HAW HAMBURG

BACHELORTHESIS
Marcel Tuleweit

Klassifikation auf Basis von EEG-Daten, automatisiert durch maschinelles Lernen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Marcel Tuleweit

Klassifikation auf Basis von EEG-Daten, automatisiert durch maschinelles Lernen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Tim Tiedemann
Zweitgutachter: Prof. Dr. Thomas Clemen

Eingereicht am: 03.03.2020

Marcel Tuleweit

Thema der Arbeit

Klassifikation auf Basis von EEG-Daten, automatisiert durch maschinelles Lernen

Stichworte

Maschinelles Lernen, Elektronenzephalographie, EEG Ableitungen, Deep Learning, Clusteranalyse, Klassifikation, Multilayer Perceptron, Convolutional Neural Network, Long Short-Term Memory, Hyperparameter Tuning

Kurzzusammenfassung

Die Eignung von Machine Learning Verfahren im Bereich der Medizin ist bereits bewiesen. Inwiefern sich ein einzelnes Verfahren für die Problemlösung des jeweiligen Anwendungsbereiches eignet, ist meist von dem zu lösenden Problem abhängig und nicht zuletzt auch von dem Charakteristikum des gewählten Verfahrens. Im Bereich der Elektroenzephalographie ist diese Eignung bereits durch viele Verfahren gezeigt worden. Die involvierten Machine Learning Verfahren gilt es in einer hinreichenden Analyse und anhand eines Beispiels gegenüberzustellen und ihre Vor- und Nachteile näher zu beleuchten.

Marcel Tuleweit

Title of Thesis

Classification based on EEG data automated by machine learning

Keywords

Machine Learning, Electroenzephalography, EEG Montages, Deep Learning, Clustering, Classification, Multilayer Perceptron, Convolutional Neural Network, Long Short-Term Memory, Hyperparameter Tuning

Abstract

The suitability of machine learning procedures in the field of medicine has already been proven. The extent to which a single method is suitable for solving the problem of the respective application area usually depends on the problem to be solved and, last but not least, on the characteristic of the chosen method. In the field of electroencephalography, this suitability has already been demonstrated by many procedures. The machine learning procedures involved must be compared in a sufficient analysis and on the basis of an example and their advantages and disadvantages must be examined in more detail.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Maschinelles Lernen erhält immer mehr Einzug in unser alltägliches Leben. Algorithmen, welche sich dieses zu Nutze machen, umgeben uns sowohl beim Surfen im Internet als auch beim Bearbeiten unserer Bilder mittels Bildbearbeitungsprogrammen. Alles in allem haben sie eines gemeinsam: Sie sind schnell, in dem was sie tun. Waren Computer einst kaum dazu in der Lage, einfache mathematische Aufgaben zu lösen, so berechnen sie heute die komplexesten in Bruchteilen von Sekunden. Die Rechenkraft heutiger Prozessoren übersteigt die eines Menschen bei weitem. Vermutlich daher werden statistische Auswertungen und Prognosen heutzutage oft von Computern getroffen werden. Zumeist sind diese für uns Menschen nicht mehr nachvollziehbar oder aus zeitlichen Gründen nicht überprüfbar.
Auch im Bereich der Medizin, in welchem es nicht selten um das Überleben von Menschen geht, wurden maschinelle Lernverfahren längst bemerkt und ihre Anwesenheit geschätzt. Oftmals entwickeln sich allerdings auch Ängste und Befürchtungen, die von aktuellen Science Fiction Filmen bestärkt, ethische Fragen diesbezüglich in den Raum stellen. Aus der Frage, was mit der Hilfe von maschinellem Lernen wohl alles möglich wäre, wird somit schnell die Frage, was möglich sein darf. In Hinblick auf die Entwicklung in diesem Bereich schürt sich bei manchen auch die Angst vor der absoluten Kontrolle oder dem Verlust der Privatsphäre. Von den Resultaten getrieben, erhofft sich der ein oder andere einen riesigen Schritt, auch hinsichtlich der medizinischen Diagnosemöglichkeiten, zu dem wir als Menschen schlichtweg nicht möglich wären.
Im Rahmen dieser Ausarbeitung wird ein Anwendungsfall im Bereich der Medizin aufgezeigt, welcher von maschinellem Lernen durchaus profitieren kann. Ob und in welchem Maße maschinelle Lernverfahren unsere Ergebnisse übertreffen können, kann nur gemutmaßt werden. Die Resultate jedoch werden einen Eindruck darüber gewähren, was maschinelles Lernen realistisch leisten kann. Hierbei ist zu beachten, dass etwaige vorgestellte Verfahren mit den nötigen Ressourcen und zeitlichen Kapazitäten zu noch weit mehr im Stande wären, wodurch die Auswertung letztlich keine Grenze setzt. Stattdessen soll sie ein Gefühl für die eingesetzten Verfahren vermitteln und den Horizont der Möglichkeiten aufzeigen.

2 Motivation

2.1 Anwendungsfall

Verkündet wurde das Interesse an der Eignung maschineller Lernverfahren für den Einsatzbereich der Anfallserkennung im Bereich der Elektroenzephalographie durch die Institution des Universitätsklinikums Hamburg-Eppendorf (UKE). Im Zuge dessen waren die Bestrebungen ebenfalls damit begründet, dass zu diesem Zeitpunkt lediglich Experten die Resultate eines EEGs auswerten, um eine Diagnose zu stellen. Generell schien das Interesse an künstlicher Intelligenz hinsichtlich dieses Anwendungsgebiets geweckt. Einzig die Frage nach realistisch zu erreichenden Resultaten galt bisher als unbeantwortet. Mit Verlauf dieser Arbeit soll dieser Fragestellung eine hinreichende Analyse zu Grunde gelegt werden, welche es ermöglicht, eine Inferenz hinsichtlich der praktischen Einsatzmöglichkeiten und Eignungen einzelner Verfahren zu ziehen. Aufgrund der Datenschutz-Grundverordnung (DSGVO) war es dem UKE selbst nicht möglich, diesbezüglich Daten zur Verfügung zu stellen. Eine gemeinsame Korrespondenz ließ jedoch einen Eindruck dessen gewinnen, in welcher Form diese Daten nach einer Messung vorliegen könnten. Dies ist entscheidend für spätere Vorverarbeitungsprozesse, welche die Daten für einzelne Verfahren aufbereiten müssen. Auch hinsichtlich der für die Durchführung notwendigen Informationen, wie bspw. dem Expertenwissen, muss dargestellt werden, in welcher Form diese zugänglich wären. Die Kernaspekte einer erfolgreichen Diagnose sollen im Folgenden grundlegend näher betrachtet werden, da sie das Fundament für die Umsetzung mittels maschineller Lernverfahren bilden.

2.1.1 Elektroenzephalographie

Wie andere Organe im Körper erzeugt auch das Gehirn elektrische Signale, die sich mittels Elektroden am Körper ableiten lassen. Diese Messung wird als Elektroenzephalogramm (EEG) bezeichnet. Beim EEG werden die Elektroden an bestimmten Stellen des Kopfes angebracht und über Kabel mit einem EEG-Gerät verbunden. Die Elektroden messen die Aktivität des Gehirns, die dann als Kurve auf einem Monitor dargestellt wird. (IQWiG 2019) Dieser Vorgang ist uns bekannt

als Elektroenzephalographie und wird zumeist ebenfalls mit den Buchstaben EEG abgekürzt. Unumstritten ist der Bereich der Elektroenzephalographie heute ein wichtiger Bestandteil der Neurologie. Die dabei gesammelten Daten einer Messung helfen bspw. bei der Diagnose von Hirn-Erkrankungen, Funktionsstörungen im Gehirn, oder Tumoren. (Reiche 2014) Ein Beispiel für die Hirn-Erkrankungen bietet die Epilepsie, welche sich durch das Auftreten nicht provozierter epileptischer Anfälle charakterisieren lässt. Hier bildet das EEG die „Untersuchungsmethode der ersten Wahl […]“ (Reiche 2014) Für eine Diagnose wären zwei nicht provozierte Anfälle oder das Auftreten von epilepsietypischen Potenzialen nach einem nicht provozierten Anfall notwendig. Epilepsietypische Muster lassen sich wiederum mit einem EEG erkennen. (Block 2019)
Es wird grundsätzlich zwischen fokalen, primär generalisierten und nicht klassifizierbaren Anfällen unterschieden. (Sturm, Biesalski und Höffken 2019) Die bei einem Anfall im EEG auftretenden Muster sind als Wellenformen wie Spitzen („spikes“), scharfe Wellen („sharp-waves“), Polyspikes oder Spitzen-Wellen-Komplexe (Spike-wave Komplexe) erkennbar. (Kursawe 2018) Die Zuverlässigkeit im Erkennen solcher Potenziale liegt demnach bei dem Experten, welcher die Elektroenzephalographie durchführt oder deren Auswertung im Nachhinein betrachtet.
Die Amplituden der EEG-Wellen liegen bei Ableitungen von der Kopfoberfläche größtenteils unter 100 µV und werden durch die Gewebsschichten zwischen Hirn und Kopfoberfläche stark reduziert. Wird die Ableitung hingegen direkt auf der Hirnoberfläche vorgenommen, erhöht sich der Wert auf das 5- bis 10-fache, also auf 500 bis 1.000 µV. Die Spannungshöhe bildet hierbei das Ergebnis der Potenzialdifferenz an den Eingängen des Differenzverstärkers an jedem EEG-Kanal und wird durch die Wahl der Elektrodenverschaltung bestimmt. Diese Verschaltungen werden als Ableitungen behandelt. (Hansen 2012)

2.1.2 Ableitungen im EEG

Durch die Verschaltung der einzelnen Elektroden untereinander ergeben sich die Ableitungen eines Elektroenzephalogramms. Diese Ableitungen werden oftmals auch als Montagen bezeichnet. Eine sinnvolle Ableitung ist hierbei die Grundlage für das zuverlässige Erkennen von Potenzialen. Wie im Folgenden näher erläutert wird, sind Anfallspotenziale je nach Ableitung prägnanter oder auch gar nicht ersichtlich.
In sogenannten Referenzableitungen werden die Elektroden jeweils zu einem gemeinsamen Referenzpunkt abgeleitet. Dieser Referenzpunkt kann sowohl durch eine natürliche Referenz, mit einer am Patienten angebrachten Elektrode, oder durch eine technisch hergestellte Referenz erfolgen. Letzteres wäre bspw. die Durchschnitts- oder Mittelwertreferenz. Bei der natürlichen Referenz wird nach einem Bezugspunkt gesucht, der durch die Hirnaktivität nicht belastet wird und somit als potenzialfreier Ableitpunkt fungiert. Idealerweise wäre dieser Punkt weit vom Hirn ent-

fernt und somit im Unterkörper. Jedoch würde eine EEG-Ableitung zu diesem Bezugspunkt das Elektrokardiogramm miterfassen, wodurch die Wahl des gemeinsamen Referenzpunkts sich auf die Lokation im Bereich des Kopfes beschränkt. Während Referenzpunkte im Bereich des Nackens oder Kinns durch einstreuende Muskelartefakte belastet werden würden, haben sich die Ohrläppchen als Bezugspunkt durchgesetzt. Dadurch, dass bei der Referenzableitung die Elektroden an einem gemeinsamen Bezugspunkt abgeleitet werden, würden an diesem Referenzpunkt auftretende Artefakte in alle EEG-Kanäle eingeschleust werden. (Hansen 2012)
Anders als bei den Referenzableitungen, entfällt bei den bipolaren Ableitungen der gemeinsame Bezugspunkt. Hierbei werden stattdessen reihenweise jeweils zwei differente Elektroden miteinander verschaltet. Die Potenzialdifferenzen zwischen den einzelnen Elektroden sind hinsichtlich der Referenzableitungen vergleichsweise gering. Dies ist vor allem darauf zurückzuführen, dass im Normalfall die Hirnrindenaktivität regional relativ gut synchronisiert ist und sich das Ausgangssignal auf zwei benachbarte Elektroden bezieht. Somit werden die EEG-Wellen in bipolaren Montagen mit reduzierter Amplitude erfasst. Potenzialdifferenzen würden bei dieser Verschaltung entstehen, wenn eine Elektrode sich in der Region der Herdstörung befindet, während die andere außerhalb des Bereiches läge. Sollten sich die benachbarten Elektroden beide im Herdbereich befinden und die Störung in gleicher Form und Ausprägung erfassen, würde sich die Hirnaktivität in der Differenz ausgleichen. Des Weiteren haben Fokale, die sich unter einer Elektrode befinden, zur Folge, dass sich in beiden benachbarten Kanälen, in welchen die Elektrode einen Ableitpunkt repräsentiert, eine Phasenumkehr der abgeleiteten EEG-Wellen bildet. Diese wird aufgrund der Nachbarschaft als enge Phasenumkehr bezeichnet. Diese Phasenumkehr unterstützt die visuelle Analyse einer Elektroenzephalographie. Grundsätzlich sind bipolare Montagen durch den Bezug auf benachbarte Elektroden im Vorteil, was die visuelle Darstellung des EEGs angeht. Durch den fehlenden Referenzpunkt sind bipolare Ableitungen generell weniger artefaktbelastet. Artefakte verschmutzen die Hirnaktivität während der Aufzeichnung, da sie als Potenzialschwankungen registriert werden, aber nicht vom Hirn ausgehen. Es ergeben sich jedoch ebenfalls auch Nachteile gegenüber der Referenzableitung. Die Amplitude der EEG-Wellen wird reduziert und Potenziale werden möglicherweise mit falscher Polarität oder in veränderter Form dargestellt. (Hansen 2012)
Da die verfälschte Darstellung von Potenzialen grundsätzlich keine Auswirkung auf die Erkennbarkeit hat, sofern sie hierdurch nicht schwerer zu erkennen sind, wird für die Problemstellung dieser Arbeit auf die bipolare Ableitung zurückgegriffen. Auch bei bipolaren Ableitungen gibt es mehrere Möglichkeiten der Verschaltung. Gemeinsam haben sie alle jedoch die reihenweise Verschaltung zweier differenter Elektroden. Um die Dimensionen etwas zu reduzieren, wird hier nur die verwendete Ableitung vorgestellt. Dabei handelt es sich um die „enge bipolare Reihe“, deren Bezeichnung sich durch ihren engen Elektrodenabstand ergibt. Wie in der Abbildung 2.1

ersichtlich, werden hierbei zwei Reihen gebildet, die in ihrer Verschaltung zwei bananenförmige Muster ergeben, weswegen sie auch gern als „Doppelbanane“ bezeichnet wird.

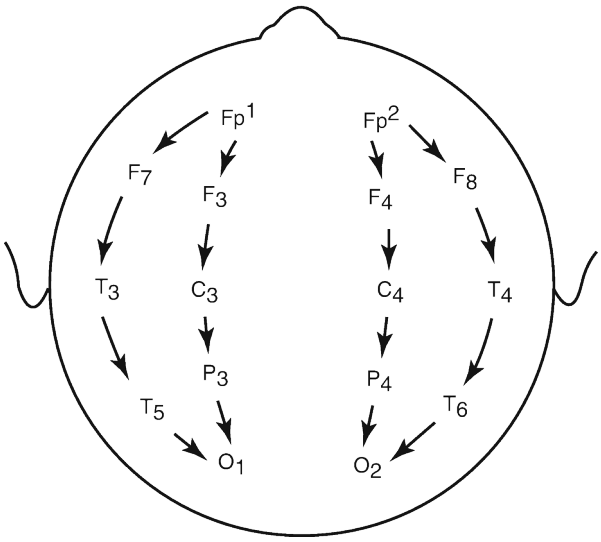

Abbildung 2.1: Bipolare Reihenmontage
(Sazgar und Young 2019)

2.2 Ziel

Es liegt also auf der Hand, dass es in diesem Umfeld besonders wichtig ist, Diagnosen richtig zu treffen und damit einhergehend Anzeichen richtig zu deuten. Für gewöhnlich ist eine solche Diagnose mit einer hohen Verantwortung belastet. Experten müssen Entscheidungen darüber treffen, ob Anzeichen von Erkrankungen, sogenannte Potentiale, gewürdigt werden oder zu vernachlässigen sind. Dies kann auch immer zu einer Reihe von Fehldiagnosen führen, sei es durch die falsche Einschätzung vom Experten selbst, oder durch nicht ausreichend ausgeprägte Anzeichen. Eine Fehldiagnose kann für den Patienten jedoch verheerende Folgen haben. Ein Experte kann zudem immer auf einen Fall treffen, der ihm vorher in dieser Form noch nie begegnet ist. Auch dies kann zu einer Fehleinschätzung führen, wie durch das Missachten eines vorliegenden Potenzials oder durch einen Fehlalarm im unkritischen Zustand. Während eines epileptischen Anfalls ändert sich die Hirnaktivität, wodurch eine Abweichung gegenüber dem normativen Zustand erkennbar wird. Die Entscheidung darüber, ob dieses Anzeichen schlussendlich zu einer Diagnose führt, obliegt jedoch weiterhin dem Experten.
Das Interesse einer medialen Einrichtung, welche EEG-Messungen durchführt, geht somit in die

Richtung der kompromissfreien Prädikation des Zustandes des jeweiligen Probanden. Sollte ein Anfallspotential vorliegen, so soll dies gewährleistend erkannt werden können, um die Einschätzung eines Experten zuverlässig ersetzen zu können.
Die Essenz dieser Arbeit liegt also darin, das Wissen eines Experten, welches für die Diagnose essenziell ist, zu extrahieren und einer Ressource zu vermitteln, die geringfügigeren Aufwand oder Kosten erfordert, als der Experte selbst es würde. Fachlich korrekt ausgedrückt, soll eine künstliche Intelligenz darauf trainiert werden, die Erfahrungen eines Experten zu sammeln, um in neuen, künftigen Situationen auf diese Erfahrung zurückgreifen zu können und somit selbst die Diagnose einer möglichen Hirnerkrankung stellen, oder zumindest empfehlen, zu können. Dieses Ziel erstreckt sich jedoch über weit mehr als diesen Kerngedanken. Ferner muss bspw. gewährleistet werden, eine Erkrankung mindestens in jedem Fall zu erkennen, indem auch ein Experte sie erkannt hätte. Die Bestrebungen gehen allerdings in die Perfektion, was bedeuten würde, eine Erkrankung in jedem Fall zu erkennen, sollte sie denn vorliegen. Auch müssen gewisse Maximalwerte von Fehl-Alarmen abgeschätzt werden. Jedem einzelnen Patienten eine Erkrankung zu diagnostizieren würde auch ausschließen, eine Erkrankung zu übersehen, ist aber keinesfalls zielführend. Ziel jedoch ist es, die Anzeichen richtig zu deuten und auch in Grenzfällen genau unterscheiden zu können. Dafür ist es unumgänglich, für das Gewinnen der Erfahrung eine Anzahl an Referenzdaten zu besitzen, deren Diagnose vorliegt und gewährleistet werden kann. Anhand dieser Daten wird die künstliche Intelligenz auf die Anzeichen sensibilisiert und lernt, Schwellwerte für Grenzen zu setzen und Merkmale entsprechend zu klassifizieren.

