ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Projektarbeit

Thema

Qualitätssicherung von Ultraschallbildern mithilfe von Deep-Learning Algorithmen

Autor: Philipp Bellia

Matrikelnummer: 3810321

Studiengang: Biomedizinische Technik

Abgabe Datum: 30.01.2023

Betreuer

Professor Dr. Robert Lemor

Inhalt

Einführung

Diese Projektarbeit soll einem einen Blick darauf verschaffen, welche Qualitätsmerkmale in Ultraschallbildern wichtig sind und wie man sie mit Hilfe von einem Deep-Learning Algorithmus kontrollieren kann.

Im Folgendem wir theoretisch darauf eingegangen, was ein neuronales Netz ist, aus welchen Bestandteilen es aufgebaut ist und welche Arten von neuronalen Netzen für die Aufgaben Sinn machen.

Der Schnittpunkt der beiden Themen wird den Mittelpunkt der Arbeit darstellen. Die Fragestellung wird sein, ob es mehrere Lösungen gibt die Problemstellung anzugehen.

Zum Schluss soll nach einem Fazit zu den Lösungsansätzen noch ein Eindruck gewonnen werden, was das Thema in naher Zukunft noch bereithält.

Im Anhang der Arbeit befinden sich Python Code Beispiele, die zur Verdeutlichung eines neuronalen Netzes dienen.

Grundlagen

Grundlagen zur Qualitätssicherung von Ultraschallbildern

Anforderungen

Die Ultraschall-Bildgebung ist die am meisten verwendete Modalität der medizinischen Bildebenen Verfahren. Dementsprechend gibt es viele Anforderungen an das Ultraschallgerät und die entstehenden Bilder, um eine bestimmte Qualität zu gewährleisten, um so eine bestmögliche Voraussetzung zu schaffen, eine korrekte Diagnose zu stellen.

Es gibt mehrere Faktoren die Einfluss auf die Bildqualität haben.

Unter diesen Faktoren sind zwei Gruppierungen, auf die man keinen Einfluss haben kann aus der Sicht des Algorithmus Entwicklers. Zum einen wäre das Patientenfaktoren wie beispielsweise die Patientenkonstitution (Adipositas oder ähnliches) oder aber auch die Erfahrung des praktizierenden Arztes.

Gerätespezifische Faktoren kann man zum einen in physische Faktoren des Gerätes und zum anderen in Geräteeinstellungen unterteilen.

Unter den physischen Eigenschaften fallen Gebrauchsspuren, Abnutzung des Schallkopfes oder aber auch Kristalldeffekte. Einstellungen, die man vornehmen kann, um das Bild zu beeinflussen wären Schallkopfeinstellungen und Bildeinstellungen. Unter Schallkopfeinstellungen fallen Sendefrequenzen, Sendefokus und Bildtiefe. Bildeinstellungen sind beispielsweise Gesamthelligkeit, Tiefenausgleich und Bildkontrasteinstellung. Vor allem letztere sind für neuronale Netzt von Interesse.

Um aber ein detailreiches Bild zu bekommen sind die Grundeinstellungen unumgänglich. Die Bildfeldtiefe beispielsweise, ist eine Einstellung, die im Grunde bestimmt, wo im Ultraschallbild das zu sehende Organ abgebildet wird.

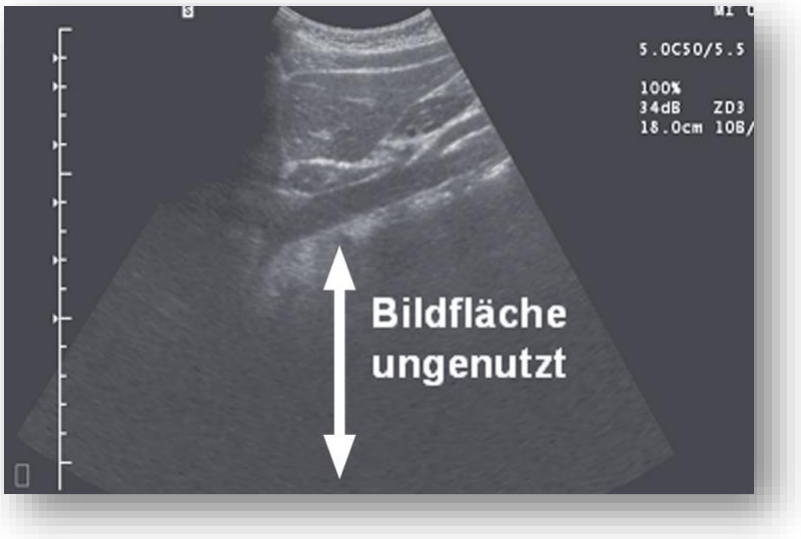

Abbildung 1: schlechte Einstellung Bildfeldtiefe am Beispiel linker Leberlappen Quelle: [2]

Abbildung 1 zeigt nun, dass durch eine schlecht gewählte Einstellung das Organ nicht das ganze Bild ausfüllt, sondern nur einen Bruchteil. Im Blick auf die Aufgabenstellung ist dies ein großer Nachteil, da dieses Problem sich auf das Lernen von neuronalen Netzen auswirkt. Im Gegensatz dazu folgt in Abbildung 2 ein gut eingestelltes Bild.

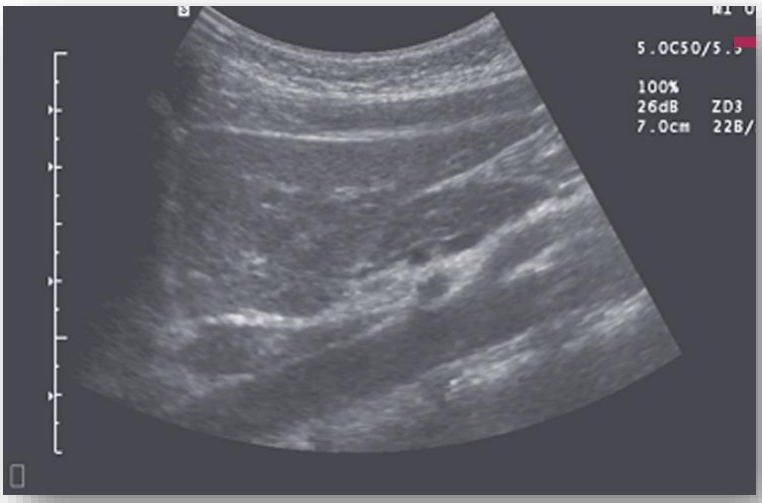

Abbildung 2: optimale Einstellung der Bildfeldtiefe am Beispiel linker Leberlappen Quelle: [2]

Man kann erkennen, dass nun der ganze Kegel gefüllt ist. Neben der Bildfeldtiefe spielt die Sendefrequenz ebenfalls eine große Rolle. Sie bestimmt, wie tief die Schallwellen in das Gewebe vordringen. Stellt man nun die Sendefrequenz sehr niedrig ein, gelangen die Ultraschallwellen tiefer in den Körper. Je höher man Sie einstellt, desto kürzer werden die Schallwellen.

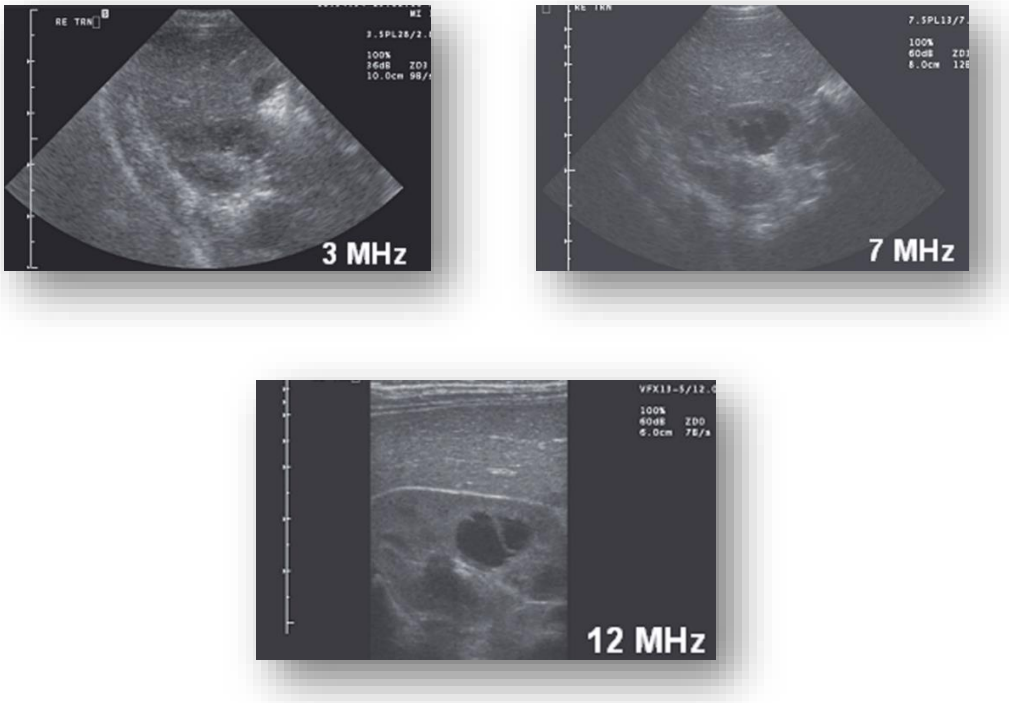

Abbildung 3: Sendefrequenzen aufsteigend dargestellt am Beispiel der Schilddrüse Quelle: [2]

Auf Abbildung 3 ist das oben beschriebene gut zu sehen. Die Schilddrüse ist ein Organ, welches nicht allzu weit im Körper liegt. Stellt man nun 3MHz als Sendefrequenz ein, dringen die Schallwellen zu weit in den Körper ein und reflektieren zudem das Gewebe, das hinter der Schilddrüse liegt. Bei den 7 MHz wird das Bild schon deutlicher, dennoch ist der Unterschied zu 12 MHz immer noch gut sichtbar. Zu erkennen ist auch, dass in der 12 MHz Abbildung ein linearer Schallkopf verwendet wird. Diese ist gut, um oberflächennahe Gewebestrukturen darzustellen. Einen Sektor-Scan macht dann Sinn, wenn große Organe darzustellen sind oder man einen Überblick über einen bestimmten Bereich haben will.

