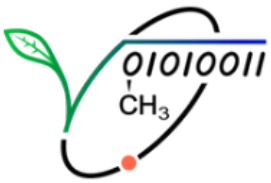

Spezialschulteil für Mathematik / Naturwissenschaften / Informatik

am Albert-Schweitzer-Gymnasium in Erfurt

Klassenstufe: *12*

Schuljahr: *2022/23*

Pneumonia Detection mithilfe von Convolutional Neural Networks

Betreuer:
Herr Süpke
Herr Golchert

Verfasser:
Dustin Marggraff
Nico Lentsch
Marvin Heyne

20.12.2023 / Erfurt

Inhaltsverzeichnis

1 Einleitung

Das Ausmaß, welches Lungenentzündungen annehmen können, konnte man anhand der Covid-19 Pandemie sehen. Lungenentzündungen fordern jährlich ca. 2,5 Millionen Tode und alleine in Deutschland sind jährlich über 650.000 Menschen von ihnen betroffen. Die aus der Überbevölkerung folgende Verschlechterung von Hygienestandards in vorallem auch Entwicklungsländern, aber auch Industrieländern wie Deutschland erleichtert Lungenentzündungen die Ausbreitung. Diese Seminarfacharbeit soll dieser Entwicklung entgegenwirken, indem ein Neuronales Netz Ärzten hilft, Lungenentzündung auf Röntgenbildern zu erkennen, welche oftmals schwer mit bloßem Auge erkennbar sind. So könnte man Lungenentzündungen schneller und besser erkennen, früher eine notwendige Behandlung einleiten und so die Ausmaße die eine Lungenentzündung annehmen kann verringern.
Aufgrund der Komplexität sowie den bereits vorhanden guten Datensätzen beschränkt sich diese Arbeit auf Pneumonie, welche die 3. häufigste Todesursache weltweit ist. Nach einem kurzen Überblick zu Lungenerkrankungen und Convolutional Neural Networks soll ein solches Netz entwickelt werden, welches diese Lungenerkrankungen auf Röntgenbildern erkennt und klassifiziert. Dies soll dem medizinischem Fachpersonal viel Unterstützung in ihrem Alltag bieten. Da nicht genügend Datensätze vorhanden sind, um weitere Faktoren wie das Alter, den BMI oder das Geschlecht als Faktor mit zu beachten, erstellt man anschließend noch eigene Datensätze durch eine Normalverteilung, indem man nach der üblichen Häufigkeit des Alters von Pneumonie-Erkrankten aufteilen und anschließend diese beiden unterschiedlichen Eingaben in zwei verschiedene Neuronale Netze eingeben und anschließend eine gemeinsame Ausgabe erhalten. Diese Funktion soll auch für die Zukunft ermöglichen, dass weitere Faktoren von der künstlichen Intelligenz bei der Diagnose berücksichtigt werden können. Dabei soll herausgefunden werden, ob diese Erweiterung mit den individuellen Faktoren eine höhere Genauigkeit liefert und dadurch auch besser auf jeden Patienten angepasst ist. Abschließend soll sich die Arbeit auch noch einmal der ethischen Betrachtung dieses Themas widmen, welche meist vernachlässigt wird, um einen Aufschluss darüber zu geben, ob es vertretbar ist diese künstlichen Intelligenzen anzuwenden und ihre Hilfe in Anspruch zu nehmen.
Python ist die beste Programmiersprache um CNN's zu entwickeln, da sie ausreichend Module besitzt und vor allem mit Numpy und matplotlib gut für Berechnungen und das plotten ist, daher beschäftigt sich diese Arbeit ausschließlich mit Python. Das Hauptprogramm mit der reinen Pneumonie Erkennung entwickeln wir von Grund auf ohne helfende Module zu benutzen, da dies viel mehr Anpassungsmöglichkeiten bietet und wir unser Netz genauer aufbauen können. Für die Erweiterung jedoch nutzen wir Tensorflow als Modul, da es bereits die verschiedensten Verbindungsmöglichkeiten bietet um so die beiden verschiedenen Eingabetypen zusammenzuführen und außerdem für diesen Zweck trotzdem ziemlich genau arbeitet. Außerdem erfolgt das Training des Netzes sehr schnell und ermöglicht so einen reibungslosen Ablauf bei der Entwicklung und dem Debuggen bei Fehlern, weil man schnell und problemlos einzelne Layer und Strukturen verändern kann.
Einen besonderen Dank gelten dabei Herr Süpke dem Fachbetreuer dieser Arbeit der bei der Erweiterungsfindung und der Entwicklung dieser tatkräftig Unterstützung geliefert hat und außerdem dem Außenbetreuer Herr Golchert, welcher mithilfe von Videokonferenzen das Thema immer mehr erläutern konnte und außerdem bei Problemen zu Rate stand. Des Weiteren hat dieser auch bei der Implementierung des Hauptprogramms sowie der Erweiterung zur Hilfe gestanden. Schließlich gilt auch der Seminarfachbetreuerin Frau Dr. Moor ein gebührenden Dank, welche bei der Gestaltung der äußerlichen Form unterstützt und konstruktive Kritik bei der Erstellung geliefert hat.

2 Lungenentzündungen

Lungenentzündungen, auch Pneumonie genannt, beschreiben eine Entzündung der unteren Atemwege der Lunge. Das heißt, es kann sowohl die Luftröhre als auch die Bronchien, Alveolen(Lungenbläschen) oder das umliegende Gewebe entzündet sein. Meistens sind Auslöser für eine Pneumonie Bakterien, Viren oder Pilze. In seltenen Fällen kann aber auch das Inhalieren giftiger Stoffe wie Abgasen oder anderer Chemikalien Auslöser sein. Es kommt dabei fast immer zu einer Entzündungsreaktion des Körpers, welche mit einer teils starken Einschränkung der Lungenfunktion einhergeht. Es kann in primäre und sekundäre Pneumonie unterschieden werden. Bei einer primären Pneumonie gibt es bei der Erkrankung keine Risikofaktoren, die den Verlauf beeinflussen, während es bei der sekundären Pneumonie prädisponierte Einflussfaktoren gibt. Risikogruppen, welche ein erhöhtes Risiko für eine Pneumonie haben, sind alte Menschen und junge Kinder, während auch Menschen mit geschwächten Immunsystemen gefährdet sind. Dies kann entweder erblich bedingt sein oder durch Krankheitserreger wie HIV ausgelöst werden. Weitere Risikogruppen sind Tabakkonsumenten, Menschen mit Lungenerkrankungen, wie beispielsweise COPD oder Asthma, Allergiker und mehr. Es ist also ersichtlich, dass es viele Risikogruppen gibt und die Ansteckung mit den Erregern schlimme Ausmaße annehmen kann.

2.1 Symptome und Ursachen

Bei den Symptomen kann man ebenfalls in zwei unterschiedliche Arten der Pneumonie unterscheiden. Die typische Lobärpneumonie zeichnet sich durch einen plötzlichen bzw. akuten Verlauf aus. Symptome hier sind starker Husten, Atemnot, eitriger Auswurf, Fieber, Schüttelfrost oder Pleuraerguss aus dem Pleuraspalt. Weniger starke, dafür aber länger anhaltende Symptome weist die atypische Pneumonie auf. Diese tritt sehr schleichend auf und verursacht schwächeren Husten, Atemnot, klaren Auswurf, Fieber und Schmerzen in Kopf und Gliedern. Die Ursache für die Symptome ist, wie vorhin erwähnt, die Immunantwort des Körpers auf Krankheitserreger wie Bakterien (Meningokokken), Viren (Grippe, Covid 19) oder Pilze (Aspergillus). Beispielsweise gibt es in unserem Immunsystem sogenannte Mastzellen, welcher Botenstoffe abgeben wenn sie in Kontakt mit Fremdkörpern bzw. den Erregern in Kontakt kommen. Diese Botenstoffe, wie zum Beispiel Zytokine oder Histamin sorgen für eine erhöhte Durchlässigkeit und Breite der Blutgefäße. So gelangt Flüssigkeit in den Entzündungsherd und mehr Blut wird gepumpt. Die Stelle wird warm, rot und geschwollen. Ein anderes Symptom, das Fieber, kommt zustande, wenn der Körper seine Temperatur erhöht, um den Krankheitserreger zu abzutöten. Nach einer bestimmten Zeit werden Antikörper gebildet, so kann der Erreger effizient bekämpft werden und man wird wieder gesund.[1]

2.2 Diagnose und Behandlung

Um eine Lungenentzündung zu erkennen, wird meistens erst der Brustkorb abgehört. Oftmals kann man bei einer kranken Person ein Knistern hören, da die Lungenbläschen verklebt sind. Als zweite Instanz greift man meistens auf Röntgenbilder zurück. Lungenentzündungen sind auf Röntgenbildern als charakterristisch erhellte Bereiche sichtbar[2]. Weitere Möglichkeiten der Diagnostik wären Ultraschall oder Computertomografie. Erst einmal sollte zur Behandlung gesagt sein, dass wenn man Hygienemaßnahmen einhält, wie Abstand halten oder Hände waschen, man das Ansteckungsrisiko seines gesamten Umfelds senken kann und gar nicht krank wird. Der Körper probiert selber, mit der Immunantwort den Erreger zu bekämpfen. Man kann ihm aber helfen, indem man mit Antibiotika (gegen Bakterien und Viren), Antimykotika (gegen Pilze) oder Virostatika (gegen Viren) den Erreger bekämpft. Weiterhin kann

[1] Jürgen Markl, Markl Biologie, S. 236, [3]
[2] Luise Heise, Röntgen-Thorax: Röntgenbild der Lunge, [15]

man auch die Symptome bekämpfen. So sollte man viel Schlaf bekommen und viel Flüssigkeit zu sich nehmen. Außerdem kann man Antifieber- oder Antischleimmittel zu sich nehmen. [3]

2.3 Künstliche Intelligenz in der Medizin

Die künstliche Revolution in der Medizin ist ein wichtiger Schritt, um neue Behandlungsmöglichkeiten zu schaffen, aber auch um die Entdeckung der Krankheiten beschleunigen zu können. Künstliche Intelligenzen unterstützen bereits Ärzte bei der Analyse von Röntgen- und Ultraschallbildern, um beispielsweise Frakturen besser erkennen zu können. Dies erleichtert vor allem den Fachkräften den Alltag, bietet aber auch eine bessere Genauigkeit bei der Untersuchung der Patienten. Kleine Brüche können durch das menschliche Auge meist nur schwierig entdeckt werden. Eine künstliche Intelligenz hingegen ist darauf trainiert, solche Brüche zu entdecken und bereits die kleinsten Unterschiede im Knochenbild des Patienten zu erkennen. Sie können also gut Röntgen-, MRT- oder Ultraschallbilder analysieren und Krankheiten erkennen, lokalisieren und klassifizieren. Das kann folgend die Diagnose erleichtern und präzisieren, was eine geringere Fehlerquote der Ärzte hervorruft.
Die Digitalisierung findet aber nicht nur bei der Erkennung der Krankheiten ihren Platz: von Apps für die einfache Nachkontrolle bis hin zu personalisierten Krebstherapien kann die KI auch in der Behandlung ihre Anwendung finden. KI-Technologien ermöglichen die Entwicklung von individuellen Behandlungsplänen für Patienten, basierend auf ihren genetischen, biologischen und klinischen Daten. Durch die Analyse all dieser Daten können künstliche Intelligenzen die Wahrscheinlichkeit vorhersagen, dass eine bestimmte Therapie auf einen Patienten Wirkung hat und Ärzten helfen, die am besten geeignetste Behandlung auszuwählen. Dies verringert unerwünschte Nebenwirkungen und erhöht auch die Wirksamkeit der Therapien bei den jeweiligen Patienten. Beispielsweise findet die KI bei der personalisierten Krebstherapie Anwendung. Sie kann basierend auf der Analyse des jeweiligen Tumors die beste Zusammenstellung der Medikamente für jeden Patienten finden.
Auch Dr. Daniel Ziemek, ein Computerwissenschaftler und KI-Experte bei Pfizer, sieht die künstliche Intelligenz als wichtigen Bestandteil der Medizin: „Wo wir noch vor zehn Jahren aus einem Tropfen Blut etwa 20000 Datenpunkte erhalten haben, bekommen wir jetzt 20000 Datenpunkte für jede einzelne der Tausenden von Zellen in diesem einen Blutstropfen.“[4] Die Sammlung von Daten zur Forschung wird durch die Digitalisierung einfacher und bietet anderen künstlichen Intelligenzen bessere Trainingsmöglichkeiten, da mehr vorhandene Daten immer genauere Ergebnisse bedeuten.
Auch bei der Allgemeinheit der Bevölkerung stößt man größtenteils auf Befürwortung bei der Frage, ob man künstliche Intelligenz in der Medizin verwenden sollte. 67 Prozent der deutschen Bevölkerung befürworten den Einsatz von KI's in der Medizin, beispielsweise im Bereich der Diagnostik. Das ergab eine repräsentative Umfrage im Auftrag des Meinungsforschungsinstituts Civey des Wirtschaftsjahres 2019.[5] Für die Ärzte sind diese Ergebnisse praktisch, da die künstliche Intelligenz ihren Alltag erleichtert aber auch die generelle Sicherheit für eine Diagnose erhöhen. Die KI sollte also assistierend zu einem Arzt genommen werden, da eine künstliche Intelligenz alleine niemals so viele Merkmale und Eigenschaften des Patienten haben wird wie der zuständige Mediziner. Auch Dr. Narges Ahmidi, Leiterin der Abteilung „Reasoned AI Decisions“ am Fraunhofer IKS, empfindet die KI als wichtige Unterstützung für die Ärzte: „Studien haben gezeigt, dass die kombinierte Leistung von klinischen Experten und KI zusammen der Leistung von Experten allein überlegen ist.“[6]

