DHBW
Duale Hochschule
Baden-Württemberg

Deep Learning auf medizinischen Bilddaten

Studienarbeit

des Studiengangs Angewandte Informatik

an der Dualen Hochschule Baden-Württemberg **Mosbach Campus Bad Mergentheim**

von

Patrick Arndt

Mai 2023

Matrikelnummer, Kurs	7414700, INF20
Gutachter	Malte Tölle

Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema *Deep Learning auf medizinischen Bilddaten* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Eckartshausen, Mai 2023

Patrick Arndt

Abstract

Diese Studienarbeit bietet eine Einführung in das Thema Deep Learning, speziell im medizinischen Bereich. In dieser Arbeit wurde ein Modell entwickelt, welches Gehirntumore auf MRT Bildern segmentiert. Ein besonderer Fokus lag auf dem Einsatz der U-Net Architektur, einer speziellen Form eines Convolutional Neural Networks (CNNs), die sich durch ihre effiziente und genaue Segmentierungsleistung in biomedizinischen Anwendungen auszeichnet.

Die Arbeit umfasste die Vorverarbeitung des Brain Tumor Segmentation Challenge (BraTS) 21 Datensatzes, einschließlich Normalisierung, Größenskalierung und Daten-Augmentation, um die Qualität und Vielfalt der Trainingsdaten zu erhöhen. Darüber hinaus wurden Konzepte wie Backpropagation, Aktivierungsfunktionen, Verlustfunktionen und das Einstellen von Hyperparametern behandelt, um den Trainingprozess des U-Nets zu optimieren.

Die erzielten Ergebnisse zeigten, dass das trainierte U-Net in der Lage ist, Hirntumoren in MRT-Bildern mit einer guten Leistung zu segmentieren. Dennoch wurden auch einige Bereiche identifiziert, in denen Verbesserungen möglich sind, wie z.B. die Feinabstimmung von Hyperparametern und die Erweiterung von Vorverarbeitungstechniken. Die Ergebnisse der Arbeit wurden für die Einführung erarbeitet und bieten einen Einblick was alles möglich ist mit Deep Learning und Künstlicher Intelligenz.

Inhaltsverzeichnis

Glossar

Algorithmus

Ein Algorithmus ist eine genau definierte Folge von Aktionen, für die Lösung eines Problems[vgl. 1].

Batch

Definiert die Anzahl an Trainingsdaten, die durch das Neuronale Netz gereicht werden, bevor die interenen Parameter in diesem angepasst werden.

Ionisierende Strahlung

Strahlung die so stark ist, das sie Elektronen aus Atomen oder Molekühlen entfernt kann, sodass diese positiv geladen sind (ionisiert). Diese Art von Strahlung kann Mensch und Umwelt schädigen.[vgl. 2].

Modell

Ein Modell beschreibt eine komplexe mathematische Funktion, das anhand von Daten trainiert wurde, um bestimmte Vorhersagen oder Entscheidungen zu treffen. [vgl. 3].

Proton

Ein stabiles, elektrisch positiv geladenes Teilchen.

Abkürzungsverzeichnis

BraTS	Brain Tumor Segmentation Challenge
CNN	Convolutional Neural Network
KI	Künstliche Intelligenz
KNN	Künstliches Neuronales Netz
MRT	Magnetresonanztomographie
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

1.1. Einführung und Hintergrund

Seit mehr als 50 Jahren gibt es in der Medizin bildgebende Verfahren, mit denen Aufnahmen aus dem Inneren des menschlichen Körpers gemacht werden können. Diese Verfahren wurden kontinuierlich weiterentwickelt und sind heute in der Lage, detaillierte Bilder zu erzeugen. Technologien wie die Computertomographie (CT) oder die Magnetresonanztomographie (MRT) sind in der Lage, präzise Informationen über innere Organe, Strukturen und Gewebe im Körper zu liefern. [vgl. 4] Mit den modernen bildgebenden Verfahren steigt die Datenmenge und -komplexität und damit auch die Herausforderung an die Analyse und Verarbeitung der Daten bzw. Bilder.

Aufgrund der sich ständig weiterentwickelnden Technologien setzen immer mehr medizinische Einrichtungen bildgebende Verfahren zur Diagnose und Behandlung von Krankheiten ein. Die Analyse und Verarbeitung dieser Daten ist jedoch im Laufe der Zeit immer komplexer geworden und stellt eine Herausforderung für Ärzte und medizinisches Personal dar. Die manuelle Analyse und Verarbeitung der Daten ist oft mit Schwierigkeiten und Fehlerquellen verbunden. Der Prozess ist in der Regel sehr zeitaufwendig, weshalb es häufig zu langen Wartezeiten bei der Auswertung für die Patienten kommen kann. Nicht nur die Wartezeit ist ein Problem, sondern auch menschliche Fehler bei der Interpretation der Daten, die zu Fehldiagnosen oder Fehlbehandlungen führen können. [vgl. 5]

Diesen Herausforderungen kann mit dem vielversprechenden Ansatz der Automatisierung begegnet werden. So kann beispielsweise die Effizienz gesteigert und die Fehlerquote gesenkt werden. Ein wichtiges Stichwort ist hier die Künstliche Intelligenz (KI), die in den letzten Jahren ebenfalls

enorme Fortschritte gemacht hat. Insbesondere im Bereich der Bildanalyse haben KI-Systeme durch die Entwicklung von Deep-Learning-Modellen enorm an Genauigkeit gewonnen. Die Anwendung von Deep Learning auf medizinische Bilddaten bietet vielversprechende Möglichkeiten für eine verbesserte Diagnose, Therapieplanung und -überwachung sowie für die Entwicklung personalisierter medizinischer Lösungen.

Gerade im medizinischen Bereich hat der Einsatz von KI-Systemen auf medizinischen Bilddaten in den letzten Jahren an Bedeutung gewonnen. Mit Hilfe von Deep-Learning-Modellen können Ärzte und Forscher komplexe Bilder aus bildgebenden Verfahren wie Röntgenaufnahmen, CT- und MRT-Scans automatisiert analysieren. Diese Analyse kann zur Erkennung von Krankheiten, zur Identifizierung von Tumoren und zur Bewertung der Wirksamkeit von Therapien eingesetzt werden.

1.2. Problemstellung und Zielsetzung

Die zunehmende Komplexität und Menge medizinischer Daten und Bilder stellt Ärzte vor große Herausforderungen. Die manuelle Analyse und Verarbeitung der Daten ist zeitaufwändig, fehleranfällig und birgt die Gefahr, dass wichtige diagnostische Informationen übersehen werden. Gerade im Bereich von Tumoren, also Krebszellen, ist die Früherkennung, aber auch die Behandlung sehr wichtig für das Wohl des Patienten.

Vor allem im Bereich des Gehirns muss mit Tumoren sehr behutsam umgegangen werden, um keine bleibenden Schäden beim Patienten zu hinterlassen. Bei Hirntumoren ist es wichtig, die genaue Position und Lage zu kennen, um eine geeignete Behandlung zu finden. Die Operation eines Hirntumors erfordert eine umfangreiche Vorbereitung und ist komplex, da die Fachärzte den Tumor zunächst genau segmentieren müssen, bevor sie die weiteren Schritte planen. Diese Vorbereitung ist zeitaufwändig und kann nur von ausgebildeten Fachärzten durchgeführt werden.

Krebs ist ein vielschichtiges Thema, mit dem sich die Medizin seit vielen Jahren beschäftigt. Vor allem bei der Früherkennung von Tumoren setzen Ärzte heute auf die Hilfe der KI. Aber nicht nur bei der Früherkennung, sondern auch bei der Behandlung solcher Tumore.

Die Problemstellung dieser Studienarbeit ist die Identifizierung und Klassifizierung von Tumoren im Gehirn, genauer gesagt eine Multiklassen-Segmentierung, bei der jedem Bildpunkt eine Tumorklasse zugeordnet wird. Die verwendeten Daten stammen aus dem öffentlichen Datensatz BraTS (Brain Tumor Segmentation Challenge), der aus ca. 1250 MRT-Bildern von Gehirnen besteht, die von Fachärzten auf Tumore untersucht und gegebenenfalls markiert wurden. Die Herausforderung besteht darin, ein geeignetes Deep-Learning-Modell zu entwickeln, das für die Segmentierung von Tumoren im Gehirn geeignet ist und möglichst genaue Ergebnisse liefert.

Das Ziel dieser Arbeit ist es, ein Modell für die Segmentierung von Gehirntumoren auf Basis des BraTS Datensatzes zu erstellen. Das Modell soll in der Lage sein, zwischen drei verschiedenen Tumorklassen zu unterscheiden und für diese eine Maske zu erstellen. Für die Entwicklung des Modells wird das von Facebook entwickelte Open Source Machine-Learning Framework PyTorch verwendet. Das Framework bietet zahlreiche Werkzeuge und Funktionen für die Entwicklung von Deep Learning-Modelle. [vgl. 6]

2. Theoretische Grundlagen

2.1. Künstliche Intelligenz

KI wird zunehmend in verschiedenen Bereichen eingesetzt und erleichtert die Arbeit der Menschen enorm. Viele Arbeiten können mit Hilfe von KI schneller und besser erledigt werden. Im folgenden Abschnitt werden die Begriffe 'Intelligenz' und 'Künstliche Intelligenz' erläutert, um ein einfaches Verständnis zu schaffen. Außerdem werden die verwandten Begriffe Machine Learning und Deep Learning in den Kontext eingeordnet.

In der Psychologie bezieht sich Intelligenz auf die Fähigkeit, logische, sprachliche, mathematische oder sensorische Probleme zu lösen, aber es gibt keine allgemeingültige Definition von Intelligenz. Die kognitiven Fähigkeiten des Menschen ermöglichen es ihm, sich an Beschreibungen oder Erklärungen von Dingen zu erinnern, die er zu einem späteren Zeitpunkt wieder abrufen und verwenden kann. Das menschliche Gehirn, das aus Milliarden von Neuronen besteht, lernt und speichert bestimmte Strukturen, Konzepte und Fähigkeiten. Beim Lernen werden die Verbindungen zwischen den Neuronen im Gehirn verstärkt. Je öfter etwas gelernt wird, desto stärker werden bestimmte Verbindungen zwischen den Neuronen im Gehirn. [vgl. 7, 8]

Künstliche Intelligenz hingegen ist ein Wissenschaftsgebiet, das sich damit beschäftigt, Computern menschliches Verhalten beizubringen. Damit ist gemeint, dass ein Computer darauf trainiert wird, ähnlich wie ein Mensch zu denken und entsprechend zu handeln. [vgl. 9] Im Gegensatz zu biologischen Neuronen lösen Computer Probleme wie die Klassifizierung eines Bildes durch die Anwendung mathematischer Funktionen und Algorithmen. Es wird versucht, die Denkfähigkeit des Menschen nachzuahmen, um einen Mehrwert bei der Problemlösung durch Computer zu erzielen. Mit Hilfe der

künstlichen Intelligenz sind Computer heute in der Lage, Probleme zu lösen, die früher ein höheres intellektuelles Verständnis erforderten. Computer können beispielsweise Bilder klassifizieren oder Vorhersagen treffen. Ähnlich wie der Mensch lernt auch der Computer aus den vorhandenen Daten und verstärkt dabei bestimmte mathematische Zusammenhänge. [vgl. 10]

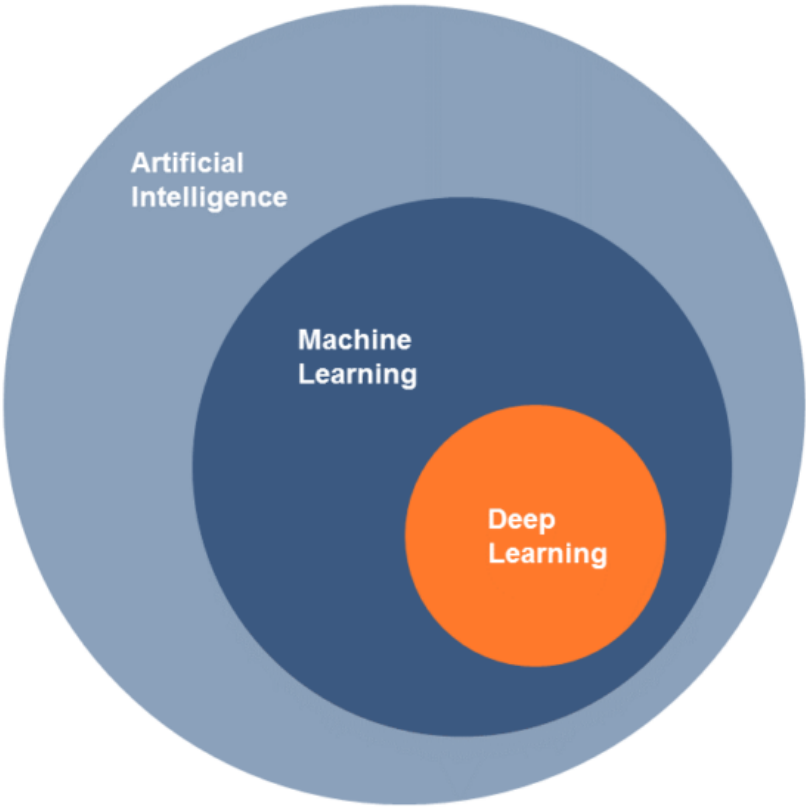


Abbildung 2.1.: Übersicht der Bereiche von künstlicher Intelligenz (Quelle: https://www.alexanderthamm.com/de/blog/ki_artificial-intelligence-ai-kuenstliche-neuronale-netze-machine-learning-deep-learning/)

Im Zusammenhang mit KI wird häufig auch von Machine Learning, Deep Learning und Neuronalen Netzen gesprochen. KI ist der Oberbegriff, der sowohl Machine Learning als auch Deep Learning umfasst. Deep Learning ist ein Teilbereich des Machine Learning, bei dem künstliche neuronale Netze zum Einsatz kommen, die es Computern ermöglichen, Dinge zu lernen und Probleme zu lösen. Im Folgenden wird der Schwerpunkt auf den Bereich Deep Learning gelegt.

2.2. Deep Learning

2.2.1. Einführung

Viele Probleme oder Aufgaben können heute effizient mit Hilfe eines Computers gelöst werden. Dazu wird in der Regel ein Algorithmus verwendet, der die Aufgabe systematisch löst. Es gibt jedoch komplexe Aufgaben, die ein Algorithmus nicht lösen kann.

Ein Beispiel hierfür ist die Vorhersage, was ein Kunde in der nächsten Zeit kaufen wird, um ihm entsprechende Vorschläge zu unterbreiten. Probleme dieser Art können nicht mit einem Algorithmus gelöst werden, sondern die Lösung muss aus den vorhandenen Daten ermittelt werden. In solchen Fällen soll der Computer selbst einen Algorithmus erstellen, der dann auf neue Daten angewendet werden kann. Der Computer lernt aus bereits vorhandenen Daten oder Erfahrungen einen Algorithmus zu extrahieren, dabei werden unter anderem Muster und Strukturen erkannt. Mit dem extrahierten Algorithmus kann dann das Kaufverhalten des Kunden approximiert werden. Möglich wird dies durch die enormen Datenmengen, die sich in den letzten Jahren angesammelt haben. Bei jedem Online-Einkauf, beim Aufruf einer Webseite oder beim Öffnen einer App werden Daten erzeugt und gesammelt. Diese Daten können schließlich genutzt werden, um Probleme wie die oben beschriebenen zu lösen.[vgl. 11]

Aufgaben wie das Klassifizieren von Bildern, das Komponieren von Musik[1] oder die Generierung von Daten aus Text sind heute mit Deep Learning möglich. Es gibt drei Hauptarten, wie ein Computer lernen kann, die in den Abschnitten 2.2.2, 2.2.3 und 2.2.4 beschrieben werden.

[1] siehe [12]

2.2.2. Überwachtes Lernen

Überwachtes Lernen ist ein wichtiger Bereich des maschinellen Lernens, bei dem ein Modell anhand von gelabelten Daten lernt. Gelabelt bedeutet, dass den Daten bereits das gewünschte Ergebnis oder der erwartete Wert zugeordnet ist. Ziel des überwachten Lernens ist es, einen Algorithmus oder ein Modell zu finden, das für neue und noch nicht gesehene Daten die richtigen Labels liefert.[vgl. 13, 14]
Das überwachte Lernen umfasst zwei verschiedene Probleme, die Klassifikation und die Regression. Bei der Klassifikation werden zu Bildern eine oder mehreren Klassen zugeordnet, es können aber auch Texte, Videos oder andere Dinge klassifiziert werden. Ein Beispiel wäre die Klassifizierung von Hunden und Katzen auf Bildern. Dazu wird eine große Anzahl von Bildern von Hunden und Katzen in das Modell eingespeist und dieses ordnet den Bildern jeweils die richtigen Klassen zu. Sieht das Modell z.B. ein Bild mit einem Hund, so kann es nach dem Training die Klasse “Hund” zuordnen. Die meisten E-Mail-Programme verwenden ein solches Klassifikationsmodell, um zwischen seriösen und Spam-E-Mails zu unterscheiden. Das Ergebnis einer Klassifikation ist immer eine Kategorie, die die zu klassifizierenden Daten am besten beschreibt.
Der zweite Bereich des maschinellen Lernens ist die Regression, die häufig für Vorhersagen und Prognosen verwendet wird. Sie ist der Klassifikation ähnlich, aber die Zielmenge ist eine andere. Es wird versucht, Werte für einen meist kontinuierlichen Bereich vorherzusagen, dazu wird der Datensatz mit einer Funktion approximiert. Das Ergebnis einer Regression sind Werte oder Punkte, die die Funktion am besten beschreiben.[vgl. 13]
Im Folgenden geht es hauptsächlich um das überwachte Lernen, genauer gesagt um die Klassifikation. Dabei wird nicht wie im obigen Beispiel ein ganzes Bild klassifiziert, sondern es werden den einzelnen Bildpunkten Klassen zugeordnet.

