Automatisiertes Ergänzen von frequenzlimitierten Mikrofonaufnahmen mittels Machine Learning anhand der eigenen Stimme

Fakultät Electronic Media
Audiovisuelle Medien
Hochschule der Medien

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Engineering

vorgelegt von

Sebastian Greim
geboren am 23.01.2000 in Bayreuth

im April 2023

Erstprüfer: Prof. Oliver Curdt
Zweitprüfer: M.Eng. Matvey Fridman

Eidesstattliche Erklärung

„Hiermit versichere ich, Sebastian Greim, ehrenwörtlich, dass ich für die vorliegende Bachelorarbeit mit dem Titel: „Automatisiertes Ergänzen von Frequenzlimitierten Mikrofonaufnahmen mittels Machine Learning“ keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden. Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 26 Abs. 2 Bachelor-SPO (6 Semester), § 24 Abs. 2 Bachelor-SPO (7 Semester), § 23 Abs. 2 Master-SPO (3 Semester) bzw. § 19 Abs. 2 Master-SPO (4 Semester und berufsbegleitend) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.“

Ort, Datum: Stuttgart, 03.04.23 Unterschrift: 

Zusammenfassung / Abstract

Während der Corona Pandemie mit Kontaktbeschränkungen und Lockdowns rückten Anrufe und Videocall Plattformen stark in den Vordergrund. Gleichzeitig nahmen aber auch die Beschwerden über schlechte Tonqualität und Ermüdung nach langen Calls zu.
In dieser Arbeit wird zunächst auf die unterschiedlichen Übertragungswege und ihre Eingriffe in den Frequenzgang des Audios eingegangen.
Anschließend versucht der Autor den Frequenzgang eines Telefonats mittels Machine Learning an den eines Studiomikrofons anzunähern. Zum Trainieren des Algorithmus wird ein Trainingsdatenset erstellt und in Python entsprechend verarbeitet. Danach werden mit unterschiedlichen Loss- und Aktivierungsfunktionen und Anpassungen im Code mehrere Epochen durchlaufen, um zu sehen, welche Konstallation eine frequenzergänzende Veränderung im Spektrum bewirkt.
Schließlich gelingt es, ein Ergebnis zu erzielen, dessen Frequenzgang in mathematischer Ebene näher an den des Studiomikrofons heranreicht als der des Telefonsignals. Jedoch werden zeitgleich starke Artefakte induziert, die das Klangergebnis beeinträchtigen.
In künftigen Arbeiten gilt es, durch diverse Optimierungen des hier beschriebenen Prozesses diese Artefakte abzuschwächen, um somit einen besseren Klang zu erhalten.

Abstract

During the Corona pandemic with contact restrictions and lockdowns, phone calls and video call platforms became very important. At the same time, complaints about poor sound quality and fatigue after long calls also increased.

In this work, the different transmission paths and their influence on the frequency response of the audio are first discussed.

Subsequently, the author attempts to approximate the frequency response of a phone call to that of a studio microphone using machine learning. To train the algorithm, a training data set is created and processed accordingly in Python. Then, several epochs are run with different loss and activation functions and adjustments in the code to see which configuration causes a frequency-enhancing change in the spectrum.

Finally, it is possible to achieve a result whose frequency response approaches that of the studio microphone more closely on a mathematical level than that of the phone signal. However, strong artifacts are induced simultaneously, which impair the sound quality.

In future work, it will be necessary to reduce these artifacts through various optimizations of the process described here in order to obtain better sound quality.

Inhaltsverzeichnis

1 Motivation

Als Deutschland im März 2020 aufgrund von Corona in den Lockdown ging, stellten der Wegfall persönlicher Kontakte, die verringerte körperlicher Betätigung und die vermehrte Mediennutzung große Teile der Gesellschaft vor Herausforderungen. Gleichzeitig rückten Videocall Plattformen wie etwa Zoom oder Webex in den Fokus[1]. Gespräche und Meetings, die zuvor persönlich abgehalten werden konnten, wurden nun auf Videotelefonte ausgelagert, jedoch mit gewissen negativen Nebenwirkungen.
Eine Studie des Instituts für Beschäftigung und Employability bezeichnete das Phänomen, bei dem sich Teilnehmer:innen von virtuellen Kommunikationsplattformen zunehmend ausgelaugt und müde fühlen als „Zoom-Fatigue“[2]. Circa 60% der Teilnehmer:innen an der Studie gaben an unter eben dieser „Zoom Fatigue“ zu leiden, was sich bei den Befragten u.a. in Form von Konzentrationsschwierigkeiten, Ungeduld und Genervtsein äußerte. Bei der Frage nach den Gründen hierfür werden vor allem die Defizite bei der sozialen Interaktion, wie etwa fehlender Small Talk, Schwierigkeiten bei der Wahrnehmung von Mimik, Gestik oder sonstiger non-verbaler Hinweise der anderen Teilnehmenden, genannt. Doch auch technische Defizite werden angeprangert. So gab ca. die Hälfte der Befragten an, sich aufgrund schlechter Tonqualität deutlich mehr konzentrieren zu müssen.
Dies deckt sich mit einer Studie der TU Berlin, bei der die Hirnströme von Probanden mittels EEG gemessen wurden, während diesen nacheinander unkomprimierte und digital komprimierte Tondateien vorgespielt wurden. Hierdurch wurde festgestellt, dass das menschliche Gehirn deutlich mehr arbeiten muss, wenn es komprimierte Signale anhört, da es permanent versucht, Lücken im Frequenzspektrum zu ergänzen. Der Studie zufolge ist dieser erhöhte Aufwand des Gehirns sogar bereits bei guter mp3-Codierung messbar, obwohl hier ein - zumindest für Laien - kaum wahrnehmbarer Unterschied vorliegt[3]. Somit stellt ein Telefonat mittels virtueller Kommunikationsplattformen wahrlich einen Kraftakt für das menschliche Gehirn dar, da zur Übertragung sowohl des Bildes als auch des Tons starke Kompressions-

[1] RAMACHANDRAN 2021.
[2] Rump und Brandt 2020.
[3] Jüngling 2015.

algorithmen angewandt werden, um die Datenmengen bei der Übertragung zu reduzieren (siehe Kapitel 2.2.6)
Das Ziel dieser wissenschaftlichen Arbeit besteht darin, einen Lösungsansatz zu finden und zu evaluieren, mit dem man frequenzlimitierte Audioaufnahmen in ihrem Frequenzspektrum ergänzen kann, ohne dabei die Bandbreite der Übertragung vergrößern zu müssen. Da im Frequenzgang der menschlichen Stimme ebenso wie in dem von Musikinstrumenten gewisse Muster und Charakteristiken erkennbar sind (siehe Kapitel 2.2.4), soll hierfür ein Fachgebiet der Informatik Verwendung finden, das darauf spezialisiert ist, ebensolche Muster zu erkennen: Machine Learning (bzw. Künstliche Intelligenz).
Im Folgenden sollen sowohl die Grundlagen der frequenzspezifischen Charakteristika der menschlichen Stimme als auch die fundamentalen Schritte von Maschinellem Lernen erläutert werden. Anschließend soll in einem Praxisbeispiel ein Lösungsansatz in Python implementiert werden, mit dem das Ergänzen von frequenzlimitierten Audioaufnahmen von der Stimme des Autors möglich sein soll.
Zuletzt sollen die Ergebnisse interpretiert sowie ein Fazit gezogen werden, ob die verwendeten Methoden zum gewünschten Ergebnis geführt haben.

2 Grundlagen

2.1 Eigenschaften der menschlichen Stimme

2.1.1 Obertonreihe

Möchte man einen bestimmten Ton auf einem Instrument (Ausnahme Percussion) spielen oder mit der Stimme singen, so erklingt nie ein reiner Sinuston, sondern vielmehr eine Mischung aus einem Grundton und dessen harmonischen Obertönen. Bei diesen Obertönen handelt es sich um ganzzahlige Vielfache der Grundfrequenz. Diese Grundfrequenz liegt bei einer männlichen Stimme im Durchschnitt bei 120Hz[1], kann aber bei basslastigeren Stimmen auch unter 100Hz liegen.
Der Grundton bestimmt die Tonhöhe der Sprache. So liegen beispielsweise 200Hz eine Oktave über 100Hz. Zwei Stimmen, die denselben Grundton singen, erzeugen demnach auch die gleichen Obertöne.

2.1.2 Vokal-Formanten

Dass dennoch keine Stimme gleich der anderen ist, liegt an den sogenannten Formanten[2]. Als Formanten bezeichnet man bestimmte Bereiche im Frequenzspektrum, die unabhängig vom Grundton durch die Stimme angehoben werden. Man kann sich diese wie Filter vorstellen, die auf das natürlich produzierte Obertonspektrum gelegt werden. Hierbei unterscheidet man zwischen Vokalformanten und Nebenformanten.
Durch die **Vokalformanten** formt man - wie der Name bereits verrät - die Vokale. Je nach Vokal werden durch Veränderungen in der Kiefer- und Lippenstellung unterschiedliche Frequenzbereiche verstärkt, wodurch die charakteristischen Klänge von „a“, „e“, „i“ und anderen Vokalen entstehen. Die Stimmbänder liefern demnach nur einen „Basisklang“, der anschließend erst durch das Ansatzrohr (Bereich oberhalb des Kehlkopfes) geformt wird. Dass die Vokale nicht in den Stimmbändern geformt werden, lässt sich dadurch beweisen,

[1] Friedrich 2008, S. 50.
[2] Friedrich 2008, S. 48.

dass man auch beim Flüstern, einer Technik, bei der die Stimmbänder inaktiv sind, in der Lage ist, Vokale auszusprechen[3].
Entscheidend bei der Vokalformung sind vor allem die ersten beiden Formanten, die auch als F1 und F2 bezeichnet werden. In Abbildung 2.1 sind die den Vokalen zugehörigen Frequenz-

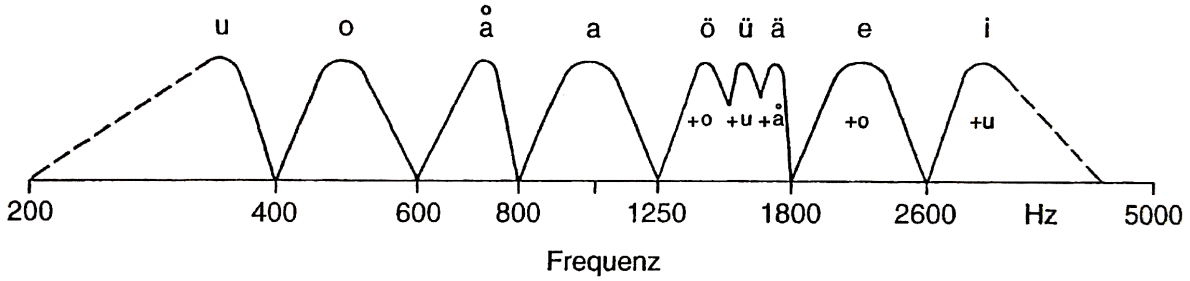

Abbildung 2.1: Frequenzbereiche der Vokalformanten in der deutschen Sprache
Weinzierl und Tonmeister 2008, S. 129

bereiche eingezeichnet, die von jedem Menschen gebildet werden müssen, damit der Hörer den entsprechenden Vokal identifizieren kann. Zwar variiert die genaue Frequenzlage der F1- und F2-Formanten je nach Geschlecht und Alter, aber dennoch gelten die in Abbildung 2.1 aufgezeigten Richtwerte, da die Vokalformanten unabhängig vom Grundton sind. Um beispielsweise ein „a“ zu singen, muss eine hohe weibliche Stimme die gleichen Vokalformanten (in diesem Fall F1=1000Hz und F2=1400Hz[4]) bilden wie eine tiefere männliche.

2.1.3 Nebenformanten

Dass sich menschliche Stimmen dennoch so stark unterscheiden, liegt vor allem an den **Nebenformanten**, die im Frequenzspektrum oberhalb der Vokalformanten liegen und in Abhängigkeit des Ansatzrohrs von Person zu Person stark variieren können[5]. Somit sind Nebenformanten dafür verantwortlich, dass keine Stimme der anderen gleicht.
Um einen besonderen Nebenformanten handelt es sich beim **Sängerformant**, der vor allem bei klassisch ausgebildeten Stimmen vorzufinden ist. Hierbei handelt es sich um einen besonders stark ausgeprägten Formanten im Bereich zwischen 2300Hz-2900Hz bei Männern und 2900Hz-3200Hz bei Frauen[6], der es Sänger:innen ermöglicht, sich mit ihrer Stimme ge-

[3]Friedrich 2008, S. 49.
[4]Friedrich 2008, S. 49.
[5]Weinzierl und Tonmeister 2008, S. 128 f.
[6]Weinzierl und Tonmeister 2008, S. 141.

genüber dem Orchester durchzusetzen, dessen Instrumente in dieser Frequenzlage bereits deutlich schwächere Komponenten aufweisen.

2.1.4 Sprachverständlichkeit

Der Frequenzbereich, der durch den Sängerformanten angehoben wird, ist des Weiteren für die Sprachverständlichkeit hauptverantwortlich. Da die meisten nichtklingenden Konsonanten (k, p, s, t, etc.) in den Bereich zwischen 2-4kHz fallen, beeinflusst ein Hochpassfilter, der sämtliche Frequenzen unterhalb von 500Hz abschneidet, die Sprachverständlichkeit lediglich um 5%[7]. Umgekehrt verschlechtert ein Tiefpassfilter, der bei 2kHz ansetzt, die Verständlichkeit um ganze 40%.
Dies belegt abermals die Bedeutung, die der Frequenzbereich von 2-4kHz für den Menschen hat. Oberhalb von 3,5 kHz fällt das Stimmspektrum - unabhängig von der Stimmlage - mit etwa 25dB pro Oktave stark ab[8].

2.1.5 Das menschliche Lautstärkeempfinden

Darüber hinaus ist das menschliche Ohr im Bereich von 2-4kHz des Spektrums am empfindlichsten, was an den sogenannten Kurven gleicher Lautstärke abzulesen ist[9]. Wenn ein Ton mit 200Hz und ein Ton mit 3kHz bei demselben Schalldruckpegel wiedergegeben werden, so nimmt der Mensch den 200Hz-Ton als leiser wahr als den 3kHz-Ton. Zwar nähert sich das menschliche Lautstärkeempfinden der beiden Töne bei steigendem Pegel an, allerdings sind die Pegel beim Telefonieren, worum es in dieser Arbeit primär gehen soll, natürlich deutlich geringer als beispielsweise bei der Beschallung eines Stadions.

Im Folgenden werden im Zuge eines ersten Tests die Frequenzgänge der Stimme des Autors bei unterschiedlichen Vokalen und verschiedenen Übertragungswegen genauer betrachtet und auf die soeben erläuterten Eigenschaften untersucht. Dabei soll ein besonderes Augenmerk auf der Wiedergabe der Frequenzgänge bei den unterschiedlichen Arten des Telefonierens liegen.

[7]DPA 2021.
[8]Weinzierl und Tonmeister 2008, S. 141.
[9]Friedrich 2008, S. 34.

2.2 Eigenschaften von Telefonieren mittels Mobilfunk, FaceTime und Zoom

2.2.1 Übersicht über Mobilfunkstandards

Nach der Abschaffung von 3G in Deutschland im Jahr 2021 werden Mobilfunktelefonate inzwischen mittels der 4G- (auch als LTE bekannt) und 5G-Technologie durchgeführt. Während ältere Generationen von 3G noch auf einer Übertragung mittels des Audioformats AMR-NB (NB für Narrow Band, auf Deutsch: Schmalband) basierten, das lediglich einen Audio-Frequenzbereich von 300Hz-3400Hz verarbeiten konnte, wartete das Nachfolge-Audioformat AMR-WB (WB für Wideband, auf Deutsch: Breitband) bereits mit einem Spektrum von 50Hz bis 7kHz auf[10].
Auf dieses AMR-WB Format greift auch das VoLTE-Verfahren zurück, das für die meisten 4G/LTE-Anrufe verwendet wird. Das VoLTE-Verfahren wird auch HD-Telefonie genannt. Somit wurde durch den Wechsel von 3G auf 4G zunächst keine zusätzliche Bandbreite im Audio-Frequenzbereich erzielt, sondern lediglich eine schnellere und stabilere Anrufverbindung[11].
Das neue Verfahren EVS (Enhanced Voice Services, auch als FullHD-Telefonie bezeichnet), das AMR-WB erweitern soll, wurde für Übertragung per 4G entwickelt. Es basiert ebenfalls auf AMR-WB, synthetisiert aber durch Duplikation und Verschiebung des AMR-WB Signals eine weitere Oktave, sodass der übertragene Frequenzbereich bis zu 14kHz erreichen kann[12].
Bei 5G, das - nicht zuletzt wegen Sicherheitsbedenken gegenüber dem chinesischen Lieferanten Huawei - in den Medien vertreten war, variiert die übertragene Audio-Bandbreite in Abhängigkeit vom Netzanbieter. Seine Vorteile liegen aber ähnlich wie beim Wechsel von 3G nach 4G in verbesserten Datenraten und höherer Geschwindigkeit.

2.2.2 Übertragung per FaceTime und Zoom

Während sich die unterschiedlichen Audioformate im Mobilfunk gut recherchieren lassen, sind Informationen über die Frequenzgänge der Anbieter Apple (FaceTime)[13] und Zoom deutlich schwieriger zu finden. Da beide Anbieter von bestimmten Backgroundnoise- oder Echo-Cancellation-Algorithmen Gebrauch machen, die ein störungsfreieres Telefonieren er-

[10] Strahlenschutz 2023.
[11] Alby 2008.
[12] Heuberger und Grill 2017.
[13] Apple 2022.

möglichen sollen, variieren die Frequenzgänge darüber hinaus je nach Benutzereinstellung. Des Weiteren verhinderten diese Algorithmen den Versuch des Autors, den Frequenzgang mittels eines Sinussweeps zu messen.
Fakt ist jedoch, dass beide Übertragungswege den Open-Source Codec OPUS nutzen[14]. Dieser Codec ist besonders effektiv darin, Audio für die Übertragung zu komprimieren und nach dem Empfangen zu dekomprimieren. Des Weiteren passt er sich der verfügbaren Internetverbindung an, weshalb der Frequenzgang auch stark vom Internetzugang des jeweiligen Nutzers abhängt.
Was das für den konkreten Fall dieser Arbeit bedeutet, soll in dem folgenden ersten Versuch erläutert werden.

2.2.3 Versuchsaufbau

In einem Testsetup im eigenen kleinen Heimstudio werden zunächst ein paar Messungen der Stimme des Autors durchgeführt, um zu prüfen, wie stark die unterschiedlichen Übertragungswege das Frequenzspektrum beeinflussen. Hierfür wird die Stimme, während der Buchstabe „A“ gesprochen wird, mit einem WarmAudio WA87 und über Mobilfunk mittels zweier Apple iPhones auf separaten Spuren aufgezeichnet. Dabei wird direkt das Output Signals des empfangenden iPhones mittels eines Klinkensteckers aufgezeichnet und nicht das Signal des Handylautsprechers. Da die kleinen Handylautsprecher selbst gewisse akustische Eigenschaften mit sich bringen, wird sich der Frequenzgang, den man als Handynutzer am Empfangsgerät zu hören bekommen wird, nochmals von der Messung unterscheiden.
Des Weiteren ist zu beachten, dass auch das Studiomikrofon gewisse klangtechnische Eigenheiten besitzt und im Frequenzgang nicht komplett linear ist. Wenngleich zwar Messmikrofone in ihrem Frequenzverhalten nochmals eine Spur linearer sind, so gilt zu beachten, dass bei Sprachaufnahmen eine leichte Färbung des Klangs sogar gewünscht und als angenehm empfunden wird, da ein komplett linearer Klang vom Hörer als unnatürlich wahrgenommen wird.
Das iPhone und das Studiomikrofon werden hierfür im Abstand von 15cm zum Mund des Sprechers aufgestellt. Die Signale werden anschließend durch die Real-Time-Analysis von Room EQ Wizard geleitet und jeweils geplottet. Hierdurch erhält man folgendes Ergebnis (siehe Abbildung 2.2):

[14]Hancke 2021.