Auch muss auf die Kombination von Bildfeldtiefe und Sendefrequenz eingegangen werden. Sollten beide Einstellungen nicht aufeinander abgestimmt sein, entstehen ebenfalls schlechte Bilder. Es macht beispielsweise keinen Sinn eine hohe Sendefrequenz mit einer tiefen Bildfeldtiefe zu kombinieren.

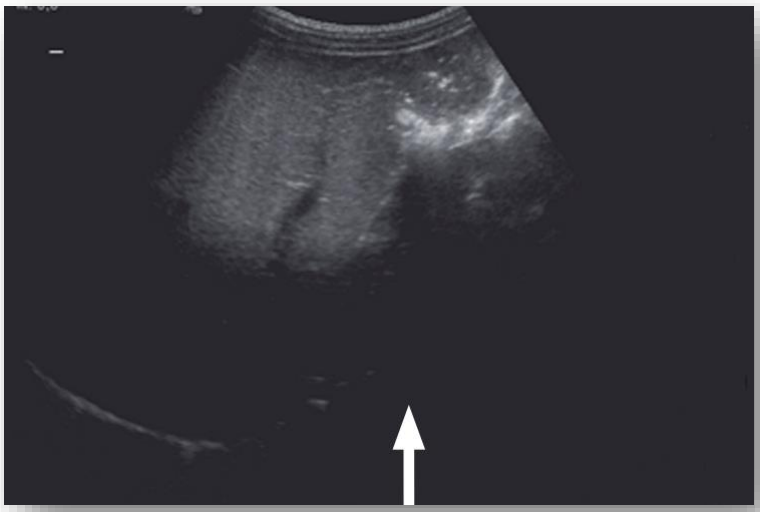

Abbildung 4: Kombination hohe Sendefrequenz und tiefe Bildfeldtiefe Quelle: [2]

Es ist zu erkennen, dass der untere Teil des Bildes schwarz ist, da die Sendefrequenz zu hoch für die Bildfeldtiefe eingestellt ist oder genau andersherum.

Auch hier wäre es ein großer Fehler, denn das neuronale Netz würde so falsche Schlüsse ziehen oder falsch aus dem Bild lernen.

Kontrast und Helligkeit sind ebenfalls Faktoren, die Einfluss auf den Lernvorgang des Algorithmus haben können. Auch im Zusammenspiel sind diese beiden Einstellungen ein großer Faktor, wenn es darum geht Details im Bild sichtbar zu machen.

Kontrast ist im Grunde ein Indikator für die Menge an Informationen im Bild. Je mehr Kontrast im Bild vorherrscht, desto mehr Geschehen ist zu sehen, spricht mehr Informationen sind auf dem Bild enthalten. Nun ist nur die Frage, ob es sich um relevante Informationen handelt. Im Bereich der Ultraschall-Bildgebung braucht der Mensch ein geschultes Auge, um diese Bilder zu interpretieren. Gleiches gilt für das neuronale Netz. Ist der Kontrast zu niedrig, wird es schwer Grenzübergänge zu erkennen oder korrekt zu interpretieren.

Die allgemeine Helligkeit im Bild und auch der Tiefenausgleich sind zum Kontrast ausschlaggebend, ob man das gesamte Bild gut sehen kann. Die Tiefenausgleich Einstellungen regulieren tiefenabhängig die Helligkeit im Bild.

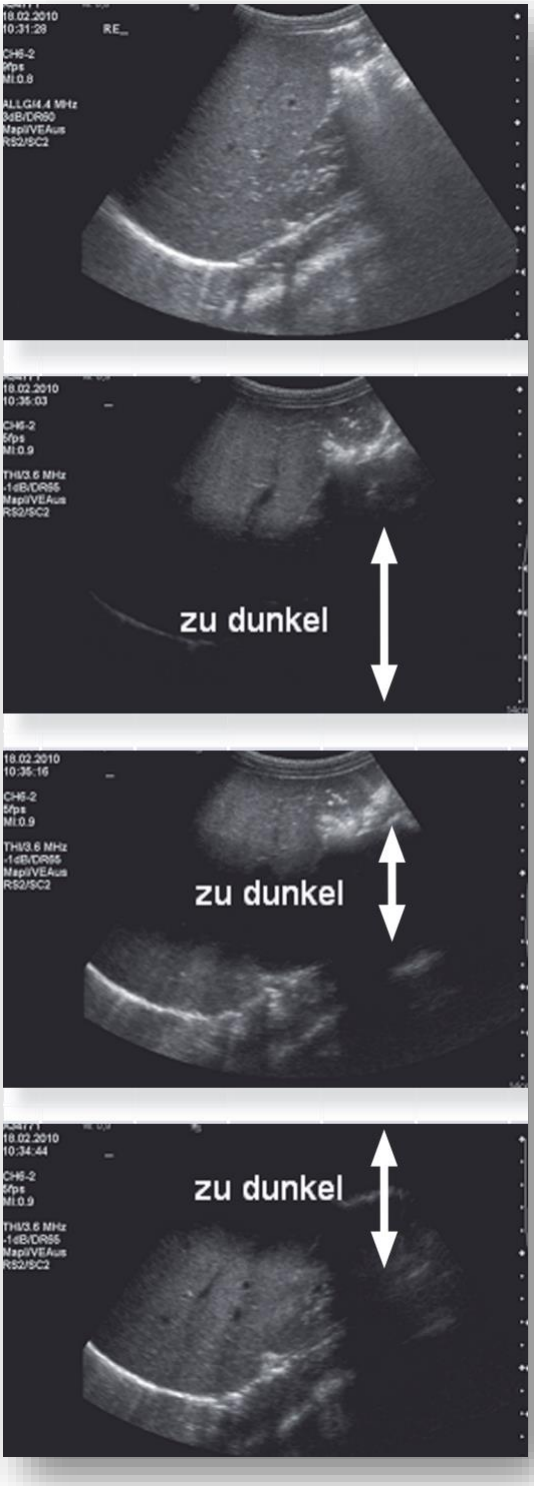

Abbildung 5: Tiefenausgleich im Ultraschall
Quelle: [2]

Auf der Abbildung 5 ist nun der Tiefenausgleich dargestellt. Das oberste Bild zeigt eine optimale Einstellung des Tiefenausgleiches.

Auf den folgenden Bildern ist immer zu sehen, dass ein Bereich des Ultraschallbildes schwarz bleibt, wenn der Tiefenausgleich nicht richtig eingestellt ist.

Alle genannten Einstellungen können das neuronale Netz bei seinem Lernvorgang beeinflussen. Es muss unbedingt genaustens ersichtlich sein, was das neuronale Netz erkennen soll. Ansonsten wird es zu Klassifikationsfehlern kommen.

Grundlagen eines Neuronalen Netzes

Künstliche Neuronen

Das künstliche Neuron ist das kleinste Element in einem neuronalen Netz. In der Informatik ist es kurz gesagt, ein Knotenpunkt, welcher einen Wert enthält. Im Falle, dass das Netz Bilder als Input erhält, wäre dieser Wert beispielsweise der Wert des Pixels.

Die Neuronen in neuronalen Netz sind miteinander verknüpft. Diese Verknüpfung wird durch eine Gewichtung realisiert. Diese Gewichtung ist eine zufällig generierte Zahl.

Der zuvor genannte Wert, welcher im Neuron steht, wird durch die Summer der Inputs multipliziert mit der Gewichtung dargestellt. Hinzu kommt der Bias (Verzerrung), welcher ebenfalls addiert wird: $\sum x_i \cdot w_i + b$ mit $x_i = Input\ Wert$ und $w_i = Wichtung$

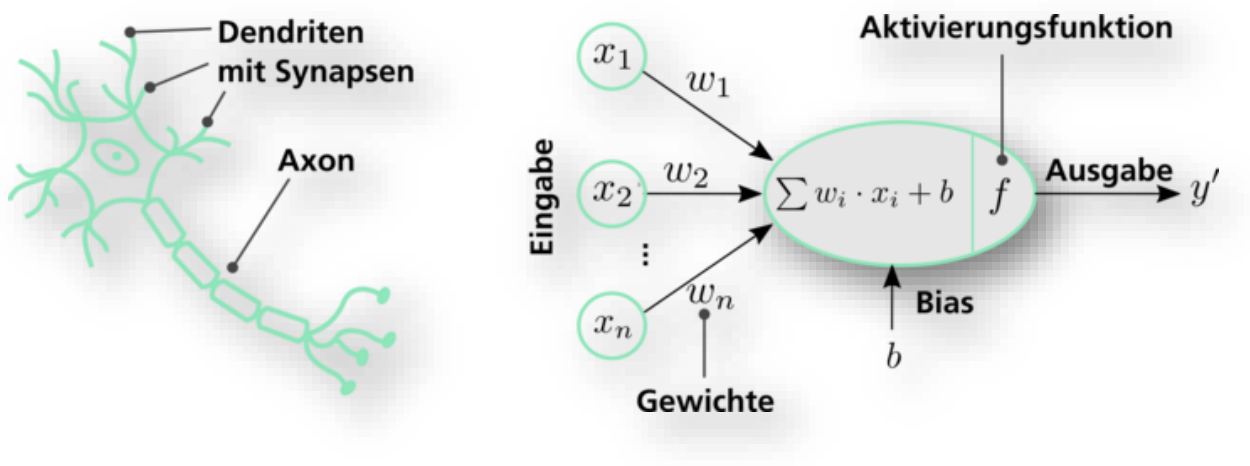

Abbildung 6: Gegenüberstellung Nervenzelle und künstliches Neuron Quelle: https://www.informatik-aktuell.de/betrieb/kuenstliche-intelligenz/neuronale-netze-ein-blick-in-die-black-box.html

Neben den Wert hat jedes Neuron eine Aktivierungsfunktion. Hier unterscheidet man zwischen:

- Linearer Funktion
- Lineare Funktion mit Grenzwert
- Binäre Funktion (Sprungfunktion)
- Logistische Funktion (Sigmoidfunktion)

Damit ein Neuron „feuert“, muss der Wert, den es enthält in die Aktivierungsfunktion fitten. Um dies klarer zu machen, ein kleines Beispiel:

Angenommen das Neuron hat als Aktivierungsfunktion eine binäre Funktion, die ab einem bestimmten Wert [a] = 1 ist. Sollte nun der Wert des Neurons größer oder gleich [a] sein, dann feuert das Neuron. Ist der Wert des Neurons kleiner [a], dann feuert das Neuron nicht und es bleibt in einem passiven Zustand.

