

FH DORTMUND

PROJEKTARBEIT IM STUDIENGANG SOFTWARE- UND
SYSTEMTECHNIK, FACHBEREICH INFORMATIK

Entwicklung eines Deep Learning Workflows zur Ein-Klassen-Segmentierung von Ulcera am Fuß

Autor:

AARON GÖCKE

MATR.-NR. 7201426

GEBOREN 20.08.1999

Betreuer:

Prof. Dr.-Ing. CHRISTOPH M.

FRIEDRICH

Zweitbetreuer:

M. SC. RAPHAEL BRÜNGEL

13. Januar 2023

Überblick

Kurzfassung

Ulcus bezeichnet in der Medizin tiefe offene Wunden, welche an verschiedenen Körperstellen auftreten und unterschiedliche Ursachen haben. Sie können beispielsweise durch Verletzungen, Infektionen oder Krankheiten, wie Diabetes oder Krebs entstehen. 40 bis 60 Millionen Menschen mit Diabetes mellitus entwickeln diabetischen Fußulcera. Die Vermessung der Wunden wird manuelle durchgeführt und ist sehr grob und zeitaufwändig. In dieser Projektarbeit wird gezeigt, wie durch Deep Learning diese Wunden automatisch in einem Bruchteil der manuell notwendigen Zeit segmentiert werden können. Dafür wurden mehrere U-Nets mit unterschiedlichen Hyperparametern trainiert und ausgewertet.

Abstract

Ulcus refers to deep open wounds in medicine, which can occur at various body locations and have different causes. They can be caused by injuries, infections, or diseases such as diabetes or cancer. 40 to 60 million people with diabetes mellitus develop diabetic foot ulcers. The measurement of the wounds is done manually and is very rough and time-consuming. This project shows how these wounds can be automatically segmented in a fraction of the time required manually, using deep learning. To do this, several U-Nets with different hyperparameters were trained and evaluated.

Inhaltsverzeichnis

Abbildungsverzeichnis	iii
1 Einführung	1
2 Vorgehensweise und Methoden	4
2.1 Deep Learning Workflow	4
2.2 Cross-Industry Standard Process for the development of Machine Learning applications with Quality assurance methodology	6
2.3 Computer Vision	8
3 Grundlagen	10
3.1 Ulcus	10
3.2 Künstliche Neuronale Netze	12
3.3 Convolutional Neuronale Netzwerke	20
4 Entwicklung	27
4.1 Business and Data Understanding	27
4.2 Data Engineering	34
4.3 Machine Learning Model Engineering	39
5 Ergebnisse und Diskussion	42
5.1 Ergebnisse	42
5.2 Diskussion	43
6 Zusammenfassung	47
A Aussortierte Bilder	49
Literatur	49
Eidesstattliche Erklärung	58

Abbildungsverzeichnis

2.1	CRISP-DM Workflow	5
2.2	CRISP-ML(Q) Ablauf	7
3.1	Ulcus am Fuß. ID 91 aus Trainingsdaten	11
3.2	Single-layer Netzwerk	13
3.3	Neural Net mit 3 Schichten	14
3.4	Sigmoid Funktion	22
3.5	U-Net Architektur	24
4.1	Beispieldaten 15 und 228	28
4.2	Größte und kleinste Maske aus den Trainingsdaten und Validierungsdaten	30
4.3	Größenunterschied zwischen ID 618 und 719	30
4.4	Die Verteilung des prozentualen Anteils der Pixel, die in den Masken zur segmentierten Wunde gehören.	31
4.5	Position der Masken	32
4.6	Beispielbilder mit leerer Maske	35
4.7	Bild mit 22 Instanzen	36
4.8	Augmentierte Bilder	38
5.1	Vorhersagen mit Dice-Ergebnis $> 0,95$	44
5.2	Vorhersagen mit Dice-Ergebnis $< 0,6$	45
5.3	Vorhersage des Modells mit Falsch-Positiver Instanz	46

1. Einführung

Künstliche Intelligenz (KI) revolutioniert das Gesundheitswesen, die Robotik, den Einzelhandel und viele weitere Bereiche (Norvig & Russell, 2010, S. 45ff). Die ersten Probleme, die mit KI gelöst wurden, waren welche die für den Menschen intellektuell herausfordernd sind, aber für den Computer relativ einfach. Diese Probleme wurden meist durch eine Liste von Formalismen beschrieben (z. B. Programme, welche Schach spielen können). Heute ist die Herausforderung für KI, Aufgaben zu lösen, die für den Menschen relativ einfach, aber schwer zu beschreiben sind. Zum Beispiel das Erkennen von gesprochenen Wörtern oder das Identifizieren bekannter Gesichter von Familienmitgliedern, Freunden, usw. (Goodfellow et al., 2016, S. 1). Diese Probleme werden meist mit Maschinelles Lernen (ML) angegangen. ML ist ein Teilbereich von Künstlicher Intelligenz, welcher sich mit der Verwendung von Daten und Algorithmen befasst und versucht, Modelle von Daten lernen zu lassen (Norvig & Russell, 2010, S. 679). In der Medizin wird ML angewendet, um unter anderem neue Medikamente zu entwickeln oder für Früherkennung, Prognose oder Diagnose von Krankheiten (May, 2021).

In dieser Projektarbeit geht es ebenfalls um ein Problem in der Medizin, dem Segmentieren von Ulcera am Fuß mithilfe von Deep Learning (DL) Algorithmen, welche wiederum ein Teilgebiet von ML sind. Ziel ist es einen Deep Learning Workflow zu entwickeln, welcher eine Ein-Klassen-Segmentierung von Ulcera Wundbetten realisiert. Der Begriff Deep Learning bezieht sich auf die Verwendung von Künstlichen Neuronalen Netzen mit mehreren Schichten. In den 1990er Jahren wurden bereits Experimente mit diesen Netzwerken durchgeführt, um z. B. handgeschriebene Zahlen zu erkennen (Lecun et al., 1998). In den darauffolgenden Jahren sind Forschungen und Investitionen in der KI stark zurückgegangen, da KI nicht die Er-

wartungen erfüllen konnte (Norvig & Russell, 2010, S. 44). Es dauerte bis 2012 bis KI bahnbrechende Ergebnisse liefert. Zuerst wurde ein DL Netzwerk für die Spracherkennung entwickelt, welches damals andere Techniken in der Spracherkennung übertraf (Hinton et al., 2012). Im selben Jahr wurde zudem das AlexNet (Krizhevsky et al., 2017) entwickelt. Beim ImageNet-Wettbewerb 2012, bei dem Bilder in eine von tausend Kategorien eingeordnet werden mussten, zeigte das AlexNet eine bemerkenswerte Verbesserung gegenüber klassischen Techniken. Der Durchbruch gelangte unter anderem, da Grafikprozessoren für das Training des Netzwerks verwendet wurden. Bevor Grafikprozessoren im DL Bereich eingesetzt wurden, waren Prozessoren der Standard für das Training der Netzwerke. Heutzutage wird das Training eines Netzwerkes mit einem Prozessor als unzureichend angesehen (Goodfellow et al., 2016, S. 439). Der Einsatz von Grafikprozessoren und die Entwicklung von immer besseren führte dazu, dass DL Netzwerke viel mehr Rechenleistung in Anspruch nehmen konnten. Dadurch war es möglich, größere DL Netzwerke zu entwickeln. Zudem erzielte das AlexNet eine gute Performance, da mit dem ImageNet (Deng et al., 2009) vortrainiert wurde. Das ImageNet ist ein Datensatz, welcher zu dem Zeitpunkt über 15 Millionen beschriftet Bilder enthielt, die in ca. 22000 Kategorien eingeordnet werden konnten (Krizhevsky et al., 2017). Die Erstellung von riesigen Datensätzen wird ermöglicht durch Big Data. Big Data ist ein Begriff, der verwendet wird, um enorme Mengen von strukturierten und unstrukturierten Daten zu beschreiben. Durch die Erfindung des World Wide Webs und die Digitalisierung in jeglichen Bereichen, werden immer mehr Daten online gespeichert und es entstehen riesige Datenbanken. Dadurch wird es einfacher, aus einzelnen Daten einen Datensatz zusammenzustellen, welche für das Training der DL Netzwerke genutzt werden können (Goodfellow et al., 2016, S. 21).

Im Rahmen dieser Projektarbeit wird gezeigt wie ein Deep Learning Workflow zur Ein-Klassen-Segmentierung von Ulcera am Fuß entwickelt werden kann. Dabei besteht der erste Abschnitt aus dieser Einführung. Im zweiten Abschnitt werden die Vorgehensweise festgelegt und Metriken im DL Bereich vorgestellt. Zuerst wird hier auf den Deep Learning Workflow eingegangen, danach auf das systematisches Prozessmodell Cross-Industry Standard Process for the development of Machine

Learning applications with Quality assurance methodology (CRISP-ML(Q)) um DL Software zu entwickeln. Zudem wird hier der Bereich Computer Vision vorgestellt. Im dritten Abschnitt werden die Grundlagen für diese Projektarbeit eingeführt. Hier wird die Krankheit Ulcus genauer erläutert und es werden Künstliche Neuronale Netzwerke vorgestellt. In Abschnitt vier geht es um die Entwicklung des Deep Learning Workflows für die Ein-Klassen-Segmentierung von Ulcera am Fuß. Dabei orientiert sich die Entwicklung am CRISP-ML(Q) Modell. Im fünften Abschnitt werden die Ergebnisse ausgewertet. Zudem werden diese diskutiert und bewertet. Der sechste und letzte Abschnitt ist eine Zusammenfassung dieser Projektarbeit. Zudem gibt es noch einen Anhang A, welcher im Rahmen des 4. Abschnitts, alle aussortierten Bilder listet.

2. Vorgehensweise und Methoden

In diesem Abschnitt geht es um den Deep Learning Workflow und dem Bereich Computer Vision. Zuerst wird gezeigt was ein Deep Learning Workflow ist und wie dieser abläuft. Danach wird das Prozessmodell CRISP-ML(Q) (Studer et al., 2021) vorgestellt, welches für die Entwicklung von ML Software genutzt werden kann. Zum Schluss wird der Bereich Computer Vision vorgestellt. Unter anderem werden hier DL Methoden zu Segmentierungsverfahren gezeigt.

2.1 Deep Learning Workflow

Anstatt dem Computer ausführbare Formalismen als Anweisungen zu geben, versucht DL einen Algorithmus zu entwickeln, der sich anhand der Daten sein Verhalten anpasst. Der Workflow beim Deep Learning bezieht sich auf den Prozess, der befolgt wird, um ein DL Modell zu entwickeln und zu trainieren. Die Schritte für den Deep Learning Workflow sind in Abbildung 2.1 dargestellt.

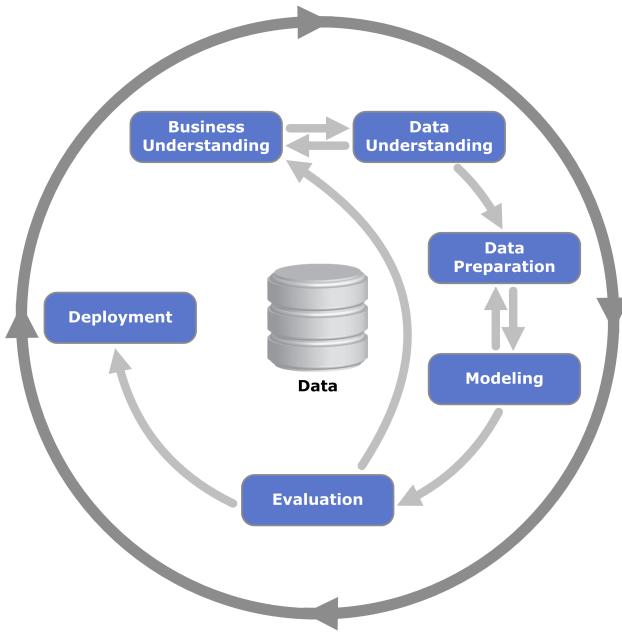


Abbildung 2.1: CRISP-DM Prozess [Bild entnommen aus Quelle: (Jensen, 2012) und (IBM, 2016)]

Die erste Phase des Workflows besteht aus der Problemdefinition und dem Zusammenstellen eines Datensatzes. Hier wird verdeutlicht, was die Eingabedaten sind, was versucht wird vorherzusagen und was für ein Problem vorliegt. Zudem wird entschieden, ob DL adäquate Methoden bereitstellt, um das Problem zu lösen (Chollet, 2017, S. 111). Anschließend wird festgelegt, wie der Erfolg des Modells gemessen werden kann. Je nach Problem, gibt es unterschiedliche passende Metriken, welche jeweils eine andere Sicht auf die Leistungsfähigkeit der Modelle, hinsichtlich des Problems, ermöglichen. Um das Modell in der Entwicklung zu Evaluieren, wird ein Ergebnisprotokoll festgelegt. Häufig wird dazu von dem Datensatz ein Validierungsdatensatz gebildet, welcher dann nicht für das Training, sondern zum evaluieren während des Trainings genutzt wird. Falls der Datensatz zu klein ist, um ein Validierungsdatensatz beiseitezulegen, kann auch K-Fache Kreuzvalidierung genutzt werden (Lakshmanan et al., 2021, S. 87). Bei der K-Fachen Kreuzvalidierung werden die verfügbaren Daten in K gleich große Partitionen aufgeteilt. Anschließend wird eine Partition zum Validieren und die restlichen zum Trainieren genutzt. Das Ganze wird so oft gemacht, bis jede Partition einmal zum Validieren genutzt wurde. Zum

Schluss wird der Durchschnitt der einzelnen Validierungsergebnisse gebildet. Im Workflow werden danach die vorliegenden Daten vorbereitet.

Der nächste Schritt besteht darin, Deep Learning-Modelle auszuwählen und die Hyperparameter festzulegen, die das Verhalten der Modelle bestimmen. Sobald eins oder mehrere Modelle definiert wurden, können diese mit den vorverarbeiteten Daten trainiert werden. Das Training besteht darin, die Parameter bzw. Gewichte der Modelle anzupassen, um den Fehler zu minimieren. Nach dem Training der Modelle, werden diese auf einem Testdatensatz bewertet. Basierend auf den Ergebnissen können die Hyperparameter angepasst oder die Modelle selbst verbessert werden, um die Leistungsfähigkeit der Modell zu steigern. Sobald ein oder mehrere geeignete Modelle mit guter Leistung gefunden wurden, können diese in einer Anwendung implementiert werden.

2.2 Cross-Industry Standard Process for the development of Machine Learning applications with Quality assurance methodology

Der Deep Learning Workflow dieser Projektarbeit orientiert sich an dem vorgeschlagenen Prozessmodell Cross-Industry Standard Process for the development of Machine Learning applications with Quality assurance methodology oder kurz CRISP-ML(Q) (Studer et al., 2021). Er ist konzipiert für die Entwicklung von maschinellen Anwendungen. Abbildung 2.2 zeigt das CRISP-ML(Q) Prozessmodell.