[3] Nastasia Heilemann, Lungenentzündung, [14]
[4] Quelle: Dr. Daniel Ziemek, KI in der Medizin: Künstliche Intelligenz für die Gesundheit [30]
[5] Quelle: Annalena Rüsche, Künstliche Intelligenz in der Medizin [22]
[6] Quelle: Dr. Narges Ahmidi, KI im Gesundheitswesen, Seite 23 bis 27 [1]

3 Convolutional Neural Networks

Künstliche neuronal Netze (KNN) sind eine Art des maschinellen Lernverfahrens, das vom biologischen Vorbild des Nervensystems inspiriert wurde. Ähnlich unseres Gehirnes besitzen KNNs Neuronen, auch Knoten genannt, welche auf Schichten (engl. Layer) angeordnet sind. Die einzelnen Neuronen einer Schicht sind mit denen anderer Schichten verbunden. Eine Eingabe in ein solches Neuronales Netz wird durch die Aktivierungsfunktionen und gewichteten Verbindungen verändert. Werden viele Layer hintereinander geschaltet, so nennt man dies ein Deep Neural Network (DNN). Mithilfe solcher Strukturen lassen sich komplexe Probleme, die zuvor für den Computer nahezu unüberwindbar waren, lösen. Dabei gibt es verschiedenen Formen von Neuronalen Netzen. In dieser Arbeit werden Convolutional Neural Networks (CNN/ConvNets) oder auf deutsch auch gefaltete Neuronale Netzwerke genannt, thematisiert und verwendet.

3.1 Allgemeine Information

Convolutional Neuronal Networks bieten durch ihren speziellen Aufbau besondere Qualifikationen für die Verarbeitung von strukturierten Datenarrays, wie Bild- und Audiodateien. Der Sehrinde des menschlichen Gehirns nachempfunden, ist dies für die Klassifizierung der Lungen-Röntgenbilder vorteilhaft.
Bevor solche Netze eingeführt wurden, mussten Merkmalsextraktionen oftmals händisch oder über andere zeitaufwendige Methoden vorgenommen werden. Ihr Ziel ist es bestimmte Geometrische Figuren oder andere für das Problem relevante Muster des Bildes, wie beispielsweise Kanten oder Kreise, herausgefiltert. Jene unverzichtbaren Merkmalsextraktionen werden in CNN's durch Anwendung von Prinzipien der linearen Algebra, wie der Matrixmultiplikation, umgesetzt. [7]Das Modell zeichnet sich durch eine hohe Verarbeitungsgeschwindigkeit und einen geringen Speicherbedarf für große Eingabedaten aus. Allerdings ist der Rechenaufwand in der Lernphase oft hoch.
Darüber hinaus arbeiten CNN's direkt auf der Rohdatei und benötigen somit keine Vorbearbeitung. Ihre Hauptaufgabe besteht in dem Erkennen von Mustern in der Eingabedatei. Solche Muster sind zum Beispiel Linien, Kurven, Kanten oder auch komplexere Strukturen, wie z.B. Augen oder Gesichter, und können ortsungebunden ermittelt werden. Um diese zu erkennen, besitzen CNN eine spezielle Schicht die Convolution Layer, welche als Filter fungieren. Diese Layer sind meist übereinander gefaltet, woraus sich der Name zusammensetzt, und ermöglichen die Extraktion komplexerer Formen. So ist es dem Netz bereits mit 3-4 Faltungsschichten möglich, handgeschriebene Ziffern zu erkennen. Menschliche Gesichter hingegen benötigen bereits 25 Schichten. [8]

3.2 Aufbau eines CNN

Gefaltete neuronale Netze sind anders aufgebaut als geradlinige/Multi-Layer neuronale Netze. Dadurch ist es möglich, die Dateneingabe als Matrix statt als Vektor vorzunehmen. Für das Problem dieser Seminararbeit, der Erkennung einer Pneumonie, kann so ein Röntgenbild der Lunge in Form einer Matrix einzelner Pixel übergeben werden. Bei Farbbildern ist es somit möglich, auch eine dreidimensionale Matrix zu nutzen, um die Farbchannel (RGB) korrekt darstellen zu können. Im Unterschied dazu müsste das Bild bei einem vollvernetzten (Dense) Netzwerk erst eine Flatting-Schicht (Glättigungsschicht) durchlaufen, also in einen eindimensionale Matrix/Vector durch Aneinanderreihung der Pixel übersetzt werden. In diesem Fall wäre es auch nicht möglich, bestimmte Muster ortsabhängig zu betrachten. [9] Ein CNN ist meist aus drei unterschiedlichen Schichtarten aufgebaut. Diese einzelnen Arten können sich dabei beliebig oft wiederholen. Meist beginnt das Neuronale Netz mit mehreren gefalteten Schichten,

[7]Quelle: Max-Ludwig Stadler, Convolutional Neural Network (CNN) [26]
[8]Quelle: Thomas Wood, Convolutional Neural Network[28]
[9]Quelle: Julia Odenthal, Convolutional Neural Networks[21]

gefolgt von einem Pooling Layer. Dieses Konstrukt wird meist wiederholt. Am Ende folgen stets einige Fully-Connectet-Layer:

3.2.1 Convolutional Layer

Die Convolutional Layer, auch Faltungsschichten genannt, sind die Kernbausteine eines Convolutional Neural Networks. Sie besitzen sogenannte Filter, die bestimmte Merkmale und Strukturen in den Eingabedateien hervorheben. Eine Faltung ist dabei eine Anwendung eines solchen Filters auf eine Eingabe, z.B. ein Röntgenbild, die zu einer Aktivierung (Ausgabe) führt. Sie ist ähnlich den Dens-Layern ebenso eine lineare mathematische Multiplikation von Eingaben mit Gewichten. Diese Gewichtungen sind ebenfalls in einer Matrix angeordnet und werden Filter oder Kernel genannt, sie sind jedoch meist kleiner als die Eingabematrix. Dieser Filter wandert nun schrittweise von der oberen linken zur unteren rechten Ecke der Eingabematrix. In jedem Schritt wird das Skalarprodukt von dem Filter und der von diesem überlappten Teilmatrix der Eingabe gebildet. Dabei werden übereinanderstehende Zahlen vom Filter und der Eingabe miteinander multipliziert und über alle Produkte die Summe gebildet. Die resultierende Zahl ist ein Pixel der Ausgabematrix dieser Schicht. Die Schrittweite (engl. Stride) gibt dabei an, wie viele Felder der Filter in jedem Schritt nach rechts und, sobald der rechte Rand erreicht ist, nach unten wandert. Ein Beispiel für diese Operation ist in (abb: 1) dargestellt. Voraussetzung für die beschriebene Operation ist jedoch, dass der Filter mindestens gleich viele Dimensionen wie die Eingabe besitzt. Es werden zum Beispiel 3 Filter für ein Bild, das in den 3 Farbkanälen dargestellt ist, benötigt. Die Ausgabegröße ist somit von der Größe der Eingabematrix und des Filters sowie der Schrittweite abhängig. Um jedoch eine größere Ausgabematrix zu erhalten, kann die Eingabe gepolstert werden. Dabei wird ihr ein Rahmen (engl. padding) aus beispielsweise Nullen angehängt, wodurch sie größer wird. [10]

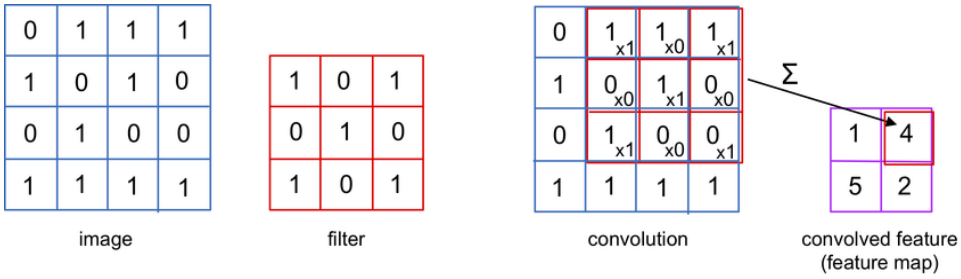

Abbildung 1: Ausführen der Convolutional Funktion[12]

Nach jeder Faltung kann eine Aktivierungsfunktion, wie zum Beispiel die ReLU-Transformation stattfinden, um die Nichtlinearität zu gewährleisten. Eine Ergänzung ist auch über Bias-Werte für jede Schicht eines ConvLayers möglich. Dieser wird auf das Skalarprodukt addiert, ähnlich wie bei Dense-Layern.
Folgen auf eine Faltungsschicht weitere, so wird das Netz hierarchisch. Diesem ist es möglich, in die vorigen rezeptiven Schichten zu sehen. Liegt beispielsweise ein Röntgenbild der Lunge vor, so kann das Netz den Brustkorb als Einheit von Einzelteilen extrahieren. So setzt sich der Brustkorb aus einzelnen Rippen zusammen, die wiederum eine Kombination aus senkrechten/waagerechten Strichen und Kanten sind. [11] Diese Conv-Layer lassen sich auch ähnlich einem vollvermaschten Neuronalen Netz darstellen[12]. Der Unterschied zu einem gewöhnlichen MLP ist, dass ein Neuron nicht mit allen Neuronen der vorigen Schicht verbunden ist und sich Neuronen die Gewichte des Kernels teilen. Dadurch entsteht

[10]Quelle: Gerhard Paaß, Dirk Hecker; Künstliche Inteligenz, Seite: 24 [2]
[11]Quelle: Mayank Mishra, Convolutional Neural Networks, Explained[18]
[12]Quelle: Eli Bendersky, Backpropagation through a fully-connected layer [7]

ein geringerer Rechenaufwand. [13]

3.2.2 Pooling Layer

Convolutional Layer sind sehr gut im Filtern von Merkmalen, jedoch ist die Datenmenge immer noch hoch. Dies hat einen hohen Rechenaufwand und Rechenzeit zur Folge. Deshalb folgen auf diese Conv-Layer meist sogenannte Pooling-Layer. Da die genaue Lokalisierung eines Objektes, wie einer Kante, nicht signifikant ist, sorgen Pooling-Schichten dafür, dass die Datenmenge reduziert wird. Dabei nutzen sie ebenfalls eine Art Filter in Form einer Matrix, welche jedoch keine Gewichte besitzen. Stattdessen werden bestimmte Operationen auf die Teilmatrizen angewendet, um auf einen Wert für die Ausgabematrix zu gelangen. Beim Max-Pooling wird von den 4 Feldern des Filters jeweils das stärkste bzw. die größte Zahl in die Ausgabematrix geschrieben. Umgekehrt ist die Vorgehensweise des Min-Pooling, bei welchem der kleinste Wert einen Pixel in der Ausgabe darstellt. Die dritte Variante ist das Average-Pooling, bei dem der Durchschnitt aller Werte errechnet und in die Ausgabematrix eingetragen wird.
Die Vorteile dieser Schichten sind zum einen die gesteigerte Effizienz, also geringere Laufzeit und Speicherbedarf. Auf der anderen Seite können auch tiefere Netzwerke entstehen. Zuletzt wirkt es ebenfalls dem Overfitten entgegen, welches ein gängiges Problem beim Lernvorgang Neuronaler Netze ist. [14]

3.2.3 Fully-connected Layer

Am Ende des Netzes folgt ein Fully-connectet-Layer (Dens-Layer), welcher oftmals ein Multy-Layer-Perceprton(MLP) ist. Bei diesem sind alle Neuronen einer Schicht mit jedem Neuronen der folgenden Schicht verbunden. Ein Neuron besitzt dabei spezifische Gewichte zu jeder Eingabe (jedem Neuron der vorhergehenden Schicht). Die Ausgabe wird abermals durch die Multiplikation der Eingabe mit den Gewichten und anschließende Aufsummierung der Produkte gebildet. Zuvor gefilterte und komprimierte Informationen werden hier ausgewertet und klassifiziert. Die Anzahl der Ausgabeneuronen entspricht in der Regel der Anzahl der Klassen, die entstehen können, also in diesem Fall z.B. Pneumonia/keine Pneumonia oder Pneumonia bakteriellen/viralen Ursprungs. Als Aktivierungsfunktion wird hier meist die Softmax-Funktion verwendet, um eine Ausgabe zwischen 0 und 1 zu generieren.