2.3 Meta-Daten

Um eine Klassifikation mit maschinellen Lernverfahren zu automatisieren, sind ausreichend Trainingsdaten im Vorfeld zu sammeln. Für die Klassifizierung von Daten, die einer Elektroenzephalographie entstammen, sind die während der Messung aufgenommenen Werte die Basis des Trainings. Diese Werte, gemessen im Millivolt (mV), enthalten die Informationen über den Gesundheitszustand des Gehirns des Probanden, deren Merkmale im Folgenden abstrahiert werden sollen. Des Weiteren ist die Information über die gestellte Diagnose erforderlich, welche im Folgenden auch als Label bezeichnet wird. Diese Daten gelten bereits als zuverlässig klassifiziert. Personenbezogene Daten, wie das Alter oder Geschlecht des Patienten, können bei einer weiteren Analyse der Häufigkeitsverteilung interessant werden, sind jedoch für die reine Klassifizierung erlässlich. Der Zeitpunkt einer Messung ist nur dann von Relevanz, wenn er für weitere Analysen gebraucht wird. Für das Erkennen des Vorliegens eines Anfalls bietet der Zeitpunkt keinerlei Mehrwert. Während einer Messung kann jedoch beispielweise durch den durchführenden Arzt ein Anfall provoziert werden und in diesem Fall wäre relevant, ob die im Elektroenzephalo-

gramm erkennbaren Muster dem Zeitpunkt der Provokation zugeordnet werden können. Wichtig für die Diagnose ist bei der Elektroenzephalographie auch die Reihenfolge der aufgezeichneten Messwerte. Diese kann nicht willkürlich verändert werden, weil dies das Ergebnis beeinflussen würde.

3 Referenzen

3.1 Literaturwerke

Um effektive Deep-Learning Anwendungen konzipieren zu können, ist es essenziell, sich mit aktuellen „Best practices“ auseinanderzusetzen. Diese lassen sich zwar dem Internet entnehmen, jedoch bietet gerade Literatur diesbezüglich eine Ansammlung von Wissen, welche zusammengefasst, aufbereitet und an einem Ort zugänglich gemacht wurde. Dies war fundamental, um sich der Konzeption und Entwicklung einer Deep-Learning Architektur auseinanderzusetzen. Eine sehr umfangreiche Literatur bot hier das Werk von François Chollet mit dem Namen *„Deep Learning mit Python und Keras“*, welches sowohl die Grundidee hinter den jeweiligen Architekturen vermittelt als auch die Umsetzung der Ideen im Rahmen einer Implementierung. François Chollet ist seines Zeichens Autor von Keras, einem führenden Deep Learning Framework für Python. (Chollet u. a. 2015) Diese Arbeit richtet sich nach den Architekturen, welche François Chollet in seinem Werk grundlegend beschreibt und im Zuge dessen wird ebenfalls auf das Keras Framework zurückgegriffen. Hinsichtlich des Grundgedankens jener Architekturen und ihrer vorzugsweisen Einsatzgebieten würde dieses Werk allein bereits ausreichend Informationen bereitstellen. Um jedoch die Kernkompetenzen einzelner Systeme für die Eignung zur Lösung der eigenen Problemstellung gegenüberzustellen, bedarf es weitreichendere Recherchen. Auch das Herausstellen der Grenzfälle und der mit der Architektur einhergehenden Nachteile lässt ein tieferes Verständnis für das eingesetzte Verfahren zur Voraussetzung werden.
In zahlreichen Werken und Fachbeiträgen zu diversen dieser Themen lässt sich zumeist zu jeder Frage die entsprechende Antwort finden. Wenn ein Verfahren Schwachstellen zeigt, lassen sich die Gründe dafür schnell herauskristallisieren und anhand einer Vielzahl von Vorschlägen auch Vorkehrungen oder gar Alternativen finden, welche den Problemen der Architektur vorbeugen. Um sich eine Meinung über die Eignung von Machine Learning Verfahren für ein Problem aus dem Bereich der Neurologie bilden zu können, war es zudem unerlässlich, auch diese Kompetenzen weiter auszuprägen. Die Neurologie als solches würde mit ihrem Umfang an Wissen den Rahmen dieser Arbeit ohne jeden Zweifel sprengen. Daher wurde explizit nach Literatur gesucht,

welche die Kernfragen der Problemstellung der Elektroenzephalographie zu klären vermag und im Idealfall einen Grundgedanken für die Herangehensweise an das Problem liefert. Es stellte sich das Werk „Klinische Elektroenzephalographie“ von Hansen Zschocke als besonders hilfreich heraus, da es sämtliches Grundwissen rund um die Elektroenzephalographie mit sich brachte. Besonders für das Themengebiet der Montagen eines EEG bot dieses Werk einen detaillierten Einblick in die zahlreichen Möglichkeiten und deren Umsetzung. Mit Hilfe der gefundenen Literatur war es möglich, sich in das Anwendungsfeld hineinzuversetzen, die Problemlösung aus anderen Blickwinkeln zu betrachten und einzelne Aspekte gegebenenfalls zu überdenken. So auch die Priorisierung hinsichtlich der zu erreichenden Fehlertoleranzen.

3.2 Heutiger Stand der Technik

Maschinelle Lernverfahren sind im Bereich der Elektroenzephalographie schon seit Jahren kein Fremdwort mehr. Das Internet ist überfüllt von etwaigen Ansätzen, welche mit Hilfe von maschinellem Lernen Aussagen über zugrundeliegende Daten treffen. Anders als in anderen Anwendungsbereichen jedoch, wirft eine künstliche Intelligenz besonders im medizinischen Umfeld auch rechtliche und ethische Fragen auf. Neben dem Aspekt des Datenschutzes geht es hierbei vor allem auch um Verantwortung und Transparenz. (BMBF 2019) Die Frage nach dem Verantwortungsbewusstsein einer KI ist vermutlich eine allgegenwärtige, die jeden von uns schon einmal beschäftigt hat. Sei es im Bereich des autonomen Fahrens oder Fliegens oder gar in alltäglichen Situationen wie bspw. im Gespräch mit dem eigenen Sprachassistenten. Gerade in der Medizin jedoch, wo es nicht selten um das Überleben von Menschen geht, wird das Thema der künstlichen Intelligenz besonders kritisch behandelt. Wie bereits im vorherigen Kapitel angedeutet, ist die Anforderung an eine KI sehr hoch, was vor allem daran liegt, dass sie keine Verantwortung tragen kann. Das Auftreten eines gesundheitskritischen Anfalls, welcher von der KI missachtet wurde, von einem Experten jedoch bemerkt worden wäre, hätte eine fatale Auswirkung auf das Vertrauen in die künstliche Intelligenz. „Obwohl wir Computer bauen können, die in der Lage sind, Schachweltmeister zu besiegen, sind wir weit davon entfernt, ein bewusstes Gehirn zu erschaffen“. (Lagercrantz 2019)
In der Sensortechnik hat sich zudem bereits einiges getan. Die Firma Cosinus GmbH, welche in München ansässig ist, entwickelte bereits einen im Ohr befestigten Minisensor, welcher mit Hilfe eines Smartphones die aufgezeichneten Signale an einen Rechner weiterleitet, um diese auf epileptische Anfälle zu überprüfen. Sowohl Ärzte als auch Angehörige könnten zudem über Ergebnisse informiert werden. (Jörg 2018)
Die Zuverlässigkeit eines maschinellen Lernverfahrens lässt sich jedoch messen, indem die Ausgabe der KI überprüft wirft und eventuelle Abweichungen vom tatsächlichen Zustand dokumen-

tiert werden. Der daraus errechenbare Genauigkeitswert einer KI, im Folgenden auch des Öfteren mit dem englischen Fachbegriff „Accuracy" bezeichnet, sollte natürlich im besten Fall bei 100% liegen. Dies ist jedoch, realistisch und kritisch gesehen, unwahrscheinlich zu erreichen. Dies kann allerdings auch nicht pauschal behauptet werden, da gewisse Anwendungsbereiche diesen Wert womöglich schon erreichen können, da sie bspw. immer exakt dieselben Bedingungen haben. Im medizinischen Umfeld ist aber davon auszugehen, dass sich jede Messung von der nächsten unterscheidet und die Bedingungen somit variieren. Basierend auf dieser Feststellung liefert eine Suche im Internet eine Vielzahl an fertigen Modellen, die bereits eine Genauigkeit von 97-99% vorweisen können.
Es sei an diesem Punkt gesagt, dass eine Vielzahl an existenten Modellen im Rahmen dieser Arbeit nicht betrachtet werden können, da ihre Anzahl den Umfang einer Eignungsanalyse bei weitem übersteigen würde. Neben den Verfahren, die im Verlauf dieser Arbeit vorgestellt werden und jenen, deren Eignung hinsichtlich der Problemstellung widerlegt wird, werden gegebenenfalls weitere Verfahren betrachtet und als Ausblick genannt.
Häufig im medizinischen Umfeld genannt wird zudem die künstliche Intelligenz namens Watson, welche von der Firma IBM geschaffen wurde. (Jörg 2018) (Gamestar: Wieselsberger 2018) Watson bezieht seine Informationen aus „[…] Lehrbüchern, Publikationen, klinischen Studien und Krankenakten". (Jörg 2018) Anhand dieser können dann seitens der KI Diagnosen und Therapien vorgeschlagen werden. Sie ist außerdem dazu in der Lage, sich selbstständig Informationen aus dem Internet zu beschaffen, um anhand ihrer Eingaben eine Vorhersage zu treffen, was sie wiederum lernfähig macht. Die Eingaben könnten bspw. sowohl Symptome und Beschwerden als auch Laborbefunde sein. Ärzte können somit innerhalb von Sekunden eine zweite Meinung zu getroffenen Entscheidungen erhalten. (Jörg 2018)
IBM beschreibt zudem den „[…] exponentiellen Anstieg an medizinischen Daten […]" (IBM: Weißmann und Deutsch 2016) als Problem für Experten, da es ihnen dadurch nahezu unmöglich wird, sich all jenes Wissen anzueignen. (IBM: Weißmann und Deutsch 2016)

3.3 Öffentlicher Datensatz

Um einen ersten Eindruck für Daten aus Elektroenzephalographien zu bekommen, wurde ein veröffentlichter Datensatz näher betrachtet. Bei dem ursprünglichen Datensatz handelte es sich um eine Sammlung von Messungen, die bei 500 verschiedenen Patienten durchgeführt worden waren. Die Daten sind in dieser Form öffentlich zugänglich, ohne weitere Metadaten über die Patienten zu enthalten. Sie enthalten lediglich die rohen Messdaten. Der Datensatz, welcher hier verwendet wurde, ist jedoch eine vorverarbeitete und umstrukturierte Version, die aus dem ursprünglichen Datensatz entstanden ist. Diese ist öffentlich zugänglich auf der Internetplattform

Kaggle.com, welche für ihre zahlreichen Beiträge zu Machine Learning und Data Science Themen bekannt ist. (Kaggle 2018) Die ursprünglichen Daten enthielten jeweils 4097 Messpunkte - im Folgenden als „Samples“ beschrieben - und entsprachen einer Messung über 23,5 Sekunden. Durch eine Zerlegung in einsekündige Intervalle, welche jeweils 178 Samples enthalten, wurden aus den vorherigen 500 Messdaten insgesamt 11500, was einen Gewinn von 11.000 Trainingsdaten bedeutet. Insgesamt enthält der Datensatz Messungen aus fünf Kategorien, welche jeweils als Klasse den entsprechenden Daten zugeordnet sind. Von diesen Kategorien gehören vier der fünf zu den Messungen, bei denen kein Anfall diagnostiziert wurde, während die fünfte Kategorie die Messung während eines Anfalls repräsentiert. In den vier anfallsfreien Kategorien wird hinsichtlich der Umstände unterschieden, unter denen gemessen wurde. So gibt es jeweils eine Kategorie für die Messung mit geöffneten und geschlossenen Augen des Patienten. Die anderen beiden Kategorien beziehen sich auf den Umstand einer vorherigen Tumordiagnose und beschreiben jeweils die Messung innerhalb und außerhalb des Bereiches, in welchem der Tumor zuvor saß. Die Verteilung zwischen den Kategorien ist gleichgewichtet, sodass jede Kategorie 20% der Daten enthält. Dadurch sind allerdings auch nur 20% der Daten Teil einer Messung, bei denen ein Anfall vorlag. Für die spätere Klassifizierung waren die anderen Kategorien nicht von Relevanz, weswegen Sie im ersten Schritt zusammengelegt wurden zu einer neuen Kategorie, welche alle Messungen in Abwesenheit eines Anfalls repräsentiert. Da sich der Datensatz bereits in einem vorverarbeiteten Zustand befand, konnte auf das Vorverarbeiten der Daten größtenteils verzichtet werden. Üblicherweise liegen Daten in Formaten vor, die im Vorfeld konvertiert werden müssten, oder weisen viele unnütze Informationen auf, die es dann entsprechend zu filtern gilt. In statistischen Anwendungsfällen kann es bspw. ebenfalls nötig sein, stark abweichende Ausreißer im Voraus aus dem Datensatz zu entfernen, um die Statistik nicht zu verschmutzen. Als Beispiel sei hier die Anzahl der Gewalttaten in den USA genannt, die mit Waffen begangen wurden. Ein einzelnes Ereignis, wie etwa ein Amoklauf oder ähnliches, würde die Gesamtstatistik zu stark beeinflussen, weswegen das Ereignis im Vorfeld gefiltert werden würde. Da im Bereich der EEG-Messungen jeder Ausreißer einer Messung entstammt und somit lediglich das breite Spektrum an Verhaltensweisen von Patienten während einer Messung untermalt, sind diese Daten für ein erfolgreiches Training unerlässlich.

4 Machine Learning - Verfahren

Um ein derartiges Problem zu lösen, eignen sich viele Verfahren, die es in einer hinreichenden Analyse gegenüberzustellen und zu vergleichen gilt. Der Erfolg eines Konzepts liegt allerdings nicht einzig in der Auswahl der Verfahren, sondern vielmehr in deren Umsetzung und Feinabstimmung. Das richtige Verfahren zu wählen, ist allerdings die Grundlage des Konzepts, da das Verfahren das Fundament einer Anwendung bildet. Ohnehin ist es gleich, wie perfekt ein Machine Learning Verfahren entworfen und getunt wurde: Wenn es sich für das Anwendungsfeld nicht eignet, wird es nie vergleichbar gute Resultate erzielen, wie ein vielleicht schlechter abgestimmtes, aber dafür geeignetes Verfahren. Grundlegend muss sich zunächst einmal bewusst gemacht werden, wie das System an Erfahrung gewinnen soll. Ferner spezifiziert muss entschieden werden, ob das System selbstständig lernen soll oder sogar muss. Denn oftmals ist diese Entscheidung auch an gewisse Gegebenheiten gebunden, in anderen Fällen wiederum willkürlich oder präferenziell. Der hierbei betrachtete Aspekt ist die Überwachung des Lernprozesses. Tendenziell wird zwischen überwachten Lernverfahren und unüberwachten Lernverfahren unterschieden. Beide Verfahren haben sowohl Vor- als auch Nachteile und eignen sich mehr oder weniger gut für bestimmte Anwendungsszenarien. Eine kurze Interpretation hierzu bieten die folgenden Absätze, um einen Rückschluss auf ausschlaggebende Gründe für die Entscheidung bei der später konkretisierten Auswahl zu ermöglichen.

4.1 Unüberwachtes Lernen

Beim unüberwachten Lernen (*engl.:* „Unsupervised Learning“) sind im Vorhinein keine Informationen über die Klasse eines Datums gegeben. Die eigentliche Klassifizierung geschieht also nicht durch das Lernen des Erkennens von Merkmalen, die einen Experten im Vorfeld dazu veranlasst haben, die Trainingsdaten den jeweiligen Klassen zuzuordnen. Stattdessen wird hierbei die Erfahrung nur aus den Daten selbst gewonnen und die Entscheidung, welches Merkmal bei der Klassifizierung mehr oder weniger stark berücksichtigt wird, trifft das Lernverfahren selbst. Der entscheidende Vorteil, gegenüber überwachten Verfahren, ist somit die Unabhängigkeit von Ex-

pertenwissen. Die Genauigkeit von unüberwachten Verfahren lässt sich allerdings erst im Nachhinein bestimmen, da es während des Lernens über keinerlei Informationen verfügt, mit welchen es seine Ergebnisse validieren könnte. Unüberwachte Lernverfahren können somit aber auch als selbstständig betrachtet werden, was vor allem dann von großem Nutzen sein kann, wenn keinerlei Expertenwissen vorhanden ist und auch nicht bereitgestellt werden kann.
Gegenüber dem Bestärkenden Lernen (*engl.:* „Reinforcement Learning“) verfügt unüberwachtes Lernen über keinerlei Belohnungswert, den das System nutzen kann, um einzelne Schritte zu belohnen und sich so dem optimalen Weg zu approximieren. Unüberwachte Lernverfahren können unter anderem für die Segmentierung von Daten eingesetzt werden. Hierbei werden Daten in einzelne, kleinere Segmente zerlegt. Das Ziel der Segmentierung hinsichtlich des Anwendungsfalles im Bereich der Elektroenzephalographie wäre außerdem eine klare Abgrenzung der einzelnen Segmente zueinander. Diese Möglichkeit, Daten zu segmentieren und voneinander abzugrenzen, bietet die Clusteranalyse.

4.1.1 Clusteranalyse

Die Clusteranalyse, oder auch Clusterung (*engl.:* „Clustering“), ist eine Methode, Daten zu segmentieren und gehört außerdem zu den unüberwachten Lernverfahren. Beim Clustering werden in einer Menge von Daten anhand von inhärenten Ähnlichkeiten gezielt Gruppen, sogenannte Cluster, erkannt und voneinander abgegrenzt. Das Resultat einer Clusteranalyse ist die Zuordnung der Daten zu den erkannten Clustern. Der Begriff „Cluster“ hebt sich zurecht von dem Wort „Klasse“ ab, da die entstehenden Cluster nicht zwingend die für uns Menschen korrekten Abgrenzungen enthalten müssen, sondern vielmehr natürliche Gruppierungen repräsentieren. (Beierle und Kern-Isberner 2019) Es kann, auch hinsichtlich des Anwendungsfalles in der Elektroenzephalographie bspw. ebenfalls vorkommen, dass ein einziges Cluster die Anfallsdaten und unkritischen Daten enthält, während ein zweites Cluster lediglich alle Grenzfälle beinhaltet. Dies wäre dann auf eine zu hohe Priorisierung eines weniger relevanten Merkmales zurückzuführen. Eine Erhöhung des Parameters der zu erzeugenden Cluster auf die Zahl drei könnte bereits Abhilfe leisten und womöglich eine Clusterung in Anfallsdaten, unkritische Daten und Grenzfälle bewirken. Dies wäre ein optimales Resultat, sofern die Cluster nur die ihnen zugehörigen Daten enthalten. Um die Genauigkeit und Korrektheit eines Clusters zu prüfen, wäre im Nachhinein ein Abgleich notwendig. Anders als bei der Klassifizierung, in welcher die Klassen dem Verfahren in Form von Labels zur Verfügung gestellt werden, wodurch die Zuordnung der Daten sich schon während der Trainingsphase validieren lässt. Diese Validierung trägt in überwachten Verfahren maßgeblich zum Erfolg des Trainings bei.
Generell wird zwischen partitionierenden, dichtebasierten und hierarchischen Algorithmen un-

terschieden. Während bei hierarchischen Verfahren wie dem „Single-Link“ oder „Complete-Link“ eine baumartige Struktur zwischen den Clustern erzeugt wird und somit gewisse Cluster wiederum eine Teilmenge von anderen Clustern darstellen können, bilden partitionierende Verfahren keinen hierarchischen Baum. Stattdessen bilden sie in einer Menge von Daten eine gewisse Anzahl an Clustern, denen sie die Objekte anhand ihrer Ähnlichkeiten zuweisen. Diese Zuweisung erfolgt anhand eines vorher definierten Schwellwerts. Dichtebasierende Clustering Algorithmen arbeiten ebenfalls partitionierend, jedoch wird hierbei die Dichte eines Objekts zu vorher errechneten Kernpunkten bestimmt und anhand derer Cluster aus zum Kernpunkt dichten Objekten erzeugt. Ein Beispiel für ein dichtebasierendes, partitionierendes Verfahren bildet das „Density-Based Spatial Clustering of Applications with Noise“ (kurz: DBSCAN), welches im Folgenden noch näher betrachtet wird. (Helmis und Hollmann 2009)

4.2 Überwachtes Lernen

Überwachtes Lernen (*engl.:* „Supervised Learning“) verfügt, gegenüber dem unüberwachtem Lernen, über die Information, welcher Klasse ein jeweiliges Datum angehört. Diese Information ist auch bekannt als Label und wird im weiteren Verlauf als solches bezeichnet. Überwachte Lernverfahren bieten schon während des Trainings Aufschluss darüber, welche Genauigkeit das Lernverfahren besitzt, da die Abweichung der Ausgabe gegenüber dem Label messbar ist. Neben der Klassifikation eignet sich das überwachte Lernen auch für andere Einsatzzwecke, wie die Regression, welche aber für die Problemstellung im Rahmen dieser Arbeit nicht weiter relevant sind und somit auch nicht betrachtet werden.