2.2.4 Allgemeine Beobachtungen

Zunächst einmal lässt sich, unabhängig von den Frequenzspektren der jeweiligen Aufzeichnung, ein klarer Grundton mit den jeweiligen Obertönen erkennen (siehe Kapitel 2.1.1). Der Grundton liegt hier bei 95Hz, alle weiteren Töne bei Vielfachen des Grundtons: Die erste Oktave liegt bei ca 190Hz, die Quinte darüber wiederum zwischen 280Hz und 290Hz, die Quarte darüber bei ca 380Hz. Bei dem Signalanteil unterhalb des Grundtons handelt es sich nicht um Nutzsignal, das von der Stimme erzeugt wird, sondern um Resonanzen im Raum oder sonstige Geräusche. Bei 700 und 800Hz ist eine Verstärkung der Obertöne zu beobachten, die auf den ersten Vokalformanten F1 hinweisen, wenngleich 700Hz etwas tiefer ist als gewöhnlich, was auf die tiefe Stimme des Autors zurückzuführen ist. Des Weiteren ist eine Anhebung des Frequenzgangs im Bereich des Sängerformanten zwischen 2,2 und 3,6 kHz zu erkennen.
Nun gilt es zu beobachten, wie die unterschiedlichen Übertragungswege diesen Frequenzgang verändern und färben.

Abbildung 2.2: Frequenzspektren bei Aussprache des Buchstaben „A“ mit Grundton 95Hz, aufgezeichnet mit einem Studiomikrofon (Rot) und über Handy mit Mobilfunk (Orange)

Quelle: Eigene Darstellung

2.2.5 Übertragung mittels Mobilfunk

Direkt deutlich zu erkennen ist, dass bei der Übertragung durch Mobilfunk vor allem der Grundton stark beschnitten wird (ca. 20dB weniger) und dass das Signal beim Telefon ab ca. 7 kHz beschnitten wird, da hier keine periodischen Schwingungen mehr zu erkennen sind. Dies deckt sich mit der Bandbreite, die mittels VoLTE übertragen wird (siehe Kapitel 2.2.1). Ob der Pegelunterschied beim Grundton durch einen Hochpassfilter mit geringer Güte bei 50Hz, durch die Übertragung oder durch den Qualitätsunterschied zwischen Studio- und Handymikrofon entsteht, lässt sich hier nicht direkt bestimmen.
Abgesehen vom Grundton vermag der Frequenzgang der Mobilfunkübertragung im Bereich 200-700Hz den des Studiomikrofons recht gut abzubilden. Ab 700Hz aufwärts gibt es immer wieder Differenzen von ca 5dB (z.B. bei 1,3kHz oder 3 kHz) und vereinzelt größere Differenzen von ca. 10dB bei 4kHz und 10kHz.
Der Sängerformant wird bei 2,4kHz um ca. 7dB verstärkt. Da das Telefonieren allem voran die Sprachverständlichkeit gewährleisten muss, ergibt diese leichte Anhebung auch Sinn, da dieser Frequenzbereich besonders stark für Verständlichkeit der Stimme sorgt (siehe Kapitel 2.1.4).

2.2.6 Übertragung mittels Videotelefonie

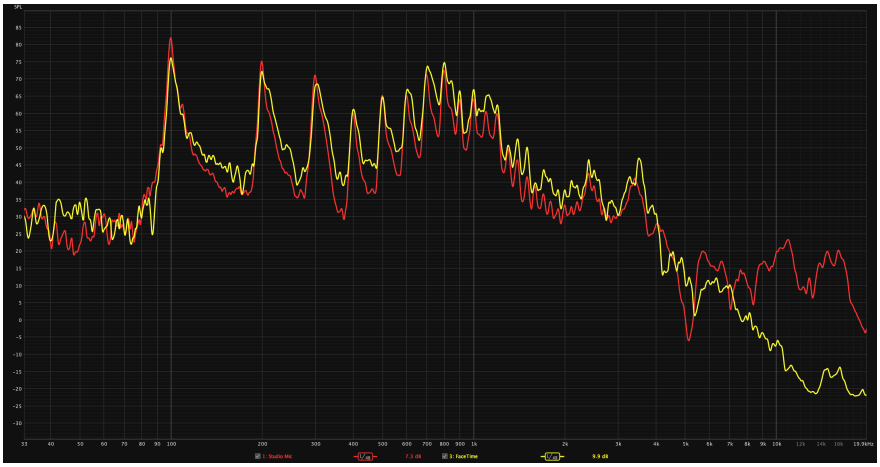

Abbildung 2.3: Frequenzspektren bei Aussprache des Buchstaben „A“ mit Grundton 95Hz, aufgezeichnet mit einem Studiomikrofon (Rot) und über FaceTime (Gelb)
Quelle: Eigene Darstellung

Bei den Frequenzgängen der Übertragung per Zoom (siehe Abb. 2.4) und per FaceTime

Abbildung 2.4: Frequenzspektren bei Aussprache des Buchstaben „A“ mit Grundton 95Hz, aufgezeichnet mit einem Studiomikrofon (Rot) und über Zoom (Grün)
Quelle: Eigene Darstellung

(siehe Abb. 2.3) sind die Unterschiede zum Studiomikrofon geringer als bei der normalen Telefonie. Die deutlichste Veränderung besteht darin, dass es den Videocall Plattformen gelingt, den Grundton bei 100 Hz besser zu übertragen (bei FaceTime ca. -5dB im Vergleich zum Studio Mikrofon).
Des Weiteren ist klar erkennbar, dass die beiden Signale ab ca. 7kHz das Originalsignal nicht mehr abbilden (ähnlich zur Übertragung im Mobilfunk, siehe Kapitel 2.2.5).
Im direkten Vergleich schneidet Zoom beim Abbilden des Referenzfrequenzgangs besser ab als FaceTime. Vor allem im Bereich bis 800Hz liefert Zoom nähere Ergebnisse am Studiomikrofon.
Bei beiden Anbietern ist jedoch eine Verringerung der Dynamik (va. im tieffrequenten Bereich) festzustellen, die sich durch eine Anhebung der Frequenzbereiche zwischen Grundton und den ersten drei weiteren Obertönen bemerkbar macht.
Des Weiteren wird bei FaceTime der Bereich um den Sängerformanten etwas angehoben und liegt ca. 2dB über der Studiomikrofonreferenz.

2.3 Zwischenfazit des ersten Versuchs

Anhand dieses ersten simplen Versuchs lässt sich feststellen, dass die Audioqualität von Videokommunikationsdiensten (va. Zoom) näher an den Frequenzgang eines Studiomikrofons herankommt als die von Telefonie über Mobilfunk. Bei sämtlichen Signalen wird das Spektrum bei ca. 7kHz abgeschnitten. Dagegen wird der Grundton über alle drei Wege über-

tragen, wenngleich er beim Telefonieren deutlich leiser ausfällt.
Ob es möglich ist, den Grundton beim Telefonieren mittels Machine Learning wieder entsprechend zu verstärken und das fehlende Spektrum ab 7kHz auf Grundlage des Grundtons und der Obertonreihe (siehe Kapitel 2.1.1) zu ergänzen, soll im Zuge dieser Arbeit herausgefunden werden. Hierfür werden zunächst die Grundlagen von Machine Learning erläutert.

3 Machine Learning Algorithmen

3.1 Das Potenzial und die Probleme von Künstlicher Intelligenz

3.1.1 KI im Alltag

Auch wenn das Thema „Künstliche Intelligenz“ durch die Veröffentlichung des Chatbots ChatGPT des amerikanischen Unternehmens OpenAI Anfang des Jahres 2023 medial stark in den Fokus gerückt ist[1], so fällt bei genauerem Hinschauen auf, dass Algorithmen, die auf Künstlicher Intelligenz basieren, bereits seit geraumer Zeit in vielen Alltagssituationen vorkommen.

So stecken diese beispielsweise hinter den Videos, Beiträgen oder Produkten, die auf den Startseiten von Instagram, YouTube, Netflix, Amazon etc. vorgeschlagen werden. Algorithmen, die im Hintergrund arbeiten, analysieren ununterbrochen das Nutzerverhalten und versuchen auf dieser Grundlage die nächsten Schritte des Users vorherzusagen, bzw. aktiv vorzuschlagen, was als Nächstes von Interesse sein könnte. Hinter dem Spamfilter im Email-Postfach verbrigt sich ebenso ein KI-Algorithmus wie hinter der FaceScan-Entsperrung beim Smartphone. Auch Sprachassistenztools wie etwa Siri, Alexa oder Cortana würden ohne KI nicht funktionieren[2].

In bestimmten Wirtschaftszweigen werden ebenfalls bereits zahlreiche Prozesse mittels KI erledigt: So überprüfen beispielsweise Banken Transaktionen der Kunden mittels KI-Algorithmen auf untypische Verhaltensweisen. In der Medizin sind Ärzte inzwischen in der Lage, KI-Werkzeuge beim Untersuchen von Röntgenbildern oder CT-Bildern ihrer Patienten zu Rate zu ziehen[3]. Selbstfahrende Autos wären ohne den Einsatz von KI-Algorithmen, die ununterbrochen von Sensoren erfasste Umgebungsdaten auswerten, undenkbar[4].

[1] Jahn 2023.
[2] EU-Parlament 2020.
[3] Helmholtz 2022.
[4] Guillot 2020.

Auch vor der Medienbranche hat diese Entwicklung nicht Halt gemacht: In Adobe Premiere Pro kann man sich mittels KI automatisch Untertitel generieren lassen, in Lightroom[5] oder DaVinci Resolve[6] gibt es Tools, die selbständig gewisse Bildelemente wie etwa Personen oder den Himmel erkennen und maskieren. Im Audiobereich erfreuen sich unter anderem die KI-basierten Plugins der Firma iZotope großer Beliebtheit, die gerne zum Mastering von modernen Musikproduktionen verwendet werden[7].

3.1.2 Übersicht über die Nachteile von KI-Algorithmen

KI-Algorithmen bringen eine Vielzahl von positiven Veränderungen für die Gesellschaft mit sich. Jedoch birgt Künstliche Intelligenz auch einige Risiken. Damit sind nicht jene dystopischen Vorstellungen gemeint, die man aus Hollywood Filmen oder Science Fiction Literatur kennt, in denen KI-gesteuerte Roboter die gesamte Weltherrschaft übernehmen. Allerdings gibt es bereits heutzutage zahlreiche Probleme, die durch KI verursacht werden: Durch gefälschte Fotos oder Videos (so genannte Deepfakes), die über das Internet verbreitet werden, können Teile der oder gar die gesamte Gesellschaft manipuliert und polarisiert werden, sowie die betroffene Person geschädigt werden. Im vergangenen Jahr 2022 sorgte ein „gefälschter“ Vitali Klitschko durch Videotelefonate mit Bürgermeistern europäischer Großstädte wie etwa Franziska Giffey in Berlin oder José Luis Martinez-Almeida in Madrid für Aufsehen[8].
Ein weiteres Problem ist, dass ein KI-Algorithmus nur so gut ist wie die Daten, mit denen er trainiert wird bzw. im Falle von unüberwachtem Training (siehe Kapitel 3.2.6) selbständig trainiert. Bereits 2016 veröffentlichte Microsoft den Twitter-Chatbot „Tay“, der zunächst in jugendlicher Sprache mit anderen Nutzern interagierte und aus diesen Konversationen lernte. Jedoch konnte der Bot nicht zwischen freundlichen und rassistischen Konversationen unterscheiden, sodass er innerhalb lediglich eines Tages twitterte, dass Hitler Recht gehabt hätte und dass er Juden und Feministen hassen würde, woraufhin er von Microsoft wieder deaktiviert wurde[9].
Auch 7 Jahre später irritiert der bereits angesprochene Chatbot Chat GPT gelegentlich mit Falschaussagen und problematischen Texten, in denen er beispielsweise in sich schlüssige Argumentationen schreibt, wonach die Amerikaner durch Vapen das Covid 19 Virus freigesetzt

[5] Adobe 2023.
[6] Design 2023.
[7] iZotope 2023.
[8] Redaktion 2022.
[9] Sickert 2016.

hätten, wobei er sich jedoch auf komplett erfundene Studien beruft[10]. Auch die aktuellste Version GPT-4 ist nach Aussage des CEOs Sam Altmann immer noch „flawed“ (fehlerhaft) und „limited“ (limitiert)[11] - immerhin wird der Nutzer des Bots aber vorher explizit auf mögliche Falschmeldungen hingewiesen.
Weitere Probleme mit KI sind beispielsweise:

- Schuldfrage nach Versagen von KI, z.B. bei von Autopiloten verursachten Autounfällen[12]
- Diskriminierende Algorithmen aufgrund problematischer Trainingsdaten, z.B. Benachteiligung von Menschen mit dunkler Hautfarbe bei KI zur Erkennung von Hautkrebs aufgrund von Unterrepräsentation im Trainingsdatensatz[13]
- Umstrukturierung bestimmter Branchen und damit einhergehend Wegfall von Berufen[14]
- Probleme durch Social Media Algorithmen: Polarisierung bis hin zu Radikalisierung, Depressionen (v.a. bei jüngeren Nutzer:innen)[15]

3.1.3 Der Wirtschaftliche Einfluss von KI

Trotz der genannten Nachteile bleibt Künstliche Intelligenz auf dem Vormarsch. Der zum Zeitpunkt dieser Arbeit amtierende Wirtschaftsminister Deutschlands, Robert Habeck, bezeichnete jüngst in einer Pressemitteilung Künstliche Intelligenz als „eine der wesentlichen digitalen Zukunfts- und Schlüsseltechnologien“[16].
Die Prognosen, wie Künstliche Intelligenz den Markt verändern wird, liegen zuweilen allerdings noch recht weit auseinander: McKinsey schätzte 2018 einen Anstieg der weltweiten Wirtschaftskraft um 13 Billionen US-Dollar bis 2030 alleine durch KI[17], Pricewaterhouse-Coopers bezifferte für denselben Zeitraum gar 15,7 Billionen US-Dollar[18], Analysis Group Inc. hingegen berechnete 2016 einen moderateren Anstieg zwischen 1.49 und 2.95 Billionen US-Dollar[19]. Unabhängig davon, welche der Berechnungen am Ende Recht behalten soll,

[10] Oerding und AP 2023.
[11] Sam Altman [@sama] 2023.
[12] Chokshi 2020.
[13] Feuerriegel, Dolata und Schwabe 2020.
[14] Guillot 2020.
[15] Primack u. a. 2017.
[16] Klimaschutz 2022.
[17] McKinsey o. D.
[18] Gillham u. a. 2018.
[19] Chen u. a. 2016.

kann man eines mit Sicherheit sagen: Der Bedarf an KI-Algorithmen wird weiter wachsen - genauso wie die Fähigkeit der Netzwerke.
Im Folgenden sollen für den Nutzen innerhalb dieser Bachelorarbeit die wichtigsten Begrifflichkeiten rund um das Thema Künstliche Intelligenz erläutert werden.

3.2 Definitionen

3.2.1 Allgemeines

„KI“, „Machine Learning“, „Deep Learning“ oder „Neuronale Netzwerke“ sind allesamt Begrifflichkeiten, die im Kontext von Algorithmen, die mit künstlicher Intelligenz arbeiten, immer wieder verwendet werden. Um einen besseren Überblick über das Thema zu erlangen, ist es essenziell, sich mit den genauen Bezeichnungen auseinanderzusetzen und diese zu definieren.
Bei **Künstlicher Intelligenz** (kurz: KI; oder englisch: AI) handelt es sich um ein Teilgebiet der Informatik, in Zuge dessen versucht wird, menschliches, kognitives Verhalten nachzuahmen. Somit stellt Künstliche Intelligenz den Überbegriff zu programmierten oder maschinell erlernten Abläufen dar[20].
Bei Maschinellem Lernen (oder englisch: **Machine Learning**) versucht der Computer, Muster und Zusammenhänge in großen Datensätzen zu finden. Am Ende sollte ein Machine-Learning-Algorithmus in der Lage sein, eine nicht im Trainingssatz enthaltene Eingabe auf die von ihm erlernten Muster abzusuchen und eine entsprechende Ausgabe zu liefern[21].

3.2.2 Das „Hello World“ des Machine Learnings

Um dies zu veranschaulichen, soll als Beispiel das MNIST Datenset herangezogen werden, das von vielen als das „Hello World“ des Machine Learnings bezeichnet wird: Dabei handelt es sich um 60.000 Trainings- und 10.000 Evaluierungsbilder, auf denen von Hand geschriebene Zahlen von 0 bis 9 zu sehen sind (siehe Abbildung 3.1).
Das Machine Learning Netzwerk trainiert anschließend mit den 60.000 Trainingsbildern, indem es diese im Zuge mehrerer sog. **Epochen** immer wieder zu Gesicht bekommt und anschließend selbständig Änderungen am eigenen System vornimmt. Am Ende des Trainings sollte es in der Lage sein, ein Bild aus dem Evaluierungsdatensatz der korrekten entsprechenden Zahl zuzuordnen.
Um zu verstehen, wie der Trainingsprozess funktioniert und wie ein Netzwerk selbständig lernen kann Zahlen zu erkennen, muss man einen genaueren Blick auf Neuronale Netze und die Parallelen zum menschlichen Gehirn werfen:

[20] Roscher, Guderitz und Hengl 2023.
[21] SAP 2023.

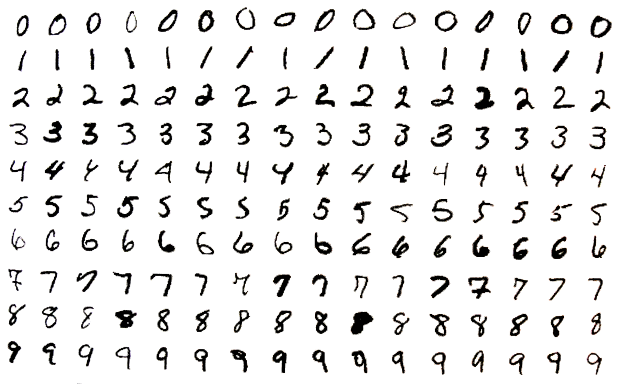

Abbildung 3.1: Ausschnitt aus dem MNIST Datenset
Quelle: Schwaiger und Steinwendner 2019, S. 216

3.2.3 Neuronen im menschlichen Gehirn

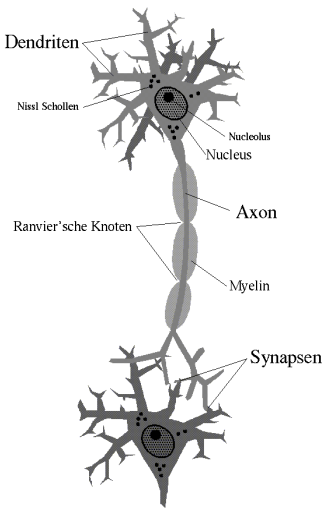

Abbildung 3.2: Aufbau eines Neurons im menschlichen Gehirn
Quelle: Mayer 2023

Neuronale Netze sind dem menschlichen Gehirn nachempfunden, wo insgesamt ca 80-100 Milliarden Nervenzellen (Neuronen) für die Weitergabe und Verarbeitung von Reizen verantwortlich sind. Neuronen können mittels ihrer Dendriten Signale empfangen und mittels des Axons weiterleiten. Bei den Signalen handelt es sich um elektrische Aktionspotenziale von ungefähr 0,1V, die binnen 1-2 Tausendstel einer Sekunde weitergegeben werden[22].
Durch diesen Impuls werden in den Synapsen, den Verbindungen zwischen zwei Nervenzellen, bestimmte Botenstoffe (**Neurotransmitter**) freigesetzt. Die empfangende Nervenzelle, das

[22]Schwaiger und Steinwendner 2019.

postsynaptische Neuron, nimmt mittels Rezeptoren die Neurotransmitter auf, welche in der Folge anregend (exzitatorisch) oder hemmend (inhibitorisch) auf die Membran des postsynaptischen Neurons wirken können. Durch die Änderung im Membranwiderstand kommt es zu einer Spannungsänderung, die anschließend vom empfangenden Neuron weiterverarbeitet wird.
Zusammengefasst bedeutet dies, dass Neuronen im menschlichen Gehirn über mehrere Eingänge verfügen, deren elektrische Impulse sie in Abhängigkeit der freigesetzten Neurotransmitter in den Synapsen mit veränderter Spannung an die anknüpfenden Neuronen weitergeben.

