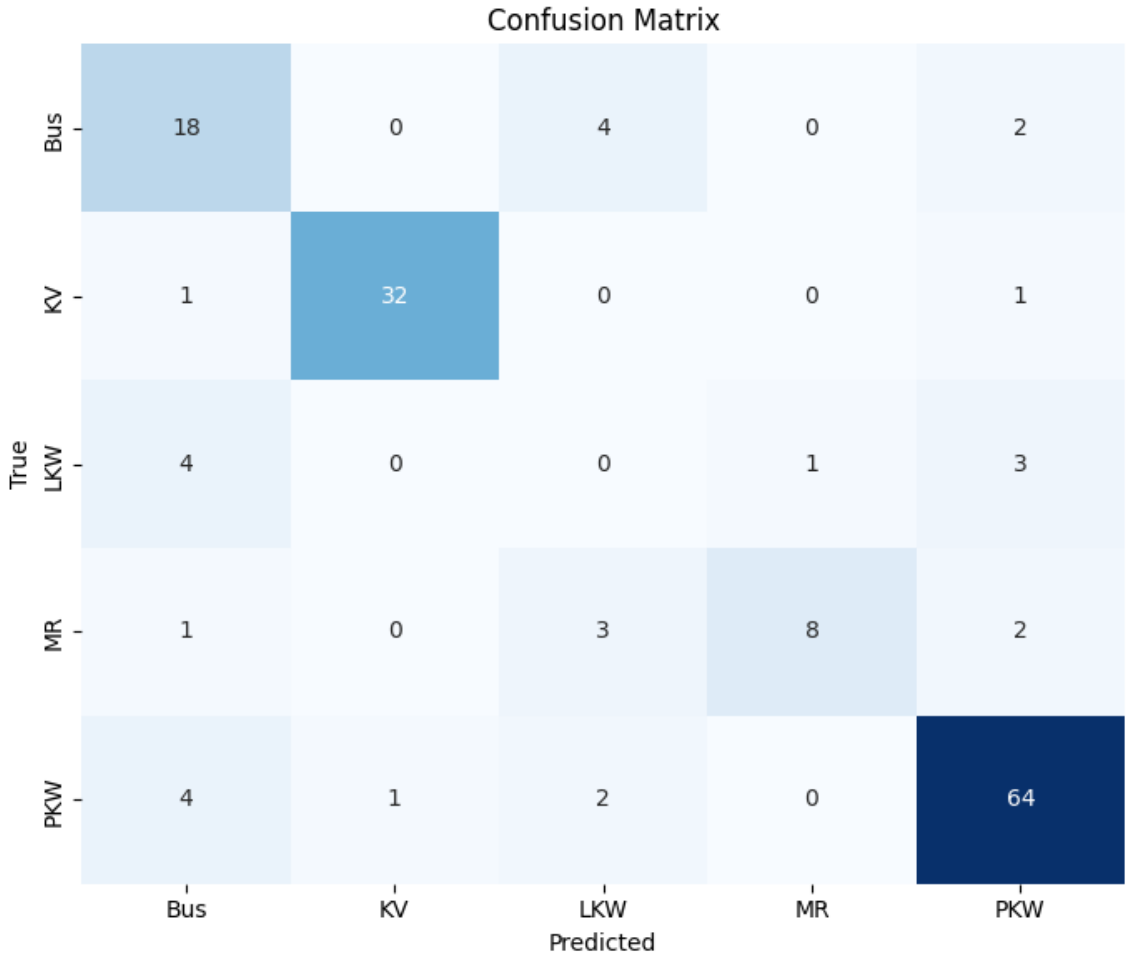

Abbildung 23: Konfusionsmatrix des angepassten Modells

Es wurden auch LogMel-Filterbanks (LogMel-Spektrogramme) in der Form 40×180 *als Merkmale getestet. Sie zeigten eine leicht bessere Leistung gegenüber den MFCCs, sowohl in Bezug auf die Rechenzeit als auch auf die Klassifikationsgenauigkeit. Da dies jedoch nicht dem ursprünglichen Ziel der Arbeit entspricht, wurde auf weiterführende Untersuchungen verzichtet.*