2.2.3. Unüberwachtes Lernen

Beim unüberwachten Lernen sind die Daten nicht mit Labels oder Kategorien versehen, wie dies beim überwachten Lernen der Fall ist. Diese Methode des maschinellen Lernens wird eingesetzt, wenn die Beschriftung der Daten nicht ohne weiteres möglich ist. Das Modell sucht ohne vorgegebene Kategorien nach Mustern und Strukturen in den Daten. Mit Hilfe statistischer Methoden wird die beste Kategorisierung der Daten gebildet. Das Ergebnis des unüberwachten Lernens sind verschiedene Gruppen von Daten, die jedoch nicht benannt, sondern durchnummeriert werden. Die verschiedenen Daten in den Gruppen haben ähnliche oder gleiche Eigenschaften.
Ein gutes Beispiel dafür ist der Bereich "Kunden kauften auch" bei Amazon. Hier werden dem Kunden ähnliche Produkte oder Artikel, die oft in Kombination gekauft werden, vorgeschlagen. Es werden keine bestimmten Produktkategorien genannt, sondern nur ähnliche Produkte vorgeschlagen. Es ist schwierig, die Ergebnisse des unüberwachten Lernens zu bewerten, da es keine vergleichbaren Daten mit entsprechenden Annotationen gibt und somit keine Fehler berechnet werden können.[vgl. 9, 13]

2.2.4. Bestärkendes Lernen

Oft gibt es Probleme, bei denen nicht genau festgestellt werden kann, ob etwas richtig oder falsch ist. Die Daten für solche Probleme, sofern vorhanden, sind daher nicht beschriftet. Es ist jedoch möglich zu sagen, ob die Ergebnisse zur Lösung des Problems geeignet sind oder nicht, in diesen Fällen kommt bestärkendes Lernen zum Einsatz. In der Robotik ist bestärkendes Lernen ein klassischer Anwendungsfall. Für Roboter stehen in der Regel keine Trainingsdaten zur Verfügung, so dass der Roboter selbst lernen muss, was gut und was schlecht ist. [vgl. 7]
Ziel des Roboters ist es, eine geeignete Lösung oder Strategie für das Problem zu finden. Beim bestärkenden Lernen spricht man oft von sogenannten Agenten, die darauf trainiert werden, ein Problem zu lösen. Auf der Suche

nach einer geeigneten Strategie werden diese kontinuierlich für gute und schlechte Aktionen belohnt oder bestraft. Ein einfaches Beispiel wäre ein Saugroboter, der in einem Hotel die Zimmer saugt. In den Zimmern, in denen der Roboter z.B. nichts umgestoßen hat, wird er belohnt. Stößt der Roboter etwas um oder saugt etwas Unerwünschtes auf, wird er bestraft. [vgl. 13] Das Belohnungssystem kann auf Punkten basieren, so dass der Saugroboter im positiven Fall Pluspunkte und im negativen Fall Minuspunkte erhält und sein Verhalten entsprechend anpasst.

2.2.5. Künstliches Neuronales Netz

Die Struktur und Funktion des menschlichen Gehirns ist das Vorbild für künstliche neuronale Netze. Sie bestehen aus einer Vielzahl von künstlichen Neuronen, die miteinander verbunden sind und Informationen verarbeiten. Ein Künstliches Neuronales Netz (KNN) wird für verschiedene Anwendungen wie Mustererkennung, Datenanalyse oder Vorhersagen eingesetzt.

2.2.5.1. Biologisches Neuron

Das biologische Neuron (siehe Abb. 2.2) ist das Vorbild für künstliche Neuronen. Es besteht aus Dendriten, die elektrische Signale von anderen Nervenzellen aufnehmen und an den Zellkörper weiterleiten. Im Axonhügel, der sich zwischen Zellkörper und Axon befindet, werden die elektrischen Signale gesammelt und summiert. Erst wenn ein bestimmter Schwellenwert überschritten ist, wird das Signal an das Axon weitergeleitet. Der Schwellenwert verhindert, dass sehr kleine Signale, die keine Relevanz haben, weitergeleitet werden. Ohne diese Signalfilterung wäre eine Verarbeitung der relevanten Signale nicht möglich. Das Signal, das schließlich über das Axon zu den Synapsen transportiert wird, heißt Aktionspotenzial. An die Synapsen schließen sich weitere Dendriten der Nervenzellen an und empfangen die weitergeleiteten Signale. Ein biologisches Neuron kann dabei bis zu

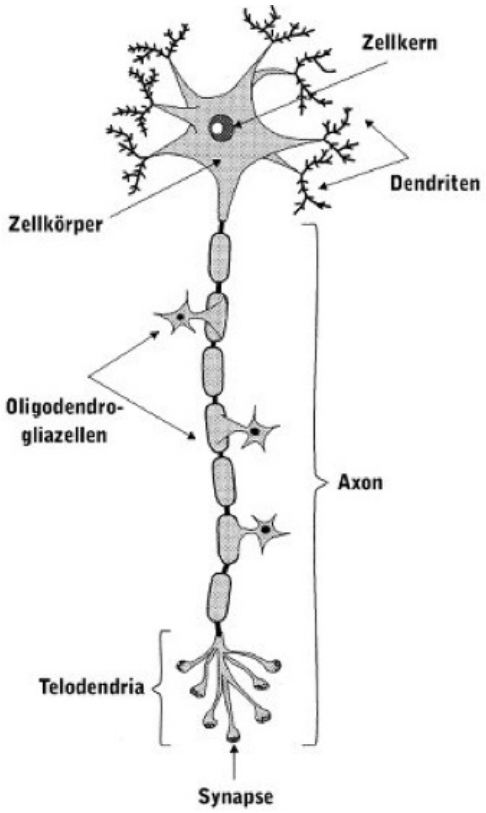


Abbildung 2.2.: Aufbau biologisches eines biologischen Neurons (Quelle: https://www.spektrum.de/lexikon/psychologie/neuron/10516)

mehrere tausend Verbindungen zu anderen Neuronen haben, die wiederum entsprechend viele Verbindungen haben können.[vgl. 8, 15]

2.2.5.2. Künstliches Neuron

Ein künstliches Neuron ist ein mathematisches Modell, das die Funktion eines biologischen Neurons nachahmt. Das künstliche Neuron in Abb. 2.3 besitzt Eingänge x_n, die mit den Dendriten des biologischen Neurons vergleichbar sind. Die Eingangsinformationen können von anderen Neuronen oder aus der Umgebung stammen.
Bevor die Eingangsinformationen in das Neuron gelangen, werden sie mit einem Gewicht w_{nj} multipliziert. Anschließend werden die Eingaben mit Hilfe einer Übertragungsfunktion (Propagationsfunktion) verknüpft, d.h. die Werte werden aufsummiert. Das summierte Ergebnis ist die Netzeingabe net_j, der die Informationen zusammenfasst, die in das Neuron eingehen. Die Aktivierungsfunktion berechnet die Aktivierung o_j, die abhängig vom

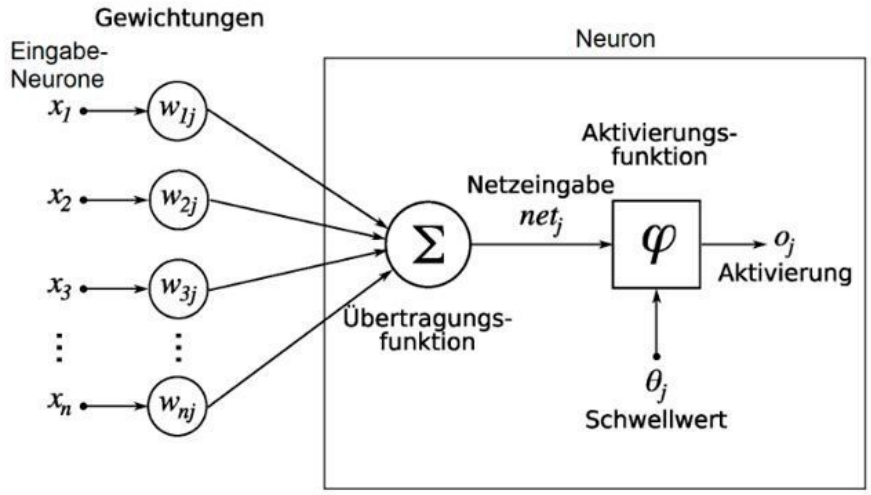


Abbildung 2.3.: Schema eines künstlichen $Neurons_j$ (Quelle: https://www.dev-insider.de/grundbegriffe-und-einfache-beispiele-a-864507/)

Schwellenwert θ_j an alle angeschlossenen Neuronen weitergegeben wird. [vgl. 15]

2.2.5.3. Das Perzeptron

Das Perzeptron ist ein einfaches künstliches neuronales Netz, das von Frank Rosenblatt als Modell vorgestellt wurde. Als Beispiel wird ein Photo-Perzeptron beschrieben, das in der Lage ist, einfache visuelle Muster zu erkennen. Es besteht aus der Retina (Netzhaut), der Assoziationsschicht und der Ausgabeschicht. Die Retina liefert binäre Eingangswerte, die über gewichtete Verbindungen an die Assoziationsschicht weitergeleitet werden. Die Neuronen der Retina sind fest mit den Neuronen der Abbildungsschicht verbunden, aber nicht alle Neuronen sind miteinander verbunden. Die Verbindungen zwischen der Abbildungsschicht und der Ausgabeschicht sind dagegen variabel. Während des Trainings, also des Lernens, werden die Gewichtungen zwischen ihnen angepasst.
Beim Training werden dem Perzeptron verschiedene Beispiele gezeigt, anhand derer es lernen soll. Ein Beispiel besteht immer aus einem Eingangsvektor x und einem Ausgangsvektor y. Die Aufgabe des Perzeptrons ist es, durch Anpassung der internen Werte zu jedem Eingabevektor x einen pas-

senden Ausgabevektor y zu erzeugen. Ziel des Trainings ist es, bisher nicht gesehenen, aber ähnlichen Eingabevektoren x' ebenfalls einen passenden Ausgabevektor y zuzuordnen, man spricht in diesem Fall von Generalisierung. [vgl. 16]
Die Struktur des Perzeptrons schränkt jedoch die Klassifikation von Daten ein. Es ist nur in der Lage, linear separierbare Datenmengen zu trennen. Das heißt, die Datenmenge kann durch eine gegebene Gerade in zwei separate Datenmengen getrennt werden. Die Ausgänge des Perzeptrons sind also auf 0 oder 1 beschränkt, wobei nicht alle Datenmengen trennbar sind, sondern nur solche, deren Trenngerade durch den Ursprung geht.[7]

2.2.5.4. Aufbau

Der klassische Aufbau eines Neuronalen Netzes besteht aus einer Eingabeschicht (engl.: Input Layer), einer oder mehreren verborgenen Schichten (engl.: Hidden Layer) und der Ausgabeschicht (engl.: Output Layer). Die Eingabeschicht nimmt Werte aus der Umgebung auf und leitet sie an die versteckte Schicht weiter, von wo aus sie an weitere versteckte Schichten oder an die Ausgabeschicht, die das Endergebnis ausgibt, weitergeleitet werden. Die Verbindungen der Neuronen zwischen den verschiedenen Schichten haben Gewichtungen, mit denen die eingehenden Werte multipliziert werden. Die Komplexität der Struktur hängt von der Anzahl der Neuronen und Schichten ab. [vgl. 13]

2.2.5.5. Training eines Neuronalen Netzes

Das Training eines neuronalen Netzes ist ein wesentlicher Aspekt des Deep Learning, bei dem die Anpassungen der Gewichte und Schwellenwerte innerhalb des Netzes erfolgen. Das Training kann je nach Art und Menge der Daten mehrere Stunden oder sogar Tage dauern. Ein neuronales Netz gilt als trainiert, wenn es zu einem Eingabevektor x zuverlässig einen passenden Ausgabevektor y bestimmen kann. Das Netz ist dann in der Lage,

auch bisher unbekannten Eingabevektoren x' passende Ausgabevektoren y' zuzuordnen. [vgl. 16]

Ein entscheidendes Konzept beim Training ist die Fehler- oder Verlustfunktion (engl.: Loss), die im Abschnitt 2.2.6 genauer beschrieben wird und die Differenz zwischen den vorhergesagten und den tatsächlichen Ergebnissen vergleicht und einen Fehler berechnet. Ziel des Trainings ist es, die Verlustfunktion durch Anpassung der internen Parameter des neuronalen Netzes zu minimieren. Die Parameter des Netzes werden zu Beginn des Trainings zufällig initialisiert und während des Trainings schrittweise angepasst. [vgl. 17]
Die Rückwärtspropagation (engl.: Backpropagation) ist ein weit verbreitetes Lernverfahren für Neuronale Netze, das auf der oben genannten Fehlerminimierung basiert. Bei der Backpropagation wird im Allgemeinen die Delta-Lernregel verwendet, die besagt, dass die Fehlerminimierung durch Änderung eines Gewichts w_{ij} einen Gradientenabstieg anstrebt. Der Ausgangspunkt dieser Lernregel ist wie folgt definiert

$$\Delta_p w_{ij} \equiv -\frac{dE_p}{dw_{ij}} \tag{2.1}$$

$\Delta_p w_{ij}$ entspricht dabei der Änderung eines Gewichts zwischen i und j. dE_p ist der gemessene Fehler innerhalb der Lernmenge und dw_{ij} ist die Gewichtung der Neuronen von der Schicht i zur Schicht j. [vgl. 18]

Der Backpropagation-Algorithmus besteht im Wesentlichen aus zwei Schritten, der Vorwärtspropagation und der Rückwärtspropagation. Der erste Schritt ist die Vorwärtspropagation, bei der die Eingabevektoren durch das Netzwerk verarbeitet werden. Die Eingaben werden von Schicht zu Schicht weitergeleitet und durch die Gewichtungen und Aktivierungsfunktionen transformiert. Am Ende dieses Schrittes steht eine Ausgabe, die für den zweiten Schritt verwendet wird.
Im nächsten und letzten Schritt des Trainings, der Rückwärtspropagation, wird die Ausgabe mit dem tatsächlich erwarteten Wert verglichen und anschließend aus der Differenz zwischen Soll- und Istwert ein Fehler berechnet.

Als Beispielfunktion für die Berechnung eines Fehlers kann der in Gleichung 2.2 definierte mittlere quadratische Fehler (MSE) verwendet werden.

$$E = \frac{1}{2} \sum (Y_i - \hat{Y}_i)^2 \tag{2.2}$$

Nach der Fehlerberechnung werden die Fehler für die Neuronen der einzelnen Schichten bestimmt. Dabei wird der Beitrag jedes Neurons zum Gesamtfehler berechnet. Anschließend wird die Gewichtung eines Neurons anhand seines Fehlers angepasst. Die Gewichte werden mit einem konstanten Faktor ν und einem weiteren Term in Abhängigkeit vom Fehler multipliziert. Dieser Prozess wird iterativ wiederholt, bis der Fehler auf ein geeignetes Minimum reduziert wurde. Ist der Fehler auf ein Minimum gesunken und ändert sich nicht mehr wesentlich, so spricht man von einer Konvergenz des Modell und das Training kann beendet werden.[vgl. 16]

2.2.6. Aktivierungsfunktionen, Verlust-Funktionen und Optimierer

Für das Training und die generelle Funktionalität eines Neuronalen Netzes werden noch einige Komponenten benötigt. Eine Komponente ist die bereits im Abschnitt 2.2.5.2 erwähnte Aktivierungsfunktion. Sie berechnet die Aktivierung o_j, die abhängig von einem Schwellwert θ_j an die nachgeschalteten Neuronen weitergegeben wird. Wird der Schwellenwert θ_j nicht überschritten, wartet das Neuron auf weitere Eingangssignale und wiederholt die Berechnung der Aktivierung o_j, bis der Schwellenwert θ_j überschritten wird und das Neuron seine Ausgabe weitergibt. Es gibt verschiedene Aktivierungsfunktionen für unterschiedliche Zwecke.

Die Identitätsfunktion ist eine einfache Aktivierungsfunktion, die eine lineare Abbildung der Eingabe auf die Ausgabe liefert. Sie eignet sich für einfache Aufgaben, wird die Problemstellung komplexer, sind nichtlineare Aktivierungsfunktionen wie Tangens hyperbolicus, Rectified Linear Unit (ReLU)

oder Sigmoid besser geeignet. Je nach Anwendung und Bedarf wird eine geeignete Funktion ausgewählt. Die Sigmoidfunktion normalisiert die Ausgaben eines Netzes auf den Bereich zwischen 0 und 1, während der Tangens hyperbolicus die Werte in den Bereich zwischen -1 und 1 bringt. Die Funktion ReLU verhält sich etwas anders als die vorhergehenden Funktionen, da sie für alle positiven Werte die entsprechenden Ausgaben und für alle negativen Werte den Wert 0 zurückgibt. Eine weitere erwähnenswerte Aktivierungsfunktion ist die Softmax-Funktion, die ebenfalls Werte in einen Bereich zwischen 0 und 1 transformiert, jedoch eine Wahrscheinlichkeitsverteilung darstellt. Die Besonderheit der Softmax-Funktion liegt darin, dass sich alle Ausgaben zu 1 addieren und somit die Ausgabewerte als Wahrscheinlichkeiten interpretiert werden können. [vgl. 19]
Eine weitere Komponente ist die Verlustfunktion, die für das Training eines Neuronalen Netzes essentiell ist. Diese hat zum Ziel, den Fehler, auch Verlust genannt, den ein Netz bei einer Vorhersage produziert, zu minimieren. Ein Fehler tritt auf, wenn das Netz eine falsche oder andere Ausgabe als die erwartete erzeugt. Um den Gesamtfehler eines Netzes zu berechnen, werden die Ausgaben des neuronalen Netzes mit den erwarteten Ergebnissen verglichen. Die Verlustfunktion hilft, die Differenz zu quantifizieren und den Gesamtfehler des Netzes zu berechnen. [19]
Wie bei den Aktivierungsfunktionen gibt es je nach Art der Aufgabe verschiedene Verlustfunktionen, wie z.B. Binary Cross Entropy (BCE), Categorical Cross Entropy oder Mean Squared Error. Die Binary Cross Entropy wird für binäre Klassifikationsprobleme verwendet, wie z.B. die Unterscheidung von seriösen und Spam-E-Mails, während die Categorical Cross Entropy für Probleme mit mehr als zwei Klassen verwendet wird. Die Wahl der Verlustfunktion hängt also auch von der Ausgabe des Netzes ab.

