FernUniversität in Hagen
Fakultät für Mathematik und Informatik

VITRONIC
the machine vision people

Bewertung von Segmentierungsverfahren, die auf neuronalen Netzen basieren, für die Detektion von Schweißnahtgrenzen

Masterarbeit

zur Erlangung des Grades einer Master of Science (M.Sc.)
im Studiengang Informatik

vorgelegt von
Badr El Yamouni

Erstgutachter:	Prof. Dr. Matthias Thimm Artificial Intelligence Group
Zweitgutachter:	Univ.-Prof. Dr. Christian Beecks Lehrgebiet Data Science
Betreuer:	Dr. David Winter Head of General Development bei Vitronic

Abgabedatum: 29.04.2024

Erklärung

Ich erkläre, dass ich die Masterarbeit selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder graphische Darstellungen. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Arbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.

	Ja	Nein
Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	✓	☐
Der Veröffentlichung dieser Arbeit auf der Webseite des Lehrgebiets Künstliche Intelligenz stimme ich zu.	✓	☐
Der Text dieser Arbeit ist unter einer Creative Commons Lizenz (CC BY-SA 4.0) verfügbar.	✓	☐
Der Quellcode ist unter einer GNU General Public License (GPLv3) verfügbar.	✓	☐
Die erhobenen Daten sind unter einer Creative Commons Lizenz (CC BY-SA 4.0) verfügbar.	✓	☐

Wiesbaden, 29.04.2024

(Ort, Datum) Unterschrift

Zusammenfassung

In dieser Arbeit wird die Eignung von Segmentierungsverfahren, die auf neuronalen Netzen basieren, hinsichtlich ihrer Fähigkeit zur Segmentierung von Schweißnähten, für die Detektion von Schweißnahtgrenzen, untersucht. Darüber hinaus werden Bildformate analysiert, die für diese Aufgabe am besten geeignet sind. Es wurden drei verschiedene Algorithmen ausgewählt: Mask R-CNN, U-Net und YOLOv8. Diese wurden aufgrund ihrer veröffentlichten Ergebnisse, frei zugängliche Implementierungen und des geeigneten Annotationsformats in deren Implementierung ausgewählt. Die Ansätze zeigten bei der Schweißnahtdetektion unterschiedliche Ergebnisse bezüglich ihrer Genauigkeit und Segmentierungszeit. Im Laufe der Untersuchungen zur Verbesserung der Leistung dieser Algorithmen wurden entweder die Hyperparameter modifiziert oder die Anzahl an Bilddaten erhöht. Das Modell mit den besten Ergebnissen unter den drei war YOLOv8. Schon beim ersten Test und ohne Anpassung der Hyperparameter und mit einer geringen Anzahl an Bildern konnten damit vielversprechende Ergebnisse erzielt werden. Wir erreichten Genauigkeitswerte von 94% IoU und eine Segmentierungszeit von 92 Millisekunden. U-Net hat vergleichbares IoU Wert jedoch die doppelte Segmentierungszeit. Das Mask-RCNN Modell war am schlechtesten. Es wurde festgehalten, dass YOLOv8 hinsichtlich Genauigkeit und Geschwindigkeit am besten abschneidet.

Abstract

In this thesis, the suitability of segmentation methods based on neural networks is examined in terms of their ability to segment weld seams for the detection of weld limits. Additionally, image formats best suited for this task are analyzed. Three different algorithms were selected: Mask R-CNN, U-Net, and YOLOv8. These were chosen based on their published results, freely available implementations, and the suitable annotation format in their implementation. The approaches showed varying results in weld detection regarding their accuracy and segmentation time. During the study to improve the performance of these algorithms, either the hyperparameters were modified or the number of image data was increased. The model with the best results among the three was YOLOv8. Even in the first test, without adjusting the hyperparameters and with a small number of images, promising results were achieved. We reached accuracy values of 94% IoU and a segmentation time of 92 milliseconds. U-Net achieved a comparable IoU value but with twice the segmentation time. The Mask R-CNN model performed the worst. It was noted that YOLOv8 was the best in terms of accuracy and speed.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

1.1 Problemstellung und Ziel der Arbeit

1.1.1 Motivation

Achsträger und Querlenker sind für die Sicherheit im Auto entscheidend. Ihre einwandfreie Qualität ist wichtig, um die Sicherheit der Insassen zu gewährleisten. Daher ist es wichtig, jeden Fehler in den Schweißnähten genau zu erkennen und sicherzustellen, dass nur fehlerfreie Teile weiterverarbeitet werden. Fehler in Schweißnähten können auch ernsthafte Probleme verursachen, inklusive Rückrufe und Imageverlust. Eine manuelle überprüfung dieser Nähte ist jedoch teuer und auch nicht immer zuverlässig, was die Notwendigkeit für effizientere Methoden unterstreicht.

Automatisierte Schweißtechniken sind heutzutage Standard in der Automobilindustrie, und eine Automatisierung der Schweißnahtprüfung verbessert den Prozess weiter. Sie ermöglicht eine konstante überwachung, unabhängig von äußeren Bedingungen, und dokumentiert darüber hinaus automatisch die Produktqualität.

Ein kritischer Aspekt der automatischen Prüfung ist die Detektion der Schweißnahtgrenzen. Sie wird von verschiedenen Faktoren beeinflusst. Dazu zählen die Oberflächenbeschaffenheit des Schweißmaterials, die Form der zu schweißenden Bleche und die Bedingungen, unter denen die Daten aufgenommen werden. Um die Auswirkungen dieser Faktoren zu minimieren und eine zuverlässige Nahtgrenzen-Detektion zu erreichen, ist momentan mit den eingesetzten Algorithmen eine ständige Anpassung der Parameter notwendig. Dieser Prozess ist aufwendig und erfordert ständige Unterstützung durch Service- oder Entwicklungsingenieure, was sowohl Zeit als auch Geld kostet. Eine Methode, die minimal oder gar nicht parametriert werden muss, könnte hierbei Ressourcen sparen. KI-basierte Segmentierungsverfahren können eine vielversprechende Lösung für dieses Problem bieten.

1.1.2 Verwandte Arbeiten und Beitrag der Arbeit

In der Masterarbeit [Sei18] meines Arbeitskollegen Stephan Seidel bei der Firma Vitronic, betitelt **Segmentierung von Schweißnähten mittels selbstlernender Algorithmen**, wurde die Machbarkeit der Anwendung künstlicher Intelligenz (KI) zur Segmentierung von Schweißnähten demonstriert. Dabei konnte eine beachtliche Genauigkeit von bis zu 94% erreicht werden, gemessen in *Intersection over Union* (IoU). Seidels Untersuchungen zeigen auf, dass die Größe der analysierten Bildausschnitte (*Patches*) sowie die Architektur des neuronalen Netzwerks entscheidend für die Effizienz des Segmentierungsprozesses sind. Zudem wurde herausgestellt, dass Modifikationen in der Struktur des Netzwerks zu Leistungsverbesserungen führen können, ohne dabei erhebliche Qualitätsverluste in Kauf nehmen zu müssen. Eine signifikante Einschränkung seines Ansatzes ist jedoch die Limitierung auf manuell markierte Bildausschnitte (siehe Abbildung 1), was den Einsatz auf die gesamte Schweißnaht erschwert. Das von Seidel eingesetzte Netzwerk benötigt mindestens

vier Durchläufe, um eine Naht vollständig zu segmentieren.