6.3 Evaluierung des Systems unter Dauereinsatzbedingungen

Die bisherigen Tests basierten auf isolierten Einzelaufnahmen, bei denen jeweils nur ein Fahrzeug oder eine Geräuschquelle vorhanden war. Dies ermöglichte zwar eine gezielte Analyse der Modellleistung, bildet jedoch die komplexen akustischen Bedingungen im realen Straßenverkehr unzureichend ab. Um die Praxistauglichkeit des Systems realistisch zu bewerten, wird es daher nun in einem kontinuierlichen Einsatzszenario getestet, in dem unterschiedlichste Verkehrsgeräusche gleichzeitig auftreten können und keine manuelle Selektion der Aufnahmen erfolgt.

Ein ideales Testszenario für den kontinuierlichen Einsatz des Systems wäre eine einspurige oder zweispurige Straße mit geringem bis mittlerem Verkehrsaufkommen. Dabei sollten die Fahrzeuge am besten einzeln oder mit ausreichend Abstand nacheinander vorbeifahren, sodass die Geräusche klar voneinander getrennt erfasst und verarbeitet werden können.

Ein solches Testszenario konnte aufgrund des begrenzten zeitlichen Rahmens der Abschlussarbeit nicht realisiert werden. Stattdessen wurden die Tests an dem in Abbildung 20 gezeigten Aufnahmeort in realer Umgebung aus einer Entfernung von etwa 1 bis 12 Metern zur Fahrbahn durchgeführt.

Ziel dieses Tests war es, das entwickelte System unter realen Verkehrsbedingungen im kontinuierlichen Betrieb zu evaluieren. Dabei sollte überprüft werden, inwiefern die Klassifikation in Echtzeit funktioniert und sich für eine potenzielle Anwendung zur Verkehrsanalyse eignet. Zur kontinuierlichen Erfassung der akustischen Umgebung wird das Python-Modul PyAudio eingesetzt. Der Audiodatenstrom wird zunächst in einem internen Puffer der Größe 4096 Frames verarbeitet. Bei einer Abtastrate von 16 kHz entspricht dies etwa 256 ms pro Lesevorgang. Die aufgenommenen Samples werden fortlaufend in einen externen Ringpuffer vom Typ deque (Double-Ended Queue) mit einer festen Länge von 32.000 Samples geschrieben, was einer Dauer von 2 Sekunden entspricht. Dieser Puffer aktualisiert sich automatisch, indem ältere Daten verworfen und neue hinzugefügt werden, sobald neue Audiodaten eintreffen. Parallel dazu läuft ein separater Thread, der in regelmäßigen Intervallen von 2 Sekunden den aktuellen Zustand dieses Puffers abruft, um daraus die MFFC-Merkmale zu extrahieren und eine Klassifikation durchzuführen. Die ermittelten Vorhersagen werden zusammen mit einem Zeitstempel sowie der berechneten Lautstärke der Aufnahme in LUFS (Loudness Units relative to Full Scale) in einer CSV-Datei protokolliert. Dies ermöglicht im Nachhinein eine Analyse des Verkehrsaufkommens über einen bestimmten Zeitraum hinweg.

Im Rahmen mehrerer 15-minütiger Beobachtungszeiträume konnten verschiedene Stärken und Schwächen des Systems unter realen Verkehrsbedingungen identifiziert werden. Bei Einzelfahrzeugen zeigte das System eine hohe Klassifizierungsgenauigkeit, jedoch wurden einzelne Fahrzeuge teilweise zwei- bis dreimal gezählt. Bei Szenarien mit mehreren Fahrzeugen gleichzeitig war die Klassifikation tendenziell weniger präzise, da dominante Fahrzeugklassen häufig bevorzugt wurden. Dennoch trat bei solchen Konstellationen ein geringerer Fehler beim Zählen auf, da meist nur das erste und letzte Fahrzeug doppelt erfasst wurden. Besonders zuverlässig wurden PKWs, Busse und Motorräder erkannt, während LKWs häufig fälschlich als Busse klassifiziert wurden. Eine Entfernung von 1–6 Metern zwischen Fahrzeugen und Mikrofon erwies sich als besonders vorteilhaft für eine fehlerarme Erkennung. Die Unterscheidung zwischen Verkehr und keinem Verkehr funktionierte in den meisten Fällen zuverlässig, mit Ausnahme von besonders leisen Fahrzeugen. Hintergrundgeräusche wie Wind hatten keinen erkennbaren Einfluss auf die Klassifizierungsgenauigkeit. Auch die Mikrofonpositionierung zeigte keinen nennenswerten Effekt, da das verwendete Mikrofon omnidirektional arbeitet. Die folgende Abbildung zeigt ein Beispiel für die Visualisierung der Systemausgabe:

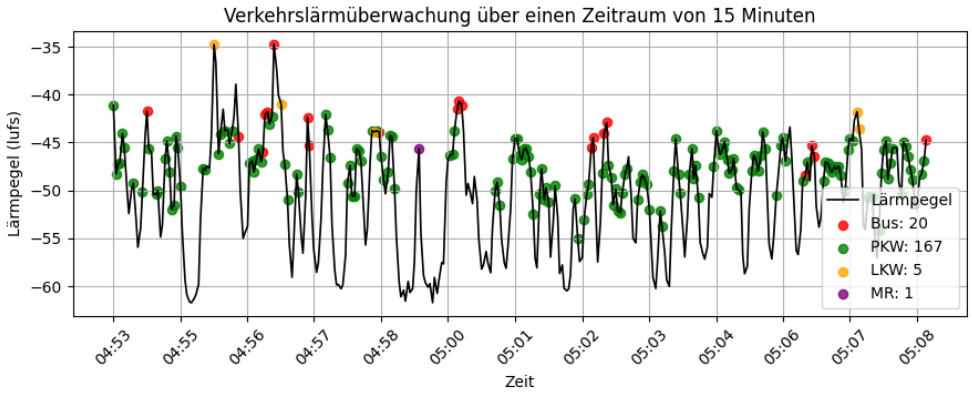

Abbildung 24: Visualisierung der Systemausgabe

Zur Verbesserung des Systems könnten insbesondere mehr Trainingsdaten für LKWs gesammelt werden. Die durchschnittliche Lautstärke lag bei Aufnahmen ohne Verkehr etwa 15 LUFS unter dem Pegel während verkehrsbedingter Geräusche. Diese Differenz könnte genutzt werden, um eine Schwelle zu definieren, ab der überhaupt eine Klassifikation ausgelöst wird. Darüber hinaus könnte bei wenig befahrenen Straßen eine Erhöhung des Abfrageintervalls für den Pufferinhalt von 2 auf beispielsweise 3 Sekunden helfen, um doppelte Zählungen zu reduzieren.

Auch hinsichtlich der Systemleistung auf dem Zielgerät, dem Raspberry Pi Zero 2, konnten im Testbetrieb positive Beobachtungen gemacht werden. Die CPU-Auslastung lag im Leerlauf bei lediglich ca. 0,2 % und stieg während der Laufzeit des Skripts nur gelegentlich auf Werte bis maximal 10 %. Die Temperatur des Prozessors blieb während des gesamten Betriebs unkritisch und erreichte höchstens 43 °C. Der Arbeitsspeicherverbrauch erhöhte sich von etwa 22 % im Leerlauf auf rund 33 % während der aktiven Signalverarbeitung und Inferenz. Diese Werte belegen, dass die gewählte technische Umsetzung effizient arbeitet und die Hardwareanforderungen gering bleiben. Insgesamt spiegeln die Messungen eine sehr gute Performance des Raspberry Pi Zero 2 im Rahmen dieser Anwendung wider.

In Bezug auf Energieverbrauch zeigte sich das System effizient. Der Raspberry Pi Zero 2 konnte über mehrere Stunden problemlos mit einer 5000 mAh-Powerbank betrieben werden. Für einen kontinuierlichen und langfristigen Einsatz wäre jedoch eine zuverlässige Stromversorgung erforderlich.

Ein großer Nachteil des Systems war die erschwerte Datenübertragung. Da die Daten lokal auf dem Raspberry Pi gespeichert werden, müssen diese manuell ausgelesen und gesichert werden. Dieser manuelle Schritt stellt zwar aus Datenschutzsicht einen Vorteil dar, da keine Übertragung über Netzwerke erfolgt, bedeutet jedoch auch ein erhöhtes Risiko für Datenverlust, insbesondere wenn die Speicherkarte beschädigt wird oder ausfällt.