Die letzte wesentliche Komponente ist der Optimierer, der die Gewichte und Biases anpasst, um Fehler zu minimieren und die Leistung des Netzes zu verbessern. Biases sind zusätzliche Konstanten, mit denen ein Neuron versorgt wird und die bei der Berechnung der Ausgabe eine Rolle spielen. Der Bias erhöht die Fähigkeit des Netzes, komplexe Beziehungen zwischen

Eingaben und Ausgaben zu modellieren, und kann so auch nichtlineare Beziehungen erkennen.
Es gibt verschiedene Arten von Optimierungsalgorithmen, der am weitesten verbreitete Algorithmus ist der Stochastic Gradient Descent (SGD), der auch die Grundlage für weitere Optimierungsverfahren bildet. Der SGD aktualisiert Gewichte und Bias auf der Basis der Ableitung der Verluste. Zur Beschleunigung der Berechnung wird eine Zufallsstichprobe aus dem Trainingsdatensatz verwendet. [vgl. 20]
Ein weiterer Optimierungsalgorithmus ist Adam, der eine Erweiterung von SGD ist und eine adaptive Lernrate verwendet, um die internen Parameter des Netzes anzupassen. Durch die adaptiven Lernraten erreicht der Algorithmus eine schnellere Konvergenz und eine höhere Genauigkeit als der SGD. [vgl. 21]

2.2.7. Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) sind eine spezielle Form von Neuronalen Netzen, die Bilddaten verarbeiten können. Sie erkennen mit Hilfe verschiedener Filter Muster, Strukturen und Geometrien in Bildern und können diese klassifizieren. Im folgenden Abschnitt wird der Aufbau und die Funktionsweise eines CNNs erläutert, wobei auf die verschiedenen Schichten innerhalb des Netzes sowie auf die mathematische Operation Faltung (engl.: Convolution) eingegangen wird.

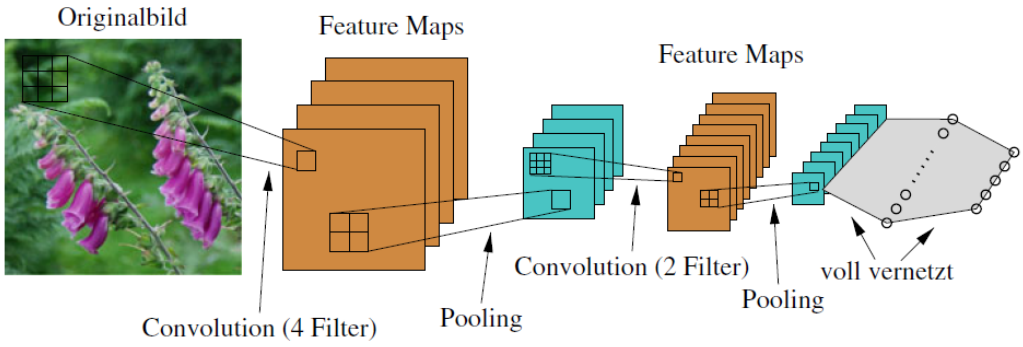


Abbildung 2.4.: Ein CNN mit zwei Faltungs-Schichten gefolgt von je einer Pooling-Schicht und am Ende zwei voll vernetzten Schichten (Quelle: [7])

Der Aufbau eines CNN besteht aus mehreren Faltungs-, Pooling- und vollvernetzten Schichten, wie in Abbildung 2.4 dargestellt

Convolutional-Layer ist die Faltungsschicht, die in der Lage ist, Merkmale wie Linien, Kanten und geometrische Formen in Bildern zu erkennen und zu extrahieren. Bei einer Faltung wandert ein Filter (engl.: Kernel) über das Eingabebild und erzeugt so ein neues Bild, auch Feature-Maps genannt. Mathematisch gesehen wandert der Filter über eine Funktion statt über ein Bild und erzeugt so ein neues Bild. Bei der Faltung eines Bildes wird meist ein Filter in Form einer 2D-Matrix verwendet, wie in Gleichung 2.3 als Beispiel dargestellt. Die Werte des Filters werden zu Beginn zufällig bestimmt und während des Trainings angepasst.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \tag{2.3}$$

Der Filter wird schrittweise über das Eingabebild bewegt, wobei jedes Filterelement mit dem darunterliegenden Pixel des Eingabebildes multipliziert und dann aufsummiert wird. Die Summe ist dann das Ergebnis des neuen Bildpunktes. Die Schrittweite des Filters wird “Stride” genannt und gibt an, um wie viele Bildpunkte (Pixel) der Filter verschoben wird. Bei einer 2D-Faltung können zwei Werte für den Stride angegeben werden, die Verschiebung in Richtung der X-Achse und die Verschiebung in Richtung der Y-Achse. Die Gleichung 2.5 gibt das Ergebnis einer Faltung mit dem Ausdruck aus 2.4 an.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} * \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \tag{2.4}$$

$$\begin{pmatrix} (a_{11}w_{11} + a_{12}w_{12} + a_{21}w_{21} + a_{22}w_{22}) & (a_{12}w_{11} + a_{13}w_{12} + a_{22}w_{21} + a_{23}w_{22}) \\ (a_{21}w_{11} + a_{22}w_{12} + a_{31}w_{21} + a_{32}w_{22}) & (a_{22}w_{11} + a_{23}w_{12} + a_{32}w_{21} + a_{33}w_{22}) \end{pmatrix} \tag{2.5}$$

Es ist zu beachten, dass das Ergebnis in 2.5 kleiner ist als das Originalbild. Dies liegt daran, dass kein Padding verwendet wurde, um die Ränder des Eingabebildes aufzufüllen. Die Größe der Ergebnismatrix kann mit der Gleichung 2.6 berechnet werden, wobei W die Eingabegröße des Bildes, F die Filtergröße, S der Stride und P das Padding ist. [vgl. 22]

$$W_{out} = \frac{W + 2P - F}{S} + 1 \tag{2.6}$$

Pooling-Layer dient der Extraktion von Bildmerkmalen (engl.: features), die eine hohe semantische Bedeutung haben. Die Eingangsdaten dieser Schicht sind die Ausgangsdaten der vorhergehenden Faltungsschicht. Diese Schicht verwendet entweder das Maximum- oder das Average-Pooling. Dabei werden überflüssige und redundante Informationen aus dem Datensatz entfernt. Durch das Entfernen der unwesentlichen Bestandteile wird die Rechenzeit reduziert und die Merkmale werden verdichtet.

Beim Pooling wird ebenfalls ein Filter, in der Regel der Größe 2×2, mit einer Schrittweite von zwei über das Bild bewegt. In Abbildung 2.5 werden beide Arten des Poolings durchgeführt. Das Max-Pooling extrahiert jeweils den höchsten Wert, der innerhalb der Filtermatrix liegt. Das Average-Pooling nimmt jeweils den Mittelwert aller Elemente innerhalb der Filter-Matrix. Durch das Pooling verkleinert sich das Bild je nach Filtergröße und Schrittweite.[vgl. 22] Im Beispiel der Abbildung 2.5 wird das Ausgangsbild um die Hälfte verkleinert, die Größe nach dem Pooling berechnet sich ebenfalls nach der Formel aus 2.6.

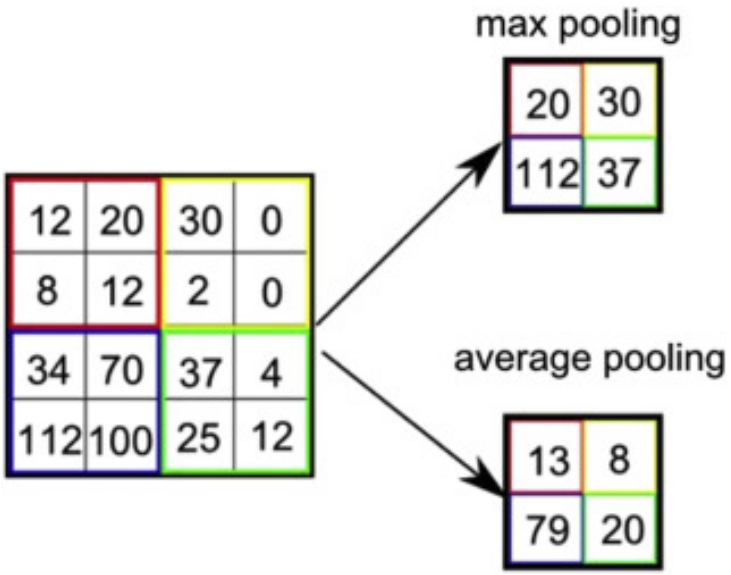


Abbildung 2.5.: Maximal und Mittelwert-Pooling, sowie das Ergebnis der beiden Methoden. (Quelle: https://www.sciencedirect.com/topics/mathematics/pooling-layer)

Fully-Connected-Layer oder vollständig verknüpfte Schicht, bildet den Abschluss eines CNNs. Die Merkmale der vorhergehenden Schicht, also der Pooling-Schicht, werden hier mit allen Ausgabemerkmalen verknüpft, es handelt sich also um ein normales Neuronales Netz.

Zunächst muss aus der Ausgabe der vorhergehenden Schicht eine 1D-Matrix erzeugt werden, dieser Vorgang wird auch "flatten" genannt. Anschließend wird jedes Element der 1D-Matrix mit jedem Element der Ausgabeschicht verbunden. Die Ausgabe des Netzes zeigt dann z.B. die Klassifikation eines Bildes, indem berechnet wird, welches Ausgabeelement die höchste Wahrscheinlichkeit hat. [vgl. 23]

2.2.8. Over- und Underfitting

Over- und Underfitting sind zwei Probleme, die beim Training eines neuronalen Netzes auftreten können. Diese Probleme beeinflussen die Generalisierungsfähigkeit des Modells, d.h. die Fähigkeit, auf neuen, bisher nicht gesehenen Daten gute Ergebnisse zu erzielen. Overfitting tritt auf, wenn ein Modell zu stark an die Trainingsdaten angepasst ist und dadurch keine

allgemeinen Strukturen mehr erkennt. In diesem Fall hat sich das Modell die Trainingsdaten eingeprägt und ist daher nicht in der Lage, auf neuen Daten gute Ergebnisse zu erzielen. Ist die Kapazität des Modells zu groß, neigt das Neuronale Netz dazu, kleine Variationen in den Daten zu lernen und verliert dadurch die Generalisierung der Daten.
Beim Underfitting hingegen ist das Modell nicht in der Lage, eine zugrundeliegende Struktur in den Daten zu erkennen, was zu einer schlechten Leistung sowohl bei den Trainings- als auch bei den Testdaten führt. Beim Underfitting hat das Modell meist eine zu geringe Kapazität, um alle relevanten Informationen zu speichern.[vgl. 20]

2.3. Magnetresonanztomographie (MRT)

Die MRT, auch Kernspintomographie genannt, ist ein bildgebendes Verfahren zur Darstellung von Struktur und Funktion der Weichteile und Organe des Körpers. Mit einer MRT können Schnittbilder des Körpers oder einzelner Körperteile erstellt werden. Für die Bildgebung werden starke Magnetfelder verwendet, die jedoch für den menschlichen Körper unschädlich sind, da bei diesem Verfahren keine Ionisierende Strahlung verwendet wird. [vgl. 24]

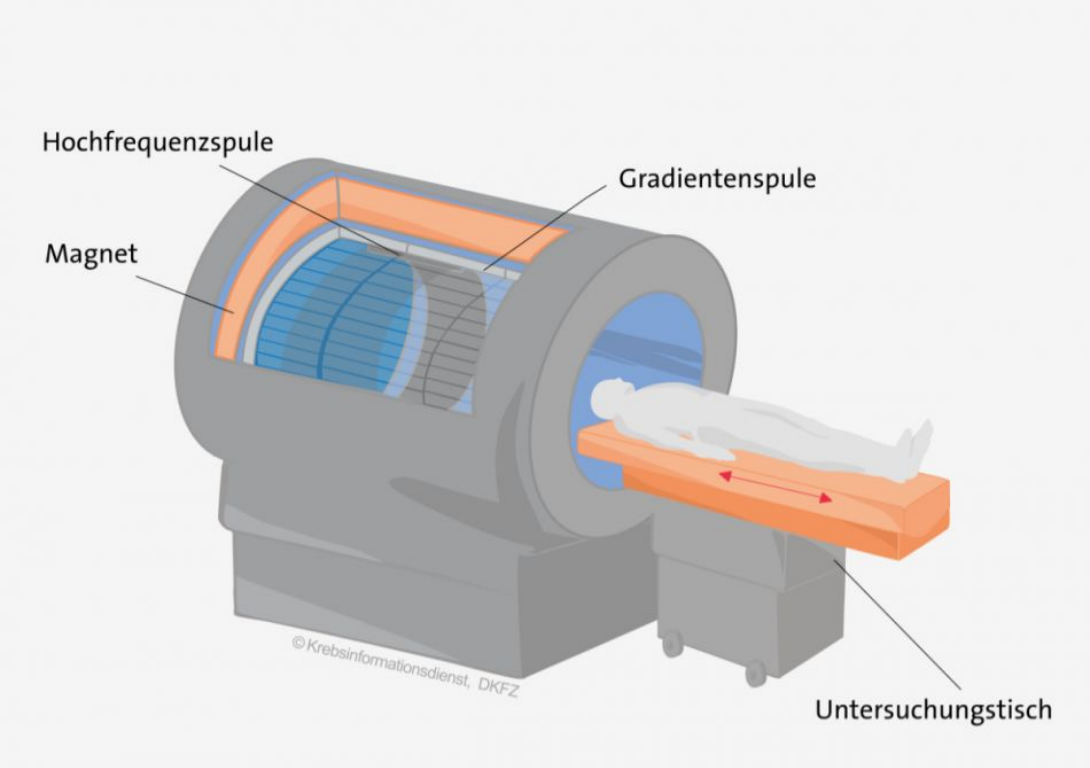


Abbildung 2.6.: Schematischer Aufbau eines Magnetresonanztomographen (Quelle: https://www.krebsinformationsdienst.de/untersuchung/bildgebung/kernspintomographie.php)

Der Magnetresonanztomograph besteht aus mehreren Komponenten:

- einem supraleitenden Hauptmagneten in einer Röhre, der ein starkes und konstanten Magnetfeld erzeugt. Ein supraleitender Magnet wird durch flüssigen Stickstoff gekühlt, sodass der Spulenwiderstand nahezu null wird. Dies ermöglicht die Erzeugung starker Magnetfelder.

- einem Gradientensystem, das zusätzliche Magnetfelder generiert, die je nach Position im Körper unterschiedlich sind und so eine räumliche Auflösung ermöglichen.
- einem Hochfrequenzsystem, welches aus einem Sende- und Empfangsspulensystem besteht. Dieses sendet Radiowellen zur Anregung der Wasserstoffatome aus, welche dadurch ihrer Ausrichtung verlieren.
- einem Computersystem, das geeignet ist für die Verarbeitung der Daten eines MRTs
- und einem Untersuchungstisch, auf welchem der Patient während der Untersuchung liegt

Das Verfahren beruht auf dem physikalischen Prinzip des Eigendrehimpulses, auch Spin genannt. Ein Proton dreht sich um seinen eigenen Schwerpunkt (Kernspin) und erzeugt dabei ein Magnetfeld. Auch die Wasserstoffkerne in unserem Körper besitzen einen Spin und verhalten sich wie kleine Magnete, die ein messbares Magnetfeld erzeugen.
Im Normalzustand sind die Kernspins ungeordnet, legt man jedoch ein starkes Magnetfeld an, richten sich die Kernspinachsen entlang der Magnetfeldlinien aus. Dabei führen sie eine Kreiselbewegung aus, bei der sie sich den Feldlinien immer weiter annähern, sich aber nie ganz mit ihnen ausrichten. Die Frequenz dieser Bewegung wird Larmorfrequenz genannt.
Die Ausrichtung der Kernspins allein reicht nicht aus, um ein Bild zu erzeugen, dazu muss ein hochfrequenter Impuls (HF-Impuls) senkrecht zum Hauptmagnetfeld angelegt werden. Der Puls muss eine bestimmte Frequenz haben, nämlich die der Kernspins, also die Larmorfrequenz. Durch diesen Puls synchronisieren sich die Protonen und einige der Kernspinachsen kippen um 90°, wobei sie Energie aufnehmen.[vgl. 25]
Nach dem Impuls kippen die Kernspins wieder in ihre ursprüngliche Position entlang des Hauptmagnetfeldes. Dabei wird die aufgenommene Energie in Form von Wärme wieder an die Umgebung abgegeben. Dieser Prozess der Wiederausrichtung wird als longitudinale Relaxation oder 'T1-Relaxation' bezeichnet. Er hängt von der Wärmeleitfähigkeit des Gewebes ab. Ein

weiterer Prozess, der beim Abschalten des HF-Impulses ausgelöst wird, ist der Verlust der synchronen Kreiselbewegung, wodurch die transversale Magnetisierung verloren geht. Unterschiedliche Gewebe können die transversale Magnetisierung unterschiedlich lange aufrechterhalten und erzeugen dadurch Kontraste in den Bildern, was auch als 'T2-Relaxation' bezeichnet wird. Im Wesentlichen misst man die Dauer der verschiedenen Prozesse und kann daraus auf die Gewebearten schließen und die Kontraste berechnen.[vgl. 26]

Zusätzliche Kontrastmittel können verwendet werden, um bestimmte Bereiche heller erscheinen zu lassen. Je nach Art des Kontrastmittels werden andere Bereiche dunkler oder heller dargestellt. Die Kontrastmittel selbst sind im Bild nicht sichtbar, wohl aber ihre Wirkung auf das umgebende Gewebe, so dass bei einer kontrastverstärkten T1-Gewichtung die Umgebung, z.B. Blut oder Tumore, heller erscheinen. [vgl. 25]

Es gibt noch weitere Methoden, die in der MRT eingesetzt werden können, um bestimmte Strukturen, Gewebearten oder Flüssigkeiten heller oder dunkler erscheinen zu lassen. Eine davon ist die "Fluid-attenuated inversion recovery", eine spezielle Methode zur Unterdrückung von Flüssigkeiten in den resultierenden Bildern. Sie unterdrückt zum Beispiel die Gehirnflüssigkeit, wodurch bestimmte Tumore besser sichtbar werden. Besitzt der Tumor jedoch einen hohen Wasseranteil, wird auch dieser im Bild unterdrückt. [vgl. 27]