4.2.1 Klassifikation

Während einer Klassifizierung werden Daten einer im Voraus gegebenen Auswahl von Klassen zugeordnet. Die Klassen ergeben sich durch die Label, welche den Daten zugeordnet sind. Ziel der Klassifizierung ist die Klassifikation, also eine Zuordnung aller Daten zu den jeweiligen Klassen, ohne dabei die Labels zu betrachten. Diese sind lediglich für die Validierung gedacht. Um das Klassifizierungsproblem zu lösen, existieren bereits viele Algorithmen. Eine Gegenüberstellung aller einsetzbaren Algorithmen würde jedoch ein immenses Ausmaß annehmen, weshalb im Folgenden die „flachen“ Algorithmen außer Betracht gelassen werden. Ferner werden jene Algorithmen betrachtet, die auf einer neuronalen Architektur beruhen und zu den Deep-Learning Architekturen zählen.
Mittels eines vorgegebenen Datensatzes wird eine entworfene neuronale Architektur auf die Erkennungsmuster der Klassen trainiert, um später anhand dieser gewonnenen Erfahrungen selbst

eine Klassifikation durchführen zu können. Das Netz passt während der gesamten Trainingsphase seine innere Gewichtung immer wieder an, bis die einzelnen Gewichte sich kaum noch verändern. Dieser Zustand wird auch als „austrainiert“ behandelt. Das Netz ist nun in der Lage, mit einer gewissen Genauigkeit, Daten zu klassifizieren. In der darauffolgenden Testphase sind dann vom Netz vorher nicht gesehene Daten von diesem zu klassifizieren. Diese Phase gibt einen Aufschluss darüber, mit welcher Genauigkeit das neuronale Netz neue Daten klassifizieren kann.

4.3 Neuronale Architekturen

Das künstliche neuronale Netz (*engl.:* „Artificial Neural Network“) ist grundlegend eine neuronale Architektur, also ein Zusammenschluss von Neuronen. Diese Neuronen sind in Schichten angeordnet und besitzen sowohl Eingabe als auch Ausgabe. Dieser Zusammenschluss kann auf einer Feedforward-Architektur basieren, was so viel bedeutet, dass die Ausgabe jeder Schicht lediglich in die nachfolgende Schicht einfließt, also ausschließlich mit der Eingabe der Neuronen der nachfolgenden Schicht verbunden ist. Des Weiteren wird ein ANN als „fully-connected“ bezeichnet, wenn jedes einzelne Neuron einer Schicht mit allen Neuronen der vorherigen Schicht verbunden ist. (Kussul, Baidyk und Wunsch 2010; Alazab und Tang 2019) Die einfachste Form eines ANN ist das Perzeptron (*engl.:* „Perceptron“), welches 1958 von Frank Rosenblatt vorgestellt wurde. (Rosenblatt 1958) Es basiert auf einer einzigen Schicht, welche zugleich Eingabe, als auch Ausgabe repräsentiert. Eine Erweiterung dieser Architektur stellt das Multilayer Perceptron (*sinngemäß:* „Vielschichtiges Perzeptron“) dar, welches neben einer jeweiligen Eingabe- und Ausgabe-Schicht auch mindestens eine versteckte Schicht (*engl.:* „Hidden Layer“) besitzt. (Bow 2002; Matignon 2005)
Für die Gegenüberstellung wird im Folgenden eine vielschichtige Variante betrachtet, die sowohl eine Eingabe- und Ausgabeschicht als auch mindestens eine versteckte Schicht besitzt. Ihre Eignung für die Klassifizierung von EEG-Daten kann je nach Konzeption variieren, weswegen sie im späteren Verlauf des Entwurfs gegebenenfalls genauer spezifiziert wird. Alle im Folgenden vorgestellten neuronalen Modelle sind gleichzeitig auch ein ANN, da dies der umfassende Begriff einer künstlichen neuronalen Architektur ist.

4.3.1 Multilayer Perceptron (MLP)

Durch das Erweitern der Architektur um versteckte Schichten entsteht aus einem Perceptron eine vielschichtige Variante, die ebenso genannt wird. Multilayer Perceptrons (kurz: MLP) sind neuronale Netze basierend auf der Feedforward-Architektur. Sie besitzen eine oder mehrere versteckte Schichten. Am häufigsten wird jedoch exakt eine versteckte Schicht verwendet. (Janczak

2005)

Die dreischichtige Architektur umfasst demnach eine Eingabeschicht, eine versteckte Schicht und eine Ausgabeschicht und ist „fully-connected“. Fully-connected Layers werden oftmals auch als Dense Layer bezeichnet. Sowohl der Input Layer als auch die Hidden- und der Output Layer eines Multilayer Perceptrons sind Dense Layer. Die Eingabeschicht (*engl.:* „Input Layer“), übernimmt hierbei das Einlesen der Eingabedaten und erhält eine Anzahl an Neuronen, die dem Eingabevektor entspricht. In der versteckten Schicht verbirgt sich im ausgelernten Zustand der Großteil des Wissens eines MLP und dieser wird als versteckt angesehen, da für gewöhnlich lediglich die Eingabe und Ausgabe eines Netzes betrachtet wird und das Netz während des Trainings seine interne Gewichtung automatisch anpasst. Die Ausgabeschicht (*engl.:* „Output Layer“) gibt letztlich die Ausgabe aus, welche wiederum die Repräsentation dessen ist, was der Anwender von einem Netz zu wissen vermag.

Das MLP bekommt demnach eine Eingabe von Daten, welche es dann, elementar betrachtet, von Schicht zu Schicht weitergibt und währenddessen mit den internen Gewichten verrechnet, um schlussendlich an jedem Ausgabeneuron einen Wert auszugeben. Diese Werte entsprechen einem Ausgabevektor, welcher als Ausgabe interpretiert werden kann. Angenommen es gäbe ein Ausgabeneuron für die Ausgabe „kein Anfall vorhanden“ und ein weiteres für die Ausgabe „Anfall vorhanden“. Dann würde das Netz für jedes dieser Neuronen einen Wert liefern, welcher im Idealfall „1“ oder „0“ wäre und somit eindeutig. Wenn eines der Ausgabeneuronen den Wert „1“ ausgibt, während das andere „0“ ausgibt, ist vom neuronalen Netz eine 100 prozentige Aussage getroffen worden. Verglichen mit dem Label, liegt das Netz in diesem Fall also entweder zu 0% richtig oder zu 100%. In der Regel ist diese Ausgabe jedoch nicht eindeutig. Denkbar wäre ebenfalls, dass ein Netz mehrere Ausgaben nahezu gleichgewichtet, wie beispielweise, wenn die Werte 0.7 an einem der Ausgabeneuronen und 0.8 am anderen entsprächen. Auch wenn die 80 prozentige Ausgabe des Netzes dem Label entspräche, würde hier von einer Abweichung gegenüber der idealen Ausgabe gesprochen werden und somit würde der daraus errechenbare Fehlerbetrag die interne Gewichtung beeinflussen. Dass ein Netz lernen kann, liegt vor allem daran, dass es seine Gewichte im Trainingsprozess fortlaufend anpasst. Mittels sogenannter Backpropagation wird die Abweichung mit den einzelnen Gewichten des Netzes verrechnet. Das Training wird für gewöhnlich so lange durchlaufen, bis der Fehlerbetrag so gering wird, dass die Gewichte sich nur noch minimal anpassen. Dieser Zustand wird dann als ausgelernt betrachtet.

Wie lange ein Training andauert und wie erfolgreich es ist, kann neben der Anzahl der Daten auch über die Wiederholungen gesteuert werden. Demnach werden dem Netz, während des Trainings, die Trainingsdaten wiederholt präsentiert. Es wird hierbei von sogenannten Epochen gesprochen. Die gewählte Anzahl an Epochen kann einen entscheidenden Einfluss darauf nehmen, wie erfolgreich ein Training verläuft. Dies kann sich sowohl positiv auswirken als auch negativ.

Sieht ein NN dieselben Daten zu häufig, verliert es die Fähigkeiten zu abstrahieren. Es würde sich zu sehr an dem Gesehenen orientieren und könnte bisher ungesehene Daten nicht ausreichend korrekt klassifizieren. Dieses Phänomen wird als Overfitting (*sinngemäß:* „Überfütterung“ *oder* „Überanpassung“) behandelt. Diesem kann unter anderem auch vorgebeugt werden, indem sogenannte Dropout Layers (*sinngemäß:* „Aussetzungsschichten“) zwischen zwei Schichten platziert werden, welche mit einer vorher spezifizierten, gewissen Wahrscheinlichkeit die Ausgabe der vorherigen Schicht auf 0 setzen würde. Zusammengefasst ist das Ziel somit, das neuronale Netz in einem Maße zu trainieren, dass es klare Unterschiede zwischen den Klassen auch in Grenzfällen erkennt und sich dennoch nicht zu sehr auf die Trainingsdaten stütz, wenn es im Folgenden bisher noch nicht gesehene Daten klassifizieren soll. Dieses Ziel wird auch als „Generalisierung“ behandelt. (StatSoft Europe 2020)

4.3.2 Convolutional Neural Network (CNN)

Eine weitere Architektur eines neuronalen Netzes im Bereich des maschinellen Lernens bildet das Convolutional Neural Network (kurz: CNN). Dieses Netz zeichnet sich vor allem durch seine vielen Schichten aus. Die Schichten eines neuronalen Netzes werden auch als Tiefe bezeichnet, weshalb im Bereich der CNNs auch von „Deep Learning“ (*sinngemäß:* „tiefes Lernen“), als besondere Form des maschinellen Lernens, gesprochen wird. (Chollet 2018; Aggarwal 2018) CNNs sind somit eine zu betrachtende Alternative hinsichtlich der bereits genannten Feedforward-Netze. Der Vorteil dieser Architektur liegt in der Erkennung von Mustern, welche sich in den Eingabedaten wiederfinden. Das CNN ist ein faltendes neuronales Netz, weil es seine Eingabedaten durch verschiedenste Operationen „faltet“. Die Eingabe erfolgt bspw. zweidimensional durch Bilddaten, welche sich aus den Pixelwerten der Dimensionen Höhe und Breite zusammensetzen. Eine dritte Dimension kann hinzukommen, wenn Farbbilder verwendet werden. Der RGB-Wert eines Pixels bildet dann die dritte Dimension. Des Weiteren sind vom CNN erlernte Muster translationsinvariant, was bedeutet, dass sie an jeder Position des Bildes wiedererkannt werden können, während ein fully-connected NN ein Muster lediglich an der Position erkennen kann, in denen es auch gelernt wurde. (Chollet 2018)
Grundlegend wird hier, gegenüber dem MLP, von drei neuen Arten von Schichten gesprochen, obgleich der Einsatz weiterer Schichten optional möglich ist. Der Convolutional Layer (*sinngemäß:* „Faltungsschicht“) scannt die Pixelwerte der Eingabedaten und berechnet anhand eines Filters (*engl.:* „Kernel“), mit dessen internen Gewichten, neue Werte. (Becker 2019) Der Filter ist hierbei ein Fenster von fester Größe (zum Beispiel 3x3) und wird auf ein größeres Fenster von Eingabedaten angewandt, was zu einer „Faltung“ führt. (Becker 2019) Der Convolutional Layer wird gängiger Weise gefolgt von einem Pooling Layer (*sinngemäß:* „Zusammenführungs-Schicht“) ver-

wendet. (Dong und Liu 2018) Dieser führt die Ausgabe des Convolutional Layer zusammen, indem er die jeweils stärksten, oder auch relevantesten, Signale an die nächsten Schichten weitergibt, wodurch sich eine Repräsentation der Daten auf eine reduzierte Anzahl ergibt. (Becker 2019) Dieser Effekt begünstigt das Lernen in großen Dimensionen, wodurch sich das CNN besonders in der Verarbeitung von Bildern bewährt. (Becker 2019; Alpar u. a. 2019) Der Dense Layer, welcher analog zum Dense Layer der Feedforward Architektur ist, wird anschließend zur Skalierung der Ausgabe auf die gewünschten Klassen genutzt. Die Matrizen, mit welchen sowohl der Convolutional Layer als auch der Pooling Layer rechnen, müssen vor der Einspeisung in einen Dense Layer in den Vektorraum eingebettet werden, indem sie zu einem eindimensionalen Vektor ausgerollt werden. Dies wird auch als „Flatten“ bezeichnet und durch den sogenannten Flatten Layer realisiert.
Die Architektur eines Convolutional Neural Networks ist in diesem Anwendungsszenario vor allem deswegen von Relevanz, da es einen entscheidenden Vorteil bedeuten kann, vorher gesehene Muster in EEG-Daten ortsunabhängig wiedererkennen zu können. Da Anfallspotentiale nicht immer denselben Verlauf nehmen, sich aber in Ihrem Vorkommen ähneln, kann hierdurch eventuell eine Abstraktion erreicht werden, die andere Architekturen nicht bieten können. Die Eignung eines CNNs für die Lösung des Kernproblems der Anfallserkennung ist somit im Folgenden näher zu beleuchten.

4.3.3 Recurrent Neural Network (RNN)

Bei der Verarbeitung von Sequenzen jeglicher Art gilt es, die Abhängigkeiten innerhalb dieser Sequenzen als weiteren Aspekt zu berücksichtigen. Während der Klassifizierung einer Sequenz kann die Korrelation einzelner Segmente den entscheidenden Hinweis für die Klassifikation bieten. Während Eingaben in Form von Bildern oder Vektoren meist unabhängig von vorherigen oder zukünftigen Eingaben sind, verhält es sich bei Sequenzen zumeist anders. Ein Text bspw. bildet eine Sequenz aus mehreren Sätzen ab. Dieser wiederum bildet eine Sequenz aus Wörtern ab, deren Zusammenhang für die korrekte Deutung des Satzes von entscheidender Bedeutung ist. Um während der Verarbeitung einer Sequenz die anfangs verarbeiteten Informationen bei der Klassifizierung berücksichtigen zu können, ist es unerlässlich, diese Informationen über die Zeit hinweg halten zu können. In diesem Zusammenhang kommen die rekurrenten neuronalen Netze zum Tragen. Sie besitzen, anders als vorher genannte Architekturen, eine Art Gedächtnis, welches es ihnen ermöglichen, vorher verarbeitete Eingaben einer Sequenz maßgeblich in die derzeitige Eingabe hineinfließen zu lassen. Des Weiteren erhalten RNNs die zeitliche Reihenfolge aller Eingaben. (Ripper 2000) Auf diese Weise ist es auch möglich, den Kontext eines Textes zu erfassen.

Grundlegend ist die Architektur eines rekurrenten neuronalen Netzes nicht allzu unterschiedlich gegenüber einem Multilayer Perceptron. Während jedoch beim MLP jedes Neuron ausschließlich mit Neuronen der darauffolgenden Schicht verbunden ist, wird beim RNN die Ausgabe jeder Schicht zum Zeitpunkt t ebenfalls in dieselbe Schicht zum Zeitpunkt t+1 erneut eingespeist. (Ripper 2000; Lange 2004) Hierdurch ergibt sich die Beeinflussung der aktuellen Eingabe durch bereits gesehene Eingaben. Dieses Merkmal verdeutlicht die folgende Abbildung.

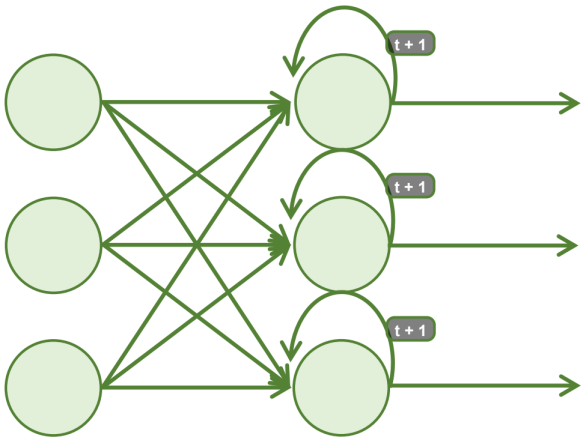

Abbildung 4.1: Schematische Darstellung der Architektur eines RNN
(Eigene Darstellung)

Im Rahmen einer Ausarbeitung mit dem Themengebiet „Intelligente Systeme“ im Jahr 2018 wurden die Charakteristika eines RNN bereits näher beleuchtet. Hierbei wurde ebenfalls festgestellt, dass „wiederkehrende“ neuronale Netze eine Beeinträchtigung des Lernprozesses hinsichtlich langer Sequenzen aufweisen. Ein RNN ist dazu in der Lage, eine Korrelation zwischen den Eingaben zu erkennen, indem vorherige Eingaben maßgeblich in die aktuell verarbeitete Eingabe hineinfließen lässt und somit deren Ausgabe beeinflusst. In diesem Zusammenhang kann es zum Schwinden oder Explodieren des Fehlergradienten kommen. (D’Errico u. a. 2015) Dies ist darauf zurückzuführen, dass während der „Backpropagation through time“ (kurz: BTT) der Fehlergradient die einzelnen Schichten durchläuft. (Sherstinsky 2018) Üblicherweise wird nach jeder Schicht eine Aktivierungsfunktion zur Skalierung der Ausgabe genutzt. Die sigmoid Aktivierungsfunktion bspw. skaliert die Ausgabe auf einen Wertebereich von 0 bis 1. Die Ableitung einer sigmoid Funktion liegt wiederum in einem Wertebereich von 0 bis 0,25 und mit eben dieser wird der Fehlergradient während der BTT multipliziert, was zu einer Skalierung des Fehlergradienten auf maximal 1/4 des ursprünglichen Wertes führt. (Aggarwal 2018) Mit zunehmender Anzahl

an Schichten wird sich der Fehlergradient somit dem Minimum approximieren, bis er gänzlich schwindet. Wenn der Fehlergradient die frühen Schichten aber nicht erreicht, können sie auch nicht trainiert werden. (D'Errico u. a. 2015) Da die späteren Schichten funktional von den vorherigen abhängen, würde dies unweigerlich dazu führen, dass das gesamte Netz beschädigt wird.