7 Fazit und Ausblick

Im Rahmen dieser Arbeit wurde ein vollständiges System zur akustischen Fahrzeugklassifikation in Echtzeit entwickelt und getestet. Ziel war es, ein kompaktes, datenschutzfreundliches und hardwareeffizientes System zu realisieren, das Verkehrslärmsignale erfasst, zwischen Verkehr und keinem Verkehr unterscheidet und Fahrzeuge den Klassen PKW, LKW, Bus oder Motorrad zuordnet. Die technische Umsetzung erfolgte auf einem Raspberry Pi Zero 2, wobei sämtliche Verarbeitungsschritte – von der Audioaufnahme bis zur Klassifikation – lokal ausgeführt wurden. Die Merkmalsextraktion basierte auf MFCCs mit einer Form von 13×180. Im Rahmen der Modellwahl wurden drei Architekturen miteinander verglichen, wobei sich ein 2D-CNN als leistungsfähigste Variante erwies und in der Folge weiterverwendet wurde. Die Anwendung gezielter Datenaugmentierung zeigte besonders bei unterrepräsentierten Klassen eine verbesserte Erkennungsleistung. Zudem konnte durch Quantisierung die Modellgröße um den Faktor 12 reduziert werden, ohne die Klassifikationsgenauigkeit zu beeinträchtigen.

Frühe Tests zeigten zunächst unplausible Ergebnisse, die sich auf unterschiedliche Ursachen wie Mikrofonverstärkung, Datenformate und Bibliotheken zurückführen ließen. Ein eigens erstellter Datensatz half dabei, diese Probleme gezielt zu analysieren und zu beheben. Nach erneuter Modellanpassung und Training mit einem erweiterten Datensatz konnte eine Genauigkeit von 81 % erreicht werden, mit einem Makrodurchschnitt des F1-Scores von 65 %. Ein anschließender Testeinsatz unter realen Bedingungen bestätigte, dass das System grundsätzlich in der Lage ist, Fahrzeuge in Echtzeit zu erkennen und Verkehrszustände zuverlässig zu klassifizieren. Einzelne Fahrzeuge wurden dabei allerdings teilweise doppelt gezählt, während bei mehreren gleichzeitig auftretenden Fahrzeugen die Genauigkeit etwas abnahm. Besonders zuverlässig funktionierte die Erkennung bei PKWs, Bussen und Motorrädern, während LKWs häufiger fälschlich als Busse klassifiziert wurden. Die Unterscheidung zwischen Verkehr und keinem Verkehr gelang in den meisten Fällen zuverlässig, wobei leise Fahrzeuge gelegentlich unentdeckt blieben. Windgeräusche oder die Mikrofonposition zeigten keinen relevanten Einfluss auf die Modellleistung. Die Systemleistung auf dem Raspberry Pi Zero 2 erwies sich als durchweg ausreichend: Die CPU-Auslastung blieb niedrig, die Temperatur stabil, und auch der Energieverbrauch ermöglichte einen mehrstündigen Betrieb mit einer handelsüblichen Powerbank. Als Einschränkung stellte sich jedoch die fehlende automatische Datenübertragung heraus, da sämtliche Daten manuell ausgelesen werden müssen.

Insgesamt zeigt die Arbeit, dass eine KI-gestützte Fahrzeugidentifikation auf kompakten, energieeffizienten Systemen unter Beachtung von Datenschutzaspekten grundsätzlich machbar ist. Für zukünftige Arbeiten bieten sich zahlreiche Weiterentwicklungen an. Insbesondere sollten mehr Trainingsdaten für LKWs erhoben werden, um deren Erkennungsrate zu verbessern. Auch alternative Merkmalsdarstellungen wie Mel-Spektrogramme könnten getestet werden, ebenso wie modernere oder effizientere Modellarchitekturen. Verbesserungen bei der Ereignisdetektion, etwa durch Triggermechanismen auf Basis von Lautstärkeverläufen, könnten dabei helfen, unnötige oder doppelte Klassifikationen zu vermeiden. Zusätzlich könnte die Einführung einer Zähllogik oder eines Online-Lernverfahrens zur kontinuierlichen Modellanpassung die Praxistauglichkeit erhöhen. Langfristig wäre auch eine automatisierte und sichere Datenübertragung über Funknetzwerke sowie eine energieautarke Stromversorgung wünschenswert. Insgesamt stellt das entwickelte System eine vielversprechende Grundlage für weitere Forschung und Entwicklung im Bereich der akustischen Verkehrsanalyse dar.