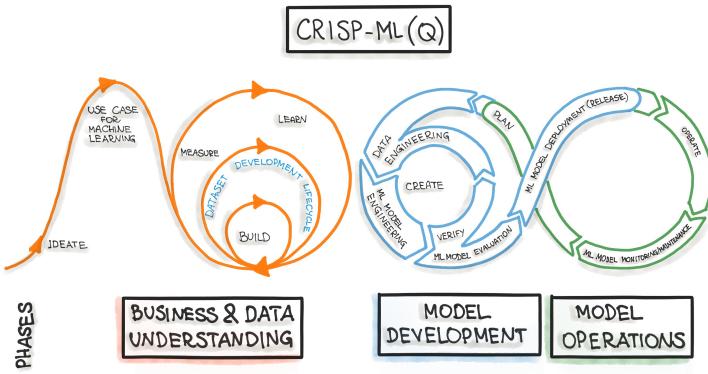


Abbildung 2.2: CRISP-ML(Q) Ablauf [Bild entnommen aus Quelle: Eberstaller, 2012]

Insgesamt besteht das Prozessmodell aus sechs Phasen:

1. Business and Data Understanding: Geschäftsproblem definieren, Ziele des ML Projektes festlegen, Identifizieren des Umfangs, Sicherstellen der Durchführbarkeit, Daten sammeln, Qualität der Daten überprüfen und Metriken festlegen.
2. Data Engineering: Daten werden vorverarbeitet, die richtigen Daten ausgewählt, die Daten standardisiert und augmentierte Daten erstellt.
3. Machine Learning Model Engineering: Ziel dieser Phase ist es ein oder mehrere Modelle zu definieren und diese zu trainieren.
4. Evaluating Machine Learning Models: Performance der Modelle wird anhand eines Testdatensatz bewertet. Die Evaluierungsphase ist auch als Offline-Testen bekannt.
5. Deployment: Ziel ist es das Modell in die Softwareumgebung zu integrieren.
6. Monitoring and Maintenance: Überwachen und Warten des Modell.

Die Phasen 5 und 6 werden vernachlässigt, da das Modell für diese Projektarbeit entwickelt wird und kein Deployment geplant ist. Was genau in den Phasen

umgesetzt wurde, wird in Abschnitt 4 beschrieben.

2.3 Computer Vision

Computer Vision ist ein Bereich in der KI, welcher sich mit dem Extrahieren von sinnvollen Informationen aus visuellen Eingaben, wie Bildern oder Videos beschäftigt. Mit den extrahierten Informationen werden dann automatische Aktionen ausgeführt. Als Beispiel aus dem Bereich der Entwicklung selbstfahrender Automobile kann die Erkennung von Ampelzuständen dienen. Wenn ein selbstfahrendes Auto z. B. eine rote Ampel identifiziert, dann wird automatisch das Auto angehalten.

Neben Bildklassifizierung (Dosovitskiy et al., 2021), Objekterkennung (Wang et al., 2022a), erweiterte Realität (engl. augmented reality) (Eckert et al., 2019) und Gesichtserkennungssystemen (Kumar et al., 2019) gehört auch die Segmentierung von Bildern zu den Anwendungsgebieten von Computer Vision. Die automatische Segmentierung wird heute z. B. bei der Computertomographie (Ecabert et al., 2008) oder der Magnetresonanztomographie (Pereira et al., 2016) eingesetzt. Bei der Segmentierung unterscheidet man zwischen drei Verfahren: semantische Segmentierung, Instanzsegmentierung und panoptische Segmentierung. Bei der semantischen Segmentierung ist das Ziel jedes Pixel im Bild einer Klasse zuzuordnen. Hier wird nicht zwischen verschiedenen Instanzen einer Klasse unterschieden. Wenn z. B. in einem Bild mehrere Autos zu sehen sind, dann werden bei der semantischen Segmentierung alle Autos als ein Objekt segmentiert. Bei der Instanzsegmentierung wird dann für das Beispiel versucht, jedes Auto im Bild als eine eigene Objektinstanz zu segmentieren. Allgemein versucht die Instanzsegmentierung für jede Objektinstanz im Bild die Klasse und die Maske vorherzusagen. Die Kombination von semantischer Segmentierung und Instanzsegmentierung wird als panoptische Segmentierung bezeichnet (Kirillov et al., 2019). Diese Projektarbeit befasst sich mit der semantischen Segmentierung. Um Modelle in diesem Bereich zu vergleichen, kann der ADE20K (Zhou et al., 2017) (Zhou et al., 2019) Datensatz genutzt werden. Dieser besteht aus mehr als 20.000 Szenenbildern. Auf diesen sind Objekte zu sehen,

welche in 150 Klassen eingeordnet werden können. Die höchste Leistung auf dem Datensatz erzielt aktuell BEiT-3 (Wang et al., 2022b).

3. Grundlagen

Dieser Abschnitt beschreibt die Grundlagen, welche für die Entwicklung des Deep Learning Workflows in dieser Projektarbeit gebraucht werden. In Abschnitt 3.1 wird die Krankheit Ulcus vorgestellt. Abschnitt 3.2 beschreibt wie Künstliche Neuronale Netze (KNN) aufgebaut sind und wie diese lernen. Der letzte Abschnitt beschreibt Convolutional Neural Networks (CNN), eine besondere Art von Künstlichen Neuronalen Netzten. Oft werden diese genutzt, um Bilder zu analysieren.

3.1 Ulcus

Ulcus (Mehrzahl Ulcera) bezeichnet eine Wunde auf der Haut oder den Schleimhäuten, die nicht schnell oder richtig abheilt. Ulcera können durch eine Vielzahl von Faktoren verursacht werden, z. B. durch Infektionen, Reizungen oder Grunderkrankungen. Diese können in Größe und Schweregrad variieren und von Symptomen wie Schmerzen, Rötung und Ausfluss begleitet sein (Dissemond et al., 2020). In dieser Projektarbeit wird nur auf den Ulcus am Fuß eingegangen. Die Behandlung von Ulcus umfasst in der Regel eine Kombination von Ansätzen. Antibiotika können verschrieben werden um Infektionen zu behandeln oder zu verhindern. Andere Medikamente können helfen die Heilung zu fördern oder zur Verbesserung der Durchblutung. Die Ulcera Wunden sauber und feucht zu halten ist wichtig, um die Heilung zu fördern. Dies kann das Waschen der Ulcera mit Kochsalzlösung oder das Auftragen von Wundaflagen beinhalten. Wenn nach ca. 3 Monaten kein Heilungsprozess zu sehen ist oder die Wunde nach 12 Monaten nicht abgeheilt ist, dann muss die Wunde chirurgisch behandelt werden (Hermanns, 2016). Dies kann Verfahren umfassen, um beschädigte Blutgefäße zu reparieren oder Hindernisse zu

entfernen, die eine schlechte Durchblutung verursachen. Schwere Ulcera Wunden können mit der Shave-OP behandelt werden. Dabei wird die Ulcus Wunde aus der Haut ausgeschnitten und anschließend überdeckt (Hermanns, 2016). Wenn Ulceras nicht in einem frühen Stadium behandelt werden, können diese zur Amputation der unteren Gliedmaßen führen. Abbildung 3.1 zeigt die Krankheit am Fuß.



Abbildung 3.1: Ulcus am Fuß. ID 91 aus Trainingsdaten

Der Aufbau der Wunde besteht aus Wundrand, Wundumgebungshaut und Wundbett. Der Wundrand beschreibt den Übergang zwischen der Wunde und der heilen Haut. Die Wundumgebungshaut ist die Haut, die an den Wundrand von außen angrenzt. Das Wundbett beschreibt den Teil, der von dem Wundrand umgeben ist. Der weltweite Markt für die Behandlung von diabetischen Fußulceras wurde im Jahr 2021 von der Precedence Research auf 8,6 Milliarden USD geschätzt (“Diabetic Foot Ulcer Treatment Market”, 2021). Bis zum Jahr 2030 wird prognostiziert, dass dieser Wert bis auf 14,8 Milliarden USD steigen wird. Im Jahr 2021 waren 537 Millionen Leute an Diabetes erkrankt und bis zum Jahr 2030 wird geschätzt, dass die Zahl auf 643 Millionen steigen wird (Sun et al., 2022). Davon entwickeln 40 bis 60 Millionen pro Jahr diabetischen Fuß Ulcus (Akhtar et al., 2022). Die wirtschaftlichen Kosten und die Anzahl an betroffenen Personen zeigt, dass Ulcera eine verbreitete Krankheit ist. Das Vermessen der Wunden in der Medizin wird manuell durchgeführt und ist sehr zeitaufwendig. Oft entstehen dadurch auch hohe Kosten. Zudem ist die Vermessung meist nicht reproduzierbar und kann zwischen den Personen, die diese erstellen, variieren. Aus diesen Gründen ist eine Segmentierung und damit eine automatische Vermessung, mithilfe eines Algorithmus zu präferieren.

ren, unter der Annahme, dass dieser Algorithmus genaue Ergebnisse erzielt.

3.2 Künstliche Neuronale Netze

In diesem Abschnitt wird beschrieben, was KNNs sind und wie diese lernen. Anschließend wird noch auf CNNs, eine spezielle Art von KNNs, eingegangen. Diese können für die Objekterkennung in Bildern genutzt werden.

KNNs wurden entwickelt, um das Neuronale Netz im biologischen Gehirn zu simulieren (Goodfellow et al., 2016, S. 13). Künstliche Neuronen sind die Bausteine von den KNNs und wurden auch durch biologische Neuronale Netze inspiriert. Wenn im weiteren Verlauf von Neuronalen Netzen (NN) gesprochen wird, sind damit KNNs gemeint und wenn von Neuronen gesprochen wird, sind damit Künstliche Neuronen gemeint.

Das einfachste Neuronale Netz ist ein single-layer Netzwerk, da es nur aus einem Neuron besteht. Der Eingang für ein Neuron ist ein Vektor $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$ mit der Dimension n (Aggarwal et al., 2018, S. 5). Dabei steht jeder Wert von \mathbf{x} für ein Feature. Features sind dabei Eigenschaften, von dem was vorhergesagt werden soll. Das Neuron speichert ebenfalls n Gewichte. Jedes einzelne Feature x wird dann mit dem Gewicht in der zugehörigen Node multipliziert. Anschließend wird die Summe aus allen Multiplikationen gebildet. Wenn der Wert der Summe dann einen bestimmten Wert hat, z. B. größer gleich 0, dann ist die Ausgabe 1. Falls der Wert kleiner als 0 ist, wird 0 ausgegeben. Der ausgegebende Wert hängt von der Aktivierungsfunktionen ϕ ab. Diese sorgt dafür, dass Nichtlinearität in das NN gebracht wird, was wichtig ist für multi-layer Netzwerke (Aggarwal et al., 2018, S. 11). Zudem sorgen viele Aktivierungsfunktionen dafür, dass die Ausgabewerte im Bereich $[-1, 1]$ bzw. $[0, 1]$ liegen, damit das Modell besser lernen kann. Mit unterschiedlichen Aktivierungsfunktionen kann man dann unterschiedliche NN entwickeln. Mathematisch kann man das single-layer Netzwerk dann wie folgt beschreiben:

$$\hat{y} = \phi\left(\sum_{i=1}^n x_i \times w_i + b\right) \quad (3.1)$$

Wobei \hat{y} die Vorhersage ist, ϕ die Aktivierungsfunktionen, x_n das jeweilige feature, w_n das jeweilige Gewicht und b der Bias. Der Bias ist ein Wert, welcher die Begrenzung der Aktivierungsfunktion ohne Abhängigkeit vom Eingabewert vom Ursprung verschieben kann. Zum Beispiel kann der Bias garantieren, dass wenn alle Eingaben 0 sind, trotzdem 1 ausgegeben wird. Die grafisch Darstellung des single-layer Netzwerk sieht man in Abbildung 3.2.

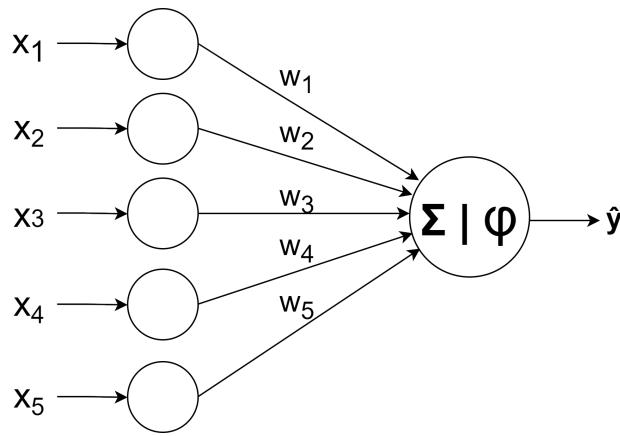


Abbildung 3.2: Single-layer Netzwerk

Dabei sei noch erwähnt, dass es sich eigentlich um zwei Schichten handelt. Zum einen die Eingabeschicht, welche die Daten zu den einzelnen Nodes weiterleitet und die Ausgabeschicht, welche die Berechnung durchführt (Multiplikation mit den Gewichten, Summieren der einzelnen Ausgaben der Nodes und die Aktivierungsfunktion). Die Eingabeschicht wird allerdings nicht mit in die Anzahl der Schichten in einem Neuronal Netz mit einbezogen (Aggarwal et al., 2018, S. 6).

Netzwerke, bei denen die Informationen nur in eine Richtung fließen, also wo es keine Schleifen innerhalb des Netzwerkes gibt, werden als feedforward Netzwerke bezeichnet (Goodfellow et al., 2016, S. 164). Das feedforward Netzwerk definiert dabei eine Funktion $y = f(x; \theta)$, welche den Eingang x zu einer Kategorie y abbildet und dabei die Gewichte θ lernt. Mit unterschiedlichen Werten für θ bekommt man dann unterschiedliche Ergebnisse. Bei dem single-layer Netzwerk besteht die Funktion f aus der Multiplikation der Eingänge x mit den Gewichten, die Summierung

aller Multiplikationen und das Abbilden dieser Summe mit der Aktivierungsfunktion.

Multi-layer Netzwerke sind NN mit mehr als einer Schicht, welche Berechnungen durchführen. Diese werden multi-layer Netzwerke genannt, weil sie sich aus mehreren Funktion zusammensetzen. Der Zusammenhang der Funktionen wird durch einen gerichteten Graph ohne Zyklus beschrieben (Goodfellow et al., 2016, S. 164). Die Anzahl der Schichten oder die Länge der Kette der Funktionen wird als Tiefe des Netzwerkes bezeichnet, daher auch der Name Deep Learning. Bei multi-layer Netzwerken werden die Schichten, die zwischen der Eingabe- und Ausgabeschicht liegen, als versteckte Schichten bezeichnet, da die Berechnungen dieser Schichten nicht sichtbar für den Benutzer sind (Aggarwal et al., 2018, S. 17). Ohne die Aktivierungsfunktion wäre das NN nur eine Kombination von linearen Transformationen, welche nacheinander addiert werden. Diese könnten somit auch zu einer linearen Kombination zusammengefasst werden und somit würde das Stapeln der Schichten nichts bringen (Goodfellow et al., 2016, S. 172). Aus diesem Grunde wird nach jeder linearen Transformation eine nichtlineare Aktivierungsfunktion genutzt. Die Nichtlinearität sorgt dafür, dass das NN komplexe Probleme lösen kann. Im folgenden wird das NN in Abbildung 3.3 genauer betrachtet.