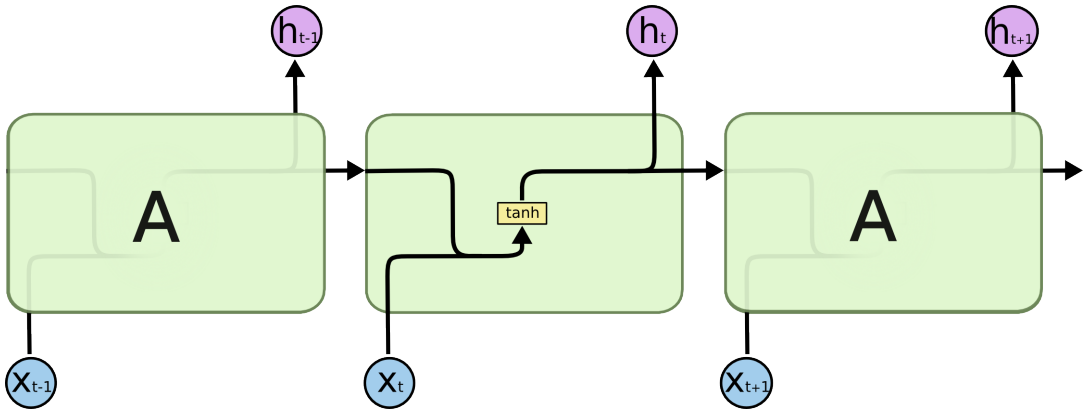

Abbildung 4.2: Zeitlich kausale Darstellung einer RNN-Zelle
(Olah 2015)

4.3.4 Long Short-Term Memory (LSTM)

Das Problem der Abhängigkeiten über einen längeren Zeitraum führt zu einer weitergehenden Entwicklung und damit ebenfalls zu einer neuen neuronalen Architektur, dem Long Short-Term Memory (*sinngemäß:* „Langes Kurzzeitgedächtnis"). Diese Architektur beugt dem Problem der schwindenden Abhängigkeiten über längerem Zeitraum vor, indem es einen sogenannten Zellzustand (*engl.:* „Cell state") einführt, welcher als Langzeitgedächtnis fungiert. Ebenfalls im Rahmen der im vorherigen Absatz genannten Ausarbeitung ist die Eignung eines LSTM zur Vorbeugung schwindender Fehlergradienten bereits erkannt worden. Im Wesentlichen wird die Architektur des RNNs durch den besagten Cell State ergänzt. Dieser erhält seinen Zustand beim Durchlaufen dreier Phasen, den sogenannten Toren des LSTM, welche jeweils durch ein Perzeptron gesteuert werden.
Das erste Tor, welches der Zellstatus hierbei passiert, ist das Forget Gate. Dieses gibt dem Zellstatus die Informationen, welche seiner Daten vergessen werden und welche erhalten bleiben sollen. Es ist somit für das Vergessen vorher gesehener Eingaben zuständig. Im nächsten Schritt durchläuft der Cell State das zweite Tor, welches als Update Gate, oder auch Input Gate, bezeichnet wird und Aufschluss darüber gibt, welche Informationen der aktuell verarbeiteten Eingabe

in den Zellstatus einfließen soll. Mit dem dritten und letzten Tor, auch bekannt als Output Gate, wird der aktuelle Zellstatus aktualisiert und die Ausgabe festgelegt, welche sowohl in die nächste Eingabe als auch in den nächsten Layer hineinfließt.
Ein entscheidender Vorteil des LSTMs ist die Eigenschaft, dass es seinen Zellzustand prinzipiell vollkommen erhalten kann, indem es das Forget Gate so schaltet, dass es keine Informationen verwirft. (D'Errico u. a. 2015) Indem das LSTM sein Forget Gate für gezielte Informationen öffnet, wird ein Durchgang geschaffen, der in der Verrechnung mit dem Gradienten dafür sorgt, dass dieser erhalten bleibt. (Arbel 2010) Ferner wird der Fehlergradient rezirkuliert und das Zustandssignal der Zelle in jedem Schritt aktualisiert. Dies beschreiben die Erfinder des Long Short-Term Memory als "Constant Error Carousel"(kurz: CEC, *sinngemäß:* „Karussel mit konstantem Fehler"). (Sherstinsky 2018; Hochreiter und Schmidhuber 1997) Durch diesen Mechanismus beugt das LSTM dem Problem der schwindenden Gradienten bei Langzeitabhängigkeiten vor. (D'Errico u. a. 2015; Sherstinsky 2018)

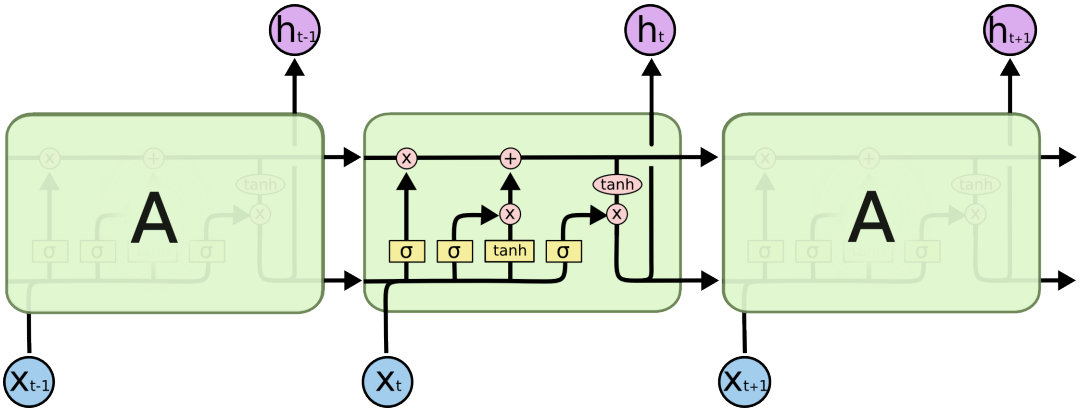

Abbildung 4.3: Zeitlich kausale Darstellung einer LSTM-Zelle
(Olah 2015)

4.4 Auswahl

Da sich jedes der beleuchteten Verfahren in gewisser Weise in der Praxis für einen speziellen Bereich bewährt hat und sich somit unter jeweils anderen Aspekten für die Lösung eignen könnte, muss im Folgenden entschieden werden, welche Verfahren verwendet werden sollen und wie sie am erfolgreichsten eingesetzt werden können. Da das LSTM als besondere Variante des rekurrenten neuronalen Netzes einige bereits genannte Vorzüge gegenüber dem klassischen RNN mitbringt, wird das gewöhnliche RNN an dieser Stelle bereits vernachlässigt. Mutmaßlich kann

den bewährten Einsatzgebieten der vorgestellten Verfahren nun eine Gewichtung hinsichtlich der Problemstellung zugeteilt werden. Da jedoch die Eignung der übrigen Verfahren von diesem Standpunkt aus nur geschätzt werden kann, wird vorerst jedes dieser Verfahren im folgenden Kapitel entworfen und implementiert. Die Auswertung wird dann Aufschluss darüber geben, ob gewisse Verfahren im Nachhinein doch noch zu vernachlässigen sind. Um das hohe Ziel der absoluten Genauigkeit weiter anzustreben, wird aus den implementierten ML-Verfahren eine Art hybrider Verbund entstehen. Diese wird die geeignetsten Verfahren sinnvoll miteinander verbinden, um zu prüfen, ob gewährleistet werden kann, dass Anfälle von der KI nicht übersehen werden. In der anschließenden Evaluation wird dann genauer erläutert, was erreicht werden konnte und wie es erreicht wurde. Im Zuge dieser Arbeit wurde bewusst verstärkt auf die überwachten Lernverfahren gesetzt, da sie die tieferen Ansätze baten. Durch den öffentlichen Datensatz stehen ebenfalls gelabelte Referenzdaten für das Training zur Verfügung, welche sich in überwachten Verfahren für die Validierung eignen. Die unüberwachten Lernverfahren sollen aber nicht gänzlich vernachlässigt werden. Ferner sollen sie die Chance bekommen, sich bei der Problemlösung zu bewähren und demnach ihren Anteil an der letztendlich erreichten Genauigkeit beizutragen.

5 Umsetzung

5.1 Bibliotheken

5.1.1 NumPy

Um wissenschaftliche Berechnungen in der Programmiersprache Python durchzuführen, kann die Bibliothek NumPy genutzt werden. Neben vielen Funktionen, wie bspw. für die lineare Algebra, bietet NumPy vor allem eine Datenstruktur, die eine Vielzahl an Manipulations- und Berechnungsmöglichkeiten mit sich bringt. Dies ist das sogenannte NumPy Array, welches ein multidimensionales Array repräsentiert. (van der Walt, Colbert und Varoquaux 2011) Für den Anwendungsbereich des maschinellen Lernens wird dieses bspw. genutzt, um die Trainings- sowie Testdaten zwischenzuspeichern und ihre Form zu verändern. Die daraus entstehenden Datenstrukturen können dann an das entsprechende Lernverfahren übergeben werden. Für neuronale Netze im komplexeren Stil, wie dem LSTM, ist ein mehrdimensionales Array die Grundlage zur Einspeisung der Daten. Für den Bereich der Elektroenzephalographie ergibt sich eine weitere Nutzungsmöglichkeit der NumPy Bibliothek, da sie ebenfalls die diskreten Fourier Transformationen beinhaltet, welche für die Extraktion spezifischerer Informationen aus einem Signal von großem Mehrwert sein kann. Dieser Punkt wird im zugehörigen Kapitel „Fast Fourier Transformation“ nochmals aufgegriffen und näher erläutert.

5.1.2 MNE-Python

Ein bekanntes Format für den Export der Daten einer Elektroenzephalographie bildet das EDF-Format (*.edf). Es bietet eine Vielzahl an Vorteilen gegenüber der Alternative, lediglich die Signale zu erfassen und in eine Textdatei zu speichern. Es bewahrt die Informationen über die einzelnen Kanäle, deren Verschaltung und die Zeitpunkte, zu denen ein bestimmtes Ereignis eingetroffen ist. Diese Zeitpunkte werden im Folgenden auch als Marker bezeichnet. Das EDF-Dateiformat kann in Python ebenfalls eingelesen werden und muss somit nicht aufwändig in eine lesbare Form gebracht werden. Diese Funktion wird von der Bibliothek „MNE-Python“ bereitgestellt. Sie

enthält zudem umfangreiche Möglichkeiten, neurophysiologische Daten grafisch aufzubereiten, visuell darzustellen und zu analysieren. (Gramfort u. a. 2013) Für diesen Anwendungsfall wurde die MNE-Python Bibliothek jedoch vor allem dafür genutzt, einzelne Bereiche von im EDF-Format vorliegenden Daten näher zu betrachten, um diese in darauffolgenden Schritten an den richtigen Stellen in kleinere Fragmente zu zerlegen.

5.1.3 pandas

Die Bibliothek „pandas“ bietet eine simple Möglichkeit, Datenstrukturen tabellarisch darzustellen und über die Zeilen und Spalten gezielt auf die Daten zuzugreifen. Die sogenannten Dataframes werden wie Tabellen behandelt und können auch auf diese Weise manipuliert werden. Spalten lassen sich bspw. über ihren Spaltennamen referenzieren und löschen. Das Hinzufügen weiterer Zeilen und Spalten an bestimmten Stellen des Dataframes ist ebenfalls möglich. Dataframes sind zudem eine geeignete Methode, Datenstrukturen visuell auf ihre Konsistenz zu prüfen. Pandas bietet ebenfalls Funktionen, um Dataframes in eine .CSV-Datei zu exportieren und diese wieder einzulesen. Somit stellt die Bibliothek auch ein Hilfsmittel für ein Backup-Konzept von Datenstrukturen dar. Diese Bibliothek wird aufgrund einheitlicher Schreibweise im Folgenden mit „Pandas“ betitelt.

5.1.4 Matplotlib

Manchmal kann es einen entscheidenden Vorteil mit sich bringen, Daten, welche in einer Datenstruktur vorliegen, grafisch dazustellen. Dies kann wie zuvor erwähnt durch die Bibliothek Pandas in tabellarischer Form geschehen. Eine übersichtlichere Form, hinsichtlich statistischer Auswertungen, bieten jedoch Diagramme. Ein Tortendiagramm bspw. kann innerhalb eines Blickes Aufschluss über die Verteilung der Daten auf die Klassen bieten. Ähnlich verhält es sich bei Zeitreihen, in welchen die Messdaten der Elektroenzephalographie ebenfalls vorliegen. Diese können mittels Zeitreihendiagramm einfach und übersichtlich dargestellt werden. Die grafische Darstellung übernimmt die Matplotlib Bibliothek, welche zahlreiche Möglichkeiten bietet, Daten zu „plotten“. Der Begriff „plot“ orientiert sich hierbei an dem englischen Begriff für „zeichnen“. Darüber hinaus bietet Matplotlib ebenfalls die Möglichkeit, Diagramme zu beschriften, mit einem Gitternetz zu versehen oder Skalen gänzlich auszublenden, sodass nur die grafisch dargestellten Daten zu sehen sind. Dies kann vor allem dann von Vorteil sein, wenn einzig das Muster der grafischen Darstellung verglichen werden soll und die Skalen keinen Mehrwert bieten. Eine alternative Bibliothek zur grafischen Darstellung von Daten wäre die „Plotly“ Bibliothek, welche ähnliche Vorzüge bietet. Diese wurde im Rahmen dieser Arbeit allerdings nicht verwendet.

5.1.5 TensorFlow mit Keras

Der Aufbau neuronaler Netze kann durch die Bibliothek „TensorFlow“ entscheidend vereinfacht werden. Simple neuronale Architekturen können mit ihr in kürzester Zeit entworfen werden. Das dabei entstehende Netz wird auch als Modell (*engl.:* „Model“) bezeichnet. Zudem bietet „TensorFlow GPU“ die Möglichkeit, Berechnungen von der GPU durchführen zu lassen, statt, wie herkömmlich, von der CPU. Berechnungen auf einer GPU durchzuführen, kann diese drastisch beschleunigen. (NVIDIA Corporation 2020) Mit der High-Level API „Keras“ wird es zudem um ein Vielfaches einfacher, tiefschichtige neuronale Architekturen zu entwerfen. Der Aufbau einer vielschichtigen Architektur kann sequenziell erfolgen, indem die einzelnen Schichten des Netzes sequenziell mittels Keras‘ Sequential API gestapelt werden. Keras enthält zudem auch vorgefertigte Schichten für das Erstellen komplexerer Architekturen, wie dem LSTM. Mit Keras entworfene Modelle können auf Basis von Trainingsdaten trainiert und im weiteren Verlauf mit Testdaten validiert werden. Keras bietet hierfür entsprechende Funktionen, die einen Einblick in die Trainingserfolge ohne eigenhändige Berechnungen ermöglichen. Mit TensorFlow und Keras entworfene Modelle können zudem für eine spätere Wiederverwendung abgespeichert werden, um sie bei Bedarf wieder einzulesen. Die auf diese Art zwischengespeicherten Modelle sind nach wie vor anpassbar und erweiterbar. Der Trainingsprozess kann durch sogenannte Checkpoints unterbrochen oder zwischengespeichert werden, sodass ein späteres Trainieren ab dem jeweiligen Checkpoint möglich ist. Besonders im Bereich von großen Datendimensionen kann dies hilfreich sein.

5.1.6 Keras Tuner

Modelle, die durch TensorFlow mit Keras modelliert werden können, fahren in vielen Fällen zunächst nicht die optimalen Resultate ein. Meist ist das Finden der idealen Parameter ein „Trial and Error“-Prinzip, bei dem immer wieder auf mutmaßlich guten Parametern ein Training begonnen wird, welches sich im Nachhinein als besser oder weniger gut herausstellen kann. Um dem Zeitaufwand der Versuche entgegenzuwirken, bieten die Entwickler der Keras Bibliothek mit dem Keras Tuner eine automatisierte Lösung an. Der Keras Tuner soll anhand von gegebenen Bedingungen sämtliche Parameter durchspielen und sich den Erfolg der Resultate merken. Die zu wählenden Parameter werden auch als Hyperparameter bezeichnet. Die Bibliothek bietet zudem mehrere verschiedene Tuner. Der RandomSearch Tuner sucht auf zufälliger Basis nach den besten Parametern in vorher definierten Bereichen. Ein weiterer Tuner ist der Hyperband Tuner, welcher zunächst die Parameter zufällig wählt und auf ihnen trainiert, bis er nach einigen Epochen unterbricht, um lediglich auf den bisher vielversprechendsten Parametern fortzufahren. (Li u. a. 2018) Dieses Vorgehen kann das Tuning entscheidend beschleunigen. Ein Modell,

welches definierte Hyperparameter besitzt, wird als Hypermodel behandelt. Es ist möglich, Hypermodels in eine Subklasse auszulagern. In diesem Fall erhalten sie die Funktion *build*, welche das Modell aufbaut. (Keras-Team 2020)

5.1.7 scikit-learn

Die Bibliothek "scikit-learn" ist für das Verwenden von ML-Verfahren in Python nahezu unerlässlich. Neben ihrer Vielzahl an Algorithmen und Datensätzen für den Bereich des maschinellen Lernens, bietet sie auch nützliche Funktionen, die bspw. das Preprocessing entscheidend erleichtern können. So wurden im Rahmen dieser Arbeit neben dem DBSCAN Algorithmus auch verschiedenste Funktionen der scikit-learn Bibliothek zum Vorverarbeiten des Datensatzes verwendet. Ein Beispiel hierfür bietet das randomisierte Mischen des Datensatzes, um einer glücklichen Verteilung der Daten vorzubeugen. Des Weiteren wurde scikit-learn dafür genutzt, die Werte der Daten auf einen Wertebereich zwischen 0 und 1 zu normieren, bevor sie an das jeweilige ML-Verfahren übergeben wurden. Diese und weitere Funktionen haben sich in der Praxis mittlerweile etabliert und scikit-learn, welches auch oft mit sklearn abgekürzt wird, vereint diese in einer Bibliothek. (Pedregosa u. a. 2011)

5.2 Entwurf

5.2.1 Multilayer Perceptron

Der erste Entwurf zielte darauf ab, ein Gefühl für die Daten zu bekommen und insbesondere ihre Beschaffenheit, Alleinstellungsmerkmale und Trainierbarkeit näher zu untersuchen. Für die ersten Trainings sollte eine Grund-Architektur verwendet werden, die in weiteren Schritten modifiziert und gegebenenfalls revidiert werden würde.
In diesem Versuch lagen die Daten bereits in einem vorverarbeiteten Stand vor. Insgesamt gab es 11500 Datenpunkte, wovon ein jeder genau eine Sekunde Messintervall repräsentiert. Diese Sekunde umfasst ein Abtastintervall von 178 Samples. Somit liegen für jede Sekunde 178 Werte vor, die in diesem Versuch auf eine Eingabeschicht von 178 Neuronen abgebildet werden. Die versteckte Schicht einer 3-Schichten-Architektur muss nun entsprechend der Dimension des Input Layers entworfen werden. Im ersten Durchlauf wurden für den Hidden Layer 1.000 Neuronen konzipiert. Die letzte Schicht, die Ausgabeschicht, sollte lediglich zwei mögliche Ausgaben treffen können. Eine davon repräsentiert das Vorliegen eines Anfalls und die andere den normativen Zustand. Für die Aktivierung der Dense Layer wurde in den ersten Schichten die „Rectified Linear Unit" Aktivierung genutzt, welche im Folgenden auch als ReLU bezeichnet wird. Einzig die Ausgabeschicht erhielt eine sigmoid Aktivierungsfunktion, da diese in der Praxis häufig zum Einsatz

kommt. (Kroll 2013) Generell werden Aktivierungsfunktionen dazu verwendet, die Neuronen einer Schicht zu aktivieren. Man spricht hierbei auch davon, dass ein Neuron „feuert“, ähnlich wie beim biologischen Neuron. (Krausz 2018) Dies kann, wie bei der sigmoid Funktion durch einen Schwellwert geschehen oder durch eine lineare Aktivierung.
Die Daten werden in zwei Instanzen aus Trainings- und Testdaten zerlegt. Der Anteil der Trainingsdaten beläuft sich hierbei auf 10000, womit für die anschließende Testphase noch 1500 Testdaten verwendbar sind. Um das Training mehrmals zu durchlaufen, wird vorab ebenfalls die Anzahl der Epochen definiert. Eine zu hohe Anzahl an Epochen kann, wie bereits im vorherigen Kapitel beschrieben, ein „Überfüttern“ begünstigen. Jedoch könnte auch eine zu gering gewählte Anzahl dazu führen, dass ein Netz Grenzfälle noch nicht ausreichend gut klassifizieren kann, da es den Schwellwert aufgrund mangelnden Wissens falsch definiert. Für den ersten Anlauf liegt die Anzahl der Epochen bei 50. Im weiteren Verlauf wurde sie auf 30 reduziert, um die Trainingszeit zu verringern und die Observation den anderen Einflussfaktoren zu widmen. Sobald ein hinreichend starkes Netz entwickelt war, wurde die Anzahl der Epochen auf 300 erhöht, um die endgültige Genauigkeit zu bestimmen. Ein noch höherer Wert schien keine positiven Auswirkungen mehr auf die Genauigkeit zu nehmen.
Einen weiteren entscheidenden Einfluss auf das Training kann die Reihenfolge nehmen, in der die Daten im Datensatz vorliegen. Angenommen die Daten würden eingangs nicht ausreichend gemischt und es befänden sich alle Anfallsdaten am Ende des Datensatzes. Somit könnte die Auftrennung in Trainings- und Testdaten dazu führen, dass das Netz niemals auf Anfallsdaten trainiert werden würde, weil der Trainingsdatensatz keine enthält. Dementsprechend schlecht wäre das Netz dann vermutlich in der Klassifizierung dieser Daten. Um diesem Problem vorzubeugen, wurden die Daten auf randomisierter Basis gemischt.