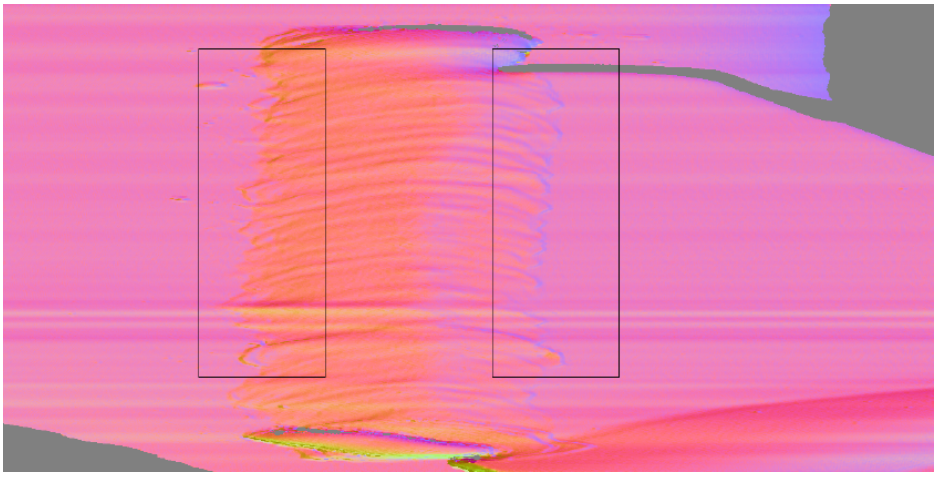


Abbildung 1: Der Bereich der für die Patches einer Naht ausgewält wurde

Die Bedeutung der aktuellen Arbeit ergibt sich aus dem Bestreben, diese Einschränkung zu überwinden. Ziel ist es, Algorithmen zu untersuchen und weiterzuentwickeln, die eine direkte Anwendung auf die gesamte Schweißnaht ermöglichen. Dieser Ansatz verspricht eine effizientere, automatisierte Segmentierung und eröffnet neue Möglichkeiten für den industriellen Einsatz, indem er den Arbeitsaufwand reduziert und die Präzision der Detektion der Schweißnahtgrenzen erhöht.

1.1.3 Aufgabenstellung

In dieser Masterarbeit, durchgeführt bei Vitronic Dr.-Ing. Stein Bildverarbeitungssysteme GmbH, wird die Effektivität von KI-basierten Segmentierungsverfahren für die Detektion von Schweißnahtgrenzen untersucht. Ziel ist es, die Tauglichkeit dieser Methoden speziell für industrielle Anwendungen zu evaluieren und Möglichkeiten zur Leistungssteigerung zu identifizieren, um deren Einsatzpotenzial in der Praxis zu optimieren.

Der erste Schritt meines Projekts bestand darin, vorhandene Verfahren zur Segmentierung mithilfe von KI-Algorithmen zu recherchieren. Da die Aufgabe das Testen dieser Verfahren einschließt, war es erforderlich, Algorithmen zu identifizieren, für die sowohl der Quellcode als auch vortrainierte Modelle veröffentlicht wurden. Angesichts der Vielzahl der Modelle war es zudem wichtig, dass diese Algorithmen ein gemeinsames Annotationswerkzeug nutzen, um den Aufwand für die Annotationen zu minimieren. Ebenso war es notwendig sicherzustellen, dass diese Algorithmen auf einer GPU lauffähig sind, um eine beschleunigte Durchführung des Trainings zu ermöglichen. Die gemäß diesen Kriterien ausgewählten Verfahren wurden anschließend auf ihre Eignung zur Segmentierung von Schweißnähten geprüft.

Um die Ergebnisse und die Leistung der ausgewählten Verfahren zu verbessern,

wurden die Hyperparameter modifiziert und die Anzahl der Trainingsdaten erhöht. Mit den so erzielten Ergebnissen wird das Verfahren abschließend mit Testdaten auf seine Eignung für den Einsatz in der Industrie untersucht.

1.2 Stand der Technik

1.2.1 3D basierte Nahtsuche

Das erste Verfahren zur Detektion von Schweißnahtgrenzen nutzt 3D-Daten, um die Grenzen der Naht im eingelernten Schweißbereich zu bestimmen. Für jeden Einzelscan werden obere und untere Grenzwerte berechnet. Durch Vergleich des berechneten Modells (Bauteil ohne Scheißnaht) mit dem aktuellen Scansignal wird die Differenz zeilenweise berechnet. Wenn die eingestellte Empfindlichkeit überschritten wird, wird die Nahtgrenze festgelegt (siehe Abbildung 2).

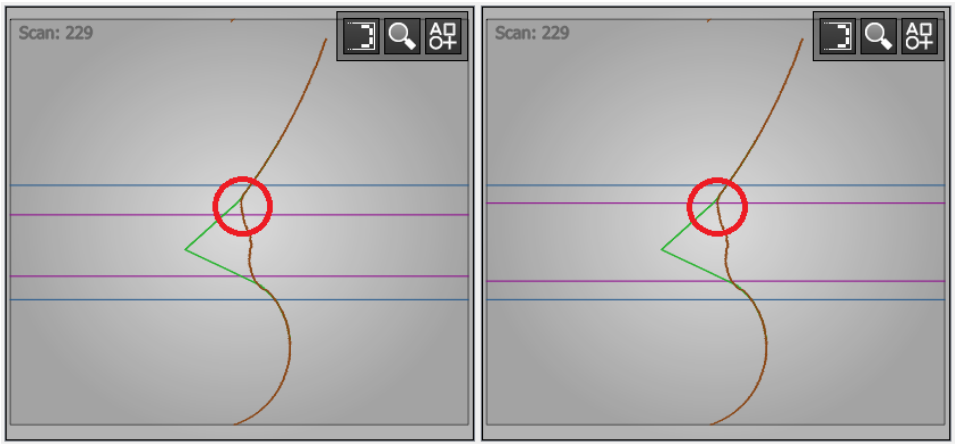


Abbildung 2: Zwei Einzelscans aus einer Kehlnaht: Links Hoher Empfindlichkeitswert, Rechts niedriger Empfindlichkeitswert

1.2.2 Kantenbasierte Nahtsuche

Um die Nahtgrenzen-Detektion auf flachen Nähten besser zu detektieren, stößt die Nahtsuche mit Differenzbildung zwischen Modell und Scan an ihre Grenze. Die flachen Nähte haben kaum überhöhung. Aus diesem Grund wurde ein zweites Verfahren entwickelt.

Das zweite Verfahren zur Detektion von Schweißnahtgrenzen ist die segmentbasierte Pfadsuche, die eine Kombination aus 2D und 3D-Daten verwendet. Wie man aus der Abbildung 3 entnehmen kann, arbeitet dieses Verfahren in mehreren Schritten: Zunächst werden auf verschiedenen Bildebenen scanweise Kantenkandidaten gesucht und aus diesen zusammenhängenden Segmenten erstellt. Dann werden passende Segmente kombiniert, um eine logische Nahtgrenze zu erstellen, die möglichst wenig Lücken und Sprünge quer zur Verfahr-Richtung aufweist.