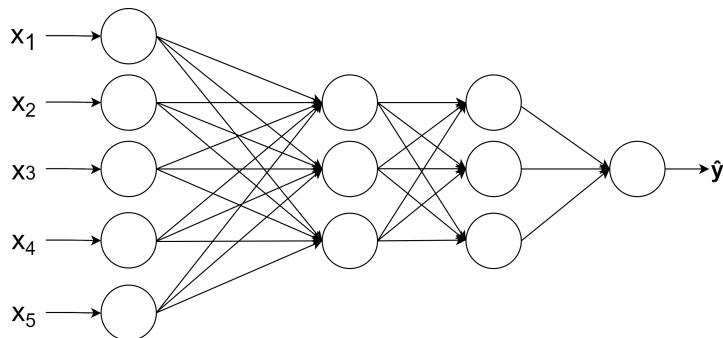


Abbildung 3.3: Neuronales Netz mit 3 Schichten

Es hat eine Tiefe von 3, hat 5 Eingabe Features und 1 Ausgabewert. Auffällig ist dabei, dass jedes Feature an jede der 3 Nodes in der ersten versteckten Schicht weitergeleitet wird. Wenn alle Neuronen in einer Schicht mit der nächsten Schicht verbunden sind, dann spricht man von einem Fully connected neural networks (FCNN).

Die Eingabe des Netzwerks kann mit einem Vektor der Dimension 5 beschrieben werden $\mathbf{x} = [x_1, x_2, x_3, x_4, x_5]^T$. Jede Node in der ersten versteckten Schicht speichert dabei 5 Gewichte, welche auch in einem Vektor mit der Dimension 5 gespeichert werden. Werden nun alle Gewichte der 3 Nodes in dieser Schicht zusammengefasst, erhält man eine 3×5 Matrix W_1 . Alle Gewichte der zweiten versteckten Schicht werden in einer 3×3 Matrix W_2 und die Gewichte in der Ausgabeschicht in einer 3×3 Matrix W_3 gespeichert. Wenn jetzt der Eingabe Vektor x in das Netzwerk gegeben wird, dann passiert folgendes:

1. Multipliziere die Ausgabe von der vorherigen Schicht mit der jeweiligen Gewichtsmatrix in dieser Schicht. Das Ergebnis ist ein Vektor mit der Dimension, welche der Anzahl der Neuronen in dieser Schicht entspricht. Die Werte in den Vektoren stellen dann die Summe der einzelnen Multiplikationen in dem jeweiligen Neuron dar.
2. Jeder Wert dieses Vektors wird in die Aktivierungsfunktion gegeben und man erhält einen Vektor mit gleicher Dimension aber anderen Werten.
3. Falls noch eine Schicht folgt, wird wieder bei Schritt 1 begonnen.

Mathematisch kann das Netzwerk dann in Formel 3.2 zusammengefasst werden.

$$NN(\mathbf{x}) = \phi(W_3 \times (\phi(W_2 \times \phi(W_1 \times \mathbf{x})))) = \hat{y} \quad (3.2)$$

Die Vorhersage des NN für den Eingabevektor \mathbf{x} ist dann \hat{y} . Das Ziel des NN ist es eine möglichst gute Vorhersage abzugeben. Um zu beurteilen, wie gut das NN zu einem Zeitpunkt funktioniert, kann es mit einer Fehlerfunktion ausgewertet werden (oft auch Verlustfunktion oder Kostenfunktion genannt) (Goodfellow et al., 2016, S. 80). Um einen möglichst kleinen Fehler zu erhalten, wird versucht, die Fehlerfunktion zu minimieren. In Bereich Deep Learning wird dies als Optimisierung bezeichnet. Optimisierung kann allerdings auch bedeuten, die Fehlerfunktion zu maximieren für bestimmte Architekturen (Goodfellow et al., 2016, S. 80). Da unter anderem Convolutional Neural Networks, welche später noch erläutert werden, die

Fehlerfunktion versuchen zu minimieren, wird im folgenden mit Optimisierung die Minimierung der Fehlerfunktion beschrieben.

Gradientenverfahren

Das Gradientenverfahren dient dazu, bei einer differenzierbaren Funktion einen stationären Punkt zu finden. Er bewegt sich iterativ in die Richtung des steilsten Abstiegs mithilfe des negativen Gradienten (Goodfellow et al., 2016, S. 81). Der Gradient ist ein Vektor, der für eine Funktion f alle partiellen Ableitungen für einen Punkt angibt (siehe Formel 3.3).

$$\nabla f(x_1, x_2, \dots, x_n) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x_1, x_2, \dots, x_n) \\ \frac{\partial f}{\partial x_2}(x_1, x_2, \dots, x_n) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x_1, x_2, \dots, x_n) \end{bmatrix} \quad (3.3)$$

Der Algorithmus wurde erstmals 1847 von dem französischen Mathematiker Augustin-Louis Cauchy vorgestellt (Cauchy, 1847, S. 536ff). Das Gradientenverfahren wird im NN genutzt, um den Fehler zu minimieren. Angenommen y soll vorhergesagt werden und das NN sagt \hat{y} voraus, dann wird der Fehler zwischen den beiden Werten mit der Fehlerfunktion $\ell(y, \hat{y})$ berechnet. Welche Fehlerfunktion ausgewählt wird, hängt von der Architektur des NN ab. Fehlerfunktionen werden genauer im Abschnitt über Convolutional Neural Network erläutert.

Der Fehler im NN in Abbildung 3.3 kann mit der Formel 3.4 berechnet werden.

$$Fehler = \ell(y, \phi(W_3 \times (\phi(W_2 \times \phi(W_1 \times \mathbf{x}))))) \quad (3.4)$$

Ziel ist es, die Gewichte W_1 , W_2 und W_3 so zu verändern, dass der Fehler möglichst klein wird. Da die ganze Funktion differenzierbar ist, kann dafür das Gradientenverfahren genutzt werden. Da die Ableitung für die Funktion ℓ mit Respekt zu jedem Gewicht berechnet werden muss, bietet sich die Jacobian Matrix an. Die Jacobian

Matrix \mathbb{J} mit der Größe $n \times m$ enthält alle erste partielle Ableitungen einer Funktion $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ (siehe Formel 3.5).

$$\mathbb{J}_{n,m} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \dots & \frac{\partial f_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m(x)}{\partial x_1} & \dots & \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix} \quad (3.5)$$

Die ersten partiellen Ableitungen werden dann benötigt, um bei der Backpropagation die Gewichte zu aktualisieren.

Backpropagation

Backpropagation ist ein Algorithmus, der aus zwei Phasen besteht: Vorwärtsphase und Rückwärtsphase (Aggarwal et al., 2018, S. 1). In der Vorwärtsphase wird zuerst der Eingabevektor in das NN geben. Die vorhergesagte Ausgabe kann mit dem, was vorhergesagt werden soll, verglichen werden und man erhält den Fehler. In der Rückwärtsphase wird dann die Ableitung der Fehlerfunktion errechnet. Die Ableitung dieser Fehlerfunktion muss nun in Bezug auf die Gewichte in allen Schichten in der Rückwärtsphase berechnet werden und der Gradient wird dann genutzt, um die Gewichte zu aktualisieren (Aggarwal et al., 2018, S. 21). Das Aktualisieren der Gewichte und damit das Minimieren der Fehlerfunktion wird vom Optimierungsalgorithmen übernommen. Fast alle DL Netzwerke nutzen Stochastic Gradient Descent (SGD, siehe Formel 3.6) oder eine modifizierte Version.

$$w_{ij}^t = w_{ij}^{t-1} - \alpha \frac{\partial \ell}{\partial w_{ij}^{t-1}} \quad (3.6)$$

Der Wert t steht für die Zeitangabe und der Wert α steht für die Lernrate, welcher sich darauf auswirkt, wie stark die Gewichte aktualisiert werden. Wenn die Lernrate zu groß ist, kann es sein, dass ein lokaler Minimalpunkt verfehlt wird. Wenn der Parameter zu klein ist, kann es zu lange dauern, bis man zu einem lokalen Minimum konvergiert. Der Optimierungsalgorithmus garantiert allerdings nicht, dass überhaupt ein lokales Minimum erreicht wird. Anderseits findet dieser oft einen

kleinen Wert, welcher genügt (Goodfellow et al., 2016, S. 150). Die Lernrate zu wählen ist schwierig, da diese die Modell Performance sehr beeinflusst (Goodfellow et al., 2016, S. 302). Als Lernrate kann ein fixer Parameter genutzt werden oder ein Set von Lernraten $\{\alpha_1 \dots \alpha_k\}$. Die Lernrate wird dann jeweils nach Iteration k aktualisiert. SGD ist oft langsam, deswegen werden oft modifizierte Versionen von SGD genutzt. Ein bekannter Optimierungsalgorithmus ist Adam (Kingma & Ba, 2015). Dieser schätzt das erste Moment und das zweite Moment von dem Gradient und aktualisiert damit die Gewichte (siehe Formel 3.7).

$$w_{ij}^t = w_{ij}^{t-1} - \alpha \frac{\hat{m}_t}{(\sqrt{\hat{v}_t} + \epsilon)} \quad (3.7)$$

Wobei \hat{m}_t und \hat{v}_t die jeweils Bias-korrigierten Schätzungen für jeweils das erste und das zweite Moment sind. Für die Berechnung von \hat{m}_t und \hat{v}_t muss jeweils die anfängliche Abklingrate β_1 und β_2 angegeben werden. Der Wert ϵ sorgt für numerische Stabilität, damit die Division durch 0 verhindert wird.

Wann die Parameter aktualisiert werden, hängt von der Batch Größe ab. Die Batch Größe bestimmt dabei, in wie viele Teile der Trainingsdatensatz unterteilt wird. Oft wird der Datensatz in Mini Batches unterteilt mit der Größe 16 oder 32. Die Epochgröße bestimmt dabei, wie oft das NN alle Trainingsdaten betrachtet, z. B. wenn die Epochgröße 4 ist, werden alle Daten 4 mal nacheinander in das NN gegeben (Aggarwal et al., 2018, S. 7). Da der Algorithmus nur mit Zahlen rechnet, sollten die Daten normalisiert und standardisiert werden (Chollet, 2017, S. 86). Die Normalisierung der Daten sorgt dafür, dass die Daten zwischen einem kleinen Intervall liegen, z. B. $[-1, 1]$ oder $[0, 1]$. Standardisierung sorgt dafür, dass die Verteilung der Daten eine Einheitsvarianz haben. Dadurch haben die Daten die Eigenschaften einer Normalverteilung, also einen Mittelwert von 0 und eine Standardabweichung von 1 (Géron, 2019, S. 72ff). Angenommen, der Eingabetensor enthält eine Spalte mit Werten zwischen 0 und 1 und eine andere mit Werten zwischen 1000 und 10000, dann geht der Algorithmus davon aus, dass die Zahlen im höheren Bereich in irgendeiner Weise überlegen sind. Der große Unterschied in der Skalierung zwischen den Werten kann zu Problemen führen, wenn die Werte als Features kombiniert

werden (Géron, 2019, S. 73). Bei der Entwicklung des Modells wird anschließend festgelegt, welche Fehlerfunktion, welche Optimisierungsverfahren und welche Aktivierungsfunktion in der letzten Schicht genutzt werden sollen. Die erste Entwicklung des Modells sollte statistische Aussagekraft erreichen (Chollet, 2017, S. 113). In der letzten Phase wird das Modell regularisiert und Hyperparameter des Modells werden abgestimmt. Zum Beispiel können hier dem Modell weitere Schichten hinzugefügt werden, andere Hyperparameter ausprobiert werden oder neue Features hinzugefügt werden. Sobald eine geeignete Modellkonfiguration gefunden wurde, können allen verfügbaren Trainings- und Validierungsdaten zum Trainieren genutzt werden. Zuletzt evaluiert man das Modell dann anhand der Testdaten.

Generalisierung

Wenn das Modell trainiert wurde und es einen möglichst kleinen Fehler bei den Trainingsdaten erreicht hat, wird es anhand von Testdaten ausgewertet. Diese Daten sind für das Modell neue bisher unbekannte Eingaben. Ziel des Modells ist es, bei diesen neuen Daten auch eine gute Performance und einen geringen Fehler zu erreichen. Als Generalisierung bezeichnet man, dass das Modell sich gut an neue unbekannte Eingaben anpasst (Goodfellow et al., 2016, S. 108). Wenn der Trainingsfehler deutlich geringer als der Test Fehler ist, dann wird dies als Überanpassung (englisch: overfitting) bezeichnet. Überanpassung kann auftreten, wenn z. B. das Modell zu komplex ist. Wenn das Modell keinen geringen Trainings Fehler bekommt, dann ist das ein Zeichen für Unteranpassung (englisch: underfitting). Unteranpassung kann auftreten, wenn z. B. die Eingabedaten zu wenig Features haben. Das Problem mit FCNNs ist folgendes: wenn z. B. ein Grauwertbild, welches 1024×1024 Pixel groß ist, in das Netzwerk gegeben werden soll, dann braucht allein ein Neuron der ersten Schicht 1048576 Gewichte um jedes Pixel der Eingabe zu betrachten. Da das Netzwerk aus mehreren Neuronen besteht, wird die Nummer an Parametern sehr groß und das Netzwerk würde sehr schnell überangepasst sein (Lecun et al., 1998).