3.2.4 Flatting und Dropout-Layer

Eine kleinere Rolle spielen die Flatting- und Dropout-Layer und sind somit nicht für jedes CNN essentiell. Die Flatting-Layer bilden ein Brücke zwischen den Convolutional- und Dens-Layern. Sie ändern die Dimension und Größe (engl. shape) der Daten. Die zweidimensionale Ausgabe bzw. Feature Matrix der Convolutional oder Pooling Layer wird hier für die auf eine Dimension beschränkte Eingabe eines Dens-Layers umgeformt. Diese Transformation wird durch Aneinanderreihen der Zeilen einer Matrix zu einem Vektor erreicht.
Die Dropout-Layer schließt bestimmte Knoten mit einer bestimmten Wahrscheinlichkeit „p“aus, um das Overfitting zu verhindern. Daher wird sie vorrangig während des Trainings aktiv. Die Auswahl der zu löschenden Knoten wird dabei zufällig vorgenommen. [15]

3.3 Lernverfahren

Ein frisch initialisiertes Neuronales Netz ist wie der Mensch kurz nach seiner Geburt. Es besitzt alle Komponenten zum Nachdenken und Analysieren, jedoch fehlt ihm die Erfahrung. Hat man kein Me-

[13] Quelle: Jason Brownlee, How Do Convolutional Layers Work in Deep Learning Neural Networks?[8]
[14] Quelle: Savyakhosla, CNN | Introduction to Pooling Layer[23]
[15] Quelle: Harsh Yadav, Dropout in Neural Networks[29]

dizinstudium oder andere medizinische Ausbildungen in der Vergangenheit bestritten, so ist es dem Menschen ebenfalls nahezu unmöglich festzustellen, ob es sich auf einem bestimmten Röntgenbild um eine Pneumonie handelt. Im Gegensatz zum Menschen ist das Netz jedoch nicht auf eine Ausbildung angewiesen, in der diesem gelehrt wird, woran eine Lungenerkrankung erkennbar ist. Das Netz eignet sich diese Fähigkeiten über ein spezifisches Lernverfahren selbst an. Dazu wird bei CNNs oftmals das überwachte Lernen angewandt. Dafür werden dem Netz sowohl Testbilder als auch die zu erwartende Ausgabe, Label genannt, übergeben, z.B. ein Bild einer Lungenerkrankung und die Information, ob es sich um eine Lungenerkrankung handelt. Die Qualität der Lerndaten ist dabei ausschlaggebend für die Qualität des Netzes. Im Allgemeinen besteht der Lernprozess beim überwachten Lernen aus zwei Phasen. Die erste ist die Forwardpropagation, welche ein normales Durchlaufen des Netzes der Daten und anschließenden Klassifizierung dieser entspricht. Daraufhin folgt die Backpropagation. Hier wird der Fehler von der Ausgabe zu dem Label berechnet und die Parameter des Netzes anhand diesem angepasst. Dies geschieht in der sogenannten Backpropagation. Bei dieser wird in der letzten Schicht begonnen, die Parameter an den Fehler anzupassen. Den über diese Schicht ausgerechneten Fehler der vorhergehenden Schicht wird nun übergeben, woraufhin deren Parameter angepasst werden. Der Name beschreibt somit die Vorgehensweise von der letzten zur ersten Schicht, also rückwärts von der Forwardpropagtion, wobei jede Schicht ihre Parameter nach dem Fehlergradienten von der folgenden Schicht anpasst und selbst den Fehlergradienten der vorhergehenden Schicht berechnet.

3.3.1 Backwardpropagation der fully-connected-Layer

Da die Fully-Connected-Layer die letzten Schichten unseres Neuronalen Netzes bilden, erfolgt der Start für die Anpassung bei diesen. Folgende Formeln gehen dabei von einer Softmax Aktivierungsfunktion für die Layer/Schichten aus, da diese weit verbreitet ist. Als erstes benötigen wir die Ableitung nach der Ausgabe des Netzes. Diese wird wie folgt berechnet: $\frac{\partial L}{\partial out_s(i} = \begin{cases} 0 & \text{falls } i \neq c \\ -\frac{1}{p_i} & \text{falls } i = c \end{cases}$. Dabei ist p_i die Wahrscheinlichkeit von Ausgabeneuron i, c die zu erwartende Klasse bzw. das richtige Ergebnis und L der Cross-Entropy-Loss. Der Gradient für die Ausgabe $out_s(c) = Softmax(input)$ berechnet sich wie folgt: $\frac{\partial out_s(k)}{partialt_k} = \begin{cases} \frac{-e^{t_c} e^{t_k}}{s^2} & \text{falls } k \neq c \\ \frac{-e^{t_c}(S - e^{t_c})}{s^2} & \text{falls } k = c \end{cases}$. Damit lässt sich der Gradient für die Gewichte: $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial out} \cdot \frac{\partial out}{\partial t} \cdot \frac{\partial t}{\partial w}$, den Bias: $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial out} \cdot \frac{\partial out}{\partial t} \cdot \frac{\partial t}{\partial b}$ und die Eingabe: $\frac{\partial L}{\partial input} = \frac{\partial L}{\partial out} \cdot \frac{\partial out}{\partial t} \cdot \frac{\partial t}{\partial input}$ berechnen. Dieser Eingabegradient wird, wie bereits beschrieben, an die vorigen Schicht weitergegeben, welche anhand dessen ebenfalls ihre Parameter anpasst. Die neuen Gradienten werden von den bisherigen Parameterwerten abgezogen $w_{neu} = w_{alt} - \frac{L}{w}$, um so den Fehler zu minimieren. Damit würden die Parameter jeweils an den Gesamtfehler angepasst werden. Das Resultat daraus ist, dass sich das Netz ein Bild einprägen anstatt, zu lernen Bilder einer Art zu erkennen. Darüber hinaus würden folgende Schichten anhand des gleichen Fehler, also das ganze Netz doppelt angepasst werden. Um das zu verhindern, wird ein Lernparameter, welcher zwischen Null und Eins liegt, erstellt und mit der Anpassung der Parameter multipliziert (für weitere Informationen siehe Anhang Seite: 18f. „Herleitung der Backpropagation“). [16]

3.3.2 Backwardpropagation der Pooling-Layer

Vor den Fully-Connected-Layer ist meist eine Poolingschicht. Da diese eine feste Kernelgröße besitzen und sonst keine variablen Parameter, müssen keine Gradient für diese Schicht berechnet werden. Allerdings benötigen die Convolutional-Layer den Ausgabegradienten der Pooling Backpropagation. Deshalb muss der Fehlergradienten nach der Eingabe der Schicht $\frac{\partial L}{\partial X}$ berechnenet werden.
Um den Fehlergradienten für die vorhergehende Schicht zu berechnen muss die erhaltene Fehlermatrix

[16] Quelle: Eli Bendersky, Backpropagation through a fully-connected layer [7]

jedoch um den gleichen Wert erweitert werden, wie die Eingabe beim Pooling geschrumpft ist. Auch muss beachtet werden, welche Pixel Einfluss auf die Ausgabe genommen haben.
Wurde das Max- oder Min-Pooling ausgeführt, so hängt die Ausgabe nur von einem von 4 (2x2) Pixeln ab. Deshalb ist die Ableitung nach den anderen drei Pixel gleich Null. In der Ausgabematrix der Backpropagtion, welche die gleichen Maße wie Eingabe dieser Schicht besitzt, muss überall eine 0 eingetragen werden, außer bei den Signifikanten Pixeln. Dort wird der erhaltene Gradient von der nächsten Schicht eingetragen.
Beim Averagepooling ist es komplizierter, da hier die Ausgabe von allen Pixeln abhängt. Daher ist der Fehler, welchen wir von der nächsten Schicht erhalten, auf alle Pixel aufzuteilen. Somit wird der Fehler durch n(Anzahl der Pixel pro Kernel) geteilt, wodurch alle Pixel den gleichen Wert erhalten. [17]

3.3.3 Backwardpropagation der Convolutional Layer

Als letztes besitzt ein CNN noch die sogenannten Convolutional Layer. Da die Werte in dem Kernel zufällig gewählt sind, müssen diese ebenfalls angepasst werden, um die richtigen Strukturen herausfiltern zu können. Somit muss zum einen der Fehlergradient nach den Parametern des Netzes und zum anderen der Fehlergradient nach der Eingabe berechnet werden. Dabei ist die Herleitung der Gradienten ähnlich der von den Dens-Layern. Zusätzlich wird die Herleitung jedoch durch die kleinere Anzahl an Parametern und den mehrfach genutzten Gewichten vereinfacht. Wird so die Funktion dieser Schicht abgeleileitet (siehe Herleitung Anhang Seite:), so erhält man folgende Formeln für die Berechnung der Gradienten:

$$\frac{\partial L}{\partial F} = Convolution(InputX, Fehlergradient\frac{partialL}{\partial O}) \tag{1}$$

$$\frac{\partial L}{\partial X} = FullConvolution(180°gedrehterFilterF, Fehlergradient\frac{partialL}{\partial O}) \tag{2}$$

$$\tag{3}$$

Der errechnete Gradient $\frac{\partial L}{\partial F}$ weißt Ähnlichkeiten, wie der Gradient $\frac{\partial L}{\partial W}$ eines MLP. Unser Filter wird ebenfalls ähnlich angepasst: $F_{neu} = F_{alt} - \alpha \cdot \frac{\partial L}{\partial F}$.
Auch die optionalen Biaswerte, die für einen Filter gesetzt sein können, werden ähnlich angepasst: $\frac{\partial L}{\partial b} = \frac{1}{M} \cdot \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} \cdot \frac{\partial O_k}{\partial b}$. Da die partielle Ableitung $\frac{\partial O_k}{\partial b}$ immer 1 ist, erfolgt die Anpassung: $b_{neu} = b_{alt} - \alpha \cdot (\frac{1}{M} \cdot \sum_{k=1}^{M} \frac{\partial L}{\partial O_k})$ (Für genauere Information siehe Anhang Seite 22 „Herleitung der Gradienten von den ConvLayern“)
[18]

3.3.4 Methoden zum Optimieren des Netzes

Um das Netz anzupassen und den Fehler für jeden Parameter zu berechnen, gibt es verschiedenen Operationen. Beim Erstellen des Netzes sorgen Parameterinitializer dafür, Startwerte für die Parameter des Netzes, wie beispielhaft die Gewichte eines vollvermaschten Netzes(Dens-Net) zu finden, mit denen der Lernprozess optimiert werden kann. Richtig gewählte Initialisierungsmethoden können dazu führen, dass die Konvergenz des Gradientenabstieges beschleunigt wird(das Netz schneller lernt), darüber hinaus führen sie zu unterschiedlichen Ergebnissen und können z.B. zufällig optimale Werte erzeugen. Beispiele dafür wäre die Normalverteilung oder zufällige Parameter im Gegensatz zu Nullen für die Parameter, welche schwieriger anzupassen sind und Komplikationen bei manchen Rechenoperationen aufweisen.

[17] Quelle: Jefkine, Backpropagation In Convolutional Neural Networks[16]
[18] Quelle: Pavithra Solai, Convolutions and Backpropagations[25]

Die gewählte Aktivierungsfunktionen beeinflussen ebenfalls das Lernverfahren und eigenen sich unterschiedlich gut für verschiedenen Aufgaben. Sie reduzieren jedoch vorrangig den Wertebereich, sodass Zahlen nicht durch die Schichten immer größer werden und immer mehr Speicherplatz benötigen. Nach dem Vorbild menschlicher Synapsen(Neuronen) welche ebenfalls sogenannte Schwellwerte besitzen, finden sie in fast jeder Schicht Anwendung. Beispiele sind der Tanges-Hyperbolicus oder die Sigmoid-Funktion.[19] Nach der Initialisierung beginnt der Lernprozess und die Anpassung der Parameter. Dafür muss als erstes der Fehler der Netzausgaben berechnet werden. Dieser Fehler wird über eine Fehlerfunktion, wie dem „Cross-Entropie-Loss“ oder dem „Mean-Squered_Error“(MSE) berechnet. Diese bilden meist die Differenz der Ausgabe mit den Labeln der Eingabebilder und modifizieren diese, um z.B. große Fehler besonders hervorzuheben(z.B. MSE). [20] Nach dem Fehler aus den Fehlerfunktionen werden anschließend die Fehlergradienten, also die Veränderung der Parameter, über Optimizer berechnet. Diese passen Attribute des Netzes an, um den Fehler des Netzes zu verkleinern. Eines der bekanntesten Optimizer ist das Gradientenabstiegsverfahren, in dem über die Ableitung der Fehlerfunktion ein Gradient berechnet wird, welcher in die Richtung des geringsten Fehlers zeigt. Entlang dieses Gradienten werden nun die Änderungen der Parameter berechnet. [21] Um während der Lernphase nicht über den Optimalpunkt hinauszuschießen oder um diesen zu pendeln, wird die Lernrate/Lernparameter, welche die Stärke der Änderung der Parameter beeinflusst, verändert (Weitere Information siehe Anhang Seite 23„Lernparameter“). Dies geschieht durch Learningrate Decays. So kann beispielsweise mit steigender Lernzeit die Lernrate verkleinert werden, um sich dem Optimum besser anzunähern.[22] Anschließend genannt seien noch die Utility Funktionen, welche den Nutzen eines Netzes bewerten, also wie gut bzw. hilfreich die Prognosen des Netzes sind. Sie spielen für CNN‘s jedoch eine untergeordnete Rolle und sind daher nicht immer essentiell.