Feuert das Neuron, so sendet es eine Ausgabe an das nächste Neuron, ebenfalls wieder mit einer zufälligen Gewichtung. Dieser Vorgang wird nun bis zum letzten Neuron fortgeführt.

Neuronales Netz

Ein neuronales Netzt, ist ein Netz, welches aus mehreren Ebenen von Neuronen besteht. Diese Ebenen sind miteinander verbunden.

Man spricht von drei Ebenen:

- Input-Ebene (Input-Layer)
- Versteckte Ebene (Hidden-Layer)
- Ausgangsebene (Output-Layer)

In der Input-Ebene wird das Netz mit Informationen gefüttert, die es in irgendeiner Form bearbeiten soll.

Die Hidden-Layer Ebenen die dem Anwender verborgen sind. Dies können beliebig viele Ebenen sein. Rechenleistung nimmt aber mit mehr Ebenen zu.

Der Output-Layer gibt nun das Ergebnis an.

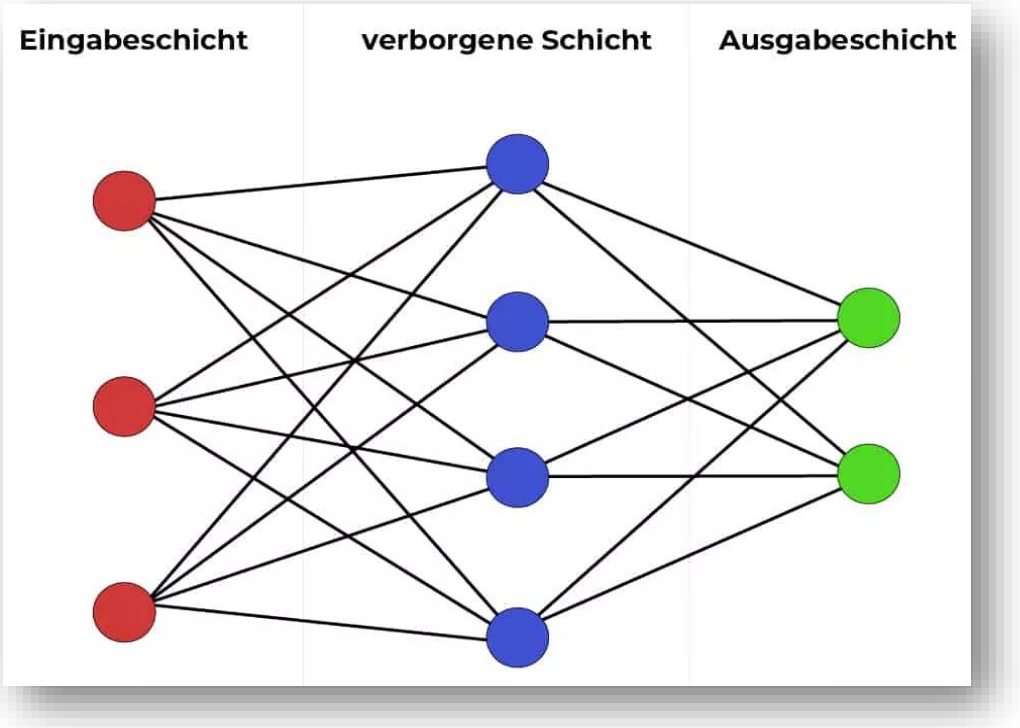

Abbildung 7: Ebenen eines neuronalen Netzes Quelle: https://datasolut.com/neuronale-netzwerke-einfuehrung/

Ebenen bestehen immer aus mehreren Neuronen. Oft ist es so, dass die Anzahl an Neuronen abnimmt, je weiter die Ebene am Output ist, dies muss aber nicht der Fall sein, denn dieser Aspekt ist immer noch Aufgaben spezifisch.

Deep-Learning

Unter einem Deep-Learning Netzwerk versteht man ein neuronales Netz, welchen viele Ebenen innerhalb des Hidden-Layer besitzt, deswegen „Deep“.

Die richtige Netzarchitektur

Es ist wichtig, dass man für seine Aufgabe die richtige Art des neuronalen Netzes auswählt. Man unterscheidet heutzutage grundlegen zwei Arten von neuronalen Netzen, die für sich, einen individuellen Nutzen haben. Es gibt zahlreiche Arten von anderen Netzwerken, im Rahmen dieser Arbeit wird aber nur auf diese beiden eingegangen.

- CNN (Convolutional Neural Network)
- RNN (Rekurrent neuronale Netzwerk)
- R- CNN (Region based Convolutional Neural Network)

Rekurrente neuronale Netze

Das RNN ist für Aufgaben entworfen, bei denen ein Gedächtnis oder ein Kontext benötigt wird. Beispiele für solche Aufgaben sind Spracherkennung, Übersetzung oder auch Wettervorhersagen oder Vorhersagen von Aktienkursen.

Die Funktionsweise der RNN ist bereits im Namen vorhanden. Eigenschaft dieser Netzwerke ist, dass es bei der Berechnung der Übertragungsfunktionen eine Rückkopplung zwischen zwei Neuronen gibt.

Eine spezielle Form der RNN sind die Long-Short-Term-Memory Netzwerke. Diese beinhalten sogenannte Memoryzellen, die es erlauben, Informationen kurzzeitig zu speichern. Eine bekannte Anwendung dieser Art von neuronalen Netzen ist die automatische Dialogführung von Google in ihrem Allo-System.

Hier lernt das Netz aus der Vergangenheit, wenn ein bestimmter Text schon einmal eingegeben wurde. Nun kann das Netzwerk den Kontext aus der Vergangenheit nutzen, um ein Vorschlag für Dialogoptionen zu geben.

Convolutional Neural Network

Die CNN sind für uns und unsere Aufgabe die interessanten Netzwerke. Das Wort Convolution wird übersetzt mit dem Wort Faltung, was in der Signalverarbeitung eine große Bedeutung hat. Die Faltung ist eine Operation, die es uns ermöglicht Signale, in unserem Fall Bilder, miteinander zu verrechnen.

Aufgabe des CNN ist es Klassifikationsaufgaben zu bewältigen, die sich vor allem um Bild und Videoerkennung drehen.

Wie jedes neuronale Netze besitz das CNN einen Input-Layer. Hier wird das Bild, welches klassifiziert werden soll, eingelesen. Anschließend kommt der sogenannte Convolution-Layer, auch Faltungs-Layer genannt. Hier wird auf das Eingabebild ein Filter angewandt, der sogenannte Feature Maps generiert. Feature Map lassen sich gut mit dem Begriff Merkmalsextraktion beschreiben. Anschließend wird auf die Feature Maps die Aktivierungsfunktion angewandt. Dies geschieht im Activation-Layer.

Nach dem Activation-Layer folgt der Pooling-Layer. Hier werden die Feature Maps so bearbeitet, dass die markantesten Stellen im Bild hervorgehoben werden. Dies geschieht über die Reduzierung der Dimension der Feature Maps. Hierbei wird der Maximalwert, in einem Kernel, der auf der Feature Map liegt, bestimmt. Um nun die Daten in klassifizierbare Objekte umzuwandeln, muss mit einem Flatten-Layer die Bildmatrix zu einer Liste umgewandelt werden. Zum Schluss folgen ein oder mehrere Dense-Layer oder auch Fully-Connected-Layer genannt. Diese sind für eine

Klassenzuordnung zuständig und geben die Klassifizierung an den Output-Layer weiter, der das Ergebnis liefert.

Der Aufbau von Convolution-Layer und Pooling-Layer, wird oft auch mehrmals hintereinandergeschaltet, um noch bessere Ergebnisse zu bekommen. Der Aufbau ist in Abbildung 3 zu sehen.

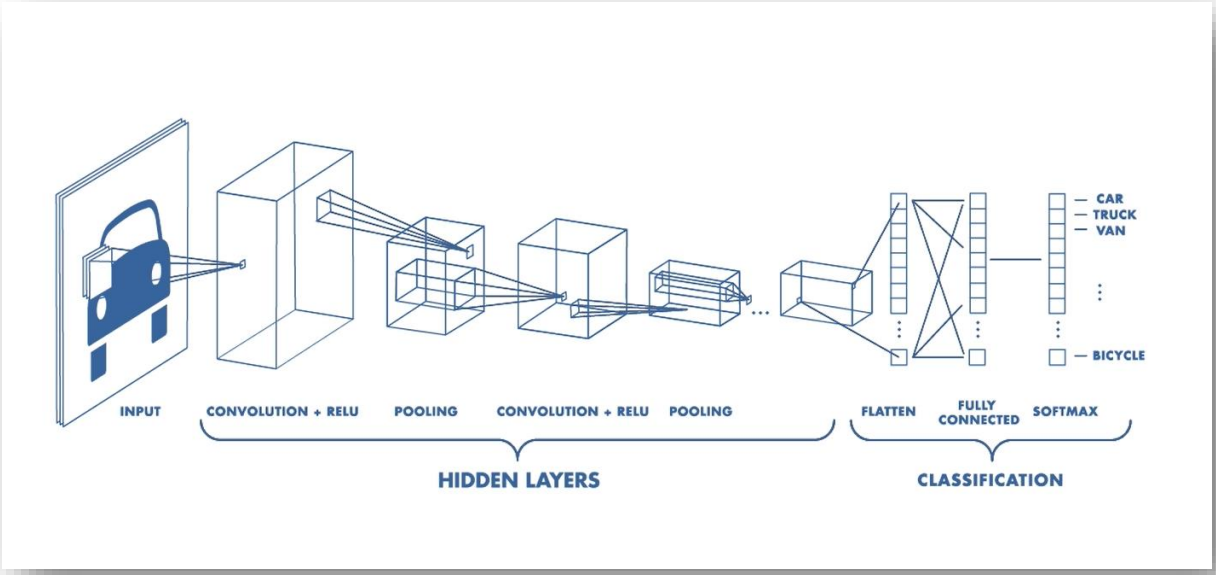

Abbildung 8: Aufbau eines Convolutional Neural Network Quelle: https://de.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html

R-CNN

CNNs können zwar klassifizieren, was auf einem Bild zu sehen ist, aber nicht, wo wir das Objekt auf dem Bild finden.