3.3 Convolutional Neuronale Netzwerke

Convolutional Neuronale Netzwerke (CNNs) sind eine spezielle Art von feedforward Netzwerken. Die Eingabedaten für CNNs haben dabei eine gitterartige Topologie, wie z. B. Bilder oder Audiodaten. Die meisten Applikationen von CNNs nutzen Bilder als Eingabedaten (Aggarwal et al., 2018, S. 17). Der Name der Netzwerke beschreibt bereits die mathematische Operation convolution (deutsch: Faltung), welche in den Netzwerken genutzt wird. Wenn diese in mindestens einer Schicht des NN genutzt wird, werden diese als CNNs bezeichnet. Die Faltung (durch den Operator $*$ beschrieben) von zwei Funktion ist definiert in Formel 3.8 (Bracewell, 1986, S. 24).

$$s(x) = (f * g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x - \tau)d\tau \quad (3.8)$$

Da die Funktion kommutativ ist, gilt auch Formel 3.9.

$$s(x) = (f * g)(x) = \int_{-\infty}^{\infty} f(x - \tau)g(\tau)d\tau \quad (3.9)$$

Das Ergebnis der Faltung beschreibt Überlappung der an der y-Achse gespiegelten Funktion g , welche über die Funktion f verschoben wird. Diese Überlappung wird dann durch die Funktion s beschrieben. In CNNs werden dabei die Eingabedaten mit der Funktion f beschrieben und die Funktion g beschreibt den Filter. Der Filter ist ein multidimensionales Array, welches die Parameter enthält, die vom Modell gelernt werden sollen. Dieser Filter wird um 180 Grad rotiert und über die Eingabedaten geschoben. Anschließend wird das Skalarprodukt aller Überlappungen gebildet. Angenommen, es gibt ein zwei-dimensionales Bild B und einen zweidimensionalen Filter K , dann kann die Faltung mit Formel 3.10 beschrieben werden (Goodfellow et al., 2016, S. 328).

$$S(i, j) = (B * K)(i, j) = \sum_m \sum_n B(i - m, j - n)K(m, n) \quad (3.10)$$

Die Ausgabe ist ein multidimensionaler Array, welcher auch als Feature Map bezeichnet wird (Goodfellow et al., 2016, S. 329). In ML Frameworks wird oft Kreuzkorrelation anstatt der Faltung implementiert (siehe Formel 3.11) (“Pytorch”, 2022).

$$S(i, j) = (B * K)(i, j) = \sum_m \sum_n B(i + m, j + n)K(m, n) \quad (3.11)$$

Die Kreuzkorrelation bewirkt das gleiche wie die Faltung, allerdings ohne den Filter um 180 Grad zu drehen bzw. ohne die Funktion g zu spiegeln. Dies wird erreicht, indem die Funktion f vor der Faltung komplex-konjugiert wird. Da unter anderem Faltungen in CNNs nie alleine genutzt werden, ist es weniger wichtig, ob CNNs die Gewichte mit dem rotierten oder nicht rotierten Filter lernen (Goodfellow et al., 2016, S. 329).

Die Filtergröße bestimmt, wie groß der Filter ist. Häufig werden 2×2 oder 3×3 Filter genutzt. Die Faltung benutzt noch zwei weitere Parameter, Schrittbreite (engl. stride) und Padding. Die Schrittbreite beschreibt, wie viele Pixel der Filter pro Iteration über das Bild gleitet. Standardmäßig ist die Schrittbreite 1. Wenn z. B. über ein 3×3 großes Bild ein 2×2 großer Filter mit Schrittgröße 1 geschoben wird, dann reduziert sich die Ausgabe auf 2×2 Pixel. Die Reduzierung kann dafür sorgen, dass am Rand des Bildes Information verloren gehen. Um das zu verhindern, nutzt man Padding. Padding bezeichnet das hinzufügen eines Randes, häufig mit den Pixelwerten 0, um den Rand des Bildes zu erweitern.

Üblicherweise besteht eine CNN Schicht aus 3 Teilen. Zuerst werden mehrere Faltungen parallel durchgeführt, welche dann mehrere Feature Maps produzieren. Um dann Nichtlinearität in das Netzwerk zu bringen, wird eine Aktivierungsfunktion genutzt. Häufig genutzte Aktivierungsfunktionen in CNNs sind Sigmoid σ (oft auch logistische Funktion, siehe Formel 3.12) (Lecun et al., 1998) und rectified linear unit (ReLU, siehe Formel 3.12) (Krizhevsky et al., 2017) (Ronneberger et al., 2015).

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \quad (3.12)$$

$$ReLU(x) = \max(0, x) \quad (3.13)$$

Die ersten entwickelten CNNs benutzten in den einzelnen Schichten Sigmoid als Aktivierungsfunktion (Lecun et al., 1998). Das Problem, unter anderem auch mit der Sigmoid Funktion ist, dass diese sich sättigen. Wenn die Aktivierungsfunktion in den Neuronen des Netzwerks überwiegend Werte ausgibt, welche nah an dem Ende des Aktivierungsfunktionsbereichs liegen, spricht man von einer Sättigung der Aktivierungsfunktion bzw. des Neurons (Rakitianskaia & Engelbrecht, 2015). Wenn bei der Sigmoid Funktion x gegen unendlich geht, dann geht $\sigma(x)$ gegen 1 (Formel 3.14) und wenn x gegen negativ unendlich geht, dann geht $\sigma(x)$ gegen 0 (Formel 3.15).

$$\lim_{x \rightarrow \infty} \frac{1}{1 + e^{-x}} = 1 \quad (3.14)$$

$$\lim_{x \rightarrow -\infty} \frac{1}{1 + e^{-x}} = 0 \quad (3.15)$$

Aus diesem Grund sättigt sich die Sigmoid Funktion $\sigma(x)$ bei 0 für kleine Werte von x und bei 1 für große Werte von x (Goodfellow et al., 2016, S. 68). Wenn bei der Backpropagation dann die Gradients gebildet werden, werden die bestimmten Werte sehr klein oder auch 0 sein, da die Sigmoid Funktion an den Rändern des Funktionsbereichs kaum Steigung hat (siehe Abbildung 3.4).

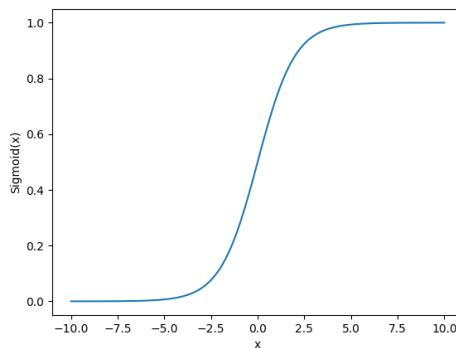


Abbildung 3.4: Sigmoid Funktion

Angenommen, das NN welches trainiert werden soll, hat mehrere gesättigte Neuronen hintereinander geschaltet, welche alle als Aktivierungsfunktion Sigmoid nutzten, dann werden bei den ersten Layern des NN die sehr kleinen Gradients miteinander multipliziert und es entstehen noch kleinere Werte. Dadurch kann das NN seine Gewichte nicht mehr anpassen und es stagniert. Das Problem ist auch unter dem Begriff "vanishing gradient" bekannt (Bengio et al., 1994). Gesättigte Neuronen machen das Gradientenverfahren und damit den Lernprozess des NN langsam und ineffizient (Rakitianskaia & Engelbrecht, 2015).

Zum Schluss der CNN Schicht wird häufig noch eine Pooling Funktion genutzt, welche dafür sorgt, dass die räumliche Dimension der Feature Map reduziert wird (Aggarwal et al., 2018, S. 326). Dadurch wird die Anzahl der Parameter reduziert, welche die nächste Schicht braucht. Das führt dazu, dass das NN nicht überangepasst wird. Zudem wird durch das Pooling versucht, die Informationen zu bewahren. Beim Pooling wird wie bei der Faltung ein Filter, meistens mit der Größe 2×2 oder 3×3 , über die Feature Map geschoben. Dazu wird hier auch die Schrittbreite definiert. Zwei oft genutzte Poolingoperationen sind Max-Pooling und Average-Pooling. Beim Max-Pooling wird bei jedem Schritt der maximale Wert im Fenster genommen und beim Average-Pooling wird der Durchschnitt des überlappenden Fensters berechnet. Wenn z. B. ein Max-Pooling Operation mit der Filtergröße 2×2 und der Schrittgröße 2 auf ein 4×4 Bild angewendet wird, dann ist der Output eine 2×2 Feature Map mit den maximalen Werten von jedem Schritt. Viele erfolgreiche CNN Architekturen wie z. B. das U-Net (Ronneberger et al., 2015) oder das AlexNet (Krizhevsky et al., 2017) nutzen die Pooling-Operation. Allerdings ist diese umstritten. Es wird gesagt, dass unter anderem Max-Pooling Operation durch eine einfache Filterung ersetzt werden könnte, ohne die Genauigkeit des CNNs zu reduzieren (Springenberg et al., 2015).

Die Fehlerfunktion die im CNN genutzt wird, hängt vom Ziel des CNNs ab. Wenn mit dem CNN etwas klassifiziert werden soll, dann wird häufig Kreuzentropie (engl. Cross Entropy (CE)) Fehler genutzt (siehe Formel 3.13) (Aggarwal et al., 2018, S. 118):

$$CE = - \sum_{i=1}^N y_i \log(o_i) \quad (3.16)$$

Wobei N die Anzahl der Klassen, y_i die Klasse und o_i die vorhergesagte Wahrscheinlichkeit für die jeweilige Klasse beschreibt. Für den Fall das N = 2 ist, also eine binäre Klassifikation ausgewertet werden soll, gilt Formel 3.15.

$$BCE = -(y \log(o) + (1 - y) \log(1 - o)) \quad (3.17)$$

Dies wird als binäre Kreuzentropie bezeichnet (engl. Binary Cross Entropy (BCE)) (Lakshmanan et al., 2021, S. 149).

U-Net

Das U-Net (Ronneberger et al., 2015) ist eine CNN Architektur, welches für die Segmentierung von Bildern genutzt werden kann. Es wurde erstmals 2015 vorgestellt und hat seinen Namen aufgrund seiner U-förmigen Architektur (siehe Abbildung 3.5).

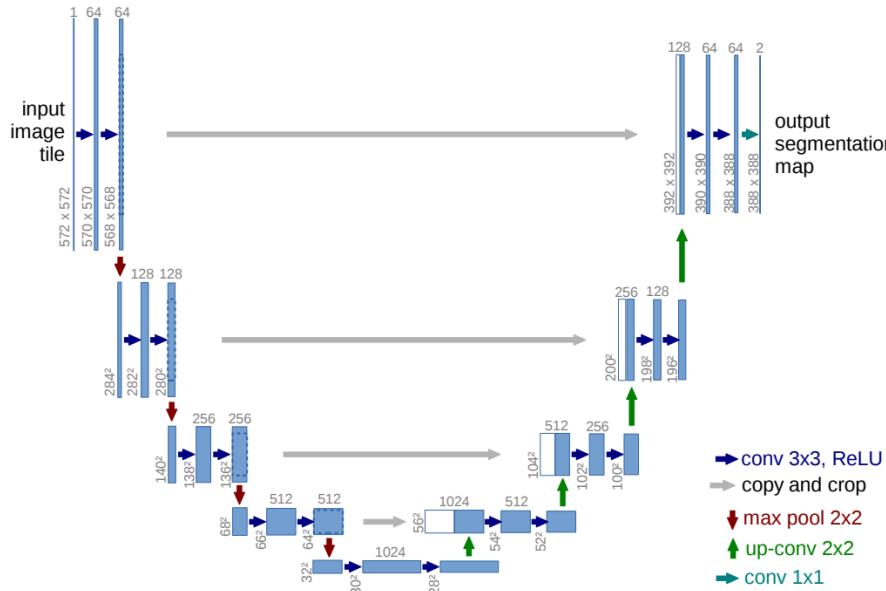


Abbildung 3.5: U-Net Architektur [entnommen aus Quelle: (Ronneberger et al., 2015)].

Es setzt sich zusammen aus einem komprimierenden Pfad (linke Seite) und einem expandierenden Pfad (rechte Seite). Die Pfade bestehen jeweils aus Blöcken. Im komprimierenden Pfad besteht jeder Block aus zwei 3×3 Faltungen mit Schrittweite 1, jeweils gefolgt von einer ReLU. Am Ende jedes Blocks im komprimierenden Pfad wird noch eine 2×2 Max-Pooling Operation mit Schrittweite 2 eingesetzt. Dadurch werden die räumliche Dimension (z. B. bei Bildern die Höhe und Weite) der Feature Maps um die Hälfte reduziert. Zudem wird nach jedem Block die Anzahl der Feature Channels verdoppelt, damit das U-Net komplexere Strukturen lernen kann. Beim expandierenden Pfad wird im jeden Block eine 2×2 Aufwärtsfaltung eingesetzt, welche die Anzahl der Feature Channels halbiert. Anschließend wird die Feature Map mit der Feature Map aus dem entsprechenden komprimierenden Pfad verkettet. Die Feature Map aus dem komprimierenden Pfad muss dabei etwas zugeschnitten werden, da durch die Faltung am Rand Pixel verloren gehen. Zum Schluss der Blöcke im expandierenden Pfad, werden wieder zwei 3×3 Faltungen mit Schrittweite 1, jeweils gefolgt von einer ReLU, eingesetzt. Auf die Ausgabe des letzten Blocks im expandierenden Pfad wird noch eine Faltung eingesetzt um die Ausgabe auf die Anzahl der gewünschten Klassen abzubilden. Im allgemeinen wird der komprimierende Pfad als Encoder bezeichnet und der expandierende Pfad als Decoder.

Generell kann das U-Net eingesetzt werden, um eine semantische Segmentierung durchzuführen. Anfangs wurde das U-Net entwickelt, um biomedizinische Segmentation durchzuführen. Die Forscher, die das U-Net entwickelten, gewannen auf der International Symposium on Biomedical Imaging (ISBI) 2015 mit der Anwendung des U-Nets 3 Wettbewerbe: den Wettbewerb für die Segmentierung neuronaler Strukturen in elektronenmikroskopischen Stapeln, den Wettbewerb für die computergestützte Erkennung von Karies in Bissflügelaufnahme und die Zellverfolgungs-Herausforderung (Ronneberger et al., 2015). Zudem finden leicht modifizierte U-Nets viele Anwendungen in der Medizin, wie z. B. bei der Segmentierung von Tumoren im Gehirn (Mehta & Arbel, 2019) (Dong et al., 2017), bei der Segmentierung von Prostatakrebs (Nowling et al., 2019) oder die Segmentierung von Lungenkrebs (Ait Skourt et al., 2018). Das U-Net findet allerdings auch außerhalb der Medi-

zin Anwendung, z. B. in latent diffusion Modellen um Bildsynthese durchzuführen (Rombach et al., 2022) oder bei der Zerlegung eines Audiosignals in seine Vokalanteile (Jansson et al., 2017).

4. Entwicklung

In diesem Abschnitt geht es um die Entwicklung mehrere Modelle für die Ein-Klassen-Segmentierung von Ulcera am Fuß. Der Abschnitt 4.1 beschäftigt sich mit dem Business and Data Understanding. Unter anderem werden hier die Daten gesammelt und die Metriken festgelegt, an denen die Modelle später ausgewertet werden. Zum Schluss werden noch verwandte Arbeiten vorgestellt. In Abschnitt 4.2 werden die Daten vorverarbeitet. Hier werden die richtigen Daten ausgewählt und standardisiert. Zudem wird gezeigt, wie der Datensatz mit augmentierten Daten erweitert wurde. Im 3. und letzten Abschnitt werden die Modelle und deren Hyperparameter spezifiziert. Zudem werden die entwickelten Modelle mit der 5-Fachen Kreuzvalidierung ausgewertet.