3. Material und Methoden

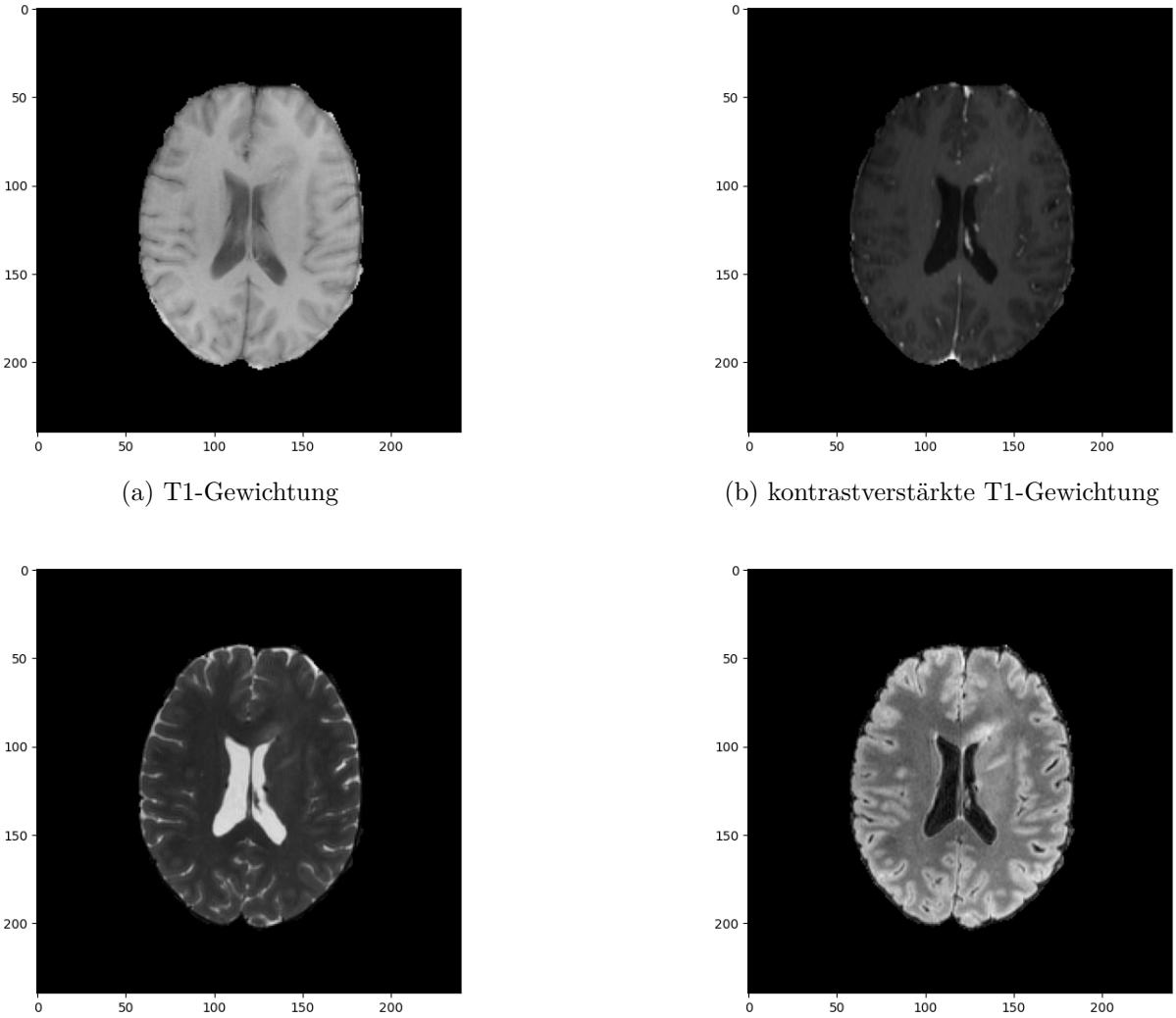
3.1. Datensatz: BraTS

Der BraTS Datensatz ist ein öffentlich zugänglicher Datensatz, der für die Entwicklung und Evaluierung von Modelle für die Segmentierung von Hirntumoren verwendet wird. Er wurde von der Radiological Society of North America in Zusammenarbeit mit der American Society of Neuroradiology(ASNR) und Medical Image Computing and Computer Assisted Interventions(MICCAI) erstellt und besteht aus über tausend MRT Bildern verschiedener Patienten mit Hirntumoren. [vgl. 28]

3.1.1. Beschreibung

Der Datensatz enthält multiinstitutionelle und multiparametrische MRT-Scans (mpMRT), die unter klinischen Standardbedingungen mit unterschiedlichen Geräten und Protokollen aufgenommen wurden. "Multiinstitutionell" bedeutet, dass die Daten für den Datensatz von mehreren medizinischen Einrichtungen geliefert und zusammengeführt wurden. "Multi-parametrisch" bedeutet, dass der Datensatz mehrere Parameter oder Merkmale enthält, die zur Segmentierung von Hirntumoren verwendet werden können.
Für die Segmentierungsaufgabe wurden alle Hirntumore mit den führenden BraTS-Algorithmen segmentiert und anschließend von freiwilligen Neuroradiologie-Experten mit unterschiedlicher Erfahrung weiter verfeinert. Die manuelle Beschriftung der Daten erfolgte nach einem strengen Protokoll, um konsistente Daten zu erhalten. Abschließend wurden die manuell verfeinerten Bilder von zertifizierten und erfahrenen Neuroradiologen mit

mehr als 15 Jahren Erfahrung in dieser Tätigkeit validiert. Die gelabelten Tumorregionen basieren auf den VASARI-Merkmalen[1], die für einen geschulten Radiologen sichtbar sind. Zu diesen Merkmalen gehören der Gadolinium-anreichernde Tumor (Kennzeichnung 4), das peritumorale ödematöse/invasive Gewebe (Kennzeichnung 2) und der nekrotische Tumorkern (Kennzeichnung 1). [vgl. 30]



(a) T1-Gewichtung

(b) kontrastverstärkte T1-Gewichtung

(c) T2-Gewichtung

(d) Flair-Gewichtung

Abbildung 3.1.: Die vier Modalitäten des MRT Scan im Datensatz (Quelle: Eigene Darstellung)

Der Datensatz besteht aus 1251 Einträgen, wobei jeder Eintrag aus T1, kontrastverstärkter T1, T2 und Flair Gewichtung (siehe Abb. 3.1) sowie

[1] VASARI-Merkmale(Visually Accessible Rembrandt Images) sind ein System zur konsistenten Beschreibung von Gliomen (Tumoren) anhand definierter visueller Merkmale [vgl. 29]

einer endgültigen Segmentierung des Hirntumors besteht. Die Dateien selbst sind im NIfTI-Dateiformat, was für "Neuroimaging Informatics Technology Initiative" steht und ein gängiges Format für medizinische Bilddaten ist. Das Dateiformat enthält die 3D-Volumendaten, die aus mehreren Bildebenen bestehen. Jedes Voxel, ein 3D-Bildpunkt, enthält einen numerischen Wert, der das Signal oder den Kontrast wiedergibt. Zusätzlich enthält das NIfTI-Dateiformat Metadaten wie räumliche Orientierung, Bildgröße, Raumkoordinaten und weitere Metadaten. [31] Die Bilder liegen in einer Größe von $150 \times 240 \times 240$ Pixeln vor, was, wie bereits erwähnt, den 3D-Volumendaten entspricht, die in 150 Schichten aufgeteilt sind, wobei jede Schicht einem Bild von 240×240 Pixeln entspricht.

3.1.2. Vorverarbeitung

Die Vorverarbeitung der Daten ist ein wichtiger Schritt, um die Qualität und Effizienz des Trainings eines neuronalen Netzes zu verbessern. Der zur Verfügung gestellte Datensatz wurde bereits einer standardmäßigen Vorverarbeitung unterzogen. Zunächst wurden die Daten vom DICOM-Dateiformat in das NIfTI-Dateiformat konvertiert, was zur einfacheren Handhabung der Daten dient. Außerdem wurde eine Anpassung an das gleiche anatomische Modell und die isotrope Auflösung vorgenommen. Abschließend wurde das Skull-Stripping durchgeführt[1] durchgeführt, um irrelevante Strukturen zu entfernen. [vgl. 30]

Normalisierung Nach diesen Schritten sind die Daten noch nicht für das Training eines neuronalen Netzes geeignet. Die Daten müssen zunächst normalisiert werden, was auf zwei Arten geschehen kann. Entweder werden die Werte auf einen Bereich von 0 bis 1 skaliert oder es wird die Z-Normierung

[1] Skull-Stripping ist der Prozess des Entfernens des Schädels und anderer nicht-hirnbezogener Strukturen aus dem Bild. [vgl. 32]

verwendet. Für eine einfache Skalierung der Werte wird folgende Gleichung verwendet

$$f(x) = \frac{x - min(x)}{max(x) - min(x)} \tag{3.1}$$

Dies führt zwar zu einer Vereinheitlichung der Werte, aber die Verteilung der Merkmale im Bild ist dadurch nicht normalisiert. Zu diesem Zweck wird die Z-Standardisierung verwendet, die den Mittelwert der Daten auf 0 und die Standardabweichung auf 1 setzt. Dadurch wird verhindert, dass bestimmte Merkmale gegenüber anderen dominieren. Außerdem wird die Genauigkeit des Modelle erhöht.[vgl. 20] Die Gleichung für die Z-Standardisierung lautet

$$f(x) = \frac{x - \mu_x}{\sigma_x} \tag{3.2}$$

wobei μ_x der Mittelwert und σ_x die Standardabweichung des Datensatzes ist. Diese Werte müssen vorab berechnet werden, in dem der Mittelwert und die Standardabweichung von jedem Bild berechnet wird, anschließend aufsummiert und durch die Anzahl der Bilder geteilt wird

$$\begin{aligned} \mu &= \frac{1}{n}\sum_{i=1}^{n} x_i \\ \sigma &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \end{aligned} \tag{3.3}$$

wobei n die Anzahl der Bilder und x_i ein Bild des Datensatzes ist.

Data Augmentation Im medizinischen Bereich sind die Bilddaten oft begrenzt, was das Training eines neuronalen Netzes erschwert. Eine Methode, dem entgegenzuwirken, ist die Datenanreicherung, bei der die Vielfalt des Trainingsdatensatzes erhöht wird. Verschiedene Augmentationen sind Bildbearbeitungsmethoden wie das Drehen, Skalieren, Beschneiden oder auch Spiegeln eines Bildes. Es ist darauf zu achten, dass alle Augmentierungen sowohl auf die Eingabebilder als auch auf die zugehörigen Segmentierungen

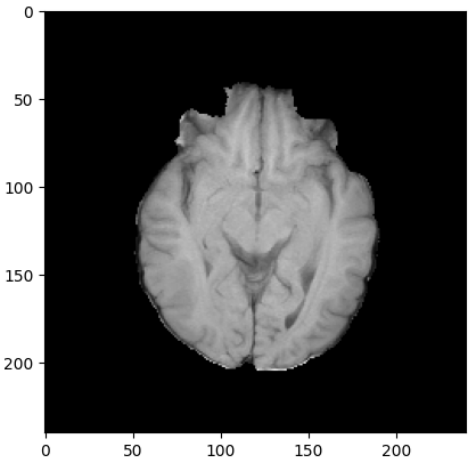
angewendet werden, da das Neuronale Netz sonst falsche Masken für die Eingaben lernt. [vgl. 33]

3D-Bild zu 2D-Bilder Die Daten liegen als 3D-Bilddaten vor, jedoch ist die Datenmenge mit 1251 Bildern recht gering. Um mehr Bilder aus den vorhandenen Daten zu erhalten, werden die 3D-Bilder in einzelne 2D-Bilder geschnitten. Wie bereits im Abschnitt 3.1.1 erwähnt, besteht ein 3D-Bild aus 150 Schichten mit einer Größe von 240×240 Pixeln. Jede einzelne Schicht wird als separates Bild gespeichert, um die Datenmenge zu erhöhen, so dass aus einem 3D-Bild 150 einzelne 2D-Bilder entstehen. Dadurch erhöht sich nicht nur die Datenmenge, sondern auch die Berechnungsgeschwindigkeit, da in der zweidimensionalen Darstellung weniger Rechenleistung und Speicherplatz benötigt wird. Die Umwandlung von 3D- in 2D-Bilder hat aber auch Nachteile, wie den Verlust des räumlichen Zusammenhangs zwischen den einzelnen Schichten. [vgl. 34]

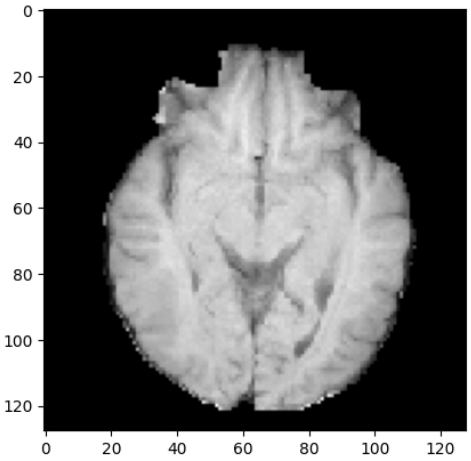
Größen Skalierung Bei einigen neuronalen Netzen kann die Bildgröße eine Rolle spielen. Es gibt Architekturen, bei denen die Bildgröße beliebig sein kann oder bei denen das Eingabebild eine bestimmte Größe haben muss, um verarbeitet werden zu können.
Die für die Segmentierung verwendete Architektur ist ein U-Net und wird im Kapitel 3.2.1 genauer beschrieben. Aufgrund der speziellen Architektur des U-Net müssen die Bilder eine bestimmte Größe haben. Der Grund für die spezifische Eingabegröße liegt in der Architektur des U-Net, genauer gesagt in den Down- und Upsampling-Pfaden. Durch Faltungs- und Max-Pooling-Schichten wird die Bildgröße auf der einen Seite schrittweise verkleinert und auf der anderen Seite wieder vergrößert.
Aufgrund des Designs des U-Net muss die Bildgröße ein Vielfaches der Anzahl der Downsampling-Layer sein. Die Größe der Bilder ist durch 2^N definiert, wobei N die Anzahl der Downsampling-Schichten ist. Wenn $N = 4$, dann muss die Eingabegröße der Bilder ein Vielfaches von $2^4 = 16$ sein. Wenn die Größe nicht korrekt ist, kann es zu Inkonsistenzen zwischen den

Größen kommen, was wiederum die Performance des Modells beeinträchtigt. [vgl. 35]

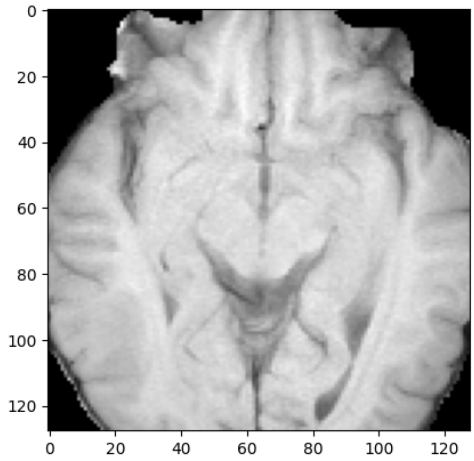
Um die Bilder auf eine geeignete Größe zu bringen, werden sie von der Mitte nach außen abgeschnitten. Die Größe des Bildausschnitts beträgt grundsätzlich 128 × 128 Pixel. Es kann jedoch vorkommen, dass Teile des Gehirns abgeschnitten werden (siehe Abb. 3.2c), daher wurde ein Puffer von 60 Pixeln pro Rand hinzugefügt. Nach dem Hinzufügen des Puffers hat das Bild nicht mehr die passende Größe und wird schließlich auf 128 × 128 Pixel skaliert (siehe Abb. 3.2b).



(a) Original Bild



(b) Ausschnitt mit Puffer



(c) Ausschnitt ohne Puffer

Abbildung 3.2.: Orginal Bild (T1-Gewichtung) im Vergleich zum Ausschnitt mit und ohne Puffer. (Quelle: Eigene Darstellung)

3.1.3. Aufteilung

Für eine erfolgreiche Modellentwicklung ist es wichtig, den Datensatz in verschiedene Teile aufzuteilen. Zu diesem Zweck wird der Datensatz in einen Trainingsdatensatz, einen Validierungsdatensatz und einen Testdatensatz unterteilt. Diese verschiedenen Datensätze ermöglichen es, die Leistungsfähigkeit und Generalisierbarkeit des Modells zu beurteilen.
Der Trainingsdatensatz wird verwendet, um das Modell zu trainieren, indem die internen Modellparameter anhand der Eingabedaten und der zugehörigen Labels angepasst werden. Je größer der Trainingsdatensatz ist, desto besser kann das Modell generalisieren, da es viele Daten mit unterschiedlichen Merkmalen lernt.
Der Validierungsdatensatz wird verwendet, um die Leistung des Modells während des Trainings zu bewerten. Anhand der Ergebnisse des Validierungsdatensatzes werden die Hyperparameter des Modells angepasst. Es ist darauf zu achten, dass sich dieser Datensatz nicht mit dem Trainingsdatensatz überschneidet, da es sonst zu falschen Ergebnissen kommt. [vgl. 23] Will man das Modell abschließend noch einmal evaluieren, wird häufig ein Testdatensatz verwendet, den das Modell noch nie gesehen hat. Nach Abschluss des Trainings kann der Testdatensatz verwendet werden, um die endgültige Leistungsfähigkeit des Modelles zu beurteilen.
Die Aufteilung der Daten sollte so gewählt werden, dass der Trainingsdatensatz genügend Daten enthält, um das neuronale Netz zu trainieren. Es ist darauf zu achten, dass ebenfalls genügend Daten für den Validierungs- und Testdatensatz verbleiben, um eine aussagekräftige Bewertung des Modells zu ermöglichen. Die Aufteilung wie sie hier in der Arbeit verwendet wird liegt bei $70 : 30$, wobei sich die 30% halbieren und je 15% für Validierungs- und Testdatensatz verwendet werden.

3.2. Modellentwicklung

Die Entwicklung des Modells ist ein anspruchsvolles Unterfangen, das im folgenden Kapitel näher erläutert wird. Zunächst wird die Architektur des Modells vorgestellt und wie es aufgebaut ist. Im nächsten Punkt wird erklärt, was Hyperparameter sind und was sie bewirken, gefolgt von den Problemen bei der Entwicklung und der Implementierung mit PyTorch. Abschließend wird der Trainingsprozess beschrieben.