5.2.2 Fast Fourier Transformation

Im weiteren Verlauf soll die Accuracy der Architektur noch weiter verbessert werden. Dies kann einerseits erreicht werden, indem die Architektur verändert wird oder durch die Veränderung oder Erweiterung der Eingabedaten. Letzteres kann zum Beispiel dadurch erreicht werden, dass aus den bereits vorhandenen Daten neue Erkenntnisse gewonnen werden.
Mittels Fouriertransformationen lassen sich aus Signalen die enthaltenen Frequenzwerte ermitteln, indem sie in eine Frequenzfunktion konvertiert werden. Die Fast Fourier Transformation gehört zu der Gruppe der Diskreten Fourier Transformationen und zeichnet sich durch seine namensgebende Schnelligkeit in der Berechnung aus. Diese wird vor allem durch Reduktion der Rechenschritte und gezieltes Vorsortieren der Summanden erreicht. (Möser 2018)
Mittels der Fast Fourier Transformation, in dem Anwendungsfall der EEG-Daten, sollen die Fre-

quenzspektren der Datenpunkte errechnet werden. Im weiteren Verlauf wird anhand eines Trainings nachgewiesen werden, ob sich der Anteil der Frequenzen eines Datenpunkts auf die Klassifizierung auswirken würde und somit dem Training dienen kann. Bei der Berechnung der vorliegenden 178 Samples mittels des FFT Algorithmus, ergeben sich weitere 178 Samples, welche eine Punktsymmetrie in der Mitte aufweisen. Somit sind hier lediglich die ersten 50% der Werte interessant, da die letzteren 50% keinen Mehrwert mehr bieten. Für das Beispiel wurden also nur die ersten 89 Samples betrachtet und den 178 Rohdaten hinten angefügt.
Durch das Hinzufügen der Frequenzwerte könnte der Trainingserfolg deutlich erhöht werden. Dies könnte darauf zurückzuführen sein, dass sich während eines Anfalls die Frequenz verändert. Im Anwendungsfall der EEG-Diagnostik ist es jedoch unabdingbar, dass das Netz kein Vorliegen übersieht. Ein Vorliegen zu erkennen, welches sich im Nachhinein als nichtig erwies, ist eher vertretbar. Im Bereich der Informatik wird beim Missachten eines Vorliegens von einem false-negative gesprochen und beim fälschlichen Erkennen eines nicht vorliegenden Ereignisses von einem false-positive. Ein wichtiger Schritt wäre also klar die Unterscheidung zwischen false-negatives und false-positives, in genau den Fällen, wo das Netz von der bekannten Lösung, dem Label, abweicht. Eine Minderung dieser false-negative Werte kann bspw. durch die Ausreißer-Erkennung, auch genannt Anomaly Detection, erreicht werden.

5.2.3 Convolutional Neural Network

Eine Besonderheit, die es in der Konzeption eines Faltungsnetzwerks zu berücksichtigen gilt, ist die erweiterte Aufbereitung der Daten. Bis zu diesem Zeitpunkt war es möglich, die Daten als Zeitreihen zu verarbeiten. Anders als beim MLP benötigt ein CNN eine zweidimensionale oder dreidimensionale Eingabe. Dies lässt wieder Rückschlüsse auf den Anwendungsbereich der Bildverarbeitung schließen, in welcher das CNN am häufigsten eingesetzt wird. Um die vorliegenden Zeitreihen in ein CNN einspeisen zu können, wäre eine Möglichkeit, sie in eine grafische Darstellung zu überführen, um sie anschließend als Byte-Array an das CNN zu übergeben. Diese Vorverarbeitung muss in der Implementierung entsprechend berücksichtigt werden.
Farbbilder haben generell eine dreidimensionale Struktur, da sie eine Auflösung besitzen, die sich aus der Breite und Höhe an Pixeln ergibt. Die dritte Dimension ist hierbei der Farbraum. Jeder Pixel besitzt einen RGB-Wert, also jeweils einen 8-Bit Wert für die Farbwerte Rot, Grün und Blau. Diese Werte können als Tiefe des Bildes betrachtet werden. Für die grafische Darstellung eines Signales werden möglicherweise lediglich Graustufen benötigt, die farbliche Darstellung mittels RGB-Werten wäre demnach eine überflüssige Funktion. Es kann somit grundlegend festgestellt werden, dass sich die Ressourcen für das Verarbeiten von Bilddaten um zwei Drittel reduzieren lassen, wenn lediglich ein 8-Bit Farbwert für die grafische Darstellung genutzt werden kann, wel-

cher die Graustufen repräsentiert.
Ein weiterer kostspieliger Aspekt, hinsichtlich der Performance und Ressourcen, ist die Auflösung eines einzelnen Bildes. Um ein Signal-Intervall von etwa einer Sekunde für einen Menschen zu veranschaulichen, wären Auflösungen im Bereich der High Definition, also 1280 Pixeln in der Breite zu 720 Pixeln in der Höhe, schätzungsweise eine gute Grundlage. In einem Bild mit einer Auflösung von 300 Pixeln in der Breite würde sich mit menschlichem Auge kaum noch etwas erkennen lassen, was wiederum eine Diagnose erheblich erschweren würde. Ein CNN jedoch betrachtet, je nach Konfiguration, jeden einzelnen Pixel eines Bildes und ist somit viel eher in der Lage, auch auf kleiner Auflösung, genaue Aussagen über ein Bild zu treffen. Wie bereits in vorherigen Kapiteln erwähnt, ist für ein effektives Training mittels neuronaler Netze eine gewisse Anzahl von Trainingsdaten notwendig. Mit zunehmender Anzahl würden sich die Kosten entsprechend erhöhen. Es gilt also, für die Bilddaten eine geeignete Auflösung zu finden, welche kostentechnisch umsetzbar ist und die Resultate der Klassifizierung nicht negativ beeinflusst.
Ist die Vorverarbeitung abgeschlossen und die Trainings- sowie Testdaten liegen in einer geeigneten Datenstruktur vor, so können sie in das entsprechende CNN eingespeist werden. Dieses faltende neuronale Netz muss den Dimensionen der Daten nach entworfen worden sein. Wie bei anderen neuronalen Netzen sind spätere Anpassungen ein wichtiger Bestandteil der Optimierung des Lernprozesses. Für den Anfang wurde die Eingabeschicht für eine Eingabe von Daten mit der Größe 300x300 vorbereitet. Im späteren Prozess stellte sich jedoch heraus, dass eine Auflösung der Bilder von 150 zu 150 Pixeln in diesem Fall vollkommen ausreichend ist. Die Eingabeschicht ist anders als beim gewöhnlichen NN ein Convolutional Layer, dessen Merkmale im vorherigen Kapitel bereits näher betrachtet wurde. Wie üblich, wurde darauffolgend ein Max-Pooling Layer verwendet, welcher die Ausgabe der vorherigen Schicht faltet und die jeweils stärksten Werte durchlässt. Diese Faltung kann die Dimensionen der Daten drastisch reduzieren, wodurch Trainings schneller oder tiefer durchgeführt werden können. Sie sorgt außerdem dafür, einer Überanpassung des Netzes entgegenzuwirken. (Chollet 2018) Basierend auf einem Modell von François Chollet (Chollet 2018) wurde die Ausgabe der ersten Schicht auf einen Wert von 32 festgelegt und mit jedem weiteren Convolutional Layer verdoppelt. Die Architektur wurde während des Trainings unzählige Male angepasst, bis das CNN schlussendlich eine zufriedenstellende Genauigkeit erreichte.
In der endgültigen Architektur befinden sich vier Convolutional Layers, welche jeweils einen darauffolgenden Max-Pooling Layer besitzen. Des Weiteren folgt ein sogenannter Flatten Layer, welcher die zweidimensionalen Daten der vorherigen Schichten, auch genannt Feature Maps, in eindimensionale Daten ausrollt. Dies ist erforderlich, damit sie im Folgenden von einer auf eindimensionalen Daten arbeitenden Schicht entgegengenommen werden können. Um einer Überfütterung entgegenzuwirken, wurde hier außerdem der eingangs erwähnte Dropout Layer ver-

wendet. Dieser fungiert hier mit einer 50 prozentigen Wahrscheinlichkeit als Aussetzer, indem er die Ausgaben vorheriger Schichten auf 0 setzt. Die Wirksamkeit konnte durch anschließende Testphasen bestätigt werden. Die ausgerollten Daten werden anschließend durch zwei gewöhnliche Dense Layer geschickt und schlussendlich ausgegeben. Die einzelnen Convolutional Layer besitzen aufsteigend 32, 64, 128 und 256 Filter und die Kernel-Größe beträgt jeweils 3x3. Die Max-Pooling Layers erhalten jeweils einen Kernel von 2x2, wodurch die Daten auf die Hälfte ihrer ursprünglichen Größe skaliert werden. Der erste Dense Layer fängt die riesige Anzahl an ausgerollten Werten ab und skaliert sie auf eine Ausgabe von 120 Neuronen, damit sie in der folgenden Schicht wiederum auf die zwei gewünschten Ausgabeneuronen skaliert werden können. Die Ausgabeneuronen repräsentieren wie bekannt die Zustände „Anfall vorhanden“ und „Anfall nicht vorhanden“. Die Aktivierungen aller Schichten, abgesehen von der Ausgabeschicht, bildet die ReLU Funktion. Die letzte Schicht erhielt eine sigmoidale Aktivierung aufgrund der vorherigen Erkenntnisse in ihrer Verwendung beim MLP. Das Training zeigte jedoch, dass eine softmax Aktivierung an der Ausgabeschicht bessere Ergebnisse erzielte.

5.2.4 Anomaly Detection

Bei der Anomaly Detection (*sinngemäß:* „Erkennung von Anomalien“), welche oft auch als Outlier Detection (*sinngemäß:* „Erkennung von Ausreißern“) betitelt wird, handelt es sich um ein Verfahren zur Ermittlung ungewöhnlicher Daten innerhalb eines Datensatzes. Dieses kann rein statistisch geschehen, etwa durch die Abweichung vom Mittelwert zuzüglich Toleranzwerten. Da es in diesem Fall jedoch um das Prüfen der Eignung maschineller Lernverfahren geht, wird die Anomaly Detection, auch Outlier Detection genannt, ebenfalls durch ein maschinelles Lernverfahren repräsentiert.
Es ist grundsätzlich denkbar zur Ausreißer-Erkennung ein neuronales Netz zu verwenden, wie es im Vorfelde bereits behandelt wurde, jedoch wäre der Mehrwert in diesem Fall nicht gegeben, da bereits ein neuronales Netz zur Klassifikation entworfen wurde. Für das Erkennen von Ausreißern wird demnach auf ein unüberwachtes Lernverfahren zurückgegriffen. In vorherigen Kapiteln wurden diese Lernverfahren bereits kurz vorgestellt und in diesem Zusammenhang auch die Clusteranalyse. Das Clustern von Daten hat den Vorteil, dass Ausreißer nicht einfach der Klasse zugeordnet werden, von derer die Abweichung am geringsten ist, so wie es bei der Klassifizierung der Fall wäre. Daten, die nicht eindeutig einem Cluster zugeordnet werden können, erhalten stattdessen ein eigenes Cluster oder werden als Ausreißer gekennzeichnet. Demnach bilden Cluster, welche lediglich ein Datum oder ein klar definiertes Minimum an Daten enthalten die Ausreißer.
Ein gängiger Clustering-Algorithmus ist der sogenannte DBSCAN Algorithmus, was für den Na-

men „Density-Based Spatial Clustering of Applications with Noise“ (*sinngemäß:* „Auf Dichte basierende, räumliche Clusteranalyse in Anwendungen mit Rauschen“) steht. Dieses Verfahren wird genutzt, um Samples mit hoher Dichte zu finden und diese zu Clustern zu erweitern. (scikit-learn developers 2019) Mittels des Epsilon Werts wird die maximale Distanz eines Samples festgelegt. DBSCAN erzeugt zunächst einige Kernpunkte, die es für zentral hält. Anhand dieser werden die ersten Samples, welche laut Epsilon erreichbar sind, diesem Kernpunkt zugeordnet und erzeugen somit ein Cluster, vorausgesetzt, die zweite Bedingung wäre ebenfalls erfüllt. Bei dieser handelt es sich um einen weiteren Wert, welcher vorgibt, wie viele Samples mindestens notwendig sind, um ein Cluster bilden zu können. Am Ende ist hierdurch eine endliche Anzahl an Clustern entstanden.

5.2.5 Long Short-Term Memory

In der Konzeptionierung eines LSTMs sind gegenüber den vorherigen Modellen weitere Aspekte zu beachten, da es aufgrund seines Aufbaus ein wesentlich tieferes neuronales Netz ist, wie bspw. das vorgestellte Modell des Multilayer Perceptrons. So besitzt ein LSTM möglicherweise ein eigenes Feedforward-Netz, in welches die Ausgaben der LSTM Schichten hineinfließen. Die LSTM Schicht als solche wurde im vorherigen Kapitel bereits näher betrachtet. Mit dem letzten Dense Layer kann die Ausgabe des Netzes dann auf die gewünschte Anzahl an Ausgabeneuronen skaliert werden.
Grundsätzlich ist der Aufbau des Long Short-Term Memorys dennoch überschaubar hinsichtlich des Modells des CNNs. Es werden zumindest im klassischen Einsatz keinerlei Faltungen oder dergleichen vorgenommen. Gemischte Ansätze zwischen dem, im Bildverarbeitungskontext angesiedelten, Convolutional Neural Network und dem, auf sequenziellen Daten arbeitendem, LSTM wären aber ebenfalls durchaus denkbar. Als Beispiel sei hier die Videoverarbeitung genannt, bei der es sowohl Bilddaten als auch sequenzielle Daten gäbe. Die Entscheidung, welcher der Kontexte mehr Aussagekraft für eine Klassifizierung bieten würde, kann je nach Anwendungsfall vorher getroffen werden. Sei es durch eine Gegenüberstellung beider Verfahren, oder eben durch eine Nutzung als hybride Architektur.
Der Aufbau des im Rahmen dieser Arbeit entworfenen Long Short-Term Memory Architektur beschränkt sich auf eine oder mehrere LSTM Schichten, gefolgt von mindestens zwei Dense Layern. Der Test würde entscheiden, wie die Anzahl der Schichten zu wählen ist. Das Ergebnis einiger Trainings zeigte, dass bereits ein einzelner LSTM Layer eine gute Wahl darstellt und das Netz im Vergleich zur tieferen Schichtung deutlich schneller lernen kann. Entscheidend für den Erfolg des Trainings waren, neben der Wahl der Anzahl jeder Schicht, vor allem die Anzahl der Gewichte. Da gerade die LSTM Schicht aufgrund ihrer Beschaffenheit einen großen Anteil an der

Performance beansprucht, war die Entscheidung, hinsichtlich zur Verfügung stehender Rechenleistung, ebenfalls kostentechnischer Natur. Die LSTM Schicht bekam letztlich 100 Neuronen und der erste Dense Layer wurde mit ebenfalls 100 Neuronen darauf angepasst.
Während des Trainings, welches sich als äußerste Herausforderung entpuppte, wurde die Anzahl der Gewichte und Schichten oft angepasst. Genannte Werte entsprechen somit dem Endresultat. Des Weiteren wurde hier ebenfalls Gebrauch von einem Dropout Layer gemacht, welcher sich, ohne großen Kostenaufwand, deutlich auf das Ergebnis auswirkte. Dieser wurde hinter dem LSTM Layer platziert. Die Wahrscheinlichkeit, mit der die Ausgabe der LSTM Schicht ausgesetzt wird, wurde auf 50% festgelegt und nach einigen Versuchen mit anderen Werten auch dabei belassen. Der letzte Dense Layer war aufgrund der gewünschten Ausgabe von zwei Werten bereits im Voraus auf 2 Neuronen festgelegt. Für den ersten Dense Layer wurde die ReLU Aktivierung genutzt. Die Ausgabeschicht erhielt eine softmax Aktivierung, da sie sich bereits beim CNN bewährte.
Für das Einspeisen der Daten in das LSTM war es notwendig, eine weitere Dimension zu erzeugen, da das LSTM keine Vektoren entgegennimmt, sondern ausschließlich Matrizen. Zur Einspeisung von Vektoren wären ebenfalls Einbettungs-Schichten, sogenannte Embedded Layer, verwendbar. Diese sind jedoch an gewisse Parameter wie bspw. die Stapelgröße (*engl.:* „batch size“) oder die Wörterbuchlänge gebunden. Nach wenigen Trainingsversuchen mit einem Embedding Layer im Modell, welcher das Training des LSTM zu blockieren schien, wurde stattdessen der vorher genannte Ansatz gewählt.

5.3 Implementierung

5.3.1 Multilayer Perceptron

Die Implementierung einer Feedforward-Architektur war in frühen Tagen um ein Vielfaches komplexer, als sie es heutzutage ist. Mussten anfangs noch alle Kalkulationen, welche innerhalb des neuronalen Netzes stattfinden, von Hand geschrieben werden, so werden diese mittlerweile von etwaigen möglichen Bibliotheken im Hintergrund entworfen. Das hat, neben dem verminderten Implementierungsaufwand, auch den Vorteil, dass es in der Berechnung der neuen Gewichtung der einzelnen Neuronen zu weniger Fehlern kommt. Abgesehen davon ist der verminderte Implementierungsaufwand immer auch eine Bereicherung hinsichtlich der Übersichtlichkeit des Programmcodes, welches die maschinellen Lernverfahren nutzt und somit ist das entworfene System auch leichter erweiterbar.
Die im vorherigen Kapitel bereits vorgestellte Bibliothek TensorFlow übernimmt zum Beispiel die Berechnung der Matrizen und Vektoren, den sogenannten Tensoren. Ferner stellt es bereits eine Vielzahl an Bausteinen für den Aufbau einer neuronalen Architektur zur Verfügung.

