Masterarbeit

ENTWICKLUNG EINER CLOUDBASIERTEN KI-LÖSUNG ZUR ANALYSE VON AUDIOSIGNALEN

ausgeführt am

Fachhochschul-Masterstudiengang
Automatisierungstechnik-Wirtschaft

von
Daniel Schwaiger, BSc.
51841210

betreut und begutachtet von
FH-Prof. Dipl.-Ing. Dieter Lutzmayr

Graz, im November 2022

.....
Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....

Unterschrift

DANKSAGUNG

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Masterarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank Herrn FH-Prof. Dipl.-Ing Dieter Lutzmayr, der meine Masterarbeit betreut und begutachtet hat. Für die hilfreichen Anregungen und die konstruktive Kritik bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken.

Des Weiteren gilt mein Dank meiner Familie und meinen Freunden, die mir während meiner Studienzeit Rückhalt gegeben haben. Besonders möchte ich mich bei Janine für das Korrekturlesen sowie die zahl- und hilfreichen Tipps bedanken.

Außerdem möchte ich mich bei meinen Studienkollegen und Studienkolleginnen für die schöne Zeit an der FH Campus 02 bedanken. In den vergangenen Jahren wurde nicht nur das Wissen erweitert, sondern auch neue Freundschaften geschlossen, wofür ich sehr dankbar bin.

KURZFASSUNG

Künstliche Intelligenz (KI) und Cloud-Computing sind treibende Kräfte der digitalen Transformation und Erfolgsfaktoren für eine nachhaltige Wettbewerbsfähigkeit. Insbesondere der Bereich der KI-basierten Audiosignalverarbeitung weist ein hohes Potential zur Fehlererkennung von Maschinen und Anlagen auf. Jedoch scheitert die Umsetzung von KI-Projekten oftmals bereits vor Projektstart aufgrund fehlender Fachkenntnisse der Unternehmen.

Das Ziel dieser Masterarbeit ist zu zeigen, wie KI-basierte Audioklassifizierungssysteme unter Verwendung von Cloud-Services implementiert werden können. Zu diesem Zweck werden die einzelnen Phasen eines KI-Projektes, von der Datenanalyse bis hin zur Bereitstellung eines fertig trainierten Modells in der Cloud-Umgebung, betrachtet. Frühere Arbeiten haben gezeigt, dass State-of-the-Art-Audioklassifizierungs-systeme auf Konzepten wie der Fourier-Analyse, Convolutional Neural Networks (CNN) und Recurrent Neural Networks (RNN) basieren. Anhand dieser Methoden wurden insgesamt 33 Klassifizierungsmodelle mittels Python, PyTorch und der cloudbasierten Plattform Google Vertex AI implementiert, trainiert und verglichen. Aufgrund der dynamischen Charakteristik der Audiodateien, wurde dazu ein komplexer Datensatz der Plattform Kaggle als Entwicklungsgrundlage verwendet (BirdCLEF2022).

Das ausgewählte Modell wurde hinsichtlich der Vorhersagegenauigkeit optimiert und auf Vertex AI zur Beantwortung von Vorhersageanfragen veröffentlicht. Dabei konnte ein auf der CNN-Architektur basierendes Klassifizierungsmodell entwickelt werden, das neun unterschiedliche Klassen mit einer Vorhersagegenauigkeit von 80,4 % klassifiziert. Weitere Ideen zur Verbesserung des Ergebnisses konnten vorgestellt werden, wodurch bewiesen wird, dass schwierige Daten mit einer Vorhersagegenauigkeit von über 90 % klassifiziert werden können. Diese Masterarbeit zeigt, wie ein KI-basiertes Audioklassifizierungssystem unter Verwendung verschiedener Cloud-Dienste und State-of-the-Art-Deep-Learning-Methoden, entwickelt werden kann.

ABSTRACT

Artificial intelligence (AI) and cloud computing are driving forces of digital transformation and a success factor in sustainable competitiveness. In particular, the field of AI-based audio signal processing displays considerable potential with regard to error detection in machines and systems. However, the implementation of AI projects often fails before the projects commence due to a lack of expertise.

The aim of this master's thesis is to demonstrate how AI-based audio classification systems can be implemented using cloud services. For this purpose, the individual phases of an AI project are considered, ranging from data analysis to the deployment of a fully trained model in the cloud environment. Previous work has shown that state-of-the-art audio classification systems are based on concepts such as Fourier analysis, convolutional neural networks (CNN) and recurrent neural networks (RNN). Using these methods, a total of 33 different classification models were implemented, trained and compared using Python, PyTorch and the cloud-based platform Google Vertex AI. Due to the dynamic characteristics of the audio files, a complex dataset from the Kaggle platform was used as the basis for development (BirdCLEF2022).

The chosen model was optimized in terms of prediction accuracy and published on Vertex AI to answer prediction queries. Ultimately a classification model based on the CNN architecture was developed that classifies nine different classes with a prediction accuracy of 80.4 %. Further ideas for additional improvement of the classification results were able to be presented, which proves that difficult data can be classified with a prediction accuracy above 90 %. This master's thesis demonstrates how an AI-based audio classification system can be developed by using various cloud services and state-of-the-art deep learning methods.

INHALTSVERZEICHNIS

1 EINLEITUNG

Werden heutzutage zukunftsweisende Technologien betrachtet, führt kein Weg an der künstlichen Intelligenz (KI) vorbei. Spätestens mit der Einführung der modernen KI und von Konzepten wie künstlichen neuronalen Netzwerken (KNN) sowie tiefem Lernen (engl. Deep Learning, DL) findet diese Technologie zunehmend Anwendung in der Praxis. Zeitgleich schreitet im industriellen Umfeld die Digitalisierung rasant voran. Um Maschinen intelligent miteinander zu vernetzen, müssen zahlreiche Daten ausgetauscht und gesammelt werden. Dabei eignen sich Cloud-Umgebungen nicht nur als zentraler Knotenpunkt für den Austausch, sondern auch zum Speichern von Daten, denn Cloud-Provider stellen kostengünstige und skalierbare Hardwareressourcen zur Verfügung. Doch das alleinige Austauschen und Sammeln von Daten macht eine Produktionsumgebung noch nicht intelligent. Die Daten müssen analysiert und weiterverarbeitet werden. Zu diesem Zweck bieten Cloud-Anbieter wie Amazon, Google und Microsoft zusätzlich Plattform- und Softwareservices im Bereich der KI an. Diese Services können einzelne Phasen eines KI-Projektes vereinfachen und unterstützen somit den Entwicklungsprozess, denn die Anwendung der KI-Technologie scheitert oftmals bereits vor Projektstart aufgrund fehlender Fachkenntnisse.

Die Anwendungsfelder von KI sind vielfältig. Neben Bereichen wie der visuellen Datenverarbeitung und der Sprachanalyse stellt auch die Audiosignalverarbeitung ein eigenes Fachgebiet dar. Letzteres kann beispielsweise zur Analyse von Maschinengeräuschen und zur Qualitätsprüfung verwendet werden. Ferner bildet die KI-basierte Audiosignalverarbeitung den Bereich mit dem geringsten Publikationsaufkommen in wissenschaftlichen Zeitschriften.[1]

Angesichts der Relevanz der drei Themengebiete KI, Cloud-Computing und Audiosignalverarbeitung soll in dieser Arbeit eine Auseinandersetzung mit ebendiesen Fachgebieten erfolgen. Das Ziel besteht darin, diese Technologien miteinander zu verknüpfen und anzuwenden, weshalb es in der vorliegenden Arbeit um die Entwicklung einer cloudbasierten KI-Lösung zur Klassifizierung von Audiosignalen geht.

[1] Vgl. Döbl u.a. (2018), S. 109 f.

1.1 Motivation

Neue Technologien bieten auch neue Chancen zur Steigerung des Unternehmenserfolgs, doch Österreich bleibt im internationalen Vergleich bei der Anwendung von KI-Methoden und Werkzeugen zurück.[2] Eine Studie des Bundesministeriums für Digitalisierung in Österreich zeigt, dass nur etwa 600 Unternehmen im Themengebiet der KI aktiv tätig sind. In dem für Österreich besonders relevanten Sektor des Maschinenbaus liegt die Anzahl der Unternehmen nur im Promillebereich, wobei die folgenden fünf Ursachen als Hindernisse für die Anwendung von KI angeführt werden:[3]

- Fachkräftemangel – Der Fachkräftemangel bezieht sich einerseits auf KI-Generalist*innen und andererseits auf KI-Spezialist*innen, die sich auf Themenkomplexe wie KNN oder die Entwicklung von KI-Software spezialisieren.
- Kosten – Diese entstehen sowohl aufgrund des notwendigen Ausgleiches des fehlenden Know-hows als auch aufgrund langer Entwicklungszeiten, die sich teils infolge der Anzahl von mehreren Entwicklungszyklen ergeben.
- Falsche Erwartungshaltung gegenüber KI – Der aktuelle Hype um das Themengebiet der KI führt oftmals zu enttäuschenden Ergebnissen, da die Erwartungshaltungen nicht erfüllt werden können.
- Ideenmangel – Unternehmen wissen aufgrund mangelnder Erfahrung und fehlender fachspezifischer Kenntnisse nicht, wie sie KI einsetzen können.
- Fehlende bzw. minderwertige Daten – Insbesondere kleine und mittlere Unternehmen (KMU) besitzen oftmals nicht die ausreichende Datenmenge in notwendiger Qualität, um KI sinnvoll anwenden zu können.

Des Weiteren sind im internationalen Bereich nur etwa 2 % der publizierten Arbeiten und Artikel dem Audioverarbeitungsbereich zuzuordnen. Dies liegt vor allem daran, dass sich die Forschung im Bereich der Audiosignalanalyse mittels KI noch am Anfang befindet.[4] Dennoch sind die möglichen Anwendungsgebiete im industriellen Umfeld bereits heute sehr vielfältig. So entwickelte beispielsweise das Fraunhofer-Institut erfolgreich eine Software zum Erkennen der Qualität von Schweißnähten mittels Audiosignalanalyse[5] oder Skoda ein App zur Diagnose von Betriebsgeräuschen von Kraftfahrzeugen.[6] Zudem beschäftigt sich das Themenfeld der prädiktiven Instandhaltung intensiv mit dem vorzeitigen Erkennen von Anomalien oder möglichen Fehlern, um Stillstandszeiten von Produktionsanlagen zu vermindern.[7] Bezogen auf den Audioverabeitungsbereich kann dies beispielsweise eine vorzeitige Erkennung von Lagerschäden mittels Soundanalyse sein.

[2] Vgl. Standard (2018), Online-Quelle [17.11.2022].

[3] Vgl. Prem/Ruhland (2019), S. 6 f.

[4] Vgl. Döbl u.a. (2018), S. 109 f.

[5] Vgl. Fraunhofer-Gesellschaft (2021), Online-Quelle [17.11.2022].

[6] Vgl. Skoda Auto DigiLab (o.J.), Online-Quelle [17.11.2022].

[7] Vgl. Fraunhofer-ITWM (o.J.), Online-Quelle [17.11.2022].

Im Fokus dieser Arbeit liegt die Anwendung der State-of-the-Art-Methoden und -Werkzeuge im Bereich der KI-basierten Audiosignalanalyse. Damit soll demonstriert werden, welche Hürden bei der Entwicklung einer Lösung zur Audiosignalklassifizierung beachtet werden müssen und welche Vorteile sich aufgrund der Nutzung von cloudbasierten KI-Plattformen für Unternehmen ergeben können.

1.2 Zielsetzung und Vorgehensweise

Das Ziel dieser Arbeit ist die Beantwortung folgender Fragestellung:

> Wie kann eine cloudbasierte KI-Lösung zur Klassifizierung von Audiosignalen entwickelt werden?

Konkret wird dabei der Entwicklungsprozess eines KI-Projektes unter Verwendung der State-of-the-Art-Methoden im Bereich der KI-basierten Audiosignalanalyse – von der Datenanalyse über die Datenvorverarbeitung, das Implementieren, Trainieren, Optimieren und Evaluieren eines Modells bis hin zur Bereitstellung eines Modells in einer Cloud-Umgebung – betrachtet. Das veröffentlichte Modell soll hierbei Klassifizierungsanfragen entgegennehmen und korrekt beantworten (z. B. Fehler/kein Fehler, Klasse A/Klasse B). Sämtliche verwendeten Hard- und Softwareressourcen werden in diesem Zusammenhang von Cloud-Anbietern zur Verfügung gestellt.

Außerhalb des Umfangs dieser Arbeit liegt die Datenerhebung der Audiodateien. Als Entwicklungsgrundlage wird daher der frei verfügbare Datensatz BirdCLEF2022 der Plattform Kaggle verwendet. Da es nur wenige frei verfügbare Audiodatensätze im Internet gibt, wird mit BirdClef2022 ein Datensatz gewählt, der Audioaufnahmen von Vogelgeräuschen enthält. Dieser kann zwar nicht direkt mit industriellen Anwendungen in Verbindung gebracht werden, jedoch zeichnet sich der Datensatz aufgrund seiner Komplexität aus. Dies bezieht sich insbesondere auf die geringe verfügbare Datenmenge und die minderwertige Aufnahmequalität der Dateien und soll somit die Anwendbarkeit von KI-Methoden auf schwierige Daten demonstrieren. Des Weiteren können aufgrund der Unstetigkeit der Signale und der unterschiedlichen Umgebungsbedingungen Parallelen zu industriellen Aufnahmen gezogen werden. Der Datensatz wird im weiteren Verlauf dieser Arbeit detaillierter erläutert (Abschnitt 5.1) und analysiert (Abschnitt 6.1).

Der gesamte soeben erläuterte Umfang dieser Arbeit wird in Abbildung 1 dargestellt, wobei der rote Rahmen die Grenzen symbolisiert.

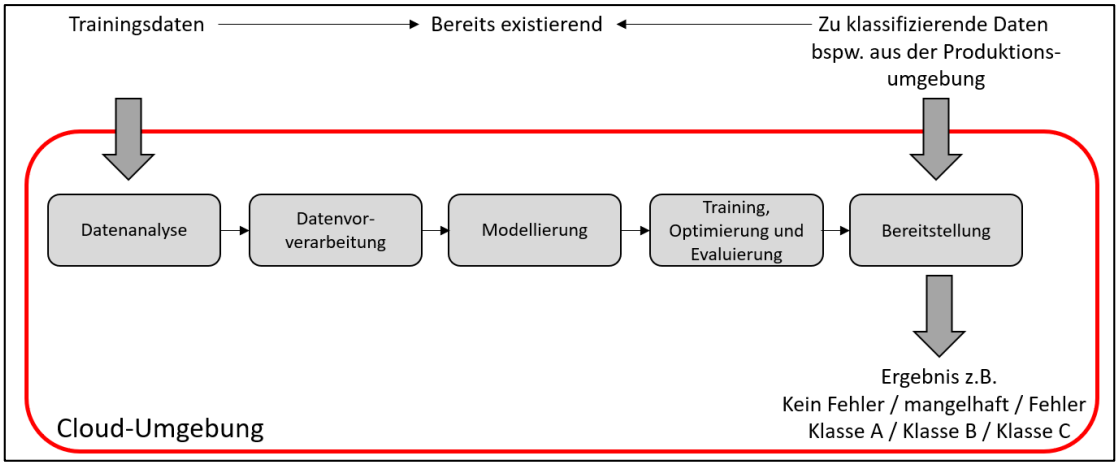


Abbildung 1: Abgrenzung der zu betrachtenden Elemente, Quelle: Eigene Darstellung.

Zur Umsetzung dieser Arbeit wird zunächst eine Literaturrecherche durchgeführt. Hierbei werden die zentralen Grundlagen und Methoden der Bereiche KI, Cloud-Computing sowie Audiosignalverarbeitung behandelt und alle notwendigen Schritte vorgestellt, die für die erfolgreiche Umsetzung eines KI-Projektes notwendig sind. Darauf aufbauend erfolgt die Auswahl einer geeigneten Cloud-KI-Plattform sowie weiterer geeigneter und benötigter Entwicklungsumgebungen. Ferner wird eine Anforderungsanalyse erstellt und ein Konzept entworfen. Anhand des erarbeiteten Konzeptes erfolgt eine softwaretechnische Implementierung der erläuterten Aufgabenstellung. Dabei werden KI-Modelle mithilfe der Programmiersprache Python erstellt, trainiert und optimiert. Im Anschluss wird ein umfangreicher Vergleich der trainierten Modelle durchgeführt, die auf den unterschiedlichen State-of-the-Art-Methoden basieren. Das beste Modell wird abschließend auf der gewählten Cloud-Plattform veröffentlicht.

1.3 Gliederung der Arbeit

Zu Beginn dieser Arbeit wird in Kapitel 2 das Themengebiet der KI im Allgemeinen betrachtet. Im Fokus sollen dabei das maschinelle Lernen (engl. Machine Learning, ML) sowie die für das weitere Verständnis benötigten theoretischen Grundlagen von KNN stehen. Zudem wird ein Vorgehensmodell zur Durchführung von ML-Projekten vorgestellt, das im weiteren Verlauf die Grundlage für die Konzeptionierung und Implementierung bildet.

Im Anschluss wird in Kapitel 3 das Gebiet der Audiosignalanalyse mittels ML-Methoden thematisiert. Konkret werden hier die State-of-the-Art-Methoden für die Klassifizierung von Audiosignalen erläutert.

Kapitel 4 bildet den letzten Teil der Literaturrecherche. Im Fokus steht an dieser Stelle das Themengebiet des Cloud-Computings. Neben den allgemeinen Vor- und Nachteilen von Cloud-Computing für Unternehmen geht es um die Auswahl einer geeigneten KI-Plattform, die als Umgebung für die zu entwickelnde Lösung dienen soll.

Auf Basis der durchgeführten Literaturrecherche wird in Kapitel 5 zunächst eine Anforderungsanalyse durchgeführt. Darauf aufbauend erfolgt der Entwurf eines Konzeptes für die zu entwickelnde Lösung. Des Weiteren werden die für die Implementierung benötigten und verwendeten Hard- und Softwarekomponenten erläutert.

In Kapitel 6 geht es um die Umsetzung des Konzeptes und der recherchierten Methoden. Dazu werden die einzelnen Phasen, die im Zuge eines ML-Projektes durchlaufen werden müssen, im Detail betrachtet und verschiedene Modellarchitekturen entworfen.

Anschließend werden die entwickelten Modellarchitekturen in Kapitel 7 trainiert und miteinander verglichen. Zudem wird das beste Modell mit verschiedenen Methoden in einem mehrstufigen Verfahren optimiert.

Im achten Kapitel dieser Arbeit wird das trainierte und optimierte Modell in die gewählte Cloud-Umgebung für den professionellen Einsatz integriert. Es wird gezeigt, wie Cloud-Services für das automatisierte Training von selbst entwickelten ML-Lösungen verwendet werden können. Des Weiteren wird das entwickelte Modell für Online-Vorhersagen auf der Cloud-Plattform veröffentlicht.

Zum Abschluss wird in Kapitel 9 ein Resümee über die gewonnenen Erkenntnisse gezogen und es wird ein Ausblick auf Verbesserungsmöglichkeiten gegeben.

2 KÜNSTLICHE INTELLIGENZ (KI)

Wenn es um die Frage der künftigen Entwicklung und Richtung von Unternehmen und ganzen Geschäftsbereichen geht, sind Begrifflichkeiten wie Industrie 4.0, digitale Transformation und KI elementar. Der KI wird dabei sowohl die Rolle als zukunftsweisendes Werkzeug aus dem Bereich der Informatik als auch die Rolle einer gefährlichen Technologie zur Entwicklung von selbstdenkenden und möglicherweise zerstörerischen Maschinen zugesprochen.[8] In diesem Kapitel wird ein Überblick über die zentralen Begriffe aus dem Bereich der KI gegeben, es wird erläutert, wie Maschinen selbstständig lernen, und der aktuelle Stand der Technik wird dargestellt. Zunächst erfolgt eine Begriffsdefinition, auf die im weiteren Verlauf aufgebaut wird.

2.1 Definition

Kein anderes Themengebiet der Informatik wird gesellschaftlich so kontrovers betrachtet wie die KI. Begründet werden kann dies einerseits durch die englische Wortwahl (Artificial Intelligence) und andererseits durch die unglückliche Übersetzung des Begriffes in die deutsche Sprache, denn durch die Bezeichnung ‚künstliche Intelligenz' ergibt sich unmittelbar die Fragestellung, ob Maschinen heutzutage oder in Zukunft wirklich denken können und was die daraus resultierenden Folgen sein könnten.[9] Eine konkrete Begriffsdefinition scheint jedoch unmöglich, da bereits die genaue Definition von Intelligenz durch Wissenschaften wie Neurologie, Biologie oder Psychologie vielfach scheiterte.[10] Daher wird für diese Arbeit die ingenieurtechnische, insbesondere die informationstechnologische Sichtweise als Einführung in das Themengebiet gewählt. Hiernach ist es nicht das Ziel, Systeme oder Maschinen mit dem Menschen gleichzusetzen oder diesen gar zu übertreffen, sondern menschliches Verhalten in gewissen Aspekten nachzuahmen. Jürgen Cleves und Uwe Lämmer definieren KI in ihrem gleichnamigen Buch als:

„Teilgebiet der Informatik, welches versucht, menschliche Vorgehensweisen der Problemlösung auf Computern nachzubilden, um auf diesem Wege neue oder effizientere Aufgabenlösungen zu erreichen."[11]

Diese Auslegung wird durch die in der Fachwelt gängige Differenzierung in schwache und starke KI gefestigt. Erstere beschäftigt sich vor allem mit dem Lösen von speziellen Problemen in von Menschen definierten Aufgabenbereichen. Dazu werden Konzepte wie künstliche neuronale Netzwerke (KNN), Machine Learning (ML) und Deep Learning (DL), die maßgeblich von der Verfügbarkeit und Qualität von Daten abhängig sind, angewendet, wohingegen eine starke bzw. allgemeine KI einen theoretisch menschenähnlichen Bewusstseinszustand erreicht, selbstständig verschiedene Problemstellungen erkennt und auf diese entsprechend reagiert.[12]

[8] Vgl. Wennker (2020), S. 1.

[9] Vgl. Cleve/Lämmel (2012), S. 13.

[10] Vgl. Wittpahl (2019), S. 23.

[11] Cleve/Lämmel (2012), S. 14.

[12] Vgl. Wittpahl (2019), S. 222.

Wird der aktuelle Stand der Forschung betrachtet, handelt es sich dabei allerdings um eine rein medial inszenierte Darstellung, da die Frage, ob und wann eine starke KI erreicht werden kann, nach wie vor ungeklärt ist und folglich alle aktuellen Lösungen einer schwachen KI entsprechen.[13]

Als KI wird somit im Allgemeinen ein Teilbereich der Informatik verstanden, der sich mit der Automatisation von intelligentem Verhalten beschäftigt. Um die Autonomie von Maschinen und Systemen zu erhöhen, werden im Bereich der KI wesentliche Methoden und Werkzeuge der Informatik zusammengefasst. Der Begriff Autonomie beschreibt hierbei die Fähigkeit von Maschinen und Systemen, selbstständig auf sich ändernde Bedingungen und Probleme zu reagieren,[14] weshalb im folgenden Abschnitt das ML thematisiert wird.

2.2 Wie Maschinen und Systeme lernen – Machine Learning (ML)

Die KI lässt sich in weitere Teilgebiete untergliedern. Beispielsweise gibt es unterschiedliche Konzepte und Methoden für Aufgabenstellungen wie Spracherkennung, Bilderkennung oder autonomes Fahren. Auf all diese Gebiete im Detail einzugehen, würde jedoch den Umfang der Arbeit überschreiten. Vielmehr soll zunächst die zentrale Fragestellung geklärt werden, wie Maschinen erlernen können, selbstständig Probleme zu lösen. Die zwei relevantesten Konzepte sind hierbei ML und DL. Abbildung 2 visualisiert die Einordnung dieser beiden Verfahren im Themengebiet der KI. Demnach stellt DL wiederum ein Teilgebiet von ML dar. Gemein ist beiden, dass sie in der Lage sind, aus vorhandenen Daten Muster und Zusammenhänge zu erkennen, aus diesen zu lernen und das erlernte Wissen auf neue Daten anzuwenden.[15]

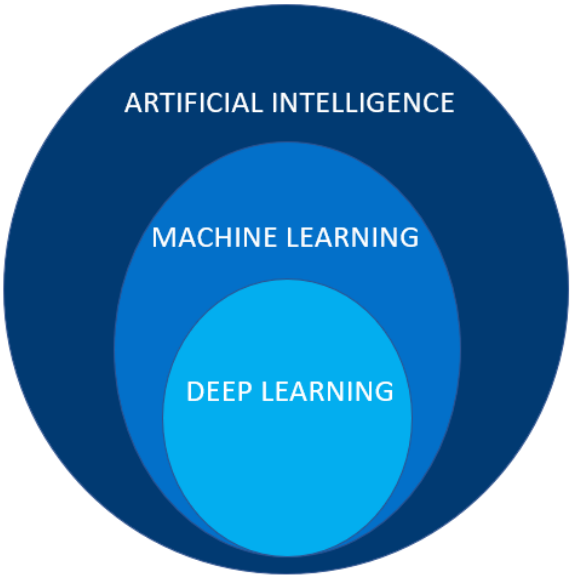


Abbildung 2: Einordnung von ML und DL im Bereich der KI, Quelle: In Anlehnung an Wuttke (2021), S. 56.

[13] Vgl. Wennker (2020), S. 7.

[14] Vgl. Elmers/Niggemann (2022), S. 13.

[15] Vgl. Wuttke (2021), S. 56.

Die grundlegende Idee des ML ist es, Programme zu erzeugen, die sich automatisch anhand von Erfahrungen verbessern.[16] Wird zwischen der traditionellen Softwareentwicklung und ML differenziert, so können folgende Unterschiede beobachtet werden (siehe Abbildung 3):

Im Bereich der Softwareentwicklung wird der Maschine (Computer) eine eindeutige Vorgehensweise durch Programmcode vorgegeben. Die Maschine arbeitet strikt nach den definierten Regeln im ‚Wenn-Dann-Modus' und kann nicht auf Regelabweichungen reagieren. Im Unterschied dazu werden beim ML keine klaren Regeln definiert. Dem Computer werden Daten (meist mit einer definierten Zielvariable) überreicht, aus denen er durch Algorithmen Muster erlernt. Die erlernten Muster entsprechen dabei einem selbst erstellten Programmcode. Die Software wird somit durch den verwendeten Algorithmus selbst erstellt und kann anschließend auf neue Daten, beispielsweise für Vorhersagen, angewendet werden.[17] Für unterschiedliche Aufgabengebiete werden verschiedene Algorithmen herangezogen. Typische Verfahren sind beispielsweise Entscheidungsbäume oder Regression. Auf die für die Audiosignalverarbeitung zentralen Verfahren wird im weiteren Verlauf im Detail eingegangen.

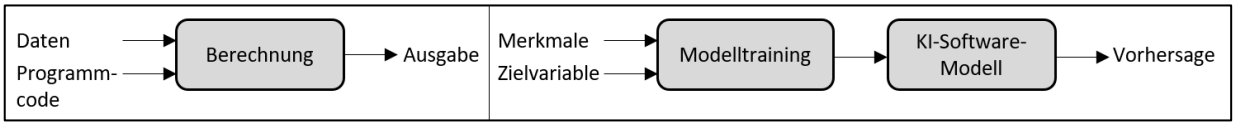


Abbildung 3: Unterschied zwischen traditioneller Softwareentwicklung und ML, Quelle: In Anlehnung an Wuttke (2021), S. 77 f.

Machine Learning unterstützt vor allem bei der effizienten Analyse von großen heterogenen Datenmengen, denn die menschlichen Fähigkeiten sind nicht dazu geeignet, die schiere Menge und Vielfalt an Daten, die unter anderem durch die weltweite Vernetzung und Nutzung von internetbasierenden Applikationen entstanden sind, in Echtzeit zu analysieren. Ermöglicht wird dies durch die hohen Rechenkapazitäten, die aktuell zur Verfügung stehen.[18] Die generelle Arbeitsweise von ML-Anwendungen und das Einsatzgebiet verdeutlichen hierbei die Relevanz der Daten. Vielfach entfallen bis zu 80 % der Zeit eines Projektes auf die Vorverarbeitung ebendieser. Beim aufwendigen Sammeln der für das Projekt und die Aufgabe geeigneten Daten muss zudem darauf geachtet werden, dass diese gewissen Qualitätsmerkmalen entsprechen. Hier gilt das Garbage-In-Garbage-Out-Prinzip, da eine qualitativ minderwertige Grundlage unweigerlich zu einem unerwarteten und insbesondere nicht erwünschten Ergebnis führt. Ferner ist es notwendig, die Daten zu codieren, von Redundanzen zu befreien und in eine für die Weiterverarbeitung geeignete Form zu transformieren.[19] Bei der Erstellung von ML-Lösungen sind Trainings- sowie Testdaten essenziell. Trainingsdaten sind ein Satz von vorklassifizierten Daten, die das zu erlernende und zu extrahierende Wissen enthalten. Hier gilt: Je mehr Daten vorhanden sind, desto besser kann das Modell lernen. Anschließend werden die Testdaten auf das trainierte Modell angewendet. Dies dient insbesondere zur Evaluierung, ob die erlernten Fähigkeiten auf neue Daten generalisiert werden können.

[16] Vgl. Mitchell (1997), S. XV.

[17] Vgl. Wuttke (2021), S. 77.

[18] Vgl. Awad/Khanna (2015), S. 5.

[19] Vgl. Wennker (2020), S. 10.

Sowohl Test- als auch Trainingsdatensätze müssen eine repräsentative Stichprobe für das zu erlernende Ziel darstellen.[20] Nach erfolgreichem Testergebnis lässt sich das Modell auf die eigentliche Aufgabe anwenden. Dazu können untere anderem die folgenden fünf zählen:[21]

- Prädiktion – z. B. Umsatzprognosen oder Vorhersage von Gruppenzugehörigkeiten (Klassifikation),
- Wahrscheinlichkeitsberechnung – z. B. Kaufwahrscheinlichkeit,
- Clustering – Gruppierung von Objekten, z. B. Kundensegmentierung,
- Korrelationsanalysen sowie
- Dimensionsreduktion.

2.2.1 Deep Learning (DL)

Wie zu Beginn von Kapitel 2.2 erwähnt, ist DL ein spezieller Bereich von ML. Bei klassischen ML-Verfahren (statistische Verfahren, basierend auf mathematischer Logik) ist es notwendig, dass strukturierte und definierte Daten vorliegen. Die Definition der Merkmale (Features), also jener Eingangsdaten, von denen der Algorithmus lernt (z. B. Anzahl der Ohren bei Katzenbildern), erfolgt dabei manuell von Fachexpert*innen (siehe Abbildung 4 oben).[22] Beispiele hierfür können Excel-Tabellen oder Datenbanksätze sein. Spalten- und Reihenwerte sind hier eindeutig und können somit ideal für den Lernprozess verwendet werden.[23] Jedoch ist eine manuelle Festlegung der Formeln für gute Features teils mit einem sehr hohen Aufwand verbunden oder gar unmöglich. Als Beispiel dient die Objektklassifizierung bei Bildern. Liegt etwa ein RGB-Bild mit 10 Millionen Pixel vor, entspricht das einem Eingangsvektor mit einer Länge von 10 Millionen. Damit annehmbare Rechenzeiten eingehalten werden können, müssen die Eingabevektoren zunächst auf kürzere Merkmalsvektoren reduziert werden. Sollten auf einem Bild unterschiedliche Objekte mit verschiedenen Merkmalen identifiziert werden, scheint dies angesichts der Unstrukturiertheit der Daten und der Menge an Eingangspunkten zunächst als unlösbare Aufgabe.[24] Als Lösung hierfür eignen sich DL-Verfahren. Bei diesen erfolgt die Feature-Extraktion automatisiert (siehe Abbildung 4 unten).

[20] Vgl. Ertel (2016), S. 195.

[21] Vgl. Wuttke (2021), S. 59.

[22] Vgl. Ertel (2016), S. 300.

[23] Vgl. Wuttke 1 (o.J.), Online-Quelle [07.08.2022].

[24] Vgl. Ertel (2016), S. 300.

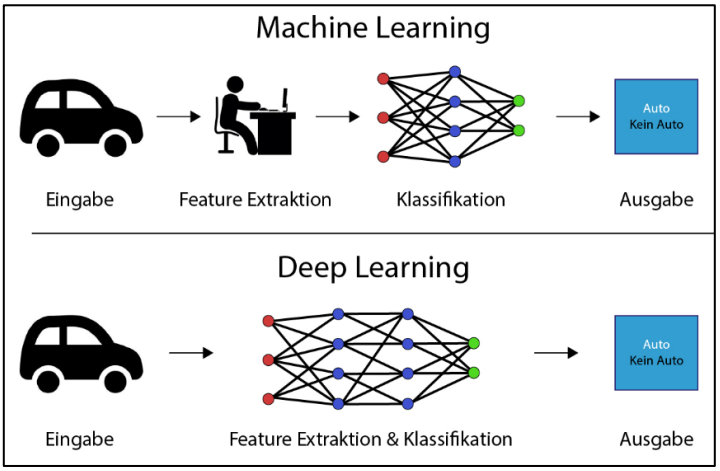


Abbildung 4: Differenzierung ML und DL, Quelle: Wuttke 1 (o.J.), Online-Quelle [07.08.2022].

Dazu werden Lernmodelle verwendet, die dem Verhalten des Menschen nachempfunden sind und im Allgemeinen als KNN bezeichnet werden. Im Sinne des DL wird insbesondere von tiefen KNNs gesprochen. Künstliche neuronale Netzwerke bestehen aus einer Vielzahl vereinfachter künstlich erzeugter Neuronen, deren Vorbild die Struktur biologischer neuronaler Netzwerke ist. Die Anzahl der Schichten und Tiefe der Netzwerke kann dabei variieren und ist von ML-Expert*innen zu definieren. Wird die Eingabe (z. B. ein Bild) durch die Lagen des KNNs gesendet, so erfolgt ein iterativer Prozess, bei dem die Daten verarbeitet und die Identifikationsmerkmale (z. B. für die Objektklassifizierung) erkannt werden,[25] wobei jede Lage des Netzwerks ein spezifisches Merkmal repräsentiert. Je höher die Lage, desto komplexer die Features. Im Falle der Objekterkennung können die höheren Schichten beispielsweise Merkmale wie die Existenz eines Gesichtes abbilden. Hingegen repräsentieren tiefere Schichten einfache Eigenschaften wie Kanten oder Linien.[26] Ein wesentlicher Nachteil dieser Vorgehensweise ist allerdings, dass für die automatisierte Extraktion der Merkmale eine deutlich größere Datenmenge als bei der manuellen benötigt wird.[27] Da KNNs für diese Arbeit von zentraler Bedeutung sind, wird auf diese in Kapitel 2.3 im Detail eingegangen.

2.2.2 Arten des Lernens

Die Mitwirkung des Menschen beim Lernprozess von ML-Lösungen hängt im Wesentlichen vom Aufgabengebiet und den gewählten Algorithmen ab. Die Art und Weise, wie die Algorithmen von den Daten lernen, lässt sich in mehrere Kategorien klassifizieren. Auf die vier relevantesten wird im Folgenden eingegangen:

Überwachtes Lernen

Beim überwachten Lernen werden dem System zunächst Trainingsdatensätze übergeben, die die konkrete Zielvariable enthalten.

[25] Vgl. Wuttke 1 (o.J.), Online-Quelle [07.08.2022].

[26] Vgl. Ertel (2016), S. 301.

[27] Vgl. Wuttke 1 (o.J.), Online-Quelle [07.08.2022].

Die Datensätze beinhalten somit bereits die Antwort auf das zu lösende Problem. Im Falle einer Klassifizierungsaufgabe ist das die korrekte Kategoriezuordnung des Datensatzes durch entsprechende Beschriftung.[28] Dieser Vorgang wird als Labeling bezeichnet. Da die meisten ML-Lösungen nach wie vor auf dem Prinzip des überwachten Lernens basieren, nimmt das Datenlabeling eine bedeutende, aber auch zeitaufwendige Rolle ein. In größeren Unternehmen werden dafür eigens organisierte Teams beschäftigt.[29] Aus den gekennzeichneten Traningsdatensätzen wird durch die verwendeten Lernalgorithmen eine Modellfunktion extrahiert, die versucht, eine allgemeine Beziehung zwischen den Merkmalsvektoren (Eingabe) und der Zielvariable (Ausgabe) herzustellen.[30] Im Anschluss wird die Qualität des Ergebnisses der Funktion bewertet. Der gesamte Vorgang ist iterativ, wobei die Erfahrungswerte aus jedem Durchlauf miteinbezogen werden. Der Prozess wird so lange wiederholt, bis ein qualitativ hochwertiges Ergebnis erreicht wird und das Modell auf unbekannte Daten angewendet werden kann. Generell wird beim überwachten Lernen zwischen den beiden Aufgabentypen Klassifikation und Regression unterschieden.[31]

Unüberwachtes Lernen

Beim unüberwachten Lernen ist kein Labeling der Daten notwendig, da keine konkreten Zielvariablen vorgegeben werden. Ferner wird kein zusätzliches Expert*innen-Wissen von außen benötigt, weshalb ein geringerer personeller Aufwand als beim überwachten Lernen erforderlich ist.[32] Da kein definiertes Ziel vorgegeben ist und somit keine Grenzen gesetzt sind, können Algorithmen vollkommen eigenständig Zusammenhänge und Muster jeglicher Art erkennen. Dies unterstützt beispielsweise beim Finden von neuen Kriterien für Kategorisierungsaufgaben. Anwendung findet das unüberwachte Lernen bei Aufgaben wie Clustering, Dimensionsreduktion und Assoziationsanalysen.[33]

Teilüberwachtes Lernen

Das teilüberwachte Lernen ist eine Kombination aus den beiden bereits erläuterten Lernverfahren. Eingesetzt wird das teilüberwachte Lernen in denselben Bereichen wie das überwachte Lernen (Klassifikation, Regression). Die wesentlichen Unterschiede ergeben sich aufgrund des zur Verfügung stehenden Datensatzes. Nur ein geringer Teil der Gesamtdatenmenge ist mit einer Zielvariable versehen.[34] Bei den restlichen Daten wird davon ausgegangen, dass nah beieinander liegende Datenpunkte eine identische Kennzeichnung besitzen. Verglichen mit dem überwachten Lernen ergeben sich hier Vorteile hinsichtlich der Reduktion des Arbeitsaufwandes.[35]

[28] Vgl. Plaue (2021), S. 190.

[29] Vgl. Huyen (2022), S. 147.

[30] Vgl. Awad/Khanna (2015), S. 9.

[31] Vgl. Brunel (o.J.), Online-Quelle [07.08.2022].

[32] Vgl. Wennker (2020), S. 16.

[33] Vgl. Brunel (o.J.), Online-Quelle [07.08.2022].

[34] Vgl. Wuttke (2021), S. 68.

[35] Vgl. Wennker (2020), S. 14.

Verstärkendes Lernen

Das verstärkende Lernen ist dem menschlichen Lernverhalten nachempfunden. Dem System wird hierbei ein konkretes Endziel vorgegeben. Jedoch müssen die Lösungswege durch Interaktion mit der Umgebung sowie ein Trial- und Error-Verfahren autonom ermittelt werden, wobei ein Belohnungssystem gute Lösungsansätze fördert und schlechte bestraft. Anders als bei den bisher erläuterten Lernverfahren stehen hier keine Trainingsdaten zur Verfügung.[36] Aufgrund der Gegebenheit, dass verstärkende Lernsysteme selbstständig Problemlösungen für unterschiedliche Aufgabenstellungen erarbeiten, gilt diese Methode als jene mit dem größten Potential zum Erreichen einer starken KI.[37]

2.2.3 Klassifikation

Wie bereits in Abschnitt 2.2 erläutert, lassen sich entwickelte ML-Modelle für verschiedene Anwendungszwecke einsetzen. Das Ziel dieser Arbeit ist es, eine Klassifikationsaufgabe für Audiodateien durchzuführen, weshalb die Klassifikation im Folgenden genauer betrachtet wird.

Bei der Klassifikation handelt es sich um ein Verfahren, das es ermöglicht, aus Daten eine Kategorie (Klasse bzw. Typ) zu erlernen und abzuleiten. Da die Zielklassen bereits vorab definiert werden müssen, entspricht die Klassifikation einer Methode des überwachten oder teilüberwachten Lernens. Obwohl dem Lernalgorithmus eine definierte Zielvariable vorgegeben wird, ist die Ausgabe eine Wahrscheinlichkeit bzw. eine Neigung zur betreffenden Klasse. Die Klassenzuordnung erfolgt also erst im späteren Verlauf durch die Festlegung eines Grenzwertes. Das Verfahren kann somit als Vorhersage von Gruppenzugehörigkeiten betrachtet werden. Geeignete Algorithmen sind beispielsweise KNN, k-Neirest-Neighbour oder die Support-Vector-Machine.[38]

Bei der Klassifikation kann zwischen den folgenden zwei Arten unterschieden werden:[39]

- Zwei-Klassen-Klassifizierung – Vorhersagen zwischen zwei Kategorien. Es gibt nur zwei Antwortmöglichkeiten (z. B. A oder B).
- Multi-Klassen-Klassifizierung – Vorhersagen zwischen mehreren Kategorien. Es werden Fragen mit mehreren Antwortmöglichkeiten beantwortet (z. B. A oder B oder C oder D).

Die Auswahl geeigneter Algorithmen hängt dabei wesentlich von der Art der Aufgabenstellung ab. Zur Veranschaulichung sind in Abbildung 5 zwei gängige Verfahren dargestellt. Im einfachsten Fall (links) ist die Datenmenge linear separabel. Für eine solche Aufgabenstellung kann ein Perzeptron, als linearer Klassifizierer, verwendet werden. Der mathematische Hintergrund ist gleich der Stufenfunktion als Aktivierungsfunktion bei künstlichen Neuronen, auf die in Abschnitt 2.3 eingegangen wird. Für komplexere Aufgabenstellungen kann ein k-Neirest-Neighbour-Verfahren angewendet werden. Die Idee besteht darin, dass nah beieinander liegende Datenpunkte derselben Klasse entsprechen müssen.

[36] Vgl. Gentsch (2019), S. 38.

[37] Vgl. Wuttke 2 (o.J.), Online-Quelle [07.08.2022].

[38] Vgl. Kersting/Lampert/Rothkopf (2019), S. 45 f.

[39] Vgl. Microsoft Corporation 1 (2022), Online-Quelle [07.08.2022].

Anders als beim Perzeptron ist die Trennlinie lediglich imaginär. Diese kann aber anhand eines Voronoi-Diagramms, in dem jeder Datenpunkt von einem konvexen Polynom umgeben ist, visualisiert werden (Abbildung 5 Mitte).[40] Nicht zu verwechseln ist die Klassifizierung mit der Clusteranalyse (Abbildung 5 rechts). Bei der Clusteranalyse sind die Zielklassen nicht vordefiniert. Die Algorithmen finden die Struktur und Häufungen selbstständig.[41] Die Clusteranalyse lässt sich somit dem unüberwachten Lernen zuordnen.

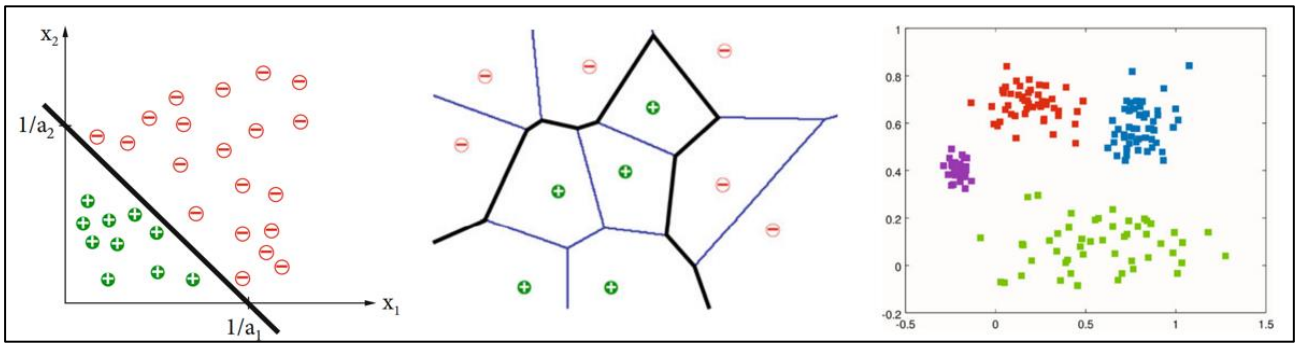


Abbildung 5: Klassifikation mit linearer Trenngerade (links), Klassifikation mittels k-Neirest-Neighbour-Verfahrens (Mitte) und Clusteranalyse (rechts), Quelle: Ertel (2016), S. 200 (links), S. 208 (Mitte), S. 250 (rechts).

2.3 Künstliche neuronale Netze (KNN)

Die menschliche Intelligenz beschreibt die Fähigkeit, sich an unbekannte Umweltbedingungen anpassen zu können und motorische sowie intellektuelle Fertigkeiten zu erlernen. Basis dafür bildet ein komplexes, adaptives Netzwerk, das aus bis zu 100 Milliarden Nervenzellen (Neuronen) besteht. Abbildung 6 (links) zeigt die vereinfachte Struktur eines einzelnen biologischen Neurons. Über die Dendriten werden Erregungen anderer Neuronen wahrgenommen und an den Zellkern weitergeleitet. Dieser stellt eine Art Speicher, ähnlich einem elektrischen Kondensator, dar und wird durch Spannungsimpulse anderer Neuronen aufgeladen. Wird ein bestimmter Schwellwert überschritten, so feuert das geladene Neuron. Das bedeutet, dass ein Spannungsimpuls über das Axon und die Synapsen an die nächsten Neuronen weitergeleitet wird.[42]

2.3.1 Das künstliche Neuron

Auf dem Gebiet der KNN wird versucht, ebendiesen Vorgang mathematisch, digital oder durch Simulation nachzubilden. Nach dem Diskretisieren der Zeitskala lässt sich das künstliche Neuron *j* (siehe Abbildung 6 rechts) mathematisch mit den folgenden fünf Funktionen bzw. Bestandteilen beschreiben:[43]

- Eingangsinformationen – Setzen sich aus direkten Eingabewerten o_1 ... o_k und Gewichtungen w_{1j} ... w_{kj} zusammen. Je nach Gewichtung ist der Einfluss eines Eingabewertes auf die Berechnung der späteren Aktivierungsfunktion größer oder kleiner.

[40] Vgl. Ertel (2016), S.199 - 208.

[41] Vgl. Ertel (2016), S. 245.

[42] Vgl. Ertel (2016), S. 265 f.

[43] Vgl. Cleve/Lämmel (2012), S. 189 ff.

- Propagierungsfunktion: $net_j = \sum_k w_{kj} \cdot o_k$ – Verknüpft Eingabewerte mit Gewichtungen und fasst Eingangsinformationen durch Aufsummieren zusammen.
- Aktivierungsfunktion: $f_{act} \rightarrow a_j = f_{act}(net_j, \theta_j)$ – Ermittelt anhand der Propagierungsfunktion und unter Berücksichtigung des Schwellwertes (Bias) θ_j die aktuelle Aktivität bzw. den aktuellen Zustand a_j des Neurons.
- Ausgabefunktion: $f_{out} \rightarrow o_j = f_{out}(a_j)$ – Bestimmt aufgrund des Zustandes den aktuellen Ausgabewert o_j, der an verknüpfte Neuronen weitergeleitet wird und dort als Eingabewert dient.
- Lokaler Speicher – Speichert die Informationen des aktuellen Zustandes a_j und enthält den Schwellwert θ_j.