3.2.1. Architektur des Neuronalen Netzes

Die gewählte Architektur ist ein U-Net, das seinen Namen seiner U-förmigen Struktur verdankt (siehe Abb. 3.3). Das U-Net wurde speziell für die Segmentierung biomedizinischer Bilder entwickelt und ist eine Sonderform des CNNs. Das U-Net ist speziell darauf ausgelegt, auch mit kleinen Datensätzen gute Ergebnisse zu erzielen. In der Medizin stehen oft nur begrenzt beschriftete Daten zur Verfügung, da der Aufwand für die Beschriftung der Daten hoch ist. Aufgrund dieser Tatsache ist U-Net für diese Zwecke besonders geeignet. Die folgende Beschreibung der Architektur basiert auf dem originalen U-Net und wird in dieser Arbeit als Basis für das Modell verwendet, welches im Abschnitt 3.2.3 genauer beschrieben wird.

Die Architektur des Netzes besteht aus zwei Teilen, dem Downsampling-Pfad, auch Encoder genannt, und dem Upsampling-Pfad, auch Decoder genannt. Der Encoder ähnelt einem klassischen CNN und besteht aus einer Reihe von Convolutional- und Max-Pooling-Operationen, aber es gibt keine vollständig vernetzten Schichten.

Der Encoder wird verwendet, um die Feature-Maps aus dem Bild zu extrahieren. Mit jedem Eingabeschritt des Encoders werden mehr Feature-Maps extrahiert und die räumliche Dimension des Bildes reduziert. Die Ausgabe wird an die nächsthöhere Schicht weitergeleitet, um so viele Details wie möglich auf verschiedenen Ebenen zu erhalten. Die Struktur besteht aus einer

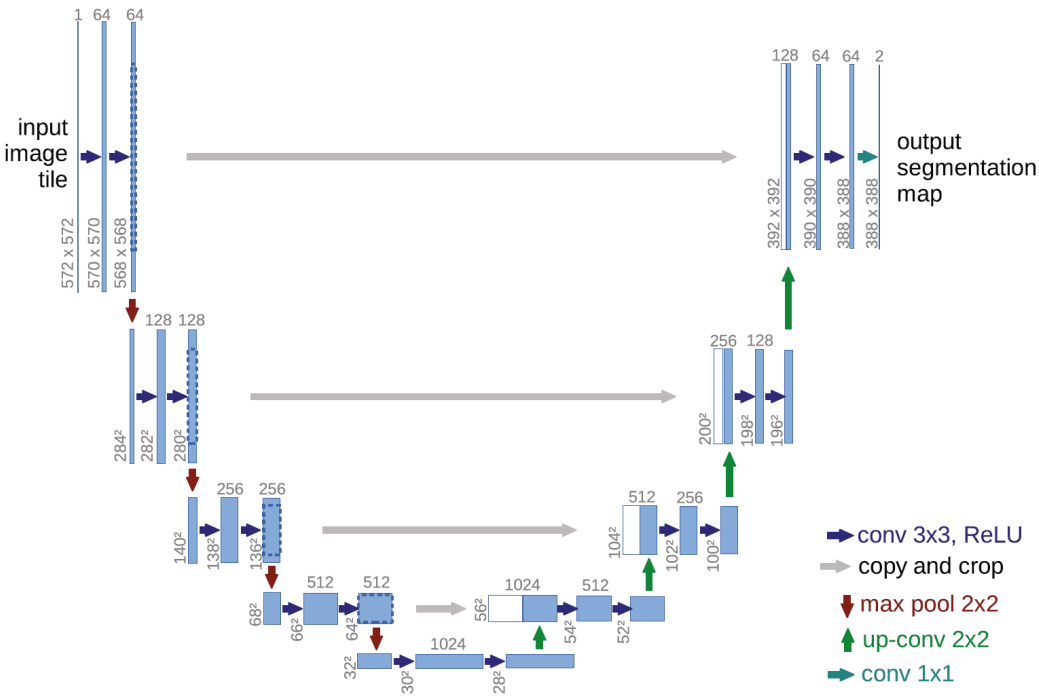


Abbildung 3.3.: Aufbau eines U-Nets mit einem Encoder und Decoder Abschnitt für die Extrahierung von Merkmalen und die Wiederherstellung des Original Bildes mit der fertigen Segmentierung (Quelle: [35])

Reihe von Doppelfaltungen mit einem 3x3-Filter, gefolgt von einer Max-Pooling-Schicht mit einer Filtergröße von 2x2 und einer Schrittweite von 2. In jedem Encoderschritt wird die Anzahl der Filter bzw. Feature-Maps in den Faltungsschichten verdoppelt.

Im Decoder wird die komprimierte Eingabe wieder auf die ursprüngliche Bildgröße skaliert, auch ”Upsampling“ genannt. Dies geschieht durch eine Abfolge von Upsampling-Layern, jeweils gefolgt von einer zweimaligen Faltung und einer Aktivierungsfunktion. Nach jedem Upsampling der Feature-Maps folgt eine Faltung mit einem 2×2-Filter, wodurch die Anzahl der Feature-Maps jeweils halbiert wird. Hierbei findet zusätzlich eine Verkettung der Feature-Maps aus dem entsprechenden Encoderabschnitt statt. Nach der 2×2 Faltung folgen immer zwei 3×3 Faltungen mit einer abschließenden ReLU Aktivierungsfunktion. Nach dem Decoder-Abschnitt folgt noch eine Faltung mit einem 1×1-Filter, der die Merkmale auf die gewünschte Anzahl von Klassen der Segmentierungsaufgabe abbildet. [vgl. 35]

3.2.2. Hyperparameter

Hyperparameter sind Parameter, deren Werte vor dem Training eines Modells festgelegt werden und die, von Ausnahmen abgesehen, während des Trainings nicht angepasst werden. Sie sind entscheidend für die Leistungsfähigkeit des Modells und müssen sorgfältig ausgewählt werden. Die Wahl der richtigen Hyperparameter ist oft eine Kunst, die Erfahrung und experimentelles Ausprobieren erfordert.

In der Praxis wird häufig eine Technik namens Hyperparametersuche oder -optimierung verwendet, bei der verschiedene Kombinationen von Hyperparametern systematisch ausprobiert werden, um diejenige zu finden, die die beste Leistung liefert. In diesem Abschnitt werden die wichtigsten Hyperparameter, die bei der Entwicklung eines Deep Learning Modells relevant sind, näher beschrieben.

Lernrate Die Lernrate ist ein entscheidender Hyperparameter in fast allen Optimierungsalgorithmen für neuronale Netze. Sie bestimmt, wie stark die Gewichte des Netzes in jedem Trainingsschritt angepasst werden. Eine zu hohe Lernrate kann dazu führen, dass das globale Minimum der Verlustfunktion übersprungen wird und das Modell möglicherweise nicht konvergiert. Bei einer zu niedrigen Lernrate wird der Fehler nur um sehr kleine Werte reduziert, was zu einer langsamen Konvergenz führt. Es kann auch vorkommen, dass der Fehler um einen Wert oszilliert und somit bei einer suboptimalen Lösung hängen bleibt. [vgl. 36]

Batch Größe Die Batch-Größe bezieht sich auf die Anzahl der Trainingsbeispiele, die das Netz gleichzeitig sieht, bevor es seine Gewichte aktualisiert. Eine größere Batch-Größe kann zu einer stabileren Aktualisierung der Gewichte führen, benötigt jedoch mehr Speicherplatz und verlangsamt das Training. Eine kleinere Batchgröße kann zu einem schnelleren, aber weniger stabilen Training führen. Übliche Batchgrößen reichen von 2, 8, 16 bis hin zu Hunderten von Bildern in einem Batch. [vgl. 37]

Anzahl der Epochen Eine Epoche bezeichnet einen Durchlauf des gesamten Datensatzes während des Trainings. Die Anzahl der Epochen, für die das Modell trainiert wird, beeinflusst die Fähigkeit des Modells, Muster aus den Daten zu lernen. Es ist wichtig, eine geeignete Anzahl von Epochen auszuwählen, um einen optimalen Trainingsprozess zu erhalten. Ist die Anzahl der Epochen zu gering, hat das Netz nicht genügend Zeit, um wichtige Muster zu lernen, was zu Underfitting führen kann. Ist die Anzahl der Epochen hingegen zu hoch, besteht die Gefahr, dass sich das Netz zu stark an die Trainingsdaten anpasst und es zu einem Overfitting kommt. [vgl. 20]

Architektur spezifische Parameter Im Kontext der U-Net Architektur gibt es auch spezifische Hyperparameter, die bei der Entwicklung des Modelles berücksichtigt werden müssen, wie z.B. die Anzahl und Größe der Filter in den Faltungsschichten, die Tiefe des Netzes und die Art der Aktivierungsfunktionen.

3.2.3. Implementierung

Die Implementierung des Modells zur Segmentierung von Hirntumoren erfolgt in Python mit PyTorch und basiert auf der in 3.2.1 beschriebenen U-Net-Architektur. Die Implementierung weicht in kleinen Teilen von der ursprünglichen U-Net Architektur ab. Die grobe Struktur des U-Net mit jeweils vier Down- und Upsampling-Layern wurde weitestgehend übernommen.

3.2.3.1. Padding

Durch die Faltungen innerhalb des neuronalen Netzes wird die Bildgröße reduziert, da keine Fülldaten verwendet werden. Um dem entgegenzuwirken, wird für eine bessere Implementierung an jedem Bildrand eine bestimmte

Anzahl von Pixeln hinzugefügt. Da die Bildgröße aufgrund der gegebenen Architektur bekannt ist (siehe Abschnitt 3.1.2), kann die Formel 2.6 zur Berechnung der Füllmenge P verwendet werden. Die Bildgröße W wird durch ein Vielfaches von 2^N definiert, wobei N die Anzahl der Downsampling-Schichten beschreibt. In diesem Fall ist $N = 4$ und die Eingabegröße der Bilder ist 128x128 Pixel, was ein Vielfaches von $2^N = 2^4 = 16$ ist. Die Schrittweite S und die Filtergröße F sind durch die Konstruktion des U-Netzes mit $S = 1$ und $F = 3$ bereits vorgegeben. Daraus ergibt sich die Füllmenge mit der Formel 2.6:

$$P = \frac{W \cdot S - 1 - W + F}{2} \tag{3.4}$$

$$P = \frac{2^N \cdot 1 - 1 - 2^N + 3}{2} = 1 \tag{3.5}$$

Da die Eingabegröße erhalten bleibt, ist die Implementierung aufgrund der einfachen Berechnung der Bildgröße für jede Schicht vorteilhaft.

3.2.3.2. Batch Normalisierung

Die Batch-Normalisierung ist eine Technik, die häufig im Deep Learning verwendet wird, um das Training zu beschleunigen und die Stabilität des Netzes zu erhöhen. Bei der Batch-Normalisierung werden die Eingaben für jede Schicht normalisiert, indem die durchschnittliche Aktivierung auf 0 und die Standardabweichung der Aktivierung auf 1 gesetzt wird.
Normalerweise wird der Mittelwert und die Standardabweichung für jede Batch berechnet, daher der Name Batch-Normalisierung. Es hat sich gezeigt, dass durch die Batch-Normalisierung höhere Lernraten verwendet werden können und das Training im Allgemeinen schneller und effektiver ist. Außerdem wird die Stabilität des Netzes erhöht und die Notwendigkeit einer sorgfältigen Initialisierung der Modellparameter entfällt. Die Schicht fungiert auch teilweise als eine Art Dropout, bei dem das Modell bestimmte Neuronen “vergisst” und neu initialisiert, um Overfitting zu vermeiden.[vgl.

38] PyTorch stellt bereits eine Batch-Normalisierungsschicht zur Verfügung, die für die Batch-Normalisierung verwendet werden kann. Diese Schicht wird nach jeder Faltung und vor der Aktivierungsfunktion angewendet.

3.2.4. Verlustfunktion

Die Wahl der richtigen Verlustfunktion ist entscheidend für den Erfolg eines Neuronalen Netzes. Bei der Klassifikation medizinischer Bilddaten gibt es einige Herausforderungen, die eine spezielle Auswahl der Verlustfunktion erfordern. Die Kreuzentropie (engl.: Cross Entropy) ist eine gängige Wahl für die Verlustfunktion bei Klassifizierungsproblemen. Sie misst, wie gut die geschätzte Wahrscheinlichkeitsverteilung des Modells mit der tatsächlichen Verteilung der Daten übereinstimmt. Wenn die Vorhersage des Modells genau mit den tatsächlichen Klassen übereinstimmt, ist die Kreuzentropie Null. Wenn die Vorhersage jedoch weit von der tatsächlichen Klasse entfernt ist, steigt der Verlust exponentiell an. [vgl. 39]
Obwohl die Kreuzentropie in den meisten Fällen recht effektiv ist, stößt sie insbesondere bei der Segmentierung medizinischer Bilder an ihre Grenzen. Eines der Hauptprobleme ist, dass die Kreuzentropie Pixel für Pixel berechnet wird und daher kein globales Verständnis der räumlichen Struktur des Bildes hat. Bei der Segmentierung von Gehirntumoren ist es jedoch wichtig, auch benachbarte Pixel zu betrachten, um die räumliche Struktur im Auge zu behalten.
Ein weiteres Problem ist die Unausgewogenheit der verschiedenen Klassen. Die zu segmentierenden Tumoren sind meist wesentlich kleiner als das Gesamtbild, was zu einer ungleichen Verteilung von Tumor- und Hintergrundklassen führt. Dadurch wird die Kreuzentropie stark durch die Hintergrundelemente beeinflusst und sinkt schnell gegen Null. [40]
Aus diesen Gründen wird häufig eine weitere Verlustfunktion verwendet. Eine der gebräuchlichsten Verlustfunktionen für die Segmentierung ist der Dice Loss. Dabei handelt es sich um eine regionsabhängige Verlustfunktion, die auch die räumliche Struktur des Bildes berücksichtigt, um den Fehler

zu minimieren. Der Dice Loss basiert auf dem Dice-Koeffizienten oder auch Sørensen-Dice-Koeffizienten, einer Metrik zur quantitativen Bewertung der Ähnlichkeit zwischen zwei Mengen, die im Abschnitt **??** näher beschrieben wird. Im Kontext der Bildsegmentierung wird der Dice-Koeffizient verwendet, um die Ähnlichkeit zwischen der vorhergesagten Segmentierung und der tatsächlichen Ground-Truth-Segmentierung zu messen. Der Dice-Koeffizient kann als Verlustfunktion dargestellt werden durch

$$1 - DSC(A, B) \tag{3.6}$$

oder alternativ durch den negativen Wert des Dice-Koeffizienten. [vgl. 41]

Es ist auch möglich, eine Kombination der beiden Verlustfunktionen zu verwenden. Bei dieser Methode wird jeder der Funktionen eine Gewichtung zugewiesen, wie stark sie den Verlust beeinflusst. Da die Kreuzentropie alleine nicht für die Segmentierung medizinischer Bilder geeignet ist, wohl aber für die allgemeine Segmentierung, wird sie häufig in Kombination mit dem Dice Loss verwendet. Durch die Verwendung des Würfelverlustes als zweite Verlustfunktion werden auch die unbalancierten Klassen berücksichtigt, was zu besseren Ergebnissen führt. [vgl. 42]

3.2.5. Probleme bei der Entwicklung

Bei der Entwicklung von Deep Learning Modellen für die Segmentierung medizinischer Bilder können verschiedene Probleme auftreten, die durch unterschiedliche Faktoren verursacht werden. Eines der häufigsten Probleme bei der Entwicklung ist der Mangel an Grafikkartenspeicher. Aufgrund der Bildgröße von MRT-Scans passen relativ wenige Bilder in den Speicher der Grafikkarte. Wenn man die Größe der Originalbilder beibehält, würde nur eine sehr kleine Größe für einen Batch funktionieren, ohne den Speicher zu überlasten.
Die Bilder wurden daher, wie im Abschnitt 3.1.2 beschrieben, in 2D-Bilder konvertiert und auf eine kleinere Bildgröße skaliert. Durch die Konvertierung

von 3D nach 2D kann nun eine deutlich größere Anzahl von Bildern pro Batch zum Training des Modell verwendet werden.
Ein weiteres Problem ist der Mangel an leistungsfähiger Hardware für schnellere Berechnungen. Die Hardware, die für das Training von Deep Learning Modellen verwendet wird, ist oft sehr teuer und verbraucht viel Strom. Die in der Entwicklung verwendete Grafikkarte ist daher nur eine Consumer-Grafikkarte, welche für das Training von relativ kleinen Modellen geeignet ist.

3.2.6. Training

Bevor ein Neuronales Netz trainiert werden kann, muss sichergestellt werden, dass die Daten ausreichend vorverarbeitet wurden und somit für das Training geeignet sind. Liegen die Daten in geeigneter Form vor, kann mit dem Training begonnen werden. Das Training eines Netzes läuft über mehrere Epochen und kann einige Zeit in Anspruch nehmen. In jeder Epoche werden alle Bilder des Trainingsdatensatzes einmal durch das Netz geschickt und anschließend die Leistung anhand eines Validierungsdatensatzes gemessen. Dieser Prozess wird für eine vorgewählte Anzahl von Epochen durchgeführt.

Der Prozess beginnt mit der Initialisierung des Modells, bei der alle internen Parameter auf einen zufälligen Wert gesetzt und während des Trainings angepasst werden. Anschließend werden die Trainingsdaten als Batches durch das Netz geschickt, auch Feedforward genannt. In diesem Teil werden verschiedene Berechnungen innerhalb des Netzes durchgeführt, um eine Vorhersage oder Segmentierung für die Daten zu erstellen.