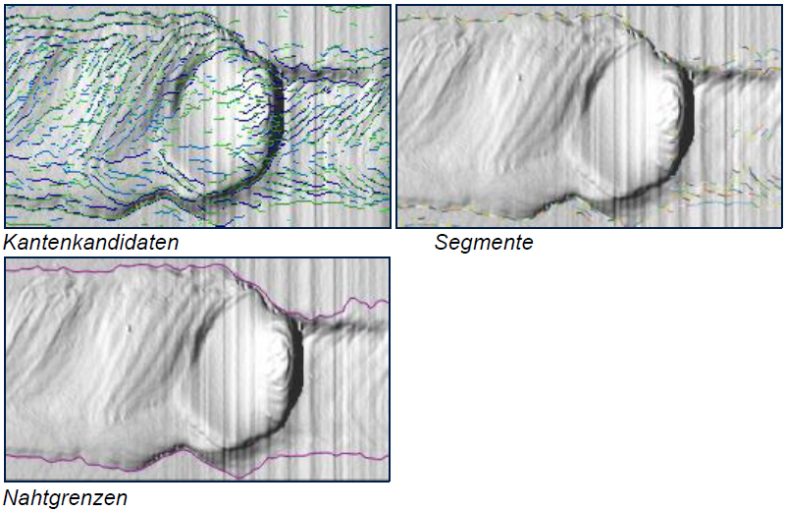


Abbildung 3: Schritte der Kantenbasierte Nahtsuche

Dieses Verfahren zeigt Schwächen bei langen Nähten und kann langsam werden. Es gibt auch Situationen, in denen die Nahtgrenze schwer zu steuern ist und unerwartete Sprünge auftreten (siehe Abbildung 4). Um dies zu vermeiden, müssen geeignete Parametrierungen getroffen werden.

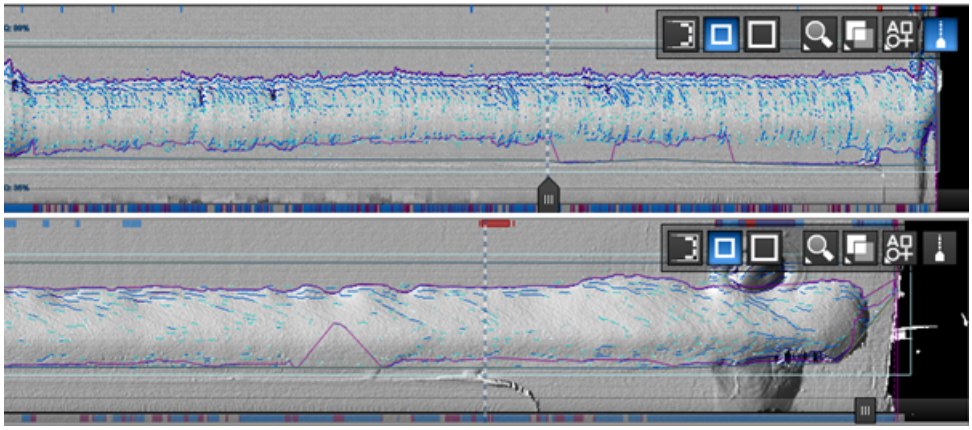


Abbildung 4: Unerwartete Sprünge von den Nahtgrenzen

1.3 Struktur der Arbeit

In diesem Abschnitt werden wir die Struktur dieser Arbeit erläutern, um den Lesern, auch solchen, die nicht mit den technischen Aspekten des Fachgebiets vertraut sind, einen klaren Leitfaden zu bieten.

Im Abschnitt Material 2 wird das verwendete Material und die technische Ausstattung beschrieben. Dazu gehören Layer-Bilder, normale Bilder und die verwendete Software (VIROwsi, Anaconda, LabelMe) sowie die genutzte Hardware, einschließlich der Grafikkarte und Prozessor.

Im dritten Abschnitt Theoretischer Hintergrund 3 wird der theoretische Hintergrund erläutert. Dies beinhaltet eine Einführung in neuronale Netzwerke, die Konzepte der Bildsegmentierung (semantisch und instanziell), die Bedeutung von *Ground Truth*, und eine übersicht über KI-Verfahren zur Bildsegmentierung, wie MASK RCNN, U-NET und YOLOV8. Zudem werden Bewertungsmetriken *Jaccard-Index* und *Average Precision* behandelt.

Im vierten Abschnitt 4 wird die Umsetzung dargestellt. Dies beinhaltet die Datensammlung und Bildannotation, Methoden zur Bildaugmentierung, das Training und die Validierung der Modelle sowie deren Optimierung.

Im fünften Abschnitt 5 werden die Ergebnisse detailliert untersucht, die durch verschiedene Tests und Experimente mit verschiedenen Bildtformate und Augmentierungsmethoden erzielt wurden. Hier werden die konkreten Resultate und deren Interpretation präsentiert.

In der Schlussbetrachtung 6 meines Projekts werden drei Hauptpunkte behandelt: Zunächst werden die bedeutendsten Ergebnisse des Projekts zusammengefasst. Danach erfolgt eine Bewertung des Projekterfolgs, und schließlich wird die Erreichung der gesetzten Ziele beurteilt. Zum Abschluss wird ein Ausblick auf mögliche weiterführende Arbeiten gegeben.

Abschließend folgt das Literaturverzeichnis, das eine vollständige Liste aller in meiner Arbeit zitierten Referenzen und Quellen enthält.

2 Material

In diesem Kapitel wird das Material vorgestellt, das für die Durchführung dieser Arbeit verwendet wurde. Es umfasst sowohl die Softwarekomponenten als auch die Hardwarekonfigurationen, die wichtig sind, um die in dieser Arbeit behandelten Fragestellungen zu bearbeiten. Ferner wird auf die verschiedenen Bildformate eingegangen, die in den Experimente genutzt wurden.

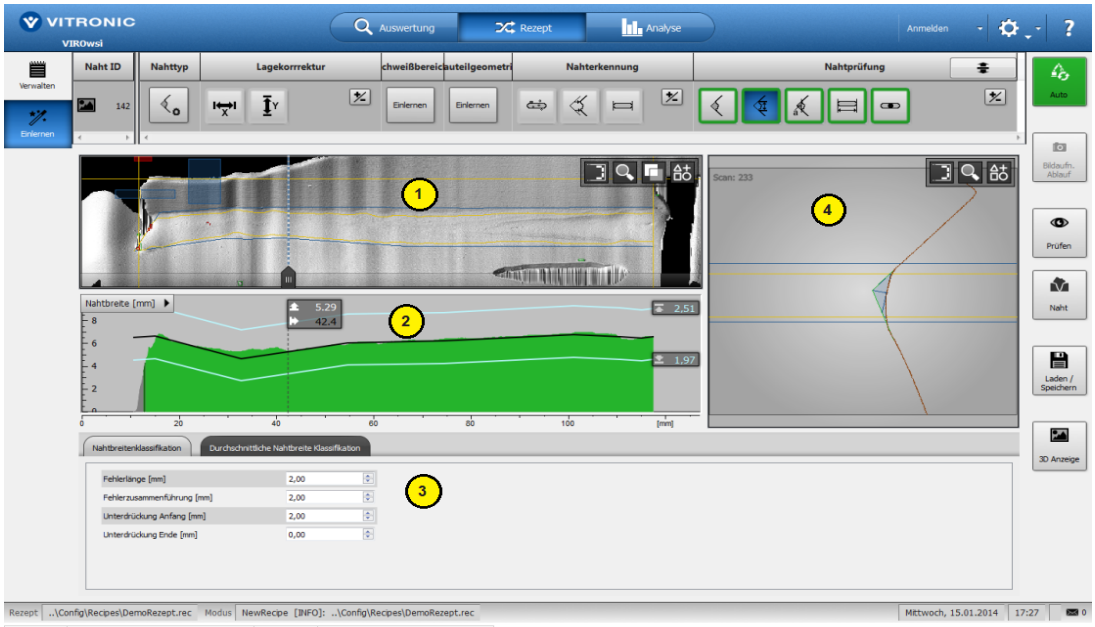
2.1 Verwendete Software

Die Auswahl der Software ist kritisch für die Durchführung wissenschaftlicher Experimente und Analysen. Im Folgenden werden die Hauptsoftwaretools beschrieben, die in dieser Arbeit verwendet wurden.