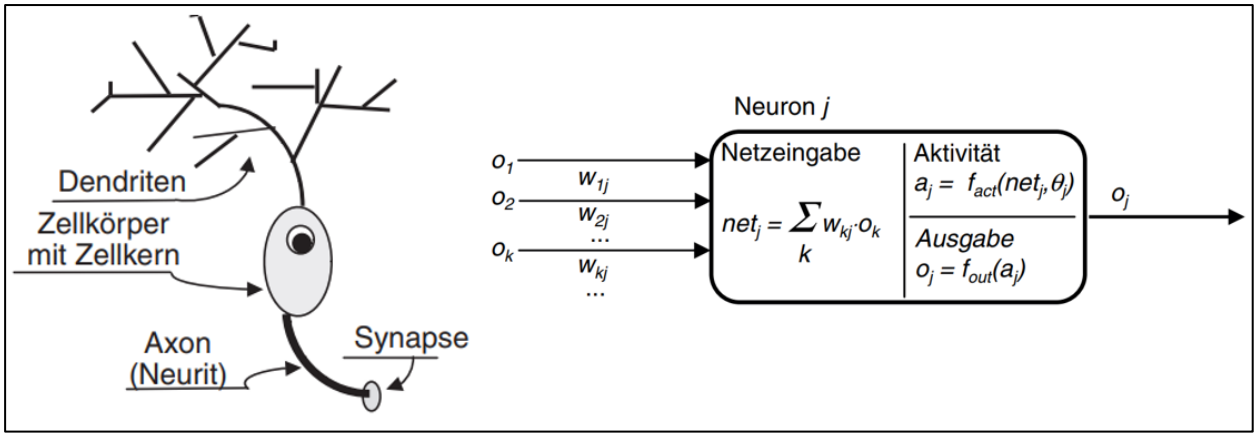


Abbildung 6: Biologisches Neuron (links) und mathematisches Modell (rechts). Quelle: Cleve/Lämmel (2012), S. 189 (links), S. 190 (rechts).

Es wird also aus den gewichteten Eingabewerten eine Summe gebildet, auf die eine Aktivierungsfunktion angewendet wird. Im Anschluss wird das Ergebnis über die synaptischen Gewichte an nachfolgende Neuronen weitergeleitet. Für die Auswahl der Aktivierungsfunktion existieren verschiedene Möglichkeiten. Im einfachsten linearen Fall $f_{act}(net_j) = net_j$ wird von einer Identität gesprochen, wobei das Neuron die gewichtete Summe berechnet und eins zu eins weitergibt. Jedoch entstehen dabei häufig Probleme bei der Konvergenz bezogen auf die neuronale Dynamik, da $f_{act}(net_j) = net_j$ nicht beschränkt ist.[44]

Weitere in der Literatur oftmals erwähnte und in der Praxis eingesetzte Aktivierungsfunktionen sind die folgenden vier:[45]

- Heavisidesche Stufenfunktion: $f_{act}(net_j) = \begin{cases} 0 & falls\ net_j < \theta_j \\ 1 & sonst \end{cases}$,
- Sigmoid-Funktion: $f_{act}(net_j) = \frac{1}{1+e^{-net_j}}$,
- Tangens hyperbolicus (Tanh): $f_{act}(net_j) = \tanh(net_j) = 1 - \frac{2}{e^{2net_j}+1}$ sowie
- Recitified Linear Unit (ReLU): $f_{act}(net_j) = \max(0, net_j)$.

[44] Vgl. Ertel (2016), S. 269.

[45] Vgl. Ertel (2016), S. 269 f; Vgl. Sonnet (2022), S. 29 ff; Vgl. Buduma (2017), S. 13 f.

Die Stufenfunktion eignet sich speziell für binäre Neuronen, da diese ohnehin nur den Wert null oder eins annehmen können. Ein fließender Übergang wird hingegen mit der Sigmoid-Funktion und der Tanh-Funktion erreicht. Beide weisen einen S-Kurvenverlauf auf, wobei sich die Zustände sigmoider Neuronen im Intervall $[0,1]$ und jene der Tanh-Funktionen im Intervall [-1,1] bewegen. Im Bereich von DL hat sich ReLU etabliert. Bei negativer Netzeingabe ist die Ausgabe null. Erst bei positiver Eingabe verhält sich die Funktion linear. Werte bei ReLU liegen im Intervall $[0, \infty)$. Die Auswahl einer geeigneten Aktivierungsfunktion für einen bestimmten Anwendungsfall ist vorab nicht immer eindeutig möglich, weshalb es sich empfiehlt, beim Training verschiedene Funktionen zu testen.[46]

Neben den in Abschnitt 2.2.2 erläuterten allgemeinen Konzepten beim Training von ML- und DL-Algorithmen werden zum Trainieren von KNN unterschiedliche Strategien angewendet, die entweder einzeln oder kombiniert Anwendung finden. Dazu zählen beispielsweise das Etablieren neuer oder das Löschen bestehender Verbindungen (Kanten), das Hinzufügen oder das Löschen von Neuronen, das Variieren der Aktivierungsfunktion, die Abänderung der Gewichtungen sowie die Veränderung des Schwellenwertes. Die beiden letztgenannten Strategien werden in der Praxis am häufigsten angewendet.[47]

2.3.2 Mehrschichtige Netze

Einzelne Neuronen sind nicht in der Lage, komplexe Lernprobleme oder Aufgabenstellungen zu lösen. Deshalb sind die Neuronen in der menschlichen Großhirnrinde (Cortex), also jenem Teil, der maßgeblich für die humane Intelligenz verantwortlich ist, in Schichten organisiert. Analog dazu wird das theoretische Modell von KNN wie in Abbildung 7 dargestellt. Dieses gliedert sich in drei Arten von Schichten:[48]

- eine Eingabeschicht,
- eine oder mehrere verdeckte Schicht(en) sowie
- eine Ausgabeschicht.

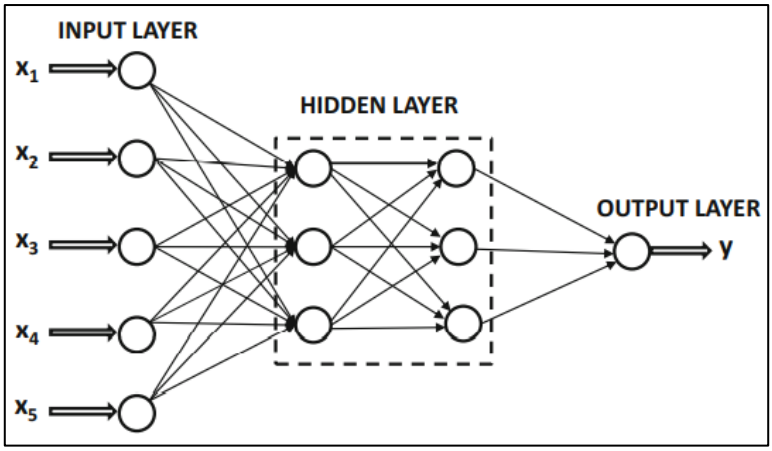


Abbildung 7: Struktur eines KNN, Quelle: Charu (2018), S. 32.

[46] Vgl. Sonnet (2022), S. 29 ff.

[47] Vgl. Sonnet (2022), S. 45 f.

[48] Vgl. Buduma (2017), S. 9 ff.

Jede Schicht besteht aus einer gewissen Anzahl von künstlichen Neuronen, die über Kanten miteinander verbunden sind. Die eingegebenen Daten (z. B. eine Audiodatei) werden in der Eingabeschicht von den Input-Neuronen aufgenommen sowie verarbeitet und an die nächste Schicht weitergeleitet. Die Hidden Layers sind maßgeblich für die Lernfähigkeit des KNN verantwortlich und können aus einer beliebigen Anzahl von Schichten bestehen. Hier werden die Neuronen bis zur Ausgabeschicht Schicht für Schicht erneut gewichtet. Der genaue Prozess ist aufgrund der Komplexität meist nicht nachvollziehbar, weshalb dieser Bereich des KNN als Blackbox betrachtet werden kann. Die Ausgabeschicht enthält die getroffenen Entscheidungen als Informationsfluss und bildet die Grundlage für die Bewertung der Ergebnisse.[49] Die Anzahl der Neuronen in der Ein- und Ausgabeschicht richtet sich nach der Struktur der zu erlernenden Daten. Dahingegen gibt es bei den Hidden Layers keine festgelegte Vorgehensweise bei der Definition der Anzahl der Schichten und Neuronen. Die Architektur und der Aufbau des Netzes müssen entsprechend der Komplexität der Daten definiert werden. Generell ist eine Überanpassung (engl. Overfitting), also eine zu genaue Anpassung des Netzes an die Trainingsdaten zu vermeiden, da im Anschluss eine Generalisierung auf Testdaten nur bedingt möglich ist und somit keine optimalen Ergebnisse erreicht werden können.[50]

Des Weiteren kann bei KNN zwischen Feedforward- und rekurrenten Netzwerken unterschieden werden. Bei Feedforward-Netzwerken sind Neuronen jeweils mit Neuronen der nächsten Schicht verbunden. Bei rekurrenten Netzwerken können Neuronen auch auf dieselbe oder die vorgelagerte Schicht zugreifen.[51] Die detaillierten Unterschiede und gängigen Methoden der beiden Architekturen werden in Abschnitt 3.2.1 und Abschnitt 3.2.2 behandelt.

2.4 Vorgehensweise beim Erstellen von ML-Lösungen

Nachdem die zentrale Fragestellung dieses Kapitels, wie Maschinen lernen können, umfassend behandelt wurde, wird in diesem Unterkapitel auf die Vorgehensweise beim Erstellen von ML-Lösungen eingegangen. Dazu erfolgt zunächst eine Erläuterung des seit 1999 etablierten Phasenmodells Crisp-DM. Ferner wird auf die Relevanz von KI-Plattformen und ML-Pipelines bei der Durchführung von KI-Projekten eingegangen.

Das Phasenmodell Crisp-DM beschränkt den Entwicklungsprozess von ML-Lösungen nicht nur auf die Auswahl geeigneter Algorithmen sowie das anschließende Training von Modellen, sondern beschreibt diesen ganzheitlich. Es wird zwischen den folgenden sechs Phasen differenziert:[52]

[49] Vgl. Wuttke 3 (o.J.), Online-Quelle [07.08.2022].

[50] Vgl. Sonnet (2022), S. 69.

[51] Vgl. Sonnet (2022), S. 69.

[52] Vgl. Chapman u.a. (1999), S. 13 f.

- Business-Verständnis – Der Schwerpunkt dieser ersten Phase liegt auf der Problemdefinition und der Festlegung sowie dem Verständnis der Anforderungen und Projektziele aus Unternehmensperspektive. Dabei sollten die Definitionen so konkret formuliert sein, dass diese in ein Datenproblem überführt werden können.
- Datenverständnis – In dieser Phase werden zunächst erstmals Daten erhoben und gesammelt, die im Anschluss beschrieben, analysiert und visualisiert sowie abschließend hinsichtlich der Qualität bewertet werden.
- Datenvorbereitung – Bei der Datenvorbereitung handelt es sich um die Aufbereitung der anfänglichen Rohdaten. Typische Aufgaben dieser Phase sind beispielsweise die Auswahl von Trainingsdatensätzen oder das Bereinigen und Transformieren von Daten.
- Modellierung – Diese Phase umfasst die Auswahl und Anwendung der Daten entsprechend geeigneten Modellierungstechniken sowie die Kalibrierung der zugehörigen Parameter (Training).
- Evaluation – In diesem Abschnitt werden die erstellten Modelle hinsichtlich der festgelegten Anforderungen und Ziele bewertet und es wird evaluiert, ob diese erfüllt werden.
- Bereitstellung – Abschließend werden die entwickelten Modelle dem Kunden präsentiert, zur Verfügung gestellt und in das Unternehmenssystem integriert. Letzteres erfolgt meist vom Kunden selbst und kann sich bzgl. des Komplexitätsgrades wesentlich unterscheiden.

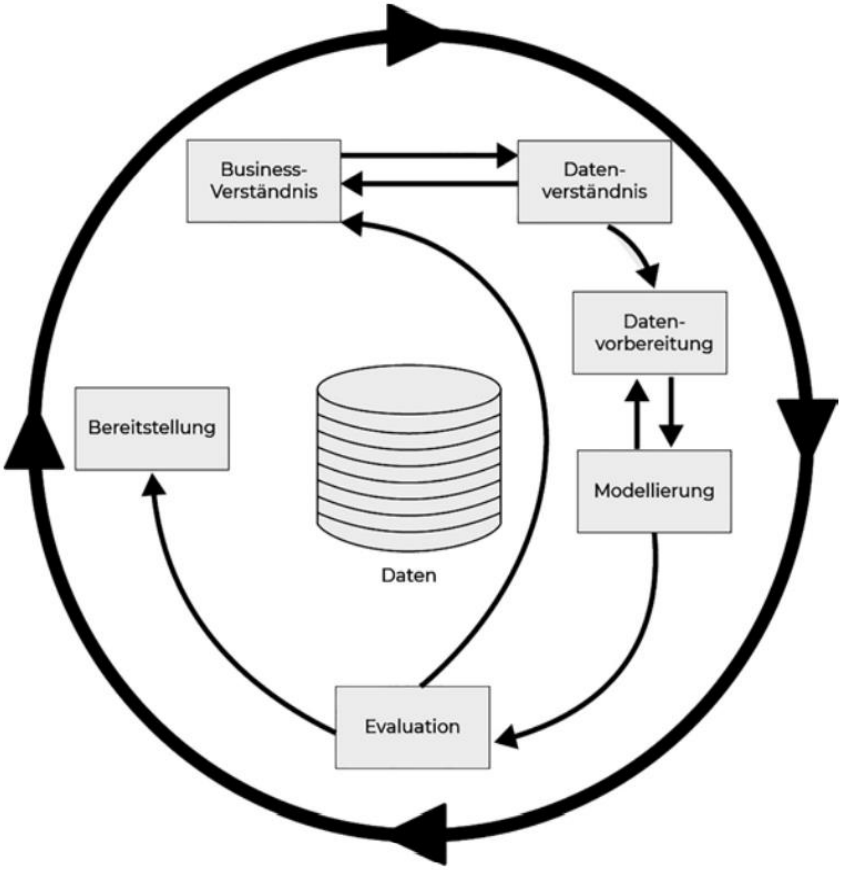


Abbildung 8: Crisp-DM Phasenmodell, Quelle: Wuttke (2021), S. 70.

2.4.1 KI-Plattform

An einem ML-Projekt sind meist Mitarbeiter*innen aus unterschiedlichen Fachbereichen beteiligt. Neben den Data-Engineers und Data-Scientists, die für die Datenerfassung, Aufbereitung, Modellierung und Analyse zuständig sind, gibt es ML-Engineers und DevOps-Engineers, die die Integration der entwickelten Modelle in das Unternehmenssystem durchführen. Des Weiteren stellen Domänenexpert*innen ihr fachbezogenes Know-how zur Verfügung und bilden somit die Grundlage für eine erfolgreiche Projektdurchführung. Eine KI-Plattform ist dabei eine zentrale Organisationseinheit für alle Projektbeteiligten. Je nach Systemausprägung basiert die Plattform entweder auf einer lokalen Softwarelösung (z. B. TensorFlow) oder einem Cloud-Dienst (z. B. Azure). Neben den Datenaufbereitungs- und Analyse-Funktionen sollte eine KI-Plattform Visualisierungsmöglichkeiten und eine umfangreiche Auswahl an Algorithmen zur Verfügung stellen sowie eine Möglichkeit zur Modellbereitstellung bieten. Im Idealfall ermöglicht es die Plattform, alle Phasen eines ML-Projektes abzubilden. Ferner muss bei der Auswahl einer geeigneten Umgebung die Flexibilität beachtet werden. Flexibilität bezieht sich in diesem Kontext auf die Offenheit der Plattform bzgl. Integration externer Open-Source-Tools sowie die Möglichkeit zur Anwendung relevanter Programmiersprachen (Python, R, Scala).[53]

2.4.2 ML-Pipeline

Unerfahrene Teams tendieren bei der Entwicklung von ML-Projekten eher dazu, problemorientiert zu arbeiten. Das bedeutet, dass in erster Linie die Erfüllung der Aufgabe im Vordergrund steht. Auf die Struktur und den Aufbau wird weniger Wert gelegt und gesamtheitliche Lösungen werden oftmals in einem einzelnen Workspace erarbeitet. Dadurch entstehen unübersichtliche und nicht wiederverwendbare Codefragmente. Machine-Learning-Pipelines verfolgen das Ziel, ebendieser Problematik entgegenzuwirken. Die Idee von ML-Pipelines ist es, die Schritte der einzelnen Phasen von der Datenerfassung bis hin zur Modellbereitstellung voneinander zu trennen und somit eine getrennte Entwicklung für die verschiedenen beteiligten Rollen (Data-Engineer, Data-Scientist etc.) zu ermöglichen. Ähnlich wie bei der modellbasierten Entwicklung oder der objektorientierten Programmierung steht hier die Abstraktion im Vordergrund. Jede ML-Pipeline ist ein eigenständig ausführbarer Workflow, der klar definierte Schnittstellen aufweist. Jeder Workflow beschreibt dabei eine Teilaufgabe (Komponente). Dies ermöglicht einerseits eine Versionspflege der einzelnen Komponenten und erhöht andererseits die Wiederverwendbarkeit sowie die Übersichtlichkeit.[54]

[53] Vgl. Wuttke (2021), S. 85 - 100.

[54] Vgl. Microsoft Corporation 2 (2022), Online-Quelle [07.08.2022].

3 AUDIOSIGNALVERARBEITUNG MIT ML-METHODEN

Nachdem in Kapitel 2 der Bereich KI im Allgemeinen betrachtet wurde, bezieht sich dieser Abschnitt konkret auf die Audiosignalanalyse mit den State-of-the-Art-ML-Methoden. Zunächst erfolgt jedoch eine Einführung in das Themengebiet der Audiosignale.

Als Audiosignal wird ein elektrisches Signal, das akustische Informationen enthält und transportiert, bezeichnet. Für die Umwandlung des Schalldruckpegels des akustischen Signals in eine elektrische Spannung kann beispielsweise ein Mikrofon (Piezo) verwendet werden. Da Audiosignale zu jedem Zeitpunkt t definiert sind, kann von einem (zeit-)kontinuierlichen Signal $x(t)$ ausgegangen werden. Ferner kann zwischen periodischen und stochastischen Signalen (Zufallssignale, Rauschen) unterschieden werden. Periodische Signale wiederholen sich im konstanten Zeitintervall:

$$x(t) = x(t + T_0) \qquad (3.1)$$

t/s Zeit
T_0/s Periodendauer der Grundschwingung

Dabei beschreibt T_0 das periodische Intervall bzw. die Länge der Periode der Grundschwingung (der ersten Harmonischen). Demnach lässt sich die Grundfrequenz des Signals folgendermaßen definieren:

$$f_0 = \frac{1}{T_0} \qquad (3.2)$$

f_0/Hz Grundfrequenz

Die Frequenzen der weiteren Harmonischen des Signals entsprechen immer einem ganzzahligen Vielfachen der Grundfrequenz (z. B. 50 Hz, 100 Hz, 150 Hz …). Jedes beliebige periodische Signal mit der Periode T_0 kann durch eine (unendliche) Summe von Sinus- und Kosinusschwingungen abgebildet werden (Fourier-Synthese):

$$x(t) = \sum_{k=-\infty}^{\infty} a(k)\, e^{j\omega_0 kt} \qquad (3.3)$$

ω_0/s^{-1} Kreisfrequenz

Die Kreisfrequenz $\omega_0 = 2\pi f_0$ ist das Maß für die Periodizität. Die eulersche Formel $e^{j\omega_0 t} = \cos(\omega t) + j\sin(\omega t)$ beschreibt den Anteil der ungeraden Sinuskomponenten sowie den der geraden Kosinuskomponenten und $a(k)$ stellt den Fourier-Koeffizienten der k-ten Harmonischen dar.[55]

Das Verarbeiten von kontinuierlichen Signalen ist mit Computersystemen nicht möglich, weshalb zunächst eine Umwandlung in ein zeit- und wertdiskretes Signal durchgeführt werden muss. Dabei werden sowohl die Amplitude (Quantisierung) als auch die Zeitachse (Abtastung, engl. Sampling) diskretisiert. Beim Sampling wird das kontinuierliche Signal $x(t)$ mit der Abtastrate f_s abgetastet (siehe Abbildung 9 unten). Um Abtastfehler sowie Aliasing-Effekte zu vermeiden, ist darauf zu achten, dass das Sampling-Theorem $f_s > 2\, f_{max}$ eingehalten wird, wobei f_{max} der höchsten im Signal enthaltenen Frequenz in Hz entspricht.[56]

[55] Vgl. Lerch (2012), S. 7 f.

[56] Vgl. Lerch (2012), S. 10.

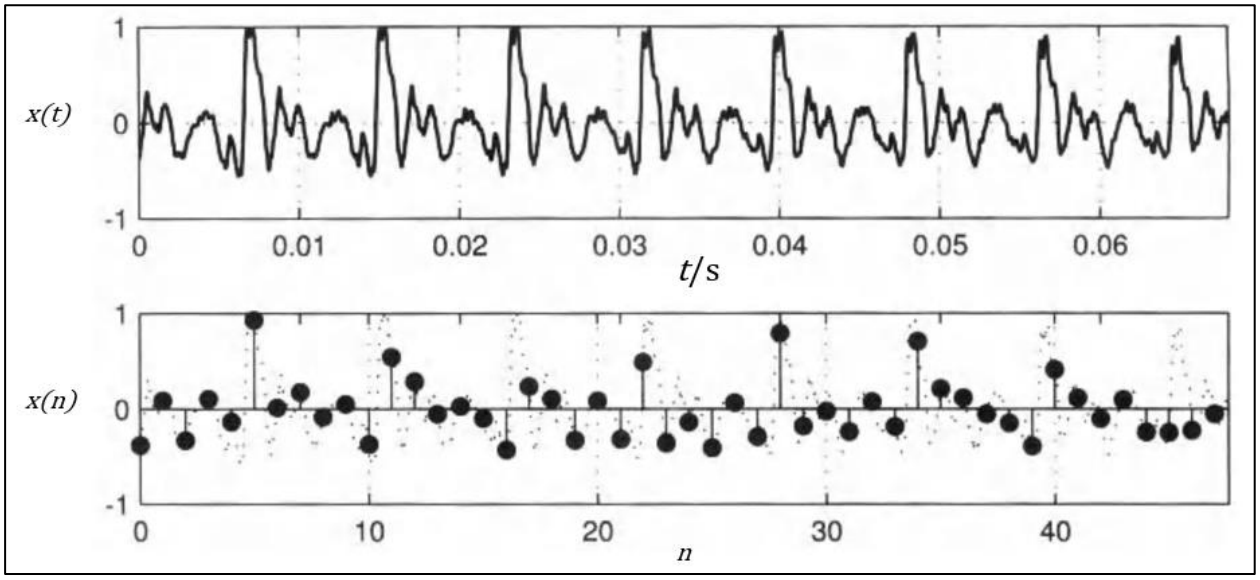


Abbildung 9: Kontinuierliches Audiosignal (oben), abgetastetes Signal mit Samplerate 700 Hz (unten), Quelle: Lerch (2012), S. 10 (leicht modifiziert).

Im Anschluss an das Abtasten wird beim Quantisieren jeder Amplitudenwert auf einen endlichen Wert gerundet. Das bedeutet, dass die Amplitudenachse in n-bit unterteilt wird. Typische Werte sind beispielsweise 8, 16 oder 24 bit, wobei eine Auflösung von 2^8, 2^{16} oder 2^{24} erreicht wird.[57]

3.1 Audio-Processing

Damit ML-Modelle in den digitalisierten Audiosignalen Muster erkennen können, ist eine weitere Datenaufbereitung notwendig. Es wird zwischen der Datenvorverarbeitung (engl. Preprocessing) und der Merkmalsextraktion (engl. Feature-Extraction) unterschieden.

Preprocessing:

In der Preprocessing-Phase geht es um die Verbesserung bestimmter Eigenschaften der digitalisierten Audiosignale. Am Beispiel einer in der U-Bahn-Station aufgenommenen Sprachaufnahme wird schnell klar, dass Audiosignale in der Praxis meist weder periodisch noch frei von Störgeräuschen sind. Zudem stammen die Daten eines Datensatzes oftmals aus verschiedenen Quellen mit unterschiedlichen Aufnahmeeinstellungen. Dazu zählen die Abtastrate und die Anzahl der Audiokanäle. Behoben werden kann dies durch Down-Mixing, also die Vereinheitlichung der Audiokanäle sowie eine Neuabtastung des Signals mit einer definierten und einheitlichen Samplingrate (Down-Sampling). Ferner kann die Qualität des Signals durch Rauschunterdrückungsmethoden verbessert werden.[58] Gängige Methoden hierfür sind die in 2.3.2 kurz erläuterten rekurrenten neuronalen Netzwerke (RNN) mit Long-Short-Term-Memory(LSTM)- oder Gated-Recurrent-Unit(GRU)-Architektur.[59] Auf diese wird in Abschnitt 3.2.2 genauer eingegangen.

[57] Vgl. Lerch (2012), S. 11.

[58] Vgl. Ellis/Plumbley/Virtanen (2018), S. 19 f.

[59] Vgl. Han u.a. (2021), S. 20; Vgl. Salmanabadi u.a. (2022), S. 9.

Feature-Extraktion

Machine-Learning-Algorithmen sind meist rechenintensiv, weshalb Audiosignalanalysen in der Regel auf der Analyse von akustischen Merkmalen (z. B. Amplitude, Frequenz) basieren, die aus den Eingangsdaten extrahiert werden. Das Eingangssignal wird somit von Redundanzen befreit sowie auf die zentralen Features reduziert. Ziel ist es, dass Merkmale unterschiedlicher Beispiele, jedoch gleicher Klasse, eine möglichst geringe Variabilität aufweisen. Gleichzeitig sollten Merkmale von Beispielen unterschiedlicher Klassen eine hohe Variabilität besitzen. Dies ermöglicht es den Lernalgorithmen, zwischen den Klassen zu differenzieren.[60] Bei der Merkmalsextraktion werden also im Idealfall die relevantesten charakteristischen Faktoren des Audiosignals auf die minimal mögliche Dateigröße abgebildet. Im Verlauf der Jahre wurde eine Vielzahl an Methoden für die Feature-Extraktion entwickelt, wodurch Merkmale von Audiosignalen in unterschiedlicher Darstellungsform extrahiert werden können.[61] Der Erfolg der Klassifizierungsaufgabe hängt maßgeblich von der Phase der Merkmalsextraktion ab, weshalb diese im Detail behandelt wird.

3.1.1 Darstellungsformen eines Audiosignals und Merkmalsextraktion

Zunächst wird auf die bedeutendsten Darstellungsformen eines Audiosignals und die jeweiligen Features eingegangen. Ein Audiosignal lässt sich unter anderem als Wellenform im Zeitbereich, als Spektrum im Frequenzbereich und als Spektrogramm oder Mel-Spektrogramm im Zeit-Frequenz-Bereich darstellen. Die Klassifizierungsaufgabe kann prinzipiell mit all diesen Formen als Input durchgeführt werden. Jedoch muss vorab eine wohlüberlegte Auswahl getroffen werden, da dies weitreichende Auswirkungen auf die weitere Vorgehensweise hat. Je nach Form unterscheiden sich die Aufwände für die Datenvorbereitung sowie die Auswahl geeigneter Algorithmen. Im Folgenden wird auf die vier Darstellungsformen eingegangen:

Wellenform im Zeitbereich

Die Wellenform entspricht der in Abschnitt 3 erläuterten Darstellungsform. Wie bereits erwähnt, sind Audiosignale in der Praxis üblicherweise nicht periodisch, was die Analyse erheblich erschwert, da das Signal eine nicht vorhersehbare dynamische Charakteristik aufweist. Zu diesem Zweck wird das Signal bei der Analyse in kleinere Teilabschnitte segmentiert. Dabei wird beispielsweise eine Rechteckfunktion von links nach rechts über den Signalbereich verschoben (siehe Abbildung 10). Diese Technik wird als Windowing bezeichnet und wird üblicherweise adaptiv ausgeführt. Das heißt, dass die Fensterbreite über den Verlauf an die jeweiligen Signalbereiche angepasst wird. Ziel ist die Transformation eines nicht stationären Signals in ein quasistationäres Signal (je Teilabschnitt).[62]

[60] Vgl. Ellis/Gold/Morgan (2011), S. 19 f.

[61] Vgl. Sharma/Umapathy/Krishnan (2019), S. 1.

[62] Vgl. Sharma/Umapathy/Krishnan (2019), S. 4.

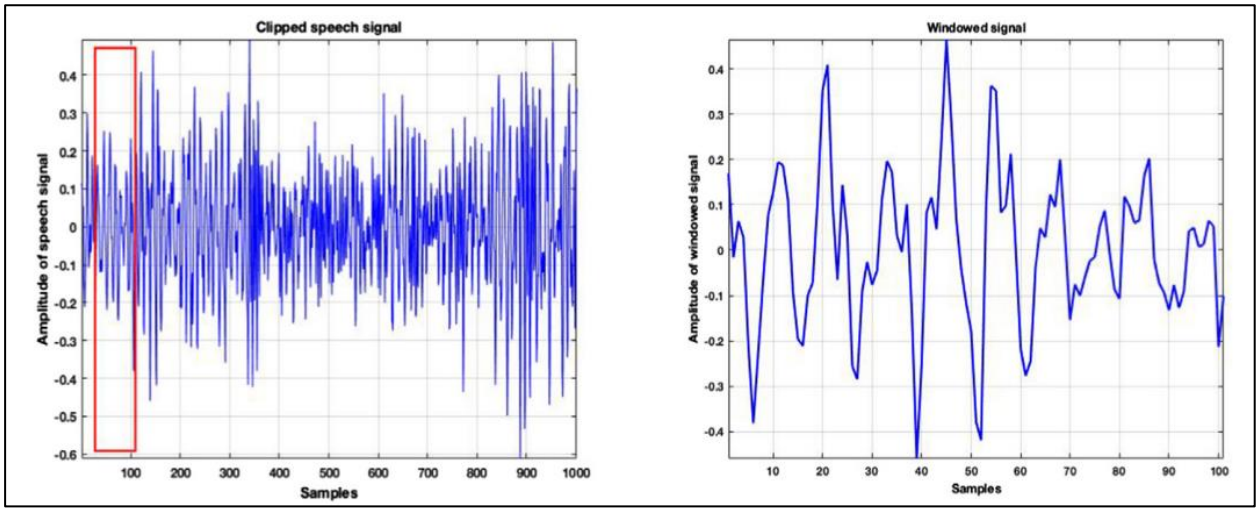


Abbildung 10: Windowing-Technik bei einem zeitdiskreten Audiosignal. Definiertes Fenster als Rechteckfunktion (links) und Ausschnitt des gefensterten Signals (rechts), Quelle: Sharma/Umapathy/Krishnan (2019), S. 5.

Im Anschluss erfolgt die Merkmalsextraktion bezogen auf den jeweils betrachteten Teilabschnitt. Im Folgenden werden drei zentrale Merkmale von Audiosignalen als Wellenform erläutert:[63]

- Zero-Crossing-Rate (ZCR) – Gibt an, wie oft die Amplitude in einem definierten Intervall die Nulllinie durchschreitet. Mit dem ZCR-Wert kann sehr einfach zwischen periodischem Signal (niedriger ZCR-Wert) und verrauschtem Zufallssignal (hoher ZCR-Wert) unterschieden werden.
- Short-Time-Energy (STE) – Entspricht der Energie innerhalb eines Frames. Der STE-Wert ist in soundbehafteten Momenten hoch und in soundlosen Momenten niedrig. Mittels STE-Werts können Umgebungsgeräusche erkannt werden.
- Tonhöhe und Lautstärke – Es gilt: Je höher die Frequenz, desto höher der Ton; je größer die Amplitude, desto höher die Lautstärke.

Spektrum im Frequenzbereich

Um ein Signal vom Zeitbereich in den Frequenzbereich zu transformieren, wird die Fourier-Transformation verwendet. Diese ermöglicht es, ein zeitbezogenes Signal in dessen einzelne Frequenzen zu zerlegen und die zugehörige Amplitude abzubilden (siehe Abbildung 11). Da ein werte- und zeitdiskretes Signal $x[n]$ vorliegt, wird die Diskrete Fourier-Transformation (DFT) angewandt. Es werden t und f durch diskrete Variablen mit ganzzahligen Indizes n und k ersetzt:[64]

$$X(k) = \sum_{n=0}^{N-1} x[n]\, e^{-\frac{j2\pi kn}{N}} \qquad (3.4) \qquad k = 0, 1, 2, \ldots, N-1$$

Um die Berechnung schnell auszuführen, ist es in der Computertechnik üblich, dass die DFT in Form der Fast Fourier-Transformation (FFT) implementiert ist. Die FFT ist ein Algorithmus zur effizienten Berechnung der DFT.[65]

[63] Vgl. Sharma/Umapathy/Krishnan (2019), S. 5 ff.

[64] Vgl. Lerch (2012), S. 20.

[65] Vgl. Lerch (2012), S. 197.

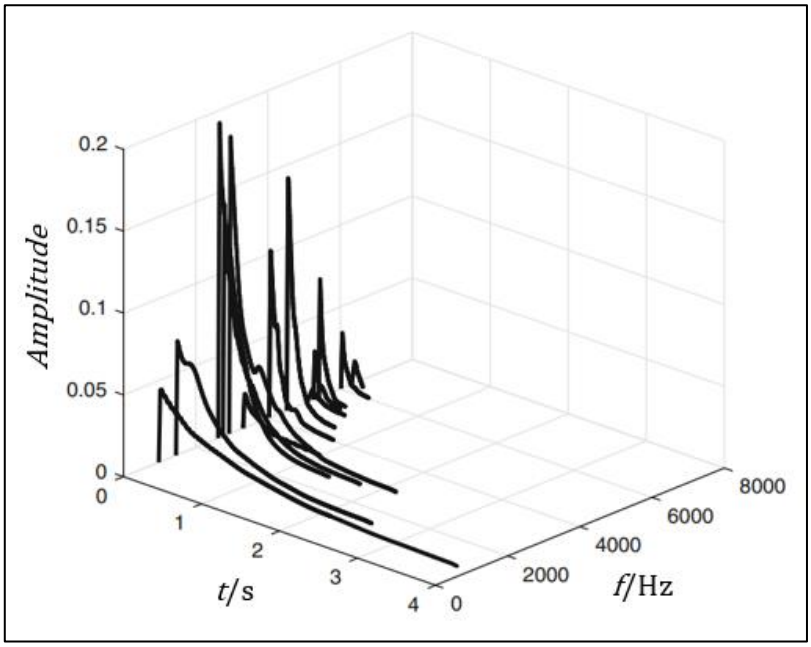


Abbildung 11: Zusammenhang zwischen der Darstellung eines kontinuierlichen Signals im Zeit- und Frequenzbereich (Spektrum), Quelle: Ellis/Plumbley/Virtanen (2018), S. 55 (leicht modifiziert).

Wie im Zeitbereich, ist die Energie auch im Frequenzbereich ein relevantes Feature. Ferner ist die Spitzenfrequenz von Bedeutung. Die Spitzenfrequenz ist die Frequenz der maximalen Leistung des Signals. Sie unterstützt beim Berechnen der Grundfrequenz und ist ein entscheidendes Merkmal bei Klassifizierungsaufgaben.[66]

Spektrogramm und Mel-Spektrogramm im Zeit-Frequenz-Bereich

Zur Transformation eines Audiosignals in ein Spektrogramm wird die Kurzzeit-Fourier-Transformation (engl. Short-time Fourier-Transformation, STFT) berechnet. Dazu wird das Audiosignal in einzelne Blöcke segmentiert (Windowing). Auf jeden Block wird die DFT in Form der FFT angewandt. Eine gängige Variante für eine Fensterfunktion ist beispielsweise die Hanning-Funktion, die einen Glockenkurvenverlauf aufweist. Im Gegensatz zur Rechteckfunktion wird dadurch ein sanfterer Übergang zwischen zwei Rahmen erreicht. Ein weiterer relevanter Parameter der STFT ist die Schrittweite zwischen zwei Fenstern (engl. Hop-Size). Üblicherweise wird diese so gewählt, dass sich die Rahmen überlappen. Dies dient einerseits ebenfalls zum Erreichen eines sanfteren Übergangs und andererseits können dadurch statistische Beziehungen zwischen den Blöcken erkannt werden.[67] Abbildung 12 (links) visualisiert die gesamte soeben erläuterte Vorgehensweise bei der Berechnung der STFT mit Dreiecksfensterfunktion. Es resultiert ein Spektrogramm (Abbildung 12 rechts). Die x-Achse bezieht sich dabei auf die Zeit in s (diskret = Anzahl der gestapelten Frames in Abhängigkeit der Hop-Size), die y-Achse auf die Frequenz in Hz (diskret = Anzahl der Frequenz-Bins in Abhängigkeit der FFT-Größe). Die Amplitude spiegelt den farblichen Verlauf wider. Da Menschen die Lautstärke logarithmisch wahrnehmen, ist es üblich, diese in dB anzugeben.[68]

[66] Vgl. Sharma/Umapathy/Krishnan (2019), S. 8.

[67] Vgl. Ellis/Plumbley/Virtanen (2018), S. 74 f.

[68] Vgl. Rajalingappaa/Rajesh (2018), S. 244.

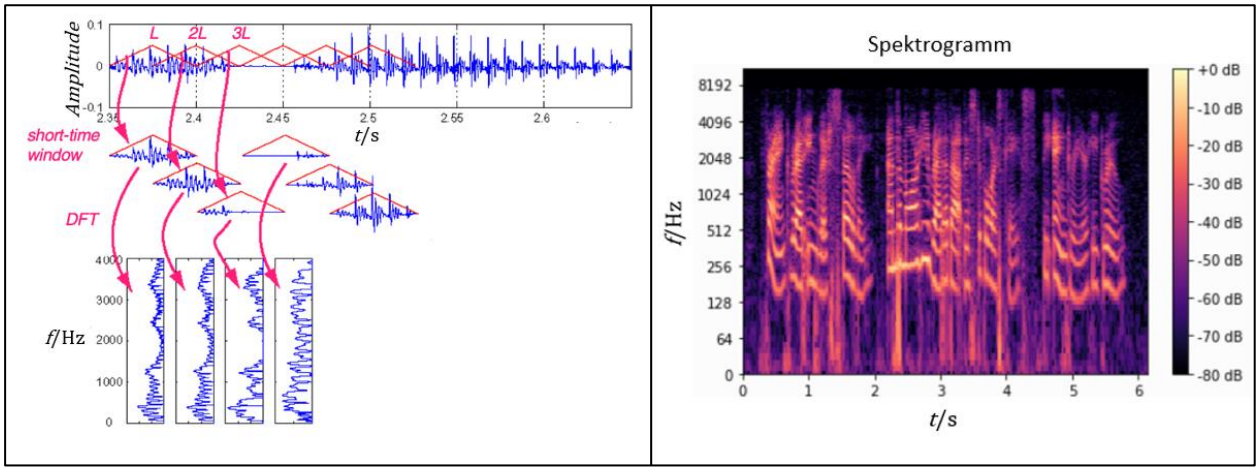


Abbildung 12: Berechnung der STFT (schematische Darstellung) und beispielhaftes Spektrogramm (rechts), Quelle: Pardo (2008), Online-Quelle [12.09.2022] (links), NVIDIA Corporation (o.J.), Online-Quelle [19.11.2022] (rechts), (beide leicht modifiziert).

Studien haben gezeigt, dass Änderungen der Tonhöhe in unterschiedlichen Frequenzbereichen vom menschlichen Gehör nicht linear wahrgenommen werden. Beispielsweise werden Änderungen tieferer Töne (z. B. 50 Hz zu 550 Hz) besser wahrgenommen als Änderungen im höheren Frequenzbereich (z. B. 15 000 Hz zu 15 500 Hz). Um diesen Effekt auf Computersystemen abbilden zu können, wird die Mel-Skala verwendet.[69] Diese ist eine Maß für die wahrgenommene Tonhöhe (Tonheit) und ist folgendermaßen definiert:

$$z = 1127 \ln(1 + \frac{f}{700}) \qquad (3.5)$$

f/Hz Frequenz
z/Mel Tonheit

Mit dem inversen Ausdruck:

$$f = 700(e^{z/1127} - 1) \qquad (3.6)$$

Abbildung 13 (links) visualisiert diesen nichtlinearen Zusammenhang zwischen Mel- und Frequenz-Skala. Wie ersichtlich ist, verstärkt sich der Effekt mit zunehmender Frequenz. Um ein Mel-Spektrogramm zu erhalten, wird eine Mel-Filterbank auf ein Spektrogramm angewendet. Eine Mel-Filterbank besteht aus einer zu definierenden Anzahl aus n-Dreiecks-Filtern, die gemäß der Mel-Skala verteilt sind (Abbildung 13 rechts). Die Frequenz-Achse der Filterbank orientiert sich dabei an der Nyquist-Frequenz ($f_s/2$). Durch das Anwenden von n-Filtern auf das Spektrogramm wird eine begrenzte Anzahl von n-Mel-Bändern extrahiert, die einen jeweiligen Frequenzbereich repräsentieren (y-Achse). Aufgrund der Beschränkung (n-Filter) resultiert eine komprimierte Darstellung eines (Mel-)Spektrogramms, was einen Vorteil für die weiteren Berechnungsschritte darstellt.[70]

Zur Veranschaulichung der Unterschiede sind in Abbildung 14 ein Spektrogramm (links) und ein Mel-Spektrogramm (rechts) derselben Audiodatei dargestellt. Zu Vergleichszwecken sind beide Diagramme mit den gleichen Skalen (Mel-Inverse) abgebildet.

[69] Vgl. Molau u.a. (2001), o.S.

[70] Vgl. Rajalingappaa/Rajesh (2018), S. 243 - 246.

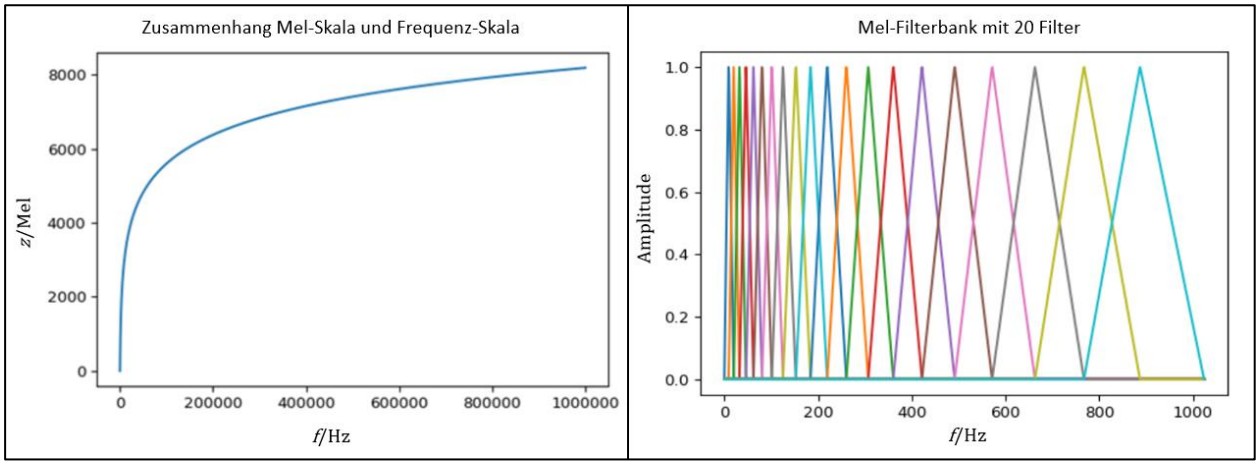


Abbildung 13: Zusammenhang zwischen Mel-Skala und Frequenz-Skala (links) und beispielhafte Mel-Filterbank mit 20 Filtern (rechts), Quelle: Rajalingappaa/Rajesh (2018), S. 243 (links) und S. 244 (rechts) (beide leicht modifiziert).

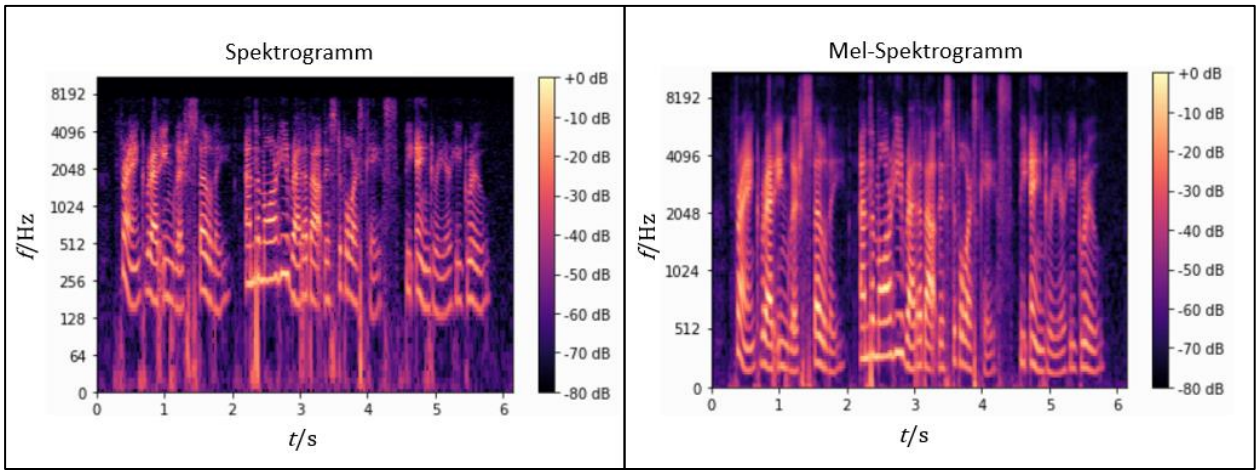


Abbildung 14: Spektrogramm (links) und Mel-Spektrogramm (rechts), Quelle: NVIDIA Corporation (o.J.), Online-Quelle [19.11.2022] (beide leicht modifiziert).

3.1.2 Diskussion zur Merkmalsextraktion

In Abschnitt 3.1.1 wurden verschiedene Darstellungsformen von Audiosignalen erläutert. Es wurde verdeutlicht, dass die Datenvorbereitung ein umfangreiches Aufgabengebiet im Zuge eines ML-Projektes ist. Die Transformationen zwischen den einzelnen Domänen (Zeit, Frequenz) und die Merkmalsextraktion basieren auf komplexer Mathematik, die aufgrund des Umfangs nur oberflächlich behandelt werden kann. Die Illustrationen vermitteln aber ein generelles Verständnis des Aufgabengebiets. In der Praxis erfolgt die Implementierung mittels Programmiersprachen (z. B. Python) und Softwarebibliotheken, beispielsweise Librosa, die pythonbasierte Bausteine für die Audiosignalanalyse zur Verfügung stellt und somit die Bearbeitung der Aufgabenstellung erheblich vereinfacht.[71]

Neben den Darstellungsformen wurde auf einige zentrale Features der jeweiligen Domäne eingegangen. Im Allgemeinen existiert noch eine Vielzahl an weiteren Merkmalen von Audiosignalen, deren Extraktion meist auf statistischen Verfahren (z. B. quadratischer Mittelwert) basiert.

[71] Vgl. Librosa (2015), Online-Quelle [12.09.2022].

In Abschnitt 2.2 sowie 2.2.1 werden die Unterschiede von ML und DL beschrieben. Demnach erfolgt beim klassischen ML die Merkmalsextraktion manuell. Das heißt, dass die Audiodaten akribisch analysiert und zentrale Merkmale von Fachexpert*innen definiert werden müssen. Dies erfordert jedoch zusätzlich umfangreiche Kenntnisse im Bereich von Audiosignalen, was sich auf Unternehmen nachteilig auswirken kann, da es bereits einen Fachkräftemangel im Bereich KI und ML gibt. Nach aktuellem Stand der Technik haben sich DL-Methoden für die Merkmalsextraktion etabliert.[72] Dabei werden DL-Algorithmen mit einer der in 3.1.1 erläuterten Darstellungsformen als Input-Feature versorgt, woraufhin die verwendeten Algorithmen automatisiert entscheidende Merkmale für die Klassifizierungsaufgabe extrahieren. Als bevorzugte Input-Features wird entweder die Wellenform oder die Mel-Spektrogramm-Form verwendet.[73] Obwohl es Arbeiten im Bereich der Audiomustererkennung gibt, die gute Ergebnisse mit der rohen Wellenform erzielen,[74] herrscht ein genereller Trend in Richtung Verwendung der Mel-Spektrogramme.[75]

Ein Durchbruch bei der Klassifizierung mittels DL konnte im Jahr 2012 erreicht werden, als im Zuge des ImageNet-Wettbewerbes faltende neuronale Netzwerke (engl. Convolutional Neural Networks, CNNs) für die Bildklassifizierung eingesetzt wurden. Das CNN mit AlexNet-Architektur konnte sich dabei erstmals gegen alle konventionellen ML-Methoden durchsetzen. Seit 2012 gewinnen ausschließlich Lösungen, die auf CNNs basieren.[76] Es hat sich gezeigt, dass CNNs nicht nur für die Bildklassifizierung, sondern auch für die Audiosignalklassifizierung eine bewährte Methode sind.[77]

Ferner konnte in vergangenen Arbeiten beobachtet werden, dass sich RNN, insbesondere jene mit LSTM- sowie GRU-Architektur, für die Audiosignalanalyse eignen.[78] Gute Ergebnisse konnten auch mit hybriden Architekturen, also der Verknüpfung von CNNs und RNNs (CRNNs), erreicht werden.[79] Eine pauschale Aussage, welche der genannten Methoden sich am besten für die Audioklassifizierung eignet, kann jedoch nicht getroffen werden. Dies scheint einerseits stark vom zugrunde liegenden Datensatz und andererseits von der Architektur (Anzahl der Schichten etc.) und der Feinabstimmung der Parameter des KNNs abzuhängen und muss für die Aufgabenstellung individuell durch Experimente evaluiert werden. Untermauert wird diese Aussage von Purwins u. a.[80] Jedoch kann der Bereich auf die bereits erläuterten Architekturen (CNN, LSTM/GRU, CRNN) beschränkt werden.