Nachdem ein Batch durch das Netzwerk gelaufen ist und die Segmentierungen erstellt wurden, wird der Fehler berechnet. Dieser gibt an, wie weit die Ausgabe des Netzes von den tatsächlichen Ergebnissen abweicht. Anhand dieses Fehlers werden im nächsten Schritt die Parameter innerhalb des

Modells angepasst. Mit Hilfe der Rückwärtsfortpflanzung werden die Fehler der einzelnen Neuronen zum Gesamtfehler addiert. Auf dieser Basis können die Gewichte und der Bias der einzelnen Neuronen optimiert werden. [vgl. 20]

4. Experimente

In diesem Kapitel werden verschiedene Experimente durchgeführt, um die Auswirkungen verschiedener Parameter und Verarbeitungsschritte zu untersuchen. Insbesondere werden die Lernrate, die Losgröße und die Vorverarbeitung näher betrachtet. Jede dieser Komponenten spielt eine wesentliche Rolle beim Training des Modells. Bei der Untersuchung der Lernrate und der Losgröße werden verschiedene Werte getestet, um zu sehen, wie der Trainingsprozess abläuft. In der Vorverarbeitungskomponente wird das Thema Skalierung bzw. Bildausschnitt eine Rolle spielen.
Zunächst werden verschiedene Metriken vorgestellt, anhand derer die Auswirkungen der verschiedenen Parameter beurteilt werden können. Anschließend werden die einzelnen Experimente näher erläutert und deren Ergebnisse vorgestellt.

4.1. Metriken

Die Bewertung eines Modells ist ein wichtiger Schritt in der Entwicklung. Metriken werden verwendet, um ein Modell zu bewerten und seine Leistung zu beurteilen. Es gibt verschiedene Methoden bzw. Berechnungen, um die Performance zu untersuchen. Im Folgenden werden verschiedene Metriken und ihre Ergebnisse für die jeweiligen Modelle vorgestellt. Metriken sind bestimmte Funktionen, mit deren Hilfe die Leistungsfähigkeit eines Modelles bewertet werden kann. Sie berechnen die Übereinstimmung zwischen Vorhersage und tatsächlichem Ergebnis. Je nach Problemstellung müssen eine oder mehrere geeignete Metriken ausgewählt werden. Oft reicht eine einzelne Metrik für eine umfassende Bewertung nicht aus, weshalb in der Regel mehrere Metriken betrachtet werden.

4.1.1. Jaccard-Index

Der Jaccard-Index, auch Intersection over Union genannt, ist eine Metrik zur Bewertung der Ähnlichkeit oder Überlappung zweier Mengen A und B. Der Jaccard-Index misst das Verhältnis der Schnittmenge der beiden Mengen zu ihrer Vereinigung. In Bezug auf die Bildsegmentierung bedeutet dies, dass der Jaccard-Index das Verhältnis der Fläche des gemeinsamen Segments zur Fläche der vereinigten Segmente misst. Der Jaccard-Index kann einen Wert zwischen 0 und 1 annehmen, wobei 0 für keine Überlappung und 1 für perfekte Übereinstimmung steht. Mathematisch wird der Jaccard-Index folgendermaßen berechnet: [vgl. 43]

$$J(A, B) = \frac{|A \cup B|}{|A \cap B|} \tag{4.1}$$

4.1.2. Dice-Koeffizient

Der Dice-Koeffizient, auch bekannt als Sørensen-Dice-Koeffizient oder F1-Score, misst ebenfalls die Ähnlichkeit zwischen zwei Mengen oder Segmentierungen A und B. Der Dice-Koeffizient berechnet das Verhältnis zwischen dem Doppelten der Schnittmenge und der Summe der Größe der beiden Mengen. In Bezug auf die Bildsegmentierung misst der Dice-Koeffizient das Verhältnis der doppelten Fläche des gemeinsamen Segments zur Summe der Flächen beider Segmente. Wie der Jaccard-Index kann der Dice-Koeffizient Werte zwischen 0 und 1 annehmen, wobei 0 für keine Überlappung und 1 für perfekte Übereinstimmung steht. Mathematisch wird der Dice-Koeffizient wie folgt berechnet

$$DSC(A, B) = \frac{2|A \cup B|}{|A| + |B|} \tag{4.2}$$

Alternativ kann der Dice-Koeffizient auch mittels boolschen Daten dargestellt werden:

$$DSC = \frac{2TP}{2TP + FP + FN} \tag{4.3}$$

”True Positive“ (TP) enthält die Anzahl der korrekt als positiv segmentierten Pixel, die tatsächlich zur gewünschten Klasse gehören. ”False Positive“(FP) gibt die Anzahl der fälschlicherweise als positiv segmentierten Pixel, die tatsächlich nicht zur gewünschten Klasse gehören. “False Negative” (FP) hingegen beinhaltet die Anzahl der Pixel an, die tatsächlich zur gewünschten Klasse gehören, aber fälschlicherweise als negativ segmentiert wurden. [vgl. 44]

4.1.3. Pixel Accuracy

Die Pixelgenauigkeit, auch Pixel Accuracy genannt, ist eine in der Bildverarbeitung und Bildsegmentierung häufig verwendete Bewertungsmetrik. Sie misst den Prozentsatz der richtig klassifizierten Pixel in einem Bild oder einer Gruppe von Bildern. Die Pixelgenauigkeit wird berechnet, indem die vorhergesagten Labels jedes Pixels im Bild mit den Ground-Truth-Labels verglichen werden. Wenn das vorhergesagte Label mit dem Ground-Truth-Label für ein Pixel übereinstimmt, wird dies als korrekte Klassifikation betrachtet. Die Pixelgenauigkeit wird dann bestimmt, indem die Gesamtzahl der korrekt klassifizierten Pixel durch die Gesamtzahl der Pixel im Bild dividiert wird:

$$PA = \frac{\sum_{i=0}^{k} p_{ii}}{\sum_{i=0}^{k} \sum_{j=0}^{k} p_{ij}} \tag{4.4}$$

Hierbei sind alle p_{ii} als True Positive anzusehen und alle p_{ij} als False Positive bzw. False Negative. [vgl. 45]

4.2. Beschreibung Experimente

Im Folgenden werden die verschiedenen Experimente bzw. Modelle beschrieben und die Rahmenbedingungen festgelegt. Der verwendete Datensatz besteht aus 25000 2D-Bildern mit einer Größe von jeweils 128x128 Pixeln. Die Vorverarbeitung umfasst das Zusammenfügen der vier Modalitäten

sowie das Beschneiden und Skalieren der Einzelbilder. Als Standardwerte für die jeweils nicht zu untersuchenden Parameter werden die Werte aus der Tabelle 4.2 festgelegt.

Komponente	**Wert**
Batch Größe	32
Lernrate	0.01
Anzahl der Filter	64, 128, 256, 512
Vorverarbeitung	Auschnitt und Skalierung des Bildes
Optimierer	Adam
Verlustfunktion	Dice Loss
Epochen	20

Tabelle 4.2.: Standardwerte für das Training des Neuronalen Netzes

Die Auswertung der Modelle erfolgt über die drei genannten Evaluierungsfunktionen. Die Ausgänge des Neuronalen Netzes werden zunächst mit Hilfe der Softmax-Funktion in den Wertebereich zwischen 0 und 1 transformiert. Die neuen Werte stellen, wie bereits im Abschnitt 2.2.6 erwähnt, eine Wahrscheinlichkeitsverteilung dar. Anschließend werden diese Werte mit Hilfe der Argmax-Funktion[1] in die eigentlichen Tumorklassen überführt.

4.2.1. 1. Experiment - Lernrate

Der erste zu untersuchende Parameter ist die Lernrate. Sie bestimmt den Grad der Anpassung der Gewichte innerhalb des Modells. Eine sehr hohe Lernrate kann dazu führen, dass wichtige Minima der Verlustfunktion übersprungen werden, während eine zu niedrige Lernrate nur sehr langsam konvergiert. Im Folgenden wurde das Neuronale Netz mit verschiedenen Lernraten trainiert, wobei sowohl das Training als auch die abschließende

[1] Identifiziert die Kategorie mit dem höchsten Wert und gibt damit die Klassifikation an

Bewertung betrachtet wurden. Ausgehend vom Standardwert 0.01 wurde die Lernrate jeweils einmal um den Faktor $1 \cdot 10^{-1}$ verringert und erhöht, so dass sich für das Experiment Lernraten von 0.1 und 0.001 ergeben.

	Dice	Jaccard Index	Pixel Accuracy
LR0.1	81,49	79,45	98,79
Standard	88,44	85,85	99,22
LR0.001	**90,32**	**87,81**	**99,33**

Tabelle 4.3.: Ergebnisse des 1. Experiments mit der Lernrate in Prozent (Quelle: Eigene Darstellung)

4.2.2. 2. Experiment - Batch Größe

Der zweite Parameter ist die Batch Größe, welche angibt nach wie vielen Trainingsbeispielen die internen Parameter angepasst werden. Es wird untersucht, wie sich das Trainingsverhalten und die endgültige Leistung des Modells in Abhängigkeit der Batch Größe unterscheiden. Die zu untersuchenden Werte ergeben sich aus dem Standardwert, indem dieser einmal halbiert und einmal verdoppelt wird. Daraus ergeben sich somit die Batch Größen von 8 und 64.

	Dice	Jaccard Index	Pixel Accuracy
B8	83,03	81,08	98,88
Standard	**88,44**	**85,85**	**99,22**
B64	85,28	82,66	99,04

Tabelle 4.4.: Ergebnisse des 2. Experiments mit der Batch Größe in Prozent (Quelle: Eigene Darstellung)

4.2.3. 3. Experiment - Vorverarbeitung

Das letzte Experiment untersucht keinen Hyperparameter, sondern die Auswirkungen der Vorverarbeitung auf die Performance des Modells. Wie bereits im Abschnitt 3.1.2 erwähnt, müssen die Eingaben für das U-Net eine bestimmte Größe haben, um verarbeitet werden zu können. Die Größe der Bilder kann auf verschiedene Weise auf die gewünschte Eingabegröße gebracht werden. Die eine Methode ist eine einfache Skalierung von der Originalgröße auf die Zielgröße, wobei es zu Einbußen in der Bildqualität kommt, da benachbarte Pixel bzw. deren Farbwerte miteinander verrechnet werden. Die zweite Methode, die im Abschnitt 3.1.2 genauer beschrieben wird, besteht darin, zunächst die irrelevanten Teile des Bildes zu entfernen und dann auf die gewünschte Zielgröße zu skalieren. Bei dieser Methode enthält das Bild weniger irrelevante Informationen und erfährt einen geringeren Qualitätsverlust.

	Dice	Jaccard Index	Pixel Accuracy
Standard	**88,44**	**85,85**	**99,22**
Scaled	81,43	79,22	98,80

Tabelle 4.5.: Ergebnisse des 2. Experiments mit der Batch Größe in Prozent (Quelle: Eigene Darstellung)

4.3. Diskussion der Ergebnisse

Im folgenden Abschnitt werden die Ergebnisse des Experiments näher betrachtet. Sowohl die Trainingsverläufe als auch die Ergebnisse der Evaluierung des Modells, anhand des Testdatensatzes, werden näher beschrieben. Bei den Verlustkurven stellt die blaue Kurve immer das Referenzmodell mit den Standardwerten dar. Es wird auch auf die Auswirkungen der verschiedenen Parameter sowie auf die Aussagekraft der Metriken eingegangen.

4.3.1. 1. Experiment

Im ersten Experiment wurde die Lernrate in Bezug auf das Training und die Leistung des Modells untersucht. Zuerst wird der Trainingsverlauf betrachtet, der im Anhang A zu sehen ist, dann werden die verschiedenen Metriken herangezogen, um die Leistung der Modelle zu vergleichen.
Zunächst wird der Verlust mit Trainings- und Validierungsdatensatz betrachtet und interpretiert. Die Abbildungen A.1a und A.1b zeigen die Verlustkurven der verschiedenen Modelle mit dem Trainings- bzw. Validierungsdatensatz.
Die folgende Diskussion und Interpretation der Kurven bezieht sich zunächst, wenn nicht anders angegeben, auf die Abbildung. A.1a und den Trainingsdatensatz. Die grüne Kurve, die das Modell "LR0.1" mit der Lernrate 0.1 beschreibt, hat von Anfang an einen geringeren Verlust als das Referenzmodell, reduziert diesen aber über die Zeit nur minimal. Möglicherweise werden hier wichtige Minima der Verlustfunktion übersprungen, weshalb das Modell "LR0.1" relativ schnell mit einem hohen Verlust im konvergiert.
Im Vergleich dazu hat das Modell "LR0.001", die rote Kurve, mit einer Lernrate von 0.001, zu Beginn einen höheren Verlust, der über einige Epochen so bleibt und dann schlagartig abfällt. Der Verlust des Modells "LR0.001" ist geringer als der des Referenzmodells und hat am Ende der 20 Epochen einen geringeren Verlust als das Referenzmodell. Aufgrund der geringen Lernrate des Modells "LR0.001" sinkt der Verlust zu Beginn vergleichsweise langsam, da die internen Parameter nur minimal angepasst werden. Langfristig führen diese kleinen Schritte jedoch zu einem besseren Ergebnis, da wichtige Minima der Verlustfunktion weniger oder gar nicht übersprungen werden und das Modell somit besser trainiert wird. Diese Beobachtungen gelten somit auch für den Validierungsdatensatz und können analog interpretiert werden. Die Kurven in Abb. A.1b sind den Kurven in A.1a ähnlich. Auch hier wird deutlich, dass eine zu hohe Lernrate nicht zu optimalen Ergebnissen führt. So konvergiert der Verlust des Modells "LR0.1" bereits

ab der 14. Epoche bei einem relativ hohen Wert von ca. 0.25, während das Modell “LR0.001” und das Referenzmodell den Verlust weiter reduzieren.

Die Ergebnisse der Metriken (siehe Tabelle 4.3) unterstreichen den Verlauf des Trainings, wobei das Modell “LR0.001” die beste Performance aufweist. Ein Blick auf die Metriken zeigt einen Dice-Koeffizienten von über 90% und einen Jaccard-Index von ca. 88%, was angesichts des kurzen Trainings und der geringen Vorverarbeitung recht gute Ergebnisse sind. Die Pixel Accuracy ist hingegen nicht sehr aussagekräftig, da sich die Werte mit einer maximalen Differenz von 0.54% nur minimal unterscheiden.

4.3.2. 2. Experiment

Im zweiten Experiment wurde der Einfluss der Batchgröße Wie im ersten Experiment (siehe Abschnitt 4.3.1) werden zunächst die Trainingskurven (siehe Anhang B) betrachtet und anschließend die Performance anhand der genannten Metriken bewertet.
Zunächst wird der Verlust mit dem Trainingsdatensatz betrachtet, der in Abbildung B.1a dargestellt ist. Die Verlustkurven verlaufen in diesem Experiment sehr ähnlich, jedoch mit unterschiedlichen Verlustwerten. Die grüne Kurve, das Modell “B64” mit einer Batchgröße von 64, hat im Vergleich zum Referenzmodell einen höheren Verlust in den ersten Epochen, nähert sich aber in den letzten Epochen dem Referenzmodell an. Die rote Kurve, das Modell “B8” mit einer Losgröße von 8, hat im Vergleich zu den beiden anderen Modellen einen hohen Verlust.
Ein interessanter Aspekt ist der Verlauf der Verlustkurven in Bezug auf den Validierungsdatensatz, der in Abbildung B.1b dargestellt ist. Es fällt auf, dass die rote Kurve von Anfang an einen geringen Validierungsverlust aufweist, aber einen eher unruhigen Verlauf mit häufigeren steilen Anstiegen hat. Im Gegensatz dazu verläuft die grüne Kurve etwas gleichmäßiger, weist aber auch einige Steigungen auf. Allerdings steigt die grüne Kurve gegen Ende des Trainings wieder an, ab der 16. Epoche zeigt das Modell “B64” einen kontinuierlichen Anstieg der Verlustwerte, was auf ein Overfitting

des Modells hindeuten könnte. Das Referenzmodell mit der blauen Kurve hat dagegen einen ruhigeren Verlauf, zeigt aber auch einen signifikanten Sprung ab etwa der Hälfte des Verlaufs. Es zeigt sich, dass das Modell "B64" zu Beginn eine bessere Performance aufweist, später aber ein Overfitting entwickelt. Das Referenzmodell hingegen zeigt insgesamt eine stabilere Performance, obwohl es einen signifikanten Sprung des Verlustwertes im Vergleich zum Validierungsdatensatz aufweist.

Bei Betrachtung der Metriken in Tabelle 4.4 wird deutlich, dass das Referenzmodell im Vergleich zu den beiden anderen Modellen die besten Ergebnisse erzielt. Sowohl der Dice Koeffizient als auch der Jaccard Index zeigen eine Verbesserung von ca. 3% für das Referenzmodell gegenüber dem Modell "B64". Die Pixelgenauigkeit liefert in diesem Fall keine aussagekräftigen Informationen über die Leistungsfähigkeit der Modelle, da die Unterschiede minimal sind. Die Wahl einer Batchgröße von 32 bei einer Datenmenge von 25000 Bildern erweist sich somit als angemessen und führt zu guten Ergebnissen.

4.3.3. 3. Experiment

Das dritte Experiment untersucht die Auswirkungen der Vorverarbeitung auf die Qualität und die Leistung des Modells. Wie bereits im Abschnitt 4.2.3 erläutert, besteht der Unterschied zwischen den beiden Modellen lediglich in einem Vorverarbeitungsschritt. Die für dieses Experiment verwendeten Hyperparameter entsprechen den Standardwerten gemäß Tabelle 4.2. Ebenso werden zunächst die Trainingskurven betrachtet und abschließend die Leistungsfähigkeit der Modelle anhand der genannten Metriken bewertet. Die Verlustkurven für dieses Experiment sind im Anhang C dargestellt.
Die Verlustkurven für den Trainingsdatensatz sind in Abb. C.1a, ist für beide Modelle sehr ähnlich. Die rote Kurve, Modell "Scaled", hat außer zu Beginn immer einen höheren Verlust als das Referenzmodell. Der Verlauf der beiden Kurven ist nahezu identisch, jedoch unterscheiden sich die Werte

der beiden Kurven um ca. $0.05 - 0.1$. Die Ähnlichkeit der beiden Kurven zeigt, dass die Vorverarbeitung, in diesem Fall das Entfernen irrelevanter Informationen, einen positiven Effekt auf die Leistung hat, jedoch keinen größeren Einfluss auf den Trainingsverlauf hat.
Parallelen zwischen den beiden Modellen finden sich im Verlustverlauf in Bezug auf den Validierungsdatensatz, der in Abbildung 3 dargestellt ist. C.1b zu sehen ist. Insbesondere zu Beginn und am Ende des Trainings verlaufen die beiden Kurven sehr analog zueinander, lediglich die rote Kurve weist wie im Trainingsdatensatz höhere Werte auf. Auch hier zeigt sich, dass durch geeignete Vorverarbeitungsmaßnahmen eine bessere Performance während des Trainings erreicht werden kann.