Im ersten Schritt wurden also die benötigten Bibliotheken geladen, um im weiteren Verlauf mit Ihnen zu arbeiten. Nun soll die in der Theorie entworfene neuronale Architektur mittels der Bibliothek TensorFlow modelliert werden. Hierzu bietet Keras, als Erweiterung zu TensorFlow, das Modul *layers* an, welches wiederrum eine Vielzahl vorgefertigter Schichtarten beinhaltet. Bei der Feedforward-Architektur fließen die Ausgaben der einzelnen Schichten ausschließlich in die nachfolgenden Schichten ein, die Verarbeitung verläuft also linear und unidirektional. Die Aktivierungsfunktion kann jeder Schicht bereits bei der Erstellung mitgegeben werden. Hier wurde anfangs die ReLU-Funktion gewählt. Die Schichten werden dem Modell bereits in einem Konstruktor übergeben. Dies ermöglicht Keras durch den Modelltyp „Sequential“, welcher einen linearen Stapel von Schichten repräsentiert. (Chollet u. a. 2015) Mittels der Funktion textitfit, welche durch das Modell aufgerufen wird, wird das Training angestoßen. Hierbei müssen der Funktion bereits die Trainingsdaten in korrekter Dimensionierung, die zugehörigen Labels in gleicher Reihenfolge und die Anzahl der zu trainierenden Epochen mitgegeben werden. Ist das Training dann abgeschlossen, kann mit der Evaluation fortgefahren werden. Mittels der Funktion textitevaluate werden die Testdaten und deren Labels dem Modell übergeben. Die Ausgabe der Testphase erfolgt als Metriken, die ebenfalls vorweg definiert werden. In diesem Fall wurde der Verlust und die Accuracy des Netzes ausgegeben.
Sobald die Architektur des Modells mittels zahlreicher Anpassungen perfektioniert wurde und nicht weiter verändert wird, kann diese erneut auf die Daten trainiert werden und als vortrainiertes Modell abgespeichert werden, um im späteren Verlauf nicht notwendigerweise vor jeder Nutzung des Netzes die Trainingsphase erneut durchlaufen zu müssen. Dies wird mittels der Funktion textitsave erreicht, welche wieder vom Modell aufgerufen wird. Keras bietet mit dieser Funktion das Speichern des aktuellen Zustandes der Gewichte und der Architektur des neuronalen Netzes in einer Datei des Formats H5 (*.h5), welche durch die zugehörige Funktion textitload wieder eingelesen werden kann und das vorher gespeicherte Modell zurückgibt. Das mittels der textitload Funktion geladene Modell kann erweitert und trainiert werden, sodass ein Training gezielt unterbrochen und später weitergeführt werden kann.

5.3.2 Vorverarbeitung der Daten

Da die erste Architektur zum Lernen der Merkmale in den Daten in seinem Grundzustand entworfen wurde, gilt es im Weiteren, die Daten auf die Eingabeschicht der Architektur vorzubereiten. Dies wird im sogenannten „Pre-Processing“ vorgenommen, welches gleichzeitig dazu dient, einen Überblick über die Daten zu gewinnen. Im ersten Schritt wird hierbei das vorliegende Format der Daten betrachtet. Dies ist notwendig, um eine geeignete Methode zu finden, sie in eine Datenstruktur zu importieren. Meist liegen Trainingsdaten in gängigen Formaten wie dem

Plaintext TXT-Format oder dem CSV-Format vor. Im letzteren Fall können die Daten wiederum von Programmen wie Excel aus dem Hause der Microsoft Corporation grafisch aufbereitet in einer Tabellenstruktur angezeigt werden. Um die Daten ähnlich aufbereitet auch im Programmcode ausgeben lassen zu können, wurde die Python Bibliothek Pandas verwendet, welche eine geeignete Datenstruktur mit dem „Dataframe“ anbietet, die wiederum zweidimensionale Daten in tabellarischer Form anzeigen kann. Die Bibliothek wurde bereits zu Beginn des Kapitels näher beleuchtet.
Bei dem Import der Daten muss das vorliegende Format entsprechend berücksichtigt werden und in diesem Fall die CSV-Datei mittels Pandas Funktion textitread_csv eingelesen. Hierbei kann zusätzlich angegeben werden, anhand welchen Kriteriums die Spalten separiert werden sollen. Meist sind im CSV-Format die Daten komma-separiert, was auf den Namen „Comma-Separated Values“, für welches CSV die Abkürzung bildet, zurückzuführen ist. Nicht unüblich ist jedoch auch das Vorliegen der Separation durch Semikola. Der sogenannte Separator kann der Funktion einfach mitgegeben werden, sodass die Daten entsprechend separiert in die Datenstruktur aufgenommen werden. Die auf diese Art eingelesenen Daten sind nun bereits in einem Dataframe vorliegend, sodass sie zur Kontrolle direkt tabellarisch angezeigt werden können. Da die Daten in diesem Fall bereits über eine Klasse verfügen, also kategorisiert sind, können sie auch kategorisiert betrachtet werden. Durch die Pandas Funktionen groupby und size lassen sich die Kategorien gruppieren und deren Anzahl an Datenpunkten jeweils aufzeigen. Die Aufteilung verrät direkt, dass jede Kategorie dieser Daten denselben Anteil aufweist. Die einzelnen Kategorien wurden bereits im Vorfeld näher erläutert. Bibliotheken wie „palettable“ können diese Verteilung dann noch grafisch anschaulicher aufbereiten, wie bspw. durch ein Tortendiagram.
Im Folgenden gilt es nun die Dimensionen der Daten abzuschätzen. Um ein erfolgreiches Training zu gewährleisten, ist es unerlässlich, Daten in ausreichender Zahl und Variation zur Verfügung zu stellen, um ein Overfitting zu vermeiden. Da Elektroenzephalographien meist über einen längeren Zeitraum durchgeführt werden und die dabei auftretenden Potenziale praktisch zu jedem Zeitpunkt erfolgen können, wäre eine Einspeisung der Rohdaten in das neuronale Netz vermutlich nicht zielführend. Besser jedoch wäre eine Zerlegung in Fragmente, die klein genug sind, um sie ausschließlich einer Kategorie zuweisen zu können. Hierbei unterscheiden sich vorverarbeitete, öffentliche Datensätze, wie der für diesen Versuchsaufbau verwendete, gegenüber rohen Messdaten oder sogar Echtzeit-Daten. Wo der öffentliche Datensatz bereits kategorisiert war und keinerlei Zerlegung mehr bedarf, müsste ein Rohdatensatz noch sinnvoll unterteilt werden. Um ein Training und anschließenden Test auch auf neuen Daten zu ermöglichen, müssten diese zunächst in das Format gebracht werden, welches auch die maschinellen Lernverfahren erwarten. Wie in vorherigen Kapiteln bereits genannt, umfassten die einzelnen Fragmente jeweils 178 Samples und entsprachen einem Intervall von einer Sekunde. Die Python Bibliothek „NumPy“

bietet zahlreiche Funktionen, die das Verarbeiten und Rechnen mit mehrdimensionalen Daten ermöglichen. Um die Daten unterteilen zu können, würden sie aus dem Dataframe in eine geeignetere Datenstruktur, wie dem NumPy-Array, gebracht werden. Diese gleicht dem klassischen Array, wie es aus anderen Programmiersprachen bekannt ist, bietet jedoch noch eine Vielzahl an Manipulationsmöglichkeiten. Die Umwandlung in ein NumPy-Array erfolgt durch die Pandas Bibliothek mit der Funktion textitvalues, welche vom Dataframe selbst aufgerufen wird und seine Werte als NumPy Array zurückgibt.

5.3.3 Fast Fourier Transformation

Die schnelle Fourier Transformation ist als spezielles Verfahren der Diskreten Fourier Transformationen durch seinen Bruchteil an Rechenschritten eine vorteilhafte Wahl für die Transformationen von Signalen dieses Ausmaßes. Durch die hohe Anzahl an Signalen kommt jedoch auch bei der schnellen Fourier Transformation schon eine beträchtliche Verarbeitungszeit zustande. Die Transformation ist in Python Bestandteil der Bibliothek NumPy, welche im Vorfeld bereits verwendet wurde, um Arrays zu manipulieren, beziehungsweise Berechnungen mit ihnen durchzuführen. Das Modul *FFT* von NumPy bietet hierzu einige Funktionen für die Verarbeitung von Signalen mittels Diskreter Fourier Transformation von eindimensionalem Raum bis hin zu n-dimensionalem Raum. Es gibt ebenfalls die Möglichkeit, die inverse DFT durchzuführen. Für diesen Anwendungsfall wurde auf die eindimensionale DFT zurückgegriffen, welche durch die Funktion textitnumpy.fft.fft repräsentiert wird. Da die Rückgabe der Fast Fourier Transformation für jedes Signal einer komplexen Zahl entspricht, ist es für die folgende Verarbeitung notwendig, den absoluten Anteil der komplexen Zahl zu erhalten, während eine Umwandlung in reale Werte diesen Anteil verwerfen würde. Somit wird nach der Transformation aller Signale der absolute Anteil der gesamten Ausgaben in eine neue Datenstruktur aufgenommen. Ein Kernmerkmal der FFT ist, dass die Ausgabe eine Symmetrie aufweist. Die 178 Samples des jeweiligen Signals werden also auf eine Ausgabe von 178 Werten transformiert, welche symmetrisch ist und somit die 89 ersten Werte spiegelt. Für das spätere Training können wie im Vorfeld bereits genannt daher die zweiten 89 Werte verworfen werden, da sie keinen Mehrwert mehr bieten. Die sich daraus ergebenden 267 Werte pro Datenpunkt (178 Samples + 89 FFT Samples) können nun jeweils wieder als Zeile in ein Dataframe geschrieben werden, von dem aus sie mittels Pandas‘ Exportfunktion to_csv unter Angabe des gewünschten Separators wieder in ein .CSV-Format gebracht werden. Diese Datei enthält nun demnach die vorverarbeiteten Daten und kann im weiteren Verlauf wieder eingelesen und dem neuronalen Netz eingespeist werden.

5.3.4 Convolutional Neural Network

Um auf den vorliegenden Daten trainieren zu können, mussten diese, wie bereits erwähnt, zunächst in ein geeignetes Format gebracht werden. Hierbei kommt die Bibliothek Matplotlib zum Einsatz, welche sich dazu eignet, Zeitreihen grafisch darzustellen. Da die Zeitstempel für die Klassifizierung von Anfällen keinen Mehrwert bieten, wurde hier stattdessen kurzerhand ein Array mit einer Range von 1 bis 267 erzeugt, welches als Dimension Zeit fungiert. Durch das „Plotten" der Daten in Relation zum kürzlich erzeugten Array wird die Amplitude sichtbar. Die Daten enthalten, wie eingangs erwähnt, jeweils 178 Samples aus der Messung selbst, gefolgt von 89 FFT-Werten. Der erzeugte Plot dient hier jedoch nur der visuellen Betrachtung, während die hinterlegte „Figure" (*sinngemäß:* „Abbildung") diese Darstellung logisch repräsentiert. Diese Figure kann nun über ein „Canvas" (*sinngemäß:* „Leinwand") gezeichnet und in ein Array transferiert werden, welches zu diesem Zeitpunkt dann bereits das Bild in Bytes beinhaltet.
Durch eine zusätzliche Bibliothek namens PIL und ihrem Modul *Image* kann nun aus einem Byte-Array wieder ein Bild erzeugt werden. Dieser Workaround war notwendig, um das Bild aus dem Plot zu extrahieren, damit es im Folgenden manipuliert werden kann. Für die Klassifizierung von Anfällen in EEG-Daten ist es keinesfalls notwendig, die RGB oder gar RGBA (RGB + Alpha) Kanäle zu besitzen. Diese würden eine Komplexität mit dem Faktor mal drei, bei RGBA sogar mal vier, bedeuten. Da die Darstellung der Werte ohnehin einfarbig in blau erfolgte, war es demnach sinnvoll, die Bilddaten zunächst in Graustufen umzuwandeln. Hierfür bot das Image Modul von PIL die entsprechende *convert* Funktion. Das daraus erzeugte Bild konnte anschließend über die NumPy Funktion *asarray* in ein Array zurückgeschrieben werden. Durch eine simple Schleife ließ sich somit der gesamte Datensatz in graustufige Bilddaten umwandeln, welche mittels *vstack* Funktion von NumPy wieder zu einem gesamten Datensatz gestapelt werden konnten.
Das Modell des CNN wurde daraufhin, dem Entwurf entsprechend, erzeugt. Hierzu wurde, wie beim MLP bereits, der Sequential Konstruktor von Keras genutzt, um aus den sequenziell angeordneten Schichten direkt das Modell erzeugen zu können. Die Schichten umfassen die vier Convolutional Layer, welche jeweils gefolgt von einem Max-Pooling Layer verwendet werden. Da aufgrund der vorherigen Konvertierung in graustufige Bilder nur noch zweidimensionale Daten vorliegen, können hier jeweils die zweidimensionalen Layers verwendet werden. Die Klassen hierfür werden in Keras als Conv2D und MaxPooling2D betitelt. Während der Convolutional Layer jeweils die Anzahl der Filter, die Kernel-Größe von 3x3 und die ReLU Aktivierungsfunktion erhielt, wurde dem Max-Pooling Layer lediglich die Pool-Größe von 2x2 mitgegeben. Als nächstes folgt der Flatten Layer, welcher durch die gleichnamige Klasse erzeugt wird und keine Parameter benötigt. Anschließend wird der bereits aus dem MLP bekannte Dropout Layer mit der Dezimalzahl 0.5 als Parameter erzeugt, welche der 50 prozentigen Aussetz-Wahrscheinlichkeit

entspricht. Zum Schluss folgen die Dense Layer mit 120 Neuronen im ersten und 2 Neuronen im zweiten Layer. Die Aktivierungen werden ebenfalls als Parameter mitgegeben und entsprechen dem Entwurf mit der ReLU Funktion im ersten Layer und der softmax Funktion in der Ausgabeschicht.
Anders als beim vorherigen Modell und beim folgenden LSTM, ist das Trainieren auf Bilddaten deutlich zeitintensiver, zeigt dafür aber wesentlich schneller Erfolge. Generell eignen sich CNNs durch diesen Umstand am ehesten für das Trainieren auf wenigen Daten. (Chollet 2018) Die Anzahl der zu trainierenden Epochen wurde demnach auf 200 Epochen gesetzt. Die Auswertung einiger Testphasen bestätigte diese Annahme, wie auch bereits erwähnt, wodurch die Anzahl der Epochen auf 150 reduziert wurde.

5.3.5 Long Short-Term Memory

Ähnlich wie beim CNN kann die Architektur des LSTM ebenfalls mittels des Sequential Konstruktors von Keras entworfen werden. Statt der beim CNN gewählten Faltungsschichten, kommt in diesem Modell die LSTM-Schicht zum Einsatz. Sie bietet bereits all jene Merkmale, die ein LSTM repräsentiert und reduziert somit den Implementierungsaufwand enorm. Die einzelnen Gatter einer LSTM-Zelle, die im Vorfeld bereits vorgestellt wurden, sind in diesem Layer bereits integriert und auch die Aktivierungsfunktionen werden bereits berücksichtigt. Dieser Schicht sind somit nur noch die Anzahl der zu verwendenden Neuronen und die zu erwartenden Eingabedimensionen mitzugeben. Mittels des Parameters *return_sequences* kann zusätzlich gewählt werden, ob der Layer als Ausgabe ebenfalls eine Sequenz liefern soll. Dies war hinsichtlich des darauffolgenden Dense Layers allerdings nicht gewünscht. Wäre ein zweiter LSTM Layer, dem ersten nachfolgend, zum Einsatz gekommen, hätte die Ausgabe einer Sequenz im ersten Layer wiederum Sinn ergeben, da dieser ebenfalls eine Sequenz verarbeiten würde. Auf Grundlage einiger Versuche folgt dem LSTM Layer im endgültigen Modell allerdings ein Dropout Layer, statt wie erwartet der Dense Layer. Dieser wird durch den gleichnamigen Layer von Keras in das Modell integriert. Als Parameter erhält dieser lediglich die Wahrscheinlichkeit, mit der er die Ausgabe vorangehender Schichten auslässt. Diese beträgt im Versuchsaufbau 50% und wird dem Layer als Dezimalzahl 0.5 übergeben. Darauffolgend werden nun die Dense Layers in das Modell aufgenommen. Da aufgrund der Perfomance die LSTM-Schicht auf 100 Neuronen beschränkt wurde, werden auch dem Dense Layer lediglich 100 Neuronen, auch bezeichnet als „units“, als Parameter übergeben. Die Aktivierungsfunktion des ersten Dense Layers bildet die ReLU Funktion, während für die Ausgabeschicht eine softmax Aktivierung gewählt wurde. Die Ausgabeschicht ist analog zu ihrem Vorgänger ebenfalls ein Dense Layer und bekommt zwei Neuronen. Diese repräsentieren, wie auch in den vorherigen Implementierungen, die Zustände „Anfall vorhanden“ und

„Anfall nicht vorhanden“ für die jeweilige, verarbeitete Sequenz.
Ebenfalls wie in den vorherigen Implementierungen, wird das Modell mittels *compile* übersetzt und anschließend das Training mittels *fit* gestartet. Die zusätzliche Dimension, die der LSTM Layer erwartet, wird durch die Angabe *None* erzeugt, welche den Arrays mit den Trainings- und Testdaten bei der Parameterübergabe angehängt wird. Die Manipulation der Arrays wird von der Bibliothek NumPy durch diese Schreibweise erheblich erleichtert. Mittels der Keras Funktion *evaluate* wird auch hier wieder die Testphase eingeleitet und das Ergebnis mitsamt der Genauigkeit des Netzes zurückgegeben. Wie bereits bekannt wird das Netz auch in diesem Fall wieder für eine spätere Wiederverwendung gespeichert.

5.4 Auswertung der Testresultate

Um ein Gefühl für die Genauigkeit der einzelnen Modelle zu bekommen, reicht es hinsichtlich des Anwendungsfalls nicht aus, die Accuracy des jeweiligen Modells zu erhalten. Diese gibt zwar korrekterweise an, mit welcher Genauigkeit das Netz in der Lage ist, Daten zu klassifizieren, jedoch wird hierbei nicht unterschieden, welcher Art die falsch klassifizierten Daten angehörten. Dies mag in vielen Bereichen auch gar nicht notwendig sein. Im medizinischen Umfeld jedoch ergibt sich ein Unterschied hinsichtlich dessen, ob eine vorliegende Erkrankung übersehen wurde, oder nur ein Fehlalarm die Ursache eines Fehlers ist. Wie bereits in der Einführung erwähnt, werden diese im Fachbereich als false-positives und false-negatives bezeichnet, wobei die false-negatives, auch namentlich erkennbar, die Fehler mit negativeren Auswirkungen repräsentieren. In diesem Beispiel wäre dies das Übersehen eines vorliegenden Anfalls.
Um dieses Wissen aus der Testphase zu extrahieren, wird statt der *evaluate* Funktion die *predict* Funktion von Keras verwendet. Diese wird auf denselben Testdatensatz angewandt und liefert als Ergebnis eine Collection mit einer Vorhersage zu jedem Datenpunkt. Da diese Vorhersage jeweils zwei Ausgaben beinhaltet, nämlich den Wert für „Anfall vorhanden“ und „Anfall nicht vorhanden“, muss im Folgenden zunächst der höhere Wert mittels *argmax* Funktion bestimmt werden. Die Position, an der dieser Wert vorliegt, repräsentiert die Ausgabe. Diese Ausgabe kann dann mit dem entsprechenden Label abgeglichen werden. Sind die Werte ungleich, liegt ein Fehler vor. Ist dann die berechnete Ausgabe „Anfall nicht vorhanden“, handelt es sich um einen false-negative Fehler, da das Netz einen Anfall nicht erkannt hat, dessen Vorliegen das Label gewährleistet. War die Ausgabe des Netzes „Anfall vorhanden“, handelt es sich demnach um einen false-positive Fehler. Durch das Aufsummieren dieser Fehler kann eine Aussage darüber getroffen werden, wie viele vorliegende Anfälle missachtet wurden und wie viele Fehlalarme es gab.