[72] Vgl. Ellis/Plumbley/Virtanen (2018), S. 85.

[73] Vgl. Purwins u.a. (2019), S. 10.

[74] Vgl. Dai u.a. (2017), o.S.; Becker u.a. (2018), o.S.

[75] Vgl. Purwins u.a. (2019), S. 10; Vgl. Zhang/Leitner/Thornton (2019), o.S.; Vgl. Tanveer u.a. (2021), S. 221.

[76] Vgl. Charu (2018), S. 316 f.

[77] Vgl. Sharma/Umapathy/Krishnan (2019), S. 15; Vgl. Ellis/Plumbley/Virtanen (2018), S. 85; Vgl. Purwins u.a. (2019), S. 10.

[78] Vgl. Purwins u.a. (2019), S. 10; Vgl. Gupta u.a. (2019), S. 10; Vgl. Petmezas u.a. (2022), S. 11.

[79] Vgl. Benito-Goron u.a. (2019), S. 16.

[80] Vgl. Purwins u.a. (2019), S. 10.

3.2 Methoden

Die eben erläuterten Methoden und deren Funktionsweise sollen nun detailliert betrachtet werden. Zudem werden in diesem Abschnitt zusätzliche Methoden behandelt, die insbesondere auf die Problemstellung von DL-Methoden in Kombination mit geringen Datenmengen eingehen.

3.2.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks sind Feedforward-Netzwerke, bei denen die Zustände jeder Schicht in räumlichen Gitterstrukturen angeordnet sind. Anders als bei den in Abschnitt 2.3.2 vorgestellten mehrschichtigen Netzen sind bei CNNs nicht alle Neuronen mit jedem Neuron der nächsten Schicht verknüpft. Stattdessen werden die räumlichen Beziehungen von einer Schicht an die nächste vererbt. Jede Schicht eines CNN besteht aus einer dreidimensionalen Struktur mit einer Höhe, einer Breite und einer Tiefe, wobei sich die Tiefe auf die Anzahl der Kanäle bzw. der Merkmalskarten (engl. Feature-Maps) bezieht. Im Falle eines RGB-Bildes mit 32 x 32 Pixeln würde dies somit einer Struktur von 32 x 32 x 3 entsprechen (siehe Abbildung 15 links). Convolutional Neural Networks bestehen aus verschiedenen Typen von Schichten. Die am häufigsten verwendeten sind Convolution-, ReLU- sowie Pooling-Layer. Namensgebender Bestandteil von CNN sind Faltungsmatrizen, die als Filter oder Kernel bezeichnet werden. Die Filter werden ebenfalls mittels einer dreidimensionalen Struktur abgebildet und über die Matrix des Eingangs bewegt. Abbildung 15 (rechts) visualisiert einen solchen Faltungsprozess mit einer 7 x 7 x 1-Input-Matrix und einem 3 x 3 x 1-Kernel. Die Ausgangsmatrix stellt alle Skalarprodukte zwischen der räumlichen Region des Inputs und des Filters dar.[81] Als Beispiel wird die Berechnung des rot markierten Bereichs in Abbildung 15 angeführt:

$$5 \cdot 1 + 8 \cdot 0 + 1 \cdot 1 + 1 \cdot 8 + 0 \cdot 0 + 1 \cdot 0 + 6 \cdot 0 + 0 \cdot 4 + 1 \cdot 2 = 16$$

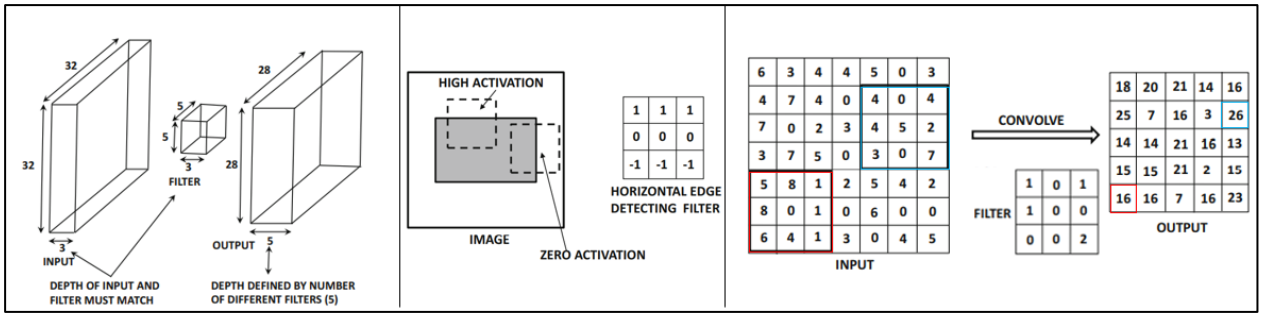


Abbildung 15: Convolution-Layer mit Kernel (links), Filter mit horizontaler Kantenerkennung (Bildmitte) sowie Berechnungsvorgang (rechts), Quelle: Charu (2018), S. 19 f. (leicht modifiziert).

Übliche Filtergrößen für Breite und Höhe sind kleine ungerade Zahlen (z. B. drei oder fünf). Die Tiefe muss der Tiefe des Convolution-Layers entsprechen und die möglichen Filterpositionen bestimmen die Matritzen-Größe des Ergebnisses. Die Idee ist es, die Parameter der Filter so anzupassen, dass bestimmte Merkmale in den Eingangsdaten (z. B. Mel-Spektrogramm) vom neuronalen Netz erkannt werden können. Der Filter in Abbildung 15 (Bildmitte) kann beispielsweise horizontale Kanten erkennen.

[81] Vgl. Charu (2018), S. 318 f.

Eine Parametrierung der Filter erfolgt im Zuge des Trainings automatisiert durch die Anpassung der Gewichtungen (siehe Abschnitt 2.3). Welche Features relevant sind und in welcher Reihenfolge diese extrahiert werden, ist dabei nicht bekannt.[82] Der Ausgang jedes Berechnungsvorgangs repräsentiert eine eigene Feature-Map, die die wesentliche Charakteristik des Inputs enthält (z. B. Kanten, Linien, Muster). Dabei korrespondiert die Anzahl der Filter mit der Anzahl der Feature-Maps. Es ist üblich, dass mehrere hunderte Filter innerhalb einer Convolution-Schicht angewendet werden, um vorhandene Muster bestmöglich erkennen zu können. Wie bereits in 2.2.1 erwähnt, tendieren neuronale Netze dazu, in den früheren Schichten komplexere Formen bzw. Muster (z. B. Form eines Gesichts) und in den tieferen Schichten primitivere Strukturen (Linien, Kanten) zu erkennen. Neben bereits erläuterten Parametern von CNN (Ausdehnung, Filter) werden die folgenden zwei kurz erläutert: [83]

- Auffüllen (engl. Padding) – Durch das Anwenden der Filter reduziert sich die räumliche Ausdehnung der Ausgangsmatrix. Um diesem Problem entgegenzuwirken, werden die Ränder mit Werten (z. B. Nullen) aufgefüllt.
- Schrittweite (engl. Stride) – Gibt die Schrittweite der Verschiebung des Filters an.

ReLU-Layer

Die ReLU-Aktivierung verhält sich bei CNNs ähnlich wie die Aktivierungsfunktion bei klassischen neuronalen Netzen (siehe Abschnitt 2.3). Dabei wird die ReLU-Aktivierungsfunktion auf jeden Wert der Convolution-Schicht (Output) angewendet. Bei Überschreiten des Schwellwertes (Wert > 0) nimmt der Ausgang linear zu. Im Bereich DL hat sich ReLU als Aktivierungsfunktion etabliert.[84]

Pooling-Layer

Die Pooling-Schicht schließt üblicherweise an eine ReLU-Schicht an und dient zur Reduktion redundanter Informationen. Ziel ist es, die wesentlichen Merkmale zu erfassen und sich nicht in Details zu verlieren. Die in der Praxis etablierte Technik wird als Max-Pooling bezeichnet, wobei jeweils der höchste Wert aus einer räumlichen Region von vier Zahlenwerten an die nächste Schicht weitergeleitet wird (siehe Abbildung 16). Ein Vorteil, der sich aufgrund des Pooling-Layers ergibt, ist ein reduzierter Rechenaufwand, wodurch tiefere Netze erzeugt werden können.[85]

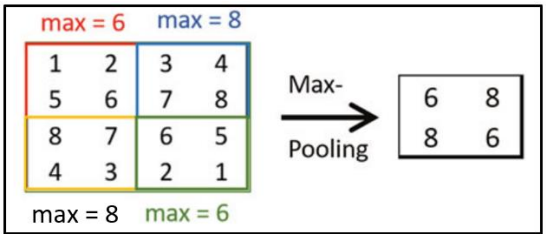


Abbildung 16: Max-Pooling CNN, Quelle: Sonnet (2022), S. 69 (leicht modifiziert).

[82] Vgl. Charu (2018), S. 319 f.

[83] Vgl. Charu (2018), S. 320.

[84] Vgl. Charu (2018), S. 325.

[85] Vgl. Sonnet (2022), S. 69 f.

Flatten- und Fully-connected Layer

Um Vorhersagen treffen zu können (Klassifikation), muss die räumliche Struktur eines CNNs am Ende in eine eindimensionale Struktur überführt werden. Dieser Prozess geschieht innerhalb der Flatten-Schicht, worauf ein Fully-connected Layer (FCL) folgt. Der Aufbau und die Funktionsweise eines FCL gleichen den in Abschnitt 2.3.2 vorgestellten traditionellen Feedforward-Netzwerken.[86]

3.2.2 Recurrent Neural Networks (RNN)

Convolutional-Neural-Networks-Architekturen sind auf die Analyse von multidimensionalen Daten (z. B. Bilddateien) ausgelegt, bei denen die Eigenschaften unabhängig voneinander sind. Es gibt jedoch Datenstrukturen, die eine sequenzielle Abhängigkeit aufweisen. Als Beispiel hierfür können Textdaten genannt werden. Dabei nimmt etwa die Reihenfolge der Wörter eine wesentliche Rolle beim Erschließen des Kontexts im Zuge einer Wortanalyse ein. Diese Zusammenhänge sind ebenfalls im Bereich von Audiosignalen erkennbar.[87] Um solche Problemstellungen zu lösen, werden in der Praxis RNN eingesetzt. Der Unterschied zu klassischen Feedforward-Netzwerken ist, dass RNN über Rückkopplungen verfügen, die ein Fortbestehen der bereits verarbeiteten Informationen ermöglichen. Verglichen mit dem menschlichen Gehirn kann dies als Gedächtnis betrachtet werden, wobei zwischen drei Arten von Rückkopplungen differenziert wird. Neuronen können sich entweder selbst stimulieren, auf andere Neuronen in derselben Schicht oder auf Neuronen einer vorgelagerten Schicht zugreifen.[88] Abbildung 17 (links) visualisiert ein solches RNN in einfachster Struktur am Beispiel einer Vorhersage von Wörtern. Für das Verständnis ist es jedoch einfacher, das RNN in ausgerollter Form (rechts) zu betrachten. Die Idee besteht darin, dass für jedes Element der Sequenz (jedes Wort) eine neue Instanz des Modells erzeugt wird. Hierbei verwendet jede Instanz dieselben Parameter sowie dieselbe Anzahl von Parametern. Zudem wird die Lebensdauer aller Netzwerkinstanzen in diskrete Zeitschritte unterteilt. Nach jedem Zeitschritt wird das Modell mit dem nächsten Element der Sequenz (dem nächsten Wort) versorgt. Die Anzahl der Schichten ist variabel, hängt also von der Anzahl der Elemente in der Sequenz ab. Bei diesem Beispiel versorgt sich das Netzwerk (h_2) mit den Informationen des vorhergehenden Zeitschritts (h_1) selbst.[89]

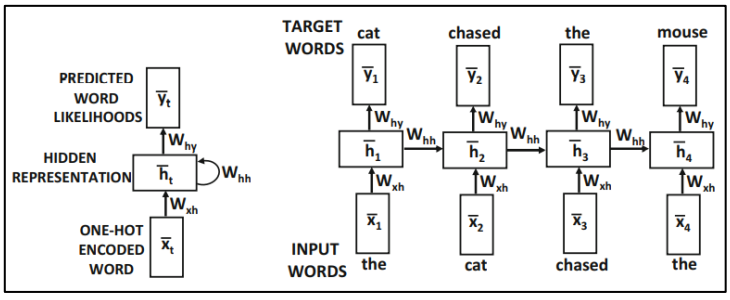


Abbildung 17: Ein RNN in einfacher Form (links) sowie die ausgerollte Darstellung des RNN (rechts), Quelle: Charu (2018), S. 276.

[86] Vgl. Charu (2018), S. 327 f.

[87] Vgl. Charu (2018), S. 272.

[88] Vgl. Sonnet (2022), S. 62 f.

[89] Vgl. Buduma (2017) S. 173 f.

Bei Betrachtung des RNN in ausgerollter Form fällt auf, dass dieses einer Feedforward-Struktur gleicht. Daher können RNN mit den gleichen Algorithmen wie Feedforward-Architekturen trainiert werden. Etablierte Techniken sind dabei die Gradientenverfahren. Üblicherweise wird im Bereich mehrschichtiger Netze beim überwachten Lernen (Klassifikation) die Backpropagation angewandt. Aufgabe dieses Verfahrens ist die Anpassung der Gewichtungen der einzelnen Neuronen. Dazu wird nach jedem Trainingsdurchgang ein Fehler aufgrund der Abweichung zwischen Ist- und Soll-Ausgang ermittelt. Der Fehler kann somit als Funktion $f(w)$ der Gewichtsvariable w betrachtet werden. Abbildung 18 zeigt eine solche Fehlerfunktion. Ziel der Backpropagation ist es, jene Gewichtungen zu finden, die den geringsten Fehler (w^*) aufweisen. Es wird mittels der ersten Ableitung $f'(w)$ geprüft, in welcher Richtung, also für welches w der Fehler am meisten verringert wird (steilster Abstieg).[90]

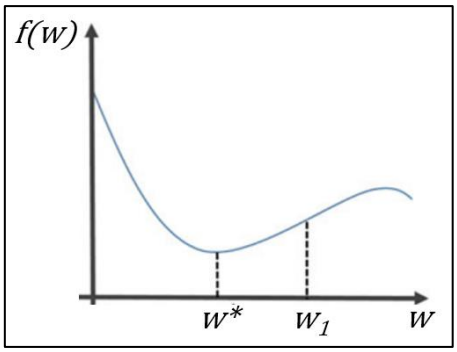


Abbildung 18: Beispielhafte Fehlerfunktion in Abhängigkeit der Gewichtsvariable w, Quelle: Sonnet (2022), S. 52 (leicht modifiziert).

Da die Anzahl der Schichten von RNNs maßgeblich vom Input abhängig sind (z. B. Anzahl der Wörter), können diese sehr tief werden, weshalb sich der Trainingsprozess meist schwierig bis unmöglich gestaltet. Ein generelles Problem, das mit der Verwendung von tiefen RNNs einhergeht, sind verschwindende oder explodierende Gradienten. Hierbei wird die Fehlerabweichung aufgrund der Tiefe des Netzes fälschlicherweise so klein (verschwindend) bzw. so groß (explodierend), dass dadurch eine korrekte Gewichtsanpassung verhindert wird.[91] Um diesem Effekt entgegenzuwirken, stellten Sepp Hochreiter und Jürgen Schmidhuber im Jahr 1997 die LSTM-Architektur vor.[92] Diese bildet im Bereich der Sprach- und Textanalyse die Grundlage für alle derzeit bekannten KI-Lösungen (z. B. Alexa, Siri) und kann auch für Audioklassifizierungsaufgaben angewendet werden.

LSTM

Die LSTM-Architektur besteht aus einzelnen LSTM-Einheiten und dient zur effizienten Langzeitspeicherung von relevanten Informationen.[93] Abbildung 19 illustriert eine solche LSTM-Einheit. Wie ersichtlich ist, besteht diese aus mehreren Komponenten, auf die im weiteren Verlauf eingegangen wird. Die dick hervorgehobene Schleife entspricht der Speicherzelle.

[90] Vgl. Sonnet (2022), S. 51 f.

[91] Vgl. Charu (2018), S. 87.

[92] Vgl. Hochreiter/Schmidhuber (1997), S. 1.

[93] Vgl. Hochreiter/Schmidhuber (1997), S. 1.

Diese beinhaltet alle relevanten Informationen, die über die Zeit erlernt werden konnten. Die LSTM-Architektur ist so entworfen, dass die Speicherzelle mit jedem Zeitschritt modifiziert wird. Dafür stehen die folgenden drei Gates zur Verfügung:[94]

- Keep-Gate – In dieser Phase wird der Eingang mit dem Ausgabe-Tensor des letzten Zeitschrittes konkateniert. Hierbei werden nur die relevanten Informationen beibehalten. Es wird eine Sigmoid-Funktion auf die Einträge angewendet, was zu einem Tensor mit Werten im Intervall [0,1] führt, wobei Werte mit ‚0' komplett vergessen und Werte mit ‚1' gänzlich behalten werden.
- Write-Gate – Im nächsten Schritt wird entschieden, welche neuen Daten auf die Speicherzelle geschrieben werden. Dazu wird zunächst eine Tanh-Funktion angewendet, die neue Speicherkandidaten festlegt, woraus ein Zwischentensor resultiert. Nach dem gleichen Prinzip wie beim Keep-Gate wird anschließend mittels der sigmoidalen Schicht entschieden, welche Werte des berechneten Zwischentensors tatsächlich für das Update der Speicherzelle verwendet werden.
- Output-Gate – Abschließend wird nach jedem Zeitschritt ein gefilterter Ausgang bereitgestellt. Die Struktur des Output-Gates ist nahezu ident mit jener des Write-Gates. Um den Ausgang zu erzeugen, wird der Zwischentensor (Tanh) mit der Bit-Tensor-Maske (Sigmoid) multipliziert.

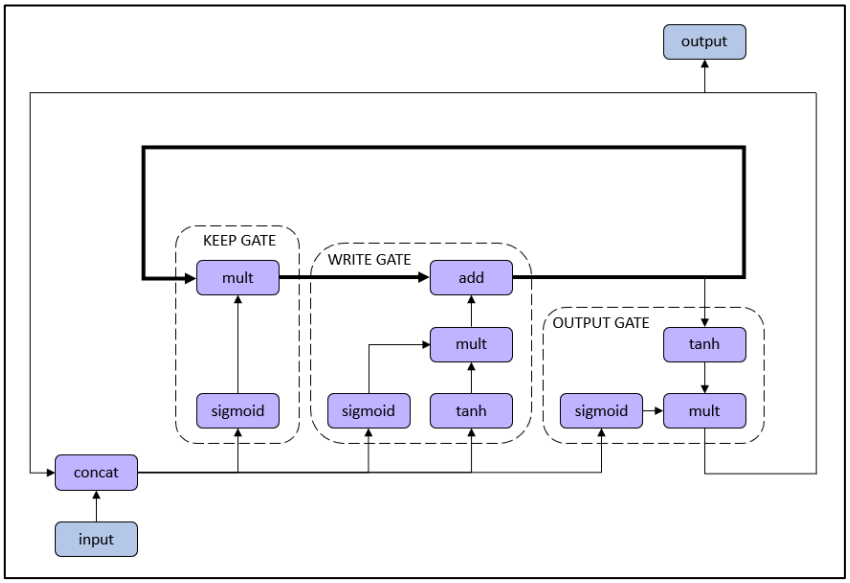


Abbildung 19: LSTM-Architektur, Quelle: In Anlehnung an Buduma (2017), S. 193.

Im Laufe der Jahre wurde eine Vielzahl an Varianten der LSTM-Architektur entwickelt. Eine seit 2014 bekannte Variante ist die GRU-Architektur, wobei GRU als Vereinfachung von LSTM betrachtet werden kann. Dabei wird einerseits auf eine interne Speicherzelle verzichtet und andererseits werden nur ein Update- und ein Reset-Gate zur Modifikation des Informationsflusses benötigt. Obwohl GRUs eine vereinfachte LSTM-Form darstellen, lässt sich keine pauschale Aussage treffen, welche Methode besser geeignet ist, um ML-Aufgaben zu lösen. Die Methoden können also als ebenbürtig betrachtet werden. Die Auswahl erfolgt anwendungsfallbezogen (experimentell).[95]

[94] Vgl. Buduma (2017), S. 178 – 183.

[95] Vgl. Charu (2018), S. 296 ff.

3.2.3 Weitere Methoden

Hybride Architekturen

In der Praxis werden die vorgestellten Methoden nicht immer isoliert voneinander betrachtet. Je nach Aufgabenstellung kann es sein, dass sich hybride Architekturen besser eignen. Im Bereich der Mustererkennung in Audiodateien gewinnt die Verwendung von Convolutional Recurrent Neural Networks (CRNNs) zunehmend an Beliebtheit.[96]

Transfer-Learning

Eine weitere Methode, die sich speziell im Bereich des DL etabliert hat, ist das Transfer-Learning. Hierbei wird das Wissen bereits vortrainierter Modelle genutzt, um dieses auf die eigene Aufgabenstellung anzuwenden. Die bereits erstellten Modelle werden also als Startpunkt für die neue Modellierungsaufgabe verwendet, wobei das Training der vortrainierten Modelle auf riesigen Benchmark-Datensätzen basiert, die eine Vielzahl unterschiedlicher Aufgabengebiete aufweisen. Dadurch soll eine allgemeine Generalisierungsfähigkeit erreicht werden. Vorteile, die sich aufgrund der Nutzung von Transfer-Learning ergeben, sind verringerte Trainingszeiten sowie eine Ermöglichung der Anwendung von DL-Algorithmen bei Datensätzen mit geringem Umfang.[97] Im Bereich der Audiomustererkennung wird dabei von Pretrained Audio Neural Networks (PAANs) gesprochen. Bekannte Architekturen, mit denen bisher gute Ergebnisse erzielt werden konnten, sind beispielsweise VGG, YAMNET sowie ResNet.[98]

Data-Augmentation

Data-Augmentation dient zur Vervielfältigung der Datenmenge. Insbesondere im Bereich des DL ist es notwendig, dass ausreichend Daten zum Training zur Verfügung stehen. Dazu werden im Zuge der Data-Augmentation leicht modifizierte Kopien des Originals erstellt.[99] Im Bereich von Audiosignalen kann das z. B. das bewusste Hinzufügen von Störungen (weißes Rauschen, rosa Rauschen) sein.

[96] Vgl. Ellis/Plumbley/Virtanen (2018), S. 138.

[97] Vgl. Wuttke 4 (o.J.), Online-Quelle [12.09.2022].

[98] Vgl. Kong u.a. (2019), S. 9; Vgl. Nanni u.a. (2021), o.S.

[99] Vgl. Ellis/Plumbley/Virtanen (2018), S. 139.

4 CLOUD-COMPUTING

Nachdem in Abschnitt 2.4 auf die Relevanz von KI-Plattformen im Zuge eines ML-Projektes eingegangen wurde, erfolgt in diesem Kapitel eine Auseinandersetzung mit der Plattform an sich. Bereits im Vorfeld dieser Arbeit wurde entschieden, dass eine cloudbasierte Plattform die Grundlage für das zu entwickelnde System bilden soll. Rein lokale Softwarelösungen, beispielsweise TensorFlow, sind zwar geeignete Lösungen, um Audiosignalanalysen mit ML durchzuführen, jedoch werden diese im weiteren Verlauf nicht weiter berücksichtigt. Hintergründe dafür sind zukunftsträchtige Technologien wie IoT und OPC UA und deren zunehmende Verankerung in der Industrie. Der Vernetzungsgedanke hinter diesen Technologien bildet die Basis für die erfolgreiche stufenlose Verknüpfung der Maschinenebene (OT) mit der Unternehmensebene (IT). Cloud-Computing-Systeme können hierbei als zentrales Sammelbecken für große Mengen von Daten dienen. Doch das alleinige Sammeln von Daten generiert noch keinen Mehrwert für das Unternehmen. Es gilt, Lösungen zu finden, um diese Daten sinnvoll zu verwerten. Eine Möglichkeit stellt dabei die Analyse von Daten unter der Verwendung von ML-Techniken dar. Neben der klassischen Methode, also der Erstellung von Modellen mittels Programmiersprachen, bieten führende Cloud-Anbieter zudem benutzerfreundlichere Werkzeuge an. Vor allem mit dem automatisierten ML (AutoML) sollen auch unerfahrene Nutzer*innen angesprochen werden. Kleine und mittlere Unternehmen können durch die Nutzung dieser Tools profitieren, denn viele KI-Use-Cases können oftmals aufgrund fehlender Fachexpert*innen nicht umgesetzt werden und müssen frühzeitig beendet werden. Im Mittelpunkt dieses Kapitels steht daher die Auswahl einer geeigneten cloudbasierten KI-Plattform sowie deren Services im ML-Bereich. Zunächst erfolgt jedoch eine kurze allgemeine Einführung.

4.1 Definition und Aufbau von Cloud-Computing-Systemen

Bereits seit den Anfängen des Internets wird der Begriff Cloud stellvertretend für das Internet verwendet. Grund dafür ist die übliche symbolische Verwendung einer Wolke in Netzwerkdiagrammen als Metapher für die Verknüpfung unterschiedlicher Netzwerke.[100] Die Idee hinter Cloud-Computing ist es, IT-Infrastruktur, Plattformen sowie verschiedene Anwendungen über ein Netzwerk bereitzustellen oder die Nutzung ebendieser Leistungen zu ermöglichen, wobei Cloud in diesem Zusammenhang andeuten soll, dass die erbrachten Leistungen des Providers im Internet bzw. Intranet abgerufen werden können.[101] Wie so oft im Bereich der IT, gibt es dabei keine generelle und einheitliche Definition. Als Einführung in das Themengebiet soll daher die folgende Beschreibung des Begriffes Cloud-Computing dienen:

„Unter Ausnutzung virtualisierter Rechen- und Speicherressourcen und moderner Web-Technologien stellt Cloud Computing skalierbare, netzwerk-zentrierte, abstrahierte IT-Infrastrukturen, Plattformen und Anwendungen als on-demand Dienste zur Verfügung. Die Abrechnung dieser Dienste erfolgt nutzungsabhängig.“[102]

[100] Vgl. Ransome/Rittinghouse (2010), S. xxvi.

[101] Vgl. Baun u.a. (2010), S. 1 f.

[102] Baun u.a. (2010), S. 4.

Wesentliches Merkmal von Cloud-Computing-Systemen ist also die Bereitstellung von abstrahierten und virtualisierten IT-Ressourcen als kostenpflichtigem Service. Beim Aufbau einer Cloud wird im Allgemeinen zwischen den drei Ebenen Software-as-a-Service (SaaS), Plattform-as-a-Service (PaaS) und Infrastructure-as-a-Service (IaaS) unterschieden.

Infrastructure-as-a-Service (IaaS)

Bei IaaS steht die Nutzung von skalierbaren Speicher-, Rechen- und Netzkapazitäten im Mittelpunkt. Eine Zuordnung geeigneter virtueller Systeme (Cloud-Instances) und Betriebssysteme (z. B. Windows-Server, Linux-Server) erfolgt eigenständig durch die Benutzer*innen. Die Anwendung ist in der Regel bedarfsorientiert. Das heißt, dass die Cloud-Instances nach individuellem Bedarf beansprucht oder freigegeben werden können. Der Provider muss garantieren, dass ausreichend Kapazitäten in Form von realer Hardware bereitgestellt werden. An welchem Ort die physische IT-Infrastruktur zur Verfügung gestellt wird, liegt ebenfalls im Verantwortungsbereich des Anbieters.[103]

Platform-as-a-Service (PaaS)

Bei PaaS werden den Anwender*innen (z. B. Software-Architekt*innen) Entwicklungsplattformen angeboten. Dabei liegt der Fokus auf der Ermöglichung einer zentralen Anwendungsentwicklung. Ferner werden Services zur Integration, Datenhaltung und Zugriffskontrolle offeriert. Bei Inanspruchnahme der angebotenen Services können Entwickler*innen unter anderem Komponenten für die darüberliegende SaaS-Ebene entwickeln.[104]

Software-as-a-Service (SaaS)

Diese Ebene bietet den Anwender*innen Services in Form von fertigen Applikationen. Die grundsätzliche Idee ist es, dass jene Applikationen, die für die optimale Unterstützung des Geschäftsprozesses relevant sind, aus einem Service-Katalog ausgewählt und entsprechend kombiniert werden können. Dazu sind die angebotenen Dienste meist standardisiert und mit Programmierschnittstellen ausgestattet.[105]

Machine-Learning-as-a-Service (MLaaS)

Bei cloudbasierten KI-Plattformen wird in der Regel von MLaaS gesprochen. Hierbei handelt es sich um Dienstleistungen, die auf den Bereich ML bezogen sind (z. B. Algorithmen, KI-Plattform).[106] Diese Lösungen können immer einer der drei Ebenen IaaS, PaaS oder SaaS zugeordnet werden.

[103] Vgl. Münzl/Pauly/Reti (2015), S. 10.

[104] Vgl. Münzl/Pauly/Reti (2015), S. 10.

[105] Vgl. Münzl/Pauly/Reti (2015), S. 11.

[106] Vgl. Blohm u.a. (2021), S. 14.

4.2 Chancen und Risiken

Es gibt viele Gründe für Unternehmen, sich für die Nutzung von Cloud-Services zu entscheiden. Jedoch dürfen die möglichen Risiken, die dadurch entstehen können, nicht vernachlässigt werden. In diesem Abschnitt werden die zentralen Vor- und Nachteile von Cloud-Computing behandelt. Im Vordergrund steht dabei die Betrachtung aus der Unternehmensperspektive.

Durch die Nutzung von IT-Ressourcen als Services wird es Unternehmen ermöglicht, Kapitalkosten in laufende Kosten zu transformieren. Insbesondere kleinere Unternehmen sowie Start-ups können von diesem Ansatz profitieren, da nicht bereits im Vorfeld Kapital für teure IT-Infrastruktur aufgebracht werden muss. Verwendete Services können je nach Bedarf an die Auftragslage angepasst werden. Aufgrund dieser flexiblen Gestaltungsmöglichkeiten können einerseits die laufenden Kosten möglichst geringgehalten werden, da Lastspitzen mittels des Versprechens der On-Demand-Verfügbarkeit ausgeregelt werden können, und andererseits wird das Risiko bzgl. Fehleinschätzung hinsichtlich einer Über- oder Unterversorgung von ausreichend Kapazitäten an den Cloud-Dienstleister ausgelagert. Neben den generellen Kosten für neue Infrastruktur entfallen zudem Arbeitsaufwände für Wartungs-, Instandhaltungs- und Administrationstätigkeiten, wodurch Unternehmen ihren Fokus vermehrt auf das Kerngeschäft richten können.[107]

Abseits der genannten wirtschaftlichen Aspekte können sich die bedarfsorientierte Nutzung sowie die Virtualisierung von IT positiv auf die Umwelt auswirken. Die benötigte Rechenleistung wird nur auf Anfrage beansprucht und verursacht somit keinen unnötigen Energieverbrauch.[108] Des Weiteren wurde von Cloud-Anbietern bereits früh erkannt, dass die Einführung von KI in Unternehmungen oftmals aufgrund fehlenden Wissens der Anwender*innen im Bereich von ML scheitert. Abhilfe schaffen Services wie ML mittels grafischer Oberflächen (Drag-and-Drop) sowie AutoML, welches sich speziell im Bereich von Cloud-Computing etabliert hat. Mit diesen Konzepten können gute Ergebnisse erzielt werden, ohne dass ein vertieftes Wissen im Fachbereich vorhanden ist.[109]

Ungeachtet der Vielzahl an Vorteilen für Unternehmen, gibt es eine Reihe von Risiken, die im Bereich von Cloud-Computing auftreten und negative Auswirkungen auf den Unternehmenserfolg haben können. Insbesondere die sicherheitstechnischen Aspekte müssen dabei beachtet werden. Die Verwendung von Cloud-Services von externen Anbietern schließt unweigerlich die Übertragung von unternehmensspezifischen Daten über das Internet mit ein. Im Falle von kritischen Daten (z. B. Kundeninformationen, Geschäftsideen) muss daher unbedingt vorab geklärt werden, wie die Daten von den externen Cloud-Dienstleistern geschützt werden. In der Regel veröffentlichen größere Cloud-Anbieter (z. B. Amazon, Google, Microsoft) zwar ihre datenschutzbezogenen Sicherheitsrichtlinien, jedoch erhalten Kunden meist keinen detaillierten Einblick, inwiefern die Daten tatsächlich geschützt sind.

[107] Vgl. Meinel u.a. (2011), S. 35.

[108] Vgl. Michelbach/Riahi (2013), S. 5 f.

[109] Vgl. Blohm u.a. (2021), S. 8 f.

Sicherheit im Bereich von Cloud-Computing basiert damit zu einem großen Teil auf Vertrauen. Ferner sind nicht nur Unternehmen Interessenvertreter*innen des Datenschutzes. Innerhalb der EU gilt es die Datenschutz-Grundverordnung (DSGVO), die sich auf den Schutz personenbezogener Daten konzentriert, einzuhalten.[110]

4.3 Auswahl einer cloudbasierten KI-Plattform

In diesem Abschnitt wird eine für die vorliegende Arbeit geeignete cloudbasierte KI-Plattform ausgewählt. Da am Markt eine unüberschaubare Anzahl von Cloud-Dienstanbietern vorhanden ist, erfolgt zunächst eine Eingrenzung der zu betrachtenden Anbieter. Im Vorfeld wird festgelegt, dass drei verschiedene Plattformen verglichen werden sollen. Hauptauswahlkriterium soll dabei der Marktanteil bezogen auf den Gesamtmarkt sein. Zusätzlich wird geprüft, ob die drei Plattformen für die Audiosignalanalyse mittels ML-Methoden geeignet sind. Da in dieser Arbeit auch mögliche Lösungen für im Bereich ML unerfahrene Unternehmen aufgezeigt werden sollen, muss der Cloud-Dienst neben generellen Programmiermöglichkeiten auch den Bereich AutoML abdecken.

Abbildung 20 visualisiert den Marktanteil in Bezug auf den Gesamtmarkt im Bereich IaaS, PaaS und private Clouds. Die Synergy Research Group schätzt, dass Unternehmen im Jahr 2021 insgesamt 178 Mrd. US-Dollar für Cloud-Dienstleistungen investierten. Die drei größten Anbieter sind Amazon mit 32 bis 33 %, Microsoft mit 21 % und Google mit 9 % Marktanteil. Gemeinsam dominieren diese den Markt mit insgesamt ca. 63 % Marktanteil, was Ausgaben von geschätzt 112 Mrd. US-Dollar entspricht. Weiters ist ersichtlich, dass Microsoft, Google und Alibaba seit 2017 zum kontinuierlich führenden Anbieter Amazon aufholen, während IBM seit 2017 einen großen Teil der Marktanteile verloren hat.[111]

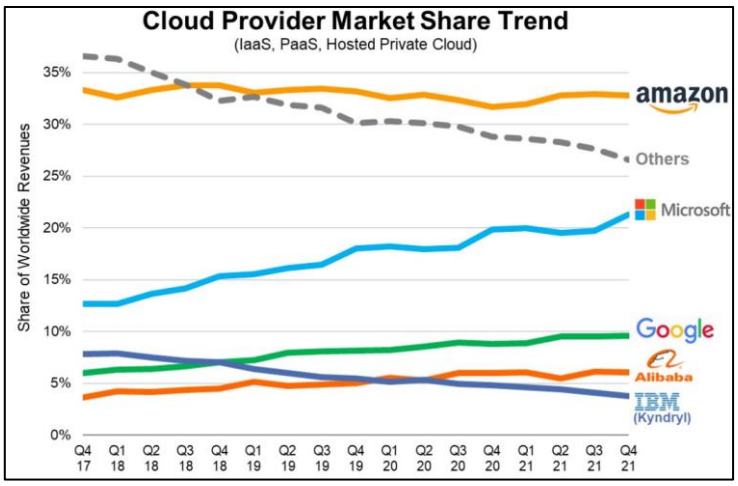


Abbildung 20: Marktanteil von Cloud-Anbietern 2017–2021, Quelle: Synergy Research Group (2022), Online-Quelle [12.09.2022].

Zudem wird abgeschätzt, ob die drei Marktführer Amazon, Microsoft und Google für die Umsetzung der Aufgabenstellung geeignet sind. Grundlage dafür ist die Studie „Gartner Magic Quadrant for Cloud AI Developer Services“ von Gartner, in der alle drei Anbieter als ‚Leader‘ klassifiziert werden.

[110] Vgl. Meinel u.a. (2011), S. 41 f.

[111] Vgl. Synergy Research Group (2022), Online-Quelle [12.09.2022].

‚Leader' besitzen unter anderem Kernkompetenzen im Bereich von Sprachverarbeitung, Bildverarbeitung und AutoML.[112] Sowohl Kompetenzen im Bereich der Spracheverarbeitung als auch in der Bildverarbeitung (siehe Abschnitt 3.1.1) sind für die Audiosignalanalyse mit ML von Bedeutung, weshalb die KI-Plattformen von Amazon, Microsoft und Google für den Vergleich ausgewählt werden.

Abbildung 21: Gartner Magic Quadrant for Cloud AI Developer Services, Quelle: Gartner (2021), Online-Quelle [12.09.2022].

4.3.1 Allgemeiner Vergleich der Anbieter

Zunächst werden die zentralen Eckdaten zu den jeweiligen Anbietern dargestellt:[113] [114]

	Microsoft	**Amazon**	**Google**
Cloud-Plattform	Azure	AWS	GCP
ML-Plattform	Azure ML	AWS SageMaker	Vertex AI
Veröffentlichung ML-Plattform	2014	2017	2021

Tabelle 1: Eckdaten der Cloud-Anbieter Amazon, Google und Microsoft. Quelle: Eigene Darstellung.

[112] Vgl. Gartner (2021), Online-Quelle [12.09.2022].

[113] Vgl. Amazon 1 (2022), Online-Quelle [12.09.2022]; Vgl. Google 1 (2022), Online-Quelle [12.09.2022]; Vgl. Microsoft Corporation 3 (2022), Online-Quelle [12.09.2022].

[114] Vgl. Amazon 2 (2022), Online-Quelle [12.09.2022]; Vgl. Google 2 (2022), Online-Quelle [12.09.2022]; Vgl. Microsoft Corporation 4 (2022), Online-Quelle [12.09.2022].

Als Entscheidungshilfe bei der Auswahl einer geeigneten cloudbasierten KI-Plattform für die Audiosignalklassifizierung mittels ML dient einerseits eine Studie des KI-Fortschrittzentrums des Fraunhofer-Instituts[115] und andererseits wird ein Praxisfall durchgeführt, der anhand von Kriterien und einer Entscheidungsmatrix bewertet wird. Die Studie des KI-Fortschrittzentrums beschäftigt sich umfangreich mit dem Vergleich unterschiedlicher cloudbasierter KI-Plattformen. Unter anderem werden allgemeine Kriterien der KI-Plattformen von Amazon, Google, Microsoft und IBM verglichen. Allgemeine Kriterien beziehen sich hierbei auf die relevantesten funktionellen, technischen und geschäftlichen Eigenschaften, die für Unternehmen von Bedeutung sind. Um einen Überblick der verglichenen Kriterien zu verschaffen, werden hier einige beispielhaft aufgelistet:[116]

- Funktionsumfang: AutoML, Spracherkennung, Objekterkennung in Bildern etc.,
- Technische Kriterien: Skalierbarkeit der Hardwareressourcen, Datenschutz, standardisierte Schnittstellen zu anderen IT-Systemen sowie Programmiermöglichkeiten (Python) etc.,
- Geschäftliche Kriterien: Standortauswahl (z. B. Datenhaltung in der EU), Preistransparenz, Support-Umfang.

Bei Betrachtung der Studienergebnisse wird deutlich, dass zwischen den drei Anbietern Amazon, Google und Microsoft hinsichtlich der angebotenen Dienste und Lösungen keine wesentlichen Unterschiede erkennbar sind. Jeder der Anbieter ermöglicht es, die Kundenanforderungen und -bedürfnisse bei der Durchführung von KI-Projekten zufriedenstellend und gleichwertig abzudecken, weshalb der im Folgenden durchgeführte Praxisfall als Entscheidungsgrundlage für eine geeignete cloudbasierte KI-Plattform im Bereich der Audioklassifizierung dienen soll.

4.3.2 Use-Case-Vergleich der Anbieter

Zunächst erfolgt eine Einführung in das Praxisbeispiel. Sinngemäß wird eine Audioklassifizierung durchgeführt. Da alle Plattformen (Amazon, Google, Microsoft) die plattformunabhängige Modellerstellung mittels Programmiersprachen ermöglichen, werden die jeweiligen AutoML-Angebote für die Umsetzung verwendet. Neben den Ergebnissen, die mit dem erstellten Modell erzielt werden können, steht daher bei der Bewertung insbesondere die einfache Handhabung im Vordergrund. Für diesen Testfall wird ein frei zur Verfügung stehender Datensatz der ML-Wettbewerbsplattform Kaggle verwendet.[117] Dieser entspricht nicht dem in 1.2 erläuterten BirdCLEF2022-Datensatz. Der Datensatz enthält 7695 Audiodateien mit insgesamt 41 verschiedenen Klassen. Die Aufgabe ist somit eine Multi-Klassen-Klassifizierungsaufgabe. Die Dateien lassen sich Klassen wie Musikinstrumente, Tiergeräusche oder von Menschen erzeugte Geräusche zuordnen. Zudem liegt eine CSV-Datei bei, die den Bildern des Trainingsdatensatzes Klassen zuordnet (überwachtes Lernen), z. B. Name der Bilddatei ist 00ad7068.wav → in der CSV-Datei gibt es einen Zeileneintrag mit 00ad7068.wav, Akustik-Gitarre.

[115] Vgl. Blohm u.a. (2021), o.S.

[116] Vgl. Blohm u.a. (2021), S. 35 – 40.

[117] Vgl. Kaggle 1 (2022), Online-Quelle [12.09.2022].

Abbildung 22 zeigt die geplante Vorgehensweise bei der Durchführung des Testfalls. Zunächst werden die Audiodateien mit .wav-Format in ein Mel-Spektrogramm mit .jpg-Format umgewandelt. Dieser Vorgang findet im Zuge des Testbeispiels am lokalen Rechner statt und wird hier nicht genauer erläutert. Wie eine Umwandlung einer Audiodatei in ein Spektrogramm implementiert werden kann, ist in Abschnitt 6.2 detailliert beschrieben. Anschließend werden die Bilddateien vom lokalen Rechner auf einen Cloud-Datenspeicher hochgeladen (IaaS) und mittels der jeweiligen KI-Plattform (PaaS) und automatisierten Services zur Datenverarbeitung (SaaS) analysiert. Abschließend werden die Ergebnisse betrachtet und es wird geprüft, welche Möglichkeiten zur Weiterverwendung des erstellten und trainierten Modells existieren.

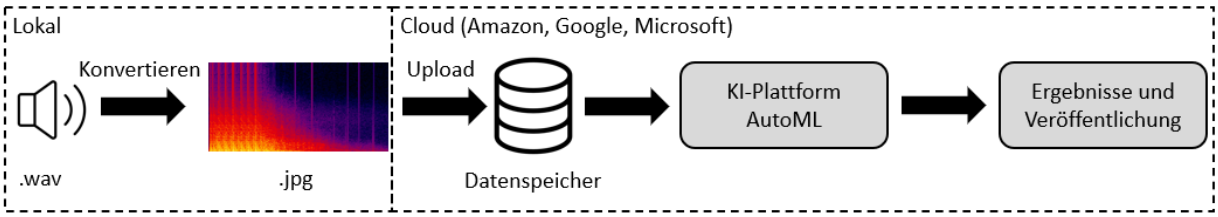


Abbildung 22: Vorgehensweise bei der Durchführung des Testfalls, Quelle: Eigene Darstellung.

Die Bewertung erfolgt anhand von vorab festgelegten Kriterien. Als Unterstützung zum Finden und Definieren relevanter Kriterien für MLaaS-Lösungen wird ein Leitfaden zur Auswahl von Softwarelösungen im Cloud-Plattform-Bereich verwendet.[118]

Kriterienliste:

- Datenimport – Wie einfach bzw. schwierig gestaltet sich der Datenimport?
- Anwendungsfreundlichkeit – Wie übersichtlich ist der AutoML-Service? Finden sich Nutzer*innen schnell zurecht? Gibt es Benachrichtigungsmöglichkeiten via Mail nach Wartezeiten (z. B. Training)?
- Ergebnisse – Gemessen werden die Ergebnisse anhand der Metriken Precision und Recall.[119]
    - Precision – Liegt im Intervall [0, 1]. Je näher der Wert bei 1 liegt, desto eher ist die positive Vorhersage korrekt.
    - Recall – Liegt im Intervall [0, 1]. Je näher der Wert bei 1 liegt, desto häufiger werden positive Proben erkannt.
- Kosten – Welche Gesamtkosten fallen für die Datenspeicherung, das Training und die Bereitstellung des trainierten Modells an?
- Transparenz (Algorithmen) – Gibt es eine Möglichkeit nachzuverfolgen, welche Algorithmen verwendet wurden (Modell, Training)?
- Exportmöglichkeiten – Zu welchen anderen Softwaresystemen gibt es Schnittstellen für die Einbindung der erstellten Lösung?
- Programmierkenntnisse – Sind für die Durchführung des Testfalls Programmierkenntnisse erforderlich?

[118] Vgl. Hanussek/Papp (o.J.), o.S.

[119] Vgl. Microsoft Corporation 5 (2022), Online-Quelle [12.09.2022].

Im Folgenden werden die Beobachtungen bei der Implementierung des Testfalls beschrieben und es wird auf die relevantesten Erkenntnisse eingegangen:

Amazon

Zur automatisierten Klassifizierung von Bilddateien (Mel-Spektrogramm) wird der Service Amazon Rekognition verwendet.[120] Voraussetzung für die Nutzung ist ein AWS-Konto. Zu Beginn muss ein Projekt erstellt werden, wofür ein S3-Bucket vorhanden sein muss. Als S3 wird der Speicher-Service von Amazon bezeichnet. Im Anschluss wird ein Datensatz angelegt. Rekognition bietet eine Funktion, die den hochgeladenen Datensatz automatisch in Trainingsdaten (80 %) und Testdaten (20 %) splittet. Importiert werden können Daten entweder von einem lokalen PC oder von einem S3-Bucket. Für das automatisierte Labeling der Daten stehen zwei Varianten zur Verfügung. Rekognition erkennt selbstständig Ordnerstrukturen (jeder Ordner ist eine Klasse) und labelt die zugehörigen Dateien mit den Ordnernamen. Liegen die Daten unstrukturiert vor, so können diese mit der beiliegenden CSV-Datei (siehe Anfang Abschnitt 4.3.2) automatisch gelabelt werden. Zwar wird das .csv-Format nicht direkt unterstützt, jedoch kann die Datei mit einem von Amazon zur Verfügung gestellten Python-Skript in das Dateiformat .manifest umgewandelt werden.[121] Dies erfordert jedoch zumindest Grundkenntnisse in Programmierung und eine Umgebung, um Python-Skripte ausführen zu können. Anschließend kann ein Trainingsvorgang gestartet werden. Eine Trainingsdauer ist nicht einstellbar (keine Kostenkontrolle). Es wird nur ein Trainingsdurchgang durchgeführt. Zudem gibt es keine Auswahlmöglichkeit für eine E-Mail-Benachrichtigung bei Abschluss des Trainings. Abschließend werden die Ergebnisse dargestellt (siehe Abbildung 23). Eine Exportmöglichkeit zu anderen Systemen (nicht Amazon) bzw. eine Dokumentation dazu kann nicht vorgefunden werden.