2.1.1 VIROwsi

Nach dem Abschluss dieser Arbeit wird ein Prototyp entwickelt und in die Software **VIROwsi** integriert. **VIROwsi**, ein Produkt der Firma Vitronic, ist ein Akronym für *Weld Seams Inspection* und bezeichnet ein fortschrittliches 3D-Schweißnaht-Inspektionssystem, das speziell für die automatische überprüfung von Schweißnähten auf Stahl- und Aluminiumoberflächen konzipiert ist. Das System setzt sich aus einem 3D-Laserscanner (siehe Abbildung 5), einem Hochleistungsrechner und der dazugehörigen Software (siehe Abbildung 6) zusammen.

Abbildung 5: 3D-Laserscanner



1	**Übersichtsbild**	3	**Parameter Einstellung**
2	**Auswertungsdiagramm**	4	**Einzelscan-Bild**

Abbildung 6: Eine Ansicht aus **VIROwsi**

Mithilfe des 3D-Laserscanners, der üblicherweise am *Tool Center Point* (TCP) eines Roboters montiert wird, werden 3D-Informationen über das Lichtschnittverfahren gewonnen. Indem man den Scanner entlang der Schweißnaht führt, extrahiert man aus der Aneinanderreihung einzelner Profile (siehe Abbildung 7) eine zusammenhängende 3D-Oberfläche (siehe Abbildung 8).

2.1.2 Python

Python, eine höhere Programmiersprache, die für ihre gute Lesbarkeit und Effizienz in der Entwicklerproduktivität bekannt ist, wurde von Guido van Rossum entwickelt und erstmals am 20. Februar1991 freigegeben. Mit einer minimalistischen Syntax und einer reichhaltigen Standardbibliothek unterstützt Python mehrere Programmierparadigmen wie objektorientierte, funktionale und imperative Programmierung. Zu den Kernfunktionen gehören dynamische Typisierung, automatische Speicherverwaltung und Unterstützung für Mehrfadenverarbeitung [Pyt24]. Python-Code wird in Funktionen, Klassen, Modulen und Paketen organisiert.

Python gilt als nahezu Standardprogrammiersprache im Bereich neuronaler Netzwerke, dank ihrer effizienten Bibliotheken wie TensorFlow, Keras und PyTorch, die den Entwicklern den Aufbau und das Training komplexer Modelle erleichtern [Pyk23]. Diese Vorteile haben Python in der KI-Forschung und -Entwicklung fest etabliert. Folglich wurde Python auch in diesem Projekt für die Arbeit mit neuronalen Netzwerken ausgewählt, um von seiner weitreichenden Unterstützung und

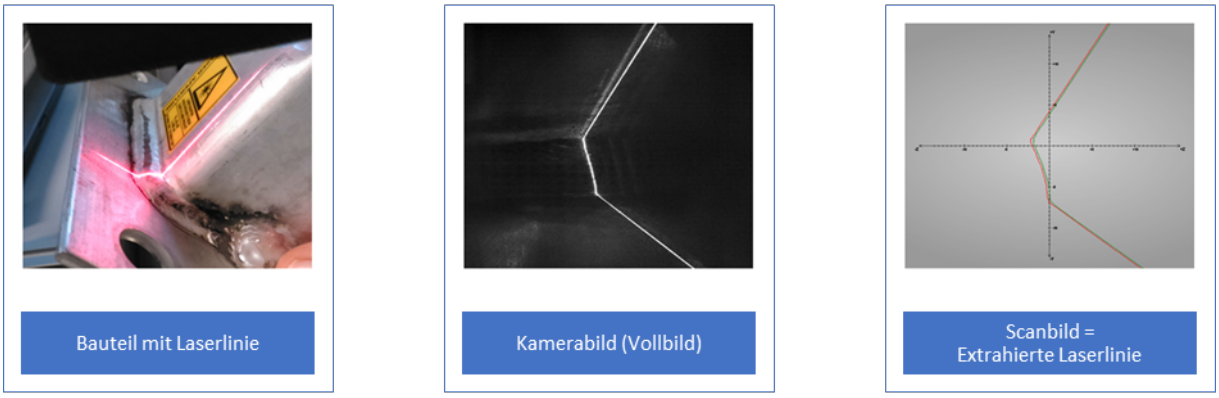


Abbildung 7: Vom Bauteil zum Profil

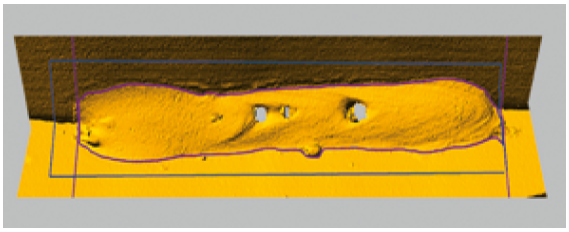


Abbildung 8: 3D-Oberfläche

Benutzerfreundlichkeit zu profitieren.

2.1.3 Anaconda

Anaconda bietet eine kostenlose und *Open-Source-Distribution*, spezialisiert auf Anwendungen im wissenschaftlichen Rechnen, einschließlich zentraler Bereiche wie *Data Science* und maschinelles Lernen. Es unterstützt Entwickler beim Erstellen benutzerdefinierter Umgebungen, was besonders nützlich ist, um *Convolutional Neural Networks* (CNNs) effizient zu implementieren. Diese Praxis ist wichtig, wenn man mit mehreren Implementierungen von neuronalen Netzwerken arbeiten möchte, die unterschiedliche Pakete und Bibliotheken erfordern. Dank Anaconda lässt sich die Erstellung neuer Umgebungen, die verschiedene Python- oder Tensorflow-Versionen enthalten, vereinfachen. Dies gewährleistet Kompatibilität und Flexibilität bei der Entwicklung von Projekten und ermöglicht eine gezielte Kontrolle über die Projektumgebung, was wiederum die Zuverlässigkeit und Reproduzierbarkeit der Forschungsergebnisse verbessert [Ana23].

Anaconda besteht hauptsächlich aus zwei Komponenten: dem *Anaconda Navigator* und der *Anaconda Command Prompt* (siehe Abbildung 9). *Anaconda Navigator* ist eine GUI, die die Paket- und Umgebungsverwaltung vereinfacht, indem sie Installation, Aktualisierung und Zugriff auf Tools wie *Jupyter Notebooks* ermöglicht. Sie

ist für Nutzer gedacht, die eine visuelle Oberfläche bevorzugen. *Anaconda Prompt* hingegen ist eine CLI für erfahrene Nutzer, die über Befehle Pakete verwalten, Umgebungen konfigurieren und Datenanalyseoperationen durchführen möchten, was eine schnelle und effiziente Arbeitsweise ermöglicht.