5.5 Hybrider Verbund der Architekturen

Um die Genauigkeit des Gesamtsystems noch weiter zu steigern, werden die einzelnen Lernverfahren, wie eingangs erwähnt, sinnvoll miteinander verbunden, um mit ihrer Hilfe eine noch genauere Gesamtaussage treffen zu können. Hierfür werden die jeweiligen Vorhersagen unterschiedlich gewichtet. Die einzelnen Verfahren werden in den folgenden Absätzen als Subsysteme behandelt.
Im ersten Ansatz ist jede Aussage gleichberechtigt und somit wird die Gesamtaussage entsprechend der einzelnen Aussagen getroffen. Wird von einem Subsystem ein Anfall erkannt, so wird die Aussage des Gesamtsystems zu der Aussage des Subsystems und meldet, dass ein Anfall erkannt wurde. Dies mindert die Anzahl der kritischen false-negative Vorhersagen, könnte aber die Anzahl der false-positives in die Höhe treiben. Möglicherweise vorliegende Anfälle werden somit am zuverlässigsten erkannt. Wenn dies die einzige Anforderung an das System sein sollte, wäre dieser Ansatz bereits die optimale Architektur.
Der nächste Ansatz gewichtet die einzelnen Subsysteme ebenfalls gleich, erhält aber einen Schwellwert für das Auslösen der Gesamtvorhersage. Läge dieser Schwellwert bei zwei, so müssten zwei Subsysteme einen Anfall erkennen, um in der Aussage des Gesamtsystems ebenfalls auf einen Anfall hinzuweisen. Dies könnte die false-positives abschwächen und möglicherweise keine negativen Auswirkungen auf die Anzahl der false-negatives haben. Denkbar wäre jedoch auch, dass in bestimmten Fällen nur eines der Subsysteme aufgrund seiner für diesen Fall sensibleren Charakteristik ausschlägt und somit von den anderen Systemen überstimmt werden würde.
Den letzten Ansatz bildet eine prioritätsbasierende Lösung. Da die Subsysteme allesamt eine Genauigkeit bei der Vorhersage der Anfälle in den Testdaten aufweisen, kann anhand dieser eine Prioritätsliste erstellt werden. Wird nun von einem Subsystem ein Anfall erkannt, welches in der Priorität unter einem gewissen Schwellwert liegt, würde diesem keine Beachtung geschenkt werden. Da dieser Ansatz allerdings zur Folge hätte, dass das zu vernachlässigende Subsystem gänzlich weggelassen werden könnte, da ihm ohnehin nie Beachtung geschenkt werden würde, wird zusätzlich ähnlich zum zweiten Ansatz ein weiterer Schwellwert angegeben, welcher ebenfalls die Gesamtaussage beeinflussen kann. Angenommen, der Schwellwert für die Anzahl der Systeme wäre zwei und es würden zwei Subsysteme einen Anfall verzeichnen, deren Gesamt-Priorität jedoch immer noch nicht ausreichen würde. In diesem Fall würde die Gesamtaussage dennoch wahr werden, da der Schwellwert von zwei erreicht wurde.
Es ist davon auszugehen, dass sich einer dieser Ansätze in dem Anwendungsfall dieser Arbeit bewähren wird. Jedoch muss berücksichtigt werden, dass sich die Eignung des Ansatzes mit den zu klassifizierenden Daten unterscheiden kann. Würde das entworfene System bspw. aus einer klassischen Neurologie in eine Kinder-Neurologie verlagert werden, so könnte angenommen werden,

dass die Grundaktivität eines Kindes weitaus energetischer ausfällt, was zu einer schwierigeren Unterscheidung zwischen Artefakten und Potenzialen führen würde. Die einzelnen Lernverfahren wären demnach vielleicht nur noch bedingt so geeignet, wie sie es unter vorherigen Bedingungen waren und je nach gewähltem Ansatz könnte den ungeeigneteren Subsystemen zu viel Beachtung geschenkt werden. Dieser Gedankengang wird während der Evaluation noch einmal genauer betrachtet und gegebenenfalls bestärkt oder verworfen werden.

5.6 Hyperparameter Tuning

Üblicherweise ist das Finden des ideal entworfenen Modells, hinsichtlich der Problemlösung eigener Anwendungsfälle, mit zahlreichen Versuchen verbunden. Hierbei kann Glück eine große Rolle spielen, doch im Regelfall dauert dieser Prozess lange an. Dies liegt zum einen darin begründet, dass die neuronalen Netze eine gewisse Zeit brauchen, bis sie sich als lukrativ oder hinfällig erweisen. Zusätzlich dazu werden neuronale Netze oftmals für neue Anwendungsbereiche konzipiert oder müssen neuen Gegebenheiten angepasst werden, sodass es unmöglich ist, auf bereits bestehende Modelle zurückzugreifen. Um einer utopischen Anzahl an Versuchen durch den Anwender entgegenzuwirken, können auch diese wieder automatisiert werden. Dies geschieht entweder durch eigens geschriebene Funktionen oder indem auf bereits bestehende Ansätze zurückgegriffen wird, wie es in diesem Beispiel der Fall ist.
Eine mögliche Bibliothek, welche das Ausprobieren aller möglichen Modellkonstellationen übernehmen kann, stellt die Keras Tuner Bibliothek dar, die eingangs bereits vorgestellt wurde. Zunächst werden der Bibliothek sinnvolle Bereiche vorgegeben, in denen die Versuche erfolgen sollen. Diese Bereiche werden auch als Hyperparameter behandelt, weswegen das Tuning auch oft als Hyperparameter Tuning (*sinngemäß:* „Hyperparameteroptimierung“) bezeichnet wird. Um die Werte innerhalb der definierten Bereiche auszuprobieren, bietet die Keras Tuner Bibliothek mehrere Möglichkeiten. Im Rahmen dieser Arbeit werden diese Versuche zufällig durchgeführt. Dies wird als zufällige Suche bezeichnet und mittels gleichnamigen Konstruktors der Klasse *RandomSearch* initialisiert. Diese gehört in der Keras Tuner Bibliothek dem Modul *tuners* an. Ein weiterer möglicher Tuner wäre der Hyperband Tuner, welcher im Vergleich zur zufälligen Suche die erzeugten Modelle lediglich einen Bruchteil der konfigurierten Epochen trainieren lässt und nur die zu diesem Zeitpunkt besten Modelle weiter trainiert. Die Anzahl der Epochen, die ein Modell je Versuch trainiert werden soll, werden als Parameter im Konstruktor übergeben. Zusätzlich kann eine maximale Anzahl an Versuchen mitgegeben werden, nach der die Suche abbrechen und die Resultate bis zu diesem Zeitpunkt ausgeben soll. Da ein einzelner Versuch mit gewählten Parametern in seinem Erfolg auch variieren kann, ist es zusätzlich möglich, eine gewünschte Anzahl an Ausführungen pro Versuch mitzugeben, um dieser Variation vorzubeugen. Jeder Ver-

such wird dann mit denselben Parametern der Anzahl entsprechend oft wiederholt.
Unabhängig von dem Typ des Netzes wurde im Rahmen dieser Arbeit die maximale Versuchsanzahl auf 100 beschränkt und jeder Versuch wurde doppelt ausgeführt. Dies ergab einen guten Kompromiss zwischen dem Erfolg und der Dauer eines Tunings. Unterschieden hat sich das Tuning jedoch jeweils in den Hyperparametern.
Beim MLP wurde zunächst die Anzahl der Dense Layer variabel auf einen Wert zwischen 1 und 4 festgelegt und die Anzahl ihrer Neuronen jeweils auf einen Bereich zwischen 512 und 4096 Neuronen, wobei die Schrittweite der einzelnen Versuche jeweils mit 128 spezifiziert wurde. Zu guter Letzt wurde die Lernrate durch eine Liste von möglichen Raten ersetzt. Diese Liste umfasste sowohl die Lernrate von 0.01 als auch von 0.001 und 0.0001. Das beste Modell erhielt vier Dense Layer mit jeweils 3968 Neuronen und eine Lernrate von 0.0001 und kam damit auf 98,26% Accuracy in lediglich fünf Epochen.
Das CNN erhielt aufgrund seiner gezeigten Instabilität keine variable Anzahl an Convolutional Layers, Max-Pooling Layers und Dense Layers. Stattdessen wurde jedoch die Anzahl der Filter und die Größe des Kernels des Convolutional Layers variiert. Der Max-Pooling Layer blieb jeweils unverändert. Die ersten beiden Convolutional Layer erhielten einen Wertebereich von 32 bis 128, hinsichtlich der Anzahl der zu verwendenden Filter, mit einer Schrittweite von 16. Als Kernel-Größe konnte entweder 3x3 oder 5x5 gewählt werden. In den hinteren beiden Convolutional Layern erhöhte sich der Wertebereich der Filteranzahl auf maximal 256 bei gleichbleibender Schrittweite und Möglichkeiten der Kernel-Größe. Zusätzlich wurde auch dem Dense Layer ein Spielraum hinsichtlich seiner Neuronenzahl eingeräumt, welche sich auf einen Bereich von 100 bis 1000 bei einer Schrittweite von 20 beschränkte. Aufgrund der Vielzahl der möglichen Parameterkonstellationen wurden beim CNN nur noch die Lernraten 0.01 und 0.001 getestet. Die Convolutional Layers des besten Modells erhielten der Reihe nach 48, 80, 224 und 128 Filter bei einer Kernel-Größe von 3x3 in den ersten drei Layers und 5x5 im letzten. Die Anzahl der Neuronen im Dense Layer beläuft sich auf 100 Neuronen und als Lernrate wurde 0.001 gewählt. In dieser Konstellation schaffte es das Modell in 5 Epochen bereits auf einen Genauigkeitswert von 98,66%.
Im letzten Modell, dem LSTM, wurden die Hyperparameter der Suche wie folgt konfiguriert. Sowohl der LSTM Layer als auch die Dense Layer erhielten einen Wertebereich von 20 bis 1000 Neuronen bei einer Schrittweite von 20. Während der LSTM Layer allein blieb, wurde die Anzahl der Dense Layer variabel auf einen Wert zwischen 1 bis 4 gesetzt, die Ausgabeschicht ausgenommen. Die Lernrate konnte wie beim MLP aus den Werten 0.01, 0.001 und 0.0001 gewählt werden. Hier erhielt das beste Modell drei Dense Layer mit jeweils einer Anzahl von 880 Neuronen, während der LSTM Layer eine Anzahl von 60 Neuronen besitzt. Als Lernrate wurde hier die 0.001 gewählt und das Modell erreicht eine Accuracy von 96,20% nach den 5 Epochen.

6 Evaluation

6.1 Ergebnis

Alle im vorherigen Kapitel aufgeführten Lernverfahren wurden sowohl einzeln als auch im hybriden Zusammenschluss ausgewertet. Das Training erfolgte separat für jedes Subsystem. Die zu trainierenden, sowie die zu testenden Daten wurden im Vorfeld vorbereitet und gemischt, um einer glücklichen Verteilung entgegenzuwirken. Außerdem wurden die Daten auf einen Wertebereich zwischen 0.1 und 0.99 normiert. Die Trainings- und Testphase wurde für jedes System mit demselben Segment von Daten durchgeführt. Zu Beginn jeder Phase erhielten somit alle Verfahren die gleichen Bedingungen. Die Anzahl der Trainingsdaten belief sich auf 10000, während die anschließende Testphase mit 1500 bisher ungesehenen Daten durchgeführt wurde.
Die Architektur des herkömmlichen Artificial Neural Network in Form des MLP zeigte sich im Test überraschend erfolgreich. Das eigens entworfene Modell schaffte es auf einen Genauigkeitswert von 98,06%. Hierbei belief sich die Zahl der false-positives auf lediglich 3, während jedoch die false-negatives bei 26 lagen. Dieser Wert ist für den Einsatz in der Praxis leider keinesfalls verwertbar. Nach einem überschaubaren Tuningprozess war ein starkes Modell entworfen, welches nach 300 Epochen eine Genauigkeit von 98,93% beim Klassifizieren der Testdaten vorweisen konnte. Im Test ergab sich außerdem ein Fehlerwert von 4 false-positives bei 12 false-negatives und damit erreicht das MLP die niedrigste Anzahl an Fehlalarmen im Vergleich zu seinen Kontrahenten. Im Vergleich zum Ergebnis des nicht getunten Modells trat hier zwar ein false-positive mehr auf, jedoch ließ sich die Zahl der false-negatives mehr als halbieren. Für die Praxis ist die Zahl 12 dennoch sehr hoch. Interessant wäre hier der Vergleich zur Erkennungsrate durch einen Experten. Jedoch unabhängig davon, ist die Bestrebung gänzlich, alle Anfälle korrekt zu erkennen.
Deutlich geringer fiel die Anzahl der Epochen beim Convolutional Neural Network aus. Mit 150 Epochen schaffte es diese Architektur auf einen Genauigkeitswert von 98,26%. Damit liegt es ohne Tuning bereits höher als sein Vorgänger. Aufgrund der Dimensionen der Bilddaten dauerte das Training des CNN am längsten. Nach nicht einmal 6 Epochen stieg die Trainings-Accuracy

bereits auf einen Wert von 98% an. Nach 200 Epochen sank der Wert jedoch wieder auf 97% ab, weshalb hier die Anzahl der zu trainierenden Epochen auf 150 reduziert wurde. Zurückzuführen könnte dies auf eine Überanpassung sein, weshalb im Vorfeld auch bereits der Dropout Layer in das Modell aufgenommen wurde. Der negative Aspekt des CNN bleibt die Umwandlung der Zeitreihen in ein geeignetes Bildformat, was im gesamten Trainingsprozess die meiste Zeit in Anspruch nahm. Der Fehlerwert lag bei ausgeglichenen 16 false-positives zu 10 false-negatives. Die Zahl der Fehlerkennungen ist somit deutlich höher, als es noch beim Vorgänger der Fall war. Jedoch ließ sich hier die Zahl der missachteten Anfälle, gegenüber dem getunten MLP, um 2 reduzieren. Nach einem, aufgrund der Anzahl der Parameter, lang ausfallendem Tuningprozess, konnte die Accuracy des CNN mit 98,06% nicht mehr den ursprünglichen Wert übertreffen und dennoch hatte das Tuning einen großen Erfolg zu verbuchen. Die Zahl der missachteten Anfälle ging auf 4 zurück, was eine deutliche Verbesserung zum vorherigen Modell bedeutet. Die Anzahl der false-positives stieg leider in diesem Rahmen auf 25, was in der Hinsicht einen Rückschritt zum vorherigen Modell bedeutet.

Das dritte und letzte neuronale Netz bildete das Long Short-Term Memory mit seiner rekurrenten Architektur. Dieses Netz schaffte es entgegen der Erwartungen nur auf 97,86% Genauigkeit und ist somit das ungenauste der drei Netze. Dies war, wie sich später herausstellen sollte, jedoch das Resultat einer nicht perfekten, zu Grunde liegenden Architektur. Das Training war ähnlich zum MLP ohne vorherige Konvertierung der Daten möglich. Was seine Zuverlässigkeit angeht, schaffte es das LSTM auf einen Fehlerwert von 9 false-negatives, musste dabei allerdings 23 false-positives verzeichnen. Dennoch sei hier angemerkt, dass die Zahl der false-negatives vergleichsweise die geringste darstellt, ginge man von den nicht getunten Modellen aus. Aufgrund der Architektur des LSTM war der anschließende Tuningprozess deutlich langwieriger, als es noch beim CNN der Fall war. Die Anzahl der Hyperparameter wurde im Vorfeld bereits reduziert, was wiederum auch ein Übersehen eines besseren Modells zur Folge haben kann. Nach 100 Iterationen mit jeweils 2 Versuchen, genau wie bei den vorherigen Modellen, war das zu diesem Zeitpunkt ideale LSTM entworfen und erreichte eine Accuracy von 98,93%. Es kommt damit auf denselben Wert, wie auch das MLP zuvor. Dieses schaffte es, wie bereits erwähnt, auf einen Fehlerwert von 4 false-positives bei 12 false-negatives. Das LSTM wiederum dreht dieses Ergebnis um und schafft es im getunten Modell auf 12 false-positives bei gerade einmal 4 false-negatives. Die Anzahl der missachteten Anfälle liegt somit gleichauf mit der des getunten CNNs. Das CNN kam hierbei jedoch nicht um seine 25 Fehlalarme herum. Damit ist das getunte LSTM gegenüber seinen Kontrahenten das zuverlässigste Netz in diesem Anwendungsfall.

Abschließend sollen die Resultate der Klassifikation nun noch einmal denen der Clusteranalyse gegenübergestellt werden. Verwendet wurde hierbei, wie eingangs erwähnt, das Clustering Verfahren DBSCAN. Aufgrund des wenig komplexen Aufbaus wurden auf die Beschreibung des

Entwurfs und der Implementierung verzichtet. Die Trainingsdaten wurden wieder in Messdaten und FFT-Daten zerlegt und anschließend der Mean jedes Datenpunkts gebildet. Der Mean der Messdaten bildet für die Clusteranalyse das erste Merkmal, während der Mean der FFT-Daten das zweite Merkmal bildet, anhand derer es Gruppierungen zu finden gilt. Hierbei sei erwähnt, dass der Mean der FFT-Daten nicht sonderlich aussagekräftig sein dürfte, weshalb diese Behauptung im Folgenden noch einmal überprüft wird, indem die Frequenz mit der jeweils höchsten Amplitude jedes Datenpunkts als Merkmal verwendet wird. Die minimale Anzahl der Datenpunkte, welche ein Cluster bilden können, wurde auf 500 angesetzt, während die maximale Distanz zwischen den Datenpunkten auf 0.005 beschränkt wurde. Diese Werte waren das Ergebnis zahlreicher Versuche und damit einhergehender Fehlschläge. Auf dem Trainingsdatensatz mit 10000 Datenpunkten ließ sich somit ein Cluster bilden, welches die unkritischen Daten repräsentieren soll. Während von 1998 möglichen zu erkennenden Anfällen 1875 korrekt als Ausreißer erkannt wurden, landeten die übrigen 123 Anfallsdaten im Cluster. 2122 unkritische Daten wurden ebenfalls als Ausreißer bezeichnet, welche wiederum als false-positives gewertet werden können. Daraufhin wurde die höchste Amplitude des Frequenzspektrums als zweites Merkmal gewählt, anstelle des Mean-Werts. Widererwarten verschlechterte sich das Ergebnis jedoch sogar im Vergleich zum vorherigen Ansatz. Während mit 1850 Datenpunkten, im Vergleich zu den vorherigen 1875, weniger Anfallsdaten erkannt wurden, sind ebenfalls 3182 unkritische Daten als Ausreißer bezeichnet worden. Die Zahl der erzeugten Cluster stieg sogar auf 3 an, womit 128 übersehenen Anfallsdaten vermutlich ein eigenes Cluster bilden konnten. Mit einem Epsilon von 0.003 und einer Minimalanzahl an Datenpunkten, welche ein Cluster bilden dürfen, von 880, ließ sich ein einzelnes Cluster erzeugen. Dieses Resultat übersah zwar lediglich ein Anfallsdatum, markierte aber zusätzlich 6224 unkritische Datenpunkte ebenfalls als Ausreißer. Es kann somit folgende Behauptung aufgestellt werden: Die Frequenzen während eines Anfalls und im Normalzustand liegen jeweils eng beieinander. Dies ist darauf zurückzuführen, dass sich bei Verwendung dieser als Merkmal zunächst mehrere Cluster bildeten, welche die Kriterien erfüllten, sowohl die maximale Distanz nicht zu überschreiten als auch die Minimalanzahl dichter Datenpunkte zu erreichen. Aufgrund der gering ausgefallenen Genauigkeit der Clusteranalyse wird diese im Folgenden außer Betracht gelassen, während die neuronalen Netze zu dem eingangs erwähnten hybriden Verbund zusammengeschlossen werden. Hierbei erhalten die einzelnen Systeme die Testdaten erneut und treffen für jeden Datenpunkt eine Vorhersage. Je nachdem, welche Gewichtung einer Vorhersage des jeweiligen Subsystems beigelegt wird, ergibt sich das in vorherigen Kapiteln bereits vorgestellte, hybride Modell.