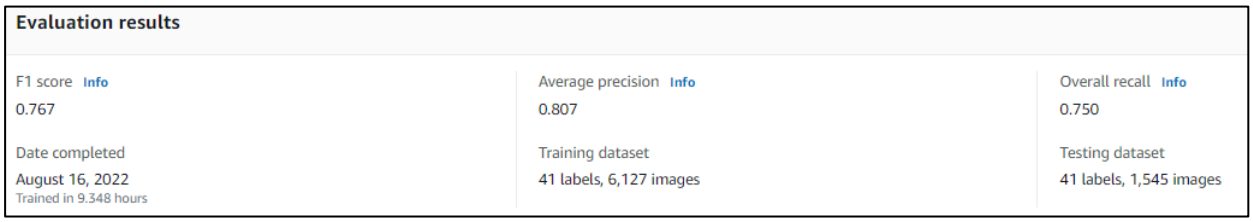


Abbildung 23: Ergebnisse Testfall Audiosignalklassifizierung – Amazon Rekognition, Quelle: Eigene Darstellung.

Google

Google stellt in der Vertex-AI-Cloud-Plattform den Service AutoML Vision zur Verfügung.[122] Die Vorgehensweise bei der Projektdurchführung verhält sich ähnlich wie bei Amazon Rekognition. Nachdem ein Konto und ein Projekt angelegt wurden, kann ein Datensatz erstellt werden. Importiert werden können Bilder vom PC oder von einem Google-Bucket. Das automatisierte Labeling funktioniert mit Ordnerstrukturen (je Ordner eine Klasse) oder direkt mit der CSV-Datei (siehe Abbildung 24 links). Für die Durchführung werden keine Programmierkenntnisse benötigt.

[120] Vgl. Amazon 3 (2022), Online-Quelle [12.09.2022].

[121] Vgl. Amazon 4 (o.J.), Online-Quelle [12.09.2022].

[122] Vgl. Google 3 (2022), Online-Quelle [12.09.2022].

Für das Training wird der Datensatz automatisch in Trainingsdaten (80 %) und Testdaten (20 %) gesplittet. Zudem gibt es die Möglichkeit, eine maximale Trainingsdauer zu wählen. Für dieses Projekt empfiehlt Google eine Trainingsdauer von sechs Stunden (Amazon: 9,348 Stunden). Bei Abschluss des Trainings wird eine Benachrichtigung per E-Mail versendet. Das Modell kann sowohl für die interne Benutzung (GCP) als auch für externe Programme (z. B. TensorFlow, Core ML, Coral) verwendet werden. Die Ergebnisse sind auf der rechten Seite in Abbildung 24 ersichtlich.

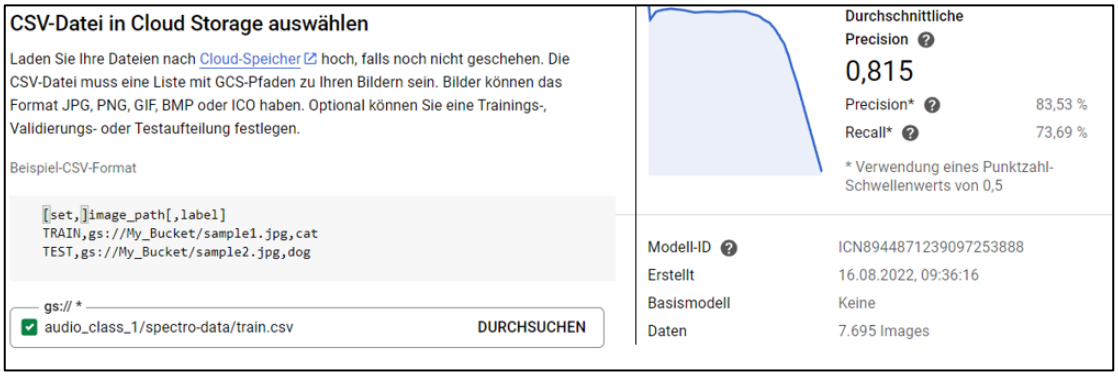


Abbildung 24: Datenimport und Ergebnisse Testfall Audioklassifizierung – Google AutoML Vision, Quelle: Eigene Darstellung.

Microsoft

Bei Microsoft wird der Service Custom Vision für die Projektumsetzung verwendet.[123] Nachdem ein Konto und ein Projekt erstellt wurden, kann ein Datensatz angelegt werden. Daten können nur direkt vom PC importiert werden. Zudem gibt es keine Option für das automatisierte Labeling. Da die Bilder unstrukturiert (ohne Ordnerstrukturen) vorliegen, müssen diese entweder manuell oder mittels eigener Software sortiert werden. Aufgrund der Anzahl der Bilder (7695) wird in diesem Fall eine C#-Software erstellt, die anhand der Informationen aus der CSV-Datei die Bilder entsprechend der 41 Klassen sortiert. Beim Hochladen müssen dennoch 41 Upload-Vorgänge (je Klasse einmal) getätigt werden, was im Vergleich zu Amazon und Google einem deutlichen Mehraufwand entspricht. Die Trainingszeit kann bestimmt werden. Custom Vision empfiehlt für diesen Datensatz eine Trainingsdauer von zwei Stunden. Ferner gibt es eine Benachrichtigungsfunktion per Mail. Das trainierte Modell kann für verschiedene externe Systeme bereitgestellt werden (z. B. TensorFlow, Core ML, ONNX).

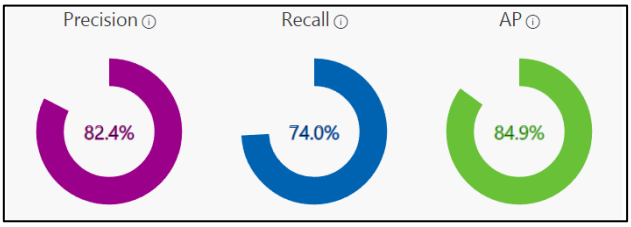


Abbildung 25: Ergebnisse Testfall Audioklassifizierung – Azure Customer Vision, Quelle: Eigene Darstellung.

[123] Vgl. Microsoft Corporation 6 (2022), Online-Quelle [12.09.2022].

Kostenübersicht und Bewertung

Die folgende Tabelle demonstriert beispielhaft die Kosten, die für die benötigten Services aufzuwenden sind. Eine Übersicht bieten die Preisrechner der jeweiligen Anbieter.[124]

	Amazon	**Google**	**Microsoft**
Speicher pro GB und pro Monat	5 GB/Monat für die ersten 12 Monate kostenlos, ansonsten regional, Beispiel: Frankfurt 0,0245 $	5 GB/Monat dauerhaft kostenlos, ansonsten regional, Beispiel: Frankfurt 0,023 $	Regional, Beispiel: Westdeutschland 0,02 $
AutoML-Service-Training	1,12 $ pro Stunde	3,465 $ pro Stunde	10 $ pro Stunde
Startguthaben (Neukunden)	12 Monate kostenlos (mit Einschränkungen), danach weiterführend viele Services kostenlos nutzbar.	300 $ zur Verwendung in drei Monaten, danach weiterführend viele Services kostenlos nutzbar.	200 $ zur Verwendung in 30 Tagen, danach weiterführend viele Services kostenlos nutzbar.

Tabelle 2: Kostenvergleich für die im Testfall genutzten Services, Quelle: Eigene Darstellung.

Es fällt auf, dass die Trainingskosten bei Microsoft deutlich höher als bei Amazon und Google sind. Jedoch gleicht sich dies aus, da Microsoft mit nur zwei Stunden bei gleichwertigen Ergebnissen die kürzeste Trainingsdauer vorweisen konnte. Die tatsächlich anfallenden Kosten können nicht nachvollzogen werden, da bei allen drei Anbietern die kostenlosen Kontingente zur Durchführung der Aufgabenstellung verwendet werden können. Abschließend erfolgt die Bewertung der vorab definierten Kriterien auf Basis der beschriebenen Vorgehensweisen. Wie ersichtlich ist, erzielt Google die höchste Punktezahl. Dies liegt vor allem an der einfachen Handhabbarkeit bei der Durchführung des Testfalls, weshalb Google bzw. die Plattform Vertex AI für diese Arbeit als Cloud-Plattform gewählt wird.

Anbieter	**Amazon**	**Google**	**Microsoft**
Kriterium	Punkte	Punkte	Punkte
Datenimport	6	6	3
Anwendungsfreundlichkeit	4	6	6
Ergebnisse	4	5	6
Transparenz (Algorithmen)	0 nicht ersichtlich	0 nicht ersichtlich	0 nicht ersichtlich
Exportmöglichkeiten	0 nicht möglich	6	6
Programmierkenntnisse	5	6	4
Summe Punkte	19	**28**	25

Tabelle 3: Auswahl einer KI-Plattform, Punkte 6 = Kriterium vollständig erfüllt, 0 = Kriterium nicht erfüllt, Quelle: Eigene Darstellung.

[124] Vgl. Amazon 5 (2022), Online-Quelle [12.09.2022]; Vgl. Google 4 (2022), Online-Quelle [12.09.2022]; Vgl. Microsoft Corporation 7 (2022), Online-Quelle [12.09.2022].

4.4 Google Vertex AI

In diesem Abschnitt werden die Funktionsumfänge der cloudbasierten KI-Plattform Google Vertex AI thematisiert. Wie in Abbildung 26 ersichtlich ist, sind die angebotenen Services umfangreich und können nicht im Detail erläutert werden. Für die meisten Prozessschritte (z. B. Training, Feature-Engineering) stehen eigene Services zur Verfügung. Die Schritte können somit sauber voneinander getrennt werden, was die rollenübergreifende Entwicklung vereinfacht. Ferner kann beobachtet werden, welche Prozessschritte vom zuvor verwendeten Google Vision (AutoML) abgedeckt werden.

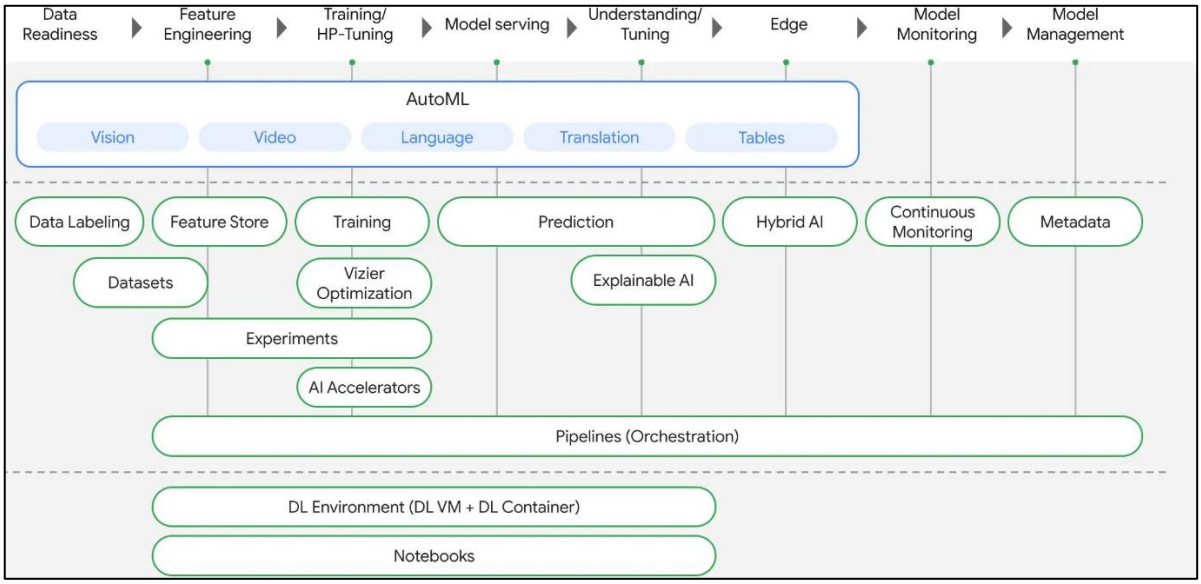


Abbildung 26: Funktionsumfang Google Vertex AI, Quelle: Google 5 (2021), Online-Quelle [12.09.2022].

Abbildung 27 (links) stellt das Hauptauswahlmenü von Vertex AI dar. Im Folgenden wird auf die einzelnen Punkte genauer eingegangen:

1. Das Dashboard bietet eine Gesamtübersicht über die letzten Aktivitäten.
2. Unter Datensätze können Trainings- und Testdatensätze für diverse Aufgabengebiete (Bild, Tabellen, Video, Text) angelegt werden. Bereits bei Erstellung des Datensatzes muss die Aufgabe (z. B. Klassifizierung) definiert werden. Datensätze sowie alle anderen Ressourcen, die Speicher benötigen, können auf einem Data-Storage (z. B. Google-Bucket) gespeichert werden.
7. Unter Training können entweder benutzerdefinierte oder AutoML-basierende Trainingsjobs erstellt werden.
9. Model Registry ist die Modellverwaltungseinheit von Vertex AI. Es ist ersichtlich, welche Modelle im Projekt vorhanden sind und in welchem Zustand sich diese befinden (Training, bereitgestellt etc.). Zudem können Auswertungen des jeweiligen Modells von Trainings-, Test- und Betriebsprozessen betrachtet werden.
10. Das Menü ‚Endpunkte' gibt eine Übersicht über alle online bereitgestellten Modelle. Bereitgestellte Modelle können beispielsweise über eine Python-Programmierschnittstelle für Betriebszwecke verwendet werden.

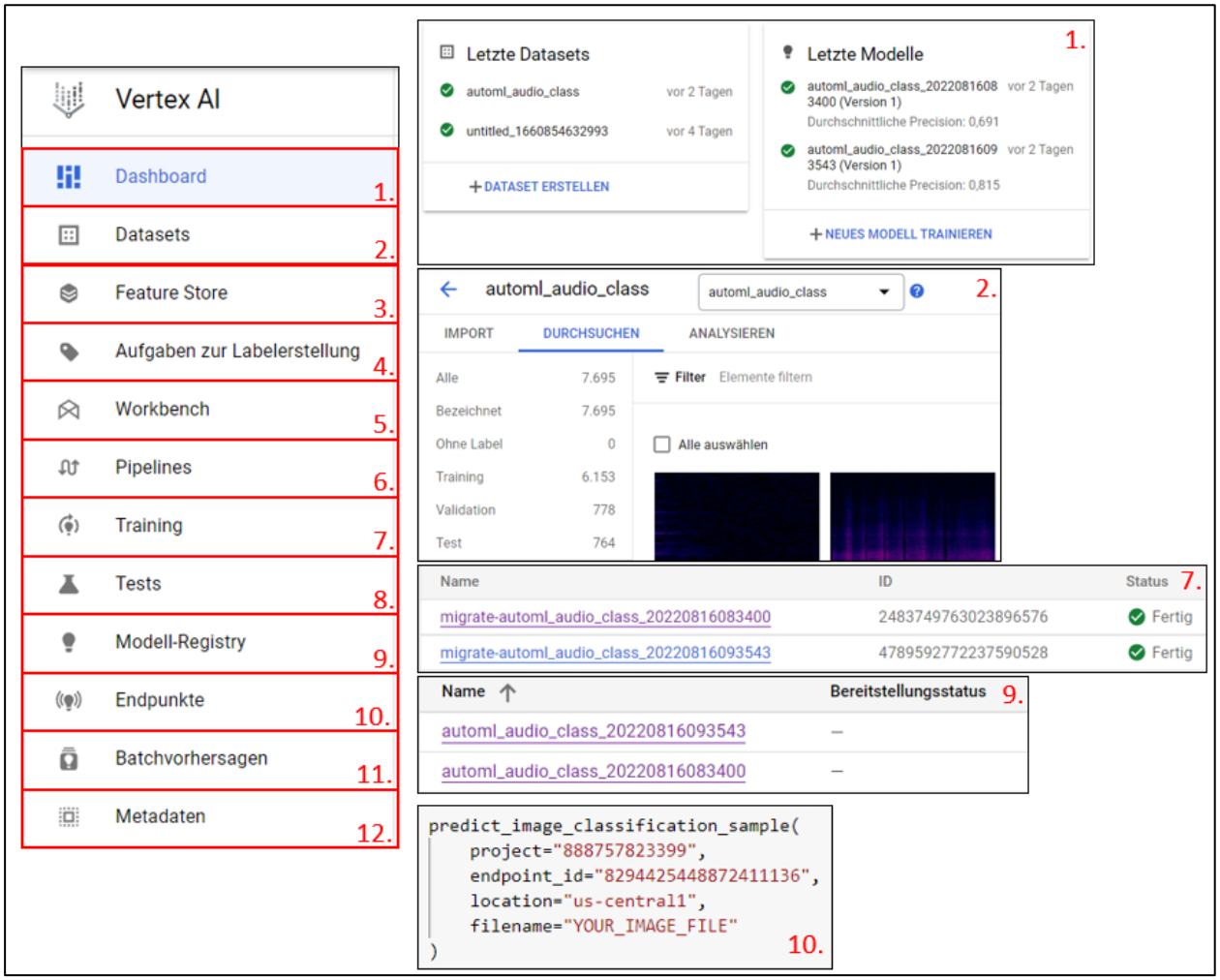


Abbildung 27: Übersicht Vertex AI, Quelle: Eigene Darstellung.

3. Der Feature-Store unterstützt Data-Scientists beim Datenvorverarbeitungsprozess (Feature-Engineering). Im Zuge der Datenvorverarbeitung werden Daten so aufbereitet, dass ML-Modelle mit einem qualitativ hochwertigen Input versorgt werden. Dies ist oftmals ein komplexer Prozess, der eine anspruchsvolle Logik voraussetzt. Der Feature-Store hilft dabei, die erstellten Features zu organisieren, und erhöht somit die Wiederverwendbarkeit des im Zuge des Feature-Engineerings entwickelten Codes.[125]
4. Der Menüpunkt ‚Aufgaben zur Labelerstellung' unterstützt Entwickler*innen beim Beschriften von Daten. Zudem können Mitarbeiter*innen von Google für das Datenlabeling engagiert werden.
5. Workbench ist eine Organisationseinheit zum Verwalten von Notebook-Instanzen. Beim Erstellen einer Notebook-Instanz wird eine virtuelle Maschine (siehe Abbildung 28 links) instanziiert. Rechenleistung und Speicher sind dabei skalierbar. Neben den Computing-Ressourcen werden JupyterLab-Dienste zur Verfügung gestellt.

[125] Vgl. Google 6 (2022), Online-Quelle [12.09.2022].

JupyterLab ist eine interaktive Entwicklungsumgebung für Notebooks (siehe Abbildung 28 rechts) und ermöglicht es, ML-Aufgaben mittels Programmiersprachen (z. B. Python) zu lösen, wobei eine Umgebung zum Interpretieren und Ausführen von Code bereitgestellt wird.[126] Notebooks werden im Verlauf dieser Arbeit hauptsächlich für die Durchführung der Experimente verwendet.

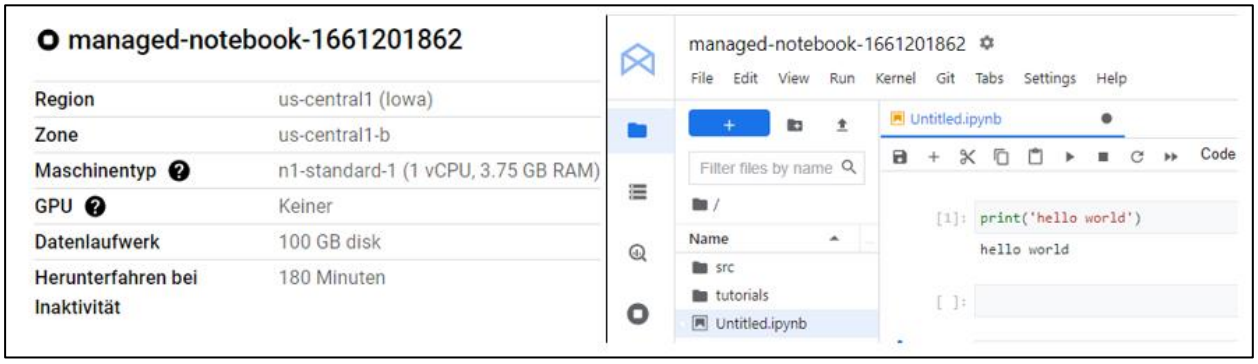


Abbildung 28: Beispiel einer Notebook-Instanz (links) und JupyterLab-Entwicklungsumgebung (rechts), Quelle: Eigene Darstellung.

6. Wie schon unter 2.4.2 erwähnt, unterstützen Pipelines bei der Automatisierung eines ML-Workflows. Abbildung 29 zeigt eine solche Pipeline. Einzelne eben erläuterte Fragmente, wie Datasets, ein Trainingsjob oder eine Modellevaluierung, können übersichtlich und logisch über definierte Schnittstellen miteinander verknüpft werden.

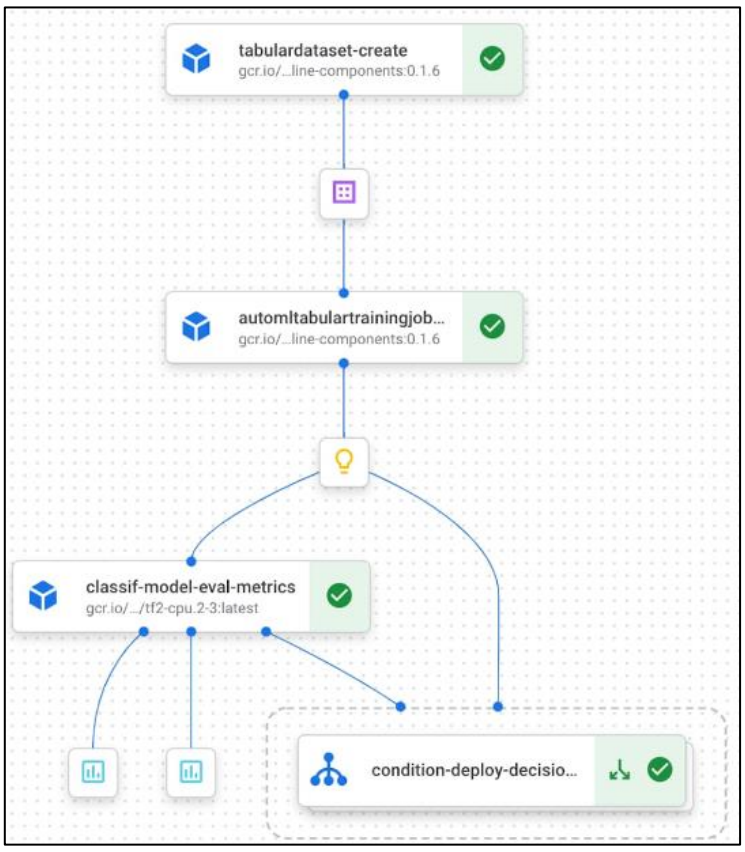


Abbildung 29: Vertex AI Pipeline, Quelle: Google 7 (2021), Online-Quelle [12.09.2022].

[126] Vgl. Jupyter (2022), Online-Quelle [12.09.2022].

8. Zur Unterstützung beim Finden des bestmöglichen Modells für das zu lösende ML-Problem kann unter ‚Tests' mit dem Service Vertex AI Experiments ein ML-Framework-Vergleich (z. B. TensorFlow, PyTorch) durchgeführt werden.[127]
11. Batchvorhersagen können für eine asynchrone Modellanfrage verwendet werden. Dieser Service kann herangezogen werden, wenn keine sofortige Antwort auf die Anfrage notwendig ist. Vorteil gegenüber Online-Vorhersagen (synchron, sofortige Antwort) ist, dass kein Modell bereitgestellt werden muss (keine Kosten).[128]
12. Unter Vertex AI Metadaten können die Metadaten eines ML-Systems aufgezeichnet werden. Dies können beispielsweise Parameterwerte oder Laufzeiten (z. B. Training) sein, die vor allem zur Unterstützung einer wissenschaftlichen Vorgehensweise dienen (Dokumentation, Analyse etc.).[129]

[127] Vgl. Google 8 (2022), Online-Quelle [12.09.2022].

[128] Vgl. Google 9 (2022), Online-Quelle [12.09.2022].

[129] Vgl. Google 10 (2022), Online-Quelle [12.09.2022].

5 ANFORDERUNGSANALYSE UND KONZEPTIONIERUNG

In den vorhergehenden Abschnitten wurde auf die theoretischen Grundlagen für die Klassifizierung von Audiosignalen sowie des Cloud-Computings eingegangen. Im weiteren Verlauf werden diese Grundlagen anhand eines Beispieldatensatzes angewandt. Die Auswahl geeigneter KI-Modelle für die gegebenen Daten und die zu entwickelnde Lösung erfolgt aufgrund von Auswertungen diverser Experimente. In diesem Kapitel soll im Detail geklärt werden, wie die praktische Aufgabenstellung durchgeführt wird und welche Anforderungen an die entwickelte Lösung gestellt werden. Zunächst erfolgt eine Einführung bzgl. des verwendeten Datensatzes.

5.1 Einführung Datensatz

Für die Durchführung der Experimente wird der Datensatz des Wettbewerbes BirdCLEF2022 der ML-Plattform Kaggle verwendet. Dieser enthält insgesamt 14 858 Dateien, wovon 14 852 Audioaufnahmen (.ogg-Format) von Vogelgeräuschen sind, die für das Training und für die Testphase verwendet werden können. Insgesamt kann der Datensatz als komplex betrachtet werden. Ein wesentlicher Grund dafür ist, dass 152 verschiedene Klassen (Vogelarten) enthalten sind. Infolgedessen sind nur wenige Daten je Klasse vorhanden, was das Erlernen von Mustern erschwert. Zudem ist der Datensatz nicht ausbalanciert. Den einzelnen Klassen sind teils viele (max. 500) und teils wenige Dateien (min. drei) zugeordnet. Ferner stammen die Aufnahmen von Hobbybiologen, weshalb die Aufnahmebedingungen hinsichtlich Qualität, Kanäle, Umgebungsbedingungen, Länge der Aufnahmen und Distanz zum Aufnahmeobjekt stark variieren. Beiliegend ist eine CSV-Datei (train_metadata.csv), die jeder Audiodatei eine Klasse (Vogelart) zuweist.[130] Alle im weiteren Verlauf durchgeführten Tests können somit dem überwachten Lernen zugeordnet werden.

Der Datensatz wird vom Cornell Lab of Ornithology zur Verfügung gestellt.[131] Mit dieser Arbeit wird nicht am Wettbewerb teilgenommen, weshalb auf die für den Wettbewerb geltenden Regeln nicht weiter eingegangen wird. Außerdem werden die weiteren fünf beiliegenden Dateien nicht beachtet. Der Datensatz kann für akademische Zwecke frei verwendet werden.[132]

Diskussion zur Auswahl des Datensatzes

Obwohl Vogelgeräusche nicht direkt mit industriellen Anwendungen in Verbindung gebracht werden können, wurde der erläuterte Datensatz aus folgenden Gründen im Vorfeld der Arbeit ausgewählt. Einerseits gibt es im Bereich von Audiosignalen nur wenige Datensätze, die frei verwendet werden können, und andererseits ist jener von BirdCLEF2022 aufgrund seiner Komplexität ideal, um die State-of-the-Art-DL-Methoden zu testen. Parallelen zu realen industriellen Anwendungen sind hier beispielsweise die Unstetigkeit der Geräusche in den Aufnahmen sowie die unterschiedlichen Umgebungsgeräusche bzw. -bedingungen. Des Weiteren enthält der Datensatz nur wenige Audiodateien je Klasse.

[130] Vgl. Kaggle 2 (2022), Online-Quelle [17.11.2022].

[131] Vgl. Kaggle 3 (2022), Online-Quelle [17.11.2022].

[132] Vgl. Kaggle 4 (2022), Online-Quelle [17.11.2022].

Damit soll vor allem die Anwendbarkeit von DL-Methoden auf Datensätze mit geringerem Umfang demonstriert werden. Wie bereits erwähnt, ist das Beschriften von Daten ein nicht zu vernachlässigender Kostentreiber eines KI-Projektes. Mit maximal 500 Beispielen je Klasse kann hier ein auch für KMU realistisches und zumutbares Szenario abgebildet werden.

Zudem fällt nach ersten Tests auf, dass die Trainingszeiten mit den zur Verfügung stehenden Hardwareressourcen (Kosten) relativ lange sind. Da im Fokus der Experimente die Anwendung vieler unterschiedlicher Modelle bzw. Methoden und nicht die Analyse von möglichst vielen Dateien steht, wird der Datensatz vorab auf neun Klassen reduziert. Die Auswahl erfolgt zufällig, jedoch muss die betreffende Klasse zumindest 50 Audiodateien enthalten. Dies vereinfacht die Aufgabenstellung, da die Balance des Datensatzes erhöht wird. Im Zuge der Experimente müssen viele Parameter ermittelt und konfiguriert werden. Um Trends erkennen zu können, werden deshalb Minitests durchgeführt, wobei diese jeweils mit vier Klassen realisiert werden. Die Analyse der Daten erfolgt in Abschnitt 6.1.

5.2 Anforderungen

Für den zu entwickelnden Audiosignalklassifizierer werden zunächst Anforderungen definiert. Diese werden in die drei Bereiche System-, Funktions- und Qualitätsanforderungen gegliedert und im Folgenden beschrieben:

Systemanforderungen

- Cloudbasiert – Sämtliche verwendete Hard- und Softwareressourcen (IaaS, PaaS, SaaS) müssen von der Google-Cloud-Plattform unterstützt oder zur Verfügung gestellt werden. Als KI-Plattform muss Google Vertex AI verwendet werden. Entwicklungstätigkeiten sind ausschließlich über Cloud-Services zu realisieren.
- Software – Die zu entwickelnden Softwarekomponenten für Experimente sowie zur Integration der entwickelten Modelle in das Unternehmenssystem müssen mit einer für ML-Projekte üblichen Programmiersprache erstellt werden (Python oder R). Ferner muss ein gängiges Framework für die Entwicklung gewählt werden (TensorFlow, Keras, PyTorch).
- Transparenz – Es muss eine Projektumgebung geschaffen werden, die eine domänenübergreifende Entwicklung (Data-Engineer, Data-Scientist etc.) ermöglicht.

Funktionsanforderungen

- ML-Phasen – Die Lösung muss die Crisp-DM-Phasen Datenanalyse, Datenvorbereitung, Modellierung, Training, Evaluierung und Modellveröffentlichung abdecken.
- Klassifizierer – Die Audiosignale sollen zuverlässig den jeweils korrekten Klassen zugeordnet werden.
- Modellbereitstellung – Die Modelle sind so bereitzustellen, dass eine On-Demand-Anfrage aus der Produktions- bzw. Maschinenebene beantwortet werden kann.

Qualitätsanforderungen

- Softwarearchitektur – Erstellte Softwarekomponenten müssen wiederverwendbar, erweiterbar und wartbar gestaltet sein. Zwischen einzelnen Komponenten sind klar definierte Schnittstellen festzulegen.
- Nachvollziehbarkeit – Die Ergebnisse von Trainings-, Validierungs- und Evaluierungsphasen müssen für die Nachvollziehbarkeit der Parametereinstellungen entsprechend dokumentiert werden.
- Effizienz – Die Zeit zwischen Vorhersageanfrage und Vorhersageantwort (Latenzzeit) muss in einem für Produktionsanlagen vertretbaren Zeitrahmen liegen.
- Robustheit – Eine Überanpassung der Modelle auf die Trainingsdaten ist zu vermeiden. Die entwickelten Modelle müssen das erlernte Wissen auf unbekannte Daten anwenden können.

5.3 Konzept

In diesem Kapitel wird der Konzeptentwurf behandelt. Es werden Entscheidungen hinsichtlich geeigneter Methoden, Modellarchitekturen sowie Hard- und Softwarekomponenten getroffen. Des Weiteren werden Strategien in Bezug auf die Vorgehensweise bei der Projektdurchführung entwickelt.

5.3.1 Diskussion zur Auswahl geeigneter Methoden

In Abschnitt 3.1.2 werden die State-of-the-Art-Methoden im Bereich der Klassifizierung von Audiosignalen diskutiert. Entscheidend bei der Durchführung des Projektes sind insbesondere die Form des Inputs des neuronalen Netzes sowie die Auswahl einer geeigneten Modellarchitektur. Dabei haben sich bei aktuellen Arbeiten Audiosignale in Form von Mel-Spektrogrammen als Input-Features etabliert. Mel-Spektrogramme sollen auch in dieser Arbeit die Grundlage für die Klassifizierungsaufgabe bilden.

Schwieriger gestaltet sich die Festlegung bei der Auswahl einer geeigneten Modellarchitektur. In der Mehrheit der aktuellen Arbeiten werden zwar Modelle mit CNN-Architektur verwendet, dennoch gibt es eine nicht zu vernachlässigende Anzahl an Arbeiten, in denen mit RNN-Architekturen (LSTM/GRU) signifikant bessere Ergebnisse erreicht wurden. Zudem findet in der Praxis zunehmend das transferierte Lernen Anwendung. Da eine Festlegung auf eine bestimmte Architektur nicht möglich ist, wird entschieden, dass im Verlauf der Experimentierphase alle Modellarchitekturen getestet werden. Zur Verringerung des Umfangs werden vorab zwei Entscheidungen getroffen. Hierbei sollen hybride Ansätze (CRNN) gänzlich vernachlässigt und vortrainierte Modelle auf die ResNet-Architektur beschränkt werden. Der Grund für die Auswahl der ResNet-Architekturen ist, dass diese bei aktuellen Arbeiten im Audiosignalbereich gute Ergebnisse erzielen (siehe 3.2.3).

Abbildung 30 visualisiert die verwendeten Modell-Architekturen und Methoden. Neben den eben erläuterten Konzepten wird zudem der bereits in 4.3.2 eingesetzte Vision-AutoML-Service auf den BirdCLEF2022-Datensatz angewandt. Dies dient vor allem als Referenz, um zu testen, wie dieser im Vergleich zu den selbst entwickelten Lösungen abschneidet.

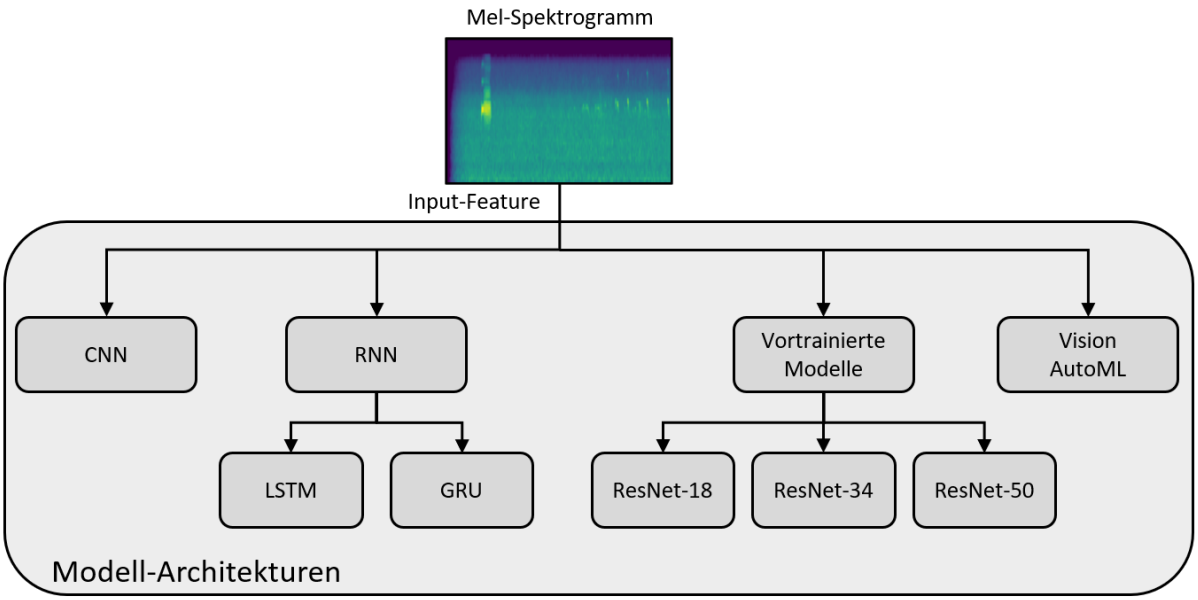


Abbildung 30: Schematische Übersicht der verwendeten Modell-Architekturen, Quelle: Eigene Darstellung.

5.3.2 Vorgehensweise

Die Vorgehensweise bei der Umsetzung des praktischen Teils lässt sich in fünf Abschnitte unterteilen (siehe Abbildung 31). Die Phasen Implementierung, Evaluierung und Integration in das Unternehmenssystem werden in einem jeweils eigenen Kapitel behandelt. Im Folgenden wird auf die beiden Phasen *Auswahl einer Entwicklungsumgebung* und *Definition einer Evaluierungsstrategie* eingegangen:

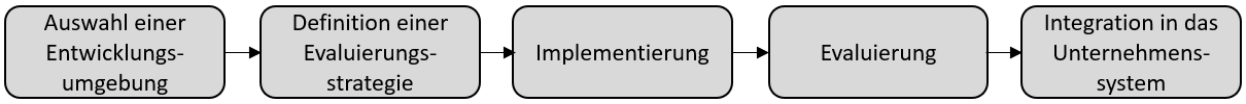


Abbildung 31: Vorgehensweise bei der Implementierung des praktischen Teils, Quelle: Eigene Darstellung.

Auswahl einer Entwicklungsumgebung

Wie in Abschnitt 4.4 beschrieben, stellt Google Vertex AI den Workbench-Service zur Verfügung. Dieser Service bietet die Möglichkeit, Notebook-Instanzen mit der interaktiven Entwicklungsumgebung JupyterLab zu erstellen. JupyterLab eignet sich als Experimentierumgebung und wird als solche in zahlreichen ML-Projekten als Ausgangspunkt verwendet. Es können von Google verwaltete oder nutzerverwaltete Notebooks erstellt werden. Für diese Arbeit wird die zweite Variante gewählt, da nutzerverwaltete Notebooks flexiblere Entwicklungsmöglichkeiten bieten. Gemeint sind damit unter anderem eine einfachere Steuerung von Zugriffsberechtigungen sowie die Möglichkeit zur Installation von diversen Softwarebibliotheken.

Als bekannte Frameworks für die softwaretechnische Umsetzung von ML-Projekten stehen TensorFlow und PyTorch zur Auswahl. Da PyTorch mit Torchaudio eine umfangreiche Softwarebibliothek zur Manipulation und Analyse von Audiosignalen bereitstellt, wird dieses als Framework gewählt.

PyTorch ist ein von Facebook entwickeltes Open-Source-ML-Framework, das auf Python basiert. Zu den Partnern von PyTorch zählen unter anderem die drei großen Cloud-Plattformen AWS, Azure und GCP, weshalb das Framework standardmäßig bei allen drei Anbietern integriert ist.[133]

Definition einer Evaluierungsstrategie

Im Verlauf dieses Projektes müssen viele Entscheidungen bzgl. der Auswahl einer geeigneten Architektur und der Abstimmung der Hyperparameter getroffen werden. Als Hyperparameter werden jene Parameter bezeichnet, die vor der Trainingsphase von den Fachexpert*innen manuell oder in einer automatisierten Umgebung abgestimmt werden müssen. Diese sind somit von den Parametern, die von den Trainingsalgorithmen angepasst werden (Gewichtungen, Bias), zu differenzieren. Um eine strukturierte und nachvollziehbare Vorgehensweise zu gewährleisten, wird die Evaluierungsphase in zwei Teile gegliedert (siehe Abbildung 32). In Phase 1 werden vorab durch Recherche und Miniexperimente Hyperparameter für die Merkmalsextraktion und das Training definiert. Der Schwerpunkt liegt hier jedoch nicht auf der Feinabstimmung ebendieser. Zunächst soll die Ermittlung einer geeigneten Modell-Architektur im Vordergrund stehen. Dazu werden die in 5.3.1 vorgestellten Architekturen mit einer verkürzten Trainingszeit getestet (reduzierter Datensatz). Im Anschluss wird evaluiert, welches Modell sich am besten für den gegebenen Datensatz eignet. In Phase 2 wird das entsprechende Modell mit dem vollständigen Datensatz trainiert und mit verbesserten Hyperparametereinstellungen optimiert. Abschließend wird das Modell auf bis dahin unbekannte Testdaten angewendet. Dadurch soll die Generalisierungsfähigkeit des erstellten Modells geprüft werden. Zudem wird die selbst entwickelte Lösung mit der AutoML-Lösung verglichen.

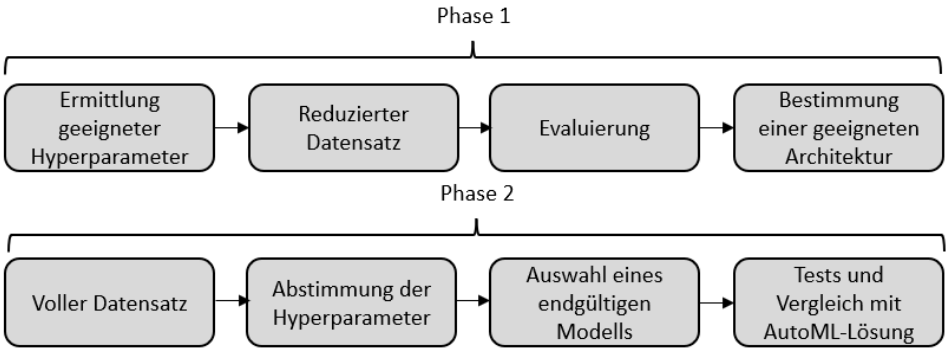


Abbildung 32: Evaluierungsstrategie zum Finden einer geeigneten Modellarchitektur und geeigneter Hyperparameter, Quelle: Eigene Darstellung.

[133] Vgl. PyTorch 1 (o.J.), Online-Quelle [17.11.2022].

5.3.3 Projektstruktur

Abbildung 33 zeigt die Projektstruktur des Audioklassifizierers. Aus Gründen der Übersichtlichkeit werden nicht alle verwendeten Cloud-Services dargestellt. Auf diese wird in Abschnitt 5.4 gesondert eingegangen. Die Abbildung ist in drei Container-Elemente gegliedert:

- Google-Cloud-Projekt – Es wird das Projekt *Audio Classification MA* angelegt. Von diesem kann auf alle notwendigen Cloud-Services zugegriffen werden. Der Zugriff von verschiedenen Google-Konten kann mit dem IAM-Service geregelt werden. Dies ermöglicht eine (theoretisch) fachteamübergreifende Entwicklung. Zum Speichern aller Daten (Audiodateien, Modelle etc.) wird der Cloud-Storage-Service verwendet. Es wird das Bucket *audio_classification_bucket* erstellt.
- Google-Vertex-AI-Service – Sämtliche Entwicklungstätigkeiten werden über eine Google DL VM (Compute-Engine-Service) ausgeführt. Diese inkludiert PyTorch als vorinstalliertes ML-Framework. Der Zugriff erfolgt über SSH oder über die grafische JupyterLab-Oberfläche.
- JupyterLab – Innerhalb der Entwicklungsumgebung wird eine Ordnerstruktur definiert. Als softwaretechnisches Strukturierungselement dienen Klassen. Jede implementierte Crisp-DM-Phase wird in eine eigene Python-Datei (Klasse) ausgelagert. Innerhalb der Ordner Analyse, Training und Evaluierung befinden sich ausführbare Notebook-Dateien (.ipynb-Format), die den jeweiligen Workflow enthalten. Die Ordner Modelle und Log-Dateien dienen als lokale Zwischenspeicher für die erstellten Modelle sowie die Trainings- und Testergebnisse. Diese werden zudem auf dem Google Cloud Storage gespeichert.

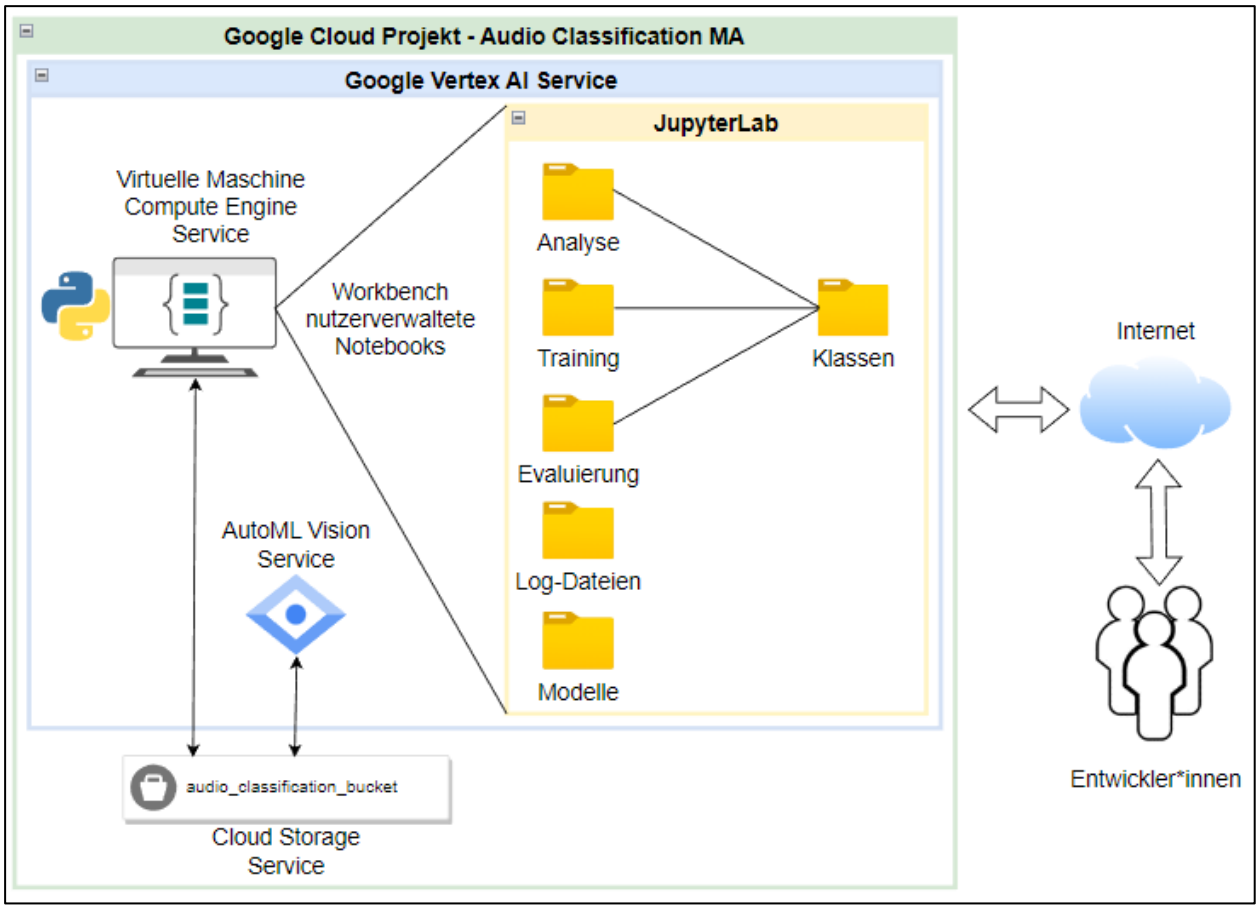


Abbildung 33: Projektstruktur des Audioklassifizierers, Quelle: Eigene Darstellung.

5.3.4 ML-Workflow

In Abbildung 34 wird der gesamte ML-Workflow dargestellt. Die Grafik dient als Übersicht und beschränkt sich auf die relevantesten Prozessschritte, die farblich voneinander getrennt sind.