4 Entwerfen eines eigenem CNN

4.1 Implementierung unseres Net-Modules

Für diese Arbeit wurde ein eigenes Paket verfasst, welches Grundbausteine für Neuronale Netze, im Besonderen der CNN’s, nach den beschriebenen Grundsätzen, bereitstellt. Dieses Paket ermöglicht einen tieferen Einblick in die Funktionsweise der einzelnen Schichten sowie eine große Anzahl an Veränderungsmöglichkeiten um das Netz bestmöglich an die Problemstellung anzupassen. Das Paket wurde vollständig in Python implementiert und greift auf eine kleine Anzahl weiterer Module zurück, welche von der Python Standardbibliothek bereitgestellt werden. Für die Implementierung der einzelnen Berechnungen sowie mathematischen Formeln nutzten wir Numpy (ausgeschrieben Numerisches Python). Da Numpy zum größten Teil in der Programmiersprache C geschrieben wurde, ist es bezüglich der Laufzeit gegenüber anderen, in Python implementierten Modulen, deutlich überlegen. Der eigene Datentyp „ndArray“ermöglicht das Erstellen von Arrays (statischer Listen) statt LinkLists (dynamischen Listen Standard von Python) und stellt viele Features für die Indexierung bereit. Auch eine Vielzahl vordefinierter mathematischer Funktionen, wie die Einsteinsche Summenkonvention, welche für die Implementierung Neuronaler Netze essentiell sind, werden durch das Modul zur Verfügung gestellt. Weiter Module, wie Matplotlib, zur Erstellung von Graphen oder tqdm, werden für die graphische Ausgabe von Lernprozessen und Bildern genutzt. Das Einlesen der Datensätze (als csv-Datei) wird von dem Modul Pandas übernommen.[23]

[19] Dave Andre, Was ist eine Aktivierungsfunktion?[5]
[20] Saily Shah, Cost Function is No Rocket Science! [24]
[21] Sanket Doshi, Various Optimization Algorithms For Training Neural Network [11]
[22] Vaibhav Haswani, Learning Rate Decay and methods in Deep Learning [13]
[23] Bernd Klein, NumPy Tutorial [**?**]

Das erstellte Packet wurde nach den verschiedenen Teilaufgaben eines Neuronalen Netzes in einzelne Module aufgeteilt. Dem ersten Modul sind Teilaufgaben der Initialisierung des Netzes und Optimierung des Lernverfahren sowie der Rechenleistung und des Speicherbedarfes des Neuronalen Netzes zugeordnet. Unter dem Modul-Namen „Rectifier“ sind die Klassen Parameterinitializer implementiert. Als Parameterinitializer wurden unter anderem die Normalverteilung sowie die zufällige Parameterinitialisierung bereitgestellt.
Des weiteren sind Aktivierungsfunktionen, wie Sigmoid, Tangens Hyperbolikus und ReLU(Rectified Linear Unit) in dieses Modul eingebunden .
Für die Optimizer wurden die weit verbreiteten Arten „Adam“ und „GD“ (Gradientenabstieg) in dieses Modul integriert. Zuletzt sind in Rectifier auch Klassen für die Utilizer Funktion sowie dem Learningrate Decay beschrieben.[24]

Diese Funktionen und Klassen beinhalten wichtige Bestandteile, welche für das nächste Untermodul benötigt werden. In dem Modul „Layer“ sind die einzelnen Schichten eines Neuronalen Netzes definiert. Alle Schichtklassen haben dabei eine „forward“ Funktion für die Forwardpropagation, eine Funktion „backward“ für die Bakcwardpropagation, „update“ als Funktion für das Anpassen der Parameter (ausgenommen der Pooling, Flatten und Dropout-Klassen), sowie eine Funktion „get_dimension“, welche die Ausgabegröße der Schicht zurückgibt. So werden die vollvermaschten Schichten durch die Klasse „Dens“ repräsentiert. Die Parameter der Klasse sind zum einen die Struktur, wie Anzahl der Neuronen(neuron), Eingabegröße (input_shape) sowie die Initializer (weight_initializer) und Regulizer (weight_regulizer). Zum Anderen auch die Parameter wie Gewichte(weight) oder Biaswerte (bias) und die Ativierungsfunktion (activation). Die Dropout-Layer sind in der Klasse „Dropout“ definiert. Als Parameter dieser Klasse ist die Dropoutwahrscheinlichkeit p(Wahrscheinlichkeit mit der ein Neuron ausgeblendet wird) gesetzt.[25] Als eigene Klasse wurde auch das Padding implementiert (Padding2D). Bei diesem wird dem Bild ein Rahmen verliehen, welches in den Pooling oder Convolutional Layern benötigt wird, um die Größe der Ausgabematrix zu variieren. Zum einen kann das Padding als feste Zahl der Zeichen ober/unter links/rechts vom Bild, oder als „same“(kein Padding) oder „valid“(Padding, damit die Ausgabegröße gleich der Eingabegröße des Poolings/Convolution ist) angegeben werden. Das Pooling ist in der Klasse Pooling2D implementiert. Diese enthält zum einen die Parameter für die Filtergröße (kernel_size), Schrittweite/stride (step) und den Padding-Typ. Des weiteren wird der Typ des Poolings ('max', 'min', 'mean'(avarage)) festgelegt. Die Besonderheit ist, dass für die Poolingoperation ein neuer View (Ansicht) der Eingabeliste erstellt wird. Dafür ist die Funktion „getSubmatrix“zuständig, diese speichert die Bereiche, welche von dem Filter in jedem Schritt abgedeckt werden als eigene Matrix ab. Dabei stimmen die einzelnen Felder immer noch mit denen der Eingabematrix überein (es werden lediglich Pointer gesetzt). So kann eine Matrix des neuen View ein gleiches Feld der Eingabe wie eine andere Matrix erhalten, da der Filter manche Felder mehrfach in unterschiedlichen Schritten abdeckt. Die beschriebene Backpropagation wird ebenfalls durch mathematische Techniken wie die Einsteinsche Summenkonvention und fortschrittliche Indexierung dank Numpy vereinfacht, um eine schnelle Laufzeit zu erreichen (für weiter Information siehe Anhang Seite 24 „Einsteinsche Summenkonvention“). Die darauffolgende Flatting Klasse „Flatten“ besitzt nur die Parameter der Eingabegröße, um diese in der Backpropagation wieder herzustellen. Die letzte Klasse der Convolutional Layer „Conv2D“ besitzt wieder ähnliche Parameter wie die Pooling Klasse. So ist für das Pooling die Filtergröße(kernel_size), Schrittweite/Stride (step) und der Padding (p) abgespeichert. Zusätzlich besitzt diese die Parameter für den Gewichtsinitializer (weight_initializer) und Geweichtsregularisierer (kernel_regularizer). Die Parameter werden ähnlich der Dens-Klasse durch die Funktion „initialize_parameters“ erstellt. Für die Forwardpropagation wird wie zuvor eine neue View (Ansicht) auf die Eingabe durch prepare_-

[24]Quelle: Patricia S. Chureland, Grundlagen zur Neuroinformatik und Neurobiologie, Seite 156 [?]
[25]Quelle: Tariq Rashid, Neuronale Netze - selbst programieren, Seite 2003ff. [4]

submatrix erstellt, auf welche leichter die Convolutional Operation angewendet werden kann. Für die Backpropagation wird sich ebenfalls wieder der Einstienschen Summenkonvention bedient, wodurch der Fehlergradient leichter berechnet werden kann.

Das Hauptmodul Net beinhalten die Klasse Network, welche die zuvor definierten Klassen zu einen Neuronalen Netz vereint. Durch „add"kann diesem eine Schicht einer der vorigen Klassen hinzugefügt werden. Dabei wird die Netzarchitektur als Parameter in „architecture"(Größe der jeweiligen Ausgabe) und layer_name(Name der Layer in Aufstellungsreihenfolge des Netzes) gespeichert. Diese Netzarchitektur kann dann über die Funktion „summary" ausgegeben werden. Trainiert wird das Netz über die Funktion „fit", welche den gespeicherten Optimizer auf die Ausgabe der Schichten anwendet, um die Änderung für die Parameter jeder Schicht zu erstellen. Daraufhin werden alle Parameter angepasst. Nachdem das Netz den Lernprozess beendet hat, kann eine Ausgabe mithilfe der Funktion „predict"erstellt werden. [26] [27]

4.2 Untersuchung der Netzstruktur

Ein Convolutional Neural Network kann in vielen Parametern verändert werden, um die höchste Genauigkeit des Netzes zu erzeugen. Man muss zu Beginn der Entwicklung eines CNN's also erst einmal herausfinden, wie man dieses entwirft und welche Eigenschaften man diesem zuspricht. Um die besten Merkmale herauszufiltern, muss man das Neuronale Netz mehrfach trainieren und dabei jeweils die Werte der einzelnen Parameter ändern. Anschließend gibt das Programm eine Akkuratesse für das jetzige Netz aus und man versucht als User die größtmögliche Genauigkeit durch die Veränderung dieser Werte zu erzeugen. Eigenschaften, welche eine hohe Zuverlässigkeit liefern, sollten beibehalten werden, während schlechte Merkmale, welche das Netz zu ungenau machen, verändert werden sollten. Diese gilt es herauszufinden und anschließend das Optimum festzulegen. Auch bei der Pneumonia Detection gibt es solche Maxima für alle Parameter, daher startet der Entwurf dieses Netzes erst einmal mit der Analyse aller Faktoren, welche Einfluss auf das Netz haben könnten. Die wichtigsten Faktoren sind hierbei die Anzahl der Convolutional Layer und die Anzahl der Epochen, welche beim Training durchlaufen werden müssen. Bei diesen Veränderungen der Parameter muss besonders auf Over- und Underfitting geachtet werden. Sie beschreiben eine Ungenauigkeit des Netzes, wobei beim Underfitting aufgrund der Einfachheit des Netzes die Muster in den Daten nicht erkannt und daher auch später in den Testdaten keine Ungewöhnlichkeiten gesichtet werden und beim Overfitting das Netz zu viel trainiert wurde und es daher an die Trainingsdaten angepasst wird, es die Testdaten aber nicht erkennt, weshalb die wahre Genauigkeit des Netzes wieder sinkt. Die Überanpassung kann man dadurch verhindern, dass man nicht zu viele Schichten in das Netz einbaut und das CNN nicht zu viele Epochen durchlaufen lässt. Unteranpassung kann man dadurch verhindern, dass man nicht zu wenig Trainingsdaten besitzt und außerdem ausreichend Epochen durchläuft, damit das Neuronale Netz ausreichend trainieren kann.

[26] vipinyada, numpy.einsum() Method [**?**]
[27] Ujjwal Khandelwal, CNN model using Tensorflow - Keras [19]