Literatur

[1] D. L. Fugal und R. G. Lyons, *Essential Guide to Digital Signal Processing*. Pearson, 2014.

[2] I. Goodfellow, Y. Bengio und A. Courville, *Deep Learning*. MIT Press, 2016.

[3] A. Moodley, *Language Identification With Decision Trees: Identification Of Individual Words In The South African Languages*, Bachelorthesis, 2016.

[4] E. Union, *Datenschutz-Grundverordnung*, Zugriff am 24.04.2025, 2016. Adresse: https://dsgvo-gesetz.de.

[5] T. Kaffka, *Neuronale Netze - Grundlagen*. mitp Verlag, 2017.

[6] M. Valenti, S. Squartini, A. Diment, G. Parascandolo und T. Virtanen, „A Convolutional Neural Network Approach for Acoustic Scene Classification," in *Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, S. 1547–1554. DOI: 10.1109/IJCNN.2017.7966035.

[7] United Nations Department of Economic and Social Affairs (UN DESA), *Grad der Urbanisierung in Deutschland und weltweit von 1950 bis 2015 und Prognose bis 2050*, Zugriff am 18.04.2025, 2018. Adresse: https://de.statista.com/statistik/daten/studie/152879/umfrage/in-staedten-lebende-bevoelkerung-in-deutschland-und-weltweit/.

[8] P. Montino und D. Pau, „Environmental Intelligence for Embedded Real-time Traffic Sound Classification," in *2019 IEEE 5th International forum on Research and Technology for Society and Industry (RTSI)*, 2019, S. 45–50.

[9] National Cyber Security Center, *Biometric Recognition and authentication Systems*, Zugriff am 24.04.2025, 2019. Adresse: https://www.ncsc.gov.uk/collection/biometrics.

[10] akob Abeßer, S. Gourishetti, A. Kátai, T. Clauß, P. Sharma und J. Liebetrau, „IDMT-Traffic: An Open Benchmark Dataset for Acoustic Traffic Monitoring Research," *EUSIPCO*, 2021.

[11] F. Chollet, *Deep Learning with Python, Second Edition*. Manning Publications, 2021.

[12] S. Wyatt, D. Elliott, A. Aravamudan u. a., „Environmental Sound Classification with Tiny Transformers in Noisy Edge Environments," in *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, 2021, S. 309–314.

[13] L. Yuxi, C. Ligong, W. Qian und Z. Xinghong, „Sound-Convolutional Recurrent Neural Networks for Vehicle Classification Based on Vehicle Acoustic Signals," in *2021 International Conference on Smart City and Green Energy (ICSCGE)*, 2021, S. 98–102. DOI: 10.1109/ICSCGE53744.2021.9654357.

[14] Z. K. Abdul und A. K. Al-Talabani, „Mel Frequency Cepstral Coefficient and its Applications: A Review," *EEE Access*, Jg. 12, 2022.

[15] M. Asif, M. Usaid, M. Rashid, T. Rajab, S. Hussain und S. Wasi, „Large-scale audio dataset for emergency vehicle sirens and road noises," *Sci Data 9, 599*, 2022.

[16] A. V. Joshi, *Machine Learning and Artificial Intelligence*. Springer, Cham, 2022.

[17] Y. Qu, X. Li, Z. Qin und Q. Lu, „Acoustic scene classification based on three-dimensional multi-channel feature-correlated deep learning networks," *Scientific Reports volume 12, Article number: 13730*, 2022.

[18] D. Sundararajan, *Signals and Systems*. Springer, Cham, 2022.