Die schwellwertbasierte Evaluation, welche einen Anfall vorhersagt, wenn der zu Grunde liegende Schwellwert übertroffen wird, schaffte es mit einem Schwellwert von 2 auf einen Fehlerwert von 3 false-positives bei lediglich 5 false-negatives. Dies stellt im Vergleich zu den einzelnen Mo-

dellen bereits ein besseres Ergebnis dar. Lediglich die Anzahl der missachteten Anfälle kann die getunten Modelle des CNN und LSTM nicht übertreffen. Die prioritätsbasierte Evaluation fügt dem Schwellwert von 2 noch eine Priorität hinzu, welche sich an den Ergebnissen der jeweiligen Subsysteme orientiert. Das MLP erhält lediglich die geringste Priorität, während das CNN die zweithöchste und das LSTM die höchste Priorität erhalten. Das LSTM wäre somit dazu berechtigt, die globale Vorhersage allein zu bestimmen, würde es einen Anfall erkennen. Das MLP und das CNN können den Wert der globalen Aussage nicht beeinflussen, außer sie werden jeweils durch ein weiteres Subsystem in der Vorhersage bestärkt. Diese Evaluationsart erreichte einen Bestwert von lediglich 2 false-negatives, nahm dafür allerdings 12 false-positives in Kauf.
Die letzte und sicherste Variante, in welcher jedes Subsystem die globale Aussage beeinflussen darf, erreichte tatsächlich den Wert von 0 false-negatives. Es wurden somit in 1500 Datenpunkten kein einziger Anfall übersehen, was wiederum ein Fundament für den Einsatz in der Praxis bieten würde. Andererseits lieferte diese Art der Evaluation ebenfalls 37 Fehlalarme, was das Ergebnis wiederum ein wenig abschwächt. Dennoch wurde hier der Zielwert erreicht und kein einziger Anfall übersehen.
Schlussendlich stellte sich das prioritätsbasierte Verfahren mit niedrigerem Prioritätsschwellwert als die beste Lösung heraus, da es ebenfalls keine Anfälle übersah und den Wert der Fehlalarme nochmals um 2 senken konnte. Hierbei wurde dem CNN das Recht eingeräumt, die globale Aussage zu beeinflussen. Somit kam das Verfahren auf einen Fehlerwert von 35 false-positives zu 0 false-negatives. Gegenüber dem sichersten Ansatz wurde hierbei lediglich dem MLP das Recht verwehrt, die globale Aussage zu beeinflussen. Es sei jedoch ebenfalls erwähnt, dass dieses Ergebnis möglicherweise nur aufgrund einer glücklichen Verteilung der Trainings- und Testdaten zustande kam. Auf anderen Testdaten hätte das MLP womöglich einen Anfall erkannt, welchen seine Kontrahenten missachtet hätten.

6.2 Kritik

Es gelang im Rahmen dieser Arbeit, den Zielwert von 0 missachteten Anfällen zu erreichen. Hierbei kam es jedoch zu 35 Fehlalarmen im besten Ansatz. In Rücksprache mit einem Experten könnte nun festgestellt werden, ob dieser in der Lage gewesen wäre, besagte Anfälle zu erkennen und dabei womöglich weniger Fehlalarme einzuräumen. Ungeachtet dessen ist die Vision einer kompromisslos zuverlässigen KI für das Erkennen von Anfällen während einer Elektroenzephalographie nicht wahr geworden. Es kann nicht ausgeschlossen werden, dass die Trainings- und Testdaten glücklich verteilt worden sind. Für eine Evaluation jeder möglichen Verteilung dieser Daten sind die Verfahren zu komplex und kosten somit zu viel Trainingszeit und Performance. In der Praxis können die Ergebnisse ebenfalls stark von den im Rahmen dieser Arbeit durchgeführten

Tests abweichen. Als Beispiel sei hier noch einmal das Enzephalogramm eines Kindes genannt, welches aufgrund der Aktivität von Kindern womöglich durch unkritische Artefakte belastet wäre, welche wiederum vom eingesetzten Verfahren als Potenzial fehlinterpretiert werden könnten. Aufgrund der Datenschutz-Grundverordnung (DGSVO) gestaltet es sich zudem schwierig, Patientendaten für ausreichende Tests zu verwenden. Demnach müssten die Patienten darüber in Kenntnis gesetzt werden und die Nutzung und Verarbeitung ihrer Gesundheitsdaten zu diesem Zweck ausdrücklich einwilligen. (Art. 9 Abs. 2a DSGVO)
Hinsichtlich der Vorverarbeitung beim CNN war zudem noch mehr Zeitaufwand notwendig. Mit dem Zugang zu mehr Ressourcen wäre es einerseits möglich gewesen, Bilddaten in größerer Auflösung zu verwenden und andererseits wäre das Ausprobieren unterschiedlicher Modelle an kürzere Trainingsphasen gebunden. Allerdings lässt sich auch hier abschätzen, dass die vorhandenen 11500 Daten für ein Erreichen von 100 prozentiger Genauigkeit dennoch nicht ausgereicht hätten. Wie bereits im Vorfeld erwähnt, würde ein längeres Training auf denselben Daten wiederum ein Overfitting begünstigen und die Genauigkeit des Lernverfahrens mindern.

7 Fazit

7.1 Schlusswort

Nachdem die Evaluation dennoch einen Erfolg verzeichnen konnte, ist die Möglichkeit zu Verbesserungen längst nicht ausgeschlossen. Mit dem richtigen Ansatz lassen sich neue Ideen verwirklichen, die wiederum neue Ergebnisse liefern werden.
Es gibt im Bereich des maschinellen Lernens unzählige Möglichkeiten, Architekturen zu entwerfen und zu verbessern. Für Letzteres wurde im Rahmen dieser Arbeit bereits die Keras Tuning Bibliothek vorgestellt, welche tatsächlich eine Verbesserung erzielen konnte. Da im Übrigen für jeden Anwendungsfall unterschiedliche Herangehensweisen zum Erfolg führen können, wie die vorherigen Kapitel bereits gezeigt haben, besteht auch immer die Möglichkeit, die ideale Herangehensweise zu übersehen. Es ist ebenfalls nicht ausgeschlossen, dass die verwendeten Modelle nach Abschluss dieser Arbeit weiter perfektioniert werden und schlussendlich ihre bisherigen Ergebnisse übertreffen.
Die im Rahmen dieser Arbeit erreichten Werte sind dennoch vorzeigbar, verglichen mit Architekturen, welche im World Wide Web gefunden werden können. Die Evaluation konnte schließlich einen nicht unbeachtlichen Erfolg einräumen. Auch sei erwähnt, dass mit der Zeit voraussichtlich auch viele weitere Einrichtungen Daten ansammeln und zur Verfügung stellen werden, sodass Algorithmen stetig auf neue Vorkommen trainiert werden können und dann im Einsatz mehr Referenzen zur Verfügung haben, wenn es um die Echtzeit-Diagnose eines Patienten geht.
Abschließend sei gesagt, dass sich die künstliche Intelligenz als solche zwar schon längere Zeit in Entwicklung befindet, jedoch eröffnen sich kontinuierlich mehr Möglichkeiten und neue Architekturen kommen hinzu, während bereits bestehende dank zahlreicher Bibliotheken immer leichter zu entwerfen sind. Es ist somit davon auszugehen, dass die Ergebnisse ebenfalls immer besser ausfallen werden, um letztlich dem vertrauenswürdigen Einsatz in lebensentscheidenden Bereichen der Praxis zu approximieren.

7.2 Ausblick

Eine weitere, mögliche Architektur, welche während der Recherche immer wieder auftauchte, bildet ein hybrider Ansatz aus RNN und CNN. Aufgesetzt auf die Architektur eines RNN, welches in vorherigen Kapiteln bereits näher betrachtet wurde, verfügt diese Architektur ebenfalls über Convolutional Layer und Pooling Layer, wie sie aus dem CNN bekannt sind. Durch die Vereinigung dieser beiden Architekturen sollen vor allem die jeweiligen Vorteile kombiniert werden. Ein Beispiel hierfür bieten die QRNNs. Diese sollen laut den Entwicklern unter anderem die Vorzüge eines RNN bieten, da die Ausgabe bspw. von der Reihenfolge aller Elemente der Sequenz beeinflusst wird. Laut den Entwicklern übertreffen die kombinierten Ansätze sogar bewährte LSTM Ansätze in Bereichen der Stimmungsklassifizierung, Sprachmodellierung und Übersetzung auf Zeichenebene. (Bradbury u. a. 2016)
Sollte es zudem möglich sein, eine Ansammlung an Patientendaten für das Trainieren der Verfahren zur Verfügung zu stellen, könnten diese auf bisher ungesehene Vorkommen in den Daten trainiert werden und sich somit dem vertrauenswürdigen Einsatz in der Praxis immer weiter annähern. Wie eingangs bereits erwähnt, blieben im Rahmen dieser Arbeit unzählige Verfahren unbeachtet, deren Eignung für die verbesserte Vorhersage auf EEG-Daten ebenfalls durchaus denkbar wäre. Da auch mit der heranschreitenden Technik notwendige Performance-Ressourcen zunehmend günstiger werden, kann auch die Zeit allein bereits die notwendigen Möglichkeiten schaffen, um Trainingsprozesse ausführlicher zu gestalten. Diese würden dann womöglich bessere Modelle zur Folge haben, welche wiederum bessere Ergebnisse erzielen würden.

Literaturverzeichnis

Aggarwal, Charu C. (2018). *Neural Networks and Deep Learning: A Textbook*. 1. Aufl. Springer International Publishing AG. ISBN: 978-3-319-94462-3.

Alazab, Mamoun und MingJian Tang (2019). *Deep Learning Applications for Cyber Security*. Springer Nature Switzerland AG. ISBN: 978-3-030-13056-5.

Alpar, Paul u. a. (2019). *Anwendungsorientierte Wirtschaftsinformatik: Strategische Planung, Entwicklung und Nutzung von Informationssystemen*. 9. Aufl. Springer Vieweg. ISBN: 978-3-658-25580-0.

Arbel, Nir (2010). *How LSTM networks solve the problem of vanishing gradients*. URL: https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577 (besucht am 27. 01. 2020).

Becker, Roland (2019). URL: https://jaai.de/convolutional-neural-networks-cnn-aufbau-funktion-und-anwendungsgebiete-1691/ (besucht am 08. 01. 2020).

Beierle, Christoph und Gabriele Kern-Isberner (2019). *Methoden wissensbasierter Systeme*. 6. Aufl. Springer Vieweg. ISBN: 978-3-658-27083-4.

Block, Frank (2019). *Praxisbuch neurologische Pharmakotherapie*. 3. Aufl. Springer. ISBN: 978-3-662-55837-9.

BMBF, Bundesministerium für Bildung und Forschung (2019). *Was KI für die Medizin bedeutet*. URL: https://www.bmbf.de/de/was-ki-fuer-die-medizin-bedeutet-9177.html (besucht am 10. 02. 2020).

Bow, Sing-Tze (2002). *Pattern Recognition and Image Preprocessing*. 2. Aufl. CRC Press. ISBN: 0-8247-0659-5.

Bradbury, James u. a. (2016). „Quasi-Recurrent Neural Networks". In: *CoRR* abs/1611.01576. arXiv: 1611.01576. URL: http://arxiv.org/abs/1611.01576.

Chollet, François (2018). *Deep Learning mit Python und Keras*. mitp Verlags GmbH & Co. KG. ISBN: 978-3-95845-791-1.

Chollet, François u. a. (2015). *Keras*. URL: https://keras.io (besucht am 01. 03. 2020).

D'Errico, Francesca u. a. (2015). *Conflict and Multimodal Communication: Social Research and Machine Intelligence*. Springer International Publishing Switzerland. ISBN: 978-3-319-14080-3.

Dong, Guozhu und Huan Liu (2018). *Feature Engineering for Machine Learning and Data Analytics*. CRC Press, Taylor & Francis Group. ISBN: 978-1-1387-4438-7.

Gamestar: Wieselsberger, Georg (2018). *IBM Watson als Doktor - Super-KI gab gefährliche Behandlungsvorschläge*. URL: https://www.gamestar.de/artikel/ibm-watson-als-doktor-super-ki-gab-gefaehrliche-behandlungsvorschlaege,3332869.html (besucht am 10. 02. 2020).

Gramfort, Alexandre u. a. (2013). „MEG and EEG data analysis with MNE-Python". In: *Frontiers in Neuroscience* 7, S. 267. ISSN: 1662-453X. DOI: 10.3389/fnins.2013.00267. URL: https://www.frontiersin.org/article/10.3389/fnins.2013.00267.

Hansen, Zschocke (2012). *Klinische Elektroenzephalographie*. 3. Aufl. Springer Medizin. ISBN: 978-3-642-19942-4.

Helmis, Steven und Robert Hollmann (2009). *Webbasierte Datenintegration*. 1. Aufl. Vieweg+Teubner.

Hochreiter, Sepp und Jürgen Schmidhuber (1997). „Long Short-Term Memory". In: *Neural Computation* 9.8, S. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://doi.org/10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

IBM: Weißmann, Alexandra und Eva Deutsch (2016). *Kognitives Assistenzsystem unterstützt Ärzte und die Transformation des Gesundheitswesens*. URL: https://www.ibm.com/de-de/blogs/think/2016/06/10/ibm-watson-gesundheitswesen/ (besucht am 10. 02. 2020).

IQWiG, Institut für Qualität und Wirtschaftlichkeit im Gesundheitswesen (2019). *Was passiert bei einer Elektroenzephalographie*. URL: https://www.gesundheitsinformation.de/was-passiert-bei-einer-elektroenzephalografie-eeg.3195.de.html (besucht am 27. 10. 2019).

Janczak, Andrzej (2005). *Identification of Nonlinear Systems Using Neural Networks and Polynomial Models*. Springer Verlag Berlin Heidelberg. ISBN: 3-540-23185-4.

Jörg, Johannes (2018). *Digitalisierung in der Medizin*. Springer. ISBN: 978-3-662-57758-5.

Kaggle (2018). *Epileptic Seizure Recognition*. URL: https://www.kaggle.com/harunshimanto/epileptic-seizure-recognition (besucht am 10. 02. 2020).

Keras-Team (2020). *Keras Tuner*. URL: https://keras-team.github.io/keras-tuner/examples/helloworld/ (besucht am 27. 02. 2020).

Krausz, Barbara (2018). *Methode zur Reifegradsteigerung mittels Fehlerkategorisierung von Diagnoseinformationen in der Fahrzeugentwicklung*. Springer Vieweg.

Kroll, Andreas (2013). *Computational Intelligence: Eine Einführung in Probleme, Methoden und technische Anwendungen*. Oldenbourg Verlag München.

Kursawe, Hubertus (2018). *Übungsbuch Klinisches EEG*. Springer. ISBN: 978-3-662-56755-5.

Kussul, Ernst, Tatiana Baidyk und Donald C. Wunsch (2010). *Neural Networks and Micromechanics*. Springer Verlag Berlin Heidelberg. ISBN: 978-3-642-02534-1.

Lagercrantz, Hugo (2019). *Die Geburt des Bewusstseins*. Springer. ISBN: 978-3-662-58222-0.

Lange, Carsten (2004). *Neuronale Netze in der wirtschaftswissenschaftlichen Prognose und Modellgenerierung*. Springer Verlag Berlin Heidelberg. ISBN: 978-3-7908-0059-3.

Li, Lisha u. a. (2018). „Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization“. In: *Journal of Machine Learning Research*.

Matignon, Randall (2005). *Neural Network Modeling Using Sas Enterprise Minor*. AuthorHouse. ISBN: 978-1-4184-2341-6.

Möser, Michael (2018). *Digitale Signalverarbeitung in der Messtechnik*. Springer Vieweg. ISBN: 978-3-662-56612-1.

NVIDIA Corporation (2020). URL: https://developer.nvidia.com/cuda-zone (besucht am 19. 02. 2020).

Olah, Christopher (2015). *Understanding LSTM Networks*. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (besucht am 01. 03. 2020).

Pedregosa, F. u. a. (2011). „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12, S. 2825–2830.

Reiche, Dagmar (2014). *Wann ein EEG zum Einsatz kommt*. URL: https://www.gesundheit.de/medizin/untersuchungen/nerven-und-gehirn/eeg (besucht am 19. 02. 2020).

Ripper, Klaus (2000). *Neuronale Netze im Portfolio-Management*. Deutscher Universitäts-Verlag. ISBN: 978-3-8244-7011-2.

Rosenblatt, Frank (1958). *The perceptron: a theory of statistical separability in cognitive systems*. Cornell Aeronautical Laboratory.

Sazgar, Mona und Michael G. Young (2019). *Absolute Epilepsy and EEG Rotation Review*. Springer Nature Switzerland AG.

scikit-learn developers (2019). *sklearn.cluster.DBSCAN*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html (besucht am 10. 02. 2020).

Sherstinsky, Alex (2018). „Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network“. In: *CoRR* abs/1808.03314. arXiv: 1808.03314. URL: http://arxiv.org/abs/1808.03314.

StatSoft Europe (2020). *Generalisierung mit neuronalen Netzen*. URL: https://www.statsoft.de/glossary/G/GeneralizationinNeuralNetworks.htm (besucht am 27. 02. 2020).

Sturm, Dietrich, Anne-Sophie Biesalski und Oliver Höffken (2019). *Neurologische Pathophysiologie*. Springer. ISBN: 978-3-662-56783-8.

van der Walt, S., S. C. Colbert und G. Varoquaux (März 2011). „The NumPy Array: A Structure for Efficient Numerical Computation“. In: *Computing in Science Engineering* 13.2, S. 22–30. ISSN: 1558-366X. DOI: 10.1109/MCSE.2011.37.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Klassifikation auf Basis von EEG-Daten, automatisiert durch maschinelles Lernen

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original