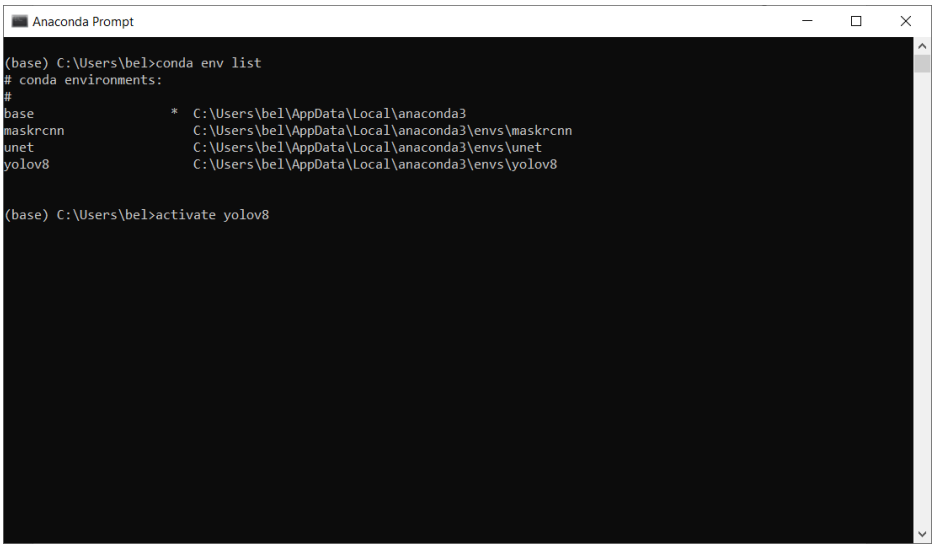


Abbildung 9: Anaconda Eingabeaufforderung

Die Open-Source-Datenpakete können einzeln aus dem *Anaconda Prompt* mit dem Befehl conda install oder mit dem Befehl pip install installiert werden, der zusammen mit Anaconda installiert wird. Pip-Pakete bieten viele der Funktionen von conda-Paketen und in den meisten Fällen können sie zusammenarbeiten.

2.1.4 LabelMe

In meiner Masterarbeit spielte die präzise Annotation von Schweißnahtgrenzen eine entscheidende Rolle für die Qualität der Forschungsergebnisse. Zu diesem Zweck wurde das Annotationstool LabelMe (siehe Abbildung 10) eingesetzt, ein weit verbreitetes, Werkzeug zur Erstellung von *Labels* für Bilder im Bereich des maschinellen Lernens.

LabelMe bietet eine umfangreiche Palette an Funktionen für die Bild- und Videoannotation sowie für die Anpassung der Benutzeroberfläche und die Datenexportierung, die es zu einem vielseitigen Werkzeug für Projekte in den Bereichen maschinelles Lernen und Computer Vision machen. Hier eine Zusammenfassung der Hauptfunktionen [Wad16]:

1. **Bildannotation**: Unterstützt das Annotieren von Bildern mit Polygonen, Rechtecken, Kreisen, Linien und Punkten.

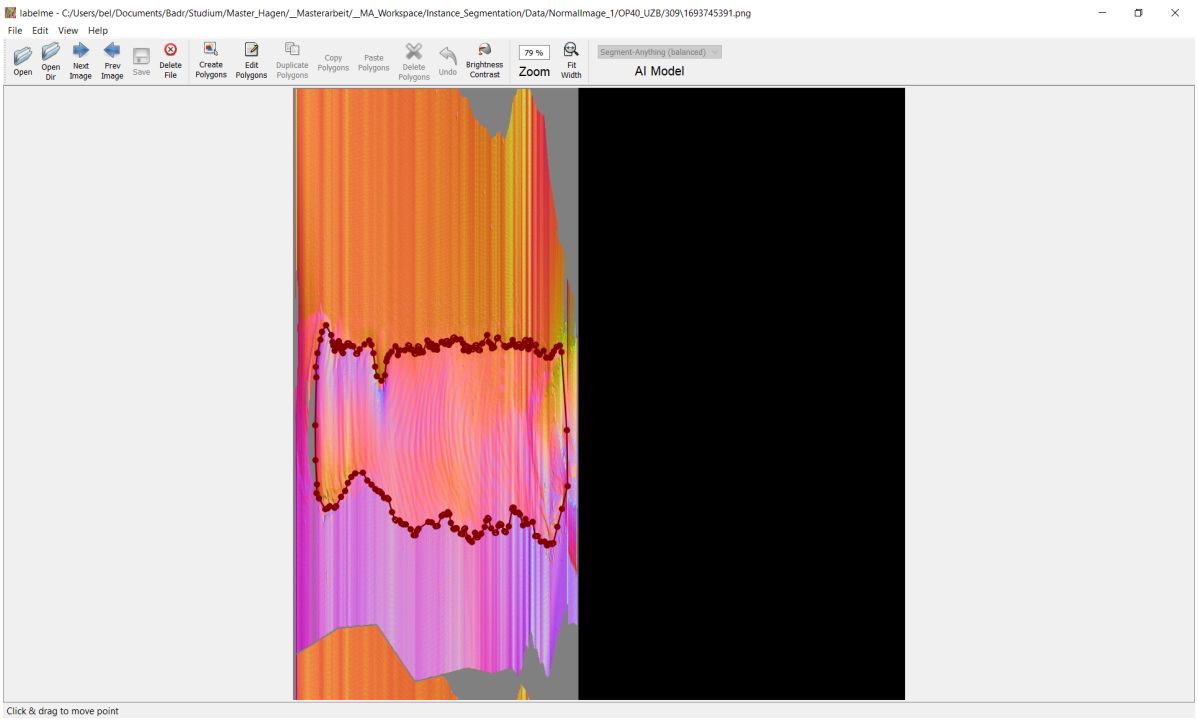


Abbildung 10: Manueller Annotationsprozess von einer Schweißnaht mittels Label-Me [Wad16]

2. **Bildflaggen-Annotation**: Ermöglicht die Klassifizierung und Bereinigung von Bildern.

3. **Videoannotation**: Erweitert die Annotierungsmöglichkeiten auf Videos, sodass Benutzer Objekte in Videoinhalten markieren können.

4. **Export im VOC-Format**: Ermöglicht das Exportieren von annotierten Datensätzen im VOC-Format für semantische und Instanzsegmentierung, was die Kompatibilität mit verschiedenen Bildverarbeitungstools sicherstellt.

5. **Export im COCO-Format**: Unterstützt das Exportieren von Datensätzen im COCO-Format speziell für die Instanzsegmentierung, was eine breite Anwendung in maschinellem Lernen und Computer Vision ermöglicht.

2.2 Hardware- und Softwarekonfiguration

Die Hardware- und Softwarekonfiguration, auf der die oben genannten Softwaretools und Methoden betrieben werden, wird hir beschrieben.

Für diese Masterarbeit wurde eine spezielle Hardwarekonfiguration genutzt, die auf den Einsatz von neuronalen Netzwerken optimiert ist. Im Mittelpunkt steht die Grafikkarte NVIDIA RTX 2070 [NVF17], die dank ihrer leistungsstarken Architektur und GPU-Unterstützung eine ideale Basis für rechenintensive Anwendungen im KI-Bereich bietet. Diese Grafikkarte wird in Kombination mit den Bibliotheken TensorFlow, PyTorch und Keras eingesetzt, die GPU-Support bieten und somit die Berechnungsgeschwindigkeit und Leistungsfähigkeit von neuronalen Netzwerke erheblich steigern. TensorFlow, PyTorch und Keras haben jeweils ihre eigenen Stärken und Einsatzbereiche. TensorFlow ist bekannt für seine umfangreiche Bibliothek und Skalierbarkeit, ideal für Produktionsumgebungen. PyTorch bietet eine dynamische Berechnungsgraph-Struktur, die Experimente erleichtert und besonders bei Forschern beliebt ist. Keras, als *High-Level-API*, zeichnet sich durch seine Einfachheit und schnelle Prototypenentwicklung aus [Pyk23]. Die Kombination dieser leistungsfähigen Software-Bibliotheken mit der NVIDIA RTX 2070 GPU ermöglicht es, die Grenzen der Leistungsfähigkeit und Effizienz bei der Entwicklung und Implementierung von KI-Modellen zu erweitern. Zusätzlich wurden alle Tests der Modelle mit den Testdaten auf einer CPU 12th Gen Intel(R) Core(TM) i7-12800H, 1800 MHz, 14 Kern(e) durchgeführt.