[19] M. Ashhad, U. Goenka, A. Jagetia, P. Akhtari, S. K. Ambat und M. Samuel, „Improved Vehicle Sub-type Classification for Acoustic Traffic Monitoring," *Twenty-Ninth National Conference on Communications(NCC)*, 2023.

[20] G. M. Iodice, *TinyML Cookbook - Second Edition*. Packt Publishing, 2023.

[21] S. Jauss, *KI im Audiobereich*, Zugriff am 05.05.2025, 2023. Adresse: https://ai.hdm-stuttgart.de/news/2023/ki-im-audiobereich-grundlagen-signalverarbeitung-ml/.

[22] H. Li, *Machine Learning Methods*. Springer, Singapore, 2023.

[23] S. Palani, *Discrete Time Systems and Signal Processing*. Springer, Cham, 2023.

[24] Pangeanic, *Audio Data Augmentation: Techniques and Methods*, 2023. Adresse: https://blog.pangeanic.com/audio-data-augmentation-techniques-and-methods.

[25] M. Shiga und S. Watanabe, *Hyperordered Structures in Materials*. Springer, Singapore, 2023.

[26] B. Somwong, K. Kumphet und W. Massagram, „Acoustic Monitoring System with AI Threat Detection System for Forest Protection," in *2023 20th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2023, S. 253–257.

[27] A. I. Yassin, K. K. M. Shariff, M. A. Kechik, A. M. Ali und M. S. M. Amin, „Acoustic Vehicle Classification Using Mel-Frequency Features with Long Short-Term Memory Neural Networks," *TEM Journal*, Jg. 12, Nr. 3, S. 1490–1496, 2023.

[28] M. Bellanger und B. A. Engel, *Digital Signal Processing*. Wiley, 2024.

[29] Cole Stryker and Eda Kavlakoglu, *What is artificial intelligence (AI)?* Zugriff am 02.04.2025, 2024. Adresse: https://www.ibm.com/think/topics/artificial-intelligence.

[30] L. B. Das, *Embedded Systems: An Integrated Approach by Pearson*. Pearson India, 2024.

[31] Deutsche Umwelthilfe, *Deutsche Umwelthilfe fordert Tempo 30 als Regelgeschwindigkeit innerorts und verschärftes Lärm-Limit für Pkw und Motorräder gegen gesundheitsschädlichen Verkehrslärm*, Zugriff am 17.02.2025, 2024. Adresse: https://www.duh.de/presse/pressemitteilungen/pressemitteilung/deutsche-umwelthilfe-fordert-tempo-30-als-regelgeschwindigkeit-innerorts-und-verschaerftes-laerm-limit/.

[32] Z. Huang, A. Tousnakhoff, P. Kozyr u. a., „TinyChirp: Bird Song Recognition Using TinyML Models on Low-power Wireless Acoustic Sensors," in *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*, 2024, S. 1–10.

[33] H. Park, S. Hong und J.-S. Kim, „Audio Classification on Low-Resource Microcontrollers," in *2024 15th International Conference on Information and Communication Technology Convergence (ICTC)*, 2024, S. 1332–1333.

[34] M. H. Sadraey, *Unmanned Aircraft Design*. Springer, Cham, 2024.

[35] A. Shabbir, A. N. Cheema, I. Ullah, I. M. Almanjahie und F. Alshahran, „Smart City Traffic Management: Acoustic-Based Vehicle Detection Using Stacking-Based Ensemble Deep Learning Approach," *IEEE Access*, Jg. 12, 2024.

[36] TÜV-Verband, *Was sind Ihrer Meinung nach die größten Probleme, die durch Verkehr verursacht werden?* Zugriff am 18.04.2025, 2024. Adresse: https://de.statista.com/statistik/daten/studie/1245548/umfrage/durch-strassenverkehr-versursachte-probleme/.

[37] C. Ünsalan, B. Höke und E. Atmaca, *Embedded Machine Learning with Microcontrollers*. Springer, Cham, 2024.

[38] World Bank, *Urbanisierungsgrad in Deutschland von 1990 bis 2023 (Anteil der Stadtbewohner an der Gesamtbevölkerung)*, Zugriff am 18.04.2025, 2024. Adresse: https://de.statista.com/statistik/daten/studie/662560/umfrage/urbanisierung-in-deutschland/.