- Datenanalyse (grün) – Ein von den anderen Prozessschritten abgekapselter Bereich. Die Analyse erfolgt auf einem eigenen Notebook (*Analyse.ipynb*).
- Preprocessing (violett) – Die Audiosignale werden vereinheitlicht (Länge, Kanäle, Abtastrate etc.) und von der ursprünglichen Wellenform in die Mel-Spektrogramm-Form transformiert.
- Datensätze (orange) – Neben dem Trainingsdatensatz (80 %) werden ein Validierungs- und ein Testdatensatz mit jeweils 10 % der Datenmenge erstellt. Mit dem Validierungsdatensatz wird nach jeder Trainingsepoche geprüft, ob das Modell robust gegen Überanpassung ist. Der Testdatensatz wird erst auf das fertig optimierte Modell angewendet.
- Training (grau) – Zunächst wird ein Modell erzeugt, das im Anschluss trainiert und geprüft wird. Für die Wiederverwendung werden fertig trainierte Modelle im Google Cloud Storage gespeichert.
- Test (blau) – In der Testphase wird das fertig trainierte und optimierte Modell aus dem Cloud Storage geladen und anhand der bis dahin unbekannten Testdaten geprüft.
- Veröffentlichung (türkis) – Abschließend wird das beste Modell auf Vertex AI registriert (Model-Registry-Service) und auf einem Endpunkt für Online-Vorhersagen bereitgestellt. Diese Phase wird in Kapitel 8 behandelt.

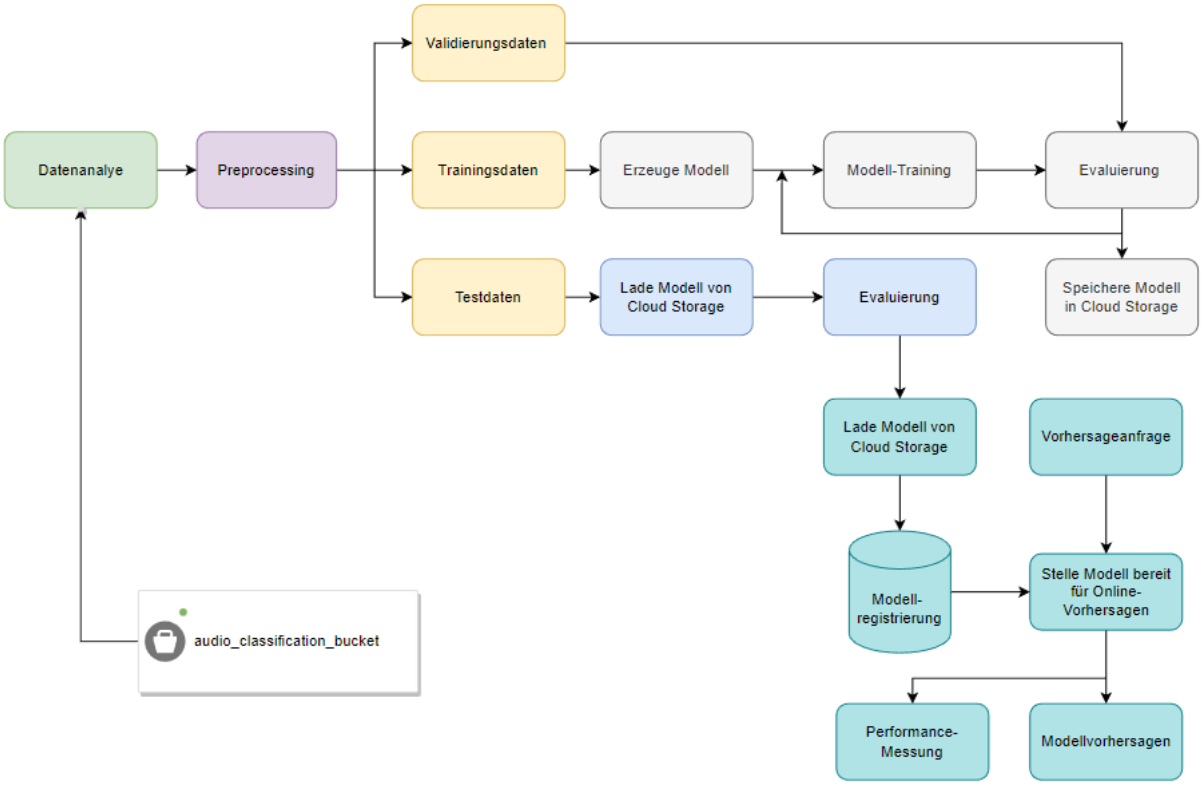


Abbildung 34: Gesamter ML-Workflow des Projektes, Quelle: Eigene Darstellung.

Sämtliche Prozessschritte, mit Ausnahme der Datenanalyse, erfolgen während der Laufzeit des Programmes. Für jeden Workflow (Training, Test, Vorhersage) wird ein eigenes Notebook erstellt, das die jeweils benötigten Klassen und Methoden enthält. Auf diese wird in Kapitel 6 genauer eingegangen.

5.4 Hard- und Software

Im Verlauf dieser Arbeit wird eine Vielzahl an unterschiedlichen Services und Softwarebibliotheken für die Umsetzung verwendet. Im Folgenden werden die zentralen Komponenten aufgelistet:

Nr.	Funktion/Bezeichnung	Service/Tool	Beschreibung/Details
1	Virtuelle Maschine 1	Compute Engine	Framework: PyTorch 1.11, Betriebssystem: Debian GNU/Linux 10, Maschinentyp: n1-standard-2 – 2 vCPUs, 7,5 GB RAM, Region: us-central1 – Iowa, Verwendung für Softwareentwicklungszwecke.
2	Virtuelle Maschine 2	Compute Engine	Framework: PyTorch 1.11, Betriebssystem: Debian GNU/Linux 10, Maschinentyp: n1-standard-16 – 16 vCPUs, 60 GB RAM, Region: us-centralt1 – Iowa, Verwendung für Trainingszwecke.
3	Speicher	Cloud Storage	Speicherklasse: Standard, Region: us-central1 – Iowa, Verwendung für Datenablage.
4	Zugriffsregelung	IAM	Verteilung von Rollen und Berechtigungen.
5	KI-Plattform	Vertex AI	Plattform zur Verwaltung aller genutzten ML-Services.
6	Nutzerverwaltetes Notebook	Vertex AI Workbench Notebooks API	Verwaltung von Notebookinstanzen.
7	Entwicklungsumgebung	Vertex AI Workbench JupyterLab	Verwendung für die Entwicklung des ML-Projektes mit Python V3.7.
8	Trainingsjobs	Vertex AI Training	Automatisiertes Training für ML-Modelle.
9	Modellverwaltung	Vertex AI Model Registry	Organisation von AutoML-Modellen und selbst erstellten Modellen.
10	Modellbereitstellung	Vertex AI Prediction	Bereitstellung der Modelle für Online-Vorhersagen.
11	Datensatzverwaltung	Vertex AI Datasets	Verwaltete Datensätze für AutoML-Service.
12	Containerverwaltung	Container Registry	Verwaltung von Container-Images, siehe Kapitel 8.
13	AutoML	Vertex Vision AutoML	Automatisiertes ML für Bildklassifizierung.
14	Kostenkontrolle	Abrechnung	Übersicht über Kosten und genutzte Services.
15	Performancekontrolle	Cloud Monitoring	Überwachung von Speicher-, Computing- und Softwareressourcen.

Tabelle 4: Übersicht über Hard- und Softwarekomponenten, Quelle: Eigene Darstellung.

In der Tabelle sind zwei virtuelle Maschinen gelistet. Aus Kostengründen werden die Computing-Ressourcen nur während der Trainingszeiten hochskaliert. Des Weiteren werden aus Kostengründen nur CPUs und keine GPUs für das Training verwendet, weshalb der Trainingsdatensatz, wie in Abschnitt 5.1 beschrieben, reduziert wird. Bei größeren Datensätzen ist die Verwendung einer oder mehrerer GPUs zu empfehlen, da die Trainingszeiten deutlich verkürzt werden.

Im Folgenden werden die relevantesten Python-Bibliotheken für die Projektumsetzung gelistet:

Nr.	Bezeichnung	Verwendet in Crisp-DM-Phase:	Verwendet für:
1	Pandas	Datenanalyse, Preprocessing	Zum Lesen und Schreiben von CSV-Dateien.
2	Matplotlib	Datenanalyse	Zum Erstellen von Diagrammen.
3	Seaborn	Datenanalyse	Zum Erstellen von Diagrammen.
4	Numpy	alle Phasen	Für die Handhabung von Arrays in Python.
5	Scikit-learn	Training, Evaluierung	Zur Ermittlung der Metriken für die Bewertung der Testergebnisse.
6	Torch	alle Phasen	Softwarepaket von PyTorch für die Handhabung von PyTorch-Tensoren und die Erstellung von neuronalen Netzen mit Python.
7	Torchaudio	Preprocessing	Zur Manipulation von Audiosignalen mit Python.
8	Torchvision	Modellierung	Bibliothek, die vortrainierte Modelle für die Bildklassifizierung bereitstellt.
9	Google Cloud Storage	alle Phasen	Zum Lesen und Schreiben des Google-Speichers.
10	Flask	Bereitstellung	Zum Erstellen einer Webanwendung mit Python.
11	Docker	Bereitstellung	Zum Erstellen von Docker-Images mit Python.

Tabelle 5: Übersicht Python-Bibliotheken, Quelle: Eigene Darstellung.

Als Grundlage zur Entwicklung der Audiosignalklassifizierungsaufgabe mit Python dienen ein PyTorch-Tutorial[134] zur Audiosignalmanipulation sowie ein DL-Tutorial im Klassifizierungsbereich.[135] Diese werden an die Aufgabenstellung dieser Arbeit angepasst.

[134] Vgl. PyTorch 2 (2021), Online-Quelle [17.11.2022].

[135] Vgl. Doshi (2021), Online-Quelle [17.11.2022].

6 IMPLEMENTIERUNG

Auf Basis der in Kapitel 5 definierten Anforderungen und Vorgehensweisen wird in diesem Kapitel die Umsetzung im Detail behandelt. Die einzelnen Unterkapitel gliedern sich nach den jeweiligen Phasen des ML-Workflows.

6.1 Datenanalyse

Der BirdCLEF2022-Datensatz wird nach den in Abschnitt 5.1 definierten Kriterien auf neun Klassen reduziert. Diese neun Klassen werden im weiteren Verlauf der Arbeit als vollständiger Datensatz bezeichnet. Zudem erfolgt für die erste Experimentierphase eine weitere Vereinfachung auf vier Klassen. Tabelle 6 listet den verwendeten Datensatz. Die hellblau markierten Elemente sind im vollständigen Datensatz sowie im reduzierten Datensatz enthalten. Der reduzierte Datensatz entspricht mit 573 Audiodateien etwa einem Viertel der Gesamtmenge und kann somit als aussagekräftig interpretiert werden. Die Metadaten sind in den CSV-Dateien *train_metadata_f.csv* (vollständiger Datensatz) und *train_metadata_r.csv* (reduzierter Datensatz) im Cloud Storage hinterlegt. In der Tabelle ist die Anzahl der Dateien für die einzelnen Phasen Training, Validierung und Test ersichtlich. Ungerade Ergebnisse sind gerundet, weshalb eine geringfügige Differenz zwischen Validierungs- und Testdatensatz vorherrscht. Des Weiteren wird im Folgenden auf die ersten beiden Spalten eingegangen:

- ID – Bildet die Basis für Vorhersagen. Beispielsweise entspricht ein Vorhersageergebnis mit dem Wert 8 einem Drosseluferläufer.
- Label – Zugehörige Audiodateien sind im Ordner mit dem jeweiligen Label gespeichert. Beispielsweise entspricht eine Datei mit dem Speicherort …/*sposan/XC31322.ogg* einem Drosseluferläufer. Die Labels werden im weiteren Verlauf als Klassenbezeichnung verwendet.

ID	Label	Deutscher Name	Gesamt	Training	Validierung	Test
0	arcter	Küstenseeschwalbe	196	157	20	19
1	brnowl	Schleiereule	500	400	50	50
2	cangoo	Kanadagans	318	255	32	31
3	caster1	Raubseeschwalbe	176	141	18	17
4	comwax	Wellenastrild	194	156	19	19
5	houfin	Hausgimpel	322	258	32	32
6	nutman	Muskatbronzemännchen	78	63	8	7
7	semplo	Amerika-Sandregenpfeifer	77	62	8	7
8	sposan	Drosseluferläufer	123	99	12	12
		Summe vollständiger Datensatz	**1984**	**1591**	**199**	**194**
		Summe reduzierter Datensatz	**573**	**460**	**58**	**55**

Tabelle 6: Reduzierte BirdCLEF2022-Datensätze, Quelle: Eigene Darstellung.

Die Datenanalyse mit Python erfolgt im Notebook *Analyse.ipynb*. In diesem werden die in Tabelle 7 gelisteten Statistiken ermittelt. Wie ersichtlich ist, wurden alle Audiodateien bereits vom Cornell Lab mit einer Abtastrate von 32 000 Hz neu abgetastet. Die Längen der Audiodateien weichen stark voneinander ab. Die hohe durchschnittliche Dauer von 44,81 s ergibt sich hauptsächlich aufgrund weniger Ausreißer. Mit 50,4 %, also exakt 1000 Audiosignalen, entsprechen die meisten Dateien einer Länge im Bereich von 5 bis 15 s. Dennoch ist eine große Anzahl der Dateien (47,94 %) länger als 15 s. Die Mehrheit der Signale (88,66 %) liegt in Stereo vor.

Bezeichnung	**Wert**	**Anzahl Dateien**
Abtastrate	32 000 Hz	1984 (alle)
Kürzeste Audiodatei	0,81 s	1
Längste Audiodatei	2012,21 s	1
Durchschnittliche Dauer	44,81 s	Mittelwert
Länge <= 5 s	1,66 %	33
Länge <= 10 s	22,63 %	449
Länge <= 15 s	27,77 %	551
Länge > 15 s	47,94 %	951
Mono	11,34 %	225
Stereo	88,66 %	1759

Tabelle 7: Statistik vollständiger Datensatz, Quelle: Eigene Darstellung.

Entscheidungen aufgrund ermittelter Statistiken

Um optimale Klassifizierungsergebnisse zu erhalten, ist es notwendig, das Input-Signal des neuronalen Netzes hinsichtlich des Eingabeformats zu vereinheitlichen. Laut Statistik sind in diesem Fall die Dauer des Signals und die Anzahl der Kanäle betroffen. Da die eindeutige Mehrzahl der Signale in Stereo vorliegt, werden auch die Monodateien in Stereo umgewandelt. Dies erfolgt durch das Kopieren des ersten Kanals. Aufgrund der starken Abweichung hinsichtlich der Dauer der einzelnen Dateien ist es schwierig, eine optimale Signallänge zu definieren. Deshalb werden 50 manuelle Hörproben ermittelt. Bei allen Proben sind innerhalb der ersten drei Sekunden Vogelgeräusche erkennbar. Ferner liegen die meisten Dateien im Bereich zwischen 5 und 15 s, weshalb eine Länge von 10 s für die Aufgabenstellung gewählt wird.

Unterschiede zwischen Dateien gleicher Klasse

Zur Veranschaulichung der unterschiedlichen Umgebungsbedingungen werden in Abbildung 35 zwei Proben gleicher Klasse mit ähnlicher Länge dargestellt. Die Werte der normalisierten Amplitude liegen im Intervall [-1, 1]. Die x-Achse spiegelt die Anzahl der abgetasteten Werte wider. Bei einer Abtastrate von 32 000 Hz entsprechen beide Signale einer Länge von ca. 9 s. Wie ersichtlich ist, unterscheiden sich beide Signale stark hinsichtlich des Amplitudenausschlags sowie der Störgeräusche. Für die Audioklassifizierungsaufgabe wirkt sich dies negativ aus, da das Erlernen von Mustern wesentlich erschwert wird.

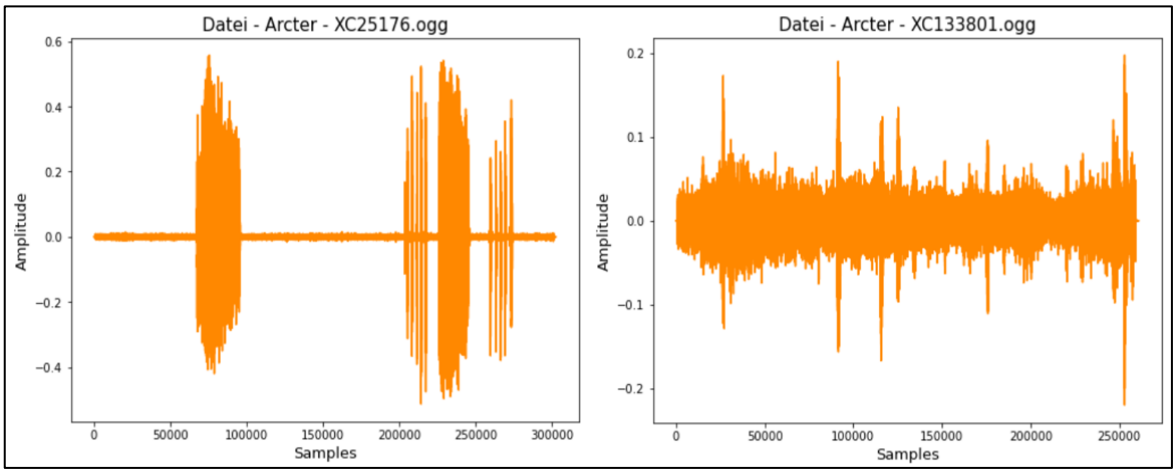


Abbildung 35: Unterschiede zwischen Audiodateien gleicher Klasse, Quelle: Eigene Darstellung.

Analyse mit PyTorch und Python

Audiosignalanalysen und Manipulationen können mit den von PyTorch zur Verfügung gestellten Python-Bibliotheken *Torch* und *Torchaudio* durchgeführt werden. Diese werden auch im weiteren Verlauf für die Bearbeitung der Aufgabenstellung verwendet. Neben der Handhabung von Audiosignalen (Laden, Öffnen, Abspielen) werden unter anderem Methoden zur Extraktion von Mel-Spektrogrammen bereitgestellt. Ein zentrales Element beim Umgang mit dem ML-Framework sind PyTorch-Tensoren. Tensoren sind multidimensionale Matritzen eines Typs (z. B. *float32*).[136] Abbildung 36 stellt die Ausgabe eines solchen Tensors einer Beispieldatei dar. In diesem Fall liegt ein Tensor mit dem Format [x, y] vor. Dabei beschreibt x die Anzahl der Kanäle und y die Länge des Signals.

```
tensor([[0.0000e+00, 0.0000e+00, 0.0000e+00,  ..., 0.0000e+00, 0.0000e+00,
         0.0000e+00],
        [0.0000e+00, 0.0000e+00, 3.0518e-05,  ..., 3.0518e-05, 0.0000e+00,
         0.0000e+00]])

torch.Size([2, 260807])
```

2-Kanäle

Anzahl Kanäle

Länge

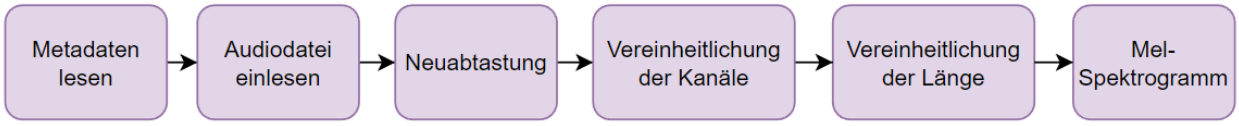
Abbildung 36: Ausgabe des Tensors der Audiodatei Arcter – *XC25176.ogg*, Quelle: Eigene Darstellung.

[136] Vgl. PyTorch 3 (2022), Online-Quelle [17.11.2022].

6.2 Preprocessing

Vor jedem Trainings- oder Testdurchlauf werden die Daten vorverarbeitet. Wie bereits erwähnt, findet dieser Vorgang während der Laufzeit des Programmes statt. Denkbar wäre es, diesen Prozess nur einmalig zu durchlaufen und die extrahierten Mel-Spektrogramme im Cloud Storage abzuspeichern. Dies würde zu verbesserten Trainingszeiten führen, jedoch müsste dies nach jeder Hyperparameteränderung erneut durchgeführt werden, was einen ausgeprägten Zeitaufwand für das Speichern nach sich ziehen würde.

Abbildung 37 visualisiert den gesamten Workflow der Preprocessing-Phase. Alle Methoden zur Audiosignalmanipulation sind in der Klasse *AudioProcessing* implementiert, die in der Datei *Preprocessing.py* ausgelagert ist. Der Aufruf der Preprocessing-Methoden erfolgt innerhalb der Klasse *AudioDataSet (DataSet.py)*. Zunächst werden jeweils der Filename (z. B. *arcter/XC25068.ogg*) und die Klassen-ID (z. B. 0 für Arcter) von jedem Zeileneintrag in der CSV-Datei erfasst. Anhand des Filenamens kann der Speicherort der jeweiligen Audiodatei im Cloud Storage ermittelt werden. Im Anschluss werden die Audiodateien mit *openAudio()* geladen und in einen PyTorch-Tensor umgewandelt. Obwohl die Trainingsdaten bereits mit einer Abtastrate von 32 000 Hz gesampelt sind, erfolgt mit *resampleAudio()* eine Neuabtastung. Insbesondere bei Vorhersageanfragen ist es möglich, dass unbekannte Daten mit anderen Abtastraten vorkommen. Daraufhin wird mittels *rechannelAudio()* die Anzahl der Audiokanäle überprüft. Ist nur ein Kanal vorhanden (Mono), wird dieser kopiert (Stereo). Der letzte Schritt bei der Bearbeitung der Wellenform beinhaltet das Vereinheitlichen der Dateilänge. Dazu werden die Tensoren entweder nach einer Länge von 320 000 (10 s) abgeschnitten oder kürzere bis dahin mit Nullen aufgefüllt (*truncateOrPad()).* Die Nullen können dabei als Stille betrachtet werden. Abschließend wird die Wellenform mittels STFT, Mel-Skala und Mel-Filterbank in ein Mel-Spektrogramm transformiert (*melSpec()*). Die Methode *__getitem__()* gibt ein Objekt mit zwei Tensoren mit dem Format [x, y] zurück, wobei x mit [Kanäle, Mel-Bänder, Frames] und y mit [Klassen-ID] beschrieben werden kann.




```
#get i'th element of csv sheet
def __getitem__(self, idx):

    #get path to google cloud storage location
    audioFile = self.data_path + self.df.loc[idx, 'filename']
    #get class id (label)
    class_id = self.df.loc[idx, 'class_id']

    #wavelet form
    tens,sr = AudioProcessing.openAudio(audioFile)
    tens,sr = AudioProcessing.resampleAudio(tens,sr,self.sr)
    tens,sr = AudioProcessing.rechannelAudio(tens,sr,self.channel)
    tens,sr = AudioProcessing.truncateOrPadAudio(tens,sr,self.duration)

    #mel spectrogram form
    tens_melspec = AudioProcessing.melSpec(tens,sr,
    self.n_fft, self.hop_length, self.n_mels, self.win)

    return tens_melspec, class_id
```

Abbildung 37: Workflow Preprocessing, Quelle: Eigene Darstellung.

Abbildung 38 zeigt ein manipuliertes Audiosignal als Wellenform (links) sowie als Mel-Spektrogramm (rechts) in einkanaliger Form. Um die definierte Länge von 10 s zu erreichen, wird der Tensor mit Nullen aufgefüllt (jeweils linker roter Rahmen). Des Weiteren wird der Zusammenhang zwischen den beiden Signalen in den unterschiedlichen Domänen verdeutlicht (jeweils rechter roter Rahmen). Die Form des Mel-Spektrogramms hängt wesentlich von den konfigurierten Hyperparametern ab. Bei der abgebildeten Implementierung ergibt sich die Anzahl der Mel-Bänder und somit die Höhe aufgrund der bei der Extraktion angewendeten Filteranzahl der Mel-Filterbank (*n_mels = 64*). Zudem resultieren bei einer eingestellten Schrittweite (*hop_length*) von 512 und einer Abtastrate von 32 000 Hz insgesamt 62,5 Frames/s bzw. 625 Frames bei einer Signallänge von 10 s. Dies entspricht einer Schrittlänge von 16 Millisekunden. Neben den zwei bereits erläuterten Hyperparametern wird bei der Mel-Spektrogramm-Extraktion mit PyTorch auf folgende sechs Parameter eingegangen (siehe Abschnitt 3.1.1):[137]

- Abtastrate – Abtastrate des Audiosignals. Standardeinstellung PyTorch: 16 000 Hz.
- Hochpassfilter (*f_min*) – Minimalfrequenz. Standardeinstellung PyTorch: 0 Hz.
- Tiefpassfilter (*f_max*) – Maximalfrequenz. Standardeinstellung PyTorch: Abtastrate / 2.
- FFT-Größe (*n_fft*) – Hat Einfluss auf die Frequenzauflösung je Frame. Es werden *1 + n_fft / 2* Frequenz-Bins erzeugt. Standardeinstellung PyTorch: 400.
- STFT-Fenstergröße (*win_length*) – Größe je Frame. Standardeinstellung PyTorch: *n_fft*.
- Fensterfunktion (*window_fn*) – Fensterfunktion, die im Zuge der STFT angewandt wird. Standardeinstellung PyTorch: Hanning-Fenster.

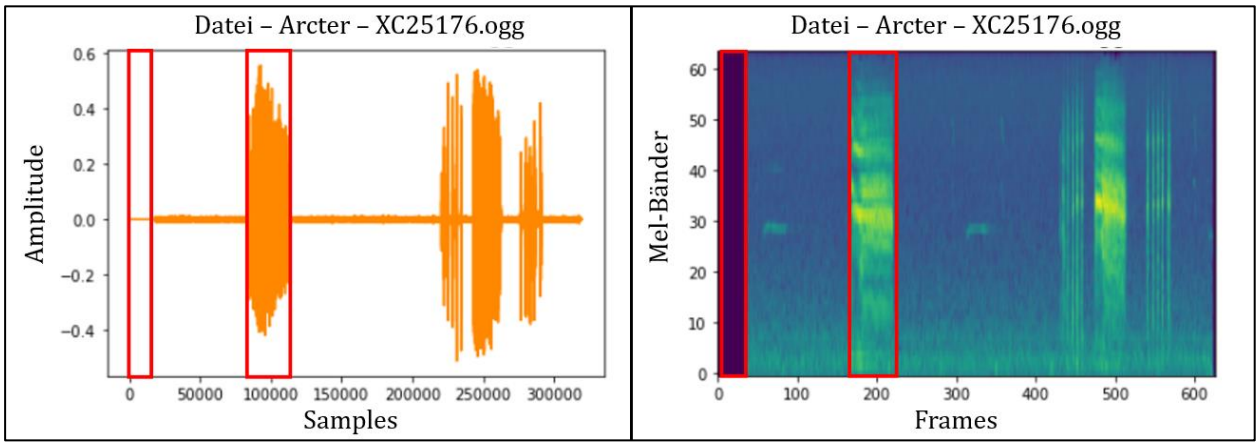


Abbildung 38: Darstellung einer Audiodatei nach der Preprocessing-Phase – Wellenform (links), Mel-Spektrogramm (rechts), Quelle: Eigene Darstellung.

[137] Vgl. PyTorch 4 (2022), Online-Quelle [17.11.2022].

6.3 Datensätze

Wie in 6.1 beschrieben, werden aus den zur Verfügung stehenden Daten insgesamt drei Datensätze für die Trainings- und Testphase erstellt. Die Vorgehensweise bei der Erstellung ebendieser wird in diesem Abschnitt behandelt. Zunächst soll jedoch auf die für das Training relevanten Hyperparameter Batch-Größe und Epoche eingegangen werden:[138]

- Batch-Größe – Im Zuge des Trainings versuchen Trainingsalgorithmen, den Fehler zu minimieren (siehe Abschnitt 3.2.2). Üblicherweise werden die Gewichtungen des neuronalen Netzes nach jedem Durchlauf (Mel-Spektrogramm) aktualisiert. Dies kann zu langen Trainingszeiten führen, da die Betrachtung einer einzelnen Probe meist nicht für eine ausreichende Annäherung an den minimalen Fehler reicht. Um diesem Problem entgegenzuwirken, werden die einzelnen Daten in Stapel (Batches) gegliedert. Das Update der Gewichtungen erfolgt dabei erst, nachdem alle Proben eines Batches durchlaufen sind. Gängige Batch-Größen sind beispielsweise 16, 32, 64 oder 128.
- Epochen – Ein Trainingsprozess wird in einzelne Epochen geteilt. Eine Epoche ist nach dem Durchlauf aller Proben, also aller Batches, abgeschlossen. Epochen ermöglichen einen frühzeitigen Trainingsabbruch. Dies kann notwendig sein, wenn bereits während des Trainingsprozesses eine Überanpassung an die Trainingsdaten auftritt. Ein Zeichen dafür ist beispielsweise, dass sich die Genauigkeit der Vorhersagen in Bezug auf die Trainingsmenge weiterhin erhöht und die Genauigkeit der Vorhersagen in Bezug auf die Validierungsmenge stagniert oder abnimmt.

Beispiel

Es liegt ein Datensatz mit 2000 Elementen vor. Die Batch-Größe beträgt 10 und die Anzahl der Epochen 100. Somit werden 200 Stapel mit jeweils zehn Proben gebildet. Nach zehn Proben (einem Batch) erfolgt eine Gewichtsanpassung. Je Epoche gibt es somit 200 Parameteranpassungen, was bei 100 Epochen insgesamt 20 000 Updates entspricht.

Abbildung 39 (unten) zeigt die schematische Darstellung der Vorgehensweise bei der Erstellung der einzelnen Datensätze für den Trainingsprozess. Zunächst werden 10 % der Daten in eigenen Ordnerstrukturen an den Cloud Storage ausgelagert. Diese dienen für die abschließenden Tests. Die Methode *buildDataSet()* ist ein Member der Klasse *AudioDataSet*. Es werden die zuvor extrahierten Mel-Spektrogramme übergeben. Die übergebene Menge wird anschließend zufällig in einen Trainings- und Testdatensatz geteilt (Codezeile 4–7). Dies erfolgt vor jedem Trainingsprozess erneut, wodurch eine Variation der Input-Features gewährleistet werden kann. Je Datensatz wird mit *torch.utils.data.DataLoader()* ein *DataLoader*-Objekt erzeugt. Die Daten werden gemischt und je nach konfigurierter Batch-Größe gestapelt. Im Verlauf des Trainings werden die einzelnen Stapel während der Laufzeit abgerufen und in die CPU geladen. Dies ist ein iterativer Prozess.

[138] Vgl. Buduma (2017), S. 30 ff.

```python
def buildDataSet(Data, train_batch_size, valid_batch_size):

    #random split between train and validation data
    num_items = len(Data)
    num_train = round(num_items * 0.9)
    num_val = num_items - num_train
    train_ds, val_ds = random_split(Data, [num_train, num_val])

    #create data loader objects
    train_dl = torch.utils.data.DataLoader(train_ds, batch_size=train_batch_size, shuffle=True)
    val_dl = torch.utils.data.DataLoader(val_ds, batch_size=valid_batch_size, shuffle=False)

    return train_dl, val_dl
```

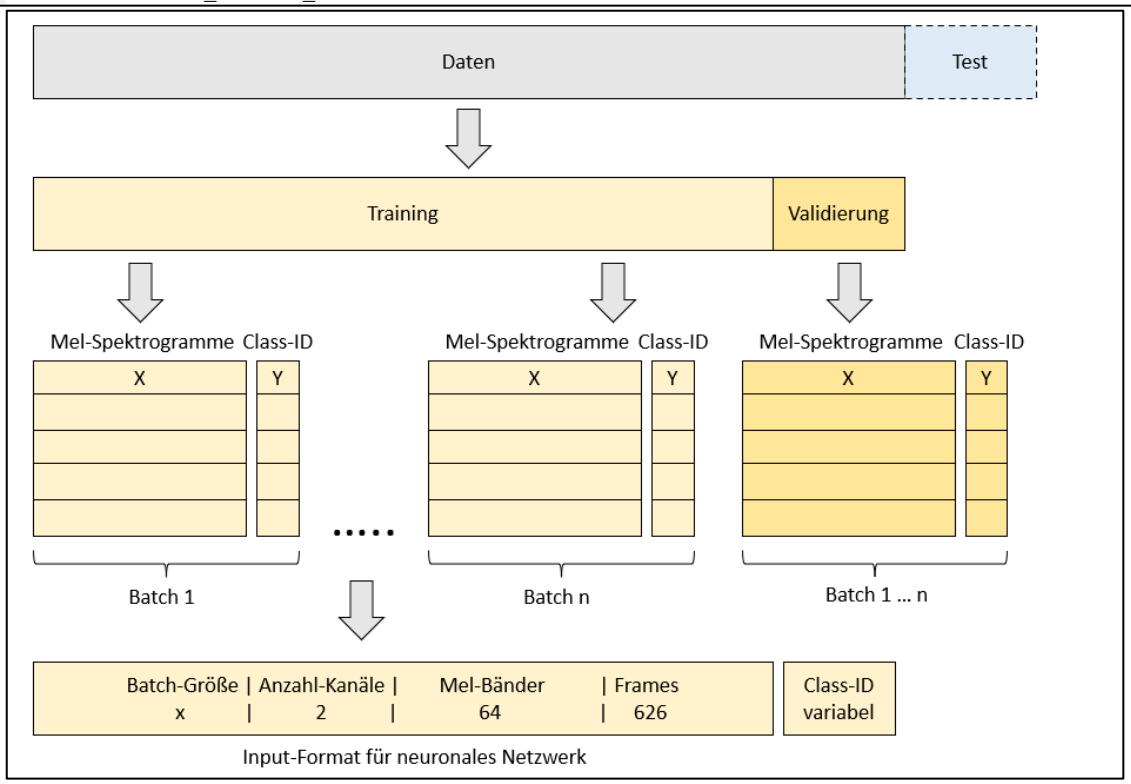


Abbildung 39: Schematische Darstellung der Datensätze, Quelle: Eigene Darstellung.

6.4 Modellierung

In diesem Abschnitt werden die in Abschnitt 5.3.1 diskutierten Modellarchitekturen behandelt, die im Zuge dieser Arbeit implementiert werden. Dabei wird auf die Spezifikation der neuronalen Netzwerke und die Vorgehensweise bei der Definition der einzelnen Schichten eingegangen. Ferner wird die softwaretechnische Umsetzung mit Python betrachtet.

6.4.1 CNN

Zunächst wird in Abbildung 40 die schematische Darstellung der Audioklassifizierung mit CNN gezeigt. Die Mel-Spektrogramme werden während der Laufzeit aus den Stapeln entnommen (*DataLoader*) und durch die Schichten des CNN geschickt. Mittels Filters erfolgt die Extraktion der Feature-Maps. Diese mehrdimensionalen Strukturen werden durch einen Flatten-Layer in eine eindimensionale Struktur überführt. Abschließend durchlaufen die Daten eine versteckte, vollverbundene Schicht. Die Anzahl der Neuronen in der Ausgangsschicht ist gleich der Anzahl der Klassen je Datensatz (reduziert oder vollständig).

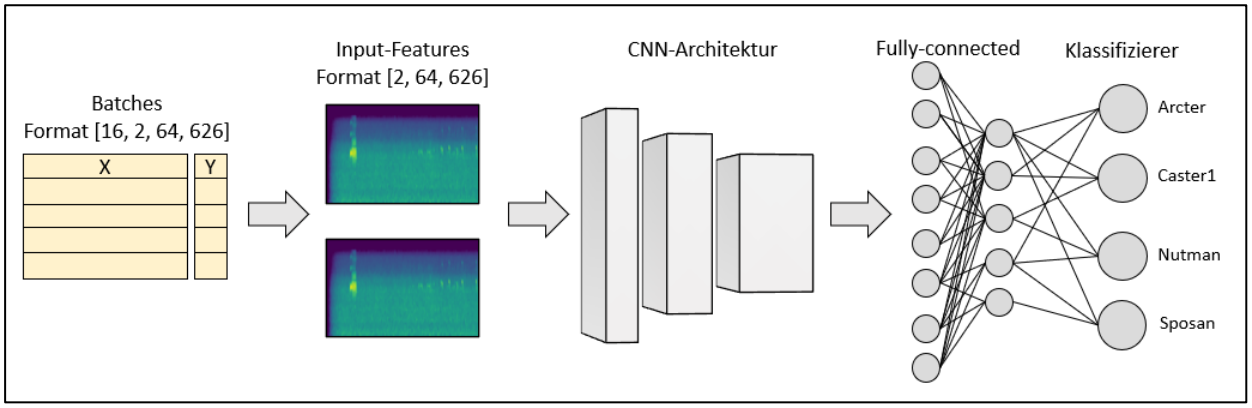


Abbildung 40: Schematische Darstellung CNN-Architektur, Quelle: Eigene Darstellung.

Bei der Spezifikation von CNN gibt es eine Vielzahl an Hyperparametern, die bestimmt werden müssen (siehe Abschnitt 3.2.1). Eine definierte Vorgehensweise existiert dabei nicht, jedoch sind in der Literatur Empfehlungen zu finden, die auch für diese Arbeit als Richtlinie dienen sollen. Dementsprechend soll nach jedem Convolutional Layer (*C*) eine ReLU-Aktivierungsschicht (*R*) folgen. Dieses Muster kann beliebig oft wiederholt werden. Es wird empfohlen, nach jedem zweiten oder dritten *CR*-Block eine Pooling-Schicht (*P*) anzufügen. Abschließend erfolgen die Überführung zur *FCL* und die Klassifizierung (*K*). Der grundsätzliche Aufbau kann sich somit beispielsweise wie folgt gestalten: *CR CRP CR CRP FCL K* oder *CR CR CRP CR CR CRP FCL K*.[139] Zudem werden für weitere relevante Hyperparameter die folgenden Werte empfohlen:[140]

- Räumliche Ausdehnung Filter – Für die erste Schicht 5 x 5, ansonsten 5 x 5 oder 3 x 3,
- Anzahl der Filter – Aufgrund schnellerer Rechenergebnisse meist eine Zweierpotenz der vorgelagerten Schicht. Die Anzahl der Filter der Input-Schicht entspricht der Anzahl der Kanäle des Eingangssignals,
- Schrittweite Filter (Stride) – 1 x 1 oder 2 x 2,
- Padding – Abhängig von der Filtergröße, z. B. 1 x 1 oder 2 x 2,
- Pooling – 2 x 2.

Als Grundlage für den Entwurf der CNN-Architektur dient ein Online-Tutorial.[141] Um eine geeignete Anzahl von Schichten für die Modell-Architektur zu definieren, wurden vorab einige Miniexperimente durchgeführt. Diese wurden frühzeitig abgebrochen und nicht aufgezeichnet. Jedoch wurde ersichtlich, dass tiefere Modelle aufgrund der höheren Komplexität und der niedrigen Anzahl an Daten mehr zur Überanpassung neigen, weswegen insgesamt vier Modelle mit bis zu maximal 25 Schichten getestet werden.

[139] Vgl. Charu (2018), S. 328.

[140] Vgl. Standford University (o.J.), Online-Quelle [17.11.2022], Vgl. Charu (2018), S. 324 ff.

[141] Vgl. Doshi (2021), Online-Quelle [17.11.2022].

Zunächst wird auf die softwaretechnische Implementierung mit Python eingegangen. Die jeweiligen Architekturen sind in Klassen in eigenen Python-Dateien ausgelagert (*CNN1_N.py … CNN4_N.py*). Abbildung 41 stellt einen reduzierten Ausschnitt einer solchen Architektur dar. Angewendet wird das Muster *CR CRP*, wobei sich der Aufbau in Blöcke gliedert. Es kann zwischen drei Arten von Blöcken differenziert werden:

- Eingangsblock (*EB*) – Der Hyperparameter für Filter (*kernel_size*) und das Verhältnis von Eingang (Anzahl Kanäle) und Ausgang (Feature-Maps) weichen von den anderen Blöcken ab.
- Block 1 (*B1*) – Eine Kombination von *CR*. Das Verhältnis von Eingang zu Ausgang verdoppelt sich.
- Block 2 (*B2*) – Eine Kombination von *CRP*. Das Verhältnis von Eingang zu Ausgang verdoppelt sich.

Die Werte für alle Hyperparameter richten sich nach den Empfehlungen und werden nicht weiter erläutert. Je Block werden zusätzlich Bias und Gewichtungen initialisiert. Dies verkürzt Trainingszeiten und wirkt schwindenden oder explodierenden Gradienten (siehe 3.2.2) entgegen. Die Feature-Maps des letzten Blocks werden in eine eindimensionale Struktur überführt und abschließend erfolgt die Vorhersage mittels linearen Klassifizierers (Codezeile 25).

```
1  #input-block
2  self.conv1 = nn.Conv2d(2, 8, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
3  self.relu1 = nn.ReLU()
4  init.kaiming_normal_(self.conv1.weight, a=0.1)
5  self.conv1.bias.data.zero_()
6  conv_layers += [self.conv1, self.relu1]
7
8  #block 2
9  self.conv2 = nn.Conv2d(8, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
10 self.relu2 = nn.ReLU()
11 self.pool2 = nn.MaxPool2d(kernel_size=2)
12 init.kaiming_normal_(self.conv2.weight, a=0.1)
13 self.conv2.bias.data.zero_()
14 conv_layers += [self.conv2, self.relu2, self.pool2]
15
16 #block 1
17 self.conv3 = nn.Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
18 self.relu3 = nn.ReLU()
19 init.kaiming_normal_(self.conv3.weight, a=0.1)
20 self.conv3.bias.data.zero_()
21 conv_layers += [self.conv3, self.relu3]
22
23 #classifier
24 self.ap = nn.AdaptiveAvgPool2d(output_size=1)
25 self.lin = nn.Linear(in_features=32, out_features=num_classes)
```

Abbildung 41: Reduzierte Ansicht einer CNN-Architektur mit PyTorch, Quelle: Eigene Darstellung.

Es ergeben sich die in Tabelle 8 gelisteten Architekturen:

Name	Blockabfolge	Blöcke	Schichten	Output Feature-Maps
CNN1	EB B2 B1 B2	4	10	64
CNN2	EB B2 B1 B2 B1 B2	6	15	256
CNN3	EB B2 B1 B2 B1 B2 B1 B2	8	20	1024
CNN4	EB B2 B1 B2 B1 B2 B1 B2 B1 B2	10	25	4096

Tabelle 8: Definierte CNN-Architekturen, Quelle: Eigene Darstellung.

6.4.2 RNN – LSTM und GRU

Der Aufbau der LSTM- und GRU-Netzwerke unterscheidet sich von dem der CNN-Netzwerke. Abbildung 42 zeigt eine schematische Darstellung mit Mel-Spektrogrammen als Input. Die Tiefe der Netzwerke hängt hier von zwei unterschiedlichen Faktoren ab. Einerseits bestimmt die Sequenzlänge (Frames) die Anzahl der LSTM- oder GRU-Einheiten (z. B. 626) und andererseits können mehrere Schichten mit derselben Anzahl an Einheiten übereinandergestapelt werden. Die Dauer der Eingangssignale ist zwar mit 10 s fix definiert, jedoch hat die Schrittweite bei der Extraktion der Mel-Spektrogramme unmittelbare Auswirkungen auf die Dimensionen (Anzahl Frames – siehe Abschnitt 6.2). Die y-Achse der Spektrogramme korreliert mit der Anzahl der Mel-Bänder (z. B. 64), die die Input-Features (je Zeitschritt) darstellen. Die letzten Schichten des Klassifizierers sind gleich aufgebaut wie bei den CNNs. Relevante Hyperparameter, die bei der Umsetzung beachtet werden müssen, sind die folgenden drei:

- Eingangsgröße (*input_size*) – Tensor im Format [Batch-Größe, Sequenzlänge, Input-Features],
- Output-Features (*hidden_size*) – Anzahl der Merkmale je LSTM- oder GRU-Einheit,
- Anzahl der Schichten (*num_layers*) – Anzahl der übereinanderliegenden LSTM- oder GRU-Einheiten.

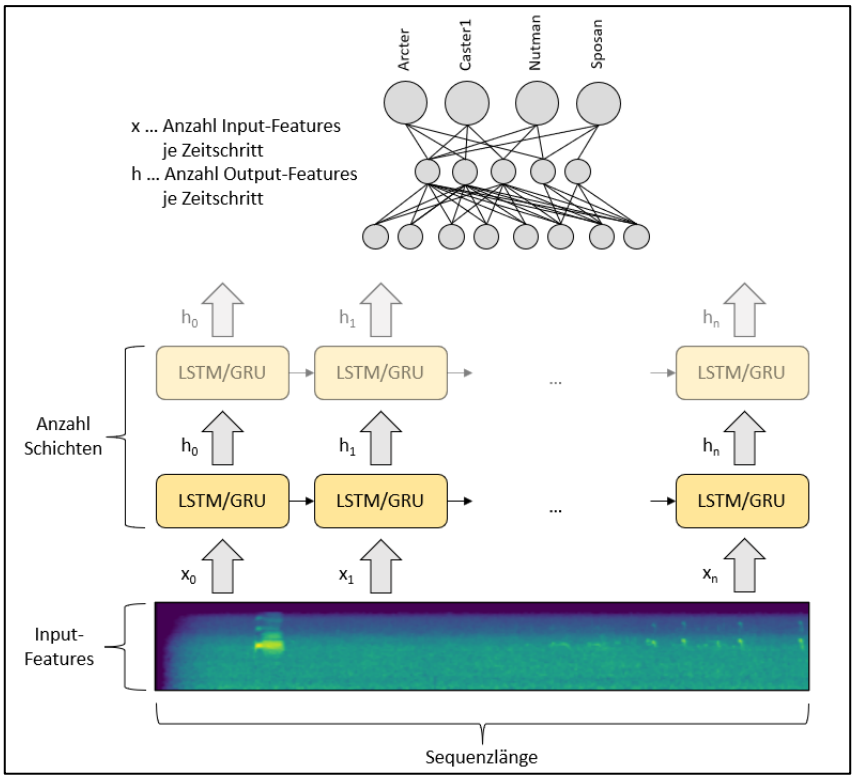


Abbildung 42: Schematische Darstellung LSTM- oder GRU-Architektur, Quelle: Eigene Darstellung.

Mit Python und PyTorch sind GRU- oder LSTM-Architekturen einfach zu implementieren (siehe Abbildung 43). Es werden jeweils eine Klasse LSTM und GRU in gleichnamige Python-Dateien ausgelagert. Der Aufbau beider Klassen ist mit Ausnahme der Schlüsselwörter LSTM und GRU ident. Das Objekt wird beim Erzeugen mit den benötigten Hyperparametern initialisiert. Anders als bei den CNNs werden hier aufgrund der flexibleren und vereinfachten Erweiterungsmöglichkeiten (Hyperparameterübergabe bei Instanziierung) keine fixen Architekturen definiert.

Des Weiteren konnte im Zuge der Recherche keine Empfehlung bzgl. Vorgehensweise bei der Definition der Hyperparameter gefunden werden, weshalb eine Vielzahl unterschiedlicher Modelle getestet wird.

```
def __init__(self, input_size, hidden_size, num_layers, num_classes):
    super(GRU, self).__init__()
    self.hidden_size = hidden_size
    self.num_layers = num_layers
    self.gru = nn.GRU(input_size, self.hidden_size, self.num_layers)
    self.fc1 = nn.Linear(hidden_size, hidden_size)
    self.relu = nn.ReLU()
    self.fc = nn.Linear(hidden_size, num_classes)
```

Abbildung 43: Reduzierte Ansicht der Klasse GRU mit PyTorch, Quelle: Eigene Darstellung.