2.3 Bildformate

Die Detektion von Schweißnahtgrenzen stellt eine herausfordernde Aufgabe dar, die durch den Einsatz spezialisierter Bildformate und Messverfahren unterstützt wird. Dieses Kapitel bietet einen umfassenden überblick über die grundlegenden Techniken und Formate, die in dieser Arbeit eine zentrale Rolle spielen.

Zu Beginn wird das Lichtschnittverfahren vorgestellt, ein fundamentales Messprinzip zur dreidimensionalen Profilvermessung. Dieses Verfahren nutzt die Lasertriangulation, um durch die Analyse einer beleuchteten Lichtlinie auf einem Objekt dessen 3D-Profil zu ermitteln.

Anschließend wird eine Einführung in die Welt der Pseudo-3D-Daten gegeben. Trotz ihres Namens liefern Pseudo-3D-Daten wertvolle Informationen über die dreidimensionale Struktur von Objekten, indem sie aus der 2D-Lasertriangulation gewonnene Daten in einem speziellen Format speichern.

Nachfolgend werden die sogenannten Layer-Bilder behandelt, welche die Basis für die Erstellung eines RGB-Bildes bilden. Die Layer *ShiftedMiddle*, *ShiftedWidth* und *ShiftedGrayValue* werden dabei verwendet, um die verschiedenen Farbkanäle des RGB-Bildes zu erstellen.

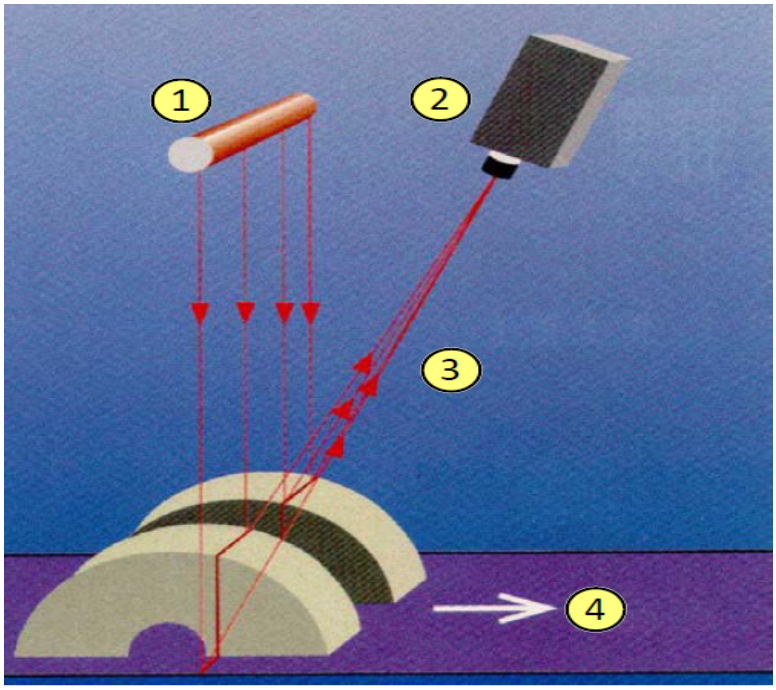
Zuletzt werden die Normalen-Bilder behandelt, welche eine Erweiterung der Bildformate darstellen und vor allem für die Visualisierung und Analyse von Oberflächenstrukturen genutzt werden. Diese Bilder basieren auf den Koordinaten des Normalenvektors und nutzen eine Farbkodierung, um ein RGB-Bild zu erstellen. Sie bieten Einblicke in die lokale Ausrichtung der Oberflächenstrukturen und unterstützen somit die Interpretation und Bewertung von Schweißnähten und anderen Oberflächenmerkmalen.

2.3.1 Messprinzip

Das Lichtschnittverfahren zur dreidimensionalen Profilvermessung stellt das grundlegende Messprinzip für die Aufnahme der 3D-Daten dar. Bei diesem Verfahren wird eine Laserbeleuchtung mit vorgesetzter Zylinderlinse, auch bekannt als Linienlaser, als Lichtquelle verwendet. Eine Videokamera wird in einem definierten Winkel zur Lichtquelle positioniert, sodass im Kamerabildfeld die beleuchtete Lichtlinie sichtbar ist (siehe Abbildung 11). Die Position und Form dieser Lichtlinie im Kamerabildfeld variiert entsprechend der Höhen des zu vermessenden Objekts [Tea23].

Der Triangulationswinkel (Winkel zwischen Kamera und Lichtquelle) bestimmt die Verschiebung und Krümmung der Lichtlinie in Höhenrichtung. Ein größerer Winkel führt zu einer stärkeren Verstärkung dieser Effekte. Durch die Kenntnis des Triangulationswinkels und der Position der Lichtlinie können die 3D-Objektpunkte ermittelt und der Profilschnitt des Objekts bestimmt werden. Während der Sensor über das Objekt bewegt wird, kann durch die Bewegungsgeschwindigkeit oder die aktuelle Scanposition ein 3D-Abbild des gescannten Objekts erzeugt werden [Tea23].

Für einen einzelnen Sensor ist es jedoch nur möglich, die dem Sensor zugewandte Objektfläche zu vermessen. Die Messdaten enthalten die X-, Y- und Z-Position der ermittelten Messpunkte eines Objekts. Der minimale Abstand zwischen zwei aufeinanderfolgenden Scans wird durch die Verfahrgeschwindigkeit des Sensors bestimmt [Tea23].



1	Linienlaser	3	Projektionsebene
2	Kamera	4	Vorschub

Abbildung 11: Prinzip des Lichtschnittverfahrens [Tea23]

Es ist wichtig zu beachten, dass das Lichtschnittverfahren optische Einschränkungen aufweist. Das Scannen von Hinterschneidungen oder anderen optisch abgedeckten Bereichen ist nicht möglich. Nicht durchlässige Materialien im Scanbereich werden in das Scanergebnis einbezogen. Darüber hinaus können dunkle oder spiegelnde Stellen sowie fleckige Oberflächen mit starken Hell-Dunkel-Kontrasten zu Lücken oder *Blooming*-Bereichen im Scansignal führen, was das Endergebnis beeinflussen kann [Tea23].

2.3.2 Pseudo-3D Daten

Pseudo-3D-Daten sind ein bedeutendes Bildformat, das in der Industrie zur Erfassung und Analyse von dreidimensionalen Strukturen verwendet wird. Im Kontext dieser Arbeit beziehen sich Pseudo-3D-Daten auf einen speziellen Datencontainer namens *ViPicturePseudo3D*. Dieser Container speichert Daten, die aus der 2D-Laser-Triangulation gewonnen werden. Die Laser-Triangulation erzeugt anfangs 2D-Daten (Y, Z), doch durch wiederholte Aufnahmen und Bewegungen des Sensors relativ zum Objekt wird die dritte Dimension erzeugt (Siehe. Absch. 2.3.1). Die aufgenommenen Daten stellen eine Art Abwicklung des Objekts dar, was bedeutet, dass keine vollständige dreidimensionale Darstellung des Objekts vorliegt, sondern lediglich eine Abwicklung davon. Aus diesem Grund wurde der Begriff *Pseudo-3D* gewählt.