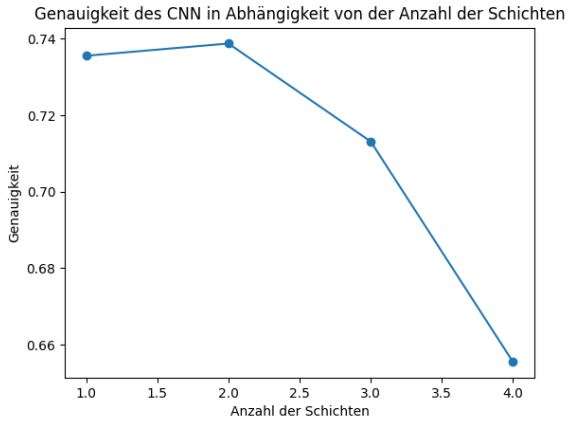

Abbildung 2: Anzahl der Schichten

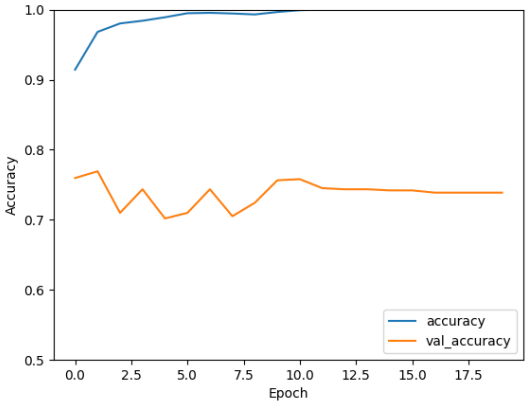

Abbildung 3: Anzahl der Epochen

Man startet damit die Anzahl der Convolutional Schichten zu analysieren, welche das Netz für das Hervorheben von Details benötigt. Dazu entwickelt man ein Programm mit einer for-Schleife, welche über die Anzahl der Schichten iteriert und mit dieser Anzahl dann immer ein neues Neuronales Netz aufbaut und dieses dann von Grund auf trainiert. Anschließend wertet man das jeweilige Netz einmal aus und speichert die Wahrscheinlichkeiten in einer Liste. Wenn alle Schichtenanzahlen durchlaufen sind, lässt man sich diese in einem Graphen mit einer Kurve ausgeben. Daran erkennt man dann am besten, ab wann Overfitting stattfindet und wo das Netz die höchste Wahrscheinlichkeit besitzt. Im Graphen dazu (Siehe Abb.: 2) erkennt man deutlich, dass ab 3 Schichten die Genauigkeit des Netzes wieder sinkt. Das symbolisiert die Überanpassung und daher entscheidet man sich hier für 2 Convolutional Layer. Folgend untersucht man die Epochenanzahl, während man die bereits herausgefilterte Schichtanzahl verwendet. Dies gelingt, indem man ein CNN entwickelt mit 2 Convolutional Layer und eine beliebig hohe Epochenanzahl in das Training eingibt. Schließlich gibt das Netz eine Präzision für jede Epoche zurück, welche man sich ebenfalls in einem Graphen speichern sollte. Im Diagramm für dieses Merkmal (siehe Abb.: 3) wird ersichtlich, dass die höchste Akkuratesse bei ungefähr 10 Epochen erzielt wird, weshalb man das Netz ebenfalls auf 10 Epochen trainieren sollte. Damit ergeben sich die besten Eigenschaften des Neuronalen Netzes für die Erkennung von Lungenerkrankungen, welche die akkuratesten Ergebnisse liefern. Unser Netz, welches wir mithilfe unseres eigenen Modules erstellten, erreichte eine Genauigkeit von über 77%. Damit ist es vergleichbaren Netzen anderer Module, wie Tensorflow überlegen. Untersuchungen mit diesen Netzen die in der Abbildung dargestellten 74%. Jedoch trainieren diese schneller, weshalb wir diese für Untersuchungen der Netzstruktur verwendeten. Das erstellte Netz kann somit mit einer Wahrscheinlichkeit von 77% ein Bild der Richtigen Klasse ('Gesund', 'virale Pneumonea', 'bakterielle Pneumonea') zuordnen.

5 Datensatz

5.1 Kaggle als Datenbank

Der Datensatz stammt von „Kaggle ". Dieser Datensatz besteht aus 5863 Röntgenbildern der Brust, zwei Kategorien („Pneumonia "und „Normal ") und drei Ordnern. Der erste Ordner enthält den Trainingsdatensatz, welcher zum Trainieren des CNNs benutzt wird. Der zweite Ordner enthält den Testdatensatz, mit welchem man nach dem Trainieren beurteilen kann, wie gut das CNN auch neue Datensätze klassifiziert. Zuletzt enthält der dritte Ordner den Validierungsdatensatz, mit welchem man das CNN gegen Overfitting, also Anpassen an den Trainingsdatensatz, optimiert. Alle diese Röntgenbilder stammen von

Kindern im Alter von 1-5 Jahren vom 'Guangzhou Women and Children's Medical Center'. Jedes Bild wurde dabei von zwei erfahrenen Ärzten diagnostiziert und die Bilder niedriger Qualität aussortiert, bevor der Datensatz zum Trainieren von KIs freigegeben wurde.[28]

5.2 Das Einlesen und Vervielfältigen von Daten

Nach dem Initialisieren eines neuronalen Netzes muss dieses auf das Problem angepasst werden. Für diese Seminarfacharbeit muss das Netz erlernen Pneumonie auf Röntgenbilder zu erkennen, dafür werden Trainingsbilder benötigt. Wichtig ist eine große Vielfalt an Trainingsdaten. Wie in diesem Fall ist nicht immer die benötigte Menge an Lerndaten vorhanden. Gründe dafür sind zum Beispiel noch fehlende Forschung auf diesem Gebiet und Komplikationen, wie dem Datenschutz, welche das Sammeln von Daten erschweren. Eine Lösung für dieses Problem ist die Vervielfältigung von bestehenden Datensätzen. Das bedeutet man erzeugt neue Bilder in dem man die vorhanden Bilder durch zum Beispiel Rotation, Verschiebung, Scherung oder Zooming modifiziert. Im idealerweise werden einzelne Bilder nicht mehr als 5 Mal vervielfältigt, um immer noch eine Varianz im Trainingsdatensatz zu gewährleisten. Datenvervielfältigung bietet neben der erhöhten Datenanzahl, weitere Vorteile, dabei hilft es dem CNN sich auf verschiedenere Szenarien vorzubereiten und robuster gegenüber Transformationen und Verzerrungen zu sein, welche in der Realität meist nicht zu vermeiden sind. Durch die erhöhte Anzahl an Trainingsbildern, die sich unterscheiden, wird das Risiko von Overfitting des CNN deutlich gesenkt. Das heißt, dass das CNN auch mit neuen Datensätzen arbeiten kann und sich nicht einfach an die Trainingsdaten anpasst.
Es wurden mehrere Module benutzt. So ist "cv2"für das Einlesen und Verarbeiten von Bildern zuständig. "numpy"wird benötigt um beispielsweise mit Arrays oder Matrizen zu arbeiten. "matplotlibïst nützlich um sich Diagramme für die Pixelanzahl der Bilder ausgeben zu lassen. ös"wird verwendet, da es essentiell für die Arbeit mit dem Verzeichnis der Dateien und zum navigieren und damit zu interagieren. Das Modul "randomïst nötig, um die Daten zu mischen, damit das CNN sich nicht an Muster in den Trainingssätzen anpasst. Das "pickleModul ist zum Zwischenspeichern von den Bildern und Labeln, damit der Trainingsvorgang pausiert und dann wieder aufgenommen werden kann, ohne neu Anfangen zu müssen. Zuletzt wird das ÏmageDataGeneratorModul von Keras importiert, um die „Data Augmentation“, also die Datenvervielfältigung, durch die Beschriebenen Methoden, durchzuführen.

6 Erweiterung der Pneumonie Erkennung

Ärzte haben den Vorteil, dass sie weitere Daten wie das Alter, den BMI oder das Geschlecht berücksichtigen können. Daher können sie genauere Aussagen bezüglich der Lungenerkrankung treffen. Sie erkennen zum einen das Röntgenbild und die darauf abgebildete Erkrankung und zum anderen haben sie die eben genannten Faktoren im Hinterkopf. Das erhöht die Genauigkeit, die richtige Entscheidung zu treffen. Um einer künstlichen Intelligenz die Möglichkeit zu geben andere Faktoren als nur das Röntgenbild zu beachten bedarf es ausreichend Daten. Alle Röntgenbilder in einem Datensatz müssen mit Alter, BMI und Geschlecht klassifiziert sein, damit Neuronale Netz erkennen kann, ob es eventuell Zusammenhänge zwischen diesen Faktoren und der Lungenerkrankung gibt. Beispielsweise findet das CNN dann heraus, in welchem Alter die Pneumonie am häufigsten auftritt und beachtet dann das Alter als weiteren Wert bei der Ergebnisfindung. Dies ist zurzeit jedoch nicht möglich, da es eben jene Datensätze aufgrund von Datenschutz und sonstigem nicht gibt und man daher kein Neuronales Netz mit dieser Erweiterung aufsetzen kann. Um dieses Problem zu lösen und eine künstliche Intelligenz zu entwickeln, welche diese Faktoren doch beachten kann, erstellt man eigene Datensätze.

[28] Paul Mooney, Chest X-Ray Images (Pneumonia), [20]

Auf der Grundlage von Studien kann man deutlich erkennen, dass die Lungenerkrankungen am häufigsten im Alter von 80 bis 89 auftreten. Die Grafik[29] lässt ebenfalls erkennen, dass die Häufigkeit pro Alter wie eine Normalverteilung aussieht. Basierend auf diesen herausgefilterten Informationen kann man nun einen eigenen Datensatz erstellen. Dafür nutzt man den schon bereits vorher verwendeten Datensatz und speichert sich die Bilder ab. Dadurch erhält man die Anzahl der Bilder, welcher dieser umfasst und muss eine Liste mit genau so vielen Altern erstellen. Dabei unterscheidet man einmal in die Bilder, auf welchen eine Pneumonie erkannt wurde und die gesunden Lungen. Für die Bilder mit einer Pneumonie erstellt man eine Normalverteilung, welche 89 als Höhepunkt der Verteilung betrachtet und eine Standardabweichung von 23 besitzt. Aus der daraus erstellten Verteilung nimmt man sich nun n zufällige Alter und stellt auch sicher, dass diese Alter zwischen 0 und 90 liegen, wobei n die Anzahl der Pneumonie Bilder darstellt. Dadurch entsteht dann eine Liste, welche am häufigsten Alter im Bereich von 80 bis 89 beinhaltet. Dasselbe macht man dann noch einmal für die gesunden Lungenbilder, jedoch nimmt man hier 0 als den Höhepunkt dieser Verteilung. Nachdem man nun 4 Listen (zwei für die Trainingsdaten und zwei für die Testdaten) erstellt hat, iteriert man nun über alle Bilder. Von diesen Bildern untersucht man das jeweilige Label, welches angibt, ob eine Pneumonie vorliegt oder nicht. Liegt eine Pneumonie vor, so weist man diesem Bild ein Alter aus der Liste der Pneumonie-Erkrankten zu. Andernfalls nimmt man ein Alter aus der Liste der gesunden Lungen. Dies geschieht so lange, bis allen Bildern ein Alter zugewiesen ist, welches man später als weiteren Faktor für die Analyse betrachten kann.
Der nächste Schritt ist die Entwicklung des Convolutional Neural Networks, welches zwei einzelne CNN's zusammenführt und eine gemeinsame Ausgabe bildet. Dazu muss man erst einmal zwei einzelne Netze aufsetzen, welche unterschiedliche Strukturen haben. Das eine ist eine normale Bilderkennung, welche aus zwei Convolutional Schichten besteht und als Ausgabe eine Dense Schicht besitzt. Das zweite ist kein CNN sondern nur ein Netz mit einer Eingabe- und einer Ausgabeschicht. Zum Abschluss entwickelt man hier noch ein Neuronales Netz welches mithilfe der Tensorflow Funktion Concatenate arbeitet. Diese ist dafür da, dass zwei Ausgaben aus vorherigen Netzen zu einer einzelnen zusammengefügt werden. Anschließend folgen noch einmal Dense Layer, welche die Ausgabe analysieren und ein Ergebnis bilden. Bei der Zusammenführung der beiden vorherigen Netze ist besonders darauf zu achten, dass die Wichtigkeit des Netzes für die Bilderkennung deutlich höher liegt als die des Netzes für die Listen. Das erreicht man zum Beispiel dadurch, dass man die Anzahl der Output Neuronen der jeweiligen Netze unterscheidet. Durch Experimentieren findet man ziemlich schnell heraus, dass das erste CNN 11 Output-Neuronen besitzt und das zweite 1 Output-Neuron. So verhindert man, dass sich die KI mehr auf das Alter konzentriert und lediglich lernt, bei welchem Alter Pneumonie auftritt. Durch diese Einstellung hat es den perfekten Kontrast zwischen genug Informationen und der richtigen Aufteilung. Anschließend trainiert man noch das Netz und hat, verständlicherweise, die Bilder und die Listen als Eingabe. Nach dem Training erhält man mit dem eben erläuterten Netz eine Genauigkeit von 80,25% wenn man den nicht vervielfältigten Datensatz verwendet.