Deutlich wird diese Schlussfolgerung auch anhand der Metriken in Tabelle 4.5. Dort hat das Referenzmodell einen deutlich höheren Dice-Koeffizienten und auch einen höheren Jaccard-Index. Der Unterschied ist in diesem Fall nicht nur ein kleiner Prozentsatz wie in den vorherigen Experimenten, sondern ca. 7%, was einen deutlichen Unterschied in der Leistung darstellt.

4.4. Fehleranalyse und Verbesserungen

Im Hinblick auf die Weiterentwicklung des Modells gilt es, potentielle Fehlerquellen zu beseitigen und Verbesserungen der Modellleistung zu erarbeiten. Die Tatsache, dass jedes Modell einen gewissen "Grundverlust" aufweist, deutet darauf hin, dass es noch Raum für Verbesserungen gibt. Ein wichtiger Aspekt, der bereits im Kapitel 3.1 behandelt wurde, ist die Vorverarbeitung der Daten. Verschiedene Faktoren wie Skalierung, Normalisierung und Datenaugmentierung können die Leistung des Modells beeinflussen.
Die Ergebnisse des dritten Experiments in Abschnitt 4.3.3 zeigen, wie wichtig es ist, irrelevante Informationen aus den Bildern zu entfernen. Die in dieser Arbeit verwendeten Vorverarbeitungsmethoden könnten weiter verbessert werden, indem die Daten besser zugeschnitten werden, um möglichst viele relevante Informationen in das neuronale Netz einzuspeisen. Außerdem

sollte eine höhere Auflösung der Bilder in Betracht gezogen werden, da die Skalierung zu Qualitätsverlusten führt und das Modell durch weniger verfügbare Pixel beeinträchtigt wird. In den Experimenten wurden nur 25.000 Bilder verwendet. Eine deutliche Erhöhung dieser Anzahl würde es dem Modell ermöglichen, mehr Bilder für das Training zu verwenden und somit mehr Strukturen zu erlernen. Aufgrund begrenzter Ressourcen und zeitlicher Beschränkungen wurde jedoch nur eine Teilmenge der verfügbaren Daten verwendet. Wird die gesamte Datenmenge von ca. 160.000 Bildern für das Training verwendet, sollte auch die Anzahl der Schichten im neuronalen Netz erhöht werden. Durch eine größere Anzahl von Schichten kann das Netz mehr Merkmalskarten erzeugen, die für die Segmentierung von Hirntumoren verwendet werden können. [vgl. 22]
Eine Verbesserung der Modellleistung kann auch durch eine optimierte Wahl der Hyperparameter erreicht werden. Dies kann nicht nur zu einer Leistungssteigerung, sondern auch zu einem effektiveren Training führen. kann auch zu einer Leistungssteigerung des Modells sowie zu einem effektiveren Training führen. Dies kann durch die systematische Auswahl verschiedener Kombinationen von Hyperparametern erreicht werden.

5. Resultate

Aus den Ergebnissen der vorangegangenen Experimente konnten die optimalen Bedingungen für das Training des Neuronalen Netzes ermittelt werden. Die Standardwerte in Tabelle 4.2 sind im Großen und Ganzen gut gewählt, jedoch hat sich aufgrund der Ergebnisse von Experiment 1 in Abschnitt 4.3.1 herausgestellt, dass eine geringere Lernrate langfristig einen besseren Erfolg zeigt. Aus diesem Grund wird die Lernrate für das endgültige Modell auf 0.001 gesetzt. Ebenso wird die Anzahl der Epochen mehr als verdoppelt und auf den Wert 50 gesetzt.

Der Trainingsverlauf, siehe Anhang D, des finalen Modells ist zu Beginn identisch mit dem Modell "LR0.001" aus dem ersten Experiment, welches sich als am besten erwiesen hat. Nach 20 Epochen nimmt der Verlust auf dem Trainingsdatensatz bis zum Ende jedoch nur minimal ab. Der Verlauf auf dem Validierungsdatensatz ist ebenfalls identisch mit dem des Modells "LR0.001", jedoch konnte durch das längere Training der Verlust weiter auf Werte unter 0.1 reduziert werden.

Die Ergebnisse des finalen Modells in Tabelle 5.1 zeigen eine deutliche Verbesserung gegenüber dem Standardmodell. Der Dice Koeffizient erreichte Werte von über 92% und der Jaccard Index Werte von über 90%. Die Pixel Accuracy hat sich ebenfalls verbessert, ist aber, wie bereits in den vorherigen Abschnitten erwähnt, nicht signifikant. Generell ist eine Verbesserung von ca. 5% durch die geringere Lernrate und das längere Training festzustellen.

Die Ergebnisse des Neuronalen Netzes zeigen, dass die Entwicklung durchaus machbar ist und teilweise gute Ergebnisse liefern kann. Die Entwicklung eines Modells ist relativ einfach, sofern bereits ein geeigneter Datensatz vorhanden ist.

	Dice	Jaccard Index	Pixel Accuracy
Final	**92,57**	**90,30**	**99,56**
Standard	88,44	85,85	99,22

Tabelle 5.1.: Ergebnisse des Finalen Modells (Quelle: Eigene Darstellung)

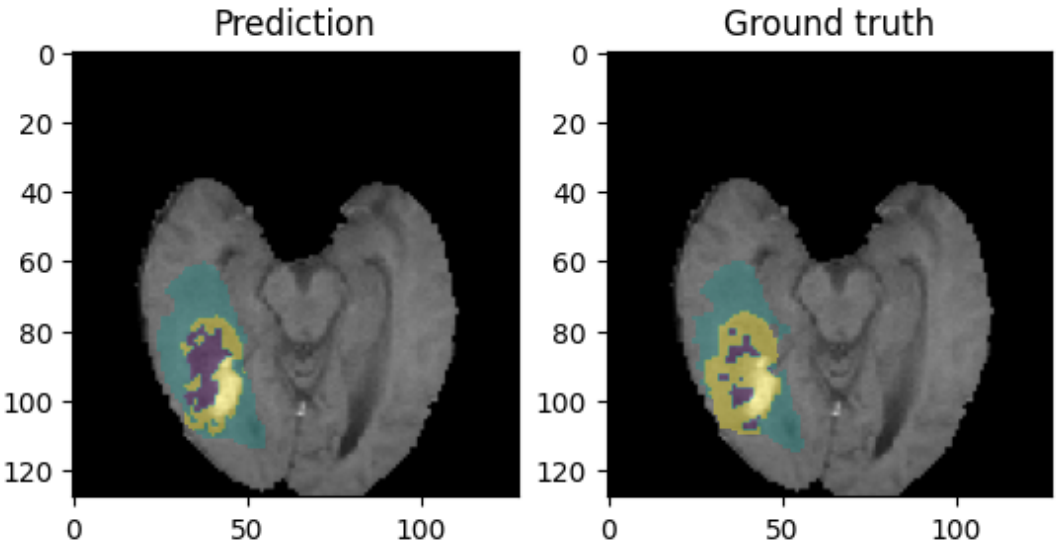


Abbildung 5.1.: Links die Segmentierung vom finalen Modell, rechts die wahrhaftige Segmentierung. (Quelle: Eigene Darstellung)

Dabei ist zu beachten, dass die Vorverarbeitung der Daten für eine gute Performance des Modells unerlässlich ist, wie die Ergebnisse des dritten Experiments in Abschnitt 4.3.3 gezeigt haben. Dazu müssen die Daten in eine geeignete Form gebracht werden und sollten möglichst nur relevante Informationen enthalten. Wichtig ist auch eine geeignete Wahl der Hyperparameter, da diese das Training stark beeinflussen. Zudem hängt es auch von den Hyperparametern selbst ab, welche Auswirkungen eine Veränderung mit sich bringt.
Das Modell hat immer einen gewissen Verlust, der durch geeignete Methoden weiter reduziert werden kann. Der Verlust macht sich auch bei der Auswertung der Bilder bemerkbar, wo das endgültige Modell in Teilen von der korrekten Segmentierung abweicht, wie Abbildung 5.1 zeigt.

6. Zusammenfassung und Ausblick

Die vorliegende Studienarbeit hat sich mit dem Einsatz von Deep Learning, insbesondere mit der U-Net-Architektur, für die Segmentierung von Hirntumoren in MRT-Bildern beschäftigt. Durch den Einsatz künstlicher neuronaler Netze konnte eine Methode zur automatisierten Segmentierung erarbeitet werden.

Im Rahmen der Arbeit wurden zunächst die theoretischen Grundlagen des Deep Learning und speziell von CNN und der U-Net-Architektur erläutert. Die U-Net-Architektur hat sich dabei als geeignet für die Segmentierungsaufgaben in der medizinischen Bildgebung erwiesen. Sie verbindet die Vorteile von Feature-Erkennung und -Lokalisierung, was zu guten Segmentierungsergebnissen führt.

Der verwendete Datensatz, BraTS, bietet eine Vielzahl an MRT-Bildern mit unterschiedlichen Hirntumoren, welche für das Training eines Modells verwendet werden können. Bei der Datenvorverarbeitung wurde auf Aspekte wie Skalierung und Aufteilung eingegangen, um eine optimale Vorbereitung für das Training des neuronalen Netzes zu gewährleisten. Im Zuge der Modellentwicklung wurden verschiedene Hyperparameter evaluiert und eine effiziente Implementierung des Modells vorgestellt.

Die durchgeführten Experimente zeigen die Auswirkungen der verschiedenen Hyperparameter und welche Parameterkombinationen für das Training des Modells am besten geeignet sind. Die Evaluierung der Modelle erfolgte anhand verschiedener Metriken und lieferte wertvolle Ergebnisse. Ebenso zeigte die Fehleranalyse, dass es noch Verbesserungs- und Optimierungsmöglichkeiten gibt.

Die Aussichten für weitere Verbesserungen in diesem Projekt sind vielversprechend. Die Performance des Modells könnte durch die Verwendung anderer Modellarchitekturen oder eines größeren Datensatzes optimiert werden. Außerdem könnten zusätzliche Datensätze, z.B. auch ältere BraTS-Datensätze, in das Training einbezogen werden, um die Generalisierbarkeit des Modells weiter zu verbessern.

Abschließend kann festgehalten werden, dass trotz der in dieser Arbeit erzielten Ergebnisse noch viele weitere Themen und Verbesserungen im Hinblick auf die Segmentierung von Hirntumoren bearbeitet werden können, um bessere Ergebnisse zu erzielen.

Literatur

1. O.V. *Was ist ein Algorithmus?* [online]. 2022-01-16. [besucht am 22. Feb. 2023]. Abger. unter: https://www.itwissen.info/Algorithmus-algorithm.html.
2. O.V. *Ionisierende Strahlung* [online]. 2020-06-05. [besucht am 13. März 2023]. Abger. unter: https://www.bmuv.de/themen/atomenergie-strahlenschutz/strahlenschutz/ionisierende-strahlung.
3. O.V. *Was ist ein Machine Learning Modell?* [online]. [besucht am 26. Feb. 2023]. Abger. unter: https://learn.microsoft.com/en-us/windows/ai/windows-ml/what-is-a-machine-learning-model.
4. HEINRICHS, Jan-Hendrik; CASPERS, Svenja; SCHNITZLER, Alfons; SEITZ, Frederike. *Bildgebung in den Neurowissenschaften*. Verlag Karl Alber, 2022. Auch verfügbar unter: https://www.ebook.de/de/product/45689204/jan_hendrik_heinrichs_svenja_caspers_alfons_schnitzler_frederike_seitz_bildgebung_in_den_neurowissenschaften.html.
5. RAMSUNDAR, Bharath; EASTMAN, Peter; WALTERS, Patrick; PANDE, Vijay. *Deep Learning für die Biowissenschaften*. Dpunkt.Verlag GmbH, 2020. Auch verfügbar unter: https://www.ebook.de/de/product/38103997/bharath_ramsundar_peter_eastman_patrick_walters_vijay_pande_deep_learning_fuer_die_biowissenschaften.html.
6. O.V. *PyTorch* [online]. [besucht am 2. Apr. 2023]. Abger. unter: https://pytorch.org/.

7. ERTEL, Wolfgang. *Grundkurs Künstliche Intelligenz*. Springer Fachmedien Wiesbaden, 2021. Auch verfügbar unter: https://www.ebook.de/de/product/41631013/wolfgang_ertel_grundkurs_kuenstliche_intelligenz.html.

8. POSTHOFF, Christian. *Computer und Künstliche Intelligenz: Vergangenheit - Gegenwart - Zukunft*. Springer Fachmedien Wiesbaden GmbH, 2022.

9. LANG, Volker. *Digitale Kompetenz*. Springer Berlin Heidelberg, 2023. Auch verfügbar unter: https://www.ebook.de/de/product/45515417/volker_lang_digitale_kompetenz.html.

10. ONLINE, Brockhaus Enzyklopädie. *Künstliche Intelligenz* [online]. [besucht am 20. Feb. 2023]. Abger. unter: https://brockhaus.de/ecs/enzy/article/kunstliche-intelligenz.

11. ALPAYDIN, Ethem. *Introduction to Machine Learning*. The MIT Press, 2014.

12. AGOSTINELLI, Andrea; DENK, Timo I.; BORSOS, Zalán; ENGEL, Jesse; VERZETTI, Mauro; CAILLON, Antoine; HUANG, Qingqing; JANSEN, Aren; ROBERTS, Adam; TAGLIASACCHI, Marco; SHARIFI, Matt; ZEGHIDOUR, Neil; FRANK, Christian. *MusicLM: Generating Music From Text*. arXiv, 2023. Auch verfügbar unter: https://arxiv.org/abs/2301.11325.

13. FROCHTE, Jörg. *Maschinelles Lernen*. Hanser, Carl GmbH + Co., 2020. Auch verfügbar unter: https://www.ebook.de/de/product/39878236/joerg_frochte_maschinelles_lernen.html.

14. O.V. *Supervised Learning* [online]. [besucht am 26. Feb. 2023]. Abger. unter: https://www.ibm.com/de-de/topics/supervised-learning.

15. JÜRGEN CLEVE, Uwe Lämmel. *Künstliche Intelligenz*. Hanser, Carl GmbH + Co., 2020. Auch verfügbar unter: https://www.ebook.de/de/product/38679436/juergen_cleve_uwe_laemmel_kuenstliche_intelligenz.html.

16. SCHERER, Andreas. *Neuronale Netze Grundlagen und Anwendungen: Grundlagen und Anwendungen*. Vieweg+Teubner Verlag, 1997.

17. CHOO, Kenny; GREPLOVA, Eliska; FISCHER, Mark H.; NEUPERT, Titus. *Machine Learning kompakt*. Springer Fachmedien Wiesbaden, 2020.

18. RUMELHART, David E.; HINTON, Geoffrey E.; WILLIAMS, Ronald J. Learning representations by back-propagating errors. *Nature*. 1986, **323**(6088), 533–536.

19. FRICK, Detlev. *Data Science Konzepte, Erfahrungen, Fallstudien und Praxis: Konzepte, Erfahrungen, Fallstudien und Praxis*. Springer Fachmedien Wiesbaden GmbH, 2021.

20. IAN GOODFELLOW Yoshua Bengio, Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

21. KINGMA, Diederik P.; BA, Jimmy. *Adam: A Method for Stochastic Optimization*. arXiv, 2014.

22. TEOH, Teik Toe. *Convolutional Neural Networks for Medical Applications*. Springer, 2023.

23. WEIDMAN, Seth. *Deep Learning - Grundlagen und Implementierung : Neuronale Netze mit Python und PyTorch programmieren: Neuronale Netze mit Python und PyTorch programmieren*. Dpunkt.Verlag GmbH, 2020.

24. KRAMME, Rüdiger. *Medizintechnik Verfahren - Systeme - Informationsverarbeitung: Verfahren - Systeme - Informationsverarbeitung*. Springer Berlin / Heidelberg, 2016.

25. CHRISTOPH PABST, Dr. med. *Magnetresonanz-Tomographie: Lernskript für Mediziner* [Grundlagen der Magnetresonanz-Tomographie] [online]. 2013. [besucht am 13. März 2023]. Abger. unter: https://www.ukgm.de/ugm_2/deu/umr_rdi/Teaser/Grundlagen_der_Magnetresonanztomographie_MRT_2013.pdf.

26. ANTWERPES, Dr. Frank. *Kernspintomographie* [online]. 2022-11-05. [besucht am 13. März 2023]. Abger. unter: https://flexikon.doccheck.com/de/Kernspintomographie.

27. GIZEWSKI, E. RM. R. Epidermoid oder Arachnoidalzyste: CISS, FLAIR und Diffusionsbilder als Ausweg aus dem diagnostischen Dilemma. *RöFo - Fortschritte auf dem Gebiet der Röntgenstrahlen und der bildgebenden Verfahren*. 2001, **173**(1), 77–78.

28. O.V. *Brain Tumor AI Challenge (2021)* [online]. 2021. [besucht am 24. Apr. 2023]. Abger. unter: https://www.rsna.org/education/ai-resources-and-training/ai-image-challenge/brain-tumor-ai-challenge-2021.

29. NAM, Yeo Kyung; PARK, Ji Eun; PARK, Seo Young; LEE, Minkyoung; KIM, Minjae; NAM, Soo Jung; KIM, Ho Sung. Reproducible imaging-based prediction of molecular subtype and risk stratification of gliomas across different experience levels using a structured reporting system. *European Radiology*. 2021, **31**(10), 7374–7385.

30. BAID, Ujjwal u. a. The RSNA-ASNR-MICCAI BraTS 2021 Benchmark on Brain Tumor Segmentation and Radiogenomic Classification. 2021. Abger. unter arXiv: 2107.02314 [cs.CV].

31. GROUP, Data Format Working. *NIfTI (Neuroimaging Informatics Technology Initiative)* [online]. 2013-12. [besucht am 27. Apr. 2023]. Abger. unter: https://nifti.nimh.nih.gov/.

32. SWIEBOCKA-WIEK, Joanna. Skull Stripping for MRI Images Using Morphological Operators. In: SAEED Khalidand Homenda, Władysław (Hrsg.). *Computer Information Systems and Industrial Management*. Cham: Springer International Publishing, 2016.