Abbildung 12 zeigt eine Ansicht der Rohdaten einer Naht mit der ID 142, während Abbildung 13 eine 3D-Ansicht derselben Naht darstellt.

Zusätzlich zu den genannten Informationen ist es wichtig zu erwähnen, dass die Breite (y-Richtung) der Pseudo-3D-Bilder konstant ist und beträgt 1248 Pixeln, während ihre Länge (x-Richtung) je nach zu kontrollierender Schweißnaht variiert. Die Abtastrate in der x-Richtung variiert ebenfalls abhängig von der Schweißnaht. Es sei darauf hingewiesen, dass Pseudo-3D-Daten ausschließlich mit einem speziellen 3D-Viewer von Vitronic visualisiert werden können; externe Viewer zeigen nur eine beschränkte Ansicht an.

2.3.3 Layer Bilder

Die Schlüsselkomponenten dieser Datenerfassung sind unter den Layern eLT_Middle, eLT_Width und $eLT\_GrayValue$ bekannt. eLT_Middle, dokumentiert in Abbildung 14 links, repräsentiert den 16-Bit-Wert des Mittelpunkts der Laserlinie, ermöglicht durch eine präzise COG (*Center of Gravity*)-Berechnung, die eine exakte Lokalisierung der Linie sicherstellt. Dieser Mittelpunkt wird aus den mit einer 2D-Kamera aufgenommenen Laserstrahlen extrahiert, wobei der Laserstrahl im Idealfall als Gauß-Kurve abgebildet wird. eLT_Width, dokumentiert in Abbildung 12 rechts, ist ebenfalls ein 16-Bit-Wert und beschreibt die Breite der Laserlinie, was essenzielle Informationen über die Ausdehnung und Form des Laserstrahls liefert. $eLT\_GrayValue$, illustriert in Abbildung 13 links, ist ein 8-Bit-Wert, der den Grauwert an der COG-Position angibt und für die Analyse der Intensität sowie Qualität

Abbildung 12: Die Pseudo-3D-Rohdaten, Naht-ID 142, in der Mitte horizontal die Schweißnaht

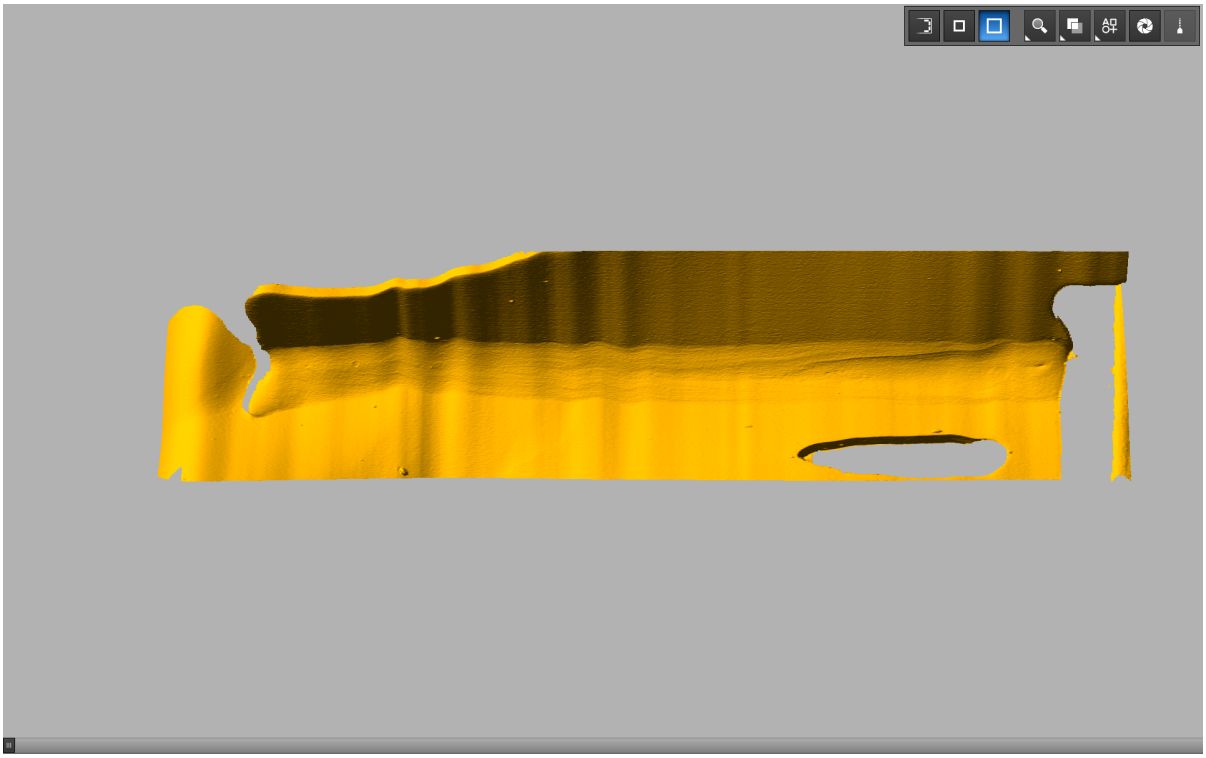


Abbildung 13: 3D-Ansicht der Naht-ID 142

des Laserstrahls von Bedeutung ist. Diese Daten sind kalibriert und mit Skalierungsfaktoren versehen, die eine Umrechnung der Pixelkoordinaten in Weltkoordinaten ermöglichen, was eine genaue Interpretation und Analyse der Pseudo-3D-Bilder unterstützt.