3.2.4 Neuronale Netzwerke am Beispiel eines Fully Connected Networks

Neuronale Netzwerke sind demnach der Versuch, das in Kapitel 3.2.3 beschriebene biologische Prinzip auf einen Computer-Algorithmus zu übertragen. Hierbei gibt es unterschiedliche Netzwerkarchitekturen, von denen sich das Fully Connected Network im Rahmen dieser Arbeit vergleichsweise simpel erklären lässt.
Wie in Abbildung 3.3 erkennbar ist, sind bei einem Fully Connected Netzwerk die Neuronen (blaue Punkte) in Schichten angeordnet, wobei alle Neuronen einer Schicht mit sämtlichen Neuronen der angrenzenden Schicht(en) verbunden sind.
Das Netzwerk in Abbildung 3.3 verfügt über 8 Input- (links) und 4 Output-Neuronen (rechts).
Zunächst muss ein Neuron mit einem Zahlenwert aktiviert werden. Diesen Zahlenwert gibt es anschließend an seine anknüpfenden Neuronen weiter. In modernen Netzwerken können hierbei mehrere Millionen Neuronen untereinander verknüpft sein.
Ähnlich wie im Gehirn, wo die Weitergabe des Aktionspotenzials verstärkt oder gehemmt wird, wird hier jeder Verbindung jeweils eine Gewichtung (**weight**) zugeordnet, wodurch reguliert werden kann, welche Neuronenverbindungen wie stark in die Gesamtausgabe einfließen.
Zudem können die Neuronen selbst ebenfalls durch den **bias** Wert stärker oder schwächer gewichtet werden.
Gibt man nun einen Input in ein Neuronales Netzwerk, werden zunächst die Neuronen in der Eingabeschicht aktiviert. Anschließend werden diese Werte durch weights und biases unterschiedlich gewichtet an eine nächste Schicht weitergegeben. Diesen Prozess bezeichnet

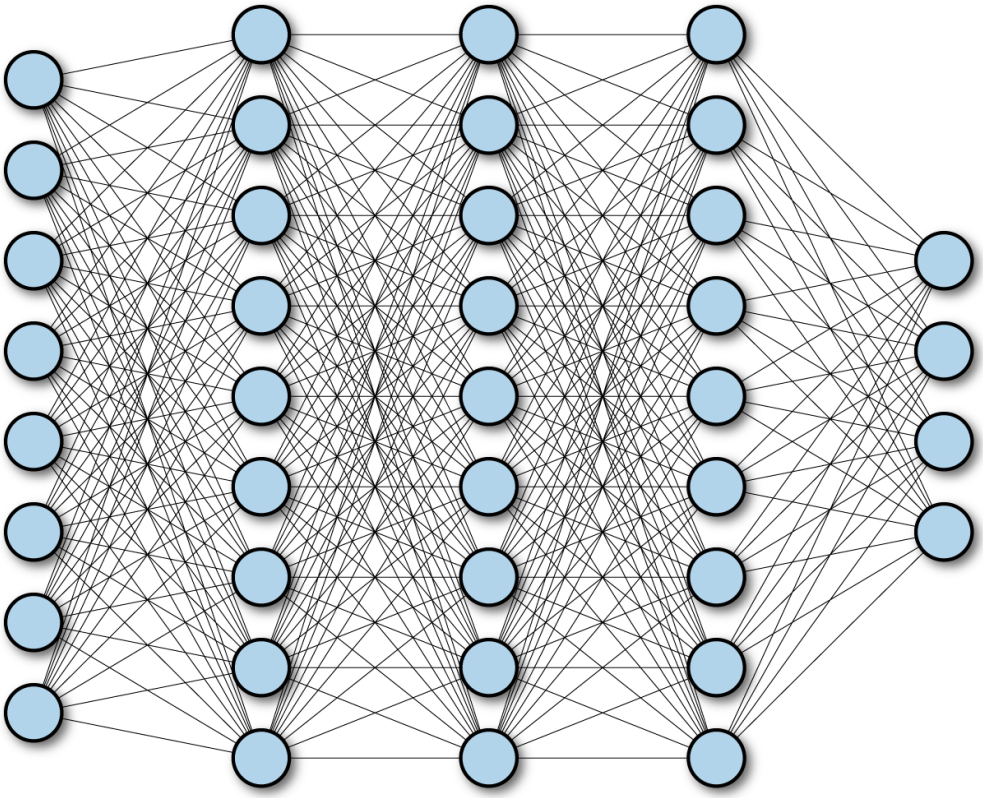

Abbildung 3.3: Beispiel für ein Fully Connected Neuronales Netzwerk
Quelle: Ramsundar und Zadeh 2018

man als **Forward Propagation**, da die Werte von Schicht zu Schicht (in diesem Fall von links nach rechts) weitergereicht werden **(Feedforward**, dt.: weiterreichen).
Sobald ein Netzwerk über mehrere solcher Schichten (auch **Hidden Layers** genannt) verfügt und mindestens eine Schicht besitzt, die nicht direkt mit Input oder Output verknüpft ist, spricht man von einem **Deep Neural Network**[23].
Mittels Feedforward werden nach und nach sämtliche Neuronen in den Schichten aktiviert. Die einzelnen Aktivierungswerte werden hierbei wie folgt berechnet:

Um den Aktivierungswert eines Neurons zu ermitteln, werden zunächst mittels der Summenfunktion

$$v = b + \sum_{i=1}^{n} x_i \cdot w_i$$

alle Eingangswerte - multipliziert mit ihren zugehörigen Gewichtungen - aufaddiert, ehe der

[23]Schwaiger und Steinwendner 2019, S. 40.

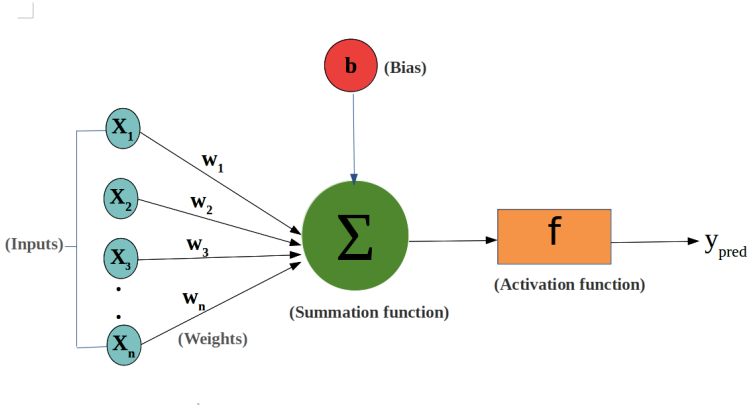

Abbildung 3.4: Alle Neuronen der vorangehenden Schicht x geben ihre Werte mit weights w gewichtet weiter. Die Summe wird mit dem Bias b verrechnet.
Quelle: Ganesh 2022

Bias hinzugefügt wird.
Hierbei steht x_i für den Wert des vorangeganenen Neurons, w_i für das zugehörige weight der Verbindung, b für den bias des Neurons und n für die Anzahl der Eingabeneuronen. Dabei haben x und w für jedes Eingabeneuron (i in n) unterschiedliche Werte, während der bias b konstant bleibt[24].
Die so erzeugte Summe wird anschließend in eine Aktivierungsfunktion gegeben (siehe Kapitel 3.2.5), die den Aktivierungswert des Neurons ermittelt.

3.2.5 Aktivierungsfunktionen

Würde man dem Neuron bereits den in Kapitel 3.2.4 ermittelten Summenwert als Aktivierungswert zuordnen, so wäre dieser lediglich auf Basis linearer Funktionen entstanden. Auf das gesamte Netzwerk übertragen würde dies bedeuten, dass Input und Output in linearem Zusammenhang stehen, was nicht dazu geeignet ist, komplexe Aufgabenstellungen, bei denen Ein- und Ausgabe in nicht linearem Verhältnis stehen, zu bewältigen. Um diese Nichtlinearität darstellen zu können, wird die berechnete Summe in eine **Aktivierungsfunktion** gegeben, welche die Werte auf nicht lineare Art beeinflusst. Bei der Wahl der Aktivierungsfunktion kann man auf diverse Optionen zurückgreifen. Eine einfache Aktivierungsfunktion

[24]Sibi, Jones und Siddarth 2013.

ist die Sigmoid-Funktion[25]:

$$\sigma(x) = \frac{1}{(1 + e^{-x})}$$

Diese transformiert den Eingabewert in einen Bereich zwischen 0 und 1, was gerade bei Klassifizierungsaufgaben (Beispiel: Ein Algorithmus ordnet Bilder des MNIST Datensets den entsprechenden Kategorien 0-9 zu; mehr dazu in Kapitel 3.2.6) von Vorteil ist.
Des Weiteren gibt es für Klassifizierungsaufgaben eine Abwandlung der Sigmoid-Funktion: die **Softmax** Funktion[26].

$$\sigma(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} for j = 1, ..., K$$

Hier wird jeder Eingabewert x durch die Summe der Eingabewerte sämtlicher Neuronen mit der Anzahl K in der eigenen Schicht (Layer) geteilt und somit in einen Bereich zwischen 0 und 1 transformiert und normiert, sodass sich sämtliche Aktivierungswerte des Layers zu 1 ergänzen.
Eine weitere Abwandlung der Sigmoid Funktion stellt die TANH-Funktion dar, die den Input auf einen Bereich zwischen -1 und 1 mappt, wobei sie durch den Nullpunkt verläuft, der zeitgleich den Symmetriepunkt darstellt. Das Verhältnis der TANH-Funktion gegenüber der Sigmoid-Funktion kann wie folgt beschrieben werden[27].:

$$f(x) = 2\sigma(2x) - 1$$
$$f(x) = \frac{2}{(1 + e^{-2x})} - 1$$

Jedoch haben die bisher vorgestellten Aktivierungsfunktionen allesamt einen entscheidenden Nachteil: Die Berechnung ist relativ betrachtet aufwändig, was vor allem bei größeren Netzwerken mit mehreren Millionen Neuronen stark ins Gewicht fällt. Aus diesem Grund erfreut sich die **ReLU** (Rectified Linear Unit) in jüngster Vergangenheit großer Beliebtheit. Sie vereint die Berechnungseffizienz von linearen Funktionen mit einer einfachen Nichtlinearität.

[25]Sibi, Jones und Siddarth 2013.
[26]Sharma, Sharma und Athaiya 2020.
[27]Sharma, Sharma und Athaiya 2020.

Sie kann wie folgt beschrieben werden[28]:

$$f(x) = max(0, x)$$

Für sämtliche Eingabewerte unterhalb von 0 ist das Neuron inaktiv, für alle Werte oberhalb von 0 entspricht der Aktivierungswert dem Input. So wird Nichtlinearität gewährleistet bei gleichzeitig hoher Effizienz. Sollten dennoch negative Aktivierungswerte gewünscht sein, kann man sich mit der **Leaky ReLU** Function[29]

$$f(x) = max(\alpha * x, x) mit(0 < \alpha < 1),$$

oder der **ELU** Function (Exponential Linear Unit) behelfen[30]:

$$0 < \alpha$$
$$x = \begin{cases} x & \text{falls } x > 0 \\ \alpha(exp(x)1) & \text{falls } x \leq 0 \end{cases}$$

Die zugehörigen Graphen sowie eine weitere Abwandlungen der ReLU Funktion, einer Shifted ReLU Funktion ($f(x) = max(-1, x)$) sind in Abbildung 3.5 dargestellt. Zusammengefasst berechnet sich in einem Fully Connected Netzwerk der Aktivierungswert eines Neurons aus der Aktivierungsfunktion eines Bias und der Summe aller Werte der vorangehenden Schicht multipliziert mit den zugehörigen weights.

3.2.6 Lernmethoden für Algorithmen

Die Besonderheit von Machine-Learning-Algorithmen besteht darin, dass sie selbständig dazu in der Lage sind, Anpassungen an ihrem eigenen System vorzunehmen, um in der Folge entsprechende Aufgaben lösen zu können.
Diese Anpassungen finden im Zuge eines Trainingsprozesses statt, bei dem das Netzwerk bestimmte Muster und Zusammenhänge in einem großen Datensatz (wie etwa dem MNIST Datensatz, siehe Kapitel 3.2.2) erlernt.
Wie genau der Output des Netzwerks anschließend aussieht, hängt von der Lernmethode des

[28] Clevert, Unterthiner und Hochreiter 2016.
[29] Clevert, Unterthiner und Hochreiter 2016.
[30] Sharma, Sharma und Athaiya 2020.

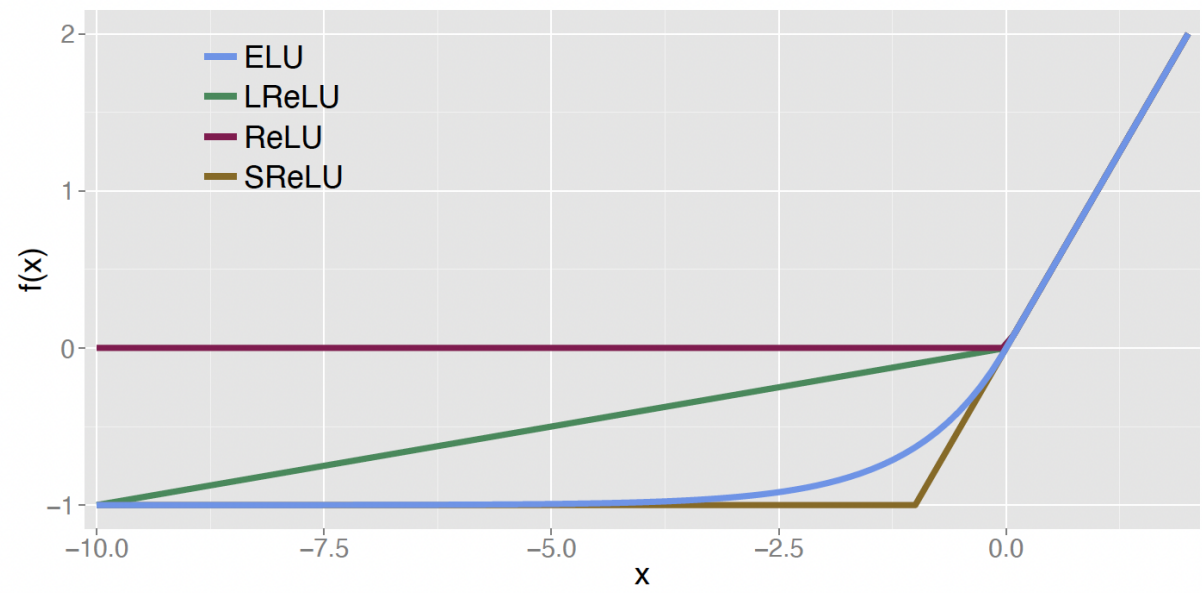

Abbildung 3.5: Plots der ReLU Funktion, der Leaky ReLU Funktion (LReLU, $\alpha = 0{:}1$), der Shifted ReLU Funktion (SReLUs), und der ELU Funktion ($\alpha = 1{,}0$).
Clevert, Unterthiner und Hochreiter 2016

Algorithmus ab[31]:

- beaufsichtigtes Lernen (**supervised learning**)
- unbeaufsichtigtes Lernen (**unsupervised learning**)
- eine Zwischenform aus super- und unsupervised learning (**semisupervised learning**)
- verstärkendes Lernen (**reinforcement learning**)

Supervised Learning

Im Falle des **Supervised Learnings** wird dem Algorithmus im Zuge des Trainingsprozesses das Lernziel vom Nutzer mitgegeben. Hierfür wird ein gelabeltes Datenset benötigt, wobei die Labels die so genannte **Ground Truth** beinhalten. Das bedeutet, dass der Algorithmus zu sämtlichen Daten das erwünschte Lernziel erhält. Während des Lernens durchläuft der Algorithmus anschließend mehrere sogenannte **Epochen**[32].

[31] SAP 2023.
[32] Paass und Hecker 2020, S. 45 f.

Eine Epoche definiert sich dadurch, dass sämtliche Trainingsdaten einmal das Netzwerk durchlaufen haben und mit den Labels abgeglichen wurden. Auf dieser Grundlage nimmt der Algorithmus Änderungen an den eigenen Parametern vor, um sich dem vom Nutzer vorgegebenen Label anzunähern.
Zur Veranschaulichung kann abermals das MNIST-Datenset (siehe Kapitel 3.2.2) herangezogen werden: Die Neuronen im Input-Layer des Netzwerks werden zunächst mit den Werten der einzelnen Bildpixel aktiviert. Anschließend reichen die Neuronen ihre Werte an die nächste Schicht weiter (Forward Propagation), bis schließlich sämtliche Neuronen in der letzten Schicht, der Ausgabeschicht, aktiviert wurden. Die Aktivierungswerte dieser Schicht werden im nächsten Schritt mit dem Label abgeglichen, um zu prüfen, wie weit die Vorhersage des Netzwerks (**Prediction**) mit der Ground Truth übereinstimmt[33].
Anschließend wird der Algorithmus rückwärts durch das Netzwerk schreiten und die weights und biases so anpassen, dass die Differenz zwischen Prediction und Label im nächsten Durchlauf geringer sein sollte. Diesen Prozess bezeichnet man als **Backpropagation** (mehr dazu in Kapitel 3.2.8).
Auf diese Weise erlernt es mittels des Datensatzes, dieses eine spezifische Problem zu lösen. Bei einer solchen Aufgabe, bei der ein Algorithmus Daten einer entsprechenden Kategorie zuordnet, in diesem Fall den Kategorien 0 bis 9, handelt es sich um eine sogenannte **Klassifizierungsaufgabe**. Gibt man nun diesem Netzwerk ein Bild von einer handgeschriebenen Zahl, das es zuvor noch nicht gesehen hat, wird es eine Prediction abgeben. Diese Prediction beinhaltet die Aktivierungswerte des Output Layers, die sich dank der Softmax Funktion (siehe Kapitel 3.2.5) zu 1 aufaddieren, weshalb sie häufig als Wahrscheinlichkeitsverteilung interpretiert werden. Aus dieser Verteilung lässt sich die **Confidence** des Netzwerks ablesen[34].
Das Netzwerk könnte beispielsweise angeben, dass es sich bei dem ihm vorgelegten Bild zu einem Wert von 0,8 um eine 5 handelt, zu 0,15 um eine 6 und zu 0,05 um eine 2, da es gewisse Muster gefunden hat, die einer 5, 6 oder 2 zuzuordnen sein könnten. Entscheidend ist hierbei der höchste Wert. Das Netzwerk wäre sich bei diesem Beispiel also zu 0,8 (80%) sicher, dass es sich bei dem Bild um eine 5 handelt.
Das Netzwerk, das mit dem MNIST Datenset trainiert wurde, wird jedoch auch bei Bildern von Katzen oder anderen Objekten eine Wahrscheinlichkeitsverteilung ausgeben, die sich zu 1 ergänzt. Somit lohnt es sich, einen näheren Blick auf die Confidence zu werfen, da ein Netzwerk, das mit anderen Daten trainiert wurde, höchstwahrscheinlich nicht sehr viel Vertrauen

[33]Schwaiger und Steinwendner 2019, S. 314 f.
[34]Schwaiger und Steinwendner 2019, S. 315.

in seine Prediction haben wird, sondern die Aktivierungswerte breiter über die möglichen Kategorien verteilen wird
Ein weiterer Aufgabentyp neben Klassifizierung, den man mittels Supervised Learnings trainieren kann, sind **Regressionsaufgaben**, die primär für Prognosen verwendet werden. Hierbei gibt der Algorithmus einen einzelnen Zahlenwert aus. In Abhängigkeit von den Trainingsdaten kann es sich hierbei beispielsweise um eine Umsatzvorhersage für das kommende Jahr oder um eine Stromverbrauchs- oder Wetterprognose handeln[35].
Regressions- und Klassifizierungsaufgaben sind nur mittels Supervised Learnings umsetzbar, bei dem sämtlichen Trainingsdaten ein Label mit der Ground Truth beigefügt ist.