Die Rubrik der R-CNN ist bei dieser Aufgabe die Lösung.

Zuerst gilt es aber zu klären, was ein R-CNN überhaupt ist. Das Region basierte CNN ist wie der Name schon vermuten lässt ein CNN, welches mit Regionen arbeitet. Es führ die CNN-Klassifikation auf Regionen in unserem Bild aus. Dies bedeutet also, dass unser Bild, welches wir untersuchen wollen, in Regionen unterteilt wird und jede einzelne Region wird durch das interne CNN klassifiziert.

Wie man sich denken kann, ist dies sehr aufwendig. Zum einen, weiß das Netzwerk nicht, in wie viele Regionen man das Bild unterteilen soll. Unterteilt man es in 100 oder sogar 1000 Regionen so muss für jedes Bild 1000 Klassifikationen durchgeführt werden. Dies für einen großen Datensatz von beispielweise 80.000 Bildern würde die Zeit des Lernprozesses sprengen. Es wäre nicht sehr effizient. Um hierfür eine Lösung zu bieten, gibt es Verbesserungen des Systems:

- Fast R-CNN
- Faster R-CNN
- Mask R-CNN

Aktuell ist es so, dass die Netzwerke, in der Aufzählung, von oben nach unten immer schneller sind. Sprich das schnellste Netzwerk für Objektlokalisation ist momentan das Mask R-CNN.

Nun muss noch geklärt werden, wie die R-CNNs arbeiten und wie sie aufgebaut sind.

Bei den schnelleren Varianten des R-CNNs ist es so, dass zuerst das Bild durch den CNN-Prozess der Merkmalsextraktion läuft, bevor die Regionen bestimmt werden. Die entstehenden Feature Maps laufen dann im selben Moment durch den sogenannten „region proposal“ Prozess, der auf den Feature Maps Regionen bestimmt. Diese können nun unterschiedliche Größen besitzen, sprich sie haben keine einheitliche Form oder Größe. Allerdings immer rechteckig.

Wie bereits erklärt, haben Bilddaten, die in ein CNN eingespeist werden, eine einheitliche Größe. Das bedeutet, dass die Regionen, die aus den Feature Maps generiert werden, die wie erwähnt eine unterschiedliche Größe besitzen, normalisiert werden müssen. Dies geschieht mit der sogenannten ROI-Pooling Methode.

Folgendes wird bei der ROI-Pooling Methode gemacht:

1. Die Feature Map und die ROI (Region of Interest) werden übereinandergelegt.
2. Wir definieren eine Größe, beispielsweise 2x2 für die neue Feature Map
3. In der Region of Interest definieren wir nun vier Felder mit ähnlicher Größe
4. In jedem der vier Felder wird nun der Maximalwert bestimmt und in die resultierende 2x2 Matrix geschrieben

Dieser Schritt wird auch Region proposal Network genannt (RPN).

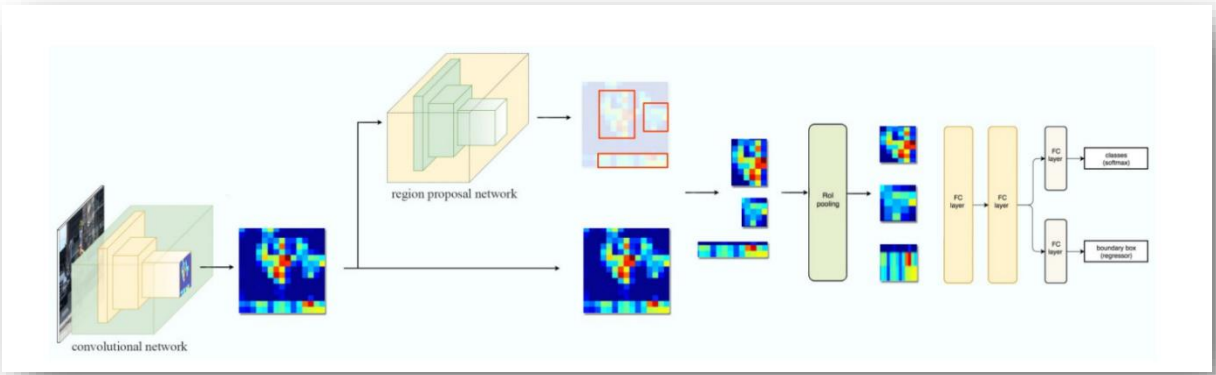

Abbildung 9: Schemata Faster R-CNN Quelle: https://jonathan-hui.medium.com/what-do-we-learn-from-region-based-object-detectors-faster-r-cnn-r-fcn-fpn-7e354377a7c9

Abbildung 4 zeigt nun den Aufbau als Schema. Um nochmal zusammen zu fassen, wie das Netzwerk funktioniert:

Das eingelesene Bild wird zuerst in ein CNN eingelesen und es werden Feature Maps erstellt. Diese Feature Maps werden nun in ein Region Proposal Network eingelesen. Dort entstehen Regionen auf den Feature Maps, die im nächsten Schritt im ROI-Pooling Layer auf ein Format gebracht werden, um sie anschließend durch zwei Schichten Fully Connected Layer zu geben. Nach den zwei Fully Connected Layer

folgen wiederum Fully Connected Layer, diesmal aber parallel, um zum einen die Klassifizierung auszugeben und zum anderen die Lokalisierungsbox zu platzieren.

Nun muss an dieser Stelle abgewogen werden ob es mehr Sinn macht ein CNN oder ein R-CNN zu verwenden. Beide Netzwerkarchitekturen sind für Klassifikationen gut geeignet.

Die Frage ist, ob man das gesamte Bild der Ultraschallaufnahme gleich gut zu erkennen sein muss oder ob es Passagen im Bild gibt, die vielleicht eine nicht so große Relevanz haben. Sollte dies der Fall sein, ist das R-CNN die bessere Wahl für die Aufgabe, wegen der örtlichen Auflösung. Gilt es aber, dass Bild als ein komplettes zu klassifizieren, so ist ein CNN völlig ausreichend und wird der Aufgabe auch gerecht.

Im weiteren Verlauf der Arbeit wird auf beide Varianten eingegangen.

Das Netzwerk

Die Daten

Der wichtigste Punkt, der schnell mal übergangen wird, wenn es darum geht ein neuronales Netz zu konstruieren ist, den Datensatz, der zur Verfügung steht, richtig zu untersuchen und darauf zu achten, dass die Bilder im Datensatz sinnvoll zugeschnitten sind.

Es kann zu mehreren Problemen kommen, wenn der Datensatz nicht richtig durchdacht ist. Die Daten müssen ordentlich annotiert sein. Falls die genannten Punkte nicht gegeben sind, muss es gar nicht an der Struktur des Netzwerkes liegen, das die KI eine falsche Einschätzung abgibt. Die kommt nun daher, dass das Netzwerk einfach falsches aus den Daten gelernt hat.

Zudem sollte es zu so wenig Korrelation der Bilder kommen wie möglich. Sonst kann das Netzwerk nur Daten richtig verarbeiten, die aus dem Dataset stammen und nicht völlig fremd sind. Dies nennt man auch *Overfitting.* Overfitting bedeutet, dass das Netzwerk die Daten aus dem Datensatz zu gut gelernt hat, nun aber nur diese auch wieder erkennt. Fremde Bilder kann es nicht zuordnen.

Bei einem guten Datensatz, der für eine Bildklassifikationsaufgabe bereitgestellt wird, sollten die Bilder zuerst geordnet und gut annotiert sein. Dies nur, damit der Programmierer die Daten gut sichten kann. Im Netzwerk selbst, wird der Datensatz zufällig gemischt und aufgeteilt. Zufällig, damit alles so unabhängig wie möglich ist und aufgeteilt, damit man mit dem einem Teil das Netzwerk trainieren und mit dem anderen es testen kann.

Im Idealfall kann das Netzwerk nun auch völlig fremde Bilder lesen und klassifizieren.

Um ein geeignetes Datenset zu finden, welches die vorliegende Klassifikation Aufgabe gut wiedergibt, muss man darauf achten, was genau erkannt werden soll.

Es wäre nicht von Vorteil einfach einen Datensatz für Ultraschall auszuwählen, da wir vorerst nur Objekterkennung machen wollen. Ideal wäre ein Datensatz, den man selbst aufnimmt, optimiert und anpassen kann. Ultraschallphantome sind hier sehr gut geeignet. Hat man nun keinen eigenen Datensatz zur Verfügung, gibt es dennoch sehr zahlreiche Möglichkeiten fündig zu werden. Webseiten wie „Kaggle“ zum Beispiel, bieten schon lange Datensätze kostenlos an, um mit ihnen Aufgaben im Bereich künstliche Intelligenz zu lösen.