[39] World Health Organization, *Deutsche Umwelthilfe fordert Tempo 30 als Regelgeschwindigkeit innerorts und verschärftes Lärm-Limit für Pkw und Motorräder gegen gesundheitsschädlichen Verkehrslärm*, Zugriff am 16.04.2025, 2024. Adresse: https://www.who.int/europe/news/item/04-08-2024-how-much-does-environmental-noise-affect-our-health--who-updates-methods-to-assess-health-risks.

[40] P. Gairí, T. Pallejà und M. Tresanchez, „Environmental sound recognition on embedded devices using deep learning: a review," *Artificial Intelligence Review*, Jg. 58, Nr. 1, S. 163, 2025.

[41] L. Hulatt, *Vanishing Gradient*, Zugriff am 05.05.2025, 2025. Adresse: https://www.studysmarter.de/studium/ingenieurwissenschaften/maschinelles-lernen-studium/vanishing-gradient/.

[42] U. Karrenberg, *Universalschlüssel Fourier-Transformation*. Springer Vieweg, Wiesbaden, 2025.

[43] Z. " Liu, *Artificial Intelligence for Engineers*. Springer, Cham, 2025.

[44] T. Rohner, Zugriff am 15.05.2025, 2025. Adresse: https://www.netguru.com/blog/cpp-vs-python.

[45] M. H. Trauth, *MATLAB Recipes for Earth Sciences*. Springer, Cham, 2025.

[46] E. B. Brixen, *Mikrofontechnologie – die Grundlagen*, Zugriff am 20.05.2025. Adresse: https://www.dpamicrophones.com/de/mic-university/technology/microphone-technology-the-essentials/.

[47] K. Doshi, *Batch Norm Explained Visually – How it works, and why neural networks need it*, Zugriff am 23.05.2025. Adresse: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739/.

[48] B. Electronics, Zugriff am 01.05.2025. Adresse: https://www.berrybase.de/.

[49] T. Gjestland, *Background noise levels in Europe*.

[50] EU-Info.Deutschland, *EU-Prüfer: Weiter schlechte Luft und Lärm in den Städten*, Zugriff am 14.05.2025. Adresse: https://www.eu-info.de/dpa-europaticker/328185.html.

[51] I. Jordal, *A Python library for audio data augmentation*. Adresse: https://github.com/iver56/audiomentations.

[52] Keras. Adresse: https://keras.io/.

[53] C. Kurrer und Z. Wala, *Luftverschmutzung und Lärmbelastung*, Zugriff am 14.05.2025. Adresse: https://www.europarl.europa.eu/factsheets/de/sheet/75/luftverschmutzung-und-larmbelastung.

[54] J. Lyons, *python_speech_features: Speech feature extraction for python*. Adresse: https://github.com/jameslyons/python_speech_features.

[55] scikit-learn. Adresse: https://scikit-learn.org/stable/.

[56] I. M. da Silva, *Umgebungslärmrichtlinie: weit entfernt von Wirksamkeit*, Zugriff am 14.05.2025. Adresse: https://de.euronews.com/my-europe/2025/02/25/umgebungslarmrichtlinie-weit-entfernt-von-wirksamkeit.

Abkürzungsverzeichnis

DSGVO Datenschutz-Grundverordnung
BDSG Bundesdatenschutzgesetz
HDL Hardware Description Language
ISO International Organization for Standardization
IEC International Electrotechnical Commission
MCU Microcontroller Unit
KI Künstliche Intelligenz
ML Machine Learning / Maschinelles Lernen
ANN Artifical Neural Network
CNN Convolutional Neural Network
RNN Recurrent Neural Network
MFCC Mel-Frequency Cepstral Coefficients
RMSE Root Mean Square Error
MLP Multi Layer Perceptron
DNN Deep Neural Network
LSTM Long Short-Term Memory
ADC Analog-Digital Converter
DAC Digital-Analog Converter
FT Fourier-Transformation