Unsupervised Learning

Wie der Name bereits verrät handelt es sich beim **Unsupervised Learning** um das Gegenteil zum Supervised Learning: Hier gibt man dem Algorithmus keine Labels mit in den Trainingsprozess. Beim Unsupervised Learning soll dieser selbständig bestimmte Muster und wiederkehrende Werte erkennen und diese entsprechend zusammenfassen, ohne dass der Nutzer ein Lernziel vorgibt[36].
Dieses Zusammenfassen wird auch als **Clustering** bezeichnet. Cluster-Analyse spielt vor allem bei Empfehlungssystemen, wie man sie von Google, Online Shops oder Social Media (Instagram, TikTok) kennt, eine entscheidende Rolle. Mittels Clustern lassen sich Nutzer mit ähnlichem Kauf- oder Browsingverhalten erkennen und zusammenfassen. In der Folge bekommt man das Video oder Produkt vorgeschlagen, für das sich andere User aus demselben Segment zuvor interessierten[37].
Des Weiteren kommt Clustering zum Tragen, wenn das Smartphone sämtliche Bilder anzeigt, auf denen eine gewisse Person zu sehen ist. Der Algorithmus kennt zwar den genauen Namen der Person nicht, erkennt aber beim Scannen der Fotogalerie wiederkehrende Muster (in diesem Fall z.B. Gesichter) und fasst diese in einer Kategorie zusammen[38]. Im Falle von äußerst umfangreichen Datenmengen (Big Data) dient Unsupervised Learning auch der **Dimensionsreduktion**, um die Daten greif- und visualisierbar zu machen.

[35] Paass und Hecker 2020, S. 46.
[36] SAP 2023.
[37] EU-Parlament 2020.
[38] Roscher, Guderitz und Hengl 2023.

Semisupervised Learning

Da in der realen Welt quasi endlos Daten vorhanden sind, es jedoch teuer und aufwändig wäre, große Datensets mit Labeln zum Supervised Learning zu versehen, nimmt das **Semisupervised Learning** den Mittelweg zwischen den anderen beiden Lernmethoden und erlernt einen Teil mittels gelabelter Daten und den anderen durch Unsupervised Learning. Die Einsatzgebiete von Semisupervised Learning sind ähnlich wie beim Unsupervised Learning[39].

Reinforcement Learning

Abbildung 3.6: Screenshot aus dem Computerspiel Atari Breakout
Quelle: Patel u. a. 2019

Um eine besondere Lernmethode handelt es sich beim **Reinforcement Learning**. Hierbei wird dem Algorithmus lediglich ein Ziel innerhalb einer Simulation oder einer Art Spiel vorgegeben. Anschließend lernt das System selbständig, durch Belohnung oder Bestrafung seine Entscheidungsfindung anzupassen[40]. Ein Beispiel hierfür ist ein Algorithmus, der das Spiel Atari Breakout spielte, bei dem es darum geht, Blöcke an der Decke mit einer Kugel abzuschießen, um Punkte zu sammeln, ohne die Kugel zu Boden fallen zu lassen (siehe Abbildung 3.6)[41].
Ohne jegliches Training, nur durch die Anweisung, den Punktestand zu maximieren, schaffte

[39] SAP 2023.
[40] Paass und Hecker 2020, S. 47 f.
[41] Paass und Hecker 2020, S. 30.

es der Algorithmus, innerhalb von 300 Spielen höhere Scores als Menschen zu erzielen. Nach 500 Spielen begann er sogar, durch taktische Kniffe Wege zu finden, um mehr Blöcke mit nur einem Schuss zu elminieren, weil dies mehr Punkte gibt[42].

3.2.7 Verlustfunktionen

Verlustfunktionen oder Loss Functions dienen dazu, einen Wert zu liefern, der angibt, wie nah die Ausgabe eines Neuronalen Netzwerks für ein Element aus dem Trainingsdatensatz an dem zugehörigen Label liegt. So liefert eine Eingabe, die unter Rücksicht der weights, biases und activation functions ein supervised learning Netzwerk durchläuft (Forward Propagation), am Ende entweder einen Ausgabewert (bei Regression) oder eine Werteverteilung (bei Klassifizierung). Damit das Netzwerk im Anschluss die eigenen Parameter selbständig anpassen kann, wird der Wert der Verlustfunktion herangezogen. Je größer die Differenz zwischen diesem Output und der Ground Truth ist, desto größer der Loss-Wert. Nach einer Trainingsepoche wird aus sämtlichen Loss-Werten der einzelnen Trainingsdaten ein Durchschnitt gebildet, den man als **Cost** bezeichnet, da er im übertragenen Sinne aussagt, wie viel es das System „gekostet“ hat, diese Trainingsdaten zu erlernen[43]. Das Ziel des Netzwerks besteht darin, diesen Cost-Wert zu minimieren. Wie das Netzwerk dies bewerkstelligt, wird in Kapitel 3.2.8: behandelt. Um den Loss Wert zu ermitteln, gibt es eine Reihe unterschiedlicher Loss Funktionen, von denen im Folgenden eine kleine Auswahl vorgestellt wird:

MSE - Mean Squared Error

[44] Für Regressionsaufgaben erfreut sich die Lossfunktion Mean Squared Error großer Beliebtheit, da große Abweichungen zwischen Prediction y_i und Ground Truth x_i durch das Quadrieren stärker in den Cost Wert einfließen als kleine. Wie der Name (mean = Mittel) bereits andeutet, handelt es sich um einen Mittelwert, der über die Anzahl D an Beoachtungen ermittelt wird:

$$MSE = \frac{1}{D}\sum_{i=1}^{D}(x_i - y_i)^2 \tag{3.1}$$

[42] Mnih u. a. 2013.
[43] Paass und Hecker 2020, S. 53 ff.
[44] Schwaiger und Steinwendner 2019, S. 160.

MAE - Mean Absolute Error

Sollten mehrere Elemente eines Trainingssets für starke Abweichungen zwischen Prediction und Label sorgen, empfiehlt es sich, auf das Quadrieren (siehe Kapitel 3.2.7) zu verzichten und stattdessen lediglich den Betrag zu nehmen.

$$MAE = \frac{1}{D}\sum_{i=1}^{D}|x_i - y_i| \tag{3.2}$$

Crossentropy Loss

Ein Crossentropy Loss eignet sich gut für Klassifizierungsaufgaben. Für den Fall, dass sowohl Prediction y_i als auch Ground Truth x_1 den Wert 1 annehmen, gibt die Lossfunktion den Wert 0 aus. C steht hierbei für die Anzahl der unterschiedlichen Kategorien im Ouput Layer. Durch den Logarithmus ergeben höhere Confidence Werte geringere Losswerte als dies bei einem linearen Verhältnis der Fall wäre.

$$CE = -\sum_{i=1}^{C} x_i \log(y_i) \tag{3.3}$$

Hinge Loss

Der Hinge Loss wird ebenfalls für Klassifizierungsaufgaben angewandt, wobei im Gegensatz zum Crossentropy Loss (siehe Kapitel 3.2.7) Ergebnisse mit geringerer Confidence durch sein Lineares Verhalten mit einem höheren Losswert „bestraft“ werden.

$$max(0, 1 - x_i \cdot y_i) \tag{3.4}$$

3.2.8 Optimierung und Backpropagation

Um den Optimierungsprozess eines Neuronalen Netzwerks zu veranschaulichen, wird häufig eine Visualisierung wie in Abbildung 3.7 verwendet. Hierbei stellt man sich vor, dass sich das Netzwerk auf der Suche nach einem Lokalen Minimum schrittweise einem Tal (einem niedrigeren Loss-Wert) annähert. Ein möglicher Weg ist hier zur Veranschaulichung in gelb eingezeichnet.
Um zu wissen, in welche Richtung die Parameter des Netzwerks angepasst werden sollen, benötigt es den Gradienten ∇C der Lossfunktion, um herauszufinden, wo die Steigung am höchsten ist - bzw. im Umkehrschluss: „wo der Weg am schnellsten ins Tal führt“. Das Netzwerk muss dementsprechend seine Werte so anpassen, dass es entgegen der Richtung des

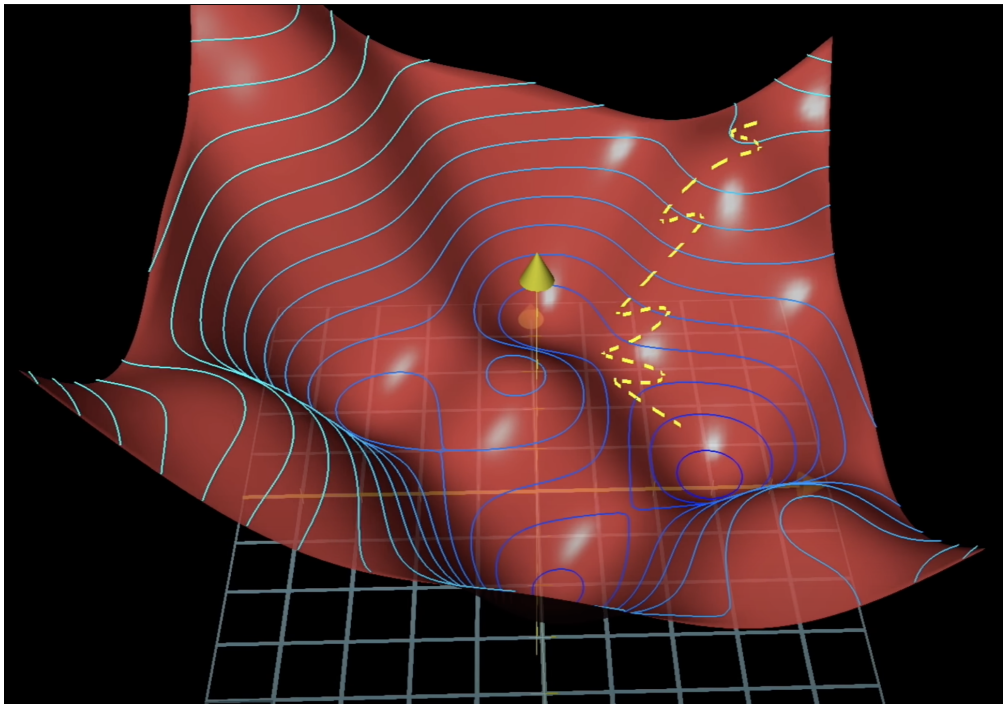

Abbildung 3.7: Visualisierung des Optimierungsprozesses eines Neuronalen Netzwerks
Quelle: Sanderson 2018

Gradienten wirkt.[45]
Ein Gradient kann in der Mathematik stets über die Ableitung der Funktion, in diesem Fall der Lossfunktion, ermittelt werden. Je nach Netzwerkarchitektur steht die Lossfunktion in Abhängigkeit von mehreren Tausend oder Millionen Parametern. Zudem variieren die mathematischen Formeln, mit denen rückwärts propagiert wird, je nach verwendetem **Optimizer**. Hier kann abermals aus mehreren Optionen (z.B. Stochastic Gradient Descent, Adam oder RMSprop) gewählt werden. Zudem kann man mittels des Parameters **Learning Rate** die Stärke der Änderungen am Netzwerk einstellen. Im übertragenen Sinne lässt sich hierdurch regeln, wie stark sich das Netzwerk entgegen der Richtung des Gradienten ∇C bewegt.
Üblicherweise nimmt der Optimizer die Änderungen am Netzwerk nicht nach jedem einzelnen Datenpunkt vor, von dem es im Zuge des Trainingsprozesses durchlaufen wird. Vielmehr wird ein Durchschnittsgradient gebildet, der sich aus mehreren Gradienten einer Anzahl einzelner Daten zusammensetzt. Diese Anzahl wird als **Batch Size** bezeichnet.[46]
Eine zu hohe Learning Rate kann dafür sorgen, dass das Netzwerk über Lokale Minima hinweggeht - eine zu geringe dafür, dass sich der Algorithmus in einem Minimum „einpen-

[45] Schwaiger und Steinwendner 2019, S. 151 ff.
[46] Paass und Hecker 2020, S. 66 ff.

delt “, obwohl es durch einen etwas größeren Schritt bei einem besseren Lokalen Minimum landen könnte. Üblicherweise werden Startwerte zwischen 0,001 und 0,1 gewählt. Bestimmte Optimizer (z.B. Adam) sind aber in der Lage, die Learning Rate dynamisch während des Trainings zu verändern.
Trainiert ein Netzwerk für viele Epochen immer wieder mit denselben Daten, so kann es passieren, dass das Netzwerk sich zu gut auf die Trainingsdaten anpasst, jedoch die Fähigkeit verliert, ihm unbekannte Daten ordnungsgemäß zu verarbeiten. Dieser Prozess wird als **Overfitting** bezeichnet und ist dringlichst zu vermeiden, da die große Stärke eines Machine Learning Algorithmus darin besteht, mit Daten umzugehen, die er zuvor noch nie zu Gesicht bekommen hat.[47]
Festzuhalten ist, dass ein Neuronales Netzwerk durch Backpropagation in der Lage ist, selbständig Veränderungen an den eigenen Parametern (weights und biases) vorzunehmen, um den Losswert auf ein lokales Minimum zu reduzieren.

Im Folgenden soll ein genauerer Blick auf das Fully Connected Netzwerk sowie auf eine alternative Architektur geworfen werden:

3.3 Bilderkennung mittels eines Fully Connected Netzwerks

Zieht man abermals das MNIST Datenset (siehe Kapitel 3.2.6) heran, bei dem sämtliche Bilder jeweils 28x28 Pixel groß sind, so müssen die Daten vorher entsprechend behandelt werden, um in ein Fully Connected Netzwerk mit seinen eindimensionalen Schichten gegeben werden zu können. In Python kann man beispielsweise mittels des Befehls *reshape* aus der NumPy Bibliothek die Werte in eine eindimensionale Form bringen. Ein Fully Connected Netzwerk, das mit dem MNIST Datenset trainiert wird, hätte dementsprechend $28 \cdot 28 = 784$ Eingabeneuronen und 10 Ausgangsneuronen (für die Kategorien 0-9).

Durch den Reshape liegen in der neuen Anordnung nun allerdings sonst vertikal zusammenhängende Pixel an unterschiedlichen Stellen. Während dieses neue Array demnach für das menschliche Auge nicht mehr als Zahl erkennbar ist, erkennt der Algorithmus nichtsdestotrotz Zusammenhänge zwischen den unterschiedlichen Eingabeneuronen und ist in der Lage durch gutes Training, Bilder mit Accuracys von bis zu 0,99 zu klassifizieren[48].

[47] Schwaiger und Steinwendner 2019, S. 204.
[48] Deotte 2018.

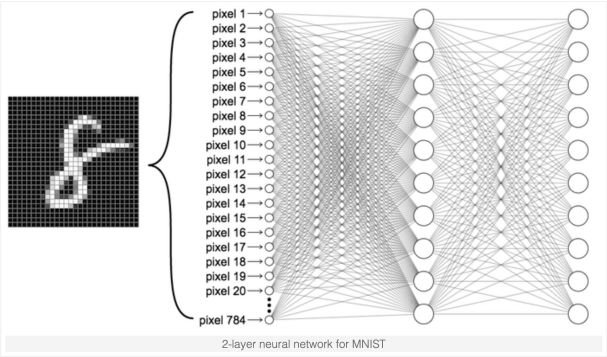

Abbildung 3.8: Reshapen eines Bilds des MNIST Datensets in eine lineare Form (784,1)
Quelle: Github o. D.

Die besten Accuracys von bis zu 0,9979 erreichen Fully Connected Networks jedoch nicht[49]. Hierfür eignet sich eine Netzwerkarchitekur mehr, die in der Lage ist, zweidimensionale Bilder zu verarbeiten, ohne dass ein Reshape in eine eindimensionale Form von Nöten ist.

3.4 Convolutional Neural Networks

Während ein Fully Connected Network einen eindimensionalen Input benötigt (siehe Kapitel 3.2.4), verarbeiten Convolutional Neural Networks (CNN) mehrdimensionale Daten, was den Vorteil mit sich bringt, dass es Muster in Bildern erkennt, die in bildlichem Zusammenhang stehen, z.B. horizontale/vertikale Linien, Rundungen etc.

In Abbildung 3.9 ist eine mögliche Convolutional Netzwerkarchitektur zur Klassifizierung des MNIST Datensets zu sehen. Die Fully Connected Layer, die nach dem CNN zum Tragen kommen, dienen lediglich dazu, den passenden Output Layer zu generieren, der anschließend mit dem Label abgeglichen werden kann.

Bei einem CNN spielen zwei unterschiedliche Layer-Arten eine entscheidende Rolle, die im Folgenden näher erläutert werden sollen:

Convolutional Layer

Entscheidend bei einem Convolutional Layer sind die Filter/**Kernels**, die man in einer bestimmten Schrittlänge (**stride**) über das Eingabebild bewegt. Anschließend werden die Werte des Bildbereichs, auf dem der Kernel liegt, mit den Werten des Kernels verrechnet und auf-

[49] Deotte 2018.

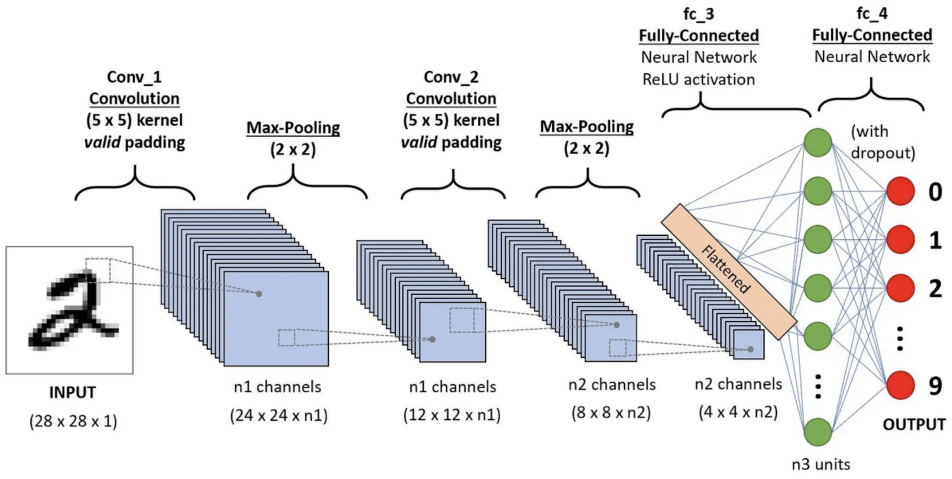

Abbildung 3.9: Beispiel für Klassifizierung des MNIST Datensets mittels einer Kombination aus CNN und Fully Connected
Quelle: Saha 2022

summiert. Hierdurch entsteht eine neue Matrix, die auch **Feature Map** oder **Activation Map** genannt wird.[50]

Da die neu entstandene Feature Map eine geringere Auflösung als das ursprüngliche Bild besitzt, kann man diese mittels **zero-padding** (oder: same-padding) am Bildrand mit 0-Werten auffüllen, um wieder die Ausgangsgröße zu erhalten.