Da es um Strukturerkennung geht empfiehlt sich ein Dataset, um geometrische Strukturen zu klassifizieren. Zusätzlich sollte man bei der Auswahl darauf achten, dass Ultraschallbilder Graustufenbilder sind und oft keine scharfen Übergänge besitzen. Zudem gibt es noch den Punkt Kontrast, den man nicht vernachlässigen sollte, da der Kontrast bei Ultraschall nicht besonders auffällig ist.

In den Grundlagen zur Qualitätssicherung von Ultraschallbildern wurde schon auf wichtige Bildeinstellungen eingegangen, die hier eine wichtige Rolle spielen.

Ein Blick auf die zu klassifizierenden Bilder

Um nun noch eine Vorstellung von den Daten zu bekommen, die das neuronale Netz klassifizieren soll, wird nun im Anschluss eine Bildanalyse folgen, um wichtige Aspekte zu erschließen, die von Relevanz sind.

Wie bereits erwähnt, soll zunächst mal ein Phantombild analysiert werden, auf dem Kreise zu lokalisieren sind.

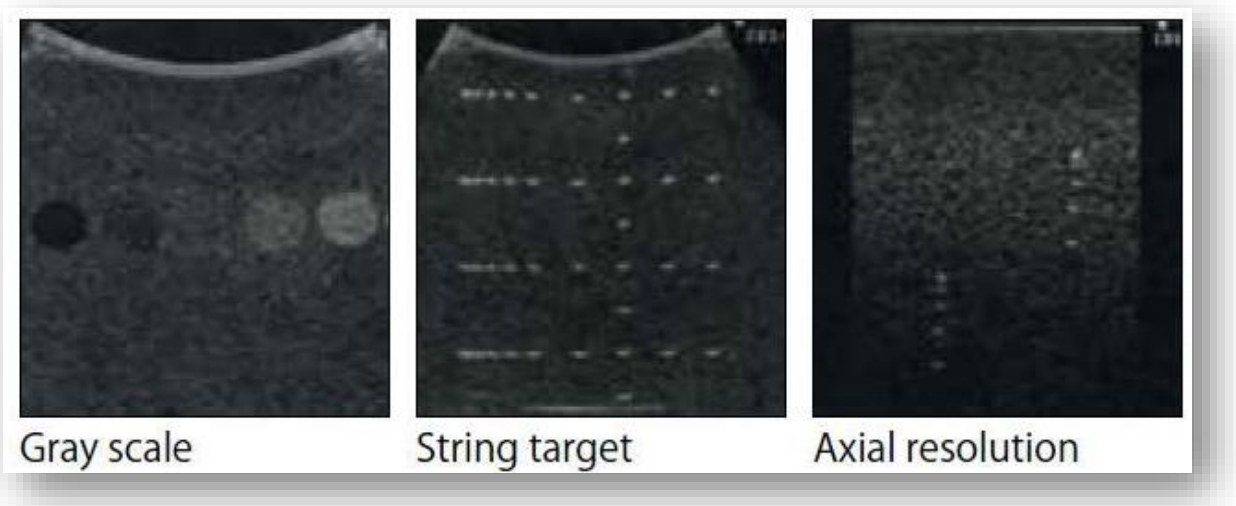

Abbildung 10: Ultraschall Phantombilder Quelle: https://www.skills-med.de/ultraschall-qa-phantom-multi

In Abbildung 5 sind nun solche Ultraschallphantombilder zu erkennen. Die Auflösung der Bilder lässt es jetzt nicht ganz zu, genauere Betrachtungen zu machen, aber dennoch wird das Grundkonzept der Bilder, die im Datensatz vorkommen klar.

Graustufen, horizontale sowie vertikale und Bildqualität mit niedriger Schallfrequenz sind vertreten. Im Ultraschallbild stellt die Graustufe visuell schon den Kontrast komplett dar, da sowieso keine Farben dargestellt werden. Das mittlere Bild soll zeigen, wie gut man von links nach rechts die Punkte voneinander unterscheiden kann oder ob sie, wenn sie zu nahe beieinander sind, als ein Objekt klassifiziert werden. Auf dem rechten Bild ist das gleiche nur in Abhängigkeit der Tiefe des Ultraschalls dargestellt. Diese Varianten von Bildern sollten in so variationsreich wie möglich im Datensatz vorhanden sein, um das Modell zu trainieren.

Im Nachhinein, wenn man den ersten Lernprozess durchgeführt wurde, wird dem Modell ein Bild gezeigt, auf dem es gelerntes quasi anwenden muss. Folgendes Bild, soll ein solches darstellen.

Abbildung 11: Phantombild mit mehr Variation zum überprüfen des Modells Quelle: https://www.cirsinc.com/products/ultrasound/zerdine-hydrogel/multi-purpose-multi-tisse-ultrasound-phantom/

Betrachtet man nun Abbildung 6, stellt man fest, dass abgesehen von dem Ultraschallbild noch Legenden und Parameter dargestellt werden. Nun muss man beachten, dass solche Objekte bei dem Lernprozess des Modells mitgelernt werden. Dies ist eine potenzielle Fehlerquelle, die vermeidbar ist. Dies bedeutet, dass Legenden und Parameter im ersten Moment störend sind. Im gleichen Zug wird dies aber nicht vermeidbar sein, denn in realen Anwendungssituationen sind diese Parameter und Legenden immer vorhanden. Es ist aber auch möglich, dass in realen Situationen die künstliche Intelligenz im Hintergrund die Rohdaten bekommt, wobei die Legenden noch nicht Teil des Bildes sind.

Man muss also aufpassen, wenn man im Schritt Dataset-Vorbereitung die Bilder anpasst, sprich zuschneidet und verkleinert, ob diese Legenden und Parameteranzeigen vorher zu entfernen sind oder ob man sie dabei lässt. Hierbei kommt es am Ende auf den richtigen Anwendungsfall an.

Ein weiterer Punkt, der beachtet werden muss, ist ob die Bilder, die untersucht werden sollen, mit unterschiedlichen Schallköpfen aufgenommen werden. Abbildung 4 und 5 zeigen Bilder mit einem konkaven Schallkopf. Sollte der Fall eintreten, dass beispielsweise eine Bilderserie mit einem linearen Schallkopf aufgenommen wird, ändert sich im Grunde die Form des Ultraschallbildes.

Abbildung 12: Schallkopfbeispiele, Ausbreitung des Schalls Quelle: https://www.unispital-basel.ch/fileadmin/unispitalbaselch/Departemente/DKTT/Angiologie/Fortbildungen/Duplexsonografie19/DE/1.1_B-Mode_Grundlagen.pdf

Die Form ist ebenfalls ein wichtiger Punkt, den es zu beachten gilt. Je nachdem, wie die Bilder bearbeitet werden, ist die Form noch zu erkennen und da an den Grenzen ein hoher Kontrast vorliegt, wird diese mit hoher Wahrscheinlichkeit mitgelernt.

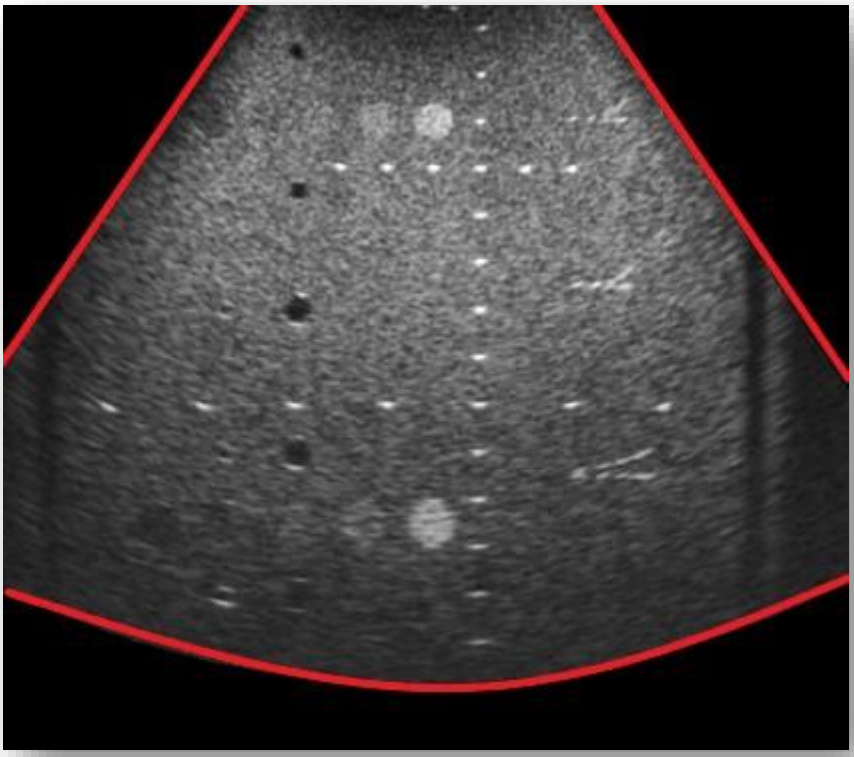

Abbildung 13: Zurechtgeschnittenes Bild mit roter Markierung

Abbildung 8 soll nun verdeutlichen, dass die Form des Ultraschallkopfes eine Rolle spielt. Die roten Kanten sollen die Übergänge des Bildes markieren und zeigen, dass die Ränder sehr präsent sind. Sollte man nun das Bild zuschneiden wollen, so würde an den Rändern Informationen verloren gehen. Besser geeignet zum Zuschneiden wären deswegen Bilder, die mit einem linearen Schallkopf aufgenommen werden.

Im Blick auf die Struktur des Datensatzes wäre es sinnvoll, Bilder aufzunehmen und diese in ihre Qualität zu unterteilen. Soll bedeuten, dass man sie in Klasseneinteilt, die

die Qualitätsunterschiede trennt. Diese kann man dann auch wieder verwenden, wenn es um die Ausgabe des Netzwerkes geht.

Aufbau des Netzes

Der Aufbau des Netzwerkes ist, wie bereits besprochen ein Faster R-CNN (Faster Region based Convolutional Network) oder eben ein CNN.