6.4.3 Vortrainierte Modelle

In diesem Abschnitt erfolgt eine Auseinandersetzung mit der Anwendung des transferierten Lernens. Wie in 5.3.1 beschrieben, werden dazu die ResNet-18-, ResNet-34- und ResNet-50-Architekturen implementiert. Die angehängte Nummer bezieht sich jeweils auf die Anzahl der Schichten der Netzwerke. ResNet-Modelle sind CNN-Architekturen, die aus sich wiederholenden Blöcken (Residual-Blöcke) bestehen. Die Besonderheit von ResNets sind ‚skip connections', durch die einzelne Schichten übersprungen werden können. Dies ermöglicht das Training tieferer Netzwerke.[142] Im Mittelpunkt dieses Abschnitts soll jedoch die Anwendung dieser Modelle stehen, weshalb die theoretischen Grundlagen nicht weiter erläutert werden. ResNet-Modelle wurden ursprünglich für die Klassifikation im Bildbereich entwickelt. PyTorch empfiehlt für die Verwendung dieser Modelle ein Eingangsformat von [3, min. 224, min. 224] mit der Beschreibung [Anzahl Kanäle, Höhe, Breite].[143] Dies wird bei der Verwendung der vortrainierten Modelle im Zuge der Implementierung berücksichtigt und entsprechend angepasst (Preprocessing). Die weitere Vorgehensweise gestaltet sich gleich wie bei den selbst erstellten CNN-Architekturen. An den letzten Residual-Block werden vollverbundene Schichten und ein linearer Klassifizierer angeknüpft.

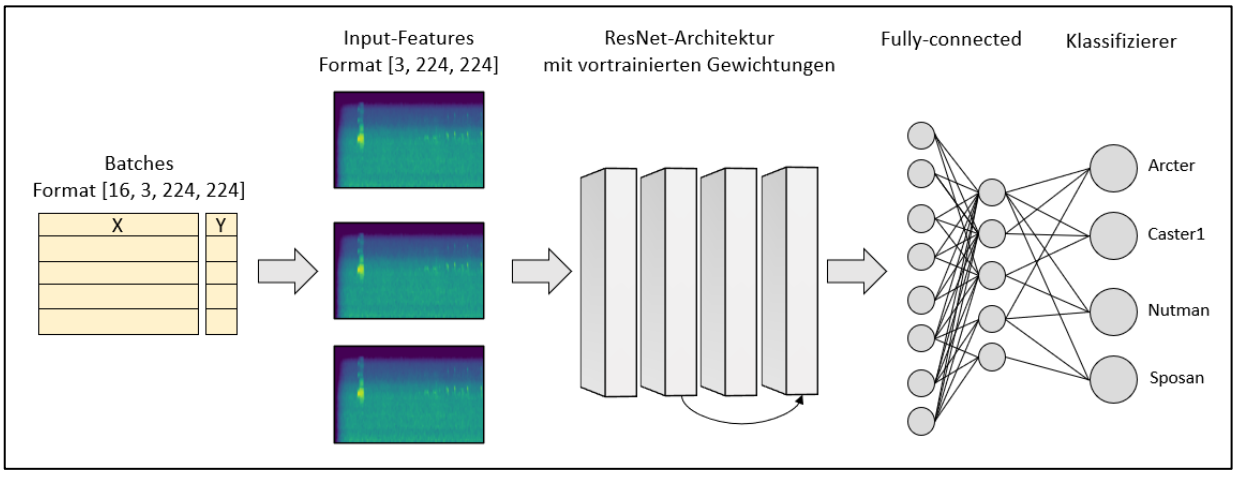
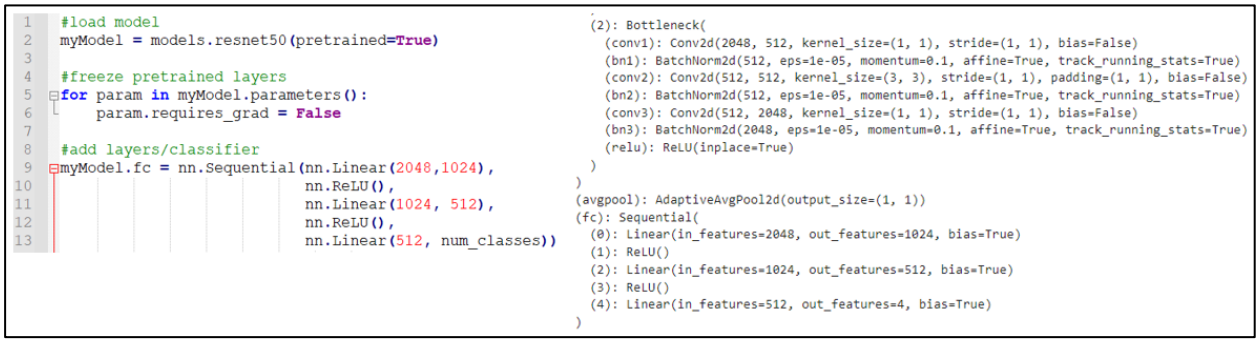


Abbildung 44: Schematische Darstellung mit ResNet-Architektur, Quelle: Eigene Darstellung.

[142] Vgl. He u.a. (2015), S. 1.

[143] Vgl. PyTorch 5 (o.J.), Online-Quelle [17.11.2022].

Abbildung 45 (links) zeigt die Umsetzung mit PyTorch. Zunächst wird die entsprechende Architektur (hier ResNet-50) mit den vortrainierten Gewichtungen aus der Torchvision-Bibliothek geladen. Im Anschluss werden alle Parameter eingefroren. Das bedeutet, dass diese während eines Trainingsdurchlaufes nicht weiter angepasst werden. Dadurch soll das bereits erlernte Wissen auf die betreffende Aufgabenstellung übertragen und somit eine verkürzte Trainingszeit erreicht werden. Während des Trainings werden lediglich die angeknüpften Schichten trainiert. In diesem Fall sind das insgesamt drei lineare, vollverbundene Schichten mit ReLU-Aktivierungsfunktion. Rechts im Bild ist der Anknüpfungspunkt dieser Schichten zu sehen. Im Zuge der Implementierung wird die Klasse *PRE* erstellt, die in der gleichnamigen *PRE.py*-Datei ausgelagert ist. Der Modellname muss bei der Objektinitalisierung (*PRE*) übergeben werden. Für das Verständnis und aufgrund der Übersichtlichkeit ist dies hier anders dargestellt.



```
#load model
myModel = models.resnet50(pretrained=True)

#freeze pretrained layers
for param in myModel.parameters():
    param.requires_grad = False

#add layers/classifier
myModel.fc = nn.Sequential(nn.Linear(2048,1024),
                           nn.ReLU(),
                           nn.Linear(1024, 512),
                           nn.ReLU(),
                           nn.Linear(512, num_classes))
```

```
(2): Bottleneck(
  (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
)
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(fc): Sequential(
  (0): Linear(in_features=2048, out_features=1024, bias=True)
  (1): ReLU()
  (2): Linear(in_features=1024, out_features=512, bias=True)
  (3): ReLU()
  (4): Linear(in_features=512, out_features=4, bias=True)
)
```

Abbildung 45: Verwendung vortrainierter Modelle mit PyTorch (links) und Ausgabe der (reduzierten) Modellarchitektur (rechts), Quelle: Eigene Darstellung.

6.5 Training und Validierung

Das Training und die anschließende Validierung übernehmen eine zentrale Rolle im Zuge eines ML-Projektes. Je nach Menge und Format der Daten können Trainingszeiten einen wesentlichen Zeit- und Kostenfaktor für Unternehmen darstellen. Im Folgenden wird auf die Implementierung der Trainingsschleife im Zuge dieser Arbeit eingegangen. Zunächst werden jedoch zwei relevante Hyperparameter erläutert:

Lernrate (*lr*)

Die Lernrate gibt vor, wie stark sich einzelne Neuronen bei der Parameteraktualisierung während des Trainings anpassen. Sie hat somit unmittelbaren Einfluss auf die Trainingszeit sowie auf das Trainingsergebnis. Bei zu klein gewählter Lernrate wird der Fehler nur langsam minimiert, woraus hohe Trainingszeiten resultieren. Bei einer zu hohen Lernrate werden die Parameter so stark angepasst, dass das Fehlerminimum nicht ermittelt werden kann.[144]

Optimierer (*optimizer*)

Da das Parametrieren der Lernrate eine große Herausforderung beim Training von tiefen Netzwerken darstellt, wurden Optimierungsalgorithmen entworfen, die eine adaptive Lernratenanapassung ermöglichen. Dabei wird versucht, die Lernrate über den ganzen Trainingsprozess so zu modifizieren, dass keiner der beiden zuvor erläuterten Fälle eintritt.

[144] Vgl. Buduma (2017), S. 21.

Adam ist einer der zurzeit am häufigsten verwendeten Optimierungsalgorithmen.[145] Dieser wird auch in der vorliegenden Arbeit für alle Trainingsvorgänge verwendet.

Als Grundlage für die Python-Implementierung der Trainingsschleife dient ein PyTorch-Tutorial für die Bildklassifizierung.[146] Die Klasse *Training* wird in die gleichnamige Datei *Training.py* ausgelagert. Ein Member der Klasse ist die Methode *train()*, die als Übergabeparameter die folgenden Objekte erwartet:

- *model* – Ein Objekt der Klasse *CNN*, *LSTM*, *GRU* oder *PRE*. Enthält eine der in 6.4 erläuterten Modellarchitekturen,
- *train_dl* – Ein *DataLoader*-Objekt. Enthält den Trainingsdatensatz (siehe Abschnitt 6.3),
- *val_dl* – Ein *DataLoader*-Objekt. Enthält den Validierungsdatensatz (siehe Abschnitt 6.3),
- *num_epochs* – Parameter zur Bestimmung der Anzahl der Trainingsepochen,
- *lr* – Parameter zur Bestimmung der Lernrate,
- *device* – CPU, in die das Modell und die Daten (Mel-Spektrogramme) geladen werden.

Abbildung 46 visualisiert den Programmablauf während der Trainings- (rot) und Validierungsphase (orange). Bei Aufruf von *train()* wird das übergebene Modell zunächst auf die CPU geladen. Zudem erfolgt eine Initialisierung relevanter Hyperparameter (*lr*, *Adam-optimizer*). Das Training ist ein iterativer Prozess. Es wird jeweils ein Stapel aus dem *DataLoader*-Objekt in die CPU geladen und an das Modell übergeben. Der *Adam-optimizer* aktualisiert die Parameter des neuronalen Netzwerkes und mindert, sofern möglich, den Verlust. Abschließend werden Modellvorhersagen durchgeführt und mit den tatsächlichen Labels verglichen. Die Anzahl der Vorhersagen korreliert dabei mit der Anzahl der Mel-Spektrogramme des Stapels. Zur Bewertung der Modell-Performance werden mehrere Metriken erfasst. Auf diese wird in Abschnitt 7 eingegangen. Der beschriebene Vorgang wiederholt sich, bis alle Batches abgearbeitet sind. Im Anschluss folgt die Validierungsphase. Diese ist nahezu ident mit der Trainingsphase, jedoch erfolgt im Zuge der Validierung keine Parameteroptimierung. Es werden die Vorhersageergebnisse betrachtet und die Metriken erfasst. Diese Phase dient somit lediglich zur Überprüfung der Generalisierungsfähigkeit des Modells. Die Batch-Größe des Validierungsdatensatzes kann von jener des Trainingsdatensatzes abweichen. Nach Abschluss der Trainings- und Validierungsphase ist eine Epoche beendet. Je nach Konfiguration (*num_epochs*) wiederholt sich der gesamte Ablauf beliebig oft. Modelle können entweder nach dem Training oder währenddessen gespeichert werden. Der Workflow der Testphase entspricht jenem der Validierungsphase und wird nicht gesondert angeführt. Die Testphase dient zur abschließenden Ermittlung der Modell-Performance.

[145] Vgl. Buduma (2017), S. 78 – 81.

[146] Vgl. PyTorch 6 (2022), Online-Quelle [17.11.2022].

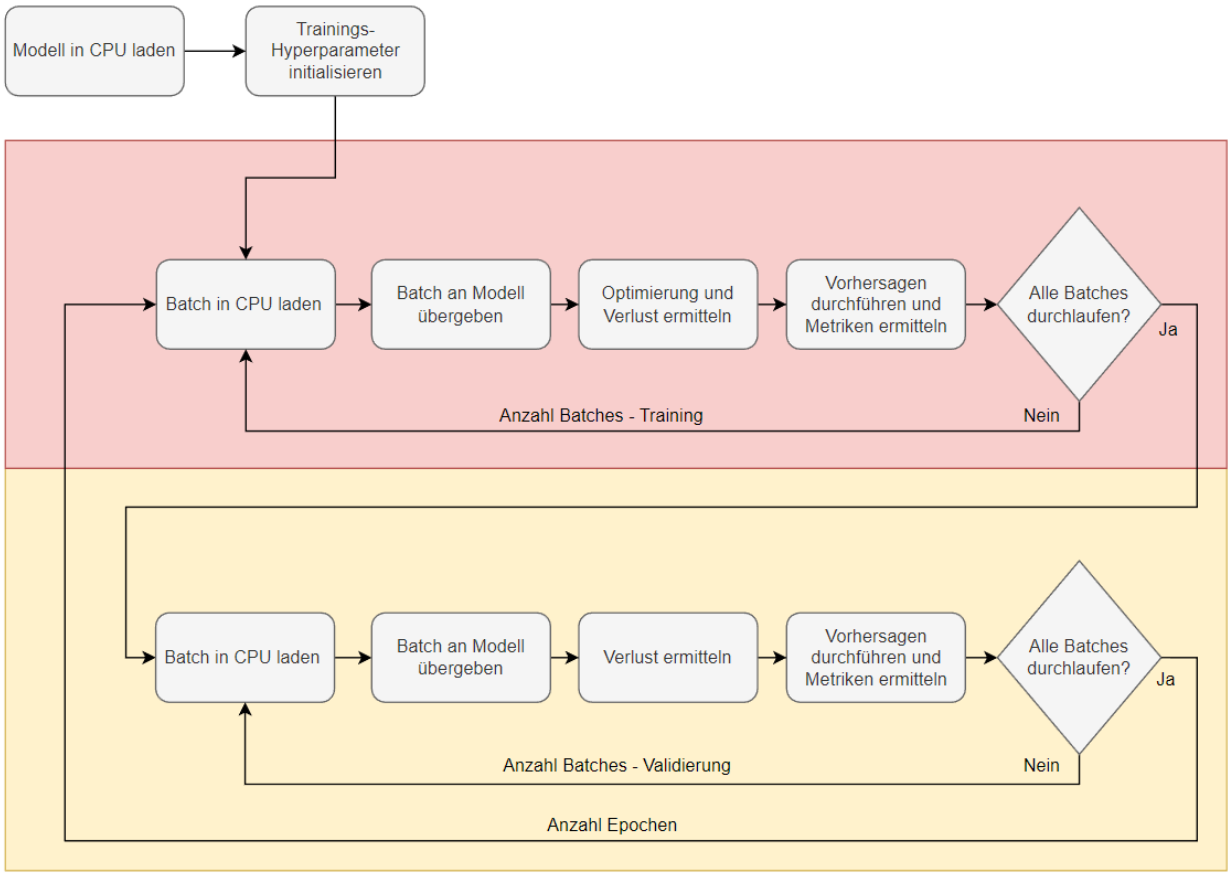


Abbildung 46: Programmablauf Training und Validierung, Quelle: Eigene Darstellung.

7 EVALUIERUNG

In diesem Abschnitt werden die Ergebnisse der durchgeführten Experimente dargelegt und diskutiert. Die Vorgehensweise richtet sich nach der in Kapitel 5.3.2 angeführten Evaluierungsstrategie. Ferner wird ein Vergleich der selbst erstellten Modelle sowie der AutoML-Lösung durchgeführt.

Als Bewertungsgrundlage dienen diverse Metriken, die sich im Klassifizierungsbereich etabliert haben. Zunächst wird auf diese, unter Verwendung der gängigeren englischen Bezeichnungen, eingegangen:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \quad (7.1)$$

$$P = \frac{TP}{TP + FP} \quad (7.2)$$

$$R = \frac{TP}{TP + FN} \quad (7.3)$$

$$F1 = \frac{2PR}{P + R} \quad (7.4)$$

$$L_{CE} = -\sum_{c=1}^{N} y_c \cdot \log\,(p_c) \quad (7.5)$$

TP	True positive
TN	True negative
FP	False positive
FN	False negative
ACC	Accuracy
P	Precision
R	Recall
$F1$	F1-Score
L_{CE}	Cross-entropy loss
c	Klasse
N	Anzahl Klassen
y_c	Binärer Indikator, 0 = Falsche Vorhersage, 1 = Richtige Vorhersage
p_c	Vorhergesagte Wahrscheinlichkeit der Klasse

Diese fünf Metriken werden im Folgenden kurz erläutert:[147]

- Accuracy – Liegt im Intervall [0, 1]. Je näher der Wert bei 1 liegt, desto öfter trifft der Klassifizierer die richtige Entscheidung.
- Precision – Liegt im Intervall [0, 1]. Je näher der Wert bei 1 liegt, desto eher ist die positive Vorhersage korrekt.
- Recall – Liegt im Intervall [0, 1]. Je näher der Wert bei 1 liegt, desto häufiger werden positive Proben erkannt.
- F1-Score – Harmonisches Mittel aus Precision und Recall.
- Cross-entropy loss – Liegt im Intervall [0, ∞). Je näher der Wert bei 0 liegt, desto geringer ist der Fehler. Ein Modell mit keinem Fehler erbringt die bestmögliche Performance.

[147] Vgl. Microsoft Corporation 5 (2022), Online-Quelle [12.09.2022].

Des Weiteren wird auf die folgenden vier Zwischenstatistiken anhand eines binären Klassifizierers mit gegebener Klasse *c* eingegangen:[148]

- True positive – Sowohl der Ausgang des Systems (Vorhersage) als auch die aktuelle Probe entsprechen der Klasse *c*.
- True negative – Weder Probe noch Vorhersageergebnis entsprechen der Klasse *c*.
- False positive – Das Vorhersageergebnis entspricht der Klasse *c*, jedoch liegt diese nicht vor.
- False negative – Das Vorhersageergebnis entspricht nicht der Klasse *c*, jedoch liegt diese vor.

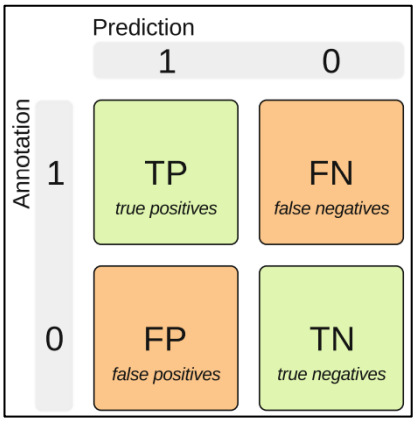


Abbildung 47: Konfusionsmatrix eines binären Klassifizierers, Quelle: Ellis/Plumbley/Virtanen (2018), S. 170.

Da Accuracy eine einfach nachzuvollziehende Metrik ist, eignet sie sich gut für die Bewertung von Klassifizierungsaufgaben. Das bezieht sich im Speziellen auf balancierte Datensätze, weil Accuracy durch das Verhältnis aller richtigen Vorhersagen (TN, TP) zu allen Vorhersagen gebildet wird. In dieser Arbeit wird jedoch ein nicht balancierter Datensatz verwendet. Die alleinige Betrachtung der Genauigkeit kann somit zu Trugschlüssen führen, denn Klassen mit nur wenigen Proben haben eine geringere Auswirkung auf das Gesamtergebnis als jene mit vielen Proben. Deshalb werden zusätzlich Precision, Recall und der F1-Score gemessen. Ermittelt werden die Metriken mittels der Python-Bibliothek Scikit-Learn. Es ist zu beachten, dass Precision, Recall und der F1-Score für Multi-Klassen-Klassifizierungsaufgaben mit nicht balancierten Datensätzen mit der Parameterkonfiguration *micro-average* erfasst werden müssen. Hierbei werden die Werte unter Berücksichtigung der verschiedenen Klassengewichtungen gemittelt.[149]

Die Erfassung der Vorhersageergebnisse erfolgt nach jedem Batch-Durchlauf. Abbildung 48 zeigt eine solche Batch-Vorhersage mit der Batch-Größe 16. Die rot markierten Bereiche entsprechen falschen Vorhersageergebnissen. Verglichen wird die jeweilige Klassen-ID.

```
predictions: tensor([3, 5, 5, 1, 1, 3, 1, 2, 8, 5, 0, 2, 1, 3, 0, 5])
labels:      tensor([5, 5, 5, 1, 5, 3, 1, 2, 8, 5, 0, 2, 1, 3, 0, 5])
```

Abbildung 48: Batch-Vorhersage, Quelle: Eigene Darstellung.

Die Ergebnisse werden in CSV-Dateien geloggt. Zur Reduktion der Datenmenge werden lediglich die Ergebnisse nach jeder Trainingsepoche in die CSV-Datei gespeichert. Es wird ein Zeileneintrag mit elf Werten erstellt. Dabei werden jeweils die fünf beschriebenen Metriken für das Training und die Validierungsphase sowie die benötigte Trainingszeit geloggt. Ein Trainingsdurchlauf mit 100 Epochen enthält somit 1100 Einträge.

[148] Vgl. Ellis/Plumbley/Virtanen (2018), S. 167.

[149] Vgl. Scikit-Learn (o.J.), Online-Quelle [17.11.2022].

Jede CSV-Datei wird mit einem Namen mit der folgenden Struktur gespeichert:

Datensatz_Nummer_Modellname_Epochen_Batch-Größe.csv

Dies lässt sich wie folgt beschreiben:

- Datensatz – reduzierter Datensatz = R, vollständiger Datensatz = F,
- Nummer – Anzahl des n-ten Trainingsdruchlaufes,
- Modellname – Name der Modellarchitektur und notwendige Zusatzinformationen für die korrekte Identifikation, z. B. CNN1,
- Anzahl Epochen – Anzahl der parametrierten Trainingsdurchläufe, z. B. E20,
- Batch-Größe – Anzahl der parametrierten Batch-Size, z. B. BS16.

Beispiel

R_34_CNN1_E50_BS16 – Der insgesamt 34. Trainingsdurchlauf mit reduziertem Datensatz und CNN1-Architektur bei einer Stapelgröße von 16 und insgesamt 50 Trainingsepochen.

7.1 Experimente Teil 1

Im Fokus der ersten Experimentierphase liegt die Ermittlung einer geeigneten Modellarchitektur für den verwendeten Datensatz. Dazu werden die in Abschnitt 6.4 beschriebenen Modelle mit unterschiedlichen Hyperparametereinstellungen (Anzahl Schichten, Anzahl Features etc.) getestet. Damit eine neutrale und vergleichbare Testumgebung geschaffen werden kann, müssen zunächst einige Entscheidungen getroffen werden. Dies bezieht sich insbesondere auf die Konfiguration aller Hyperparameter, die nicht unmittelbar mit der Architektur des Modells in Verbindung stehen. Deshalb werden vorab Default-Parameter definiert, die für die gesamte erste Experimentierphase als Basis verwendet werden. Als Richtlinie zur Ermittlung geeigneter Parameter dienen wissenschaftliche Arbeiten, auf die in 3.1.2 verwiesen wird, sowie Online-Tutorials, auf die in 5.4 verwiesen wird. Um Trends erkennen zu können, werden außerdem Miniexperimente mit einer CNN-Architektur eines PyTorch-Tutorials durchgeführt. Zur besseren Nachvollziehbarkeit und aus Übersichtsgründen werden in Tabelle 9 die zentralen Default-Einstellungen gelistet:

Hyperparameter	**Workflow**	**Einstellung**	**Begründung**
Abtastrate (*sr*)	Preprocessing	32 000 Hz	Auswahl aufgrund von Datenanalyse.
Signallänge (*duration*)	Preprocessing	10 s	Auswahl aufgrund von Datenanalyse.
Anzahl Kanäle (*channels*)	Preprocessing	2 (CNN / RNN) 3 (ResNet)	Auswahl aufgrund von Datenanalyse bzw. Vorgabe durch ResNet-Architektur.
Schrittweite (*hop_size*)	Preprocessing	512	Auswahl aufgrund von Miniexperimenten.
Mel-Bänder (*n_mels*)	Preprocessing	64	Auswahl aufgrund von Miniexperimenten.
Hochpassfilter (*f_min*)	Preprocessing	0 Hz	Auswahl aufgrund von Tutorials und Paper.
Tiefpassfilter (*f_max*)	Preprocessing	16 000 Hz	Auswahl aufgrund von Tutorials und Paper.
FFT-Größe (*n_fft*)	Preprocessing	1024	Auswahl aufgrund von Miniexperimenten.
Fenstergröße (*win_length*)	Preprocessing	1024	Default-Einstellung PyTorch win_length = n_fft.
Fensterfunktion (*window_fn*)	Preprocessing	Hanning-Fenster	Auswahl aufgrund von Tutorials und Paper.
Datensatz	Training	Reduziert / vollständig	Auswahl aufgrund von Evaluierungsstrategie. Phase 1 = reduziert, Phase 2 = vollständig.
Anzahl Klassen (*num_classes*)	Training	4 / 9	Auswahl aufgrund des verwendeten Datensatzes.
Batch-Größe (*batch_size*)	Training	16	Auswahl aufgrund begrenzten Arbeitsspeichers.
Epochen (num_*epochs*)	Training	20 / 50 / 100	Auswahl aufgrund von Miniexperimenten.
Lernrate (*lr*)	Training	0,001	Auswahl aufgrund von Miniexperimenten.
Optimierer (*optimizer*)	Training	Adam-optimizer	Auswahl aufgrund von Tutorials und Paper.

Tabelle 9: Default-Parametereinstellungen zur Ermittlung einer geeigneten Modell-Architektur, Quelle: Eigene Darstellung.

Ergänzende Anmerkungen zur Tabelle

Während der Laufzeit werden die einzelnen Stapel in die CPU geladen und die neuronalen Netzwerke trainiert. Dafür wird ein Arbeitsspeicher der CPU (Anzahl Daten, temporäre Zwischenspeicher) reserviert. Die Menge des benötigten Arbeitsspeichers steht hierbei in Beziehung zur parametrierten Batch-Größe. Grundsätzlich wird die Verwendung von möglichst großen Stapelgrößen empfohlen. Entgegen diesen Empfehlungen wird mit der Größe 16 eine kleine Dimension gewählt. Dies ergibt sich aufgrund der zur Verfügung stehenden Hardware-Ressourcen, die aus finanziellen Gründen beschränkt sind. Die gewählte Einstellung stellt einen Kompromiss dar, denn einerseits erhöhen sich mit kleinen Stapelgrößen die Trainingszeiten und andererseits können aufgrund der geringeren Speicherauslastung mehrere Trainingsprozesse parallel betrieben werden. Insgesamt wirkt sich Letzteres positiv auf den Gesamtzeitaufwand aus.

Nachdem die benötigten Hyperparametereinstellungen definiert sind, wird in den folgenden Abschnitten auf die Ergebnisse der getesteten Architekturen eingegangen. Zur Verringerung der Trainingszeiten wird in der ersten Experimentierphase ausschließlich der reduzierte Datensatz verwendet. Ferner werden alle Experimente dieser Phase mit 20 bzw. 50 Epochen durchgeführt. Im Verlauf der Miniexperimente hat sich gezeigt, dass dies zur Erkennung von Trends ausreichend ist. Für die zweite Experimentierphase werden 100 Trainingsepochen als Obergrenze festgelegt.

Zunächst werden die Ergebnisse je Architekturart gelistet. Abschließend erfolgt ein Vergleich der besten Modelle. Die Metriken Precision und Recall sind in Form des F1-Scores (harmonischer Mittelwert) dargestellt und werden aus Übersichtsgründen nicht eigens in den Tabellen angeführt. Die Tabellen enthalten je Metrik den jeweils besten Wert aller Trainingsdurchläufe.

7.1.1 CNN

Zuerst werden die Ergebnisse der vier in Abschnitt 6.4.1 implementierten Modelle mit CNN-Architektur in Tabelle 10 gelistet. Diese teilt sich in die zwei Abschnitte 20 Epochen (oben) und 50 Epochen (unten):

	Training			**Validierung**		
Bezeichnung	L_{CE}	***ACC*/%**	***F1*/%**	L_{CE}	***ACC*/%**	***F1*/%**
R_T51_CNN1_E20_BS16	0,777	70,5	56,1	**1,034**	**60,9**	**50,6**
R_T24_CNN2_E20_BS16	0,457	85,2	67,6	1,117	57,4	44,5
R_T25_CNN3_E20_BS16	**0,291**	**89,3**	**71,6**	1,254	59,4	46,9
R_T26_CNN4_E20_BS16	0,372	87,3	66,0	1,872	53,9	40,2
R_T50_CNN1_E50_BS16	0,485	83,2	65,1	**0,837**	**67,8**	**58,5**
R_T39_CNN2_E50_BS16	0,230	91,7	76,0	0,971	61,7	49,1
R_T40_CNN3_E50_BS16	**0,155**	**93,7**	**79,6**	0,993	66,1	53,3
R_T45_CNN4_E50_BS16	0,223	93,2	76,6	1,592	56,5	47,4

Tabelle 10: Trainings- und Validierungsergebnisse CNN-Architekturen, Quelle: Eigene Darstellung.

Bereits nach 20 Durchläufen können mit allen vier Modellen gute Trainingsergebnisse erreicht werden. Insbesondere das CNN3-Modell erzielt bezogen auf die Trainingsmenge hohe Werte für Accuracy und den F1-Score. Generell fällt auf, dass alle Modelle zur Überanpassung neigen, da Trainings- und Validierungsergebnisse deutlich voneinander abweichen. Dieser Effekt verstärkt sich mit zunehmender Tiefe der Modelle. Komplexere Architekturen sind daher für den entsprechenden Datensatz weniger geeignet. Dennoch ist ersichtlich, dass sich die Validierungsergebnisse verglichen mit den Trainingsergebnissen bei 50 Epochen stärker erhöhen. Bezogen auf die Generalisierungsfähigkeit der Modelle kann dies als positiv wahrgenommen werden. Das CNN1-Modell mit der geringsten Komplexität weist die geringste Differenz zwischen Trainings- und Validierungsergebnissen auf und scheint somit am robustesten gegen Overfitting zu sein. Hingegen ist das CNN3-Modell schneller lernfähig. Für die weiterführende Betrachtung in Abschnitt 7.1.4 sollen beide einbezogen werden. Die Modelle CNN2 und CNN4 werden verworfen.

7.1.2 RNN – LSTM und GRU

Bei den LSTM- bzw. GRU-Architekturen werden jeweils zwölf Modelle trainiert, wobei die beiden Hyperparameter *hidden_size* (Anzahl Output-Features) und *num_layers* (Anzahl Schichten) variiert werden. Die Einstellwerte sind aus der Dateibezeichnung (HS und L) ersichtlich. Tabelle 11 enthält die Ergebnisse der trainierten LSTM-Architekturen. Zunächst wird die Anzahl der Output-Features angepasst (oben) und anschließend die Anzahl der Schichten verändert (unten):

	Training			**Validierung**		
Bezeichnung	L_{CE}	***ACC*/%**	***F1*/%**	L_{CE}	***ACC*/%**	***F1*/%**
R_T1_LSTM_HS626_L2_E20_BS16	1,331	33,6	35,0	1,325	36,5	31,2
R_T5_LSTM_HS1000_L2_E20_BS16	1,330	34,3	35,5	1,315	34,8	36,2
R_T9_LSTM_HS300_L2_E20_BS16	**1,319**	**36,9**	**38,5**	**1,261**	**43,5**	**38,2**
R_T12_LSTM_HS1252_L2_E20_BS16	1,324	34,5	36,5	1,348	34,8	38,1
R_T13_LSTM_HS500_L2_E20_BS16	1,336	35,8	35,7	1,310	36,5	33,9
R_T19_LSTM_HS2000_L2_E20_BS16	1,330	33,8	32,2	1,329	35,7	36,2
R_T22_LSTM_HS4000_L2_E20_BS16	1,335	34,3	35,1	1,279	33,9	36,7
R_T9_LSTM_HS300_L2_E20_BS16	**1,319**	**36,9**	**38,5**	**1,261**	**43,5**	**38,2**
R_T44_LSTM_HS300_L5_E20_BS16	1,314	35,4	35,6	1,316	31,3	38,6
R_T45_LSTM_HS300_L3_E20_BS16	1,314	35,8	35,9	1,358	32,2	36,4
R_T46_LSTM_HS300_L4_E20_BS16	1,340	35,6	36,6	1,274	39,1	36,5
R_T49_LSTM_HS300_L1_E20_BS16	1,338	33,0	36,3	1,286	39,3	36,0

Tabelle 11: Trainings- und Validierungsergebnisse LSTM-Architekturen, Quelle: Eigene Darstellung.

Die Lernerfolge sind im Vergleich zu den CNN-Architekturen wesentlich geringer. Generell sind zumindest bei 20 Epochen nur wenige Auswirkungen der Hyperparameteränderungen auf das Ergebnis erkennbar. Das Modell mit einer Anzahl von zwei Schichten und 300 Output-Features hebt sich etwas von den anderen Modellen ab. Hier liegen die Werte für Trainings- und Validierungsergebnisse nah beieinander. Üblicherweise werden LSTM-Architekturen für Sprach- oder Textanalysen verwendet. Bei diesen Aufgabenstellungen stehen die einzelnen Sequenzen miteinander in Beziehung. Dies ist beim verwendeten Datensatz nicht der Fall, was eine mögliche Erklärung für das schlechte Lernverhalten sein kann. Jedoch weisen die Lernkurven der verschiedenen Architekturen einen steigenden bzw. sinkenden (Fehler-)Trend auf (aus CSV-Datei ersichtlich). Dementsprechend müssten die Trainingsdurchläufe erhöht werden. Aufgrund der großen Abweichung von den CNN-Modellen wird dies nicht durchgeführt. Die LSTM-Architekturen werden somit nicht weiter behandelt.

Tabelle 12 zeigt die Ergebnisse der zwölf trainierten GRU-Modelle. Die Tabelle gliedert sich in drei Abschnitte. Zunächst variiert die Anzahl der Output-Features (oben) und im Anschluss die Anzahl der Schichten (Mittelteil). Abschließend wird das beste Modell mit 50 Trainingsdurchläufen trainiert (unten):

	Training			**Validierung**		
Bezeichnung	L_{CE}	***ACC*/%**	***F1*/%**	L_{CE}	***ACC*/%**	***F1*/%**
R_T4_GRU_HS626_L2_E20_BS16	1,144	44,3	35,3	**1,225**	38,3	32,7
R_T6_GRU_HS1000_L2_E20_BS16	1,314	38,6	37,2	1,346	34,8	31,8
R_T10_GRU_HS300_L2_E20_BS16	1,189	43,0	32,8	1,333	38,3	36,6
R_T14_GRU_HS1252_L2_E20_BS16	1,289	38,9	36,5	1,349	38,3	36,0
R_T16_GRU_HS500_L2_E20_BS16	**0,959**	**55,2**	**38,5**	1,288	**43,5**	**38,6**
R_T21_GRU_HS2000_L2_E20_BS16	1,331	34,1	31,6	1,304	40,0	27,2
R_T23_GRU_HS4000_L2_E20_BS16	1,320	35,4	33,1	1,337	40,9	37,9
R_T14_GRU_HS500_L2_E20_BS16	1,141	44,5	**38,5**	**1,266**	**43,5**	**38,6**
R_T43_GRU_HS500_L3_E20_BS16	1,263	38,6	38,1	1,296	38,3	36,1
R_T44_GRU_HS500_L4_E20_BS16	1,256	38,9	36,5	1,288	33,9	35,6
R_T47_GRU_HS500_L5_E20_BS16	1,255	39,1	37,1	1,317	39,1	36,7
R_T48_GRU_HS500_L1_E20_BS16	**1,049**	**48,7**	38,3	1,378	41,7	38,2
R_T52_GRU_HS500_L2_E50_BS16	**0,654**	**72,9**	**51,8**	**0,972**	**59,2**	**44,3**

Tabelle 12: Trainings- und Validierungsergebnisse GRU-Architekturen, Quelle: Eigene Darstellung.

Verglichen mit den LSTM-Modellen können bereits bei 20 Trainingsdurchläufen signifikant bessere Ergebnisse erreicht werden, wobei sich die Modelle mit einer geringeren Anzahl an Output-Features (HS300, HS500 und HS626) sowohl in der Trainings- als auch in der Validierungsphase etablieren. Insbesondere jenes mit 500 Output-Features liefert in beiden Phasen die besten Ergebnisse, weshalb dieses als Grundlage für die Variation der Schichten verwendet wird. Wie ersichtlich ist, sinkt die Modell-Performance bei tieferen Modellen (mehr Schichten). Ebenso wie bei den CNN-Architekturen, führen auch bei den GRU-Architekturen Modelle mit geringerer Komplexität zu einem besseren Ergebnis. Obwohl das Modell GRU_HS500_L1 die besseren Trainingsergebnisse erzielt, wird für das erweiterte Training jenes mit zwei Schichten genutzt. Dies kann mit der besseren Generalisierungsfähigkeit begründet werden (Validierung). Der Vergleich mit den anderen Architekturarten erfolgt in Abschnitt 7.1.4.

7.1.3 Vortrainierte Modelle

Die Ergebnisse der implementierten ResNet-Architekturen sind in Tabelle 13 gelistet. Die Tabelle gliedert sich in zwei Sektoren. Zunächst werden 20 Trainingsdurchläufe durchgeführt (oben) und im Anschluss 50 (unten):

	Training			**Validierung**		
Bezeichnung	L_{CE}	***ACC*/%**	***F1*/%**	L_{CE}	***ACC*/%**	***F1*/%**
R_T2_ResNet18_E20_BS16	1,282	40,8	26,7	1,350	35,7	36,7
R_T3_ResNet34_E20_BS16	**1,268**	**42,1**	31,2	**1,229**	41,7	38,9
R_T11_ResNet50_E20_BS16	1,311	39,3	**34,1**	1,294	**44,4**	**39,8**
R_T15_ResNet18_E50_BS16	1,201	43,7	32,6	1,238	42,6	38,7
R_T30_ResNet34_E50_BS16	**1,173**	**47,6**	**36,2**	**1,228**	**50,4**	**40,1**
R_T29_ResNet50_E50_BS16	1,188	45,9	35,2	1,236	48,7	39,3

Tabelle 13: Trainings- und Validierungsergebnisse ResNet-Architekturen, Quelle: Eigene Darstellung.

Nach 20 Trainingsdurchläufen liegen die Ergebnisse aller Modelle sowohl in der Trainings- als auch in der Validierungsphase nah beieinander. Eine eindeutige Aussage, welches Modell sich am besten für den verwendeten Datensatz eignet, ist nicht möglich, weshalb alle Modelle mit einer Anzahl von 50 Trainingsepochen trainiert werden. Generell weisen die Modelle eine hohe Generalisierungsfähigkeit auf, was vermutlich daran liegt, dass diese mit einem umfangreichen Benchmark-Datensatz vortrainiert wurden. Das Modell mit ResNet-34-Architektur erzielt in beiden Phasen die besten Ergebnisse und soll somit für den weiteren Vergleich im folgenden Abschnitt verwendet werden.

7.1.4 Gesamtauswertung Teil 1

In Tabelle 14 werden die jeweils besten Ergebnisse der unterschiedlichen Architekturen zusammengefasst. Nicht inkludiert sind die LSTM-Modelle, da mit diesen im Verlauf des Trainings keine guten Ergebnisse erzielt werden konnten. Alle Ergebnisse beziehen sich auf 50 Trainingsepochen:

	Training			**Validierung**		
Bezeichnung	L_{CE}	**ACC/%**	**F1/%**	L_{CE}	**ACC/%**	**F1/%**
R_T50_CNN1_E50_BS16	0,485	83,2	65,1	**0,837**	**67,8**	**58,5**
R_T40_CNN3_E50_BS16	**0,155**	**93,7**	**79,6**	0,993	66,1	53,3
R_T52_GRU_HS500_L2_E50_BS16	0,654	72,9	51,8	0,972	59,2	44,3
R_T30_ResNet34_E50_BS16	1,173	47,6	36,2	1,228	50,4	40,1

Tabelle 14: Experimente Teil 1 – Vergleich der Ergebnisse, Quelle: Eigene Darstellung.

Mit allen vier Modellarten können passable Ergebnisse erzielt werden. Es zeigt sich, dass jene mit CNN-Architektur die Muster wesentlich schneller erlernen. Bereits nach 50 Epochen weicht die Validierungsgenauigkeit um mehr als 17 % zwischen CNN1 und ResNet-34 ab. Dennoch weist das ResNet-Modell die geringste Differenz zwischen Trainings- und Validierungsergebnissen auf. Die niedrigen Werte für Accuracy und F1-Score sowie die hohen Werte für den Verlust sind aber ein Anzeichen dafür, dass das vortrainierte Wissen nicht auf die verwendeten Daten übertragen werden kann, was aber aufgrund der hohen Dynamik der Daten (Unstetigkeit, Umgebung etc.) nicht überraschend ist. Eventuell eignen sich andere vortrainierte Modelle (z. B. VGG-Net oder YAMNET) besser für die Daten. Dies soll aber im Verlauf dieser Arbeit nicht weiter beachtet werden. Das GRU-Modell neigt gleich wie die CNN-Modelle zur Überanpassung. Die Differenz zwischen Trainings- und Validierungsergebnissen ist zwar etwas geringer, dennoch lernt das Modell wesentlich langsamer. Da sich die weiteren Trainings auf 100 Trainingsdurchläufe beschränken werden, wird das GRU-Modell nicht weiter betrachtet.

Als abschließende Tests zur Auswahl einer geeigneten Modellarchitektur sollen nun die beiden Modelle CNN1 und CNN3 mit 100 Durchläufen trainiert werden. Dazu wird der vollständige Datensatz mit neun Klassen und 1983 Dateien verwendet. Tabelle 15 zeigt die Ergebnisse des Trainingsvorgangs:

	Training			**Validierung**		
Bezeichnung	L_{CE}	**ACC/%**	**F1/%**	L_{CE}	**ACC/%**	**F1/%**
F_T53_CNN1_E100_BS16	0,292	91,7	73,7	**1,110**	**70,0**	**52,5**
F_T54_CNN3_E100_BS16	**0,104**	**97,0**	**81,7**	1,665	63,5	45,9

Tabelle 15: Experimente Teil 1 – Vergleich von CNN1 und CNN3 mit 100 Trainingsepochen, Quelle: Eigene Darstellung.

Auch bei vollständigem Datensatz und mehr Trainingsdurchläufen verhalten sich die Lerneffekte ähnlich wie beim reduzierten Datensatz. Der Kreuzentropieverlust (L_{CE}) in der Validierungsphase ist wesentlich höher als bei den Trainings mit reduziertem Datensatz. Das ist darauf zurückzuführen, dass beim vollständigen Datensatz mehr Klassen mit nur wenigen Proben enthalten sind.

Zur Veranschaulichung werden die Lernkurven in Abbildung 49 dargestellt. Die Diagramme umfassen die Werte für Accuracy (links) und den Kreuzentropieverlust (rechts) für die Phasen Training (durchgehende Linien) und Validierung (strichlierte Linien). Bereits nach wenigen Epochen driften die Linien für Training und Validierung auseinander. Die Modelle lernen die Muster aus den Trainingsdaten auswendig und können nicht sehr gut auf neue Daten angewendet werden. Mögliche Gegenmaßnahmen wären ein frühzeitiger Trainingsabbruch und ein erneuter Trainingsbeginn, denn vor jedem Training werden die beiden Datensätze für Trainings- und Validierungsphase auf zufälliger Basis ausgewählt und erstellt. Zunächst steht jedoch die Auswahl eines geeigneten Modells im Vordergrund. Anhand der Diagramme wird deutlich, dass sich das CNN1-Modell aufgrund der höheren Robustheit gegen Overfitting wesentlich besser für den gegebenen Datensatz eignet. Dieses wird somit für alle Trainings in der zweiten Experimentierphase verwendet.

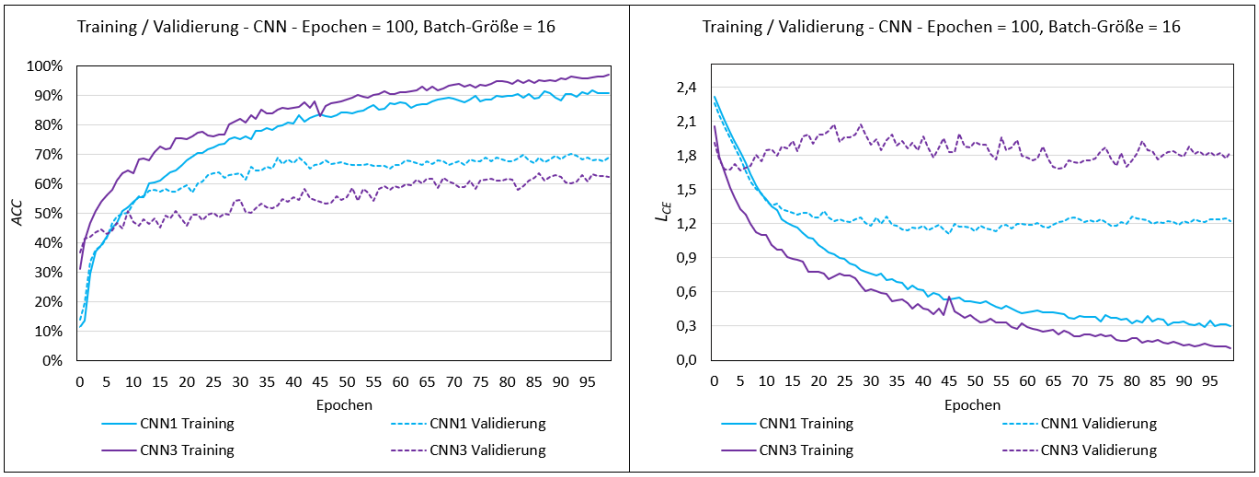


Abbildung 49: Lernkurven CNN1 und CNN3, Epochen = 100, Batch-Größe = 16, Quelle: Eigene Darstellung.

7.2 Experimente Teil 2

Der Fokus dieser Phase liegt auf der Optimierung des in Phase 1 ausgewählten Modells. Es werden relevante Hyperparameter variiert und Maßnahmen zur Prävention von Overfitting vorgestellt sowie angewendet. Des Weiteren wird ein AutoML-Modell trainiert und mit dem optimierten Modell verglichen. Abschließend wird ein Ausblick auf weitere Optimierungsmöglichkeiten gegeben.

7.2.1 Optimierung der Hyperparameter

Zur Reduktion des Umfangs werden nicht alle in Abschnitt 7.1 vorgestellten Parameter variiert. Zunächst erfolgt eine Ermittlung der Parameter, die den größten Einfluss auf das Trainingsergebnis haben. Dies wird anhand von Miniexperimenten realisiert, die manuell abgebrochen und somit nicht mitgeloggt werden. Sämtliche Trainings der zweiten Phase werden mit dem vollständigen Datensatz durchgeführt. Tabelle 16 enthält die verschiedenen Hyperparameter, die im Zuge der zweiten Experimentierphase verändert werden:

Hyperparameter	Workflow	Phase 1	Phase 2
Schrittweite (*hop_size*)	Preprocessing	512	160 / 320 / 1024
Mel-Bänder (*n_mels*)	Preprocessing	64	50 / 128 / 256
FFT-Größe (*n_fft*)	Preprocessing	1024	512 / 2048 / 4096
Fenstergröße (*win_length*)	Preprocessing	1024	256 / 1024 / 2048
Batch-Größe (*batch_size*)	Training	16	4 / 8 / 12
Lernrate (*lr*)	Training	0,001	0,0001 / 0,01 / 0,5

Tabelle 16: Hyperparametervariationen Experimentierphase 2, Quelle: Eigene Darstellung.

Für alle angeführten Hyperparameter werden jeweils drei verschiedene Einstellungen getestet, wobei für jeden Trainingsdurchlauf der betreffende Hyperparameter angepasst wird. Alle restlichen Konfigurationen sind mit jenen der ersten Phase ident. Zu Beginn wird der Parameter *hop_size* variiert. Die jeweilige Einstellung ist aus der Dateibezeichnung ersichtlich (z. B. HOP160). Als Referenzmessung dient das CNN1-Modell mit den Hyperparametereinstellungen aus Phase 1 (orange markiert). Die Ergebnisse sind in Tabelle 17 angeführt:

	Training			Validierung		
Bezeichnung	L_{CE}	***ACC*/%**	***F1*/%**	L_{CE}	***ACC*/%**	***F1*/%**
F_T57_CNN1_HOP160_E20_BS16	0,999	68,5	44,3	1,229	61,5	40,6
F_T58_CNN1_HOP320_E20_BS16	0,981	69,2	**50,2**	**1,110**	**65,7**	**46,3**
F_T59_CNN1_HOP1024_E20_BS16	0,957	**70,0**	47,6	1,207	59,7	40,6
F_T68_CNN1_HOP512_E20_BS16	**0,952**	69,8	47,3	1,184	59,9	41,2

Tabelle 17: Variation der Schrittweite (*hop_size*), Quelle: Eigene Darstellung.