Sollte die Verkleinerung der Activation Map gewünscht sein, kann man dies durch **valid-padding** erzielen. Hierdurch spart man sich nicht nur Rechenleistung (vor allem bei großen Activation Maps), sondern wirkt auch Overfitting entgegen, da der Algorithmus sonst versuchen könnte, Muster zu erkennen, die in Zusammenhang mit den ergänzten 0-Werten stehen[51].

Bei **cause-padding** handelt es sich um eine Sonderform von Padding, die vor allem bei eindimensionalen Daten, die entlang einer Zeitachse ermittelt wurden, Verwendung findet. Cause-padding sorgt dafür, dass output[t] nicht in Zusammenhang mit Input [t+1] steht, damit der Algorithmus quasi „nicht in die Zukunft schaut“[52].

Dank des Kernels lässt sich nun gezielt nach bestimmten Formen suchen, da es sich bei den Kernel-Werten um Neuronen handelt, die im Laufe des Trainings vom Netzwerk eigenständig angepasst werden können. Üblicherweise lässt man mehrere solcher Filter über das Eingabebild laufen, die allesamt potenziell unterschiedliche Muster erkennen können. Nach

[50] Paass und Hecker 2020, S. 126.
[51] Schwaiger und Steinwendner 2019, S. 193 f.
[52] Keras o. D.

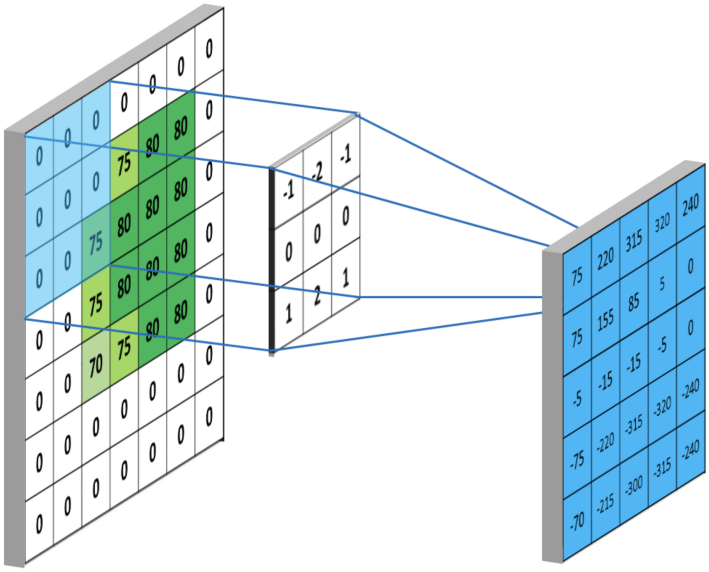

Abbildung 3.10: Beispiel für einen 3x3 Kernel eines Convolutional Layers mit stride=1
Quelle: Robinson 2017

dem Durchführen der Convolution wird eine Aktivierungsfunktion auf die neu entstandenen Feature Maps angewandt, um für Nichtlinearität zu sorgen (siehe Kapitel 3.2.5).

Pooling Layer

In den meisten Fällen folgt auf einen Convolutional Layer ein Pooling Layer (siehe Abbildung 3.9), der die Bildgröße reduziert (üblicherweise halbiert), während die essentiellen Bestandteile des Bilds erhalten bleiben. Ähnlich wie beim Convolutional Layer bewegt sich hierfür ein Kernel über das Bild, allerdings mit einer größeren Stride, wodurch sich die Kernels nicht überlappen und das resultierende Array verkleinert wird. Des Weiteren wird kein Padding angewandt.
Es gibt zwei unterschiedliche Arten von Pooling Layern: zum einen **MaxPooling**, zum anderen **AveragePooling**[53].
Bei MaxPooling wird, wie in Abbildung 3.11 zu sehen ist, der höchste Wert des vom Kernel abgetasteten Bereichs übernommen. Beim Average Pooling hingegen wird der Durchschnitt des Bereichs ermittelt und in das neue Array geschrieben.
Durch die Durchschnittsbildung beim AveragePooling werden scharfe Konturen in den Fea-

[53]Paass und Hecker 2020, S. 127.

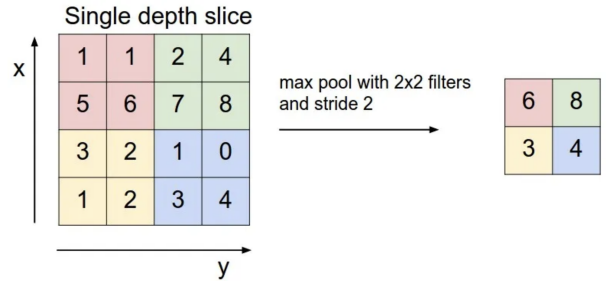

Abbildung 3.11: Beispiel für 2x2 Kernel eines MaxPooling Layer mit stride=2
Quelle: Udofia 2019

ture Maps beim Verkleinern weichgezeichnet, was nur bei Bildern mit Störelementen (beispielsweise Rauschen) zu empfehlen ist. Ansonsten wird vorzugsweise MaxPooling verwendet, da Muster und Konturen besser erhalten bleiben.

Möchte man ein CNN nicht für Klassifizierungsaufgaben verwenden, sondern um das Eingabebild wieder in seiner vollen Größe mit gewünschten Modifikationen zurückzuerhalten (z.B. scharfgezeichnetes Bild; um Menschen oder Tiere ergänzte Masken etc.), werden ein paar Schritte mehr benötigt. Diese werden im Folgenden erläutert:

3.5 UNET Architektur

Die UNET Architektur wurde erstmals 2015 von einem Team um Prof. Dr. Olaf Ronneberger von der Universität Freiburg präsentiert, welche das Netzwerk für Biomedizinische Bildsegmentierungsaufgaben verwendete, wie etwa das Erkennen und Maskieren von Zellen in Mikroskopaufnahmen[54].

Ein UNET setzt sich aus einem Contraction- und einem Expanding-Pfad zusammen. Im Zuge des Contraction Pfads werden auf das Eingabebild zwei Convolutional und darauffolgend ein MaxPooling Layer angewandt. Dieser Vorgang wird dreimalig wiederholt, wobei die Anzahl der Feature Maps pro Wiederholung verdoppelt wird[55].

Im Durchgang hin zum Expanding Path werden abermals zwei Convolutional Layer angefügt, ehe mittels Up-Convolution die Auflösung wieder verdoppelt wird. Des Weiteren werden auf dem Expanding Path die zuvor auf dem Contraction-Path erlernten Feature Maps wieder hinzugefügt (concatenated), ehe die Zahl der Feature Maps im Zuge zweier weiterer Convolutional Layer halbiert wird. Dieser Vorgang wird drei Mal wiederholt, bis am Output ein

[54]Ronneberger, Fischer und Brox 2015.
[55]Ronneberger, Fischer und Brox 2015.

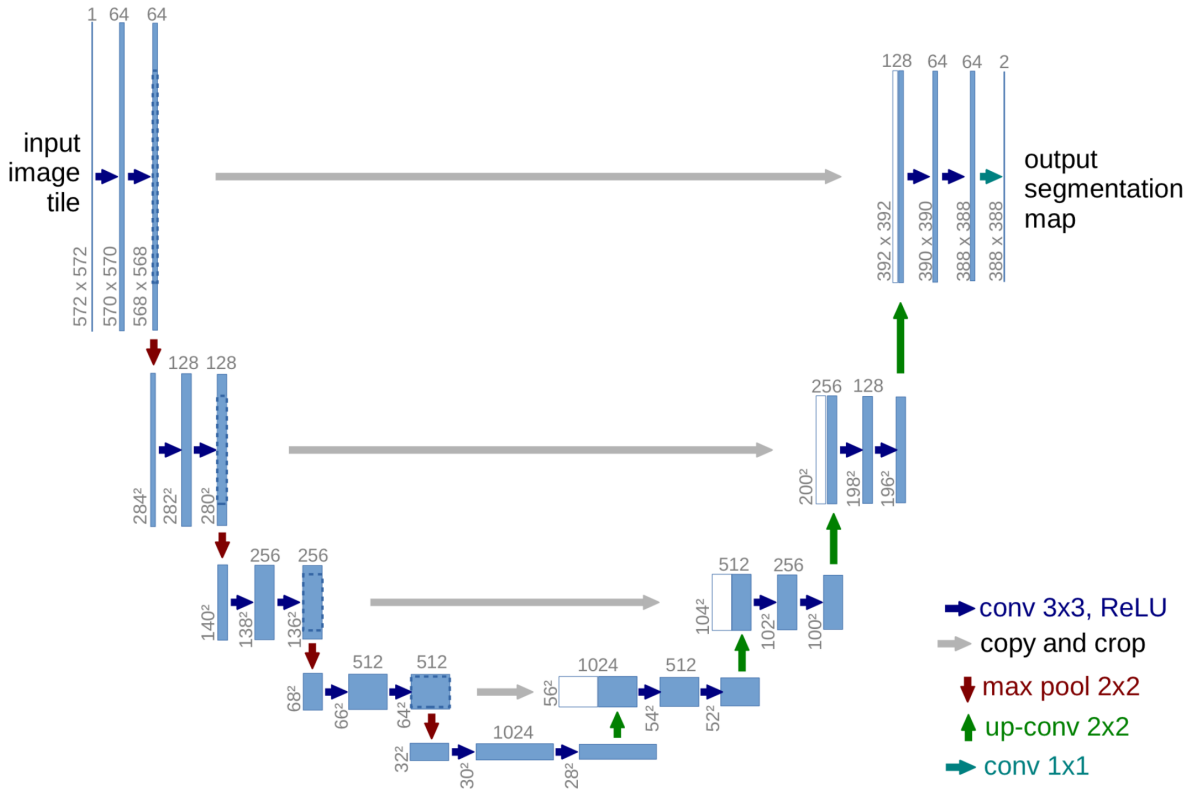

Abbildung 3.12: Netzwerk Architektur: UNET
Quelle: Ronneberger, Fischer und Brox 2015

Bild mit derselben Auflösung wie der des Eingabebilds generiert wird[56].

Um frequenzlimitierte Audiodaten mittels Machine Learning zu ergänzen, bedarf es zunächst eines Datensatzes zum Trainieren des Netzwerks.

[56]Paass und Hecker 2020, S. 147 f.

4 Erstellen eines Trainingsdatensatzes

4.1 Zielsetzung des Datensatzes

Das Ziel bei der Erstellung des Datensatzes bestand darin, ein und dasselbe Signal (in diesem Fall die Stimme des Autors) auf unterschiedlichen Übertragungswegen aufzuzeichnen, sodass am Ende die frequenzlimitierten Aufnahmen, die durch die Telefon-Übertragungswege entstehen, gemeinsam mit der Referenzaufnahme des Studiomikrofons auf parallelen Spuren anliegen. Dies soll im weiteren Verlauf die Handhabung der Daten und somit das Trainieren der Machine Learning Algorithmen vereinfachen. Als Input des Netzwerkes wird in der Folge eines der frequenzlimitierten Telefonsignale gewählt und als Label bzw. Ground Truth die Aufnahme des Studiomikrofons mitgegeben.

4.2 Aufnahme des Datensatzes

Hierfür wurde im Tonstudio des Studiengangs Medienwirtschaft an der Hochschule der Medien Stuttgart ein Aufbau errichtet, bei dem vor einem Sprecher das Studiomikrofon Schoeps Mk41, ein Apple iPhone 13, ein Apple iPad Pro 11“ (2021) und ein Apple MacBook Pro 16“ (2021) aufgestellt wurden (siehe Abbildung 4.1).

Das iPhone wurde zum Telefonieren über Mobilfunk, das iPad Pro zum Telefonieren per FaceTime, das MacBook Pro zum Telefonieren über Zoom genutzt. Bei FaceTime handelt es sich um eine Videocall Applikation von Apple selbst, die lediglich Apple Nutzern zur Verfügung steht[1].
Auf der Empfangsseite wurden ein Apple iPhone X für das Telefonat über Mobilfunk und ein Apple MacBook Pro 16“ (2021) zum Empfangen und zur getrennten Ausgabe des FaceTime und des Zoom Signals verwendet.
Aufgezeichnet wurden unter anderem sämtliche Vokale und stimmhaften Konsonanten einzeln in unterschiedlichen Tonhöhen, um den Algorithmus später die Zusammenhänge der

[1] Apple 2022.

Abbildung 4.1: Aufnahme des Trainingsdatensatzes mittels unterschiedlicher Signalwege
Quelle: Eigenes Bild

Obertöne (siehe Kapitel 2.1.1) trainieren lassen zu können. Darüber hinaus wurde ein langer Text aufgenommen, der sämtliche Buchstaben des deutschen Alphabets enthält sowie ein Text, der im weiteren Verlauf dieser Arbeit zur Evaluierung des Netzwerks dienen soll.
Sämtliche Signale wurden anschließend mit der DAW Cubase aufgezeichnet.

4.3 Aufbessern der Daten in Cubase

Da durch die Telefonwege Latenzen entstanden, wurden die aufgezeichneten Signale anschließend mittels des Audio Alignment Panel von Cubase synchronisiert.
Im nächsten Schritt wurden sämtliche Aufnahmen entsprechend der folgenden Nomenklatur als .wav-Dateien exportiert, sodass sie in der Folge korrekt vom Algorithmus einsortiert werden konnten:

```
1 Bsp.: a_01_Sebastian_Studio Mic_Schoeps_01.wav
```

Listing 4.1: Nomenklatur der exportierten Audiodaten

Sofern diese Benennung eingehalten wird, ermöglicht das auch in Zukunft anderen Sprechern und Sprecherinnen mittels einfacher Befehle einen Machine Learning Algorithmus zu trainieren.

4.4 Verarbeitung der Audiodaten mittels Python

4.4.1 Herangehensweise und Zielsetzung

Um die folgenden Schritte und deren Absichten besser verstehen zu können, soll zunächst die Herangehensweise des Autors erläutert werden. Zuerst soll ein Fully Connected Neural Net-

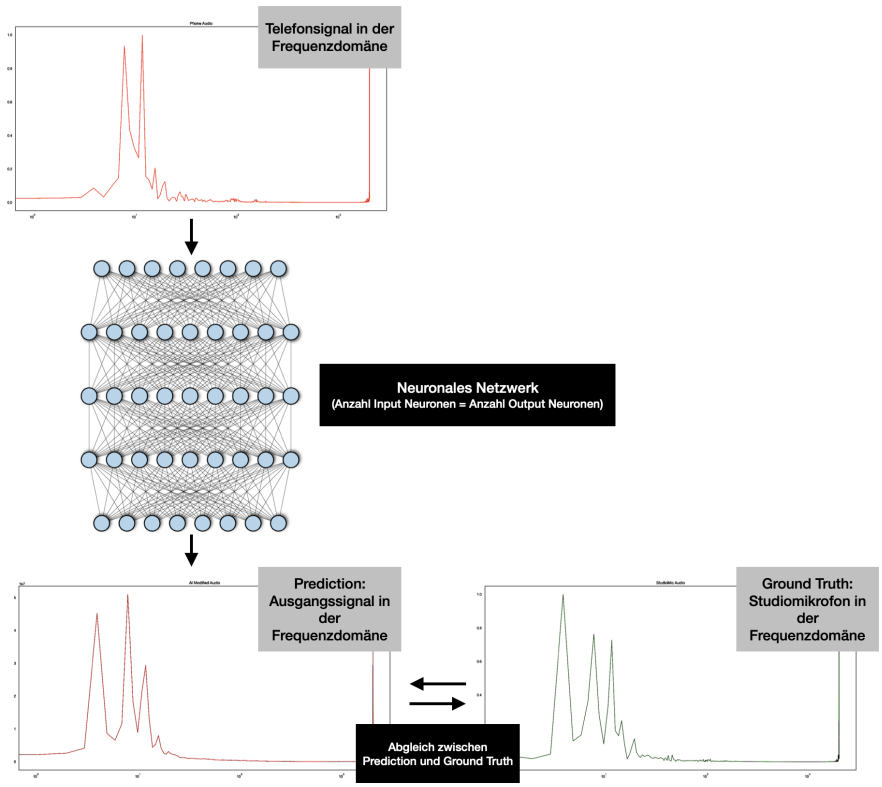

Abbildung 4.2: Herangehensweise an den Trainingsprozess anhand des Beispiels eines Fully Connected Networks

Quelle: Eigene Darstellung

work angelegt werden, das anschließend mit Frequenzgängen der frequenzlimitierten Aufnahmen (in diesem Fall: Telefonsignal) trainiert wird. Als Ground Truth werden die Frequenzgänge der Studiomikrofonaufnahme mitgegeben. Damit das Netzwerk zu jedem Frequenzgang die zeitlich richtige Referenz erhält, müssen die Audioaufnahmen des Trainingsdatensatzes, wie in Kapitel 4.3 beschrieben, synchronisiert werden.
Im Folgenden soll das Netzwerk versuchen, die Frequenzgänge der Telefonaufnahmen an die der Studioaufnahme anzugleichen.
Sobald das Netzwerk ausreichend trainiert wurde, sollte es in der Lage sein, eine Handy-

mikrofonaufnahme in eine im Frequenzgang verbesserte Audiodatei umzuwandeln. Dieser Prozess soll in Grafik 4.3 vereinfacht dargestellt werden:

Im ersten Schritt soll ein Ausschnitt des zeitbasierten Signals der Handymikrofonaufnahme

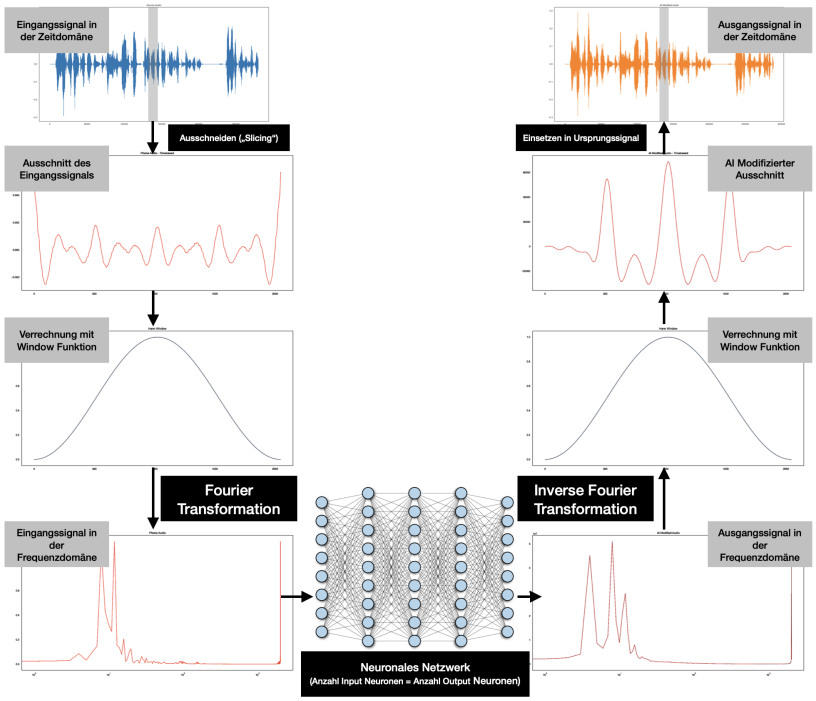

Abbildung 4.3: Herangehensweise an das Ergänzen von frequenzlimitierten Mikrofonaufnahmen

Quelle: Eigene Darstellung

mit einer Fensterfunktion verrechnet werden, damit keine diskontinuierlichen Signale entstehen. Das hieraus resultierende Array soll anschießend Fourier Transformiert werden, d.h. von einer zeitbasierten in eine frequenzbasierte Domäne umgewandelt werden. Dieses FFT-transformierte Array soll im nächsten Schritt in ein Deep Neural Network gegeben werden, das die Frequenzkoeffizienten entsprechend anpasst, sodass man einen neuen Frequenzgang erhält, der dem der Referenz (einer Studiomikrofonaufnahme) ähnelt.