Künstliche Intelligenz und Python

Um nun das Netzwerk auch in Python-Code zu realisieren, benötigt man ein paar Bibliotheken, die nicht vorinstalliert sind.

Um in Python ein neuronales Netz zu programmieren, gibt es zwei Bibliotheken, die heutzutage nicht weg zu denken sind. Zum einen ist das die von Google erstellte Bibliothek Tensorflow zum anderen die von Facebook erstelle Bibliothek PyTorch.

In dieser Projektarbeit wird mit Googles Tensorflow-Bibliothek gearbeitet. Hier nutzt man, um übersichtlich zu bleiben, die High-Level API Keras, die bereits in Tensorflow implementiert ist. Mit Keras ist es besonders Benutzerfreundlich ein Netzwerk aufzubauen.

Neben Tensorflow nutzen wir die Bibliothek Open-CV, um Bilder einzulesen und darzustellen. Diese vor allem, um das Dataset vorzubereiten. Zusätzlich hierzu auch Numpy, die es einem ermöglicht, Mathematische Operationen an Arrays auszuführen.

Die Umgebung, in der das Netzwerk geschrieben wird, ist nicht so wichtig, es sollte nur die aktuelle Python Version unterstützt werden. Eine große Empfehlung ist das Google Collaboratory Notebook. Dies ist eine Jupyter-Notebook ähnliche Umgebung, die es einem ermöglicht, Zeilen von Code in Zellen zu schreiben und diese Zellen separat auszuführen. Dies ist vergleichbar mit dem MatLab live-Skript.

Es bringt aber den Vorteil, alles bereit installiert zu haben. Es hat die neuste Python Version mit allen nötigen Bibliotheken vorinstalliert. Dies bedeutet, man muss sich erst gar nicht Python herunterladen. Zudem kommt, dass man auf externen Servern von Google, die Netzwerke trainieren kann. Um mit dem eigenem Computer neuronale Netze zu trainieren, benötigt es eine große Rechenleistung der Grafikkarte. Sollte man nun auf einem normalen Laptop, ohne starke Grafikkarte arbeiten, so wie es auch in diesem Projekt der Fall ist, ist das Google Collaboratory eine sehr gute Alternative die Aufgabe anzugehen.

Nach der Trainingsphase kann man die Parameter einfach speichern und herunterladen. Klassifikationen können dann auch ohne Probleme auf dem eigenem Computer durchgeführt werden. Der Aufwendige Teil für den Computer ist nämlich das Training des Netzwerkes.

Theoretische Herangehensweise

Um nun den ersten Ansatz zu liefern, wie die Aufgabe mit einem CNN oder R-CNN angegangen wird, nutzen wir Pseudo-Code, um es anschaulicher zu machen. Der richtige Code wird am Ende der Arbeit hinzugefügt. Dennoch wird vorerst nur theoretisch an der Problemstellung gearbeitet.

Bearbeitung der Rohdaten

Der erste Schritt wird sein, die Daten, die zum Trainieren verwendet werden zu bearbeiten. Üblicherweise werden neuronalen Netzen keine Bilder zum Trainieren gegeben, die ein hohe Auflösung besitzen. Dennoch benötigen wir sie, da wir die Bilder zuerst auf ein passendes Format verkleinern werden. Das bedeutet, dass wir aus einem beispielsweise 500x500 Bild ein 64x64 Bild machen. Durch diesen Schritt gehen uns viele Informationen verloren. Die folgende Abbildung soll dies verdeutlichen.

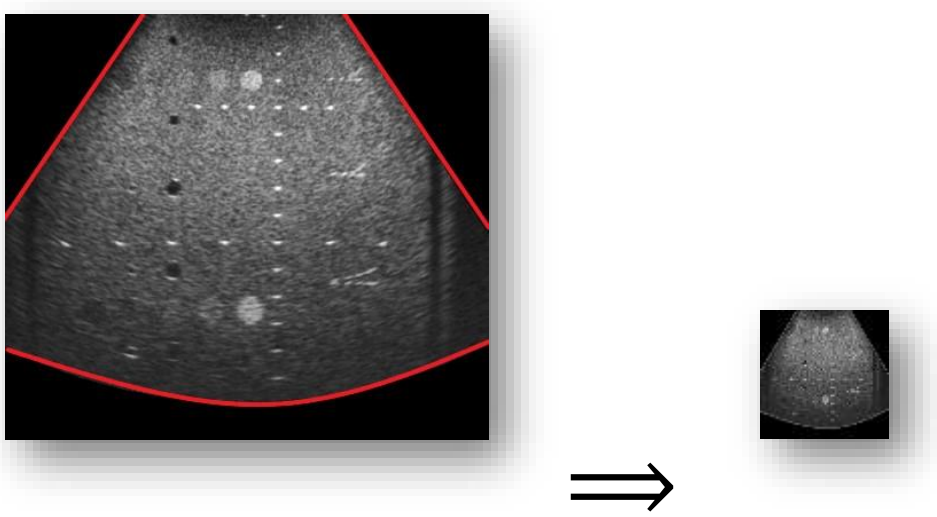

Abbildung 14: Beispiel für Verkleinerung des Bildes Abbildung 11 von 422x372 auf 64x64 Pixel

Der rote Rahmen geht hier im Beispiel verloren, da im Verkleinerungsprozess das Bild als Graustufenbild eingelesen wird. Bei genauem Hinsehen kann man erkennen, dass im verkleinerten Bild immer noch Strukturen des Phantoms enthalten sind. Mit weiteren Bildern, auf denen andere Strukturen besser zu erkennen sind, kann ein gutes Ergebnis bei der Klassifikation gelingen. Nun ist die Frage, ob man die Ränder versucht zu eliminieren, indem man simpler Weise einfach hereinzoomen würde. Macht man dies auf verschiedenen Positionen im Bild, so hätte man mehr Daten für das Datenpacket.

Abbildung 15: verkleinertes Bild aus Abbildung 14 herangezoomt

Das Heranzoomen führ wieder dazu, dass das Bild in seiner Größe angepasst werden muss. Es kommt wieder zu Informationsverlust, da man logischerweise keine Informationen aus dem Nichts erschaffen kann. Dies ist in der theoretischen Ausarbeitung aber von keiner großen Relevanz, da in einem echten Szenario die Rohbilder eine bessere Auflösung hätten.

Der Prozess, der hier auf das Bild angewandt wird, muss auf alle Bilder im Dataset angewandt werden. Ist der Schritt getan, muss der Datensatz noch normalisiert werden. Da alle Bilder, Graustufenbilder sind, teilen wird alle Pixel in einem Bild durch 255. Damit haben die Pixel alle einen Wert zwischen Null und Eins. Im Anschluss hat man ein Datenpacket, welches nun an das Netzwerk weitergegeben werden kann.

Für ein Mask RCNN muss deutlich mehr Aufwand gebracht werden. Hierbei muss darauf geachtet werden, dass alle Bilder sorgsam annotiert werden. Für die verwendeten Bilder müssen innerhalb der Klassen auch noch die Masken händig eingetragen werden, damit das System sie mit dem selbst produzierten Masken vergleichen kann, um so den Verlust klein zu halten.

Lösung mit CNN

Zu Beginn muss einem klar sein, welche Bausteine man benötigt, um das gewollte neuronale Netz zu erstellen. Grundlegen werden Input-Layer, Convolution-Layer, Pooling-Layer verwendet, um das Grundgerüst zu erstellen.

Pseudocode:

1. Modell Initialisieren (Sequenziell)
    1. Die Layer in der richtigen Reihenfolge hinzufügen
        1. Input-Layer aus 64 Neuronen
        2. Convolution-Layer mit Anzahl der Kernels
        3. Maxpooling-Layer mit der Pool-Größe (2,2)
        4. Convolution-Layer gleiche Anzahl wie vorher
        5. Maxpooling-Layer mit Pool-Größe (2,2)
        6. Dense-Layer mit 128 Neuronen
        7. Dense-Layer mit Neuronen = Klassenanzahl

Im Pseudocode wird nun wie auch in der Einführung das Grundgerüst erstellt. Nach der Initialisierung muss das Modell kompiliert werden. Hierzu folgt wieder Pseudocode:

1. Modell kompilieren
    a. Parameter 1: Optimizer
    b. Parameter 2: Verlustfunktion
    c. Parameter 3: dargestellte Messdaten

Die Parameter Optimizer und Verlustfunktion haben aktiv Einfluss darauf, wie das Training des Netzwerkes abläuft. Einfach dargestellt, hat der Optimizer die Aufgabe, die Berechnungen der Wichtungen der Neurone zu optimieren. Damit hat er auch direkten Einfluss auf die Verlustfunktion des Modells.

Die Verlustfunktion berechnet die Differenz zwischen erwarteten Aussagen und tatsächlichen aussagen des neuronalen Netzwerkes. Ziel ist es die Differenz so klein wie möglich zu bekommen damit die Erwartung schlussendlich das Ergebnis der Klassifizierung darstellen. Man unterscheidet verschiedene Verlustfunktionen. Es gibt die quadratische Fehlerfunktion, die für Regressionsprobleme verwendet wird und die Kreuzentropiefunktion, die für Klassifikationsaufgaben benötigt wird.

In unserem Fall verwenden wir die Kreuzentropiefunktion, dennoch wird im Quellcode die sogenannte „sparse_categorical_crossentropy“ verwendet, da wir nach Kategorien klassifizieren möchten und denen einen bestimmten Namen geben. Ohne die „sparse_categorical_crossentropy“ müsste man die Ergebnisse normieren und im Nachhinein den Klassen Labes geben. Diesen Schritt sparen wir uns an der Stelle.

Nach dem Kompilieren muss angegeben werden, welches Datenpacket das Trainingsset darstellt und welches zur Validierung dient. Zudem werden weitere Parameter definiert.