33. SHORTEN, Connor; KHOSHGOFTAAR, Taghi M. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*. 2019, **6**(1).

34. STEVENS, Eli Eli; ANTIGA, Luca Luca; VIEHMANN, Thomas Thomas. *Deep Learning with Pytorch*. Manning Publications Company, 2020.

35. RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. U-Net: Convolutional Networks for Biomedical Image Segmentation. 2015. Abger. unter arXiv: 1505.04597 [cs.CV].

36. PFANNSTIEL, Mario A. (Hrsg.). *Künstliche Intelligenz im Gesundheitswesen*. Springer Fachmedien Wiesbaden, 2022.

37. YU, Tong; ZHU, Hong. Hyper-Parameter Optimization: A Review of Algorithms and Applications. 2020. Abger. unter arXiv: 2003.05689 [cs.LG].

38. IOFFE, Sergey; SZEGEDY, Christian. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015. Abger. unter arXiv: 1502.03167 [cs.LG].

39. MURPHY, Kevin P. *Machine learning a probabilistic perspective: a probabilistic perspective*. MIT Press, 2012.

40. YEUNG, Michael; SALA, Evis; SCHÖNLIEB, Carola-Bibiane; RUNDO, Leonardo. *Unified Focal loss: Generalising Dice and cross entropy-based losses to handle class imbalanced medical image segmentation*. arXiv, 2021.

41. SUDRE, Carole H; LI, Wenqi; VERCAUTEREN, Tom; OURSELIN, Sébastien; CARDOSO, M. Jorge. Generalised Dice overlap as a deep learning loss function for highly unbalanced segmentations. 2017, 240–248. Abger. unter arXiv: 1707.03237 [cs.CV].

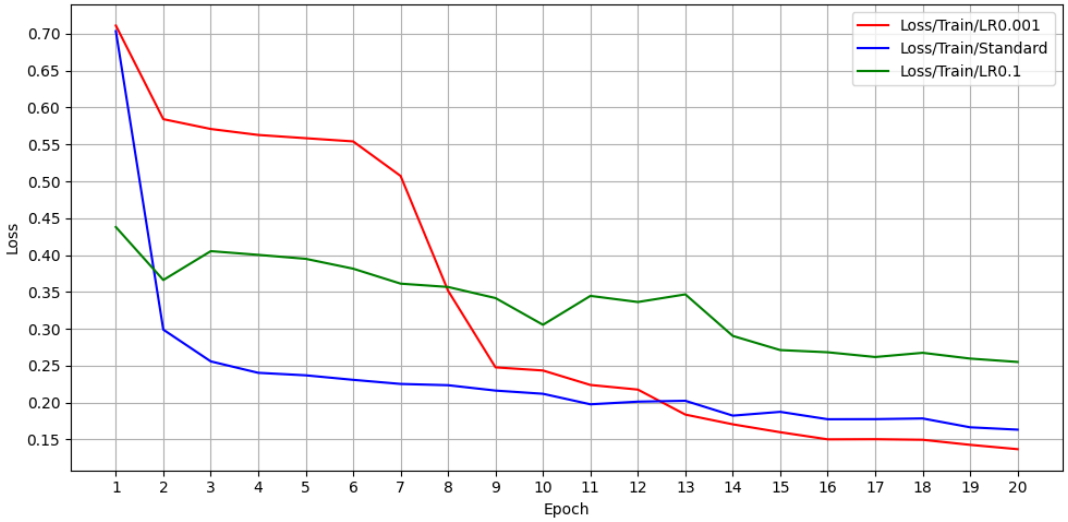
42. JADON, Shruti. A survey of loss functions for semantic segmentation. *2020 IEEE International Conference on Computational Intelligence in Bioinformatics and Computational Biology*. 2020. Abger. unter arXiv: 2006.14822 [eess.IV].

43. GARCIA-GARCIA, Alberto; ORTS-ESCOLANO, Sergio; OPREA, Sergiu; VILLENA-MARTINEZ, Victor; GARCIA-RODRIGUEZ, Jose. A Review on Deep Learning Techniques Applied to Semantic Segmentation. 2017. Abger. unter arXiv: 1704.06857 [cs.CV].

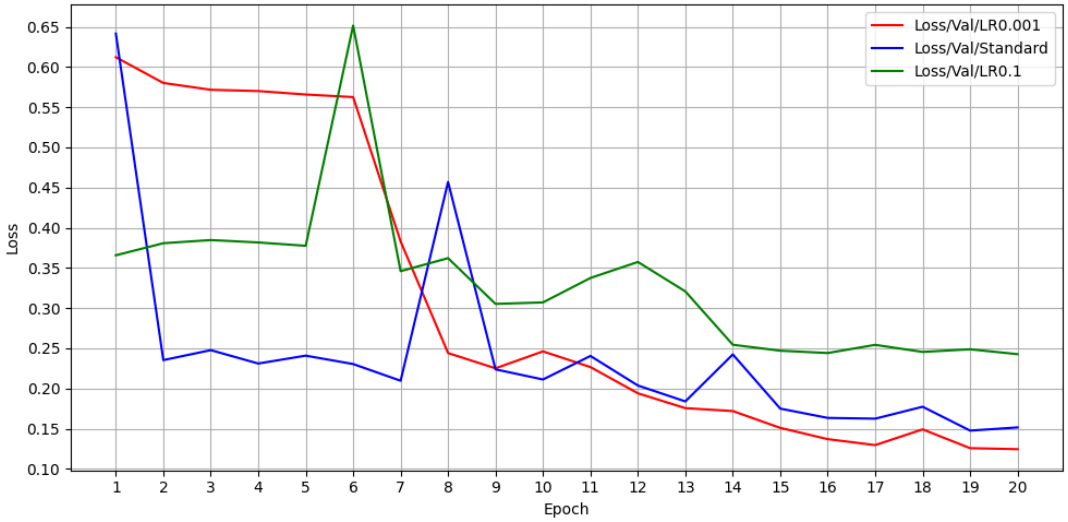
44. DICE, Lee R. Measures of the Amount of Ecologic Association Between Species. *Ecology*. 1945, **26**(3), 297–302.

45. HURTADO, Juana Valeria; VALADA, Abhinav. Semantic scene segmentation for robotics. In: *Deep Learning for Robot Perception and Cognition*. Elsevier, 2022, S. 279–311.

A. Experiment 1 - Verlustkurven



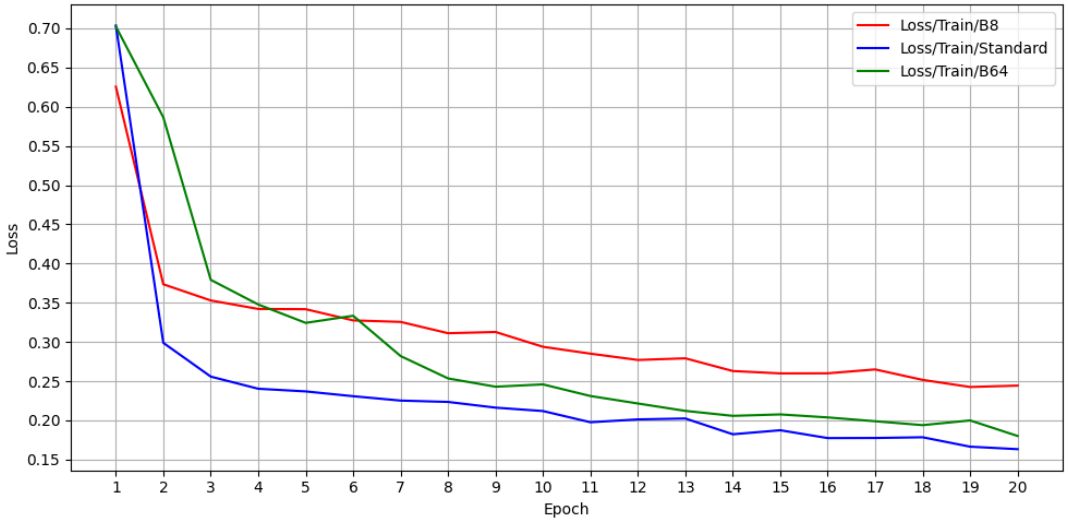
(a) Trainingsverlust



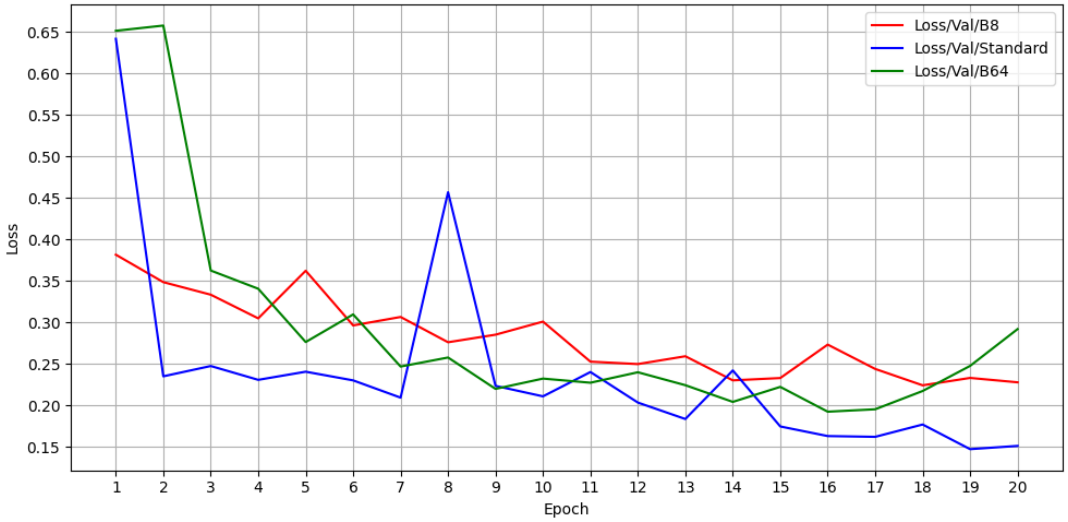
(b) Validierungsverlust

Abbildung A.1.: Experiment 1 - Verlustkurven von Training und Validierung über 20 Epochen, mit den Lernraten 0.1, 0.01(Standard) und 0.001 (Quelle: Eigene Darstellung)

B. Experiment 2 - Verlustkurven



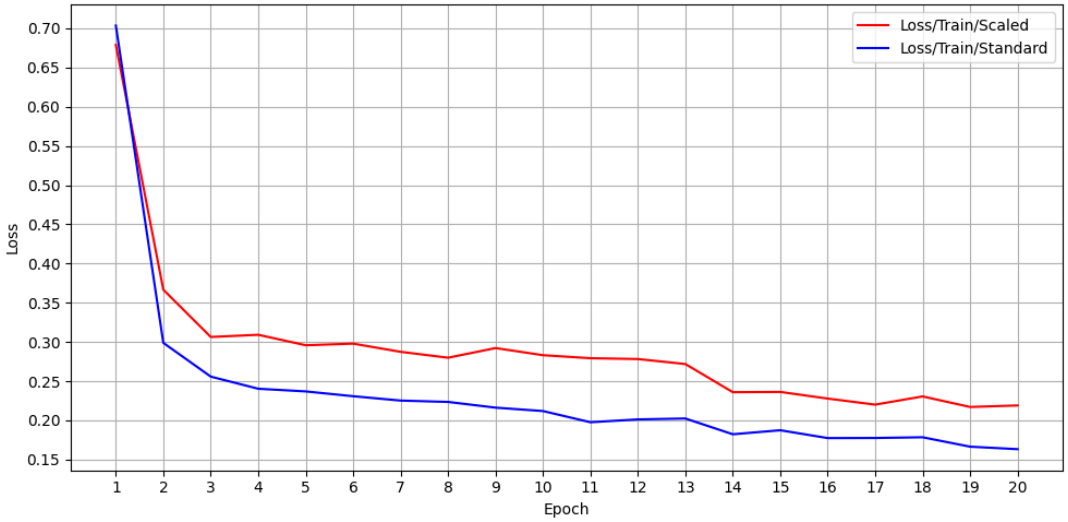
(a) Trainingsverlust



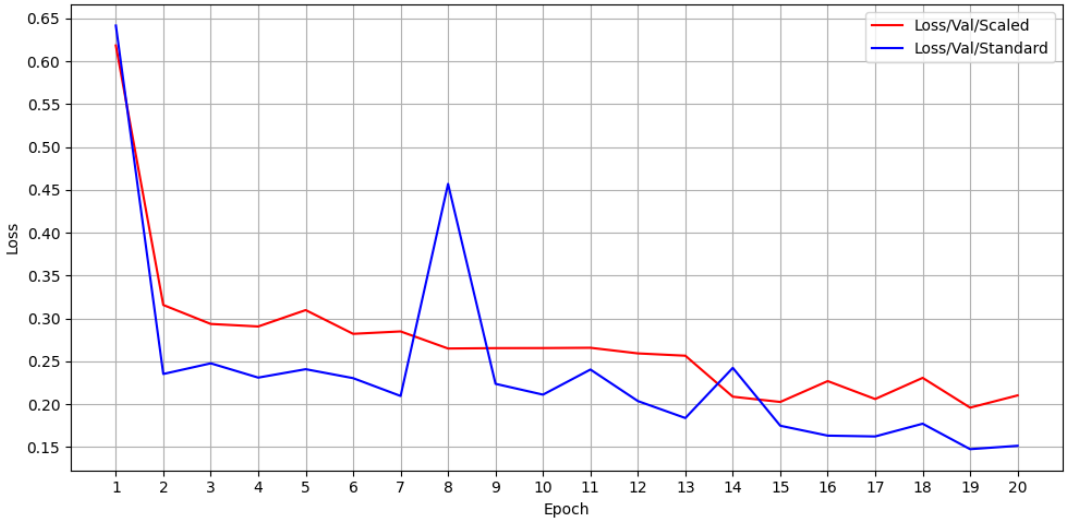
(b) Validierungsverlust

Abbildung B.1.: Experiment 2 - Verlustkurven von Training und Validierung über 20 Epochen, mit den Batch Größen 8, 32(Standard) und 64 (Quelle: Eigene Darstellung)

C. Experiment 3 - Verlustkurven



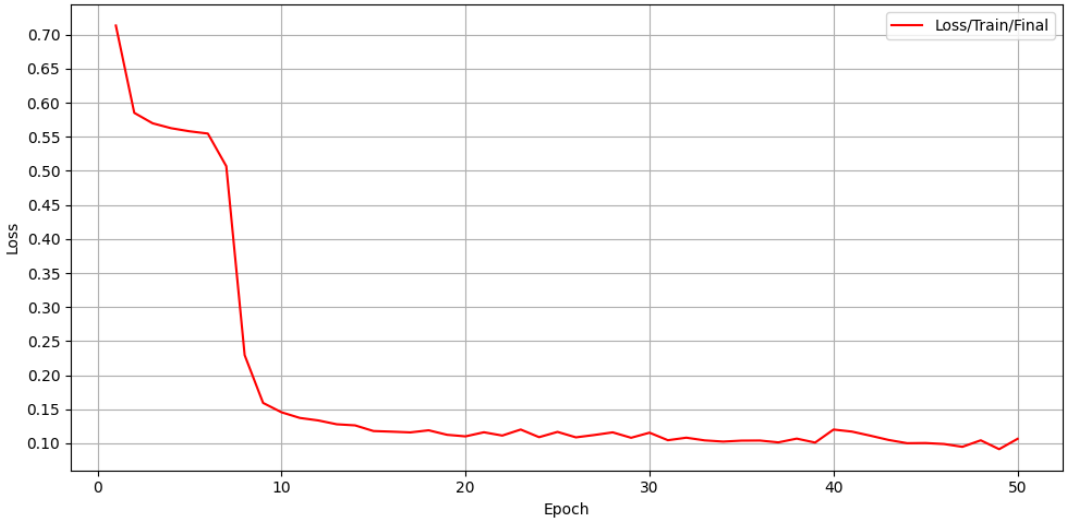
(a) Trainingsverlust



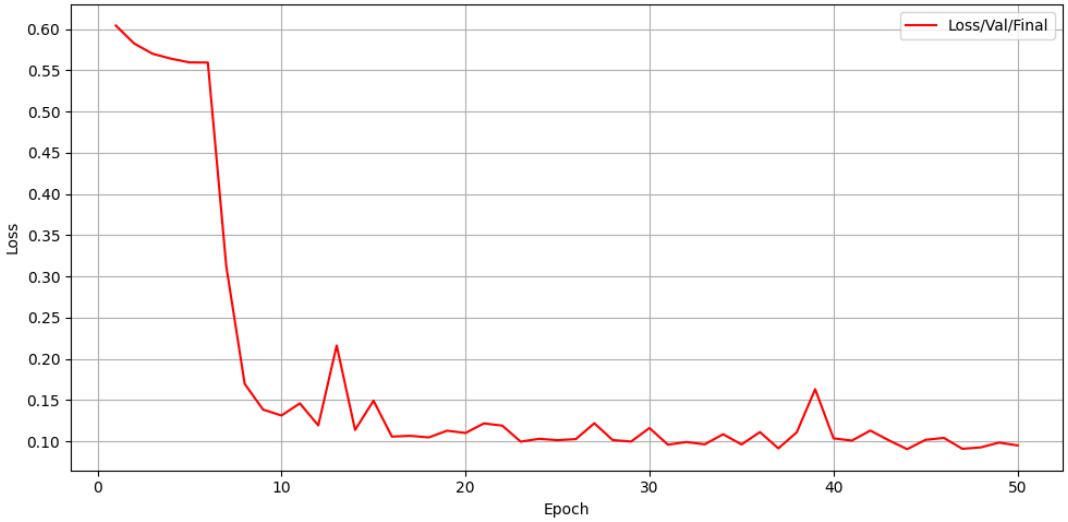
(b) Validierungsverlust

Abbildung C.1.: Experiment 3 - Verlustkurven von Training und Validierung über 20 Epochen, mit je Zuschneiden (Standard) und Größenskalierung (Scaled) (Quelle: Eigene Darstellung)

D. Finales Modell - Verlustkurven



(a) Trainingsverlust



(b) Validierungsverlust

Abbildung D.1.: Finales Modell - Verlustkurven von Training und Validierung über 50 Epochen (Quelle: Eigene Darstellung)