Der vom Machine Learning Algorithmus modifizierte Ausschnitt wird anschließend mittels der invertierten FFT (IFFT) in die Zeitdomäne zurücktransformiert. Da der Algorithmus rein frequenzbasiert - ohne Rücksicht auf die Zeitdomäne - arbeitet, wird nach der IFFT abermals eine Window-Funktion darauf angewandt, bevor der Ausschnitt wieder an derselben Stelle in das Ursprungssignal eingesetzt wird, an der er vorher entfernt wurde.

Während des Evaluierungsprozesses erweist sich eine Fenstergröße von 2048 Samples als ein guter Kompromiss zwischen zeitlicher und frequenzbasierter Auflösung.

In den folgenden Kapiteln werden die Schritte beschrieben, welche der Autor unternommen hat, um den in Kapitel 4.2 aufgenommenen Trainingsdatensatz so zu verarbeiten, dass ein Machine Learning Model damit trainiert werden kann:

4.4.2 Erstellen einer Ordnerstruktur

Für einen übersichtlicheren Workflow sollen die ganzen unsortierten Audiodateien in entsprechende Ordner einsortiert werden, sodass am Ende jede Datei in einem eigenen Ordner liegt. Dies ermöglicht später ein erleichtertes Auslesen, da die Audiodateien nicht alle in eine lange Liste geladen werden, wodurch leichter Fehler bei der Wahl des richtigen Listenindex unterlaufen, sondern gezielt nach den Ordnern gesucht werden kann und dann einfach die eine Audiodatei extrahiert wird, die sich darin befindet. Hierfür öffnet man das browserbasierte Virtual Environment „Jupyter Notebook“, mit dem im Folgenden mittels der Programmiersprache Python programmiert wird.
Im ersten Schritt müssen diese ganzen Ordner definiert und angelegt werden. Dies erfolgt mittels der Definition create_folders, wo zunächst die unterschiedlichen Subfolders und deren jeweilige Unterunterordner bestimmt werden.

```
1 def create_folders(self):
2
3 subfolders = (a, e, i, o, u, m, n, ng, l,Text1, Text2, Text3,  Evaluierung1,
    Evaluierung2)
4
5 subsubfolders = (01_Studio Mic,  02_Telefonat normal,
6  03_Telefonat FaceTime,04_Zoom Call,
7  05_Angelmikrofon,  06_Studio Mic 2)
8 name = Sebastian Greim
```

Listing 4.2: Definieren der Ordnernamen

Im Folgenden werden mittels des Python Packages „os“ und dem Befehl *mkdir* (“make directory“ = „erstelle den Ordner“) die vorher definierten Ordner angelegt:

```
1 import os
2 os.mkdir(“Training Data“)
3 os.mkdir(f“Training Data/{name}“)
4
5 # für jedes Element in subfolders wird ein Ordner angelegt, der jeweils die
    Unterordner aus der Liste subsubfolders bekommt
6 for i in subfolders:
7     path = f“Training Data/{name}/{i}“
```

```
8
9       os.mkdir(path)
10      for j in subsubfolders:
11          os.mkdir(f"{path}/{j}")
12  (...)
```

Listing 4.3: Anlegen der Ordnerstruktur

Nachdem die Ordnerstruktur angelegt wurde, gilt es nun die entsprechenden Dateien einzusortieren. Dies erfolgt im Zuge der *sort_into_folders*-Definition. Hier werden zunächst das Package „re“ (*import re*), wodurch man Regular Expressions innerhalb von Python verwenden kann, und das Package „shutil“ (*import shutil*) zum Kopieren der Daten importiert. Anschließend werden die Dateinamen des Ordners, der die Trainingsdaten enthält, mittels os.mkdir aufgelistet. Sofern die Benennung der Files korrekt eingehalten wurde, extrahieren die folgenden Zeilen mittels Regular Expressions zunächst den Inhalt der Datei (z.B. „a“, „e“,..., „Text3“, „Evaluierung1“, „Evaluierung2“) und anschließend das Mikrofonsignal (01 für Studiomikrofon, 02 für normales Telefon, 03 für FaceTime, 04 für Zoom).

```
1  content = os.listdir(dir)
2
3  for i in range(len(content)):
4     temp_filename = content[i]
5
6    # Extrahieren des ersten Strings (zB a,e,..., Text1 usw)
7     temp_index = re.search(r'([A-Z]|[a-z]|[0-9])+', temp_filename).group()
8
9    # Extrahieren des zweiten Strings (gibt einem 01/02/03 usw aus)
10    temp_subindex = re.search(r'[0-9]{2}', temp_filename).group()
```

Listing 4.4: Zuweisung der Trainingsdaten zu den korrespondierenden Ordnern

Mittels dieser Informationen können im nächsten Schritt sämtliche Files in die korrespondierenden Ordner einsortiert werden.

```
1 #Kopieren der Daten von Quellenpath (src) zum Zielpath (dst)
2 src_path = r {dir}/{temp_filename}
3 dst_path = r Training Data/{name}/{temp_index}/{temp_subfolderstring}/{
      temp_filename}
4
5 shutil.copy(src_path, dst_path)
```

Listing 4.5: Kopieren von Source nach Destination Path mittels Shutil

4.4.3 Einlesen der Daten mittels librosa

Sobald sämtliche Daten korrekt einsortiert sind, gilt es, diese in Arrays zu laden, sodass für jede Aufnahme wieder die entsprechenden Spuren - ähnlich wie in der DAW - übereinander liegen. Somit erhält man pro Signal (Studiomikrofon, Telefon usw.) jeweils eine Zeile in einem Array. Die Länge der Zeilen stimmt mit der Anzahl der Samples überein. Zum Umgang mit Audiofiles wird das Python Package „Librosa“ verwendet, da es neben der Funktion, Audiofiles von der Festplatte in das Virtual Environment zu laden, weitere elementare Funktionen beinhaltet, wie etwa das Erstellen von Spektrogrammen oder das Konvertieren von Amplituden- in Dezibelwerte und umgekehrt. Mittels des soeben angesprochenen Befehls *librosa.load(file)* werden die .wav-Dateien nacheinander in die Virtual Environment geladen. Der Befehl *librosa.load()* gibt die Amplitudenwerte als eindimensionales Numpy Array zurück[2]. Durch eine Verkettung von for-Schleifen werden zusammengehörige Files - die unterschiedlichen Signale, die gleichzeitig aufgenommen wurden - stets nacheinander geladen. Da diese Signale dieselbe Anzahl an Samples besitzen, können diese im nächsten Schritt in ein leeres Numpy Array geladen werden, welches vorher mit folgendem Befehl angelegt wurde:

```
# Großes Array anlegen, in dem sämtliche synchronisierten Files übereinander liegen
dataset = np.empty((number_of_signals+1, samples), dtype=float)

# Anlegen einer Art “ Timecode“ an Index 0
dataset[0] = np.arange(samples)
```

Listing 4.6: Anlegen eines leeren Arrays mit der Länge Samples

Das leere Array ist so angelegt, dass es eine Zeile mehr besitzt als unterschiedliche Signale vorliegen. Dies hat den Grund, dass in der ersten Zeile (Index 0) eine Art Timecode angelegt wird, der nach dem Entfernen der Sprechpausen (siehe Kapitel 4.4.4) zwischen den einzelnen Buchstaben eine wichtige Rolle spielt. Bevor die einzelnen zusammengehörigen Signale in das Array geladen werden, werden diese zunächst normiert, um den Trainingsprozess des Machine Learning Algorithmus zu vereinheitlichen, da dieser stets den gleichen Umfang an Werten erhält. Die Variable counter beinhaltet den Index, an welchem das Signal in das Array einsortiert werden soll. Da an Index 0 bereits der angesprochene Timecode liegt, sortiert der Counter alle weiteren Signale ab Index 1 ein.

```
# höchsten Pegel des Signals finden
```

[2]Librosa 2023.

```
max_wert = y.max()

# Verhältnis vom Maximum zur Referenz herausfinden
faktor = max_wert/normalize_reference

# Signal mit dem Faktor verrechnen
y = y/faktor

# Hinzufügen der Audiodatei an der Stelle " counter"
dataset[counter] = y
```

Listing 4.7: Normalisieren des Pegels der Einzelspuren

4.4.4 Entfernen der Pausen

Um den Machine Learning Algorithmus ausschließlich mit Nutzsignal zu trainieren, werden im Zuge der Funktion *delete_gaps* die Sprechpausen zwischen den Vokalen entfernt. Hierfür wird das Array abermals mit einer Schleife abgetastet, die wiederum eine Schnittliste erstellt, welche die Timecodes sämtlicher Pausenanfänge und -enden jeweils als Tupel abspeichert. Anhand dieser Tupel kann nun im folgenden Schritt das große Array, in dem sämtliche Signale übereinanderliegen, durch Slicing (Ausschneiden) in kleinere pausenfreie Abschnitte unterteilt werden, die wiederum in eine neu angelegte Liste eingeordnet werden.

```
for i in range(len(cut_list)-1):
  # Slicing
  temp_arr = temp_dataset[:, int(cut_list[i][1]):int(cut_list[i+1][0])]
  # Das " gesclicte" Stück wird einer neuen Liste hinzugefügt
  sub_dataset_clean.append(temp_arr)
```

Listing 4.8: Entfernen der Pausen mittels Slicing

4.4.5 Fourier Transformation der Daten

Das lückenfreie Datenset aus Kapitel 4.4.4 wird im nächsten Schritt in Ausschnitte unterteilt, die jeweils 2048 Samples lang sind. Anschließend werden diese mit einer Windowfunktion verrechnet (in diesem Fall: Hann-Window), damit die Ausschnitte „ein- und ausfaden".

```
# Festlegen der Ausschnittsgröße
FRAME_SIZE = 2048

# Berechnung der Fensterfunktion
```

```
hann_curve = np.hanning(FRAME_SIZE)

# Ausschnitt aus der Audiodatei und Verrechnung mit Hann Curve
hann_audio_for_fft=np.multiply(raw_audio[start_of_slice:end_of_slice],
    hann_curve)
```

Listing 4.9: Verrechnung mit Fensterfunktion

Danach wird eine Fourier Transformation darauf angewandt, deren Formel wie folgt lautet:

$$\hat{g}(f) = \int_{t_1}^{t_2} g(t) e^{-2\pi i t f} dt$$

Dabei steht $g(t)$ für die zeitbasierte Audioaufnahme, bzw. für den eindimensionalen Strang aus Amplitudenwerten, den man beim Laden der Audiodateien erhält (siehe Kapitel 4.4.3). $\hat{g}(f)$ stellt im Gegensatz dazu das transformierte Signal dar, nun aber mit f (Frequenz) statt t (Zeit) als Achse. Anschließend wird das Integral von Anfangspunkt t_1 bis Endpunkt t_2 gebildet, wodurch man $\hat{g}(f)$ erhält.
Aus einem Array mit Amplitudenwerten entlang einer Zeitachse wird demnach ein Array aus Komplexen Zahlen, das nun in der Frequenzdomäne steht (siehe 4.10).
Die FFT Funktion ist in Python bereits durch die Library „numpy“ implementiert. Nach Durchführung der FFT werden die transformierten Werte in einen Bereich zwischen -1 und 1 normiert, damit das Netzwerk besser damit umgehen kann.

```
# Durchführung der FFT
audio_fft = np.fft.fft(hann_audio_for_fft)

# Normierung der FFT -> Werte zwischen -1 und 1
max_value = np.max(np.abs(audio_fft))
norm_factor = max_value/1
audio_fft = audio_fft/norm_factor

print(type(fft_transformed_evalutaion_audio))
print(type(fft_transformed_evalutaion_audio[0]))

############### Output ##################
numpy.ndarray
numpy.complex128
```

Listing 4.10: Normierung der Trainingdaten

Zuletzt werden die Daten in die neue Liste *fft_transformed_dataset* einsortiert.

4.4.6 Erstellung der Trainingsarrays für ein Fully Connected Network

Wenngleich das Netzwerk erst im nächsten Kapitel angelegt wird, soll hier bereits vorweggenommen werden, wie die Trainingsdaten final vorliegen müssen. Im Folgenden werden Neuronale Netzwerke der Python Library *Keras* verwendet, für die sich der Autor entschieden hat, da diese eine benutzerfreundliche, High-Level-Schnittstelle für den Aufbau und das Training von Deep Neural Networks bereitstellt.
Im konkreten Fall dieser Arbeit bedarf es zweier großer Arrays. In ersterem befinden sich sämtliche Input-Daten, in zweiterem die ganzen Labels. Zudem müssen die einzelnen Daten, die anschließend aus dem Array extrahiert werden, in eindimensionaler Form vorliegen, damit die Neuronen im Eingabelayer direkt aktiviert werden können.

```
1 audio_resized_studio_mic_stacked.shape
2
3 ########## Output ######################
4 (93243, 1, 2048)
```

Listing 4.11: Verarbeitung der FFT-Daten zu einer für den Machine Learning passenden Shape

Bei diesem Code handelt es sich um ein Beispiel, in welcher Form das Array am Ende vorliegt. Sämtliche FFT-transformiertern Ausschnitte (jeweils 2048 Samples lang) müssen zunächst in die Form (1, 2048) gebracht und anschließend „gestackt“ werden (zusammengefasst). Dies geschieht sowohl für die frequenzlimitierten Mikrofonaufnahmen als auch für die Studiomikrofon-Referenz, wodurch man am Ende zwei Arrays erhält, die in ihrer Shape gleich sind.
Mit diesen Daten kann nun ein Fully Connected Network trainiert werden.

5 Fully Connected Network zum Ergänzen frequenzlimitierter Audiodaten

5.1 Anlegen eines Fully Connected Networks in Python

Dank der Library Keras von tensorflow kann man mit nur wenigen Zeilen ein Machine Learning Model anlegen. Zur Verarbeitung der Trainingsdaten, die 2048 Samples lang sind, benötigt das Model jeweils 2048 Eingangs- und Ausgangsneuronen.

```
1  model_fully_connected = Sequential([Dense(units=FRAME_SIZE, activation=relu,
       input_shape=(1,2048)), Dense(units=FRAME_SIZE, activation=relu), Dense(
       units=FRAME_SIZE, activation=relu)])
2  model_fully_connected.summary()
3
4  ############## Output ############################
5  Model: "sequential_1"
6  _________________________________________________________________
7   Layer (type)                 Output Shape              Param #
8  =================================================================
9   dense_24 (Dense)             (None, 1, 2048)           4196352
10
11  dense_25 (Dense)             (None, 1, 2048)           4196352
12
13  dense_26 (Dense)             (None, 1, 2048)           4196352
14
15 =================================================================
16 Total params: 12,589,056
17 Trainable params: 12,589,056
18 Non-trainable params: 0
```

Listing 5.1: Anlegen eines Fully Connected Netzwerks

Dieses Model verfügt über 12,5 Millionen anpassbare Parameter, die im Zuge des Trainings modifiziert werden können.
Über den Befehl

```
model_fully_connected.compile(optimizer=Adam(learning_rate=0.1),
              loss= “categorical_crossentropy“)
```

Listing 5.2: Compiling des Fully Connected Netzwerks

lassen sich wichtige Einstellungen, wie etwa die Lossfunktion (siehe Kapitel 3.2.7) und Optimizer mit Learning Rate (siehe Kapitel 3.2.8), vornehmen.

5.2 Trainingsprozess

Das angelegte Netzwerk lässt sich mittels des Befehls

```
model_fully_connected.fit(x=audio_resized_bad_signal_stacked,
              y=audio_resized_studio_mic_stacked,
              validation_split=0.1,
              batch_size = 256, epochs = 10,
              shuffle=True)
```

Listing 5.3: Trainieren des Fully Connected Netzwerks

trainieren. Das Model bekommt als Input x die frequenzlimitierten Aufnahmen und als Label y die Studiomikrofon-Referenzen. Des Weiteren spaltet das Netzwerk automatisch einen Teil der Trainingsdaten ab und nutzt diese zum Validieren. Beim Trainieren werden diese dem Netzwerk vorenthalten. Durch den Validation Split lassen sich verlässlichere Aussagen über die Performance des Netzwerks treffen, was vor allem dabei helfen kann, Overfitting zu verhindern. Steigt nämlich der Accuracy-Wert bei den Trainingsdaten immer weiter an, während der über den Split berechnete Validation-Accuracy-Wert sinkt, ist dies ein klares Zeichen von Overfitting.
Die Batch Size bestimmt, dass das Netzwerk selbständig nach 256 unterschiedlichen Eingaben aus dem Trainingsset Änderungen an den eigenen Parametern vornimmt.
Durch die aktivierte Shuffle-Funktion werden die Trainingsdaten gemischt - Inputs und Labels gleichermaßen. Dies kann vor allem bei vorsortierten Daten von großem Nutzen sein. Als Beispiel soll abermals das MNIST-Datenset (siehe Kapitel 3.2.6) herangezogen werden, wo sämtliche Zahlen zunächst sortiert vorliegen. Würde man ein Netzwerk nun bei deaktivierter Shuffle-Funktion trainieren, würde es zunächst nur mit Bildern von handgeschriebenen Nullern, anschließend mit Einsern usw. trainieren. Das Netzwerk würde seine Parameter

immer wieder auf eine bestimmte Zahl ausrichten, jedoch direkt nach einem Wechsel der Zahl schlechtere Ergebnisse liefern. Durch das Mischen werden sämtliche Daten in zufälliger Reihenfolge trainiert, wodurch die Klassifizierung deutlich verbessert wird.

Dieses soeben angelegte Fully Connected Network besitzt mehrere Ausgangsneuronen, was eher Eigenschaft einer Klassifizierungsaufgabe (siehe Kapitel 3.2.6) ist, andererseits nimmt es aber keine Klassifizierung vor, sondern passt Fourierkoeffizienten an, was für eine Regressionsaufgabe spricht. Somit bedarf es einiger Versuche, bis zu sehen ist, welche Loss-Funktion, welcher Optimizer und welche Activation Function zum besten Ergebnis führen.