1. Modell „fitten“
    a. X als Trainingsset angeben
    b. y als Validierungsset angeben
    c. Batchsize definieren
    d. Trainingsepochen definieren

Umgangssprachlich spricht man von „das Modell fitten“. In diesem Schritt beginnt das Training des Modells. Zuerst wird ein Trainingsdatensatz angegeben. Danach wird ein Datensatz angegeben, der zur Validierung dient, sprich das trainierte nochmals validiert. Die Batchsize ist ein Parameter, der angibt, wie viele Bilder wir pro Schritt in einer Epoche verwenden wollen. Als letzter angegebener Parameter definieren wir noch, wie viele Epochen an Trainingsdurchläufen wir durführen möchten. Es ist wichtig abwägen zu können, wie viele Epochen Sinn machen, denn an diesem Punkt kann es schnell zu Overfitting kommen.

Ist das Training nun abgeschlossen, so kann das neuronale Netzwerk verwendet werden. Im optimalen Fall sollte es nun in der Lage sein, Datensatz unabhängig Bilder zu klassifizieren.

Sollte der Fall eintreten, dass es falsche Klassifizierungen abgibt, so kann ein zu geringes Training, sprich zu wenig Epochen oder zu kleine Batchsize der Grund sein. Es ist üblich, dass man an diesen Parametern schrauben muss, bis ein zufriedenstellendes Ergebnis zustande kommt.

Der schlimmste Fall wäre, wenn das Netzwerk keinerlei richtige oder nur sehr wenig richtige Klassifikationen abgibt. Overfitting ist nun das Stichwort und man muss nochmal alles durchgehen. Es gibt aber auch Anpassungen, die man einbauen kann, ohne den Datensatz anzufassen. Layer, die einem bei diesem Problem Abhilfe schaffen können heißen „Dropout-Layer“. Die Dropout-Layer, lassen immer, wenn sie auftreten, Neuronen fallen. Das bedeutet, dass nach dem Zufallsprinzip, innerhalb dieses Layers, Neuronen einfach nicht mehr weiterleiten. Damit kann einem Overfitting

vorgebeugt werden. Dropout-Layer werden nach einem Maxpooling-Layer in das Modell eingebaut.

Im Blick auf den richtigen Code ist zu sagen, dass mit dem Beispieldataset MNIST programmiert wurde. Es soll nur zur Veranschaulichung des Netzwerkes dienen und nicht schon Ultraschallbilder klassifizieren, da mit medizinischen Daten aus dem Internet vorsichtig umzugehen ist. Der Code ist im Anhang zu finden.

Lösung mit Mask R-CNN

Es ist zu beachten, dass ein Datensatz, der für ein Mask RCNN gedacht ist, anders aufbereitet werden muss als bei einem einfachem CNN. Es müssen für die Masken und die Regionen gut annotierte Daten vorhanden sein, damit das Netzwerk Vergleichswerte hat um das Training zu verbessern. Die Loss-Funktionen werden die Annotierten Daten brauchen, um die Verlustkurve aufzustellen.

In diesem Schritt wird die Aufgabe mit einem bereits vorgefertigten Mask R-CNN bearbeitet, da die Implementierung eines solchen Netzwerkes von Hand zu Fuß den Rahmen dieser Projektarbeit sprengen würde. Das verwendete Mask R-CNN wird von Matterport auf GitHub zu Verfügung gestellt. Es wird einem ermöglicht, bereits trainierte Netzwerke zu verwenden oder aber auch das Netzwerk auf ein eigenes Dataset zu trainieren. Zudem besteht die Möglichkeit, mehrere Datenpakete einzubinden, falls mehrere Datensätze zur Verfügung stehen und von Relevanz sind. (Quelle: GitHub Mask R-CNN)

Um das Netzwerk nun zu erstellen, wird empfohlen, dass man eine leistungsstarke GPU – Einheit besitzt. Um dieses Problem zu umgehen, nutzen wir, wie bereits erwähnt das Google – Collaboratory Notebook. Wir müssen daran denken, das von GitHub geklonte Repository in das Notebook hochzuladen.

Der Aufbau des kompletten Netzes wir im Folgendem in Pseudocode vereinfacht dargestellt:

1. Backbone erstellen (ResNet50)
2. Region proposal Network aufbauen
    a. Convolutional Layer
    b. Batchnormalization Layer
    c. Convolutional Layer
    d. Batchnormalization Layer
    e. Convolutional Layer
    f. Batchnormalization Layer
3. Erstellen Roi-Align Schicht
    a. Über die API Keras
4. Erstellen einer Klassifikationsvorhersage
    a. Dense Layer
    b. Batchnormalization Layer
    c. Dropout Layer
    d. Dense Layer

5. Erstellen der Maskenvorhersage
    a. Convolutional Layer
    b. Batchnormalization Layer
    c. Convolutional Layer
    d. Batchnormalization Layer
    e. Convolutional Layer
    f. Batchnormalization Layer
6. Erstellen der Loss-Funktionen über alle vorher beschriebenen Vorgänge
    a. Klassifikationsverluste
    b. Verluste über Regionen-proposal Rasterkoordinaten
    c. Verluste über vorgeschlagene Regionen
    d. Maskenverluste über vorgeschlagene Regionen

Das am Anfang vorkommende CNN wird umgangssprachlich *„Backbone“* genannt. Das CNN wird als ResNet50 realisiert. ResNet Netzwerke sind spezielle CNNs, die vor allem damit herausstechen, dass sie sehr tiefe verborgene Schichten besitzen. Normalerweise tritt bei CNNs ein Problem auf, wenn die verborgene Schicht, zu tief ist, sprich aus zu vielen Schichten besteht. Das Problem ist, dass der Gradient, der im Verlaufe des Lernens berechnet wird, verschwindet. Wieder spricht man hier von Overfitting. Um dem ganzen vorzubeugen, haben die Entwickler der ResNet Architektur eine Lösung entworfen, welche es ermöglicht, den Gradienten nicht verschwinden zu lassen und noch bessere Ergebnisse zu erreichen als vorher. Die Lösung, ist der sogenannte *„Identity-Block“*. Hierbei wird nach jedem zweiten oder jedem dritten Layer, die sogenannte Identität, in den Neuronen hinzuaddiert.

Wenn die mathematische Funktion eines Neurons ohne Identity-Block so aussieht:

$$y = F(x, \{W_i\})$$

So würde die Funktion mit Identity ein vorheriges [x] hinzuaddieren:

$$y = F(x, \{W_i\}) + x$$

Visuell dargestellt sieht der Identity-Block wie folgt aus:

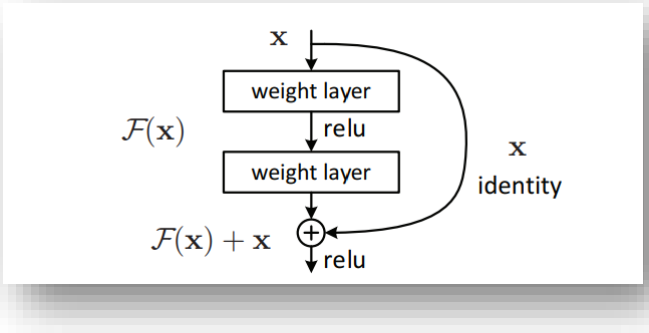

Abbildung 16: Identity-Block im ResNet Vgl: Quelle ResNet

Das Backbone des Mask-RCNN kann auch variieren. ResNet können aber auch verschiedene groß sein. In sehr aufwendigen Mask-RCNNs werden häufig ResNet101 verwendet.

Im zweiten Schritt wird nun ein Region proposal Network aufgebaut. Dies dient, wie auch in den Grundlagen erwähnt, zum Abschätzen, welche Regionen von Interesse sein könnten. Es erhält die Feature Maps aus dem Backbone. Aufgebaut wird es grundlegend aus Convolutional Layern und den Batchnormalization Layern. Die Batchnormalization Layer dienen dazu, Overfitting zu vermeiden genau so wie Dropout Layer. Sie normalisieren die Daten aus einem Layer mithilfe der Standardabweichung und Mittelwerte, um eine extreme Skalierung der Werte zu verhindern. Im ResNet kommen Sie auch häufig vor, da sie ebenfalls das Problem des verschwindenden Gradienten verhindern können. Die Ausgabe des Region proposal Network sind nun Regionen auf den Feature Maps, die das Netzwerk für interessant hält.

Nach dem Region proposal Network folgt die RoI-Align Schicht. Diese Schicht ist zuständig für das sogenannte RoI-Pooling.

Nach dem RoI-Pooling folgt nun die Klassifikation. Die RoI-Align Schicht gibt einen festen Wert zurück, den man durch ein Dense-Layer weiterleiten kann. In der Klassifikationsschicht wird das auch gemacht. Nach dem ersten Dense Layer folgt ein Batchnormalization Layer und ein Dropout Layer, um die Lernrate zu erhöhen und das Overfitting zu verhindern. Zum Schluss folgt ein weiterer Dense-Layer.

Im Anschluss wird die Maskenvorhersage getroffen. Dies wird mit einem Aufbau realisiert, der im Grunde dem Aufbau des Region proposal Network sehr ähnelt. Es werden Convolutional Layer und Batchnormalization Layer hintereinandergeschaltet, dies drei Mal.

Bevor man nun das Netzwerk aufbaut, müssen noch die einzelnen Loss Funktionen definiert werden. Die Loss Funktionen beschreiben die Lernkurve, die da System aufweist. Der „Loss“, also der Verlust des Lernens, soll möglichst klein werden. Die Loss-Funktion

Das vorliegende System muss drei Verlustfunktionen aufweisen:

- Verlustfunktion für das Region proposal Network und Klassifikation
- Verlustfunktion für die Regionen an sich
- Verlustfunktion für die Maskenbestimmung

Für die drei Loss-Funktionen werden verschiedene Arten verwendet, um die Fehler zu berechnen. In der API Keras sind die Verlustfunktionen bereits vorhanden und können leicht implementiert werden.