Die Schrittweite hat unmittelbaren Einfluss auf das Format des Mel-Spektrogramms (siehe Abschnitt 6.2), wobei bei kleineren Schritten insgesamt mehr Frames extrahiert werden und somit das Mel-Spektrogramm breiter ist. Bei einer Einstellung von 320 wird ein signifikant besseres Ergebnis erzielt. Insbesondere in der Validierungsphase ist eine deutliche Differenz zu den restlichen Konfigurationen zu erkennen.

Im Anschluss wird der Hyperparameter *n_mels* angepasst. Tabelle 18 zeigt die Ergebnisse der einzelnen Trainingsdurchläufe:

	Training			Validierung		
Bezeichnung	L_{CE}	***ACC*/%**	***F1*/%**	L_{CE}	***ACC*/%**	***F1*/%**
F_T60_MEL_MELS50_E20_BS16	**0,892**	**71,5**	**50,3**	**1,079**	**66,2**	**43,0**
F_T61_MEL_MELS128_E20_BS16	0,911	**71,5**	46,9	1,191	63,2	40,1
F_T62_MEL_MELS256_E20_BS16	1,318	59,4	38,4	1,416	54,9	33,2
F_T68_CNN1_MELS64_E20_BS16	0,952	69,8	47,3	1,184	59,9	41,2

Tabelle 18: Variation der Anzahl der Mel-Bänder (*n_mels*), Quelle: Eigene Darstellung.

Die Anzahl der Mel-Bänder der Filterbank hat unmittelbaren Einfluss auf das Format des Mel-Spektrogramms (siehe Abschnitt 6.2). Je höher die Anzahl, desto höher ist das Mel-Spektrogramm. Wie aus der Tabelle ersichtlich, ist die Konfiguration mit *n_mels* = 50 am besten für den Datensatz geeignet.

Als nächstes werden die Parameter *fft_size* und *win_length* angepasst, wobei diese jeweils im Verhältnis von 2:1 angepasst werden (z. B. *fft_size* = 4096, *win_length* = 2048). Aus Tutorials wird ersichtlich, dass dies eine gängige Vorgehensweise ist. Damit die Berechnung der FFT schneller durchgeführt werden kann, entsprechen die Werte immer einer Zweierpotenz. Die Ergebnisse sind in Tabelle 19 gelistet:

	Training			**Validierung**		
Bezeichnung	L_{CE}	***ACC*/%**	***F1*/%**	L_{CE}	***ACC*/%**	***F1*/%**
F_T62_MEL_FFT512_WIN256_E20_BS16	0,937	70,4	47,6	1,210	64,0	40,9
F_T63_MEL_FFT2048_WIN1024_E20_BS16	0,935	69,6	47,2	1,162	64,0	42,5
F_T63_MEL_FFT4096_WIN2048_E20_BS16	**0,878**	**72,7**	**49,7**	**1,006**	**69,5**	**48,0**
F_T68_CNN1_FFT1024_WIN1024_E20_BS16	0,952	69,8	47,3	1,184	59,9	41,2

Tabelle 19: Variation der FFT-Größe (*n_fft*) und der Fenstergröße (*win_length*), Quelle: Eigene Darstellung.

Die FFT-Größe und die Fenstergröße haben Einfluss auf die Auflösung des Mel-Spektrogramms sowie auf die Größe der Rahmen, die im Zuge der STFT benötigt werden (siehe Abschnitt 6.2). Wie ersichtlich ist, haben höhere Einstellwerte einen deutlichen Einfluss auf die Testergebnisse. Insbesondere die Einstellung mit *fft_size* = 4096 und *win_length* = 2048 hebt sich von den restlichen Hyperparametern ab.

Die veränderten Hyperparameter beziehen sich auf die Preprocessing-Phase und die Mel-Spektrogramm-Extraktion. Im weiteren Verlauf werden die für das Training relevanten Hyperparameter angepasst. Zunächst wird jedoch ein Training mit den jeweils besten Parametereinstellungen für das Preprocessing durchgeführt und mit den Standardeinstellungen aus Phase 1 verglichen. Die neuen Parametereinstellungen sind:

- *hop_size* = 320,
- *n_mels* = 50,
- *n_fft* = 4096,
- *win_length* = 2048.

Tabelle 20 zeigt die Ergebnisse des Vergleichs:

	Training			**Validierung**		
Bezeichnung	L_{CE}	***ACC*/%**	***F1*/%**	L_{CE}	***ACC*/%**	***F1*/%**
F_T67_CNN1_V1_E20_BS16	0,974	69,4	**47,6**	**1,035**	**68,8**	**46,5**
F_T68_CNN1_E20_BS16	**0,952**	**69,8**	47,3	1,184	59,9	41,2

Tabelle 20: Zwischenergebnis nach Variation der Preprocessing-Hyperparameter, Quelle: Eigene Darstellung.

Dabei wird ersichtlich, inwiefern sich die Hyperparametervariationen auf die Generalisierungsfähigkeit des Modells auswirken. Bei einer Anzahl von 20 Trainingsepochen ist nahezu keine Differenz zwischen Trainings- und Validierungsergebnissen vorhanden (CNN1_V1). Dieses Modell soll nun für die weiteren Tests als Referenz und als Basis verwendet werden.

Im Anschluss wird die Batch-Größe variiert, wobei die Größe 16 aufgrund des beschränkten Arbeitsspeichers nicht überschritten wird. Tabelle 21 listet die Trainingsergebnisse:

	Training			**Validierung**		
Bezeichnung	L_{CE}	**ACC/%**	**F1/%**	L_{CE}	**ACC/%**	**F1/%**
F_T69_CNN1_V1_E20_BS4	0,979	70,7	49,0	1,366	56,9	40,7
F_T70_CNN1_V1_E20_BS8	**0,902**	**72,5**	**50,8**	1,125	64,0	43,9
F_T71_CNN1_V1_E20_BS12	0,974	69,4	47,2	1,212	63,5	42,1
F_T67_CNN1_V1_E20_BS16	0,974	69,4	47,6	**1,035**	**68,8**	**46,5**

Tabelle 21: Variation der Batch-Größe (*batch_size*), Quelle: Eigene Darstellung.

Die Batch-Größe hat Einfluss auf die Häufigkeit der Parameterupdates der Gewichtungen im Zuge des Trainings eines neuronalen Netzwerkes (siehe 6.3). Je höher die Batch-Größe, desto weniger oft werden die Parameter aktualisiert. Wie sich zeigt, eignet sich die bisher verwendete Stapelgröße von 16 am besten für den Datensatz (Validierung). Möglichst große Batch-Größen werden auch in der Literatur empfohlen.

Abschließend wird die Lernrate variiert, wobei die Parameter des neuronalen Netzwerkes immer mit dem Adam-Optimierungsalgorithmus angepasst werden. Dieser versucht, die Lernrate für jeden Durchlauf zu optimieren. Die angegebene Lernrate entspricht somit der maximalen Lernrate. Die Ergebnisse sind in Tabelle 22 gelistet:

	Training			**Validierung**		
Bezeichnung	L_{CE}	**ACC/%**	**F1/%**	L_{CE}	**ACC/%**	**F1/%**
F_T72_CNN1_V1_LR10E-4_BS16	1,579	48,9	26,7	1,576	50,6	26,2
F_T73_CNN1_V1_LR10E-2_E20_BS16	**0,564**	**82,2**	**59,2**	**0,898**	**72,8**	**56,0**
F_T74_CNN1_V1_LR5E-2_E20_BS16	0,633	80,2	56,5	0,961	70,0	51,2
F_T67_CNN1_V1_LR10E-3_E20_BS16	0,974	69,4	47,6	1,035	68,8	46,5

Tabelle 22: Variation der Lernrate (*lr*), Quelle: Eigene Darstellung.

Die Lernrate hat darauf Einfluss, wie stark sich die Parameteraktualisierungen während des Trainings anpassen. Je höher die Lernrate, desto stärker werden die Parameter angepasst (siehe Abschnitt 6.5). Dieser Effekt zeigt sich an den Ergebnissen. Dazu sind das Modell mit der höchsten Lernrate (LR10E-2 = 0,01) und das Modell mit der niedrigsten Lernrate (LR10E-4 = 0,0001) zu beachten. Die Trainingsergebnisse Accuracy und F1-Score weichen fast um das Doppelte ab.

Der Kreuzentropieverlust bei erhöhter Lernrate beträgt nach 20 Epochen nur ein Drittel. Obwohl das Modell mit der höchsten Lernrate in allen Kategorien am besten abschneidet, wird entschieden, dass weiterhin die Lernrate mit der Einstellung 0,001 (LR10E-3) verwendet wird, denn damit verhält sich das Modell robuster gegen Overfitting und weist dennoch eine akzeptable Lerngeschwindigkeit auf.

Zusammengefasst ergeben sich somit die folgenden Hyperparameterkonfigurationen für die weiteren Tests:

- *hop_size* = 320,
- *n_mels* = 50,
- *n_fft* = 4096,
- *win_length* = 2048,
- *batch_size* = 16,
- *lr* = 0,001.

Im nächsten Schritt wird das CNN1-Modell (CNN1_V1) mit den neuen Parametereinstellungen und 100 Trainingsepochen trainiert und mit dem CNN1-Modell (CNN1) der ersten Experimentierphase verglichen.

	Training			**Validierung**		
Bezeichnung	L_{CE}	***ACC*/%**	***F1*/%**	L_{CE}	***ACC*/%**	***F1*/%**
F_T53_CNN1_E100_BS16	0,292	91,7	73,7	1,110	70,0	52,5
F_T76_CNN1_V1_E100_BS16	**0,113**	**96,1**	**75,9**	**0,622**	**78,9**	**65,8**

Tabelle 23: Vergleich der CNN1-Architekturen aus Experimentierphase Teil 1 und mit optimierten Hyperparametereinstellungen, Quelle: Eigene Darstellung.

Wie ersichtlich ist, können sowohl Trainings- als auch Validierungsergebnisse signifikant verbessert werden. Die Differenz des Verlustes wird vermindert, was sich auch auf die Genauigkeit und den F1-Score auswirkt. Das Ergebnis zeigt, dass die Form des Inputs (Auflösung, Breite, Höhe) einen starken Einfluss auf das Lernverhalten hat. Dennoch passt sich das Modell weiterhin zu sehr an die Trainingsdaten an. Deshalb werden im folgenden Abschnitt weitere Optimierungsmöglichkeiten vorgestellt, implementiert und evaluiert.

7.2.2 Optimierungsmaßnahmen gegen Überanpassung

Für die weitere Optimierung sollen zunächst zwei Methoden vorgestellt werden, die sich bei neuronalen Netzwerken positiv auf deren Generalisierungsfähigkeit auswirken können:

Drop-out

Zu komplexe Modelle (viele Schichten bzw. Parameter) neigen dazu, die Trainingsdaten auswendig zu lernen und das erlernte Wissen nicht auf unbekannte Daten übertragen zu können. Dies wird auch durch die Tests der verschiedenen Architekturen in Experimentierphase 1 bestätigt. Bei der Drop-out-Technik werden während des Trainings Neuronen mit einer bestimmten Wahrscheinlichkeit deaktiviert. Dies verringert die Komplexität des Modells.

Zudem wird verhindert, dass das Netz zu sehr von den Informationen einzelner Neuronen oder kleinen Gruppen von Neuronen abhängig wird. Der Vorgang wiederholt sich je Iteration zufällig (parametrierte Wahrscheinlichkeit).[150]

Batch-Normalisierung

Input-Daten (z. B. Mel-Spektrogramme) eines neuronalen Netzwerkes werden üblicherweise normalisiert, bevor sie durch das KNN geschickt werden. Dadurch wird verhindert, dass Eingangsvariablen in einem unterschiedlichen Wertebereich liegen, was sich positiv auf die Trainingsgeschwindigkeit auswirkt. Bei der Batch-Normalisierung wird dieses Konzept nicht nur auf die Eingangsschicht, sondern auf alle Schichten angewendet. Es hat sich gezeigt, dass daraus nicht nur verkürzte Trainingszeiten, sondern auch Modelle mit erhöhter Generalisierungsfähigkeit resultieren können. Des Weiteren werden durch die Verwendung der Batch-Normalisierung Methoden wie die L2-Regularisierung überflüssig.[151] Die L2-Regularisierung ist eine weitverbreitete Methode gegen Overfitting, die hier aber nicht weiter behandelt werden soll.

PyTorch stellt mit *BatchNorm2D* und *Dropout* zwei Klassen zur einfachen Implementierung der erläuterten Methoden bereit. Dabei wird die in 6.4.1 vorgestellte CNN1-Architektur mit den folgenden Elementen erweitert:

- BatchNorm2D – Nach jedem der vier definierten Blöcke wird eine Batch-Normalisierungsschicht hinzugefügt.
- FCL – Nach dem letzten Convolution-Block wird eine FCL hinzugefügt.
- Drop-out – Nach der angefügten FCL wird ein Drop-out-Layer hinzugefügt.

Bei den vier Batch-Normalisierungsschichten werden die Default-Hyperparameter von PyTorch übernommen und für alle weiteren Tests verwendet. Beim Drop-out-Layer wird der Hyperparameter *probability* variiert. Mit den zusätzlichen Schichten werden einige Trainings mit 100 Epochen durchgeführt. Der Hyperparameter wird dabei zwischen dem Wert 0,1 und 0,7 in Zehntelschritten (steigend) angepasst. Die Ergebnisse dieser Trainings sind in den CSV-Dateien mit den Nummern T77–T83 geloggt. Aus Übersichtsgründen werden nicht alle Ergebnisse gelistet. Die besten Ergebnisse können mit einem Einstellwert von 0,5 erreicht werden. Das bedeutet, dass die Neuronen der betreffenden Schicht während des Trainings mit einer Wahrscheinlichkeit von 50 % deaktiviert werden. Während der Validierungsphase wird der Drop-out-Layer deaktiviert. Dazu muss das Modell vor jedem Validierungsvorgang mittels *model.eval()* in den Evaluierungsmodus gesetzt werden. Die finalen Trainingsergebnisse sind in Tabelle 24 gelistet. Es werden das Modell aus Experimentierphase 1 (CNN1), das Modell nach der Optimierung der Hyperparameter (CNN1_V1) sowie das Modell mit Drop-out- und Batch-Normalisierungsschichten (CNN1_V2) verglichen:

[150] Vgl. Buduma (2017), S. 36.

[151] Vgl. Buduma (2017), S. 106.

	Training			Validierung		
Bezeichnung	***L_{CE}***	***ACC/%***	***F1/%***	***L_{CE}***	***ACC/%***	***F1/%***
F_T53_CNN1_E100_BS16	0,292	91,7	73,7	1,110	70,0	52,5
F_T76_CNN1_V1_E100_BS16	**0,113**	**96,1**	**75,9**	0,622	78,9	65,8
F_T80_CNN1_V2_E100_BS16	0,373	88,9	72,3	**0,490**	**83,8**	**70,2**

Tabelle 24: Endergebnisse Trainings- und Validierungsphase, Quelle: Eigene Darstellung.

Mit der Erweiterung der Schichten und den erläuterten Hyperparameterkonfigurationen kann die Generalisierungsfähigkeit des Modells verbessert werden. Die Ergebnisse des Verlustes, der Accuracy und des F1-Scores weichen zwar weiterhin voneinander ab, jedoch können die Differenzen wesentlich vermindert werden. Des Weiteren fällt auf, dass das neue Modell (CNN1_V2) die schlechtesten Ergebnisse in der Trainingsphase erzielt. Dies kann insbesondere mit dem Drop-out-Layer begründet werden. Durch das zufällige Deaktivieren von Neuronen wird dem Auswendiglernen entgegengewirkt, was schlussendlich in einem robusteren Modell resultiert. Abschließend werden die Lernkurven in Abbildung 50 dargestellt. Abgebildet sind die Kurven des Modells nach der ersten Experimentierphase (CNN1 – blau) sowie das Modell nach Abschluss der zweiten Experimentierphase (CNN1_V2 – rot). Dabei entsprechen die durchgehenden Linien den Trainingsergebnissen und die strichlierten Linien den Validierungsergebnissen. Wie ersichtlich ist, driften die Linien des optimierten Modells nur geringfügig auseinander. Des Weiteren ist ein steigender Trend hinsichtlich des Lernerfolges erkennbar. In diesem Fall empfiehlt es sich, die Anzahl der Trainingsepochen zu erhöhen. Da für diese Arbeit eine Obergrenze von 100 Epochen definiert ist, wird dies nicht durchgeführt. Das Modell CNN1_V2 soll im weiteren Verlauf auf die unbekannten Testdaten angewendet, mit dem Modell des AutoML-Services verglichen sowie für Online-Vorhersagen (z. B. aus der Produktionsebene) veröffentlicht werden.

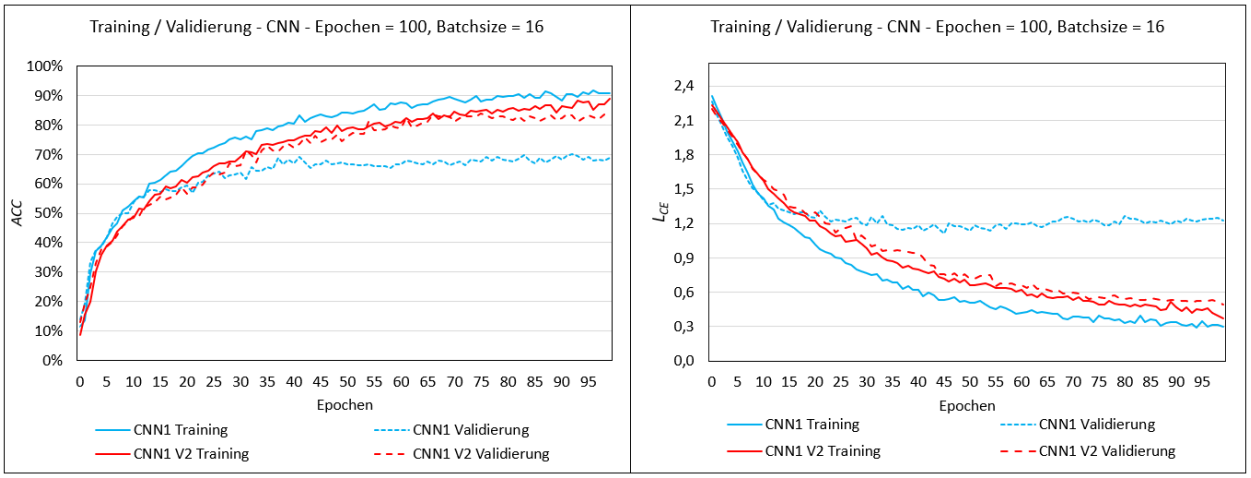


Abbildung 50: Lernkurven CNN1 und CNN1_V2, Epochen = 100, Batch-Größe = 16, Quelle: Eigene Darstellung.

7.2.3 AutoML

Zusätzlich zu den selbst erstellten Modellen wird der Datensatz mit dem AutoML-Service Vertex AI Vision trainiert. Die Vorgehensweise bei der Durchführung ist ident wie in Abschnitt 4.3.2, weshalb auf diese nicht weiter eingegangen wird. Verwendet wird der volle Datensatz mit 1984 Bildern. Dazu werden die extrahierten Mel-Spektrogramme als .png-Datei im Cloud Storage gespeichert. Als Hyperparameter werden die optimierten Einstellungen nach der zweiten Experimentierphase verwendet. Gewählt wird eine Trainingsdauer von 16 Knotenstunden. Dadurch entstehen Kosten von 55,64 $ (Stand: 09.10.2022).

7.2.4 Gesamtauswertung Teil 2

Im letzten Teil der Evaluierungsphase wird das Modell CNN1_V2 auf die bisher unbekannten Testdaten (194 Proben) angewendet. Abschließend erfolgt ein Vergleich mit der AutoML-Lösung, wobei die Vorhersagegenauigkeit je Klasse im Mittelpunkt stehen soll. Dargestellt werden die finalen Ergebnisse anhand einer Konfusionsmatrix. Diese bildet die Häufigkeit der richtigen klassenbezogenen Vorhersagen (grün markiert) ab. Des Weiteren sind die häufigsten Fehlentscheidungen je Klasse (hell- bzw. dunkelgrau) dokumentiert. Tabelle 25 enthält die Ergebnisse des CNN1_V2-Modells und Tabelle 26 die Ergebnisse der AutoML-Lösung:

		Vorhergesagte Klasse								
		arcter	brnowl	cangoo	caster1	comwax	houfin	nutman	semplo	sposan
Tatsächliche Klasse	arcter	79,3 %	6,9 %	-	-	-	10,3 %	-	3,4 %	-
	brnowl	2,5 %	92,5 %	-	2,5 %	-	-	-	-	2,5 %
	cangoo	-	13,8 %	86,2 %	-	-	-	-	-	-
	caster1	3,3 %	6,7 %	3,3 %	76,7 %	3,3 %	6,7 %	-	-	-
	comwax	-	9,5 %	4,8 %	-	81,0 %	-	-	-	4,8 %
	houfin	3,6 %	7,1 %	-	-	-	82,1 %	-	-	7,1 %
	nutman	-	28,6 %	-	14,3 %	-	-	57,1 %	-	-
	semplo	-	-	-	-	-	-	11,1 %	88,9 %	-
	sposan	-	-	-	-	-	-	-	40,0 %	80,0 %

Tabelle 25: Konfusionsmatrix CNN1_V2-Modell, Quelle: Eigene Darstellung.

Wird der Mittelwert aller richtigen Vorhersageergebnisse des CNN1_V2-Modells gebildet, resultiert eine durchschnittliche Vorhersagegenauigkeit von **80,4 %**. Das Ergebnis zeigt, dass das erlernte Wissen aus der Trainingsphase auf unbekannte Daten übertragen werden kann. Mit einer Genauigkeit von **92,5 %** wird die Klasse Brnowl (500 Proben) am häufigsten korrekt klassifiziert. Am meisten Fehlentscheidungen werden bei der Klasse Nutman (78 Proben) getroffen. Hier kann insgesamt eine Vorhersagegenauigkeit von **57,1 %** erreicht werden. Die restlichen Vorhersageergebnisse liegen in einem Bereich von **76,7 %** bis **88,9 %.**

		Vorhergesagte Klasse								
		arcter	brnowl	cangoo	caster1	comwax	houfin	nutman	semplo	sposan
Tatsächliche Klasse	arcter	85,7 %	-	-	9,5 %	4,8 %	-	-	-	-
	brnowl	-	94,0 %	2,0 %	-	-	2,0 %	-	-	2,0 %
	cangoo	-	6,1 %	90,9 %	-	-	3,0 %	-	-	-
	caster1	-	5,3 %	-	89,5 %	-	5,3 %	-	-	-
	comwax	-	5,0 %	-	-	85,0 %	5,0 %	-	5,0 %	-
	houfin	-	3,0 %	-	-	3,0 %	93,9 %	-	-	-
	nutman	11,1 %	-	-	-	11,1 %	11,1 %	55,6 %	-	11,1 %
	semplo	11,1 %	-	-	-		11,1 %	-	77,8 %	-
	sposan	7,7 %	-	-	-	7,7 %	7,7 %	-	-	76,9 %

Tabelle 26: Konfusionsmatrix AutoML-Modell, Quelle: Eigene Darstellung.

Bei Bildung des Mittelwerts aller richtigen Vorhersageergebnisse des AutoML-Modells resultiert eine durchschnittliche Genauigkeit von **83,3 %**. Gleich wie beim CNN1_V2-Modell trifft die AutoML-Lösung bei der Klasse Brnowl mit einer Genauigkeit von **94 %** am häufigsten die korrekte und bei der Klasse Nutman mit einer Genauigkeit von **55,6 %** am häufigsten die falsche Entscheidung. Alle anderen Ergebnisse liegen in einem Bereich von **76,9 %** bis **93,9 %**.

Die Ergebnisse zeigen, dass die mit AutoML erstellte Lösung geringfügig bessere Entscheidungen trifft. Beide Modelle klassifizieren Audiosignale der Klasse Brnowl am häufigsten korrekt. Mit 500 Proben ist dies jene Klasse, von der die meisten Proben im Datensatz enthalten sind. Dies spiegelt die Relevanz von genügend Daten im Zuge eines ML-Projektes wider. Die Klasse Nutman ist mit 78 Proben nur selten vertreten. Dementsprechend sind die schlechteren Vorhersageergebnisse wenig überraschend. Dennoch zeigt sich anhand beider Lösungen, dass auch bei Klassen mit wenigen Proben gute Ergebnisse erreicht werden können. Bei der Klasse Semplo (77 Proben) wird ein Genauigkeitswert von **88,9 %** (CNN1_V2) und **77,8 %** (AutoML) erzielt. Dies ist ein Indikator für eine höhere Datenqualität der Klasse Semplo im Vergleich zur Klasse Nutman. Gute und standardisierte Aufnahmebedingungen sind ein weiterer relevanter Erfolgsfaktor für die erfolgreiche Entwicklung eines Klassifizierers im Bereich von Audiosignalen. Werden falsche Vorhersagen getroffen, so entsprechen die Falschergebnisse meistens der Klasse Brnowl oder Houfin (322 Proben). Dies steht ebenfalls mit der höheren Anzahl an Proben in Verbindung.

Generell kann aufgrund der Komplexität des Datensatzes bei beiden Lösungen (CNN1_V2 und AutoML) von einem guten Ergebnis gesprochen werden. Jedoch verlangen die meisten Lösungen für den professionellen Einsatz in Unternehmen eine höhere Zuverlässigkeit, denn häufige Fehlentscheidungen können auch einen hohen Kostenfaktor darstellen. Im folgenden Abschnitt soll deshalb diskutiert werden, welche weiteren Möglichkeiten zur Verbesserung der Modell-Performance durchgeführt werden könnten.

7.2.5 Weitere Optimierungsmöglichkeiten

Die Diskussion der einzelnen Optimierungsmöglichkeiten soll sich auf die Phasen des Crisp-DM-Modells beziehen:

Datenerhebung

Die effektivste Maßnahme, um Overfitting zu vermeiden und eine bessere Modell-Performance zu erhalten, ist die Erhöhung der Datenmenge. Wie aus den Tests ersichtlich, können bereits mit etwa 200 Proben gute Ergebnisse (Genauigkeit größer als 90 %) erzielt werden. Diese Aussage bezieht sich auf den vorliegenden Datensatz. Die tatsächlich benötigte Datenmenge eines ML-Projektes variiert je nach Aufgabe. Deshalb ist zu empfehlen, dass bereits im Vorfeld eine wohlüberlegte Entscheidung hinsichtlich der Anzahl der benötigten Daten getroffen wird. Außerdem soll bei der Datenerhebung darauf geachtet werden, dass die Aufnahmequalität möglichst hoch ist. Dies gilt für Trainings-, Validierungs- und Testdaten gleichermaßen wie für die Daten, die schlussendlich klassifiziert werden sollten. Im industriellen Bereich betrifft diese Phase insbesondere die Automatisierungstechniker*innen, die für die Auslegung der Sensorik zuständig sind und somit auch in das ML-Projekt einzubeziehen sind.

Datenvorbereitung

Wie im Verlauf der Hyperparameteroptimierungen ersichtlich wird, hat die Preprocessing-Phase einen wesentlichen Einfluss auf die Modell-Perfomance. Im Zuge dieser Arbeit werden die Audiosignale auf eine Gesamtlänge von 10 s vereinheitlicht. Längere Abschnitte werden verworfen und kürzere Signale werden mit Nullen aufgefüllt. Der Inhalt dieser ersten 10 s wird nicht weiter beachtet und als gegeben hingenommen. Eine Möglichkeit zur besseren Nutzung der vorhandenen Daten wäre die Entwicklung eines zweistufigen Prozesses. In erster Linie könnten die Audiosignale in kürzere Segmente (z. B. drei Sekunden) zerlegt und mit gleichem Label abgespeichert werden. Aus einem Audiosignal mit 60 s resultieren somit 20 Proben. Ferner wäre eine Vorklassifizierung denkbar, die die Signale in ‚Vogelgeräusch vorhanden' und ‚kein Vogelgeräusch vorhanden' trennt, wobei im Anschluss nur die Daten mit Vogelgeräuschen für das Training verwendet werden. Zusätzlich bietet die in 3.2.3 vorgestellte Datenvervielfältigungsmethode Raum für weitere Optimierungen. Diese Technik kann sowohl auf die Wellenform als auch auf die Mel-Spektrogramm-Form angewendet werden. Abbildung 51 zeigt ein Beispiel einer solchen bewussten Datenmanipulation. Hierbei wird jeweils ein zufälliger Bereich der Zeit- und der Frequenzachse (rot markiert) ausgeblendet. Mit dieser Technik wird eine Datenvielfalt garantiert.

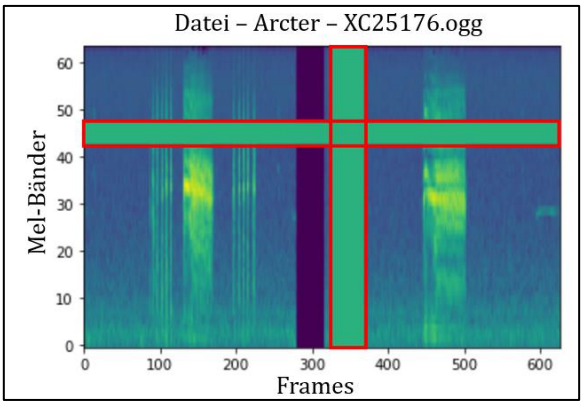


Abbildung 51: Anwendung einer Zeit- und Frequenzmaskierung (Mel-Spektrogramm), Quelle: Eigene Darstellung.

Modellierung

Das Wissen der getesteten vortrainierten Modelle kann im Verlauf der Experimentierphase nicht erfolgreich auf den verwendeten Datensatz übertragen werden. Hier wäre es denkbar, dass sich andere vortrainierte Modellarchitekturen besser eignen. Eine zusätzliche Modellvariation ist zu empfehlen.

Training

Um die einzelnen Einflüsse von Hyperparametervariationen zu demonstrieren und Vergleiche zu ermöglichen, werden die Modelle im Verlauf dieser Arbeit immer von Grund auf trainiert. Modelle können auch gespeichert sowie für das weitere Training geladen werden. Dies erspart Zeit und ermöglicht schnellere Trainingserfolge. Des Weiteren ist es ratsam, das Training bei Overfitting frühzeitig zu beenden.

7.3 Diskussion

Zum Abschluss dieses Kapitels sollen die zentralen Erkenntnisse der Evaluierungsphase zusammengefasst und diskutiert werden. Während der ersten Experimentierphase wurden 33 verschiedene Modellarchitekturen trainiert und validiert. Dabei stellte sich heraus, dass sich CNN-Architekturen am besten für die Aufgabenstellung eignen. Das Modell mit der besten Performance wurde in der zweiten Phase durch Variation der Hyperparameter sowie durch Regularisierungsmethoden optimiert. Insgesamt wurde für den Prozess eine Vielzahl an Experimenten durchgeführt, wovon 84 Trainingsdurchläufe geloggt wurden. Viele Trainingsdurchläufe mussten aufgrund von Fehlkonfigurationen oder wegen schlechter Lernerfolge vorab beendet werden. Es konnte ein Modell entwickelt werden, das eine durchschnittliche Vorhersagegenauigkeit von 80,4 % aufweist. Ferner wurde der AutoML-Service von Google für die Umsetzung der Aufgabenstellung verwendet. Mit diesem wurde eine durchschnittliche Vorhersagegenauigkeit von 83,3 % erreicht. Abschließend wurde ein Ausblick auf weitere Optimierungsmöglichkeiten gegeben.

Die Aufgabenstellung demonstriert die Schwierigkeiten, mit denen ML-Expert*innen umgehen müssen. In den Phasen Preprocessing, Modellierung und Training müssen viele Hyperparameter konfiguriert werden. Zwar kann die Literatur als unterstützendes Element bei der Parametrierung betrachtet werden, jedoch müssen die meisten Hyperparametereinstellungen durch ein Trial-and-Error-Verfahren selbstständig ermittelt werden. Das Finden einer optimalen Hyperparameterkonfiguration sowie einer perfekten Modellarchitektur ist dabei aufgrund der umfangreichen Anzahl an Kombinationsmöglichkeiten, auch in einer automatisierten Umgebung, nicht möglich. Im Fokus muss daher die Annäherung an das perfekte Modell liegen, wobei ein Kompromiss zwischen investierter Zeit und Modell-Performance geschlossen werden muss. Hier können sich einerseits die Erfahrungen der ML-Expert*innen und andererseits die zur Verfügung stehenden Rechenkapazitäten positiv auf das Endergebnis auswirken. Zweiteres verkürzt die Trainingszeiten, wodurch in gleicher Zeit mehr unterschiedliche Einstellungen und Architekturen getestet werden können. Das Ergebnis der zweiten Experimentierphase verdeutlicht, wie relevant es ist, viele unterschiedliche Konfigurationen zu testen. Durch die Hyperparametervariationen kann die Genauigkeit der Validierungsphase um 13,8 % gegenüber dem Endergebnis der ersten Phase erhöht werden.

Mit dem AutoML-Service kann zwar ein geringfügig besseres Ergebnis erreicht werden, dennoch weist diese Lösung einige Schwachstellen auf. Die drei großen Cloud-Anbieter Amazon, Google und Microsoft bieten Services im Bereich der Sprachanalyse (z. B. Sprache zu Text) an, jedoch gibt es keinen Service für die Klassifizierung von Audiosignalen. Mit Vertex AI Vision wird ein Service verwendet, der eigentlich für die Klassifizierung von Bilddateien gedacht ist. Im Falle von Audiosignalen bedeutet dies, dass diese erst durch eine aufwendige Vorverarbeitung in Spektrogramme oder Mel-Spektrogramme transformiert werden müssen. Zwar gibt es Programme, die diese Transformationen ohne Programmierkenntnisse ermöglichen, jedoch ist der dafür benötigte Zeitaufwand für eine große Menge von Daten nicht vertretbar. Das bedeutet, dass zumindest für diesen Aufgabenteil Softwareentwicklungskenntnisse notwendig sind, was den Grundgedanken von AutoML relativiert. Des Weiteren sind die Trainingskosten des Services hoch. Wenige Trainingsdurchläufe sind dabei auch für kleine und mittlere Unternehmen vertretbar, doch eine Modelloptimierung, wie sie im Zuge der Experimentierphase durchgeführt wird, kann nur mit einem hohen Kostenaufwand bewältigt werden. Weiters ist zu erwähnen, dass der AutoML-Service im Zuge der Durchführung dieser Arbeit bereits die optimierten Mel-Spektrogramme als Input erhielt.

Mit einer durchschnittlichen Vorhersagegenauigkeit von 80,4 % konnte zwar keine Modell-Performance erreicht werden, die sich für den professionellen Einsatz in Unternehmungen eignet, dennoch kann mit den abschließend erläuterten Methoden ein Ausblick auf eine Vielzahl an weiteren Optimierungsmöglichkeiten gegeben werden. Dadurch kann die realistische Annahme getroffen werden, dass die zuverlässige Klassifizierung von komplexen und wenigen Audiodateien mit den vorgestellten DL-Methoden möglich ist.

8 INTEGRATION IN DAS UNTERNEHMENSSYSTEM

Nachdem in Abschnitt 6 und 7 gezeigt wurde, wie ML-Modelle für die Audiosignalklassifizierung mittels einer experimentellen Umgebung (Jupyter-Notebooks bzw. PyTorch) und Programmiersprache (Python) implementiert, trainiert und optimiert werden können, erfolgt in diesem Abschnitt eine Auseinandersetzung mit Cloud-Services, die das professionelle Training und die Veröffentlichung von KI-Modellen für Online-Vorhersagen ermöglichen. Im Fokus stehen dabei die beiden Services Vertex AI Training und Vertex AI Prediction. Des Weiteren wird eine Übersicht über die für diese Arbeit angefallenen Kosten der Cloud-Services gegeben.

8.1 Automatisiertes Training mit Vertex AI

Das Trainieren und Optimieren von Modellen mit Notebooks ist eine übliche Vorgehensweise bei der Durchführung von ML-Projekten. Jedoch ist diese nur bedingt für größere Projekte geeignet. Wie auch aus der Evaluierungsphase dieser Arbeit ersichtlich wird, ist das Trainieren und Optimieren der Modelle eine zeit- und kostenintensive Aufgabenstellung. Deshalb werden Notebooks bei der professionellen Projektdurchführung überwiegend zum Erkennen von Trends und geeigneten Modellen verwendet. Die entsprechenden Modelle werden darauffolgend in automatisierten Trainingsumgebungen an die Problemstellung angepasst. Dadurch kann eine bestmögliche Modell-Performance mit geringem Zeitaufwand erreicht werden. Zu diesem Zweck bietet die Google-Cloud-Plattform den Service Vertex AI Training an. Dieser ermöglicht nicht nur das automatisierte Training von AutoML-Lösungen, sondern auch von selbst entwickelten Modellen. Für Zweiteres müssen einige Schritte durchgeführt werden, die je nach verwendetem ML-Framework geringfügig voneinander abweichen können. Die im Folgenden erläuterte Vorgehensweise soll sich auf das PyTorch-Framework beziehen.

Für das automatisierte Training von PyTorch-Modellen auf Vertex AI stellt Google eine Anleitung zur Verfügung, die an die eigene Aufgabenstellung angepasst werden muss. Zum Erstellen eines benutzerdefinierten Trainingsjobs müssen die folgenden drei Schritte durchgeführt werden:[152]

- Erstellen einer Python-Quelldistribution, die den Trainingscode in Form eines Python-Pakets enthält,
- das erstellte Python-Paket im Cloud Storage speichern,
- das Python-Paket mittels grafischer Oberfläche des Vertex-AI-Trainingsservices vom Cloud Storage laden und den Trainingsjob konfigurieren.

Erstellen einer Python-Quelldistribution

Zunächst muss der benutzerdefinierte Trainingscode als Python-Quelldistribution verpackt werden. Dazu wird die Python-Bibliothek Setuptools verwendet. Google gibt für das Paket die in Abbildung 52 (links) dargestellte Projektstruktur vor, die aus den folgenden vier Python-Dateien besteht:[153]

[152] Vgl. Google 11 (2021), Online-Quelle [17.11.2022].

[153] Vgl. Google 12 (2022), Online-Quelle [17.11.2022].

- *setup.py* – Gibt für Setuptools vor, wie die Quelldistribution erstellt werden sollte.
- *__init__.py* – Eine leere Python-Datei. Jedes Unterverzeichnis, das eine __init__.py-Datei enthält, wird beim Erstellen des Paketes miteinbezogen.
- *task.py* – Dient als Einstiegspunkt für den benutzerdefinierten Trainingscode und bildet die Schnittstelle bzgl. Hyperparameterübergabe zwischen Vertex AI und dem definierten Trainingscode.
- *training.py* – Enthält den ML-Workflow. Dieser gestaltet sich nach der folgenden Struktur: Audiodateien aus Cloud Storage laden → Preprocessing → Datensätze erstellen → Modellarchitektur laden → Training → Trainiertes Modell in Cloud Storage speichern. Die Implementierung ist ident mit jener in Abschnitt 6.

Abbildung 52 (Bildmitte) zeigt den Inhalt der *setup.py*-Datei. Diese stellt unter anderem sicher, dass die Google-Trainingsmaschine die für den ML-Trainingsworkflow notwendigen Programmbibliotheken installiert (roter Pfeil). In Abbildung 52 (rechts) wird ein Ausschnitt des Source-Codes der *task.py*-Datei dargestellt. In dieser werden die Argumente (Hyperparameter) definiert, die an Vertex AI übergeben werden. Diese können jede Phase des ML-Workflows betreffen (z. B. Preprocessing). Wie aus der Abbildung ersichtlich, werden hier die Batch-Größe sowie die Anzahl der Trainingsepochen übergeben. Des Weiteren wird die *run()*-Funktion der *training.py*-Datei aufgerufen. Der Einstiegspunkt für den Trainingsservice von Google ist die *main()*-Funktion.

Die Quelldistribution wird mit dem Befehl *python setup.py sdist --formats=gztar* erstellt und anschließend im Cloud Storage gespeichert.

Abbildung 52: Dateistruktur Trainingspipeline für das automatisierte Training (links), Inhalt der *setup.py*-Datei (Bildmitte) und Ausschnitt aus der Datei *task.py* (rechts), Quelle: Eigene Darstellung.

Konfiguration des Trainingsjobs auf Vertex AI

Im Anschluss kann eine Trainingspipeline erstellt werden. Abbildung 53 zeigt die relevantesten Einstellungen, die dabei vorzunehmen sind. Auf diese fünf Punkte wird im Folgenden eingegangen:

1. Der Vertex AI Trainingsservice unterstützt das ML-Framework PyTorch.
2. Das erstellte Python-Paket wird vom Cloud Storage geladen. Des Weiteren wird der Einstiegspunkt für den Trainingsjob definiert (trainer.task) sowie ein Speicherort für das fertig trainierte Modell ausgewählt.
3. Es können die in *task.py* definierten Hyperparameter übergeben werden.
4. Optional kann eine Hyperparametervariation konfiguriert werden. Vertex AI testet in diesem Fall die eingestellten Konfigurationen automatisiert. Bei diesem Beispiel werden die Batch-Größen 16, 32, 64, 128 und 256 getestet.
5. Es wird die zu verwendende Trainingsmaschine konfiguriert. Diese Einheit ist skalierbar und bestimmt die anfallenden Kosten für den Trainingsjob.

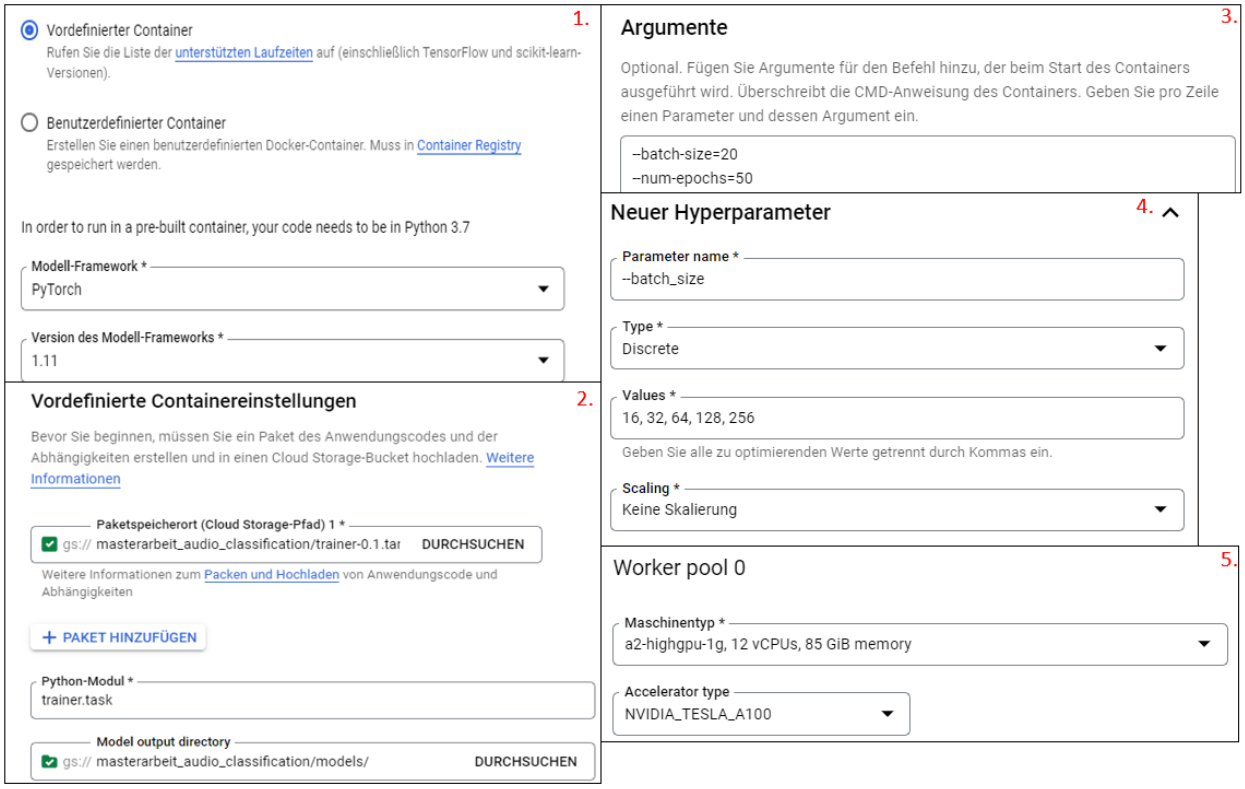


Abbildung 53: Konfiguration eines Trainingsjobs Vertex AI, Quelle: Eigene Darstellung.

Die erstellte Trainingspipeline kann im Anschluss beliebig oft ausgeführt werden. Es können verschiedene Hyperparametervariationen automatisiert durchgeführt werden. Google versucht dabei, die Modell-Performance zu erhöhen. Dazu wird die konfigurierte Metrik (siehe Abbildung 54, wird ebenfalls als Argument übergeben) optimiert. Das beste Modell wird abschließend im Cloud Storage gespeichert.

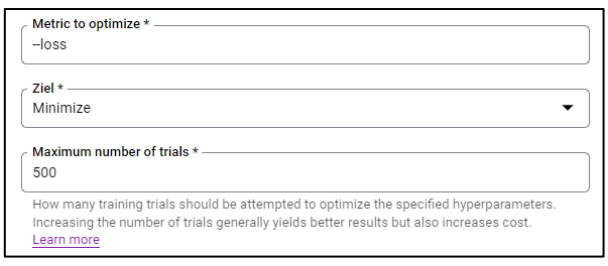


Abbildung 54: Konfiguration der zu optimierenden Metrik, Quelle: Eigene Darstellung.

8.2 Bereitstellung eines Modells auf Vertex AI

Um ein entwickeltes Modell für die Beantwortung von Vorhersageanfragen nutzen zu können, muss dieses bereitgestellt werden. Die Bereitstellung kann sowohl auf einem lokalen Endgerät (PC, Handy, etc.) als auch auf einer Cloud-Plattform erfolgen. Zweiteres eignet sich aufgrund der hohen Rechenkapazitäten besser für die Integration eines Modells in eine Umgebung mit hohen Anfrageaufkommen (z. B. Produktion). Des Weiteren sind die Cloud-Anbieter (z. B. Google) darauf konzentriert, die Latenzzeiten gering zu halten, wodurch Vorhersagen schnell durchgeführt werden können. Bei zeitkritischen Prozessen ist dies von Vorteil.

Bei Vertex AI können Modelle mit dem Prediction-Service (Endpunkte) veröffentlicht werden. Jedoch muss das entwickelte Modell zunächst bei Vertex AI registriert werden. Im Folgenden sollen alle Schritte, die zur Veröffentlichung eines PyTorch-Modells auf Vertex AI notwendig sind, dargelegt und implementiert werden.

Der Online-Prediction-Service von Google basiert auf Docker-Container-Images, die unter anderem einen HTTP-Server bereitstellen und ausführen. Dabei stellt Google für bestimmte ML-Frameworks (TensorFlow, XGBoost etc.) vorgefertigte Container zur Verfügung. Das ML-Framework PyTorch wird hierbei nicht unterstützt. Jedoch gibt es die Möglichkeit zur Definition benutzerdefinierter Container. An diese werden die folgenden Anforderungen gestellt:[154]

- Bei Ausführung des Containers durch Vertex AI muss zusätzlich ein HTTP-Server ausgeführt werden. Dieser kann auf jede beliebige Art und Weise implementiert sein. Google empfiehlt für die Implementierung beispielsweise das Python-Web-Framework Flask.
- Der HTTP-Server muss eine Systemdiagnoseroute enthalten. Vertex AI prüft mittels HTTP-GET-Anfrage, ob der Server bereit für Vorhersageanfragen ist. Eine Systemdiagnoseanfrage muss mit dem Statuscode 200 quittiert werden.
- Der HTTP-Server muss eine Vorhersageroute enthalten. Diese muss Vorhersageanfragen (Abbildung 55 links) im JSON-Format akzeptieren und Vorhersageantworten (Abbildung 55 rechts) im JSON-Format bereitstellen. Das JSON-Objekt *instances* ist ein dynamisches Array (ein oder mehrere Werte) beliebigen Typs. Dabei entspricht jeder Wert einer Instanz (z. B. Speicherort für Audio-File). Das JSON-Objekt *parameters* ist optional und wird nicht weiter betrachtet. Das JSON-Objekt *predictions* enthält die Vorhersageantwort (z. B. Klassenname).
- Das Docker-Image muss für die Verwendung durch Vertex AI in Artifact Registry oder Container Registry gespeichert werden.