5.3 Algorithmus zur Evaluierung des Klangs

Um beurteilen zu können, welche Parameter auf welche Weise das Ergebnis beeinflussen, werden die Werte der Loss-Funktionen - sowohl für das Training als auch den Validation Split - sowie das Ergebnis des Algorithmus in der Anwendung betrachtet und untereinander verglichen. Um den Machine Learning Algorithmus zum Verbessern von Audiodaten nutzen zu können, muss zunächst noch der Code hierfür implementiert werden.
Der Hintergrundgedanke hierfür wurde bereits in Kapitel 4.4.1 erläutert.
In Python geschrieben sieht dieser Ablauf wie folgt aus:

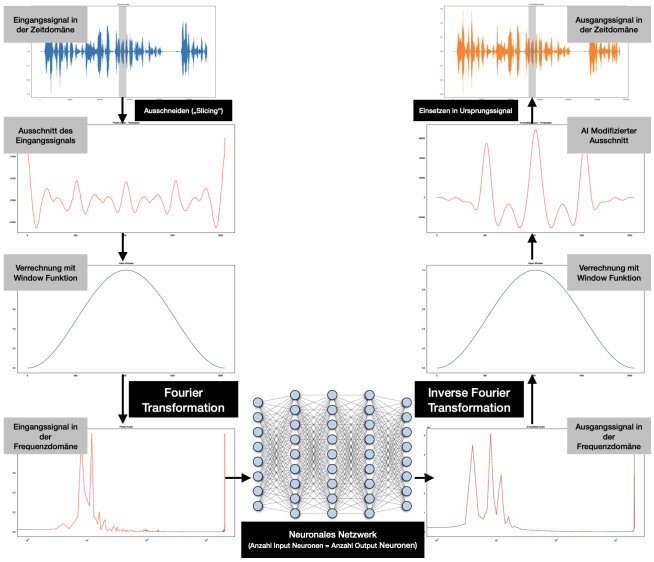

Abbildung 5.1: Herangehensweise an das Ergänzen von frequenzlimitierten Mikrofonaufnahmen

Quelle: Eigene Darstellung

```
# Laden der Audiodatei
y, sr = librosa.load(path, sr = samplerate)

# Berechnung des Hann-Fensters
hann_curve = np.hanning(FRAME_SIZE)

# Ermittlung der Länge des Audiofiles
y_length = np.shape(y)[0]

# Berechnung der möglichen Hops innerhalb des Audiofiles
index = int(y_length/VAL_HOP_SIZE)

```

```
# Ausschneiden der 2048 Samples langen Ausschnitte
for i in range(index):
    start_of_slice = int(i*VAL_HOP_SIZE)
    end_of_slice = int(start_of_slice + FRAME_SIZE)

# Verrechnung mit dem Hann-Fenster
hann_evaluation_audio_for_fft = np.multiply(y[start_of_slice:end_of_slice],
    hann_curve)
```

Listing 5.4: Laden einer Testdatei und Slicing und Verrechnung mit Hann Fenster

Im nächsten Schritt wird die Fourier Analyse durchgeführt,

```
fft_transformed_evaluation_audio = np.fft.fft(hann_evaluation_audio_for_fft)
```

Listing 5.5: Durchführung der FFT an Ausschnitten der Testdatei

Anschließend wird das Signal auf einen Wertebereich zwischen -1 und 1 genormt, da der Machine Learning Algorithmus mit genau diesen Werten trainiert wurde. Nachdem das Signal mittels eines *reshapes* in die der KI vertrauten Form gebracht wurde, gibt diese ihre Prediction ab, wie das Audio mehr nach Studiomikrofon klingt.

```
fft_transformed_evaluation_audio = model_fully_connected.predict(
    fft_transformed_evaluation_audio.reshape(1,1,FRAME_SIZE))
```

Listing 5.6: Prediction des Netzwerks

```
ifft_transformed_evaluation_audio = np.fft.ifft(
    fft_transformed_evaluation_audio)

# Verrechnung mit dem Hann-Fenster
modified_ifft_transformed_evaluation_audio = np.multiply(
    ifft_transformed_evaluation_audio, hann_curve)

# Einsetzen des AI modifizierten Audios an der Stelle, wo es entfernt wurde:
ai_modified_audio[start_of_slice:end_of_slice] = np.add(
    modified_ifft_transformed_evaluation_audio,ai_modified_audio[
    start_of_slice:end_of_slice])
```

Listing 5.7: Inverse FFT und Verrechnung mit dem Hann Fenster und Einsetzen in ein neues Array

Diese Vorgehensweise ist in der Lage, ein Audiofile in viele kleine Einzelteile zu zerkleinern und anschließend wieder zusammenzusetzen. Bei einem Test, bei dem der Machine Learning

Algorithmus gebypassed wurde, war kein Unterschied zwischen Ursprungsignal und dem Signal, das den Algorithmus (ohne den KI-Teil) durchlief, hörbar.

5.4 Algorithmus zur rechnerischen Evaluierung des KI-Audios

Während das so neu entstandene Signal auf seinen Klang geprüft werden kann, wurde eine weitere Funktion programmiert, mit der das neue Signal auf mathematischer Ebene beurteilt werden kann, indem es einen Zahlenwert ausgibt, der anzeigt, wie stark der KI-Audio-Frequenzgang von dem des Studiomikrofons abweicht. Hierdurch lässt sich auch beurteilen, ob das KI-modifizierte Signal näher an die Referenz heranreicht als das Telefonsignal.

```
# Konvertierung der Signale in die Frequenzdomäne
fft_ai_modified_audio = np.abs(np.fft.fft(ai_modified_audio))
fft_phone_audio = np.abs(np.fft.fft(phone_audio))
fft_studio_mic = np.abs(np.fft.fft(studio_mic_audio))

# Berechnung der durchschnittlichen Differenz der Signale
differenz_phone = np.average(np.abs(fft_studio_mic-fft_phone_audio))
differenz_ai = np.average(np.abs(fft_studio_mic-fft_ai_modified_audio))
```

Listing 5.8: Ermittlung der Abweichung von Telefon- und KI-Signalen gegenüber der Studiomikrofon-Referenz

5.5 Ergebnisse

5.5.1 Ergebnisse mit Komplexen Trainingsdaten

Zunächst wurden die Aktivierungsfunktion ReLU und die Verlustfunktion MeanSquaredError (MSE) gewählt. Als Optimizer wurde Adam gewählt, da dieser in der Lage ist, die Learning Rate während des Prozesses entsprechend anzupassen.
Ausgehend von diesen Einstellungen wurden anschließend weitere Trainingsprozesse mit leicht veränderten Parametern durchgeführt.
Trainiert wurde mit den Fouriertransformierten Arrays, die komplexe Zahlen beinhalten. Hierbei wurde auf sämtliche Trainingsdaten (Vokale, stimmhafte Konsonanten und Textbeispiele) zurückgegriffen.

Anschließend wurde die trainierte KI in den in Kapitel 5.3 (siehe Abbildung 5.1) beschriebenen Algorithmus eingebunden, um eine Testdatei (Telefonsignal von Evaluierungstext 2) in ihrem Frequenzspektrum an ein Studiomikrofon anzunähern.
Zuletzt wurde die Differenz zwischen den Frequenzgängen des KI-generierten Audios und des Studiomikrofons (entsprechend das Studiomikrofonsignal von Evaluierungstext 2) gebildet, um zu ermitteln, ob das KI-Audio auf rechnerischer Ebene näher an das Studiomikrofon herankommt als das Telefon-Audio.
Ist diese Differenz gleich 0, bedeutet dies, dass das Telefonaudio fehlerfrei in das Studiomikrofonsignal umgewandelt werden konnte. Je höher die Differenz ist, desto unterschiedlicher sind die Signale. Als Referenz dient hierbei der Unterschied zwischen Telefon- und Studiomikrofonsignal, der für die Aufnahme „Evaluierungstext 2“ **7,7225** Einheiten beträgt.
Eine Übersicht über die hierbei erzielten Ergebnisse ist der folgenden Tabelle zu entnehmen.

Nr.	Datensatz	Activation	Loss Function	Diff KI	Anzahl Epochen	Anmerkungen
1	Komplexe Zahlen	ReLU	MSE	nan	30	Not a number -> Stille. Kein Ergebnis
2	Komplexe Zahlen	Leaky ReLU	MSE	13,8672	30	Nur Artefakte, Keine Sprachverständlichkeit
3	Komplexe Zahlen	Sigmoid	MSE	nan	30	Not a number -> Stille. Kein Ergebnis
4	Komplexe Zahlen	ReLU	MAE	nan	50	Not a number -> Stille. Kein Ergebnis
5	Komplexe Zahlen	Leaky ReLU	MAE	14,2437	50	Nur Artefakte, Keine Sprachverständlichkeit
6	Komplexe Zahlen	TANH	MAE	18,7589	50	Nur Artefakte, Keine Sprachverständlichkeit
7	Komplexe Zahlen	ReLU	CC	8,6856	50	Nur Artefakte, Keine Sprachverständlichkeit
8	Komplexe Zahlen	Leaky ReLU	CC	17,307	50	Nur Artefakte, Keine Sprachverständlichkeit

Sämtliche trainierten Modelle sowie die zugehörigen generierten Audiofiles sind im Digi-

talen Anhang unter der Nummer in der linken Spalte zu finden.
Im Zuge der aufgezeigten Versuche war es nicht möglich ein Audiofile zu generieren, das sprachverständlich war (geschweige denn nach einem Studiomikrofon klang). Weder Veränderungen an der Aktivierungs- noch an der Verlustfunktion führten positive Veränderungen herbei.
Rein rechnerisch lag Categorical Crossentropy (CC) in Kombination mit ReLU mit 8,69 am nähesten bei der Referenz des Telefons von 7,72. Da jedoch eine Zahl unterhalb von 7,72 angestrebt wurde und auch in dieser Konstellation keinerlei Sprachverständlichkeit erzielt werden konnte, ist diese erste Versuchsreihe als gescheitert zu betrachten.
Eine mögliche Erklärung hierfür liegt möglicherweise in der Verwendung von komplexen Zahlen. Da diese Zahlen über zwei Komponenten (Real und Imaginär) verfügen, die beide entsprechend angepasst werden müssen, könnte dies - zumindest mit der verwendeten Netzwerkarchitektur - zu Schwierigkeiten führen.
Somit gilt es einen neuen Weg zu finden, mit dem das Netzwerk mit Reellen Zahlen trainieren kann und dennoch beim Rücktransformieren mittels der IFFT nutzbare Ergebnisse liefert.

5.5.2 Anpassungen im Code

Diese Anforderungen lassen sich mittels des folgenden mathematischen Ablaufs erfüllen: Über den Befehl *np.abs()* lässt sich der Betrag (entspricht der Amplitude) einer komplexen Zahl ermitteln, durch die Funktion *np.angle()* die Phase des Signals.

```
# Initialisierung eines random Arrays mit 5 Werten
array = np.random.rand(5)

# FFT des Arrays + Aufspaltung in Phase und Amplitude
fft_arr = np.fft.fft(array)
fft_amp = np.abs(fft_arr)
fft_phase = np.angle(fft_arr)

# neues Array zur Wiederzusammensetzung
fft_reconstructed = fft_amp*np.exp(1j*fft_phase)

# IFFT berechnen und Real-Anteil isolieren
arr_reconstructed = np.fft.ifft(fft_reconstructed)
arr_reconstructed = np.real(arr_reconstructed)

print("Ursprüngliches Array: ", array)
```

```
17 print("Wiederhergestelltes Array: ", arr_reconstructed)
18
19 ############## Output #################
20 Ursprüngliches Array: array([0.9544728 , 0.42905577, 0.69126642, 0.83502601,
       0.05476647])
21 Wiederhergestelltes Array: array([0.9544728 , 0.42905577, 0.69126642,
       0.83502601, 0.05476647])
```

Listing 5.9: Aufspalten einer FFT in Amplitude und Phase und anschließendes Wiederzusammensetzen am Beispiel eines Beispiel-Arrays

Wie in Code-Beispiel 5.9 zu sehen ist, lässt sich das Audio auch nach der Aufspaltung in Amplitude und Phase wieder zusammensetzen, wobei die beiden Parameter jeweils reelle Werte sind.
Da vor allem eine Änderung der Amplitude an bestimmten Stellen des Frequenzspektrums gewünscht ist, werden die in Kapitel 4.4.5 beschriebenen Schritte bei der Erstellung des Datensatzes durch *np.abs()* so angepasst, dass nur noch die Amplituden statt der komplexen Zahlen in das Trainingsdatenset geschrieben werden.
Bei der in Kapitel 5.3 Vorgehensweise müssen ebenfalls Anpassungen getroffen werden, sodass die Phase vor der Prediction des Models abgespeichert wird, um nachträglich wieder mit dem Ergebnis der KI verrechnet zu werden(siehe Listing 5.10).

```
1 fft_transformed_evaluation_audio = np.fft.fft(hann_evaluation_audio_for_fft)
2
3 fft_transformed_evaluation_audio_ampl = np.abs(
      fft_transformed_evaluation_audio)
4
5 fft_transformed_evaluation_audio_phase = np.angle(
      fft_transformed_evaluation_audio)
6 (...)
7 # Prediction des Models für neue Amplitudenwerte
8 predicted_audio_ampl = model_1.predict(fft_transformed_evaluation_audio_ampl.
      reshape(1,1,FRAME_SIZE))
9 (...)
10 # Zusammensetzen von neuen Amplituden und alter Phase
11 predicted_audio = predicted_audio_ampl * np.exp(1j*
      fft_transformed_evaluation_audio_phase)
12
13 # Realteil der IFFT nehmen
14 ifft_transformed_evaluation_audio = np.real(np.fft.ifft(
      fft_transformed_evaluation_audio))
```

Listing 5.10: Anpassungen des Evaluierungsalgorithmus zur Trennung von Phase und Amplitude

Die KI passt demnach nur noch die Amplitudenwerte an. Die Ergebnisse sind im folgenden Kapitel zu sehen.

5.5.3 Ergebnisse mit Amplituden-basiertem Trainingssatz

Die in Kapitel 5.5.2 angewandten Änderungen liefern bei gleicher Vorgehensweise wie in Kapitel 5.5.1 folgende Ergebnisse:

Nr.	Datensatz	Activation	Loss Function	Diff KI	Anzahl Epochen	Anmerkungen
1	Amplituden	ReLU	MSE	5,992	50	Sprache unverständlich und mit starken Artefakten
2	Amplituden	Leaky ReLU	MSE	5,7635	50	Sprache unverständlich und mit starken Artefakten
3	Amplituden	ReLU	CC	5,9206	50	Sprache unverständlich und mit starken Artefakten
4	Amplituden	Leaky ReLU	CC	6,3272	50	Sprache unverständlich und mit starken Artefakten

Auch bei dieser Versuchsreihe sind sämtliche trainierten Modelle sowie die zugehörigen generierten Audiofiles sind im Digitalen Anhang unter der Nummer in der linken Spalte zu finden.
Zwar sind die klanglichen Ergebnisse so stark artefaktbehaftet, dass die unmodifizierte Telefonaufnahme für den Nutzer noch immer besser klingt, jedoch liegen die hierdurch erzielten Frequenzgänge rein rechnerisch näher an denen der Studiomikrofonsignale als die des Telefons.
Am Besten schneidet in diesem Versuch die Kombination aus Mean Squared Error und Leaky

ReLU ab, die mit ihrem Abweichungswert von 5,76 ein gutes Stück unter den 7,72 des Telefonsignals liegt.
Daraus lässt sich schließen, dass, wenn das FFT-Signal in Amplitude und Phase umgerechnet wird und der Algorithmus anschließend mit den Amplitudenwerten trainiert, dieser seine Aufgabe ordentlicher ausführt als beim Training mit komplexen Zahlen (siehe Kapitel 5.5.1).
Wirft man einen Blick auf die Frequenzgänge einzelner 2048-Samples Ausschnitte, lässt sich daraus ablesen, wo die prägnantesten Unterschiede im Spektrum zu finden sind, die für die hörbaren Artefakte sorgen. Wie in Abbildung 5.2 zu sehen ist, entstehen durch den Algorithmus vor allem unterhalb des Grundtons stärkere Amplitudenanhebungen. Zwar liegen diese

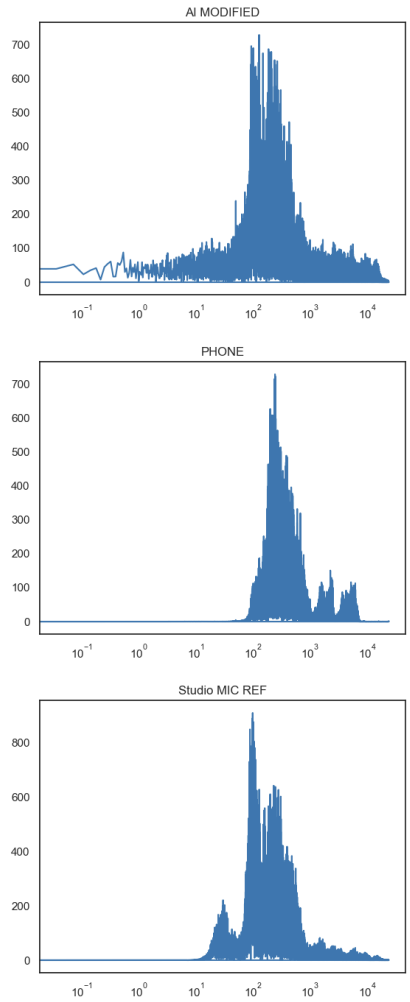

Abbildung 5.2: Frequenzgänge des Telefonsignals (Mitte), des Studio Mikrofons (Unten) und des KI generierten Signals (Oben) für Evaluationsaudio 2
(*x: Frequenzen, y: Amplitude*)
Quelle: Eigene Darstellung

teilweise bereits außerhalb des vom Menschen hörbaren Frequenzbereichs, könnten aber dennoch durch der Verrechnung mit der IFFT bestimmte Artefakte erzeugen. Dennoch schafft es die KI den Grundton der Stimme des Autors (ca. 100Hz) zu ergänzen, während dieser im Telefonspektrum bereits sehr stark beschnitten ist.
Insgesamt ist davon auszugehen, dass die Artefakte mit einem besser trainierten (und womöglich anders aufgebautem Netzwerk) geringer werden.

6 Future Work

Ausgehend von dieser Arbeit kann in Zukunft in zahlreiche weitere Bereiche geforscht werden.
Zum einen gilt es, ein Netzwerk zu erstellen, das mit komplexen Zahlen umzugehen vermag, da dies unausweichlich ist, um sämtliche Komponenten des zu verbessernden Signals (also auch die Phase) gleichzeitig zu behandeln. Das Berechnen neuer Amplitudenwerte und das anschließende Verrechnen mit der unbearbeiteten Phase könnte somit umgangen werden.
Des Weiteren können Versuche unternommen werden, ob sich mittels einer komplexeren Netzwerkarchitektur bessere Ergebnisse erzielen lassen. Als Beispiel hierfür wurde in dieser Arbeit bereits die Netzwerk Architektur „UNET“ vorgestellt. Da die Möglichkeit besteht, Audio in Form von Spektrogrammen (Frequenzspektrum entlang einer Zeitachse) zweidimensional darzustellen, wäre dies ein Ansatz, der möglicherweise bessere Ergebnisse erzielt. Denkbar wäre auch ein Cluster aus mehreren Neuronalen Netzwerken, die jeweils für gewisse Ausschnitte des Frequenzspektrum zuständig sind. Die Tatsache, dass mit einem simplen Fully Connected Netzwerk auf mathematischer Ebene bereits nutzbare Ergebnisse erzielt werden konnten, lässt hoffen, dass mit komplexeren Architekturen noch präzisere, artefaktärmere Audioumwandlungen möglich sind.
Darüber hinaus könnte versucht werden, mittels Bandpassfiltern nur bestimmte Bereiche des Telefonspektrums durch das KI-bearbeitete Signal zu ersetzen. Somit ließen sich evtl stärker artefaktbelastete Bereiche herausfiltern.
Auch beim Trainingsprozess besteht die Möglichkeit, mehr ins Detail zu gehen: Mit einer stärker auf den Trainingsprozess ausgerichteten Arbeit könnten hier ebenfalls durch längeres Trainieren mit unterschiedlichen Architekturen noch geringere Loss-Werte erzielt werden.
Im Zuge der Optimierung des Trainingsprozesses ist auch eine alternative Berechnung der Lossfunktion denkbar: Bildet man den Loss-Wert erst nach der Rücktransformation in die Zeitdomäne, wobei das Netzwerk als Input ein Frequenzspektrum erhält, aber den Output zunächst IFFT transformiert und anschließend dieses Signal mit einem zeitbasierten Label abgleicht, könnten möglicherweise ebenfalls optimierte Ergebnisse erzielt werden, da das Netzwerk dann den Zusammenhang zwischen Frequenz- und Zeitdomäne trainieren könnte.