4.1 Business and Data Understanding

In diesem Abschnitt wird auf die erste Phase von CRISP-ML(Q) (Studer et al., 2021), Business and Data Understanding, eingegangen. Business and Data Understanding beschäftigt sich mit dem Identifizieren des Umfangs der ML Anwendung. Ziel ist die Sicherstellung der Durchführbarkeit des Projektes. Zuerst wird das Geschäftsproblem ermittelt und definiert. Zudem werden hier die Daten gesammelt, die Qualität der Daten überprüft und die Metriken festgelegt (Studer et al., 2021). Der Datensatz wurde von der Universität Wisconsin–Milwaukee und dem AZH Center bereitgestellt (<https://github.com/uwm-bigdata/wound-segmentation>). Diese haben den Datensatz mit der Lizenz CC BY-NC 2.0 veröffentlicht. Er besteht insgesamt aus 1210 Bildern, davon gehören 810 Bilder zum Trainingsdatensatz, 200 zum Validierungsdatensatz und 200 zum Testdatensatz (UWM-Bigdata,

2021). Beim Trainings- und Validierungsdatensatz ist zu jedem Bild jeweils noch die Maske (engl. ground truth) bereitgestellt. Alle folgenden Statistiken zu den Daten wurden mit Python erstellt. Dabei wurden die Bibliotheken Pytorch (“Pytorch”, 2022), NumPy (“NumPy”, 2022), Matplotlib (“Matplotlib: Visualization with Python”, 2022) und Pillow (“Pillow”, 2022) verwendet. Zwei Beispiele aus dem Trainingsdatensatz sind in Abbildung 4.1 dargestellt.

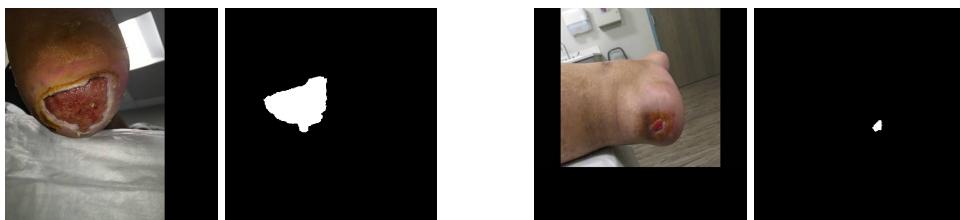


Abbildung 4.1: Bild und Maske aus den Trainingsdaten. Links: ID 228, rechts: ID 15.

Auffällig ist der schwarze Balken am Rand der Bilder. Dieser entsteht aufgrund des Paddings, damit alle Bilder die gleiche Größe haben. Dies wird später wichtig für die Erstellung des Modells. Bei den Daten wird das Padding am rechten, linken oder/und am unteren Bildrand eingesetzt. Alle Bilder haben eine Auflösung von 512×512 Pixel, nutzen als Grafikformat Portable Network Graphics (PNG) und verwenden den RGB Farbraum. Dabei besteht jedes Pixel des Bildes aus 3 Zahlen zwischen 0-255, welche die Intensität von jeweils Rot, Grün und Blau darstellt. Die Masken sind auch 512×512 Pixel groß, nutzen ebenfalls als Grafikformat PNG und als Farbraum RGB. Allerdings haben die Masken auf jedem RGB Kanal die gleich Intensität, deswegen können diese als Grauwertbilder interpretiert werden. Hier besteht jeder Pixel aus einer Zahl zwischen 0-255, wobei 0 Schwarz und 255 Weiß darstellt. Dabei ist das segmentierte Wundbett weiß und der Rest im Bild schwarz. Zwischen den einzelnen Daten variiert es zwischen der Anzahl der Instanzen bzw. zwischen der Anzahl der Wundbetten in einem Bild. Tabelle 4.1 beschreibt, wieviele Bilder wieviele Instanzen haben. Die erste Spalte beschreibt die Anzahl der Instanzen. Die zweite, dritte und vierte beschreiben jeweils, wieviele Bilder es in den Trainingsdaten, Validierungsdaten und insgesamt gibt. Zudem wird noch der

prozentuale Anteil zu dem jeweiligen Datensatz dargestellt.

Tabelle 4.1: Statistiken zu der Anzahl Instanzen pro Bild in den Trainings- und Validierungsdaten

Anzahl Instanzen	Bilder in Trainingsdaten	Bilder in Validierungsdaten	Insgesamt
0	19 (2,3%)	5 (2,5%)	24 (2,3%)
1	618 (76,2%)	151 (75,5%)	769 (76%)
2	118 (14,5%)	30 (15%)	148 (14,6%)
3	37 (4,4%)	7 (3,5%)	44 (4,3%)
4	11 (1,3%)	7 (3,5%)	18 (1,7%)
5	5 (<1%)	4 (2%)	9 (<1%)
6	1 (<1%)	3 (1,5%)	4 (<1%)
22	1 (<1%)	0 (0%)	1 (<1%)
Summe	810 (100%)	200 (100%)	1010 (100%)

Die Mehrheit der Bilder haben nur eine Instanz (76%), gefolgt von Bildern mit zwei (14,6%) und drei Instanzen (4,3%). Bilder mit mehr Instanzen machen ca. 2,8% aller Daten aus. Zudem haben ca 2,3% aller Daten 0 Instanzen. Große Unterschiede zwischen dem prozentualen Anteil der Instanzen in den Trainings- und Validierungsdaten gibt es nicht. Auffällig ist nur, dass es bei den Trainingsdaten eine Maske mit 22 Instanzen gibt. Um die Masken genauer zu untersuchen, wurden Statistiken für die Anzahl der Pixel, die zur segmentierten Wunde gehören, erstellt. Diese sind in Tabelle 4.2 dargestellt

Tabelle 4.2: Pixel Statistiken für die Masken in den Trainings- und Validierungsdaten. Leere Masken wurden hier nicht beachtet.

	Trainingsdaten	Validierungsdaten	Alle Daten
Minimum Pixelanzahl	24	20	20
Maximum Pixelanzahl	34678 (13,2%)	20164 (7,7%)	34678 (13,2%)
Durchschnitt Pixelanzahl	3374 (1,28%)	3203 (1,22%)	3340 (1,27%)
Median	1898	1909	1955
Standardabweichung	4187 (1,59%)	3964 (1,51%)	4143 (1,58%)

Bei der Minimalanzahl an Pixeln, die zu einer segmentierten Wunde gehören, gibt

es keinen Unterschied zwischen Trainings- und Validierungsdaten. Anders sieht es bei der Maximalanzahl aus. Die größte Maske aus den Trainingsdaten hat einen prozentualen Anteil von ca. 13,2%, während die größte Maske in den Validierungsdaten nur einen prozentualen Anteil von ca. 7,7% hat. Der Median und die Standardabweichung zwischen den Trainings- und Validierungsdatensatz haben keinen großen Unterschied. Die Masken mit den geringsten und meisten Pixeln aus dem Trainings- und Validierungsdatensatz sind in Abbildung 4.2 dargestellt.



Abbildung 4.2: Größte und kleinste Maske aus den Trainingsdaten (links) und Validierungsdaten (rechts). Die größte Maske aus den Trainingsdaten hat einen prozentualen Anteil von 13,2% und die größte Maske aus den Validierungsdaten von 7,7% am Bild. IDs von links nach rechts: 71, 995, 816 und 365.

Daraus lässt sich schließen, dass die Größe der segmentierten Wunde sehr variiert. Zum einen hängt das mit der Größe der Wunde zusammen, zum anderen allerdings auch mit der Entfernung und dem Winkel, aus dem die jeweiligen Bilder aufgenommen wurden. Deutlich wird dies, wenn man zwei Beispiele aus den Trainingsdaten betrachtet (siehe Abbildung 4.3).

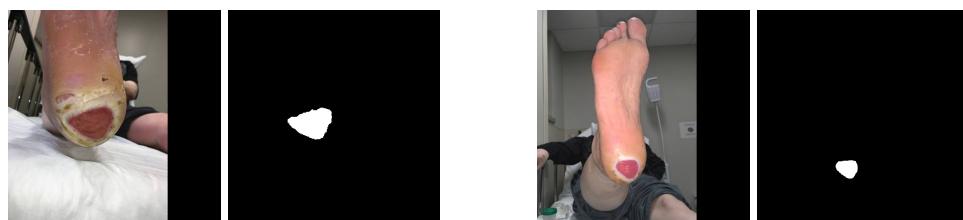


Abbildung 4.3: Größenunterschied zwischen zwei Datensätzen. Links ID 618, rechts ID 719.

Bei den beiden Beispielen in Abbildung 4.3, gibt es einen deutlich anderen Abstand von der Kamera zur Wunde.

Die Statistik in Abbildung 4.4 zeigt die Verteilung des prozentualen Anteil der Pixel, die in den Masken zur segmentierten Wunde gehören.

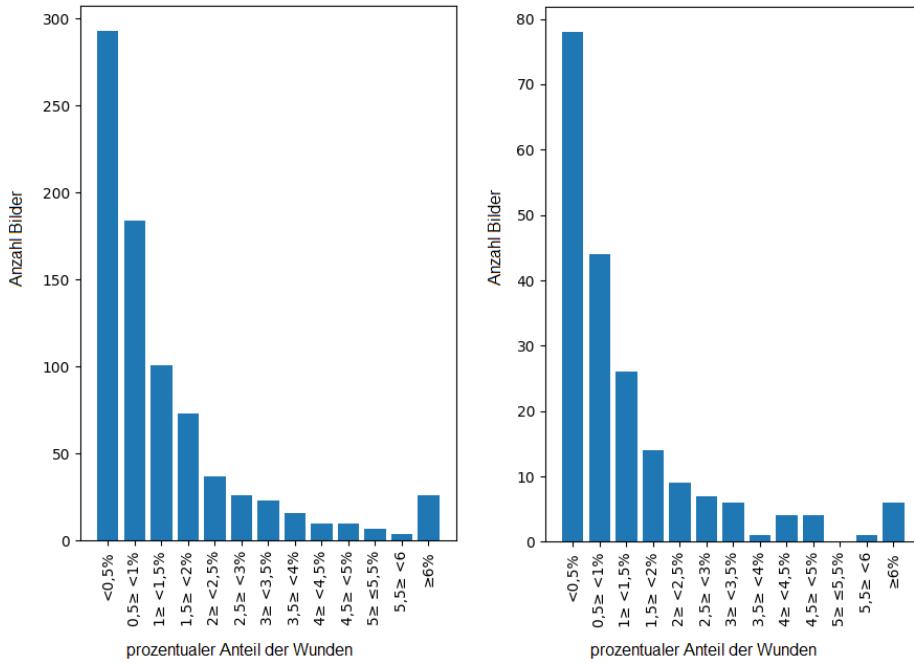


Abbildung 4.4: Die Verteilung des prozentualen Anteils der Pixel, die in den Masken zur segmentierten Wunde gehören. Links: Trainingsdaten, rechts: Validierungsdaten

Mit Abstand die meisten Masken haben nur einen Anteil von weniger als 0,5% am Gesamtbild (293 in den Trainingsdaten und 78 in den Validierungsdaten), gefolgt von denen mit einem Anteil zwischen $\geq 0,5$ und $< 1\%$ (184 in den Trainingsdaten und 44 in den Validierungsdaten). Ab dem Anteil von 2% gibt es in beiden Datensätzen nur noch wenige Exemplare. Daraus lässt sich schließen, dass die Daten überwiegend aus kleinen Instanzen bestehen und es nur relativ wenig große Instanzen gibt.

Unterschiede gibt es auch bei der Position der segmentierten Wunde in den Masken. Dafür wurden die Masken in 4 gleich große Bereiche geteilt (Oben links und rechts, Unten links und rechts) und anschließend bestimmt, in welchem der 4 Bereiche die meisten Pixel der segmentierten Wunde liegen (siehe Abbildung 4.5)

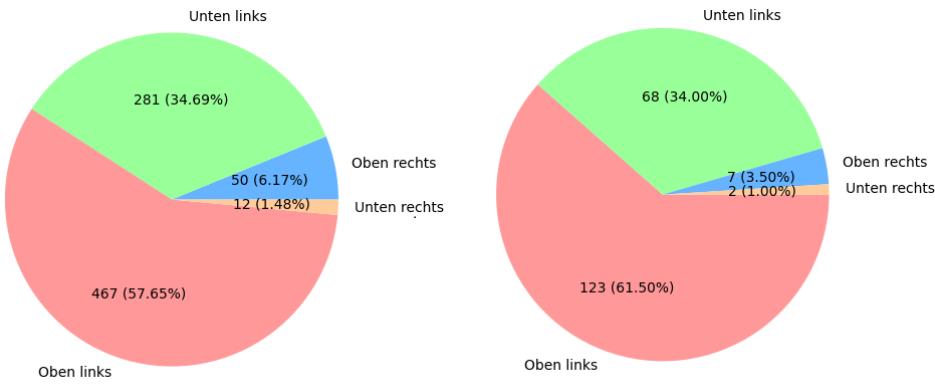


Abbildung 4.5: Position der Masken. Links: Trainingsdaten, Rechts: Validierungsdaten

In den Trainings- und Validierungsdaten liegen die meisten segmentierten Wunden überwiegend oben links (57,67% in den Trainingsdaten und 61,50% in den Validierungsdaten), gefolgt von unten links (34,69% in den Trainingsdaten und 34% in den Validierungsdaten). In beiden Datensätzen liegt auf der rechten Seite der Bilder nur selten die segmentierte Wunde und wenn, dann oben rechts. Dass die Daten größtenteils auf der linken Seite liegen, war zu erwarten, da das Padding vorwiegend auf der rechten Seite des Bildes eingesetzt wird und dadurch die Daten nach links geschoben werden.

Das Ziel ist für jeden Pixel sagen zu können, ob dieser zu der Wunde gehört oder nicht. Dabei handelt es sich um ein Klassifikationsverfahren, da jeder Pixel einer Klasse zugeordnet werden soll. Wenn der Pixel zur Wunde gehört, wird er mit 1 markiert und wenn nicht, dann mit 0. Am Ende erhält man eine Maske aus 0 und 1. Um dann das Modell zu bewerten, wird die vorhergesagte Maske mit der originalen Maske, welche ebenfalls nur aus 0 und 1 besteht, verglichen. Dabei werden Metriken verwendet. Wie bei der Zieldefinition vorgegeben, wird das Modell mit dem Dice-Koeffizient ausgewertet.

Dice-Koeffizient

Der Dice-Koeffizient (Dice, 1945) (englisch: Dice similarity coefficient (DSC)) ist eine Statistik, welche benutzt werden kann, um die Ähnlichkeit von zwei Beispie-

Tabelle 4.3: Confusion Matrix

		Maske	
		Pixel mit 1	Pixel mit 0
Vorhersage	Pixel mit 1	<i>TP</i>	<i>FN</i>
	Pixel mit 0	<i>FP</i>	<i>TF</i>

len zu vergleichen (auch bekannt unter F1-Score). Definiert wurde er 1945 von Lee Raymond Dice. Um den Dice-Koeffizienten zu berechnen, ist es sinnvoll, vier weitere Statistiken für die Segmentierung zu definieren (Lakshmanan et al., 2021, S. 287):

- True positive (TP): Anzahl der Pixel, welche bei der Maske und bei der Vorhersage beide 1 sind
- True negative (TN): Anzahl der Pixel, welche bei der Maske und bei der Vorhersage beide 0 sind
- False positive (FP): Anzahl der Pixel, welche bei der Maske 0 ist aber bei der Vorhersage 1 ist
- False negative (FN): Anzahl der Pixel, welche bei der Maske 1 ist aber bei der Vorhersage 0 ist

Diese können auch in der Confusion Matrix zusammengefasst werden (siehe Tabelle 4.3). Der Dice-Koeffizient lässt sich nun mit Formel 4.1 berechnen.

$$DSC = \frac{2 \times TP}{2 \times TP + FP + FN} \quad (4.1)$$

Da der Koeffizient 1948 auch von Thorvald Sørensen (unabhängig von Lee Raymond Dice) definiert wurde, ist dieser auch unter dem Sørensen-Dice Koeffizient

bekannt. Im folgenden wird die Metrik als Dice bezeichnet. Als Optimalziel wurde definiert, dass das zu entwickelnde Modell ein Dice von mehr als 0,8 bei den Validierungsdaten erzielt.