Abbildung 55: Format der Vorhersageanfrage sowie der Vorhersageantwort eines benutzerdefinierten Containers, Quelle: Google 13 (2022), Online-Quelle [17.11.2022].

[154] Vgl. Google 13 (2022), Online-Quelle [17.11.2022].

Im weiteren Verlauf wird auf die vier wesentlichen Schritte, die zur Veröffentlichung eines PyTorch-Modells auf Google Vertex AI notwendig sind, eingegangen:

- Erstellung eines Docker-Images,
- Registrierung des Docker-Images in Container Registry,
- Import des Modells (Containers) in Model Registry sowie
- Bereitstellung des Modells auf einem Vertex-AI-Endpunkt.

Als Implementierungsgrundlage werden eine Anleitung von Google[155] und ein Online-Tutorial verwendet.[156] Diese werden an die Aufgabenstellung dieser Arbeit angepasst.

Erstellung eines Docker-Images für Online-Vorhersagen auf Vertex AI

Der Aufbau der Anwendung gliedert sich wie in Abbildung 56 (links) dargestellt. Die vier enthaltenen Elemente können wie folgt beschrieben werden:

- *Dockerfile* – Ein Dockerfile ist eine Textdatei, die Kommandozeilen-Anweisungen zum Erstellen von Images enthält.[157] Anhand der Anweisungen innerhalb dieser Datei wird das Docker-Image erstellt und der HTTP-Server (Port 8081) gestartet. Die Datei ist ident mit jener im Online-Tutorial und wird hier nicht weiter angeführt.
- *requirements.txt* – Enthält Informationen zu benötigten Programmbibliotheken, die auf der Google-Maschine installiert werden müssen (Abbildung 56 rechts).
- *CNN1_V2.pth* – Ist das trainierte PyTorch-Modell mit gespeicherten Parametern (z. B. Gewichtungen, Bias).
- *main.py* – Enthält die Flask-Webanwendung sowie die Diagnose- und Vorhersageroute für Online-Vorhersagen.

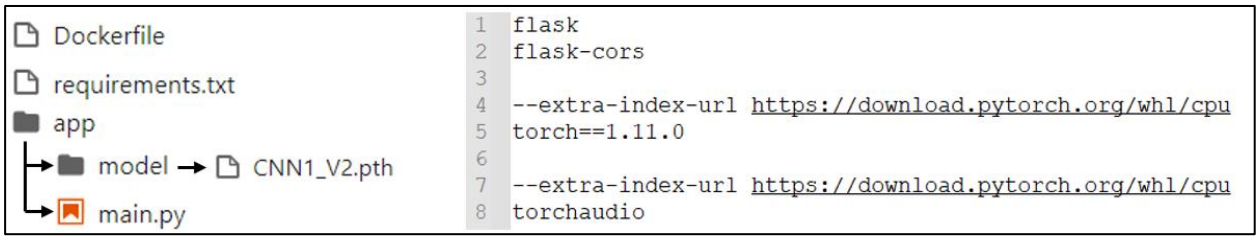


Abbildung 56: Aufbau Applikation für Online-Vorhersagen, Quelle: Eigene Darstellung.

Abbildung 57 (links) fasst die zentralen Elemente der *main.py*-Datei zusammen. Der Programmablauf ist dabei aufgrund der Übersichtlichkeit auf das Wesentlichste reduziert. Nach dem Starten der Anwendung (Codezeile 23 und 24) wird zunächst die Modellarchitektur instanziiert und mit den trainierten Parametern in die CPU/GPU geladen (Codezeile 9–16). In diesem Fall wird das optimierte CNN1_V2-Modell der Evaluierungsphase verwendet.

[155] Vgl. Google 13 (2022), Online-Quelle [17.11.2022].

[156] Vgl. Chiusano (2022), Online-Quelle [17.11.2022].

[157] Vgl. Docker (o.J.), Online-Quelle [17.11.2022].

Die Vorhersageroute (Abbildung 57 rechts) nimmt HTTP-Anfragen über den Google-Endpunkt entgegen. Im Folgenden werden die JSON-Daten eingelesen und dekodiert (Codezeile 5–15). Im Verlauf dieses Beispiels werden nicht direkt die Audiodateien, sondern die Speicherorte der jeweiligen Datei übergeben. Im Anschluss wird der folgende ML-Workflow durchlaufen:

Lade Audiodatei von Speicherort → Preprocessing → Vorhersage durchführen → Vorhersage auswerten → Antwort übermitteln.

Die Implementierung der Preprocessing-Phase ist ident mit jener in Abschnitt 6 und wird hier nicht dargestellt.

```python
#import flask relevant classes
from flask import Flask, request, Response, jsonify

#import other classes (preprocessing, model...)

#instance Flask application object
app = Flask(__name__)

#load model architecture
myModel = CNN1_N(config.num_classes_reduced)
#load parameters
PATH = './model/CNN1_V2.pth'
myModel.load_state_dict(torch.load(PATH))
#put it on device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
myModel = myModel.to(device)

#diagnose route (placeholder)

#prediction route (placeholder)

#run flask application
if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=8081)
```

```python
#prediction route
@app.route("/predict", methods=["POST"])
def predict():

    req_json = request.get_json()

    #get instances
    json_instances = req_json["instances"]

    #get instance values
    for i in json_instances:
        json_audio = i

    #get file location (e.g. Cloud Storage)
    json_audioFile = json_audio["audio"]

    #workflow for preprocessing (placeholder)

    # Get predictions
    outputs = myModel(inputs)
    _, prediction = torch.max(outputs,1)

    #evaluate prediction
    if prediction == 0:
        response = "arcter"
            .
            .
            .
    #parse and response to request
    return jsonify({
        "predictions": [response]
    })
```

Abbildung 57: Programmausschnitt des Programmablaufes der *main.py*-Datei (links) und Programmausschnitt der Prediction-Route der *main.py*-Datei (rechts), Quelle: Eigene Darstellung.

Registrierung des Docker-Images in Container Registry

Im Anschluss wird das Docker-Image auf Basis des Dockerfiles und mittels Konsolenbefehlen (siehe Abbildung 58 oben) erstellt sowie im Container-Registry-Service registriert. Dabei ist folgendes Format zu beachten: [grc.io / Google-Cloud-Projekt-ID / Image-Name].

```
docker build -t gcr.io/graphic-mission-359510/ma-docker-api3 .
docker push gcr.io/graphic-mission-359510/ma-docker-api3
```

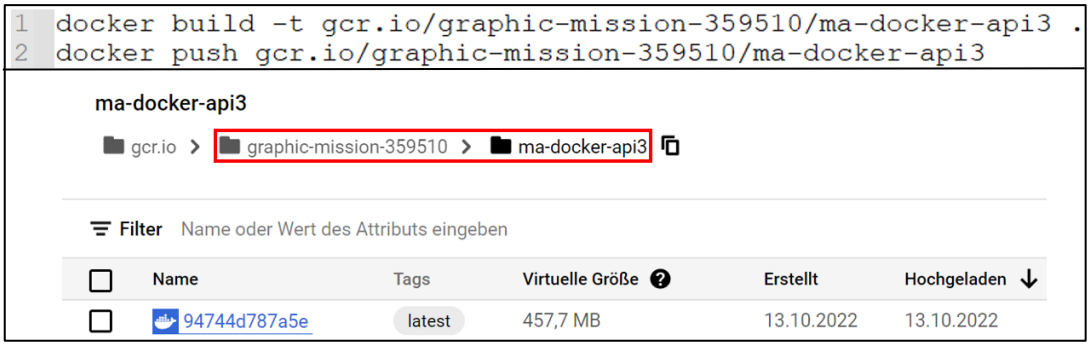


Abbildung 58: Konsolenbefehle (Linux) zum Erstellen eines Docker-Images sowie zum Registrieren des Images im Container-Registry-Service (oben) und Ausschnitt aus Container-Registry-Service (unten), Quelle: Eigene Darstellung.

Import des Containers in Model Registry

Damit ein Modell auf einem Endpunkt bereitgestellt werden kann, muss es zunächst in Model Registry registriert werden. In Abbildung 59 sind die zentralen Schritte zusammengefasst:

1. Laden des Docker-Images aus Container Registry.
2. Angabe der Vorhersageroute, der Systemdiagnose-Route sowie des Ports.

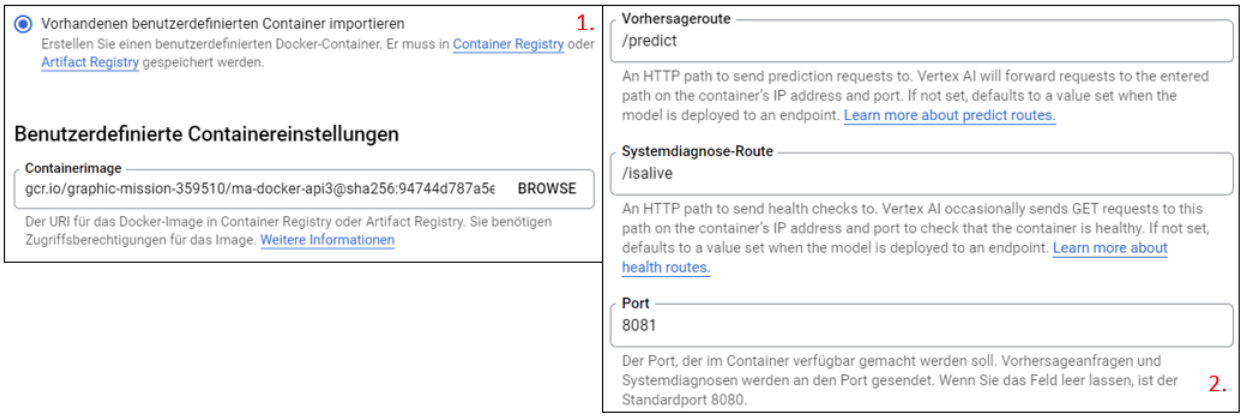


Abbildung 59: Konfigurationen Model Registry, Quelle: Eigene Darstellung.

Bereitstellung des Modells als Endpunkt

Im Anschluss kann das Modell unter Vertex AI ‚Endpunkte' für Vorhersageanfragen bereitgestellt werden. Dazu können, je nach voraussichtlichem Anfrageaufkommen, eine oder mehrere CPUs bzw. GPUs mit skalierbaren Rechenkapazitäten konfiguriert werden. Für Anfragen werden von Google verschiedene Programmierschnittstellen zur Verfügung gestellt (z. B. REST oder Python-API). Für den Test in dieser Arbeit wurde die Python-Schnittstelle verwendet. Abbildung 60 stellt eine Anfrage mittels Python an einen Vertex-AI-Endpunkt dar. Neben der Projektnummer und der Nummer des bereitgestellten Endpunktes (Modells) wird ein Speicherort einer Audiodatei übergeben (*instances*). Nach dem Senden einer HTTP-Anfrage wird das Signal aus dem Speicherort geladen, vorverarbeitet und klassifiziert. Wie ersichtlich ist, wird die Probe in diesem Beispiel korrekt erkannt. Eine Audiodatei kann auch direkt übergeben werden.

Als Unterstützung zur Umsetzung stellt Google ein Codebeispiel zur Verfügung, das unter anderem die Methode *predict_custom_trained_model_sample()* implementiert.[158]

Abbildung 60: Beispiel-Anfrage, ausgeführt in einem Jupyter-Notebook, Quelle: Eigene Darstellung.

[158] Vgl. Google 14 (o.J.), Online-Quelle [17.11.2022].

Des Weiteren stellt Google einen Service zur Perfomance-Messung der veröffentlichten Modelle bereit. Die Performance hängt wesentlich von der konfigurierten CPU bzw. GPU ab. Mit der gewählten CPU (16 vCPUs, 60 GB RAM) kann eine Latenzzeit von 200 ms erreicht werden, was fünf Anfragen pro Sekunde entspricht.

8.3 Kostenübersicht

Zur Erfassung der Kosten bietet Google einen Abrechnungsservice. Generell gestaltet es sich schwierig, die anfallenden Kosten vorab abzuschätzen. Es ist nicht immer eindeutig, welche Services tatsächlich im Hintergrund mitgenutzt werden und ob für diese Kosten anfallen oder nicht. Zudem hängt die Höhe der Kosten meist von den konfigurierten Ressourcen ab. Als Referenz bietet Abbildung 61 eine Übersicht der für diese Arbeit angefallenen Kosten. Diese beziehen sich auf die in Abschnitt 5.4 erläuterte Hard- und Software. Dabei wird über einen Zeitraum von etwa drei Monaten und eine VM-Laufzeit von etwa 300 Stunden ein Betrag von **598,88 €** erreicht. Diesbezüglich ist anzumerken, dass der Vision-Service für die Klassifizierung der beiden Datensätze (Use-Case-Vergleich und Experimentierdatensatz) mit 307,74 € mehr als die Hälfte der Gesamtkosten ausmacht.

Dienst	Kosten
Compute Engine	252,55 €
Notebooks	23,09 €
Cloud Storage	3,27 €
Artifact Registry	0,00 €
Cloud Logging	0,00 €
Cloud Vision API	307,74 €
Google Click to Deploy Deep Learning VM	0,00 €
Vertex AI	12,33 €
VM Manager	0,00 €

Abbildung 61: Kostenübersicht, Quelle: Eigene Darstellung.

8.4 Diskussion

In diesem Kapitel wurden zwei verschiedene Möglichkeiten zur Integration von selbst entwickelten ML-Lösungen in die Google-Cloud-Umgebung demonstriert. Diese ermöglichen den professionellen Umgang mit ML-Modellen (Training) und deren professionelle Nutzung (Bereitstellung). Die Implementierung basiert dabei auf vorgefertigten Docker-Container-Images (Trainingsservice) und benutzerdefinierten Docker-Container-Images (Prediction-Service). Wie ersichtlich wurde, gestaltet sich die Umsetzung nicht einfach. Dennoch können Unternehmen davon profitieren, denn der ML-Workflow im Bereich der Audiosignalklassifizierung ist meist projektübergreifend ident. Das bedeutet, dass beispielsweise die erstellten Trainingspipelines auch für neue Projekte wiederverwendet werden können. Des Weiteren bietet Google mit der automatisierten Hyperparametervariation eine kostensparende Lösung zum Trainieren und Optimieren von Modellen.

Zwar ist das Mieten einer Trainingsmaschine im Vergleich zu einer Notebook-Instanz kostenintensiver, jedoch bietet der automatisierte Trainingsservice zwei wesentliche Vorteile. Einerseits können dadurch personelle Ressourcen eingespart werden und andererseits können Trainingspipelines einfach über die grafische Oberfläche von Vertex AI konfiguriert und gestartet werden. Dies ermöglicht es auch Nutzer*innen, die keine Softwareentwicklungskenntnisse besitzen, Modelle zu trainieren. Beides kann sich positiv auf den Unternehmenserfolg auswirken. Ferner stellt Google eine mächtige Recheninfrastruktur für die Veröffentlichung von Modellen für Vorhersageanfragen zur Verfügung. Dies eignet sich im Speziellen für Umgebungen, die eine niedrige Reaktionszeit fordern. Durch die Skalierbarkeit der Rechenressourcen können diese je nach Anforderung einfach angepasst werden. In Abhängigkeit von der gewählten Konfiguration fallen Kosten in unterschiedlicher Höhe an. Wie aus dem Abschnitt 8.3 ersichtlich, sind die Beträge für die von Google gemanagten Software-Services (Google Vision) wesentlich höher als für die reine Nutzung von Hardwareressourcen. Um den maximalen Erfolg bei der Umsetzung eines KI-Projektes erreichen zu können, ist dies zu berücksichtigen.

9 FAZIT UND AUSBLICK

Das Ziel dieser Arbeit war es, eine cloudbasierte KI-Lösung zur Klassifizierung von Audiosignalen zu entwickeln. Dazu wurde zunächst eine Literaturrecherche in den Bereichen KI, Cloud-Computing und KI-basierte Audiosignalverarbeitung durchgeführt. Als Grundlage für die Konzeptionierung und Implementierung der Lösung diente das Crisp-DM-Phasenmodell. In diesem Vorgehensmodell werden die einzelnen Phasen eines ML-Projektes von der Datenerhebung bis hin zur Veröffentlichung eines Modells ganzheitlich betrachtet und es wird gezeigt, dass viele verschiedene Rollen an einem KI-Projekt beteiligt sind.

Um eine Entwicklung von einer zentralen Stelle aus zu ermöglichen, wurde deshalb eine Cloud-Umgebung als Entwicklungsumgebung gewählt. Diese stellt Rechen- und Speicherkapazitäten sowie Software-Services zur Verfügung, die bei der Umsetzung unterstützen. Aufbauend auf einem umfangreichen Vergleich der drei größten Anbieter Amazon, Google und Microsoft wurde die KI-Plattform Google Vertex AI als Basis für die weitere Implementierung gewählt.

Ferner wurden im Zuge der Literaturrecherche die State-of-the-Art-Methoden im Bereich der Audiosignalklassifizierung ermittelt und erläutert. Es hat sich gezeigt, dass es üblich ist, Audiosignale zunächst mit Methoden aus dem Bereich der Fourier-Analyse in Mel-Spektrogramme zu transformieren und anschließend verschiedene DL-Algorithmen für die Analyse dieser Mel-Spektrogramme zu verwenden. Konkret handelt es sich dabei um KNN mit CNN- oder RNN-Architektur.

Als Implementierungsgrundlage wurde dazu ein frei verfügbarer Datensatz der ML-Plattform Kaggle herangezogen. Dieser enthält von Hobbybiologen aufgenommene Vogelaufnahmen mit nur wenigen Proben je Klasse (1984 Dateien, neun Klassen). Auf diese Weise wurde die Anwendbarkeit von DL-Methoden auf Datensätze mit geringem Umfang sowie Daten mit nicht vereinheitlichten Aufnahmebedingungen demonstriert. Hierzu wurde eine auf Python und dem ML-Framework PyTorch basierende Softwarelösung entwickelt, die die Transformation der Eingangsdaten in Mel-Spektrogramme sowie das Training von KNN ermöglicht.

Im Zuge einer zweistufigen Experimentierphase wurden 33 verschiedene Modellarchitekturen getestet, wovon das beste Modell optimiert wurde. Als Ergebnis konnte ein auf der CNN-Architektur basierendes Modell mit einer Vorhersagegenauigkeit von 80,4 %, entwickelt werden. Des Weiteren wurde der AutoML-Service Google Vision, der die Automatisierung einzelner ML-Phasen (Training, Modellierung, Evaluierung) ermöglicht, auf denselben Datensatz angewendet. Mit diesem konnte eine Vorhersagegenauigkeit von 83,4 % erreicht werden. Somit konnte mit beiden Lösungen erfolgreich eine Klassifizierung von Audiosignalen durchgeführt werden.

Ferner wurden zusätzliche Optimierungsmöglichkeiten präsentiert, die sich weiterhin positiv auf das Ergebnis auswirken können. Zum Abschluss dieser Arbeit wurde gezeigt, wie selbst entwickelte KI-Lösungen zur Audiosignalklassifizierung in die Google-Umgebung integriert werden können. Die ML-Phasen Datenanalyse, Datenvorverarbeitung, Modellierung, Training, Evaluierung sowie Veröffentlichung des Modells im Bereich der Klassifizierung von Audiosignalen konnten erfolgreich bearbeitet werden. Dabei konnten Anwendung und Verknüpfung der beiden zukunftsträchtigen Technologien Cloud-Computing und KI im Audiobereich demonstriert werden.

Es hat sich gezeigt, dass die Entwicklung einer cloud- und KI-basierten Lösung zur Audiosignalanalyse umfangreich und komplex ist. Das Gesamtsystem besteht aus vielen einzelnen Komponenten, die ein domänenübergreifendes Wissen erfordern. Der Entwurf der Lösung benötigt deshalb eine bedachte und strukturierte Vorgehensweise.

Mit dem Crisp-DM-Vorgehensmodell lässt sich der Entwicklungsprozess in einzelne Phasen unterteilen, wodurch eine getrennte Entwicklung der Komponenten ermöglicht wird. Als Strukturierungselement werden im Zuge der Implementierung Klassen verwendet. Einzelne Klassen, die sich nicht auf einen speziellen Datensatz beziehen (z. B. Preprocessing), können somit projektübergreifend wiederverwendet werden, was sich auf die Entwicklungszeiten ähnlicher Projekte positiv auswirkt.

Eine weitere Hürde, die ML-Expert*innen bei der erstmaligen Umsetzung eines KI-Projektes überwinden müssen, ist die Auswahl geeigneter Entwicklungsumgebungen. Der Vergleich der drei größten Anbieter Amazon, Google und Microsoft demonstriert dabei vor allem die Vielfältigkeit der angebotenen Services der Provider. Dies bedeutet, dass grundsätzlich alle drei Anbieter engagiert sind, ein möglichst breites Spektrum an Anwendungsfällen abzudecken, und somit gleichwertige Services zur Entwicklung einer Lösung für die Audiosignalklassifizierung offerieren. Es kann daher kein bevorzugter Anbieter empfohlen werden.

Mit Google Vertex AI wurde eine junge KI-Plattform (2021) gewählt, die es ermöglicht, alle Crisp-DM-Phasen abzudecken. Die Services, die Vertex AI bietet, sind umfangreich und konnten im Verlauf dieser Arbeit nicht alle genutzt werden. Jedoch kann die Handhabung der verwendeten Services als komfortabel beschrieben werden. Beispielsweise lässt sich das Aktivieren bzw. Deaktivieren oder das Skalieren von Hardwareressourcen in nur wenigen Sekunden durchführen. Teilweise gestaltet es sich schwierig, den Überblick über die Kosten zu behalten, denn es ist nicht immer eindeutig nachvollziehbar, welche Services im Hintergrund mitgenutzt werden. Des Weiteren zeigte sich, dass von Vertex AI angebotene ML-Software-Services teuer sind (z. B. Google Vision). Die Durchführung von größeren KI-Projekten auf Vertex AI kann sich deshalb kostenintensiv gestalten.

Ferner wird für die softwaretechnische Implementierung ein ML-Framework benötigt. Ähnlich wie bei den Clouds gibt es auch hier die drei großen Frameworks Keras, TensorFlow und PyTorch, die als gleichwertig zu betrachten sind. Der Vorteil von PyTorch ist, dass das Framework mit Torchaudio eine umfangreiche Softwarebibliothek zur Analyse und Manipulation von Audiosignalen inkludiert. Die Implementierung mittels PyTorch ist zu empfehlen, da einerseits eine umfangreiche Dokumentation zur Verfügung steht und andererseits viele Codebeispiele im Bereich der Audioklassifizierung bereitgestellt werden.

Bereits während der Literaturrecherche wurde ersichtlich, dass es keine allgemeine Lösung zur KI-basierten Audiosignalklassifizierung gibt. Jede Lösung hängt maßgeblich von den Daten ab. Je nach Qualität und Menge der (Trainings-)Daten können die Klassifizierungsmodelle hinsichtlich Architektur, Schichten und Hyperparameterkonfigurationen stark variieren. Das Hauptproblem stellt hierbei die Anzahl der Hyperparameter dar, denn die Möglichkeiten unterschiedlicher Parameterkombinationen sind scheinbar unendlich. Zum Finden geeigneter Modelle und Einstellungen können Jupyter-Notebooks gewählt werden. Nachteil dieser Methode ist, dass dafür fortgeschrittene Softwareentwicklungskenntnisse benötigt werden und sich das Training, Optimieren und Evaluieren der Modelle zeitintensiv gestaltet. Für größere ML-Projekte ist die ausschließliche Nutzung dieser Notebooks daher nicht zu empfehlen.

Eine Alternative bietet beispielsweise der vorgestellte Vertex-AI-Trainingsservice, wobei denkbar ist, dass Notebooks zunächst zum Erkennen von Trends verwendet werden und die anschließende Optimierung mit dem automatisierten Training erfolgt. Letztendlich muss ein Kompromiss zwischen aufgewendeter Zeit bzw. Kosten und Modell-Performance geschlossen werden, da es aufgrund der Vielzahl an Parameterkombinationsmöglichkeiten nicht möglich ist, ein ‚perfektes' Modell zu entwickeln.

Des Weiteren bietet Google eine Reihe an AutoML-Services für unterschiedliche Aufgabenstellungen an. Für eine Audiosignalklassifizierung gibt es dabei keinen dezidierten Service. Jedoch kann aufgrund der Mel-Spektrogramme der eigentlich für die Bildklassifizierung gedachte AutoML-Service Google Vision für die Umsetzung verwendet werden. AutoML-Vision von Google kann aufgrund der hohen Kosten und der Tatsache, dass für die Extraktion der Mel-Spektrogramme Programmierkenntnisse benötigt werden, nur bedingt empfohlen werden. Dennoch ist es denkbar, ein mit AutoML trainiertes Modell als Ausgangsbasis für weitere Optimierungen zu verwenden.

Obwohl die beiden entwickelten Modelle mit einer Vorhersagegenauigkeit von 80,4 % und 83,3 % nicht für den Einsatz in einer professionellen Umgebung geeignet sind, kann aufgrund der weiters vorgestellten Optimierungsmöglichkeiten ein positives Resümee gezogen werden. Generell wurde gezeigt, dass es in jeder Phase, die bearbeitet wurde, noch Möglichkeiten zur Verbesserung gibt. Insbesondere im Zuge des Preprocessing könnten die Daten auf verschiedene Art und Weise ohne das Hinzufügen von neuen Dateien vervielfältigt werden. Dass die Anzahl der Dateien einen wesentlichen Einfluss auf das Ergebnis haben kann, wurde im Zuge der zweiten Evaluierungsphase bestätigt, denn die Klasse mit den meisten Proben (500) erreichte eine Vorhersagegenauigkeit von 94 %.

Des Weiteren bietet Google mit Vertex AI eine Vielzahl an zusätzlichen Möglichkeiten für den vereinfachten Umgang mit ML-Elementen. Die Integration von selbst entwickelten Komponenten gestaltet sich dabei schwierig. Dennoch können sich daraus Vorteile ergeben. So kann beispielsweise die integrierte Trainingspipeline auch für neue Aufgabenstellungen verwendet werden. Ferner stellt Google für die Modellveröffentlichung eine mächtige Recheninfrastruktur zur Verfügung, die schnelle Reaktionszeiten garantiert. Insbesondere bei zeitkritischen Prozessen ist dies eine zentrale Anforderung an das Modell.

Für künftige Anwendungen wäre es zudem denkbar, weitere Elemente des Crisp-DM-Modells zu entkoppeln, in Vertex AI zu integrieren und die Vorteile der Cloud-Umgebung zu nutzen. Gemeint ist damit beispielsweise die Entkopplung der Preprocessing-Phase. Anwender*innen könnten im Anschluss das Preprocessing-Element in Vertex AI mittels grafischer Oberfläche und ohne Programmierkenntnisse konfigurieren. In Kombination mit dem AutoML-Vision-Service würde dies eine ganzheitliche Lösung ohne Softwareentwicklungstätigkeiten ermöglichen.

LITERATURVERZEICHNIS

Gedruckte Werke (29)

Awad, Mariette; Khanna, Rahul (2015): *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*, Apress Media, LLC, New York

Baun, Christian; Kunze, Marcel; Nimis, Jens; Tai, Stefan (2010): *Cloud Computing: Web-basierte dynamische IT-Services*, Springer-Verlag, Berlin Heidelberg

Blohm, Matthias; Brandt, David; Grigorjan, Arthur; Hanussek, Marc; Harald, Papp; Hennebold, Christoph; Kintz, Maximilien; Oberle, Michael; Bauer, Willhelm (Hrsg.); Bauernhansl, Thomas (Hrsg.); Huber, Marco (Hrsg.); Kraus, Werner (Hrsg.); Peissner, Matthias (Hrsg.); Renner, Thomas (Hrsg.) (2021): *Cloudbasierte KI-Plattformen: Chancen und Grenzen von Diensten für Machine Learning As A Service*, ohne Verlagsangaben, Fraunhofer-Institut, Stuttgart

Buduma, Nikhil (2017): *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*, O'Reilly Media, Inc., Sebastopol, California

Chapman, Pete; Clinton, Julian; Kerber, Randy; Khabaza, Thomas; Reinartz, Thomas; Shearer, Colin; Wirth, Rüdiger (1999): *Crisp-DM 1.0: Step-by-step data mining guide*, Crisp-DM Consortium

Charu, Aggarawal (2018): *Neural Networks and Deep Learning*, Springer International Publishing AG, New York

Cleve, Jürgen; Lämmel, Uwe (2012): *Künstliche Intelligenz*, 4. Auflage, Carl Hanser Verlag, München

Döbel, Inga; Leis, Miriam; Molina Vogelsang, Manuel; Neustroev, Dmitry; Petzka, Henning; Rüping, Stefan; Voss, Angelika; Wegele, Martin; Welz, Juliane: *Maschinelles Lernen: Kompetenzen, Anwendungen und Forschungsbedarf*, ohne Verlagsangaben, Fraunhofer-Gesellschaft, München

Ellis, Dan; Gold, Ben; Morgan, Nelson (2011): *Speech and Audio Signal Processing: Processing and Perception of Speech and Music*, Springer International Publishing AG 2018, New York

Ellis, Dan; Plumbley, Mark; Virtanen, Tuomas (2018): *Computational Analysis of Sound Scenes and Events*, Springer International Publishing AG 2018, New York

Elmer, Miriam; Niggemann, Oliver (2022): *Künstliche Intelligenz in Produktion und Maschinenbau: Hintergründe, Anwendungsszenarien, Expertentipps*, VDE Verlag GmbH, Berlin

Ertel, Wolfgang (2016): *Grundkurs Künstliche Intelligenz: Eine praxisorientierte Einführung*, 4., überarbeitete Auflage, Springer Vieweg, Wiesbaden

Gentsch, Peter (2019): *Künstliche Intelligenz für Sales, Marketing und Service: Mit AI und Bots zu einem Algorithmic Business – Konzepte und Best Practices*, 2., überarbeitete Auflage, Springer Fachmedien Wiesbaden GmbH, Wiesbaden

Hanussek, Mark; Papp, Harald (o.J.): *A Methodology for Deriving Evaluation Criteria for Software Solutions, Universität Stuttgart*, Stuttgart

Huyen, Chip (2022): *Designing Machine Learning Systems: An Iterative Process for Production-Ready Applications*, O'Reilly Media, Inc., Sebastopol, California

Kersting, Kristian; Lampert, Christoph; Rothkopf, Constantin (2019): *Wie Maschinen lernen: Künstliche Intelligenz verständlich erklärt*, Springer Fachmedien Wiesbaden GmbH, Wiesbaden

Lerch, Alexander (2012): *An Introduction to Audio Content Analysis: Applications in Signal Processing and Music Informatics*, John Wiley & Sons, Inc., Hoboken, New Jersey

Meinel, Christoph; Roschke, Sebastian; Schnajkin, Maxim; Willems, Christian (2011): *Virtualisierung und Cloud Computing: Konzepte, Technologiestudie, Marktübersicht*, Universitätsverlag Potsdam, Potsdam

Michelbach, Thomas; Oussama, Riahi (2013): *Vergleich der Energieeffizienz von Cloud Infrastrukturen*, Universität Stuttgart, Stuttgart

Mitchell, Tom (1997): *Machine Learning*, McGraw-Hill Science/Engineering/Math, Portland, OR

Münzl, Gerald; Pauly, Michael; Reti, Martin (2015): *Cloud Computing als neue Herausforderung für Management und IT*, Springer-Verlag, Berlin Heidelberg

Plaue, Matthias (2021): *Data Science: Grundlagen, Statistik und maschinelles Lernen*, Springer Spektrum, Berlin Heidelberg

Prem, Erich; Ruhland, Sascha (2019): *Artificial Intelligence Potenzial Österreich: Zahlen, Daten, Fakten*, Ohne Verlagsangaben, Bundesministerium für Verkehr, Innovation und Technologie Österreich

Ransome, James; Rittinghouse, John (2010): *Cloud Computing: Implementation, Management and Security*, CRC Press, Boca Raton

Rajalingappaa, Shanmugamani; Rajesh, Arumugam (2018): *Hands-On Natural Language Processing with Python: A practical guide to applying deep learning architectures to your NLP applications*, Packt Publishing Ltd., Birmingham

Sonnet, Daniel (2022): *Neuronale Netze kompakt: Vom Perceptron zum Deep Learning*, Springer Fachmedien Wiesbaden GmbH, Wiesbaden

Wennker, Phil (2020): *Künstliche Intelligenz in der Praxis: Anwendung im Unternehmen und Branchen: KI wettbewerbs- und zukunftsorientiert einsetzen*, Springer Fachmedien Wiesbaden GmbH, Wiesbaden

Wittpahl, Volker (2019): *Künstliche Intelligenz: Technologie* | *Anwendung* | *Gesellschaft*, Springer Vieweg, Berlin Heidelberg

Wuttke, Laurenz (2021): *Praxisleitfaden für Künstliche Intelligenz in Marketing und Vertrieb: Beispiele, Konzepte und Anwendungsfälle*, Springer Gabler, Wiesbaden

Wissenschaftliche Paper (16)

Becker, Sören; Ackermann, Marcel; Lapuschkin, Sebastian; Müller, Klaus-Robert; Samek, Wojciech (2018): *Interpreting and Explaining Deep Neural Networks for Classification of Audio Signals*, arXiv: 1807.03418

Benito-Gorron, Diego de; Lozano-Diez, Alicia; Toledano, Doroteo; Gonzalez-Rodriguez, Joaquin (2019): *Exploring convolutional, Recurrent, and hybrid deep neural networks for speech and music detection in a large audio dataset*, in: EURASIP Journal on Audio, Speech and Music Processing, Articel number 2019/9, S. 1-18

Petmezas, Georgios; Cheimariotis, Grigorios-Aris; Leandros, Stefanopulos; Rocha, Bruno Paiva; Rui Pedro; Katsaggelos, Aggelos; Maglaveras, Nicos (2022): *Automated Lung Sound Classification Using a Hybrid CNN-LSTM Network and Focal Loss Function*, in: *Sensors* 22, no. 3: 1232

Dai, Wei; Dai, Chia; Qu, Schuhui; Li, Jungcheng; Das, Samarjit: (2017): *Very deep convolutional neural networks for raw waveforms*, arXiv: 1610.00087

Gupta, Gaurav; Kshirsagar, Meghana; Zhong, Ming; Gholami, Shahrzad; Ferres, Juan Lavista (2021): *Comparing recurrent convolutional neural networks for large scale bird species classification*, in: Sci Rep 11, 17085

Han, Shipeng; Meng, Zhen; Zhang, Xingcheng; Yan, Yuepeng (2021): *Hybrid Deep Recurrent Neural Networks for Noise Reduction of MEMS-IMU with Static and Dynamic Conditions*, in: Micromachines 2021, 12, S. 214 – 237

He, Kaiming; Zhang, Xiangyu; Ren, Schaoqing; Sun, Jian (2015): *Deep Residual Learning for Image Recognition*, arXiv:1512.03385

Hochreiter, Sepp; Schmidhuber, Jürgen (1997): *Long Short-Term Memory*, in: Neural Computation, vol. 9, no. 8, S. 1735 – 1780.

Kong, Qiuqiang; Cao, Yin; Iqbal, Turab; Wang, Yuxuan; Wang, Wenwu; Plumbley, Mark (2019): *PANNs: Large-Scale Pretrained Audio Neural Networks for Audio Pattern Recognition (Pretrained Models)*, arXiv: 1912.10211

Molau, Sirko; Pitz, Michael; Schlüter, Ralf; Ney, Hermann (2001): *Computing Mel-frequency cepstral coefficients on the power spectrum*, in IEEE International Conference on Acoustics, Speech, and Signal Processing (Cat.01Ch37221)

Nanni, Loris; Maguolo, Gianluca; Brahnam, Sheryl; Paci, Michelangelo (2021): *An Ensemble of Convolutional Neural Networks for Audio Classification*, in: Applied Sciences 11, Vol. no. 13, 5796

Purwins, Hendrik; Li, Bo; Virtanen, Tuomas; Schlüter, Jan; Chang, Shuoyin; Sainath, Tara (2019): *Deep Learning for Audio Signal Processing*, in: Journal of Selected Topics of Signal Processing, Vol. 13, No. 2, Mai 2019, S. 206 – 2019

Salmanabadi, Homa; Ghaffari, Iman; Romaric, Evina; Tatu, Serioja; Dufour, Steven (2022): *A Hybrid LSTM-ResNet Deep Neural Network for Noise Reduction and Classification of V-Band Receiver Signals*, in: IEEE Access (Volume 10), January 2022, S. 14797 – 14806

Sharma, Garima; Umapathy, Kartikeyan; Krishnan, Sridhar Garima (2019): *Trends in audio signal feature extraction Methods*, in: Applied Acoustics 158, 2020, S. 1 - 21.

Tanveer, Hassan; Zhu, Hongxiao; Waqar, Ahmed; Antony, Thomas; Basti Muhammed, Imran; Muhammad, Salman (2021): *Mel-spectrogram and Deep CNN Based Representation Learning from Bio-Sonar Implementation on UAVs*, in: International Conference on Computer, Control and Robotics, January 2021

Zhang, Boyang; Leitner, Jared; Thornton, Sam (2019): *Audio Recognition using Mel Spectrograms and Convolutional Neural Networks*, University of California, San Diego.

Online-Quellen (56)

Amazon 1 (2022): *Was ist AWS?* https://aws.amazon.com/de/what-is-aws/?nc1=f_cc [Stand 12.09.2022]

Amazon 2 (2022): *Amazon SageMaker* https://aws.amazon.com/de/sagemaker/ [Stand 12.09.2022]

Amazon 3 (2022): *Amazon Rekognition* https://aws.amazon.com/de/rekognition/ [Stand 12.09.2022]

Amazon 4 (o.J.): Creating a manifest file from a CSV file https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/ex-csv-manifest.html [Stand 12.09.2022]

Amazon 5 (2022): *AWS Pricing Calculator: Estimate the cost for your architecture solution.* https://calculator.aws/#/?ch=cta&cta=lower-pricing-calc [Stand 12.09.2022]

Brunel (o.J.): *Maschinelles lernen* https://www.brunel.net/de-de/management-ratgeber/maschinelles-lernen#:~:text=Beim%20%C3%BCberwachten%20maschinellen%20Lernen%20lernt,Anschlie%C3%9Fend%20wird%20die%20Vorhersage%20bewertet. [Stand 07.08.2022]

Chiusano, Fabio (2022): *Deploy a PyTorch Model with Flask on GCP Vertex AI* https://medium.com/nlplanet/deploy-a-pytorch-model-with-flask-on-gcp-vertex-ai-8e81f25e605f [Stand 17.11.2022]

Docker (o.J.): *Dockerfile reference* https://docs.docker.com/engine/reference/builder/ [Stand 17.11.2022]

Doshi, Ketan (2021): *Audio Deep Learning Made Simple: Sound Classification, Step-by-Step* https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5 [Stand 17.11.2022]

Fraunhofer-Gesellschaft (2021): *Qualitätskontrolle via Sound* https://www.fraunhofer.de/de/presse/presseinformationen/2021/maerz-2021/qualitaetskontrolle-via-sound.html [Stand 17.11.2022]

Fraunhofer-ITWM (o.J.): Predictive Maintenance: *Die Effektivität von Anlagen durch Machine Learning erhöhen* https://www.itwm.fraunhofer.de/de/abteilungen/sys/maschinenmonitoring-und-regelung/predictive-maintenance-instandhaltung-machinelearning.html [Stand 17.11.2022]

Gartner (2021): *Gartner Magic Quadrant for Cloud Infrastructure and Platform Services* https://www.gartner.com/en/documents/3989743 [Stand 12.09.2022]

Google 1 (2022): *Google Cloud* https://cloud.google.com/?hl=de [Stand 12.09.2022]

Google 2 (2022): *Vertex AI* https://cloud.google.com/vertex-ai?hl=de [Stand 12.09.2022]

Google 3 (2022): *Vision AI* https://cloud.google.com/vision?hl=de [Stand 12.09.2022]

Google 4 (2022): *Google Cloud Pricing Calculator* https://cloud.google.com/products/calculator?hl=de [Stand 12.09.2022]

Google 5 (2021): *New to ML: Learning path on Vertex AI* https://cloud.google.com/blog/topics/developers-practitioners/new-ml-learning-path-vertex-ai [Stand 12.09.2022]

Google 6 (2022): *Einführung in Vertex AI Feature Store* https://cloud.google.com/vertex-ai/docs/featurestore/overview?hl=de&_ga=2.48546350.-960570938.1659990308 [Stand 12.09.2022]

Google 7 (2021): *Announcing Vertex Pipelines general availability* https://cloud.google.com/blog/products/ai-machine-learning/serverless-machine-learning-pipelines-on-google-cloud [Stand 12.09.2022]

Google 8 (2022): *Einführung in Vertex AI Experiments* https://cloud.google.com/vertex-ai/docs/experiments/intro-vertex-ai-experiments?hl=de&_ga=2.11758140.-960570938.1659990308 [Stand 12.09.2022]

Google 9 (2022): *Batchvorhersagen abrufen* https://cloud.google.com/vertex-ai/docs/predictions/batch-predictions?hl=de&_ga=2.11329980.-960570938.1659990308 [Stand 12.09.2022]

Google 10 (2022): *Einführung in Vertex ML Metadata* https://cloud.google.com/vertex-ai/docs/ml-metadata/introduction?hl=de [Stand 12.09.2022]

Google 11 (2021): *PyTorch on Google Cloud: How to train and tune PyTorch models on Vertex AI* https://cloud.google.com/blog/topics/developers-practitioners/pytorch-google-cloud-how-train-and-tune-pytorch-models-vertex-ai?hl=en [Stand 17.11.2022]

Google 12 (2022): *Python-Trainingsanwendung für einen vordefinierten Container erstellen* https://cloud.google.com/vertex-ai/docs/training/create-python-pre-built-container [Stand 17.11.2022]

Google 13 (2022): *Benutzerdefinierte Containeranforderungen für Vorhersagen* https://cloud.google.com/vertex-ai/docs/predictions/custom-container-requirements#server [Stand 17.11.2022]

Google 14 (o.J.): *Vorhersage für benutzerdefiniertes Modell* https://cloud.google.com/vertex-ai/docs/samples/aiplatform-predict-custom-trained-model-sample#aiplatform_predict_custom_trained_model_sample-python [Stand 17.11.2022]

Jupyter (2022): *Free software, open standards, and web services for interactive computing across all programming languages* https://jupyter.org/ [Stand 12.09.2022]

Kaggle 1 (2018): *Freesound General-Purpose Audio Tagging Challenge* https://www.kaggle.com/competitions/freesound-audio-tagging/data [Stand 12.09.2022]

Kaggle 2 (2022): *BirdCLEF 2022: Data Description* https://www.kaggle.com/competitions/birdclef-2022/data [Stand 17.11.2022]

Kaggle 3 (2022): *BirdCLEF 2022: Overview* https://www.kaggle.com/competitions/birdclef-2022/overview [Stand 17.11.2022]

Kaggle 4 (2022): BirdCLEF 2022: Rules https://www.kaggle.com/competitions/birdclef-2022/rules [Stand 17.11.2022]

Librosa (2015): *Librosa* https://librosa.org/doc/latest/index.html [Stand 12.09.2022]

Microsoft Corporation 1 (2022): *Machine Learning Algorithm Cheat Sheet for Azure Machine Learning Designer* https://docs.microsoft.com/en-us/azure/machine-learning/algorithm-cheat-sheet [Stand 07.08.2022]

Microsoft Corporation 2 (2022): *What are Azure Machine Learning pipelines?* https://docs.microsoft.com/en-us/azure/machine-learning/algorithm-cheat-sheet [Stand 07.08.2022]

Microsoft Corporation 3 (2022): *Microsoft Azure* https://azure.microsoft.com/de-de/ [Stand 12.09.2022]

Microsoft Corporation 4 (2022): *Azure Machine Learning* https://azure.microsoft.com/de-de/services/machine-learning/#features [Stand 12.09.2022]

Microsoft Corporation 5 (2022): *Evaluate automated machine learning experiment results* https://docs.microsoft.com/en-us/azure/machine-learning/how-to-understand-automated-ml [Stand 12.09.2022]

Microsoft Corporation 6 (2022): *Custom Vision https://azure.microsoft.com/de-de/services/cognitive-services/custom-vision-service/#overview* [Stand 12.09.2022]

Microsoft Corporation 7 (2022): Preisrechner https://azure.microsoft.com/de-de/pricing/calculator/ [Stand 12.09.2022]

NVIDIA Corporation (o.J.): *Audio Spectrogram*, Audio spectrogram — NVIDIA DALI 1.19.0 documentation [Stand 19.11.2022]

Pardo, Bryan (2008): *The Digital Fourier Transform* https://interactiveaudiolab.github.io/teaching/eecs352stuff/CS352-topic6-DFT.pdf [Stand 12.09.2022]

PyTorch 1 (o.J.): *PyTorch: From research to production* https://pytorch.org/ [Stand 17.11.2022]

PyTorch 2 (2021): *Audio manipulation with PyTorch* https://pytorch.org/tutorials/beginner/audio_preprocessing_tutorial.html [Stand 17.11.2022]

PyTorch 3 (2022): *Tensors* https://pytorch.org/docs/stable/tensors.html [Stand 17.11.2022]

PyTorch 4 (2022): *MelSpectrogram* https://pytorch.org/audio/stable/generated/torchaudio.transforms.MelSpectrogram.html#torchaudio.transforms.MelSpectrogram [Stand 17.11.2022]

PyTorch 5 (o.J.): *ResNet https://pytorch.org/hub/pytorch_vision_resnet/* [Stand 17.11.2022]

PyTorch 6 (2022): *Training a classifier* https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html [Stand 17.11.2022]

Scikit-Learn (o.J.): *Metrics and scoring: quantifying the quality of predictions* https://scikit-learn.org/stable/modules/model_evaluation.html [Stand 17.11.2022]

Skoda Auto DigiLab (o.J.): *Sound Analyzer: Application for quick diagnostics of vehicle noises* https://skodaautodigilab.com/en/projects/sound-analyser [Stand 17.11.2022]

Standard (2018): *Künstliche Intelligenz als Milliardengeschäft: Österreich hinkt hinterher* https://www.derstandard.at/story/2000077600017/physiker-mathematiker-fuer-kuenstliche-intelligenz-gesucht [Stand 17.11.2022]

Standford University (o.J.): CS231n Convolutional Neural Networks for Visual Recognition CS231n Convolutional Neural Networks for Visual Recognition [Stand 17.11.2022]

Synergy Research Group (2022): *As Quarterly Cloud Spending Jumps to Over $50B, Microsoft Looms Larger in Amazon's Rear Mirror* https://www.srgresearch.com/articles/as-quarterly-cloud-spending-jumps-to-over-50b-microsoft-looms-larger-in-amazons-rear-mirror [Stand 12.09.2022]

Wuttke, Laurenz 1 (o.J.): *Machine Learning vs. Deep Learning: Wo ist der Unterschied?* https://datasolut.com/machine-learning-vs-deep-learning/ [Stand 07.08.2022]

Wuttke, Laurenz 2 (o.J.): *Reinforcment Learning: Wenn KI auf Belohnungen reagiert,* https://datasolut.com/reinforcement-learning/ [Stand 07.08.2022]

Wuttke, Laurenz 3 (o.J.): *Künstliche Neuronale Netzwerke: Definition, Einführung, Arten und Funktion* https://datasolut.com/neuronale-netzwerke-einfuehrung/ [Stand 07.08.2022]

Wuttke, Laurenz 4 (o.J.): *Transfer Learning: Grundlagen und Einsatzgebiete* https://datasolut.com/neuronale-netzwerke-einfuehrung/ [Stand 12.09.2022]

ABBILDUNGSVERZEICHNIS

Abbildungsverzeichnis

Abbildungsverzeichnis

TABELLENVERZEICHNIS