7 Ethische Betrachtung

Immer wenn es um künstliche Intelligenzen (KIs) geht, muss man immer auch den ethischen Aspekt mitbetrachten. Und so auch hier. Es wird schon heute, wo das Zeitalter der KIs erst anfängt, schon stark diskutiert, wo man Grenzen zieht und wie weit man gehen darf. Viele Menschen sind sich aber sicher, dass eine KI egal in welchem Anwendungsgebiet aktuell noch keinen Menschen ersetzen sollte, sondern ihn lediglich unterstützen. In der Medizin ist es zum derzeitigen rechtlichen Stand einer KI

[29] Quelle: Klaus Dalhoff, Ambulant erworbene Pneumonie bei Erwachsenen[10]

nicht möglich, genaue Diagnosen oder Behandlungen zu ermitteln, da diese die Hintergrunddaten wie beispielsweise die medizinische Vorgeschichte, das Alter oder das Geschlecht des Patienten nicht kennt. Ein weiteres Problem mit KI ist Rassismus, Vorurteile und Diskriminierung. Zum Beispiel gab es eine Studie von Forschern aus Australien, Kanada und den USA, wo festgestellt wurde, dass tiefe neuronale Netze die Ethnie eines Patienten bestimmen können, ohne Daten zur Verfügung zu haben, die dazu Aufschluss geben würden. Dabei wurde festgestellt, dass eine KI zur Versorgungspriorisierung in den USA Afroamerikaner benachteiligt. Da die Studie sich nicht mit anderen visuellen Merkmalen wie Alter, BMI, Geschlecht oder Knochendichte der Patienten befasst hat, konnten sich die Forscher die Merkmale nach denen klassifiziert wurde, nicht erklären. Der vermutete Grund dafür ist, dass die Datensätze die Vorurteile des Erstellers und damit auch meist der Gesellschaft übernehmen. Die KI will nicht rassistisch sein, wird aber dazu trainiert. Ein Lösungsvorschlag ist, dass ein generatives Modell künstliche Datensätze, mit denselben statistischen Eigenschaften des Trainingsdatensatzes erschafft. Entstehen in einem Datensatz also Lücken, die zu diskriminierenden Entscheidungen führen, füllt das Modell diese Lücke auf. So sollen gerechte KIs geschaffen werden.[30] [31]

[30] David Beck, Warum KI nicht frei von Vorurteilen ist, [6]
[31] Anna-Lena von Hodenberg, Rassismus im Code: Wie KIs Vorurteile verstärken, [27]

8 Fazit

Lungenentzündung fordern jährlich ca. 2,5 Tode. Eine Ursache dafür ist meist eine Fehldiagnose oder verspätete Erkennung der Symptome. Die Arbeit leistet einen wichtigen Beitrag zur Verbesserung der Diagnose von Lungenentzündung mit Hilfe von künstlicher Intelligenz. Sie zeigt, dass das Modell nicht nur die Genauigkeit der Bilderkennung erhöht, sondern auch die individuellen Merkmale der Patienten berücksichtigen kann. Diese Methode übertrifft die bisherigen Ansätze, sowohl in Bezug auf die Leistung als auch auf die Anpassbarkeit. Jedoch hat diese Arbeit auch einige Herausforderungen und Einschränkungen aufgedeckt, die in zukünfitgen Forschungen gelöst werden müssen. Zum Beispiel wurde ersichtlich, dass die Qualität und Quantität der verfügbaren Daten einen großen Einfluss auf die Ergebnisse des Modells haben. Es waren verschiedene Techniken nötig, um mehr Lerndaten aus den vorhanden zu generieren und so das Netz robuster gegenüber unreinen Datensätzen zu machen. Außerdem haben wir die Hyperparameter unseres Modells durch mehrere Experimente optimiert, um die bestmögliche Konfiguration zu finden.
Diese Robustheit des Netzes wurde anhand verschiedener Tests untersucht. Diese Schritte waren notwendig, um die Zuverlässigkeit und Reproduzierbarkeit unserer Methode zu gewährleisten, aber sie erforderten auch viel Zeit und Rechenressourcen. Durch diese zahlreichen Testserien konnte eine optimale Netzstruktur für das Problem gefunden und so eine bestmögliche Genauigkeit erzielt werden. Durch das eigens dafür erstellte Modul konnte mit einem von Grund auf entwickelten Netz eine Genauigkeit von über 77% erreicht werden, womit das Netz mehr Bilder korrekt bestimmt, als manche Ärzte. Da unser Modell die Erkennung zusätzlich in Sekundenbruchteilen vornimmt bietet es bereits zahlreiche Vorteile für die Anwendung im Medizinischen Bereich. Ein bisheriger Vorteil, welchen Ärzte gegenüber künstlichen Intelligenzen besaßen war, dass sie mehr Daten über den Hintergrund der Patienten besitzen. Um eben jenen Vorteil für ein Neuronales Netz auszugleichen, wurden Datensätze erstellt, welche das Alter der Patienten bereits berücksichtigten. Damit wurde bereits eine Genauigkeit von 80,25% erzielt, was bereits deutlich aufzeigt, dass das Neuronale Netz mit diesem erweiterten Datensatz die Ergebnisse viel genauer bestimmen kann.

Eine weitere Schwierigkeit bestand im Gruppenklima, welches anfangs Probleme in der Kommunikation aufwies. Dies führte zu Missverständnissen, Verzögerungen und Schwierigkeiten beim Zusammenfügen einzelner Codeteile. Um dieses Problem zu lösen, haben wir einen festen Arbeitsplan erstellt, der die Aufgaben, die Ziele und die Termine für jedes Gruppenmitglied klar definiert hat. Dadurch wurde die Arbeit koordinierter und effizienter. Dies wurde unter anderem auch durch einen guten und freundlichen Umgang innerhalb der Gruppe ermöglicht. Auch wenn immer noch Probleme wie Fehler in den Datensätzen auftreten und eine geringe Anzahl an Daten vorliegen, ist bereits eine großflächige Anwendung diese Netze denkbar.Da die Netze die Effizienz und Genauigkeit der Bestimmung Radiologischer Befunde erhöht, können Ergebnisse direkt nach dem Röntgen erstellt und die Patientenversorgung beispielsweise automatisiert werden. Dies könnte man in Zukunft beispielsweise dadurch erreichen, dass man weitere Eckdaten von Patienten berücksichtigt und das Neuronale Netz so noch spezifischer macht. Des weiteren wird es in Zukunft eventuell möglich sein, dass auch Krankenhäuser diese Daten veröffentlichen können und so genauere Datensätze vorliegen welche nicht über eine Normalverteilung erstellt werden müssen. Dies würde ebenfalls wieder eine höhere Genauigkeit bedeuten.

9 Anhang

Herleitung der Backpropagation

Im Folgenden wird die Herleitung der mathematischen Berechnungen der Backpropagation vorgenommen. Dafür wird das Gradientenabstiegsverfahren genutzt, da es die Grundlage vieler weiterer Optimizer bildet.

Der Gradient Descent oder Gradientenabstieg ist ein mathematischer Algorithmus zum Lösen von Optimierungsproblemen. Er wird beim maschinellen Lernen dazu eingesetzt, den niedrigsten Fehler beziehungsweise die beste Genauigkeit zu finden. Dazu stelle man sich einen Graphen vor, in dem die Abhängigkeit des Fehlers von den Gewichten oder des Bias dargestellt ist. Der Gradient zeigt immer in die Richtung des größten Anstiegs einer Funktion, in diesem Fall der Verlustfunktion. Dies wird genutzt, indem wir die entgegengesetzte Richtung, die auf den größten Abstieg zeigt, nutzen. Die Schrittweite ist dabei von der Stärke der Steigung abhängig. Sie kann durch einen Lernparameter e (siehe Anhang: Lernparameter e, S.23) beeinflusst werden, indem man ihn mit der Gewichtsänderung multipliziert. Dieser Prozess wird wiederholt, um so die tiefste Stelle (den kleinsten Fehler) zu erreichen und ist vergleichbar mit einem Ball, der einen Berg hinunterrollt. Dadurch erhalten wir die Gewichts- und Biasänderung und können diese somit an unsere Daten anpassen.[**?**]

Fehlerfindung und Verlustfunktion

Um die Gewichtsänderung zu berechnen, muss man zuvor den Fehler der jeweiligen Schicht/Layer kennen. Dabei unterscheidet man in mehrdimensionalen Netzen bei der Berechnung des Fehlers zwischen output layer/Ausgabeschicht und hidden layer/versteckte Schichten.

Der Fehler des Netzes wird mit der Fehlerfunktion - Cost-function oder Loss-function $C(y_{net}, y_{soll})$ genannt - berechnet. In der Funkion wird die Ausgabe des Netzes y_{net} mit dem optimalen beziehungsweise erwarteten Ergebnis y_{soll} verglichen. Sie gibt an, wie gut das neuronale Netz arbeitet.
Es gibt verschiedene Fehlerfunktionen.
Die erste ist die des Mean square error/Mittleren Quadratischen Fehlers (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{net} - y_{soll})^2 \tag{4}$$

$$HMSE = \frac{1}{2n}\sum_{i=1}^{n}(y_{net} - y_{soll})^2 \tag{5}$$

$$SMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{net} - y_{soll})^2} \tag{6}$$

(2) und (3) sind dabei Variationen des MSE. Beim HMSE wird nochmal die Hälfte des quadratischen Fehlers genommen. Der SMSE ist die Wurzel aus dem quadratischen Fehler.
Beim MSE wird aus einem Testsatz für jede Eingabe die Differenz zwischen der Netzausgabe und der gewünschten entnommen und addiert. Diese Summe wird dann quadriert und durch die Anzahl der Testdaten geteilt, um den durchschnittlichen Fehler des Testsatzes zu finden.

Der MAE ist der mittlere absolute Fehler. Der Unterschied zum MSE besteht darin, dass die Summe der Differenzen nicht quadriert wird, sondern nur der Betrag der Differenz maßgebend ist. Die Formel

lautet:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_{net} - y_{soll}|$$

Bei diesen beiden Funktionen ist es egal, ob $y_{net} - y_{soll}$ oder $y_{soll} - y_{net}$ gerechnet wird, da das Vorzeichen durch das Quadrieren beziehungsweise durch den Betrag entfernt wird. MSE und MAE unterscheiden sich dabei in der Einbeziehung der Entfernung von sogenannten „Ausreißern". Der MSE gewichtet entferntere Datenpunkte (von der Durchschnittsgerade) mehr, der MAE geht wenig auf „Ausreißer" (siehe Abb.: 4) ein. Ist der Fehler zum Beispiel: 10, so ist er beim MAE ebenfalls 10, aber beim MAE 100.

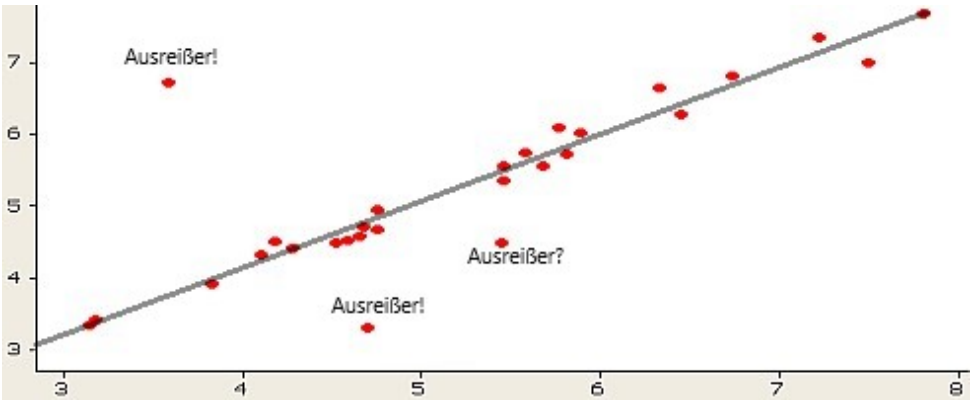

Abbildung 4: Ausreißer

Diese Funktionen sind beide sehr extrem, deshalb gibt es noch eine weitere Funktion, die die Vorteile beider Funktionen vereint. Sie heißt „Huber Loss" oder auch „Smooth Loss". Bei ihr werden Fehler bis zu einem bestimmten Deltawert (δ) nach der MSE Funktion und über diesem Wert mit der MAE Funktion behandelt. Dies bietet den Vorteil, dass das Netz nicht zu „fein"lernt und dadurch nur ein lokales Minimum findet, aber auch nicht zu „grob" und hierdurch den Optimalpunkt überspringt. Der Deltawert kann für jedes Problem angepasst werden:[17]

$$L(y_{net}, y_{soll}) = \begin{cases} \frac{1}{2n}\sum_{i=1}^{n}(y_{net} - y_{soll})^2 & \text{für } |y_{net} - ysoll| \leq \delta \\ \delta \cdot \frac{1}{n}\sum_{i=1}^{n}|y_{net} - y_{soll}| \quad \cdot \frac{1}{2}\delta & \text{für } |y_{net} - ysoll| > \delta \end{cases}$$

Für die Backpropagation wird die Ableitung der Fehlerfunktion benötigt. Diese müssen wir nach jedem Gewicht im Netz berechnen:

$$\frac{\partial C}{\partial W_{jk}^{(i)}}$$ [32]

Die Fehlerfunktion wird für den letzten Layer berechnet. Allerdings besitzt ein neuronales Netz oftmals mehr als eine Schicht. Diese anderen Schichten werden nach der jeweils zuvor berechneten Schicht verändert. Da die Fehlerfunktion auch von diesen Schichten abhängt, können wir diese mithilfe der Kettenregel berechnen. Sie besagt, dass für eine Funktion z in Abhängigkeit von y, wobei y(Ergebnis)

[32] $W_{jk}^{(i)}$ = Gewichtung der Verbindung vom Neuron j zu Neuron k in Schicht i

von x(Eingabedaten) abhängt, die Ableitung von z in Bezug auf x gegeben ist durch:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Da die Ableitung einer Schicht von der folgenden Schicht abhängt, kann man rückwärts durch das Netz gehen und die Ableitung für die nächste zu berechnende Schicht speichern. Somit spart man sich doppelte Berechnungen. Dies ist der Prozess der Backpropagation. Für die Berechnung der Änderung muss die Fehlerfunktion auch noch mit der Ableitung der Aktivierungsfunktion multipliziert werden, weshalb diese differenzierbar sein muss und ihre Ableitung nicht überall gleich null sein darf (für eine Beispielrechnung siehe Anhang: Beispielrechnung zur Backpropagation, S.**??**).[**?**]

Herleitung der Gradienten von den Dens-Layern

Um den Gradienten zu berechnen müssen wir uns das Netz als Funktion vorstellen. Dies lässt sich am besten an den fully-connected-Layer erklären. Diese besitzen jeweils Eingabeneuronen, versteckte/Hidden Layer und eine Ausgabeschicht.