Jaccard Index

In wissenschaftlichen Arbeiten über DL basierenden Segmentierungen wird oft die Metrik Jaccard Index (Jaccard, 1912) (auch bekannt als Intersection over Union) genutzt, um die Ergebnisse auszuwerten. Der Jaccard Index kann auch aus den vier Statistiken in der Confusion Matrix (Tabelle 4.3) berechnet werden (siehe Formel 4.2).

$$Jaccard = \frac{TP}{TP + FN + FP} \quad (4.2)$$

Im allgemeinen sagt der Jaccard Index wie viel Prozent Überschneidung es zwischen der vorhergesagten Maske und der originalen Maske gibt. Im folgenden wird die Metrik als Jaccard bezeichnet.

4.2 Data Engineering

Die Leistung von Algorithmen für maschinelles Lernen hängt stark von der Darstellung und der Menge der Daten ab (Sun et al., 2017). Beim Data Engineering werden deswegen die Daten vorverarbeitet. Das Ziel ist, ein Datensatz für das zu entwickelnde Modell zu erstellen und sicherzustellen, dass das Modell reproduzierbar ist. In dieser Phase werden unter anderem die richtigen Daten ausgewählt (Data selection), die Daten standardisiert (Data standardization) und der Datensatz mit leicht veränderten Kopien der Daten erweitert (Data Augmentation). Am Ende dieser Phase steht ein Datensatz bereit, welches für das Trainieren der Modelle in Phase 3 des CRISP-ML(Q) genutzt werden kann (Studer et al., 2021).

Data Selection

Data selection beschäftigt sich mit der Auswahl der richtigen Daten für das Datensatz. Hier werden die Daten genauer betrachtet und Daten, welche nicht helfen das Modell zu trainieren oder nicht der Qualität der anderen Daten entsprechen, aussortiert (Studer et al., 2021).

Von den insgesamt 1210 Bildern des bereitgestellten Datensatzes können die 200 Bilder vom Testdatensatz nicht für dieses Projekt genutzt werden, da für diese keine Masken vorliegen. Für das Training und für die Validierung des Modells werden allerdings die Masken benötigt. Wenn man die restlichen 1010 Bilder aus den Trainings- und Validierungsdaten genauer betrachtet, fällt auf, dass eine Anzahl an Masken komplett schwarz sind (siehe Abbildung 4.6).

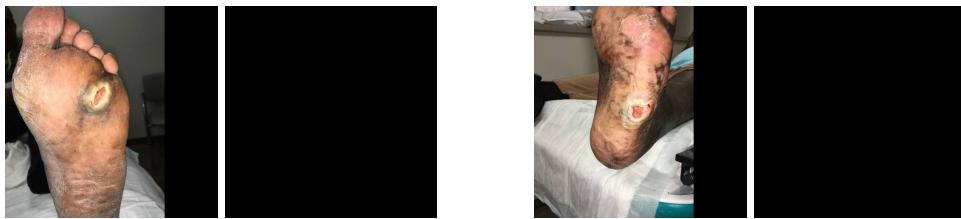


Abbildung 4.6: Bilder mit leerer Maske, welche aussortiert wurden. Links ID 223, rechts ID 417

Insgesamt sind 19 Masken aus dem Trainingsdatensatz und 5 Masken aus dem Validierungsdaten komplett schwarz (siehe Anhang A). Diese 24 Masken werden aussortiert und somit verringert sich der Datensatz auf 986 Bilder (Trainingsdaten 791 Bilder und Validierungsdaten 195 Bilder). Wenn man die leeren Masken für das Training des Modells nutzt, dann würde das Modell schlechtere Ergebnisse liefern (siehe Abschnitt 5). Das Modell würde beim Backpropagating versuchen zwischen der vorhergesagten und der leeren Masken einen Fehler zu bestimmen und aufgrund dieses Fehlers seine Gewichte anpassen. Da die leere Maske allerdings keine Informationen, über das was das Modell eigentlich lernen soll, enthält, würden die Beispiele mit leeren Masken beim Training stören und das Modell würde eine schlechtere Performance haben (siehe Abschnitt 5.1). Zudem würde es bei den leeren Masken zur Verzerrung kommen. Das Modell würde Probleme mit den leeren

Masken haben, da es die leeren Masken keiner Klasse zuordnen kann.

Bei der Datenexploration wurde festgestellt, dass eine Maske 22 Instanzen hat (siehe Abbildung 4.7).



Abbildung 4.7: ID 500 aus den Trainingsdaten mit 22 Instanzen

Anscheinend sollte das Bild nur aus 2 Instanzen bestehen. Bei der linken Instanz wurde vergessen, den Bereich der Wunde auszufüllen. Genau wie bei den leeren Masken, würde das Modell anhand dieses Beispiel nicht genau lernen, was es eigentlich soll. Aus diesem Grund wird dieses Beispiel auch aussortiert. Der ausgewählte Trainingsdatensatz besteht aus insgesamt 790 Bildern und der Validierungsdatensatz besteht aus 195 Bildern.

Data Augmentation

Data Augmentation beschäftigt sich mit dem Erstellen von künstlichen Daten, um die Anzahl an Trainingsdaten zu erhöhen (Goodfellow et al., 2016, S. 236). Die Erweiterung der Trainingsdaten ist sinnvoll, da mehr Daten die Performance der Modelle erhöht (Halevy et al., 2009) (Banko & Brill, 2001). Gerade im Bereich der Computer Vision kann durch das Hinzufügen von künstlichen Daten die Leistung der Modelle erhöht werden (Sun et al., 2017). Aus diesem Grund wird auch der vorliegende Trainingsdatensatz mit augmentierten Bildern erweitert. Bei der Datenexploration wurde festgestellt, dass der Großteil der Wunden auf der linken Seite liegt (siehe Abbildung 4.11). Zudem wurde hier auch gezeigt, dass der Winkel und die Entfernung, aus dem die Bilder aufgenommen, variiert. Deswegen bietet es sich an, die Positionen der Wunde zu verändern und anschließend mit in den Datensatz aufzunehmen. Dafür wurden 3 Augmentierungsverfahren verwendet: Rotation, horizontales/vertikales Flippen und Scherung. Bei der Rotation wurde jedes Bild

zufällig zwischen einem Winkel von -30 bis 30 Grad gedreht und in den Datensatz mit aufgenommen. Hier sei erwähnt, dass bei der Rotation und den anderen Augmentierungsverfahren jeweils das Bild und die Maske mit dem gleichen Parameter verändert werden muss. Wenn zum Beispiel das Bild um 30 Grad gedreht wird, dann muss die Maske auch um 30 Grad gedreht werden. Horizontales/vertikales flippen sorgte dafür, dass jedes Bild aus dem originalen Datensatz zufällig horizontal oder vertikal geflippt wurde und dann auch mit in den Datensatz aufgenommen wurde. Bei der Scherung wurde ebenfalls jedes Bild zufällig mit einem Parameter zwischen -30 und 30 geschert und dem Datensatz hinzugefügt. Dadurch vergrößert sich der Trainingsdatensatz auf 3160 (4×790) Bilder. Durch die statische Augmentierung wird die Reproduzierbarkeit erhöht. Alle Augmentierungen wurden mit Pytorch umgesetzt. Beispiele für die augmentierten Bilder werden in Abbildung 4.14 gezeigt, wobei die erste Spalte das Originalbild, die zweite das rotierte Bild, die dritte das horizontale/vertikale geflippte Bild und die vierte das gescherte Bild zeigt.



Abbildung 4.8: Augmentierte Bilder. Von links nach rechts: Originalbild, rotiertes Bild, horizontales/vertikales Bild und geschertes Bild. Von oben nach unten: ID 11, 216, 410 und 459

Data Standardization

Das Standardisieren der Daten sorgt dafür, dass die Eingabedaten die Eigenschaften einer Standardnormalverteilung haben. Dadurch haben die Daten einen Mittelwert von 0 und eine Standardabweichung von 1. Das ganze wird erreicht, indem von jedem Datensatz der Durchschnitt aller Daten (μ) subtrahiert wird und anschließend durch die Standardabweichung (σ) geteilt wird (siehe Formel 4.3) (Aggarwal et al., 2018, S. 127):

$$x_{\text{standardisiert}} = \frac{x - \mu}{\sigma} \quad (4.3)$$

Das Verfahren ist unter dem Namen Z-Score-Normalisierung bekannt. Durch das Standardisieren der Daten konvergiert das Gradientenverfahren schneller und dadurch wird der Lernprozess effizienter (LeCun et al., 2012). Alle Daten werden standardisiert.

Verwandte Arbeiten

Der Datensatz wurde im Rahmen der Fußulcera Segmentierung Herausforderung 2021 veröffentlicht. Die Bestenliste wurde auf der Website der Challenge bereitgestellt (UWM, 2021). Das Modell der Gewinner (Mahbod et al., 2021) war ein Ensemble aus LinkNet (Chaurasia & Culurciello, 2017) und U-Net. Das LinkNet wurde mit den Gewichten aus einem vortrainierten EfficientNetB1 (Tan & Le, 2019) genutzt und das U-Net mit EfficientNetB2 (Tan & Le, 2019). Zudem nutzten Sie eine Kombination von Dice-Fehler und Focal-Fehler (Lin et al., 2020) und als Optimizer Adam. Das Modell erzielte einen Dice von 0,888. Der zweite Platz nutzte als Modell das U-Net mit HarDNet68 (Chao et al., 2019) als Encoder und erzielte einen Dice von 0.8757 (UWM, 2021). Die Ergebnisse sind nicht mit den Ergebnissen aus dieser Arbeit vergleichbar, da das Ergebnis bei der Herausforderung anhand eines Testdatensatzes berechnet wurde. Da die Masken für den Testdatensatz nicht veröffentlicht wurden, kann kein Ergebniss berechnet werden.

4.3 Machine Learning Model Engineering

In der Machine Learning Modell Engineering Phase ist das Ziel, eins oder mehrere Modelle zu definieren. Im allgemeinen werden hier die Modelle ausgewählt, spezialisiert und trainiert. Zudem sollten hier alle Metadaten, wie z. B. die Lernrate und die Batch Größe, gesammelt werden, um die Reproduzierbarkeit des Modells sicherzustellen (Studer et al., 2021).

Ziel des Modells ist es für jeden Pixel zu sagen, ob es zur Wunde gehört oder nicht. Wenn der Pixel zur Wunde gehört wird er mit 1 markiert und wenn nicht mit 0. Das Modell verwendet die Bilder als Eingabe und bildet daraus dann eine Maske. Dementsprechend sollte das zu entwickelnde Modell eine Eingabe von

$512 \times 512 \times 3$ und eine Ausgabe von $512 \times 512 \times 1$ haben. In Abschnitt 3.3 wurde unter anderem auch das U-Net vorgestellt, welches gute Ergebnisse in Medizinischen Segmentierungen liefert. Aus diesem Grund wird das U-Net auch für diese Aufgabe als Basis-Architektur verwendet. Um die Modelle zu vervollständigen, müssen die Hyperparameter definiert werden. Mit unterschiedlichen Hyperparametern werden im folgenden mehrere Modelle definiert. Das U-Net wird mit der Encoder-Tiefe 3, 4 und 5 genutzt, d.h. dass das Modell aus 3, 4 bzw. 5 Schichten im Encoder und 2, 3 bzw. 4 im Decoder besteht (siehe Abschnitt 3.3). Die Modelle werden mit dem Mini-Batch Gradientenverfahren trainiert, hier wird als Batch Größe 16 genutzt. Als Fehlerfunktion wird unabhängig voneinander in verschiedenen Experimenten die Binäre Kreuzentropie und die Dice Fehlerfunktion verwendet. Der Dice-Koeffizient wurde in Abschnitt 4.1 als Metrik definiert, dieser kann auch als Fehlerfunktion genutzt werden (Sudre et al., 2017). Als Optimizer wird in allen Modellen Adam mit Lernrate 0,001, $\beta_1 = 0.9$, $\beta_2 = 0.999$ und $\varepsilon = 10^{-8}$ verwendet (siehe Abschnitt 3.2). Zudem wurde bei allen Modellen als Encoder ResNet 34 (He et al., 2016) verwendet. Im ML Bereich hat sich gezeigt, dass die Gewichte von einem Modell zu einem anderen Modell übertragbar sind, um eine schnellere Generalisierung zu erreichen (Yosinski et al., 2014). Aus diesem Grund, wurden alle Modelle vorher mit den Gewichten aus einem vortrainierten Modell auf ImageNet-1k (Deng et al., 2009) initialisiert. Die einzelnen Modelle sind in Tabelle 4.4 definiert.

Tabelle 4.4: Modelle

	U3_Dice	U4_Dice	U5_Dice	U3_BCE	U4_BCE	U5_BCE
Encoder Tiefe	3	4	5	3	4	5
Fehlerfunktion	Dice	Dice	Dice	BCE	BCE	BCE
Anz. Parameter	3.009.153	10.338.273	24.436.369	3.009.153	10.338.273	24.436.369

Beim Training der Modelle wurde 5-fache Kreuzvalidierung verwendet um die Modelle zu validieren. Der Trainingsdatensatz wurde in 5 gleich große Datensätze geteilt und anschließend wurde der Durchschnittliche Dice und Jaccard ermittelt. Alle Modelle wurden 80 Epochs lang trainiert. Für das Trainig wurde die NVIDIA TITAN X verwendet. Die Laufzeit für das Training der U-Nets mit 5 Schichten

im Encoder beträgt ca. 196 Minuten, mit 4 Schichten ca. 171 Minuten und mit 3 Schichten ca. 167 Minuten. Die Ergebnisse für die Kreuzvalidierung sind in Tabelle 4.5 dargestellt.

Tabelle 4.5: Durchschnittlicher Dice und Jaccard für die einzelnen Modelle bei der 5-fachen Kreuzvalidierung. In den Klammern steht jeweils noch die Standardabweichung der Ergebnisse.

	U3_Dice	U4_Dice	U5_Dice	U3_BCE	U4_BCE	U5_BCE
Ø-Dice	0,8666 (0,05)	0,8953 (0,05)	0,8989 (0,05)	0,8523 (0,05)	0,9023 (0,05)	0,9091 (0,05)
Ø-Jaccard	0 (0,03)	0 (0,03)	0 (0,03)	0,7842 (0,03)	0,8083 (0,03)	0,8158 (0,03)

Die besten Modelle sind U4_BCE und U5_BCE mit einem Dice-Ergebnis von 0,90. U4_Dice und U5_Dice schneiden etwas schlechter ab, mit einem Dice-Ergebnis von 0,89. Die U-Nets mit 3 Schichten zeigen eine deutlich schlechtere Performance. U3_Dice hat nur ein Dice-Ergebnis von 0,86 und U3_BCE von 0,85.