Da Letztere am Ende entscheidend zum Abspielen und Exportieren des Audios ist, könnte ein Netzwerk mit stärkerem Fokus auf die Zeitdomäne artefaktfreiere Resultate erzielen.
Darüber hinaus würde ein umfangreicherer Trainingsdatensatz höchstwahrscheinlich ebenfalls zu präziseren Ergebnissen führen. Wenn im Zuge des Recordings des Trainingssatzes beispielsweise Texte mit unterschiedlichen Emotionen in variierenden Tonhöhen aufgenommen werden, wird die Wahrscheinlichkeit höher sein, dass der Algorithmus auch bei Sätzen, die der Sprecher nicht in seiner natürlichen Stimmlage spricht, genauere Ausgaben liefert.
Um den Algorithmus perspektivisch auf sämtliche (Video-)Telefonate anwenden zu können, werden zudem größere Trainingsdatensätze von mehreren Sprecher:innen von überall auf der Welt benötigt werden. In einer weiterführenden Arbeit könnte derselbe Algorithmus demnach mit mehreren Sprecher:innen trainiert werden, um zu beobachten, wie sich das auf das Gesamtergebnis auswirkt.
Sobald ein Algorithmus, der mit komplexen Zahlen umgehen und artefaktfreies Audio wiederherstellen kann, geschaffen wird, können perspektivisch Versuche unternommen werden, wie der Algorithmus in moderne Computer und Telefone implementiert werden könnte. Um den Frequenzgang in Zukunft beim Telefonieren ohne große zeitliche Verzögerung in Real-Time anpassen zu können, bedarf es zudem eines umfangreichen Optimierungsprozesses.

7 Fazit und Ausblick

In dieser Arbeit wurde ein Ansatz präsentiert, wie man mittels Machine Learning frequenzlimitierte Aufnahmen in ihrem Frequenzspektrum ergänzen kann.
Hierfür wurde vom Autor ein neuer Datensatz erstellt, der anschließend mit Python in eine entsprechende Form gebracht wurde, um damit einen Machine-Learning-Algorithmus trainieren zu können. Bei der Netzwerk-Architektur wurde ein Fully Connected Neural Network mit mehr als 12,5 Millionen anpassbaren Parametern gewählt. Es wurden Versuche mit unterschiedlichen Loss-Funktionen und unterschiedlichen Anpassungen im Programmiercode durchgeführt, um zu evaluieren, ob ein Fully Connected Netzwerk in der Lage ist, Audiodaten zu verarbeiten und im Frequenzgang manipuliert wiederherzustellen.
Beim Umgang mit komplexen Zahlen konnten die unterschiedlich modifizierten Netzwerke im Trainingsprozess keine zufriedenstellenden Ergebnisse erzielen - sowohl klanglich als auch mathematisch (siehe Kapitel 5.5.1). Bessere Ergebnisse ließen sich im Zuge dieser Arbeit durch das Trainieren mit der Amplitude des Signals (Betrag der komplexen Zahlen) erzielen (siehe Kapitel 5.5.3), was jedoch zunächst Informationen beseitigt, die zur Wiederherstellung mittels der IFFT benötigt werden. Das Trennen von Amplitude und Phase und das anschließende Verrechnen ermöglicht zwar die Durchführung der IFFT, allerdings wird ein KI modifiziertes Signal demnach mit der Phaseninformation des unmodifizierten Signals verrechnet (siehe Kapitel 5.5.2). Eine vollständigere Signalwiederherstellung ließe sich höchstwahrscheinlich mit einem Netzwerk, das komplexe Zahlen verarbeiten kann, erzielen.
Das mathematisch korrekteste Ergebnis konnte durch die Verlustfunktion Mean Squared Error und die Aktivierungsfunktion Leaky ReLU erzielt werden. Doch auch trotz des guten Differenzwerts zum Studiomikrofon sind auch hier Artefakte hörbar, die den Klang stark beeinflussen und in dem aktuellen Stadium den Mehrwert des Algorithmus nichtig machen. Nichtsdestotrotz konnte im Zuge dieser Arbeit der Frequenzgang des Telefonsignals an den des Studiomikrofons angenähert werden.
Weitere Forschung ist nötig, um vor allem die Artefakte zu beseitigen. Durch ein Netzwerk, das mit komplexen Zahlen umgehen kann, durch alternative Netzwerkarchitekturen, durch einen umfangreicheren Trainingsdatensatz und durch einen stärkeren Fokus auf den Trai-

ningsprozess werden höchstwahrscheinlich bessere Ergebnisse erzielt werden können (siehe Kapitel 6). Perspektivisch wäre vorstellbar, dass solch ein Algorithmus bei sämtlichen Telefonaten in Echtzeit mitläuft und den Nutzern ein angenehmeres Hörerlebnis bietet.

Wie in Kapitel 3.1.1 bereits in einem kleinen Ausschnitt zu sehen war, ist das Potenzial von Künstlicher Intelligenz groß. Manch einem kommt dies gar beängstigend vor. Selbst Elon Musk, Chef der Firmen Tesla und Twitter - ein Mann, der sonst visionär nach vorne blickt - setzte sich zusammen mit KI-Forschern und Technikexperten (darunter auch Apple-Mitgründer Steve Wozniak) in einem offenen Brief für einen 6-monatigen Stopp für die Entwicklung großer KI-Systeme, um entsprechende rechtliche Grundlagen und Sicherheitsvorkehrungen treffen zu können[1].
Welche gesellschaftlichen Veränderungen ChatGPT und Co. noch mit sich bringen und welche Neuerungen es im Audiobereich geben wird, bleibt abzuwarten, aber es lässt sich mit Sicherheit sagen, dass der Mensch im laufenden Jahrhundert nicht zum letzten Mal angesichts der Fähigkeiten von KI gestaunt haben wird.

[1] Armbruster 2023.

Abbildungsverzeichnis

Literatur

Adobe (2023). *Das Werkzeug „Maskieren“*. URL: https://helpx.adobe.com/content/help/de/de/lightroom-classic/help/masking.html (besucht am 31. 03. 2023).

Alby, Tom (2008). *Das mobile Web: 3G, 3GP, 4G, ANDROID, EDGE, GSM, HSPA, IPHONE, LBS, PTT, UMTS, WAP, WCDMA, WIMAX, WML, WURFL*. München: Hanser. 220 S. ISBN: 978-3-446-41507-2.

Apple (3. Nov. 2022). *FaceTime mit dem iPhone oder iPad verwenden*. Apple Support. URL: https://support.apple.com/de-de/HT204380 (besucht am 10. 03. 2023).

Armbruster, Alexander (29. März 2023). “Elon Musk und Forscher fordern, die Entwicklung riesiger KIs zu stoppen”. In: *FAZ.NET*. ISSN: 0174-4909. URL: https://www.faz.net/aktuell/wirtschaft/digitec/elon-musk-und-forscher-fordern-die-entwicklung-riesiger-kis-zu-stoppen-18784251.html (besucht am 01. 04. 2023).

Chen, Nicholas u. a. (2016). “Global Economic Impacts Associated with Artificial Intelligence”. In.

Chokshi, Niraj (25. Feb. 2020). “Tesla Autopilot System Found Probably at Fault in 2018 Crash”. In: *The New York Times*. ISSN: 0362-4331. URL: https://www.nytimes.com/2020/02/25/business/tesla-autopilot-ntsb.html (besucht am 20. 03. 2023).

Clevert, Djork-Arné, Thomas Unterthiner und Sepp Hochreiter (22. Feb. 2016). *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. arXiv: 1511.07289[cs]. URL: http://arxiv.org/abs/1511.07289 (besucht am 20. 03. 2023).

Deotte, Chris (2018). *Digit Recognizer*. URL: https://kaggle.com/competitions/digit-recognizer (besucht am 24. 03. 2023).

Design, Blackmagic (2023). *DaVinci Resolve 18 – Color | Blackmagic Design*. URL: https://www.blackmagicdesign.com/de/products/davinciresolve/color (besucht am 31. 03. 2023).

DPA (22. Feb. 2021). *Fakten zur Sprachverständlichkeit*. DPA. URL: https://www.dpamicrophones.de/mikrofon-universitaet/fakten-zur-sprachverstaendlichkeit (besucht am 30. 03. 2023).

EU-Parlament (14. Sep. 2020). *Was ist künstliche Intelligenz und wie wird sie genutzt? | Aktuelles | Europäisches Parlament*. URL: https://www.europarl.europa.eu/news/de/headlines/society/20200827STO85804/was-ist-kunstliche-intelligenz-und-wie-wird-sie-genutzt (besucht am 28. 03. 2023).

Feuerriegel, Stefan, Mateusz Dolata und Gerhard Schwabe (1. Aug. 2020). "Fair AI". In: *Business & Information Systems Engineering* 62.4, S. 379–384. ISSN: 1867-0202. DOI: 10.1007/s12599-020-00650-3. URL: https://doi.org/10.1007/s12599-020-00650-3.

Friedrich, Hans Jörg (2008). *Tontechnik für Mediengestalter: Töne hören - Technik verstehen - Medien gestalten*. X.media.press. Berlin Heidelberg: Springer. 346 S. ISBN: 978-3-540-71869-7.

Ganesh, Satya (7. Sep. 2022). *What's The Role Of Weights And Bias In a Neural Network?* Medium. URL: https://towardsdatascience.com/whats-the-role-of-weights-and-bias-in-a-neural-network-4cf7e9888a0f (besucht am 28. 03. 2023).

Gillham, Jonathan u. a. (Feb. 2018). "The macroeconomic impact of artificial intelligence". In.

Github (o. D.). *Looking inside neural nets*. URL: https://ml4a.github.io/ml4a/looking_inside_neural_nets/ (besucht am 24. 03. 2023).

Guillot, Jaume Duch (29. Sep. 2020). *Künstliche Intelligenz: Chancen und Risiken | Aktuelles | Europäisches Parlament*. URL: https://www.europarl.europa.eu/news/de/headlines/society/20200918STO87404/kunstliche-intelligenz-chancen-und-risiken (besucht am 20. 03. 2023).

Hancke, Philipp (14. Juni 2021). *FaceTime finally faces WebRTC - implementation deep dive*. webrtcHacks. URL: https://webrtchacks.com/facetime-finally-faces-webrtc-implementation-deep-dive/ (besucht am 30. 03. 2023).

Helmholtz (24. Jan. 2022). *Wie KI die Medizin revolutioniert*. Helmholtz-Gemeinschaft Deutscher Forschungszentren. URL: https://www.helmholtz.de/newsroom/artikel/wie-ki-die-medizin-revolutioniert/ (besucht am 16. 03. 2023).

Heuberger, Prof. Dr.-Ing. Albert und Prof. Dr.-Ing. Bernhard Grill (März 2017). *FraunhoferIIS_Technica Paper_EVS_Bandwidth Phone Calls.pdf*.

iZotope (2023). *Ozone 10—Audio Mastering Software*. iZotope. URL: https://www.izotope.com/en/products/ozone.html (besucht am 16. 03. 2023).

Jahn, Thomas (22. März 2023). *KI von OpenAI: Wie ChatGPT funktioniert*. URL: https://www.handelsblatt.com/technik/it-internet/ki-von-openai-wie-chatgpt-funktioniert/28941524.html (besucht am 28. 03. 2023).

Jüngling, Thomas (15. Okt. 2015). *Warum uns Digitalklang nach Komprimierung so nervt - WELT*. DIE WELT. URL: https://www.welt.de/wissenschaft/article120646901/Warum-uns-komprimierter-Digitalklang-so-nervt.html (besucht am 12. 03. 2023).

Keras, Team (o. D.). *Keras documentation: Conv1D layer*. URL: https://keras.io/api/layers/convolution_layers/convolution1d/ (besucht am 24. 03. 2023).

Klimaschutz, BMWK-Bundesministerium für Wirtschaft und (6. Dez. 2022). *EU-Verordnung zu Künstlicher Intelligenz*. URL: https://www.bmwk.de/Redaktion/DE/Pressemitteilungen/2022/12/20221206-zitat-bundesminister-robert-habeck-eu-verordnung-zu-kunstlicher-intelligenz.html (besucht am 20. 03. 2023).

Librosa (2023). *librosa.load — librosa 0.10.1dev documentation*. URL: https://librosa.org/doc/main/generated/librosa.load.html (besucht am 15. 03. 2023).

Mayer, Jörg (2023). *Sprache und Gehirn | Anmerkungen zur Physiologie*. URL: https://www2.ims.uni-stuttgart.de/sgtutorial/physiologie.html (besucht am 31. 03. 2023).

McKinsey (o. D.). *McKinsey Market Prediction.pdf*.

Mnih, Volodymyr u. a. (19. Dez. 2013). *Playing Atari with Deep Reinforcement Learning*. arXiv: 1312.5602[cs]. URL: http://arxiv.org/abs/1312.5602 (besucht am 21. 03. 2023).

Oerding, Henrik und AP (24. Jan. 2023). "Fake News: ChatGPT schreibt überzeugende Falschinformation". In: *Die Zeit*. ISSN: 0044-2070. URL: https://www.zeit.de/digital/2023-01/chatgpt-fake-news-desinformation-newsguard?utm_referrer=https%3A%2F%2Fwww.google.com%2F (besucht am 20. 03. 2023).

Paass, Gerhard und Dirk Hecker (2020). *Künstliche Intelligenz: was steckt hinter der Technologie der Zukunft?* Wiesbaden [Heidelberg]: Springer Vieweg. 496 S. ISBN: 978-3-658-30210-8. DOI: 10.1007/978-3-658-30211-5.

Patel, Devdhar u. a. (Dez. 2019). "Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to Atari Breakout game". In: *Neural Networks* 120, S. 108–115. ISSN: 08936080. DOI: 10.1016/j.neunet.2019.08.009. URL: https://linkinghub.elsevier.com/retrieve/pii/S0893608019302266 (besucht am 21. 03. 2023).

Primack, Brian A. u. a. (Apr. 2017). "Use of multiple social media platforms and symptoms of depression and anxiety: A nationally-representative study among U.S. young adults". In: *Computers in Human Behavior* 69, S. 1–9. ISSN: 07475632. DOI: 10.1016/j.chb.2016.11.013. URL: https://linkinghub.elsevier.com/retrieve/pii/S0747563216307543 (besucht am 20. 03. 2023).

RAMACHANDRAN, VIGNESH (23. Feb. 2021). *Four causes for 'Zoom fatigue' and their solutions*. Stanford News. Section: Social Sciences. URL: https://news.stanford.edu/2021/02/23/four-causes-zoom-fatigue-solutions/ (besucht am 09. 03. 2023).

Ramsundar, Bharath und Reza Bosagh Zadeh (2018). *TensorFlow for deep learning: from linear regression to reinforcement learning*. First edition. OCLC: on1030582228. Beijing: O'Reilly Media. 240 S. ISBN: 978-1-4919-8045-3.

Redaktion, BR24 (25. Juni 2022). *Offenbar Deep Fake: Giffey telefonierte mit "falschem"Klitschko*. BR24. URL: https://www.br.de/nachrichten/deutschland-welt/offenbar-deep-fake-giffey-telefonierte-mit-falschem-klitschko,T9k0jTj (besucht am 20. 03. 2023).

Robinson, Rob (7. Apr. 2017). *Convolutional Neural Networks - Basics · Machine Learning Notebook*. URL: https://mlnotebook.github.io/post/CNN1/ (besucht am 24. 03. 2023).

Ronneberger, Olaf, Philipp Fischer und Thomas Brox (18. Mai 2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv: 1505.04597[cs]. URL: http://arxiv.org/abs/1505.04597 (besucht am 04. 03. 2023).

Roscher, Karsten, Anna Guderitz und Hans-Thomas Hengl (2023). *Künstliche Intelligenz (KI) und maschinelles Lernen - Fraunhofer IKS*. Fraunhofer-Institut für Kognitive Systeme IKS. URL: https://www.iks.fraunhofer.de/de/themen/kuenstliche-intelligenz.html (besucht am 16. 03. 2023).

Rump, Jutta und Marc Brandt (Dez. 2020). *IBE-Studie-Zoom-Fatigue.pdf*.

Saha, Sumit (16. Nov. 2022). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Medium. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (besucht am 24. 03. 2023).

Sam Altman [@sama] (14. März 2023). *here is GPT-4, our most capable and aligned model yet. it is available today in our API (with a waitlist) and in ChatGPT+. https://openai.com/research/g 4 it is still flawed, still limited, and it still seems more impressive on first use than it does after you spend more time with it.* Twitter. URL: https://twitter.com/sama/status/1635687853324902401 (besucht am 20. 03. 2023).

Sanderson, Grant (2018). *Was macht Backpropagation wirklich? | Kapitel 3, "Deep Learning-YouTube*. URL: https://www.youtube.com/ (besucht am 23. 03. 2023).

SAP (2023). *Was ist maschinelles Lernen? | Definition, Arten, Beispiele | SAP Insights*. SAP. URL: https://www.sap.com/germany/insights/what-is-machine-learning.html (besucht am 16. 03. 2023).

Schwaiger, Roland und Joachim Steinwendner (2019). *Neuronale Netze programmieren mit Python*. 1. Auflage, 1. korrigierter Nachdruck. Rheinwerk Computing. Bonn: Rheinwerk Verlag. 447 S. ISBN: 978-3-8362-6142-5.

Sharma, Siddharth, Simone Sharma und Anidhya Athaiya (10. Mai 2020). "ACTIVATION FUNCTIONS IN NEURAL NETWORKS". In: *International Journal of Engineering Applied Sciences and Technology* 04.12, S. 310–316. ISSN: 24552143. DOI: 10.33564/IJEAST.2020.v04i12.054. URL: https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf (besucht am 20. 03. 2023).

Sibi, P, S Allwyn Jones und P Siddarth (2013). "ANALYSIS OF DIFFERENT ACTIVATION FUNCTIONS USING BACK PROPAGATION NEURAL NETWORKS". In: . *Vol.* 47.

Sickert, Teresa (24. März 2016). "Microsoft: Twitter-Bot Tay - vom Hipstermädchen zum Hitlerbot". In: *Der Spiegel*. ISSN: 2195-1349. URL: https://www.spiegel.de/netzwelt/web/microsoft-twitter-bot-tay-vom-hipstermaedchen-zum-hitlerbot-a-1084038.html (besucht am 20. 03. 2023).

Strahlenschutz, Bundesamt für (2023). *Was ist Mobilfunk?* Bundesamt für Strahlenschutz. Publisher: BfS. URL: https://www.bfs.de/DE/themen/emf/mobilfunk/basiswissen/basiswissen.html (besucht am 30. 03. 2023).

Udofia, Udeme (20. Sep. 2019). *Basic Overview of Convolutional Neural Network (CNN)*. DataSeries. URL: https://medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17 (besucht am 24. 03. 2023).

Weinzierl, Stefan und Verband Deutscher Tonmeister, Hrsg. (2008). *Handbuch der Audiotechnik*. Berlin: Springer. 1197 S. ISBN: 978-3-540-34300-4.