Die Ausgabe der ersten versteckten Schicht ist wie folgt definiert:

$$H_1 = f_1(W_1 \cdot x + b_1) \quad [33]$$
$$H_2 = f_2(W_2 \cdot H_1 + b_2)$$

Die Ausgabe der letzten Schicht/output layer bzw. das Ergebnis des Netzes ist demzufolge:

$$y_{net} = W_N \cdot H_{N-1} + b_N$$

Für ein Neuron lässt sich die Ausgabe wie folgt berechnen:

$$o_j = f_j(\sum_{i=1}^{n} w_{ji} \cdot x_i \quad + b_j) \quad [34]$$

Es ist erkennbar, dass sich die Ausgabe eines Neurons aus der Summe der Eingaben plus den Bias-Wert und diese eingesetzt in die Aktivierungsfunktion ergibt.

Dabei ergibt das komplette Netz eine verschachtelte Funktion. Für den Gradienten müssen wir diese Funktion nun ableiten.
Für die Ausgabeschicht mit einer Softmax Funktion berechnet ist die Ausgabe wie folgt definiert: $out_s(c) = \frac{e^{t_c}}{\sum_{i=1}^{n} e^{t_i} = \frac{e^{t_c}}{S} = e^{t_c} S^{-1}}$ mit $S = \sum_{j=1}^{n} e^{t_i}$ und t_i als Summe aller Eingaben an dem Neuron i. Mithilfe der Kettenregel können wir dies nun für alle Neuronen $k \neq c$ umstellen zu : $\frac{\partial out_s(c)}{\partial t_k} = \frac{\partial out_s(c)}{\partial S}\frac{\partial S}{\partial t_k} = -e^{t_c} S^{-2}(\frac{\partial S}{\partial t_k}) = -e^{t_c} S^{-2}(e^{t_k}) = \frac{-e^{t_c} \cdot e^{t_k}}{S^2}$. Die Ableitung für die Klasse C können wir mithilfe der Quotientenregel wie folgt ableiten: $\frac{\partial out_s(c)}{\partial t_c} = \frac{Se^{t_c} - e^{t_c}\frac{\partial S}{\partial t_c}}{S^2} = \frac{Se^{t_c} - e^{t_c}e^{t_c}}{S^2} = \frac{-e^{t_c} \cdot (S - e^{t_c})}{S^2}$. Zusammengesetzt ergibt sich folgende allgemeine Formel für

[33] H_i = Ausgabe des Layers i; W_i = Gewichtsmatrix des Layer i; b_i = Biasmatrix des Layer i; $f_i()$ = Aktivierungsfunktion des Layer i;
[34] o_i = Ausgabe des Neurons j; w_{ij} =Gewicht vom Neuron i zu Neuron j

diese letzte Softmax-Schicht: $\frac{\partial out_s(k)}{partialt_k} = \begin{cases} \frac{-e^{t_c} e^{t_k}}{S^2} & \text{falls } k \neq c \\ \frac{-e^{t_c}(S-e^{t_c})}{S^2} & \text{falls } k = c \end{cases}$. Dabei ist L die Loss oder Verlustfunktion und *out* ist die Ausgabe der Softmax Schicht.

Gradient nach den Parametern

Um nun die Parameter des Netzes wie Bias Gewichte Anpassen zu können müssen wir den Gradienten nach den Parametern ableiten. Wie bereits beschrieben ist die Ausgabe eines Neurons: $t = w \cdot input + b$. Mithilfe dieser Gleichung können wir nun die benötigten gradienten bilden :

$$\frac{\partial t}{\partial w} = input$$
$$\frac{\partial t}{\partial b} = 1$$
$$\frac{\partial t}{\partial input} = w$$

Wenn wir nun alles zusammensetzten erhalten wir folgende Gleichung:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial out} \cdot \frac{\partial out}{\partial t} \cdot \frac{\partial t}{\partial w}$$
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial out} \cdot \frac{\partial out}{\partial t} \cdot \frac{\partial t}{\partial b}$$
$$\frac{\partial L}{\partial input} = \frac{\partial L}{\partial out} \cdot \frac{\partial out}{\partial t} \cdot \frac{\partial t}{\partial input}$$

Hier ist w das Gewicht, b der Bias und *input* die Eingabe dieser Schicht.

Die Berechnung der Gewichtsänderung

Für unser neuronales Netz bringt uns der Fehler alleine nichts, wir müssen es auch so verändern können, dass der Fehler kleiner wird, um eine höhere Genauigkeit zu erreichen.
Dafür verändern wir die Gewichte und den Bias jedes Layers. Für die Gewichtsänderung brauchen wir die Ableitung der Fehlerfunktion zum Gewicht des Layers und den Lernparameter. Die Formel lässt sich vereinfachen zu $delta_j$ (dem Fehler vom Neuron j) multipliziert mit o_i (der Ausgabe des Neurons i:

$$\Delta w_{ij} = -\eta \frac{\partial C}{\partial w_{ij}} = -\eta \delta_j o_i$$

Bei der Berechnung von δ_j muss wieder zwischen Ausgabeschicht/output layer und versteckter Schicht/hidden layer unterschieden werden:

$$\delta_j = \begin{cases} f(w_j \cdot input)(o_i - y_{soll}) & \text{falls j ein Ausgabeneuron ist} \\ f(w_j \cdot input) \sum_{k=1}^{K} \delta_k w_{jk} & \text{falls j ein verdecktes Neuron ist} \end{cases}$$ [35]

Das neue Gewicht wird dann mit Hilfe der zuvor berechneten Gewichtsänderung berechnet:

$$w_{ij}^{neu} = w_{ij}^{alt} + \Delta w_{ij}$$

Um das neuronale Netz noch mehr anzupassen, kann der Lernparameter η während der Lernphasen angepasst werden, zum Beispiel mit jedem Lerndurchlauf wird η verkleinert, um am Anfang schnelle Ergebnisse zu erhalten und am Ende den Optimalpunkt durch einen zu großen Lernparameter nicht zu

[35] K = Anzahl der Neuronen der folgenden Schicht; k = k Neuron der folgenden Schicht

überspringen.
Eine weitere Möglichkeit besteht darin, die letzte Gewichtsänderung(zum Zeitpunkt t) in die aktuelle Gewichtsänderung(zum Zeitpunkt t+1) mit zu integrieren. Dies wird durch einen Trägheitsterm (α) erreicht. Je größer α ist, umso mehr ist die aktuelle Gewichtsänderung von der letzten abhängig. Dieser Prozess kann mit dem Ball, der einen Hang hinunterrollt, verglichen werden, da dessen Geschwindigkeit nicht nur von der aktuellen Steigung, sondern auch von der eigenen Trägheit abhängt. Der Term lautet dann:

$$\Delta w_{ij}(t+1) = -((1-\alpha)\eta\delta_j x_i + \alpha\Delta w_{ij}(t))$$

Dieser Prozess behebt die Probleme des Gradientenabstiegs bei steilen Abhängen oder langen Ebenen. Bei einer Ebene wird die Loss function sehr klein und kommt zum Stillstand, während sie mit dem Trägheitsterm den vorigen Schwung nutzen kann, um die Ebene zu verlassen.[**?**]

Herleitung der Gradienten von den ConvLayern

Im Folgenden wird die Berechnung der Gradienten, welche für die Backpropagation der ConvLayer benötigt wird ausführlich hergeleitet. Damit die Backpropagation von einer Matrixmultiplikation durchgeführt werden kann, wird Formel für die Ausgabe eines ConvLayer genutzt: $O_11 = X_11 \cdot F_{11} + X_{12} * F_{12} + X_{21} \cdot F_{21} + X_{22} \cdot F_{22}$ (Im Folgenden wird beispielhaft eine 3x3 Eingabematrix und ein 2x2 Filter angenommen). Somit ist die Ausgabe/Output O_i einer Matrixmultiplikation definiert durch die Formel: $O_i = \sum X_k \cdot F_k$ (für alle k Felder die der Filter an O_i abdeckt). So lässt sich auch als Ausgabe eines Summenneurons mit der Eingabematric X und der Gewichtsmatrix F ohne Aktivierungsfunktion vorstellen. Durch diese lineare Formel kann nun wie zuvor nach der Kettenregel abgeleitet werden: $\frac{\partial L}{\partial F} = \frac{\partial L}{\partial O} \cdot \frac{\partial O}{\partial F}$ wobei $\frac{\partial L}{\partial F}$ der für die Parameteranpassung benötigte Gradient, $\frac{\partial L}{\partial O}$ der Fehlergradient der vorhergehenden Schicht und $\frac{\partial O}{\partial F}$ der Gradient dieser Schicht bzw. der lokale Gradient ist. Wie beschrieben, handelt es sich hierbei um eine partielle Ableitung einer Matrix O nach einer Matrix F, weshalb die Formel wie folgt erweitert werden muss: $\frac{\partial L}{\partial F_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} \cdot \frac{\partial O_k}{\partial F_i}$. Für die Beispielmatrix würde das ausgeschrieben wie folgt aussehen:
$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_1 1} \cdot \frac{\partial O_{11}}{\partial F_{11}} + \frac{\partial L}{\partial O_{12}} \cdot \frac{\partial O_{12}}{\partial F_{11}} + \frac{\partial L}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial F_{11}} + \frac{\partial L}{\partial O_{22}} \cdot \frac{\partial O_{22}}{\partial F_{11}}$
$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} \cdot \frac{\partial O_{11}}{\partial F_{12}} + \frac{\partial L}{\partial O_{12}} \cdot \frac{\partial O_{12}}{\partial F_{12}} + \frac{\partial L}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial F_{12}} + \frac{\partial L}{\partial O_{22}} \cdot \frac{\partial O_{22}}{\partial F_{12}}$
Um die Formel weiter zu vereinfachen kann der lokale Gradient nach der Formel (Formel für O_1 einsetzen) berechnet werden. Die Gradienten für O_{11} sind daher: $\frac{\partial O_{11}}{\partial F_{11}} = X_{11} \quad \frac{\partial O_{11}}{\partial F_{12}} = X_{12} \quad \frac{\partial O_{11}}{\partial F_{21}} = X_{21} \quad \frac{\partial O_{11}}{\partial F_{22}} = X_{22}$ (somit lassen sich auch die Gradienten für O_{12}, O_{21} und O_{22} finden). Eingesetzt ergibt dies
$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} \cdot X_{11} + \frac{\partial L}{\partial O_{12}} \cdot X_{12} + \frac{\partial L}{\partial O_{21}} \cdot X_{21} + \frac{\partial L}{\partial O_{22}} \cdot X_{22}$
$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} \cdot X_{12} + \frac{\partial L}{\partial O_{12}} \cdot X_{13} + \frac{\partial L}{\partial O_{21}} \cdot X_{22} + \frac{\partial L}{\partial O_{22}} \cdot X_{23}$
Das diese Berechnung genau der einer Faltung entspricht muss eine Faltung auf die Eingabematrix mit der Gradientenmatrix wie folgt genutzt werden(bild cnn_ gradienten_ convolution einfügen).