5. Ergebnisse und Diskussion

In diesem Abschnitt wird auf die vierte Phase aus dem CRISP-ML(Q), Evaluating Machine Learning Models, eingegangen. Abschnitt 5.1 zeigt die Ergebnisse der einzelnen Modelle. In Abschnitt 5.2 werden die Ergebnisse diskutiert und bewertet.

5.1 Ergebnisse

Zum Testen der Modelle werden diese mit dem bearbeiteten Trainingsdatensatz aus Abschnitt 4 trainiert und anschließend die Ergebnisse mit dem bearbeiteten Validierungsdaten aus Abschnitt 4 ermittelt. Wie bei der Kreuzvalidierung in Abschnitt 4, wurde für das Training eine NVIDIA TITAN X genutzt und 80 Epochs lang trainiert. Es wurde die Batch Größe 16 genutzt und als Optimizer wird in allen Modellen Adam mit Lernrate 0,001, $1 = 0.9$, $2 = 0.999$ und $\gamma = 108$ verwendet. Die Laufzeit der U-Nets mit 5 Schichten im Encoder beträgt ca. 196 Minuten, mit 4 Schichten ca. 171 Minuten und mit 3 Schichten ca. 167 Minuten. Bei den Ergebnissen in Tabelle 5.1 wurde jeweils der höchste Dice bzw. Jaccard der Modelle notiert.

Tabelle 5.1: Höchster Dice und Jaccard für die Modelle anhand Validierungsdaten

	U3_Dice	U4_Dice	U5_Dice	U3_BCE	U4_BCE	U5_BCE
Dice	0,8789	0,9102	0,9160	0,8913	0,9203	0,9221
Jaccard	0,7752	0,8454	0,8449	0,7826	0,8514	0,8555

Die beste Performance zeigt das Modell U5_BCE mit Dice 0,9221 und Jaccard 0,8555. Etwas schlechter ist das U4_BCE mit Dice 0,9203 und Jaccard 0,8514. Die

Modelle mit der Dice-Fehlerfunktion schneiden etwas schlechter ab. Das U5_Dice erzielt ein Dice von 0,9160 und Jaccard von 0,8449. Am schlechtesten sind die Modelle mit nur 3 Schichten im Encoder. Das U3_Dice erzielt ein Dice von 0,8789 und Jaccard von 0,7752. Im Vergleich dazu, ist das U3_BCE etwas besser mit einem Dice von 0,8913 und Jaccard von 0,7826.

Das U5_BCE wurde genutzt, um zu vergleichen wie die Ergebnisse variieren, wenn die Augmentierung und/oder die Aussortierung der Masken aus Abschnitt 4 beim Training weggelassen wird. Die Ergebnisse wurden mit dem modifizierten Validierungsdatensatz aus Abschnitt 4 gemessen und sind in Tabelle 5.2 dargestellt.

Tabelle 5.2: Dice und Jaccard für unterschiedlich modifizierte Datensätze. Die Ergebnisse wurden mit dem modifizierten Validierungsdatensatz aus Abschnitt 4 gemessen.

	Dice	Jaccard
Datensatz mit Augmentierung, mit Aussortieren	0,9221	0,8555
Datensatz mit Augmentierung, ohne Aussortieren	0,9208	0,8532
Datensatz ohne Augmentierung, mit Aussortieren	0,9081	0,8361
Datensatz ohne Augmentierung, ohne Aussortieren	0,9031	0,8267

Die beste Performance erreicht das U5_BCE mit dem augmentierten und aussortierten Datensatz.

5.2 Diskussion

Die Modelle U4_BCE und U5_Dice haben die besten Ergebnisse. Auffällig ist, dass das U4_BCE weniger als die Hälfte an trainierbaren Parametern (10.338.273 zu 24.436.369 Parameter) hat als das U5_BCE und trotzdem fast die gleichen Ergebnisse liefert. Die Erweiterung des originalen Datensatzes mit augmentierten Bildern, zeigt, dass das U5_BCE den Dice um 0,0177 Punkte (von 0,9031 auf 0,9208) und Jaccard um 0,0265 Punkte (von 0,8267 auf 0,8532) erhöht. Wenn der augmentierte Datensatz zusätzlich noch, wie in Abschnitt 4 beschrieben, bearbeitet wird, dann

kann das U5_BCE den Dice um 0,0013 und den Jaccard um 0,0023 erhöhen.

Im folgenden wird das Modell U5_BCE genauer ausgewertet. Abbildung 5.1 zeigt gute Ergebnisse des Modells U5_BCE mit einem Dice-Ergebnis von > 0,95 und Abbildung 5.2 schlechte Ergebnisse mit einem Dice-Ergebnis von < 0,6.

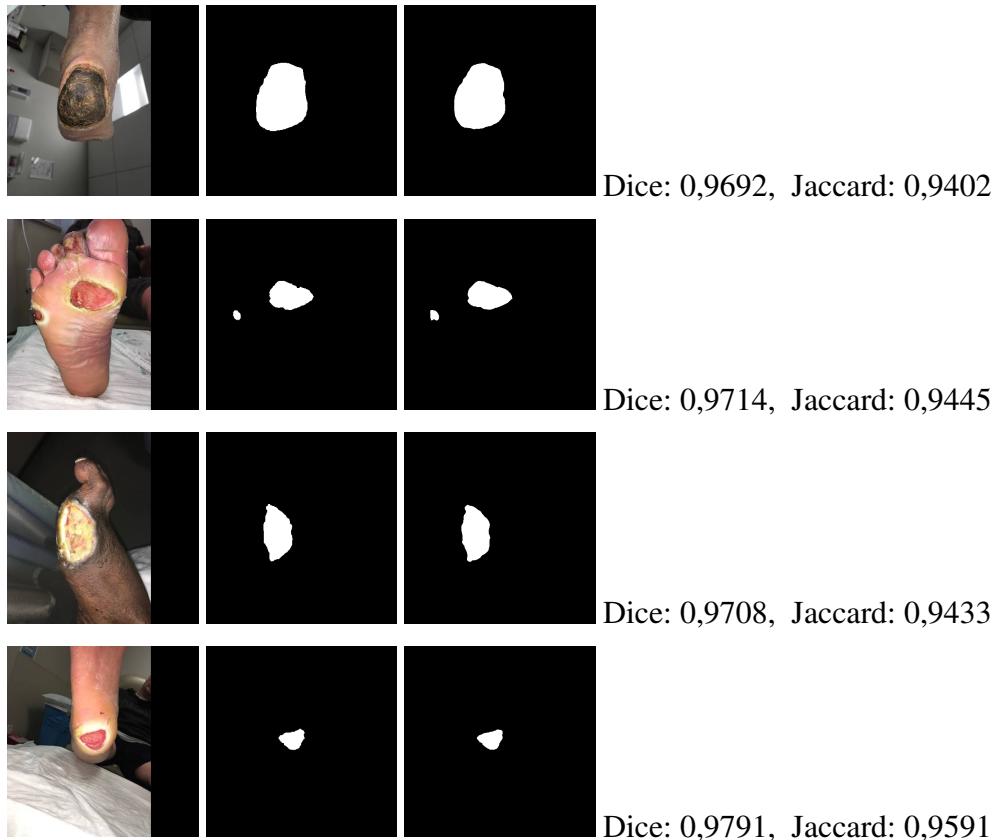


Abbildung 5.1: Gute Vorhersage des Modells U5_BCE mit Dice-Ergebnis > 0,95. Links: Bild, Mitte: Maske, rechts: Vorhersage. Von oben nach unten: ID 365, 545, 438, 503

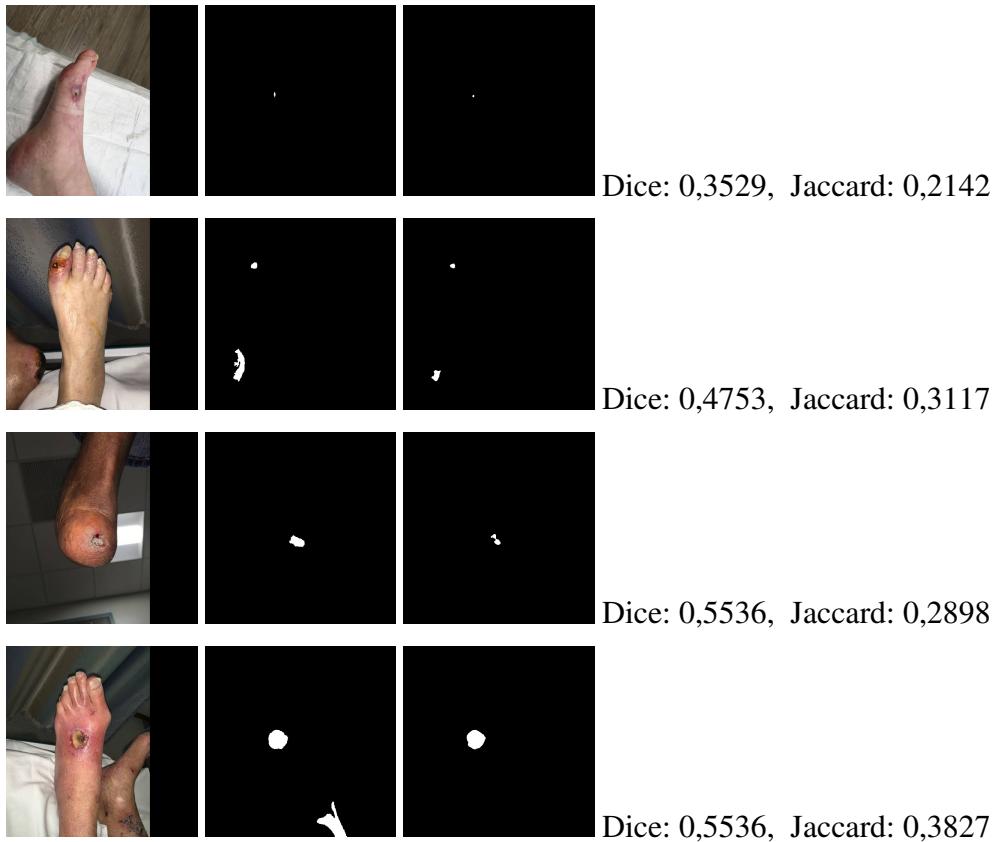


Abbildung 5.2: Schlechte Vorhersage des Modells U5_BCE mit Dice-Ergebnis < 0,6. Links: Bild, Mitte: Maske, rechts: Vorhersage. Von oben nach unten: ID 79, 349, 365, 548

Wenn das Wundbett in der Mitte liegt und etwas größer ist, dann zeigt das Modell gute Ergebnisse (siehe z. B. Bild 365). Zudem kann das Modell auch gute Ergebnisse liefern, wenn z.B. zwei Instanzen im Vordergrund sind (siehe z. B. Bild 545). Das Modell performt schlechter, wenn das Wundbett sehr klein ist (siehe z. B. Bild 79). Im Bild 349 liegt eine schwarze Ulcus Instanz im Hintergrund, mit der das Modell Probleme hat. In Bild 548 ist ebenfalls eine schwarze Ulcus Instanz im Hintergrund, welche vom Modell nicht erkannt wird. Das Modell hat oft Schwierigkeiten, wenn eine Instanz nicht im Vordergrund liegt. Zudem hat das Modell Probleme mit anderen Flecken auf der Haut (siehe Abbildung 5.3).



Abbildung 5.3: Vorhersage des Modells mit Falsch-Positiver Instanz. ID 412

Beim Bild 412 hat das Modell eine Falsch-Positive Instanz vorhergesagt. Das Modell klassifiziert den orangen Fleck auf dem Fuß als eigene Instanz.

Das hier entwickelte Modell kann auch für weite Ulcus Wunddaten verwendet werden. Allerdings nur für Bilder mit einer Auflösung von 512×512 . Zudem sollten diese Wunddaten die Ulcera Instanzen im Vordergrund haben, da sonst das Modell Probleme hat diese zu erkennen. Auch sollte die Verfahrensanweisung für die Bildaufnahme bei den neuen Daten ähnlich zu der in diesem Datensatz sein.

6. Zusammenfassung

In dieser Projektarbeit wurde ein Deep Learning Workflow zur Ein-Klassen-Segmentierung von Ulcera am Fuß entwickelt. Zuerst wurde die Vorgehensweise beschrieben und Methoden im Bereich Computer Vision gezeigt. Hier wurde der Deep Learning Workflow genauer erläutert und das das Prozessmodell CRISP-ML(Q) vorgestellt. Anschließend wurde im Abschnitt Grundlagen die Krankheit Ulcus vorgestellt. Zudem wurde gezeigt, wie Künstliche Neuronale Netze funktionieren und es wurden CNNs vorgestellt. Danach wurde gezeigt die Entwicklung des Deep Learning Workflows gezeigt. Dabei wurde auf die einzelnen Phasen Business and Data Understanding, Data Engineering und Machine Learning Model Engineering eingegangen. Hier wurden unter anderem augmentierte Daten erstellt und es wurden mehrere U-Nets mit unterschiedlichen Hyperparametern trainiert und deren Leistung evaluiert. Zum Schluss wurden die Ergebnisse der einzelnen U-Nets ausgewertet und diskutiert. Die beste Performance lieferte das U-Net mit der Fehlerfunktion BCE und mit 5 Schichten im Encoder. Das Modell erreichte ein Dice von 0,9221 und Jaccard von 0,8555 anhand der Validierungsdaten. Verbessert werden die Modelle indem noch mehr Trainingsdaten gesammelt werden. Zusätzlich können andere Optimierer, Fehlerfunktionen, Parameter zu einer besseren Performance führen. Zudem können noch bessere Augmentierungsverfahren verwendet werden, wie z. B. Farbverschiebungen, verschiedene Rauschmuster, (Un-)Schärfe, Dropout, Helligkeits- und Kontrastanpassungen. Dadurch lässt sich der Datensatz noch erweitern und es kann eine noch bessere Performance geben. Des weiteren können noch modifizierte U-Nets, wie z. B. das U-Net++ (Zhou et al., 2018), oder andere Segmentierungsmodelle, wie z. B. das DeepLabv3++ (Chen et al., 2018) oder das LinkNet (Chaurasia & Culurciello, 2017) verglichen werden. Ensemble-

methoden, also z. B. die Kombinierung von U-Net und DeepLabv3++, könnten auch noch bessere Ergebnisse liefern. Zusätzlich kann die Kombination von Fehlerfunktionen genutzt werden, wie z. B. von BCE und Dice-Fehler, damit das Modell eine bessere Generalisierung erreicht.