1. Loss-Funktion für die Klassifikationsaufgaben
    a. Kategorial Cross Entropie oder Binary Cross Entropie je nach Anwendungsfall
2. Loss-Funktion für die Box-Regression für die einzelnen Regionen
    a. Smooth L1 Loss
3. Loss-Funktion für die Maskenbestimmung
    a. Ebenfalls Binary Cross Entropie
4. Zusammenfügen aller Verlusten in einem Gesamtverlust

Zuletzt muss das Modell noch kompiliert und zum Schluss trainiert werden. Damit dies so funktioniert, wie es im ersten CNN war, muss zuerst das Modell zusammengefügt

werden. Wir kreieren dafür ein eigenes Modell. Dafür definieren wir einen Input und einen Output aus den vorher definierten Blöcken (Backbone, RPN, Klassifikationsvorhersage und Maskenvorhersage).

Der Inputteil des Modells ist das Backbone. Hier werden die Daten in das System hineingeführt. Die anderen drei Bausteine werden als Output definiert.

1. Modell erstellen
    a. Input gleich Backbone setzen
    b. Output gleich RPN, Klassifikation und Maskenbestimmung setzen

Zu beachten ist, dass der Output als Liste definiert werden muss, da es mehrere Komponenten enthält.

Ist das Modell erstellt, kann es nun kompiliert werden. Man muss folgende Parameter festlegen:

1. Modell kompilieren
    a. Optimizer festlegen
    b. Loss Funktion angeben
        i. In diesem Fall sind es drei beziehungsweise vier Loss Funktionen als Dictionary

Um nun den Trainingsprozess zu starten, müssen die Trainingsdaten angegeben und die Parameter zur Batchsize sowie die Anzahl an Epochen bestimmt werden. Dies ist nun auch etwas anders, da der Datensatz für ein Mask RCNN sich von einem Datensatz von einem CNN unterscheidet.

Zu diesem Netzwerk wird kein Code hinzugefügt, da er jederzeit Online verfügbar ist und nicht eigenhändig erarbeitet wurde. Die Quelle wird im Literaturverzeichnis aufgeführt.

Fazit

Fazit zur Lösung mit einem CNN

Convolutional Neural Networks sind eine sehr gute Variante Bildklassifikation umzusetzen. Ich gehe davon aus, dass es sehr gut möglich ist, ein CNN zu trainieren, welches zur Qualitätssicherung von Ultraschallbildern dient. Es muss vorher gut festgehalten werden, welche Merkmale von besonderer Wichtigkeit sind. Im Blick auf medizinische Zwecke umso mehr.

Im Laufe der Arbeit sind auch die sogenannten ResNet, also Residual Neural Networks, aufgetaucht, die man im Grunde für eine solche Aufgabe gut verwenden kann, da sie im Vergleich zu „normalen“ CNNs besser abschneiden. Man kann sie bedeutend tiefer in ihrer Struktur aufbauen, ohne Overfitting zu riskieren.

Zudem ist die Rubrik der CNNs ein nicht zu komplexes Konzept von Deep Learning, welches gut geeignet ist, um sich hineinzuarbeiten. An Flexibilität mangelt es auch nicht, da man im Allgemeinen viel einstellen kann über die Hyperparameter.

Fazit zur Lösung mit einem Mask RCNN

Im Gegensatz zu den CNNs sind die Mask RCNNs deutlich komplexer in ihrer Struktur. Sie haben verschiedene Arten von Netzwerken integriert und kombinieren diese. Dies ist von Vorteil, da es Modularer ist.

Zusätzlich dazu hat es noch den Vorteil, dass es eine örtliche Auflösung besitzt, sprich Objekte innerhalb eines Bilder klassifizieren kann. Ein einfaches CNN kann nur das Bild im Gesamten klassifizieren.

Von Nachteil kommt die Komplexität, jeweils im Aufbau und in der Berechnung sowie Bestimmung der Masken. Zudem benötigt es einen besser aufbereiteten Datensatz, der wohlmöglich mehr Speicherplatz benötigt und somit die Kosten steigert. Auch die Rechenleistung des Computers muss höher sein und erfordert unbedingt eine Grafikkarte. Auch hier werden die Kosten erhöht.

Nun kommt es darauf an, was die Aufgabe des Systems sein soll. Ein CNN wird ausreichen, um ein bestimmtes Organ auf einem Ultraschallbild zu klassifizieren. Dennoch ist es besser ein System zu erstellen, welches eine örtliche Auflösung besitzt, damit das System die Region innerhalb des Bildes angeben kann, die möglicherweise für den Arzt von Relevanz ist. Ein CNN kann keinen genauen Ort angeben, sondern nur eine Diagnose stützen.

Zum Schluss ist zu sagen, dass beide Systeme gut geeignet sind, beide Vorteile haben, aber abzuwägen ist, was genau gefordert ist und welche Voraussetzungen man mitbringt. Da es um Qualitätssicherung für Ultraschallbilder geht, wird das CNN gut sein, um allgemeine Anforderungen zu sichern. Geht es um genaue Bestimmung und Erkennung von Formen ist das Mask RCNN besser geeignet.

Literaturverzeichnis

[1]: Aarohi, C. W. (20. Juli 2021). *Youtube*. Von https://www.youtube.com/watch?v=t1MrzuAUdoE abgerufen

[2]: Haber, P. D. (https://www.kvno.de/fileadmin/shared/pdf/online/genehmigungen/ultraschall/sonografie_dokupruefung.pdf. Januar 2011). *https://www.kvno.de/fileadmin/shared/pdf/online/genehmigungen/ultraschall/sonografie_dokupruefung.pdf.* Von https://www.kvsaarland.de/qualitatssicherung/-/asset_publisher/QCPVl2mI8T7d/content/id/569901?_com_liferay_asset_publisher_web_portlet_AssetPublisherPortlet_INSTANCE_QCPVl2mI8T7d_redirect=https%3A%2F%2Fwww.kvsaarland.de%2Fqualitatssicherung%3Fp_p_id%3Dcom abgerufen

[3]: He, K. (10. Dezember 2015). *Arxiv.* Von https://arxiv.org/abs/1512.03385: https://arxiv.org/pdf/1512.03385.pdf abgerufen

[4]: Höfling, M. J. (1. April 2022). *Datacenter-Insider*. Von https://www.datacenter-insider.de/was-ist-resnet50-a-1107209/#:~:text=Die%20Zahl%20hinter%20dem%20Begriff,50%20Schichten%20(Layern)%20handelt. abgerufen

[5]: Kumar, A. (kein Datum). *Vitalflux*. Von https://vitalflux.com/keras-categorical-cross-entropy-loss-function/ abgerufen

[6]:La, F. L. (April 2019). *Microsoft.com*. Von https://learn.microsoft.com/de-de/archive/msdn-magazine/2019/april/artificially-intelligent-how-do-neural-networks-learn abgerufen

[7]: Matterport. (1. April 2019). *GitHub*. Von https://github.com/matterport/Mask_RCNN abgerufen

[8]: Matthieu Deru, A. N. (2020). *Deep Learning mit TensorFlow, Keras und TensorFlow.js.* Saarbrücken: Rheinwerk.

[9]: *mohitjainweb.* (kein Datum). Von https://mohitjainweb.files.wordpress.com/2018/03/smoothl1loss.pdf abgerufen

[10]: Moser, A. (18. März 2021). *fbmn.h-da.de.* Von https://fbmn.h-da.de/fileadmin/Dokumente/Studium/DS/2021_MDS_MoserArthur_THE.pdf abgerufen

[11]: Mukherjee, S. (18. August 2022). *Towardsdatascience*. Von https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758 abgerufen

[12]: Mwiti, D. (31. Januar 2023). *Neptune.ai*. Von https://neptune.ai/blog/keras-loss-functions abgerufen

[13]: TensorFlow. (18. November 2022). *Tensorflow.org*. Von https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50/ResNet50 abgerufen

Abbildungsverzeichnis

Anhang

Code zum CNN:

```
## **Importierte** **Bibliotheken**
"""

import tensorflow as tf
from tensorflow import keras
import numpy as np
from matplotlib import pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, Dense, Conv2D, MaxPooling2D, BatchNormalization, Flatten, Dropout
import cv2 as cv
from tensorflow.keras.datasets import mnist

"""## **Dataset laden** """

(x_train, y_train), (x_test, y_test) = mnist.load_data()

#Normalisieren Sie die Bilder
x_train = x_train / 255.0
x_test = x_test / 255.0

# Ändern Sie die Form der Bilder, damit sie in das CNN passen
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

"""## **Modell aufbauen**

### Sequentielles aufbauen
"""

beispiel_modell = Sequential()
beispiel_modell.add(InputLayer(input_shape=(28,28,1)))

beispiel_modell.add(Conv2D(32, (3,3), activation='relu'))
beispiel_modell.add(MaxPooling2D(pool_size=(2,2)))
beispiel_modell.add(Dropout(0.1))

beispiel_modell.add(Conv2D(32, (3,3), activation='relu'))
beispiel_modell.add(MaxPooling2D(pool_size=(2,2)))
beispiel_modell.add(Dropout(0.1))

beispiel_modell.add(Conv2D(16, (3,3), activation='relu'))
beispiel_modell.add(MaxPooling2D(pool_size=(2,2)))
beispiel_modell.add(Dropout(0.1))

beispiel_modell.add(Flatten())

beispiel_modell.add(Dense(512, activation='relu'))
beispiel_modell.add(Dense(10, activation='softmax'))

"""### Kompilieren"""

beispiel_modell.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])

"""### Modell *fitten*"""

beispiel_modell.fit(x_train, y_train, batch_size=20, epochs=2, validation_split=0.2)

"""## **Test ob das Netzwerk funktioniert**"""

test_bild = x_test[11].reshape(-1, 28, 28, 1)
test_bild_darstellung = test_bild.reshape(28,28)
plt.imshow(test_bild_darstellung)

pred = beispiel_modell.predict(test_bild)
max_wert= np.argmax(pred)
print("Das Bild zeigt eine: ", max_wert)
```