Anhang

Code Beispiele

```
import numpy as np
import matplotlib.pyplot as plt

# Zeitbereich
# 100 Werte zwischen 0 und 1 (1 Sekunde)
zeit_achse = np.linspace(0, 1, 100)

# kontinuierliches Signal definieren
signal_kontinuierlich = 5 * np.sin(2 * np.pi * 100 * zeit_achse ) + \
    1 * np.cos(2 * np.pi * 400 * zeit_achse ) + \
    0.5 * np.sin(2 * np.pi * 800 * zeit_achse)


# Abtastung
sample_step = 3
time_samples = zeit_achse[::sample_step]
signal_diskret = signal_kontinuierlich[::sample_step]

# Digitalisiertes Signal (quantisiert)
quant_levels = 8  # 8 Zahlen -> 3 Bits
value_min = np.min(signal_diskret)
value_max = np.max(signal_diskret)

signal_norm = (signal_diskret - value_min) / (value_max - value_min)

signal_digital = (np.round(signal_norm * (quant_levels - 1))) / (
   quant_levels - 1)
signal_digital = signal_digital * (value_max - value_min) + value_min

# Plot
fig, axs = plt.subplots(3, 1, figsize=(8, 10))

# 1. Zeitkontinuierlich
```

```
axs[0].plot(zeit_achse, signal_kontinuierlich, label="Zeitkontinuierliches Signal")
axs[0].set_title('A - Zeitkontinuierliches Signal')
axs[0].set_xlabel('Zeit [s]')
axs[0].set_ylabel('Amplitude')
axs[0].set_xticks(np.linspace(0,1,11))
axs[0].legend(loc='upper right')
axs[0].grid(True)

# 2. Zeitdiskret
axs[1].plot(zeit_achse, signal_kontinuierlich, color="gray", alpha=0.7, label="Zeitkontinuierliches Signal")
axs[1].stem(time_samples, signal_diskret, basefmt=" ", label="Zeitdiskretes Signal")
axs[1].set_title('B - Zeitdiskretes Signal')
axs[1].set_xlabel('Zeit [s]')
axs[1].set_ylabel('Amplitude')
axs[1].set_xticks(np.linspace(0,1,11))
axs[1].legend(loc='upper right')
axs[1].grid(True)

# 3. Digitales Signal (quantisiert)
axs[2].plot(zeit_achse, signal_kontinuierlich, color="gray", alpha=0.3, label="Zeitkontinuierliches Signal")
axs[2].scatter(time_samples, signal_diskret, color="gray", alpha=0.6, label="Zeitdiskretes Signal")
axs[2].step(time_samples, signal_digital, where="mid", label="Digitales Signal")
axs[2].set_title('C - Digitales Signal (zeit- und amplitudendiskret)')
axs[2].set_xlabel('Zeit [s]')
axs[2].set_ylabel('Amplitude')
axs[2].set_xticks(np.linspace(0,1,11))
axs[2].legend(loc='upper right')
axs[2].grid(True)

plt.tight_layout()
plt.show()
```

Listing 1: Signalformen generieren

```
import numpy as np
import matplotlib.pyplot as plt

N = 2000
T = 1.0
fs = N / T

zeit_achse = np.linspace(0, 1, 2000)

# kontinuierliches Signal definieren
signal_kontinuierlich = 5 * np.sin(2 * np.pi * 100 * zeit_achse ) + \
    1 * np.cos(2 * np.pi * 400 * zeit_achse ) + \
    0.5 * np.sin(2 * np.pi * 800 * zeit_achse)

# Fourier-Transformation
spectrum = np.fft.fft(signal_kontinuierlich)
freqs = np.fft.fftfreq(N, d=1/fs)

# Nur positive Frequenzen betrachten
idx = np.argsort(freqs)
```

```
positive_freqs = freqs[:N//2]
magnitude = 2.0 / N * np.abs(spectrum[:N//2])  # Amplitude korrekt skalieren

# Plot
plt.figure(figsize=(10, 4))
plt.plot(positive_freqs, magnitude)
plt.title("Frequenzspektrum des Signals")
plt.xlabel("Frequenz [Hz]")
plt.ylabel("Magnitude")
plt.xticks(np.linspace(0,1000,11))
plt.yticks(np.linspace(0,5,11))
plt.grid(True)
plt.show()
```

Listing 2: Fourier-Transformation berechnen