A. Aussortierte Bilder

Im Folgenden werden die Bilder aufgelistet, welche aus dem Datensatz entfernt wurden. Bilder aus dem Trainingsdaten mit komplett schwarzer Maske: 92, 131, 223, 281, 335, 361, 457, 502, 564, 600, 701, 725, 736, 778, 832, 899, 912, 931, 950. Bilder aus dem Validierungsdaten mit komplett schwarzer Maske: 128, 417, 483, 533, 869.

Literatur

- Norvig, P. & Russell, S. J. (2010). *Artificial intelligence a modern approach*. Pearson Education, Inc., Auflage 4.
- Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep Learning* (Bd. 1) [<http://www.deeplearningbook.org>]. MIT Press, Auflage 1.
- May, M. (2021). Eight ways machine learning is assisting medicine. *Nature Medicine*, 27(1), 2–3. <https://doi.org/10.1038/s41591-020-01197-2>
- Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324. <https://doi.org/10.1109/5.726791>
- Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N. & Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29(6), 82–97. <https://doi.org/10.1109/MSP.2012.2205597>
- Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2017). ImageNet Classification with Deep Convolutional Neural Networks. *Communications of the ACM*, 60(6), 84–90. <https://doi.org/10.1145/3065386>
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. <https://doi.org/10.1109/CVPR.2009.5206848>
- Studer, S., Bui, T. B., Drescher, C., Hanuschkin, A., Winkler, L., Peters, S. & Müller, K.-R. (2021). Towards CRISP-ML (Q): a machine learning process mo-

- del with quality assurance methodology. *Machine Learning and Knowledge Extraction*, 3(2), 392–413. <https://doi.org/10.48550/arXiv.2003.05155>
- Jensen, K. (2012). CRISP-DM Prozess Diagramm. https://commons.wikimedia.org/wiki/File:CRISP-DM_Process_Diagram.png.
- IBM. (2016). IBM SPSS Modeler CRISP-DM Guide. <http://public.dhe.ibm.com/software/analytics/spss/documentation/modeler/18.0/en/ModelerCRISPDM.pdf>.
- Chollet, F. (2017). *Deep learning with Python* (Bd. 1). Manning, Auflage 1.
- Lakshmanan, V., Görner, M. & Gillard, R. (2021). *Practical Machine Learning for Computer Vision* (Bd. 1). O'Reilly Media, Inc., Auflage 1.
- Eberstaller, S. (2012). CRISP-ML(Q) Diagramm. <https://ml-ops.org/content/crisp-ml>.
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. & Houlsby, N. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *International Conference on Learning Representations*. <https://openreview.net/forum?id=YicbFdNTTy>
- Wang, W., Dai, J., Chen, Z., Huang, Z., Li, Z., Zhu, X., Hu, X., Lu, T., Lu, L., Li, H., Wang, X. & Qiao, Y. (2022a). InternImage: Exploring Large-Scale Vision Foundation Models with Deformable Convolutions. <https://doi.org/10.48550/ARXIV.2211.05778>
- Eckert, M., Volmerg, J. S. & Friedrich, C. M. (2019). Augmented Reality in Medicine: Systematic and Bibliographic Review. *JMIR Mhealth Uhealth*, 7(4), e10967. <https://doi.org/10.2196/10967>
- Kumar, A., Kaur, A. & Kumar, M. (2019). Face Detection Techniques: A Review. *Artificial Intelligence Review*, 52. <https://doi.org/10.1007/s10462-018-9650-2>
- Ecabert, O., Peters, J., Schramm, H., Lorenz, C., von Berg, J., Walker, M. J., Vembunar, M., Olszewski, M. E., Subramanyan, K., Lavi, G. & Weese, J. (2008). Automatic Model-Based Segmentation of the Heart in CT Images. *IEEE*

Transactions on Medical Imaging, 27(9), 1189–1201. <https://doi.org/10.1109/TMI.2008.918330>

Pereira, S., Pinto, A., Alves, V. & Silva, C. A. (2016). Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images. *IEEE Transactions on Medical Imaging*, 35(5), 1240–1251. <https://doi.org/10.1109/TMI.2016.2538465>

Kirillov, A., He, K., Girshick, R., Rother, C. & Dollár, P. (2019). Panoptic Segmentation. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 9396–9405. <https://doi.org/10.1109/CVPR.2019.00963>

Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A. & Torralba, A. (2017). Scene Parsing through ADE20K Dataset. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5122–5130. <https://doi.org/10.1109/CVPR.2017.544>

Zhou, B., Zhao, H., Puig, X., Xiao, T., Fidler, S., Barriuso, A. & Torralba, A. (2019). Semantic Understanding of Scenes Through the ADE20K Dataset. *International Journal of Computer Vision*, 127(3), 302–321. <https://doi.org/10.1007/s11263-018-1140-0>

Wang, W., Bao, H., Dong, L., Bjorck, J., Peng, Z., Liu, Q., Aggarwal, K., Mohammed, O. K., Singhal, S., Som, S. & Wei, F. (2022b). Image as a Foreign Language: BEiT Pretraining for All Vision and Vision-Language Tasks. *arXiv e-prints*. <https://doi.org/10.48550/arXiv.2208.10442>

Dissemund, J., Bültmann, A., Gerber, V. et al. (2020). Standards für die Diagnostik und Therapie chronischer Wunden Stand 2020 (Positionspapier der Initiative chronische Wunde e. V.) <https://www.icwunden.de/wundwissen/standardsdefinitionen/>.

Hermanns, H. (2016). Chirurgie des Ulcus cruris—Eine aktuelle Übersicht. *Vasomed*, 28(04), 1–7.

Diabetic Foot Ulcer Treatment Market. (2021). <https://www.precedenceresearch.com/diabetic-foot-ulcer-treatment-market>.

Sun, H., Saeedi, P., Karuranga, S., Pinkepank, M., Ogurtsova, K., Duncan, B. B., Stein, C., Basit, A., Chan, J. C., Mbanya, J. C. et al. (2022). IDF Diabetes

- Atlas: Global, regional and country-level diabetes prevalence estimates for 2021 and projections for 2045. *Diabetes research and clinical practice*, 183, 109119. <https://doi.org/10.1016/j.diabres.2021.109119>
- Akhtar, S., Ali, A., Ahmad, S., Khan, M. I., Shah, S. & Hassan, F. (2022). The prevalence of foot ulcers in diabetic patients in Pakistan: A systematic review and meta-analysis. *Frontiers in Public Health*, 10. <https://doi.org/10.3389/fpubh.2022.1017201>
- Aggarwal, C. C. et al. (2018). *Neural networks and deep learning* (Bd. 1). Springer, Auflage 1.
- Cauchy, A.-L. (1847). Methode generale pour la resolution des systemes d'equations simultanees. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 1.
- Kingma, D. P. & Ba, J. (2015). Adam: A Method for Stochastic Optimization. In Y. Bengio & Y. LeCun (Hrsg.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. <http://arxiv.org/abs/1412.6980>
- Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems* (Bd. 2). O'Reilly Media, Inc., Auflage 2.
- Bracewell, R. (1986). *The Fourier Transform Its Application* (Bd. 3). McGraw-Hill Science/Engineering/Math, Auflage 4.
- Pytorch. (2022). Pytorch. <https://pytorch.org/docs/>.
- Ronneberger, O., Fischer, P. & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In N. Navab, J. Hornegger, W. M. Wells & A. F. Frangi (Hrsg.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (S. 234–241). Springer International Publishing. https://doi.org/10.1007/978-3-319-24574-4_28
- Rakitianskaia, A. & Engelbrecht, A. (2015). Measuring Saturation in Neural Networks. *2015 IEEE Symposium Series on Computational Intelligence*, 1423–1430. <https://doi.org/10.1109/SSCI.2015.202>

- Bengio, Y., Simard, P. & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5, 157–66. <https://doi.org/10.1109/72.279181>
- Springenberg, J. T., Dosovitskiy, A., Brox, T. & Riedmiller, M. A. (2015). Striving for Simplicity: The All Convolutional Net. In Y. Bengio & Y. LeCun (Hrsg.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*. <http://lmb.informatik.uni-freiburg.de/Publications/2015/DB15a>
- Mehta, R. & Arbel, T. (2019). 3D U-Net for Brain Tumour Segmentation. In A. Crimi, S. Bakas, H. Kuijf, F. Keyvan, M. Reyes & T. van Walsum (Hrsg.), *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries* (S. 254–266). Springer International Publishing. https://doi.org/10.1007/978-3-030-11726-9_23
- Dong, H., Yang, G., Liu, F., Mo, Y. & Guo, Y. (2017). Automatic Brain Tumor Detection and Segmentation Using U-Net Based Fully Convolutional Networks. In M. Valdés Hernández & V. González-Castro (Hrsg.), *Medical Image Understanding and Analysis* (S. 506–517). Springer International Publishing. https://doi.org/10.1007/978-3-319-60964-5_44
- Nowling, R. J., Bukowy, J., McGarry, S. D., Nencka, A. S., Blasko, O., Urbain, J., Lowman, A., Barrington, A., Banerjee, A., Iczkowski, K. A. & LaViolette, P. S. (2019). Classification before Segmentation: Improved U-Net Prostate Segmentation. *2019 IEEE EMBS International Conference on Biomedical Health Informatics (BHI)*, 1–4. <https://doi.org/10.1109/BHI.2019.8834494>
- Ait Skourt, B., El Hassani, A. & Majda, A. (2018). Lung CT Image Segmentation Using Deep Neural Networks [PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON INTELLIGENT COMPUTING IN DATA SCIENCES, ICDS2017]. *Procedia Computer Science*, 127, 109–113. <https://doi.org/10.1016/j.procs.2018.01.104>
- Rombach, R., Blattmann, A., Lorenz, D., Esser, P. & Ommer, B. (2022). High-Resolution Image Synthesis With Latent Diffusion Models. *Proceedings*

- of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 10684–10695. <https://doi.org/10.48550/arXiv.2112.10752>
- Jansson, A., Humphrey, E., Montecchio, N., Bittner, R., Kumar, A. & Weyde, T. (2017). *Singing voice separation with deep U-Net convolutional networks*. <https://openaccess.city.ac.uk/id/eprint/19289/>
- UWM-Bigdata. (2021). Wound segmentation. *GitHub repository*. <https://github.com/uwm-bigdata/wound-segmentation>.
- NumPy. (2022). *NumPy*. <https://numpy.org/>.
- Matplotlib: Visualization with Python. (2022). *Matplotlib*. <https://matplotlib.org/>.
- Pillow. (2022). *Pillow*. <https://pillow.readthedocs.io/en/stable/>.
- Dice, L. R. (1945). Measures of the Amount of Ecologic Association Between Species. *Ecology*, 26(3), 297–302. <https://doi.org/10.2307/1932409>
- Jaccard, P. (1912). THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1. *New Phytologist*, 11(2), 37–50. <https://doi.org/https://doi.org/10.1111/j.1469-8137.1912.tb05611.x>
- Sun, C., Shrivastava, A., Singh, S. & Gupta, A. (2017). Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. *2017 IEEE International Conference on Computer Vision (ICCV)*, 843–852. <https://doi.org/10.1109/ICCV.2017.97>
- Halevy, A., Norvig, P. & Pereira, F. (2009). The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems*, 24, 8–12. http://www.computer.org/portal/cms_docs_intelligent/intelligent/homepage/2009/x2exp.pdf
- Banko, M. & Brill, E. (2001). Mitigating the Paucity-of-Data Problem: Exploring the Effect of Training Corpus Size on Classifier Performance for Natural Language Processing. *Proceedings of the First International Conference on Human Language Technology Research*, 1–5. <https://doi.org/10.3115/1072133.1072204>
- LeCun, Y. A., Bottou, L., Orr, G. B. & Müller, K.-R. (2012). Efficient BackProp. In G. Montavon, G. B. Orr & K.-R. Müller (Hrsg.), *Neural Networks: Tricks of the Trade: Second Edition* (S. 9–48). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-35289-8_3

- UWM. (2021). Wound Segmentation challenge leaderboard. <https://uwm-bigdata.github.io/wound-segmentation/>.
- Mahbod, A., Schaefer, G., Ecker, R. & Ellinger, I. (2021). Automatic Foot Ulcer Segmentation Using an Ensemble of Convolutional Neural Networks. <https://doi.org/10.48550/ARXIV.2109.01408>
- Chaurasia, A. & Culurciello, E. (2017). LinkNet: Exploiting encoder representations for efficient semantic segmentation. *2017 IEEE Visual Communications and Image Processing (VCIP)*, 1–4. <https://doi.org/10.1109/VCIP.2017.8305148>
- Tan, M. & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In K. Chaudhuri & R. Salakhutdinov (Hrsg.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA* (S. 6105–6114). PMLR. <http://proceedings.mlr.press/v97/tan19a.html>
- Lin, T., Goyal, P., Girshick, R. B., He, K. & Dollár, P. (2020). Focal Loss for Dense Object Detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(2), 318–327. <https://doi.org/10.1109/TPAMI.2018.2858826>
- Chao, P., Kao, C.-Y., Ruan, Y., Huang, C.-H. & Lin, Y.-L. (2019). HarDNet: A Low Memory Traffic Network. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 3551–3560. <https://doi.org/10.1109/ICCV.2019.00365>
- Sudre, C. H., Li, W., Vercauteren, T., Ourselin, S. & Jorge Cardoso, M. (2017). Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations. In M. J. Cardoso, T. Arbel, G. Carneiro, T. Syeda-Mahmood, J. M. R. Tavares, M. Moradi, A. Bradley, H. Greenspan, J. P. Papa, A. Madabhushi, J. C. Nascimento, J. S. Cardoso, V. Belagiannis & Z. Lu (Hrsg.), *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support* (S. 240–248). Springer International Publishing. https://doi.org/10.1007/978-3-319-67558-9_28

- He, K., Zhang, X., Ren, S. & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. <https://doi.org/10.1109/CVPR.2016.90>
- Yosinski, J., Clune, J., Bengio, Y. & Lipson, H. (2014). How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence & K. Weinberger (Hrsg.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc. <https://doi.org/10.1109/ICASSP.2019.8683043>
- Zhou, Z., Rahman Siddiquee, M. M., Tajbakhsh, N. & Liang, J. (2018). UNet++: A Nested U-Net Architecture for Medical Image Segmentation. In D. Stoyanov, Z. Taylor, G. Carneiro, T. Syeda-Mahmood, A. Martel, L. Maier-Hein, J. M. R. Tavares, A. Bradley, J. P. Papa, V. Belagiannis, J. C. Nascimento, Z. Lu, S. Conjeti, M. Moradi, H. Greenspan & A. Madabhushi (Hrsg.), *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support* (S. 3–11). Springer International Publishing.
- Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F. & Adam, H. (2018). Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In V. Ferrari, M. Hebert, C. Sminchisescu & Y. Weiss (Hrsg.), *Computer Vision – ECCV 2018* (S. 833–851). Springer International Publishing. https://doi.org/10.1007/978-3-030-01234-2_49

Eigenständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig angefertigt und mich keiner fremden Hilfe bedient sowie keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen, die wörtlich oder sinngemäß veröffentlichten oder nicht veröffentlichten Schriften und anderen Quellen entnommen sind, habe ich als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Dortmund, den 13. Januar 2023