Als nächstes muss noch der lokale Gradient nach der Eingebe X finden: $\frac{\partial L}{\partial X}$. Dieser lässt sich ähnlich $\frac{\partial O}{\partial F}$ aus der Formel $O_{11} = X_{11} \cdot F_{11} + X_{12} * F_{12} + X_{21} \cdot F_{21} + X_{22} \cdot F_{22}$ berechnen: $\frac{\partial O_{11}}{\partial X_{11}} = F_{11} \quad \frac{\partial O_{11}}{\partial X_{12}} = F_{12} \quad \frac{\partial O_{11}}{\partial X_{21}} = F_{21} \quad \frac{\partial O_{11}}{\partial X_{22}} = F_{22}$. Wird abermals die Kettenregel: $\frac{\partial L}{\partial X_i} = sum_{k=1}^{M} \frac{\partial L}{\partial O_k} \cdot \frac{\partial O_k}{\partial X_i}$ angewendet, so erhält

man folgende Formeln für die Gradienten der Beispielmatrix:

$$\frac{\partial L}{\partial X_{11}} = \frac{\partial L}{\partial O_{11}} \cdot F_{11} \tag{7}$$
$$\frac{\partial L}{\partial X_{12}} = \frac{\partial L}{\partial O_{11}} \cdot F_{12} + \frac{\partial L}{\partial O_{12}} \cdot F_{11} \tag{8}$$
$$\frac{\partial L}{\partial X_{13}} = \frac{\partial L}{\partial O_{12}} \cdot F_{12} \tag{9}$$
$$\frac{\partial L}{\partial X_{21}} = \frac{\partial L}{\partial O_{11}} \cdot F_{21} + \frac{\partial L}{\partial O_{21}} \cdot F_{11} \tag{10}$$
$$\frac{\partial L}{\partial X_{22}} = \frac{\partial L}{\partial O_{11}} \cdot F_{22} + \frac{\partial L}{\partial O_{12}} \cdot F_{21} + \frac{\partial L}{\partial O_{21}} \cdot F_{12} + \frac{\partial L}{\partial O_{22}} \cdot F_{11} \tag{11}$$
$$\frac{\partial L}{\partial X_{23}} = \frac{\partial L}{\partial O_{12}} \cdot F_{22} + \frac{\partial L}{\partial O_{22}} \cdot F_{12} \tag{12}$$
$$\frac{\partial L}{\partial X_{31}} = \frac{\partial L}{\partial O_{21}} \cdot F_{21} \tag{13}$$
$$\frac{\partial L}{\partial X_{32}} = \frac{\partial L}{\partial O_{21}} \cdot F_{22} + \frac{\partial L}{\partial O_{22}} \cdot F_{21} \tag{14}$$
$$\frac{\partial L}{\partial X_{33}} = \frac{\partial L}{\partial O_{22}} \cdot F_{22} \tag{15}$$

Auch diese Operation kann als Faltung dargestellt werden. Jedoch muss die Filtermatrix um 180°gedreht werden, um zusammen mit dem Fehlergradienten aus der nächsten Schicht ($\frac{\partial L}{\partial X}$) den lokalen Gradienten in Form einer Matrix zu berechnen. Hierbei wird nun zwischen dem gedrehten Filter und den Fehlergradienten eine ganze oder auch „full"Convolution durchgeführt. Das heißt, dass der Filter in dem Beispiel an der Position (-1|-1) ((0|0) = oben links) außerhalb des Fehlergradientenmatrix startet und beim ersten Schritt nur ein Feld des Fehlergradienten abdeckt. Wird nun die gleiche Schrittweite (1) genutzt, so besitzt der errechnete Gradient die gleichen Maße, wie die Eingabe der Schicht.
Zusammenfassend lassen sich die benötigten Vektoren wie folgt berechnen:

$$\frac{\partial L}{\partial F} = Convolution(InputX, Fehlergradient\frac{partialL}{\partial O}) \tag{16}$$
$$\frac{\partial L}{\partial X} = FullConvolution(180°gedrehterFilterF, Fehlergradient\frac{partialL}{\partial O}) \tag{17}$$
$$\tag{18}$$

Der errechnete Gradient $\frac{\partial L}{\partial F}$ weißt Ähnlichkeiten, wie der Gradient $\frac{\partial L}{\partial W}$ eines MLP. Unser Filter wird ebenfalls ähnlich angepasst: $F_{neu} = F_{alt} - \alpha \cdot \frac{\partial L}{\partial F}$.
Auch die optionalen Biaswerte, die für einen Filter gesetzt sein können, werden ähnlich angepasst: $\frac{\partial L}{\partial b} = \frac{1}{M} \cdot \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} \cdot \frac{\partial O_k}{\partial b}$. Da die partielle Ableitung $\frac{\partial O_k}{\partial b}$ immer 1 ist, erfolgt die Anpassung: $b_{neu} = b_{alt} - \alpha \cdot (\frac{1}{M} \cdot \sum_{k=1}^{M} \frac{\partial L}{\partial O_k})$

Der Lernparameter *e*

Für die Berechnung der Gewichte ist ein Lernparameter e oder η notwendig. Er gibt an, wie stark sich die Gewichte verändern. Ist der Lernparameter bei einem neuronalen Netz mit nur einem Layer 1, so passen sich die Gewichtungen vollkommen dem gewünschten Ergebnis der Eingabesatzes an. Dadurch prägt er sich nur diesen Eingabesatz ein und wird andere Eingabedaten nicht erkennen/fehlinterpretieren. Ist er 0, so lernt das neuronale Netz nicht, da er mit der Änderung der Gewichte und Bias multipliziert wird. Somit wird meist ein Wert zwischen 0 und 1, zum Beispiel 0.1 gewählt.

Generell gilt, ist der Lernparameter zu groß, kann es passieren, dass die Schritte zu groß sind und

so der optimale Punkt übersprungen wird. Es kann zu einem Hin- und Herspringen über dem Optimalpunkt kommen. Ist er allerdings zu klein, so beansprucht das Lernen viel Zeit und Daten. Außerdem kann es vorkommen, dass statt dem globalen nur ein lokaler Tiefpunkt gefunden wird und so in diesem „stecken“bleibt (siehe Abb: 5).[?]

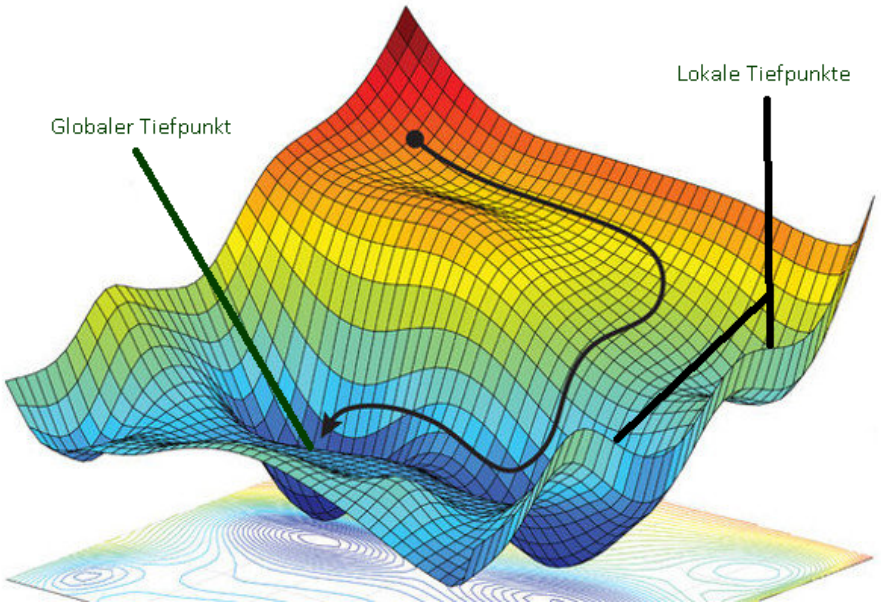

Abbildung 5: Das Gradientenabstiegsverfahren mit lokalen und globalen Tiefpunkten an einem Beispielgraph.[9]

Einsteinsche Summenkonvention

Die Einsteinsche Summenkonvention ist eine Konvention zu Notation mathematische Ausdrücke und stellt ein Indexschreibweise dar. Sie wurde 1916 von Albert Einstein eingeführt,von welchem sie auch ihren Namen erhielt. Bei dieser werden Summenzeichen weggelassen und stattdessen über doppelt auftretende Indizes summiert, was zu einer Verringerung des Schreibaufwandes und höheren Übersichtlichkeit führt. Darüber hinaus hebt diese die Symmetrie und Zusammenhänge hervor. In Numpy wird dies durch die Funktion „einsum()“implementiert und bietet so eine Vielzahl von mathematischen Operationen wie Skalarprodukt oder Kreuzprodukt.

10 Literaturverzeichnis

Gedruckte Literatur

[1] Dr. Narges Ahmidi. *KI im Gesundheitswesen, Österreichische Zeitschrift für das ärztliche Gutachten*. Manz Verlag, Wien, 2023.

[2] Dirk Hecker Gerhard Paaß. *Künstliche Inteligenz*. Springer Vieweg, Braunschweig/Wiesbaden, 2020.

[3] Jürgen Markl. *Markl Biologie*. Ernst Klett Verlag, Stuttgart, 2010.

[4] Tariq Rashid. *Neuronale Netze - selbst programieren*. dpunkt, Heidelberg, 2017.

Internetliteratur

[5] Dave Andre. Was ist eine aktivierungsfunktion? https://www.allaboutai.com/de-de/ki-glossar/was-ist-eine-aktivierungsfunktion/, Zugriffsdatum: 20.02.2023.

[6] David Beck. Warum KI nicht frei von Vorurteilen ist. https://www.tagesschau.de/wissen/ki-vorurteile-101.html, Zugriffsdatum: 13.05.2023.

[7] Eli Bendersky. Backpropagation through a fully-connected layer. https://eli.thegreenplace.net/2018/backpropagation-through-a-fully-connected-layer/, Zugriffsdatum: 13.02.2023.

[8] Jason Brownlee. How do convolutional layers work in deep learning neural networks? https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/, Zugriffsdatum: 28.12.2022.

[9] Non convex-optimization-We-utilize-stochastic-gradient-descent-to-find-a-local optimum. Alexander-amini. https://www.researchgate.net/profile/Alexander-Amini/publication/325142728/figure/fig1/AS:766109435326465@1559666131320/Non-convex-optimization-We-utilize-stochastic-gradient-des jpg, Zugriffsdatum: 07.05.2022.

[10] Klaus Dalhoff. Ambulant erworbene Pneumonie bei Erwachsenen. https://link.springer.com/article/10.1007/s11298-017-5989-y, Zugriffsdatum: 14.01.2023.

[11] Sanket Doshi. Various optimization algorithms for training neural network. https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6, Zugriffsdatum: 14.12.2023.

[12] Convolution filter-and-feature-map-Inspired-by-Karns-data-science-blog post. Jean-pierre briot.

[13] Vaibhav Haswani. Learning rate decay and methods in deep learning. https://medium.com/analytics-vidhya/learning-rate-decay-and-methods-in-deep-learning-2cee564f91 Zugriffsdatum: 25.02.2023.

[14] Nastasia Heilemann. Lungenentzündung. https://stiftung-gesundheitswissen.de/wissen/lungenentzuendung/diagnostik, Zugriffsdatum: 05.08.2023.

[15] Luise Heise. Röntgen-Thorax: Röntgenbild der Lunge. https://www.lungeninformationsdienst.de/diagnose/bildgebung/roentgen-thorax, Zugriffsdatum: 05.02.2023.

[16] Jefkine. Backpropagation in convolutional neural networks. https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/, Zugriffsdatum: 13.02.2023.

[17] Tomer Kordonsky. Loss function. https://medium.com/artificialis/loss-functions-361b2ad439a0, Zugriffsdatum: 07.05.2022.

[18] Mayank Mishra. Convolutional neural networks, explained. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939, Zugriffsdatum: 27.12.2022.

[19] CNN model using Tensorflow Keras. Ujjwal khandelwal. https://pythonandml.github.io/dlbook/content/convolutional_neural_networks/cnn_architecture.html, Zugriffsdatum: 22.11.2023.

[20] Paul Mooney. Chest X-Ray Images (Pneumonia). https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia, Zugriffsdatum: 09.08.2022.

[21] Julia Odenthal. Convolutional neural networks. https://www.mi.uni-koeln.de/wp-znikolic/wp-content/uploads/2019/06/11-Odenthal.pdf, Zugriffsdatum: 27.12.2022.

[22] Annalena Rüsche. Künstliche Intelligenz in der Medizin. https://www.mittelstand-nachrichten.de/gesundheit/kuenstliche-intelligenz-in-der-medizin/, Zugriffsdatum: 14.01.2023.

[23] Savyakhosla. Cnn | introduction to pooling layer. https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/, Zugriffsdatum: 03.01.2022.

[24] Saily Shah. Cost function is no rocket science! https://www.analyticsvidhya.com/blog/2021/02/cost-function-is-no-rocket-science/, Zugriffsdatum: 20.02.2023.

[25] Pavithra Solai. Convolutions and backpropagations. https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c, Zugriffsdatum: 11.02.2023.

[26] Max-Ludwig Stadler. Convolutional neural network (cnn). https://tinyurl.com/ytc78n5d, Zugriffsdatum: 26.12.2022.

[27] Anna-Lena von Hodenberg. Rassismus im Code: Wie KIs Vorurteile verstärken. https://hateaid.org/ki-rassismus/, Zugriffsdatum: 18.12.2023.

[28] Thomas Wood. Convolutional neural network. https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network, Zugriffsdatum: 27.12.2022.

[29] Harsh Yadav. Dropout in neural networks.
https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9, Zugriffsdatum: 03.01.2022.

[30] Dr. Daniel Ziemek. KI in der Medizin: Künstliche Intelligenz für die Gesundheit.
https://www.pfizer.de/newsroom/news-stories/ki-in-der-medizin-kÃijnstliche-intelligenz-fÃijr-die-g
Zugriffsdatum: 14.01.2023.

Eidesstattliche Erklärung

Wir erklären eidesstattlich, dass wir die Arbeit selbständig angefertigt, keine anderen als die angegebenen Hilfsmittel benutzt und alle aus ungedruckten Quellen, gedruckter Literatur oder aus dem Internet im Wortlaut oder im wesentlichen Inhalt übernommenen Formulierungen und Konzepte gemäß den Richtlinien wissenschaftlicher Arbeiten zitiert, durch Fußnoten gekennzeichnet oder mit genauer Quellenangabe kenntlich gemacht haben.

Nico Lentsch Dustin Marggraff Marvin Heyne

Erfurt, den __.__.____