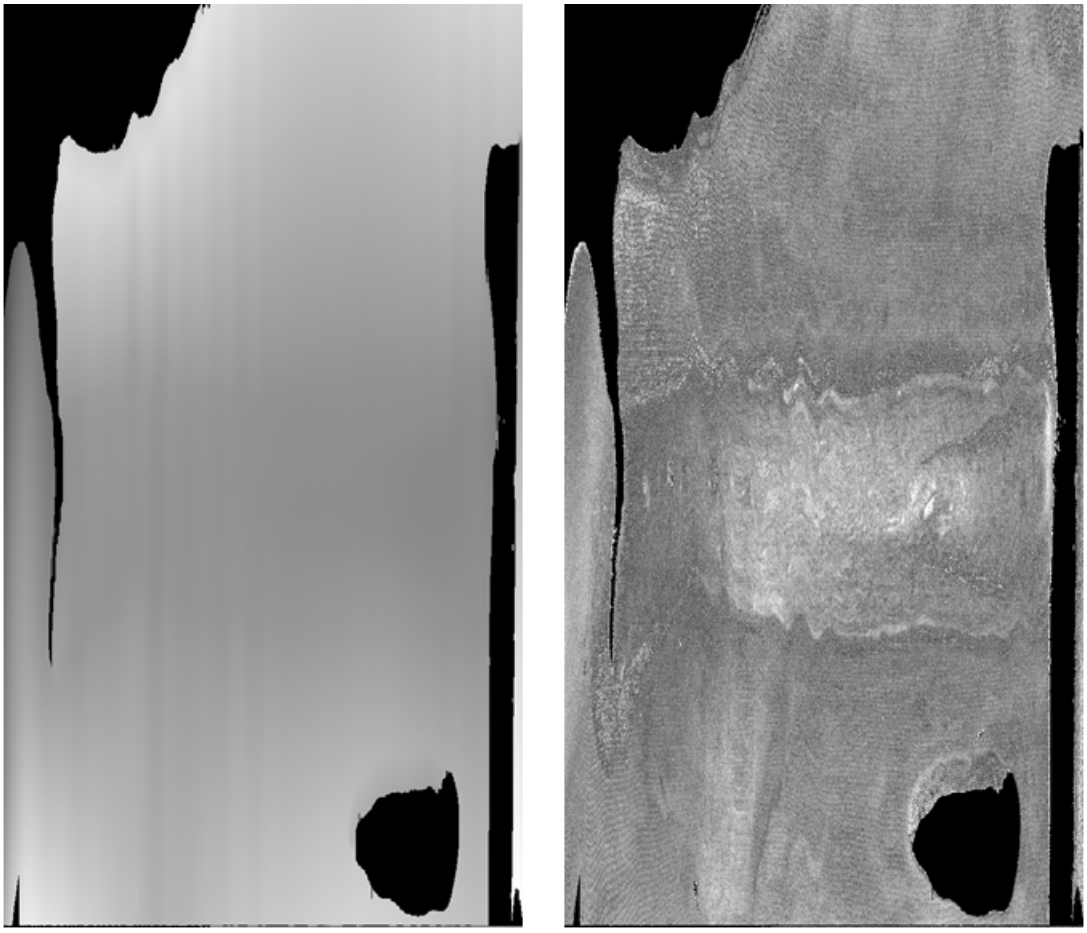


Abbildung 14: links: *middle layer*, rechts: *width layer*

In dieser Arbeit werden diese drei Layer – der Mittelpunkt, die Breite und der Grauwert des Laserstrahls – in einem fortgeschrittenen Verarbeitungsschritt zusammengeführt, um ein umfassendes RGB-Bild zu generieren, wie in Abbildung 15 rechts illustriert. Dieser Prozess ermöglicht eine visuell intuitive Darstellung der gesammelten Daten und bietet eine reichhaltige Informationsgrundlage für die weiterführende Analyse und Interpretation der Pseudo-3D-Bilder.

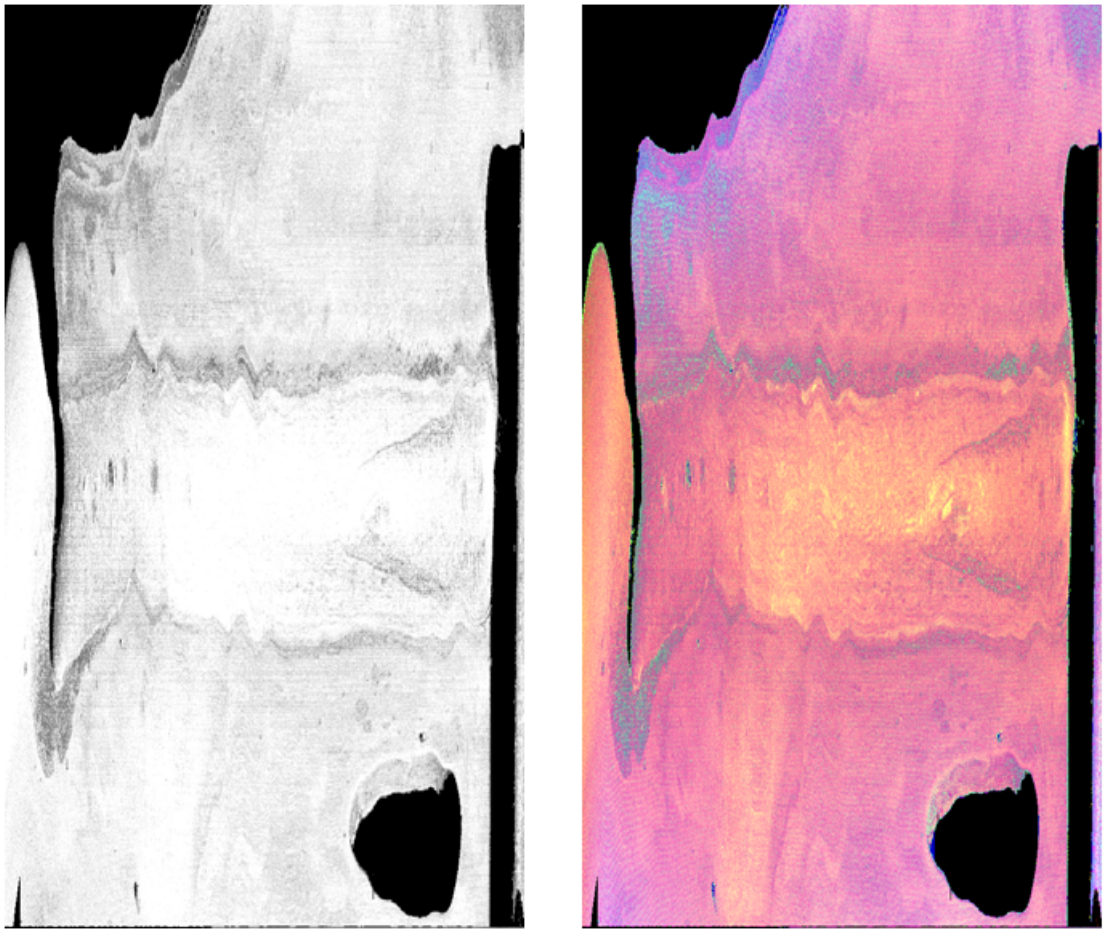


Abbildung 15: links: Grauwert Layer, Rechts: Layer-Image

2.3.4 Normalen-Bilder

Normalenbilder (siehe Abbildung 1) stellen in der **VIROwsi** eine wesentliche Erweiterung der Bildformate dar, die speziell für die Untersuchung von Oberflächenstrukturen, wie Löchern und Perlen, genutzt werden. Sie werden direkt aus den Pseudo-3D-Daten erzeugt, was eine detaillierte und anschauliche Analyse der Oberflächenstrukturen ermöglicht. Im Unterschied zu Pseudo-3D-Daten und Layer-Bildern, die vornehmlich Informationen über die Höhe und Breite einer Oberfläche bereitstellen, bieten Normalenbilder durch die Darstellung der ortsabhängigen Normalenvektoren tiefergehende Einblicke in die lokale Ausrichtung der Oberflächenstrukturen. Diese Normalen werden üblicherweise farbig kodiert, um die verschiedenen Ausrichtungen visuell unterscheidbar zu machen und somit die Visualisierung zu verbessern.

Besonders wertvoll erweist sich der Einsatz von Normalenbildern bei der Analyse von Schweißnähten und ähnlichen Oberflächenstrukturen. Sie visualisieren die Richtung der Oberflächennormalen und ermöglichen dadurch eine präzisere Interpretation der Struktur und Beschaffenheit der Schweißnähte. Diese detaillierte Darstellung ist entscheidend für die Erkennung und Bewertung von Defekten, Unregelmäßigkeiten oder Qualitätsmängeln.

3 Theoretischer Hintergrund

In diesem Projekt werden neuronale Netzwerke eine wichtige Rolle bei der Bewältigung von Herausforderungen im Bereich der Segmentierung von Schweißnähten spielen. Die Implementierung und Optimierung neuronaler Netzwerke erfordert nicht nur ein grundlegendes Verständnis der zugrundeliegenden Algorithmen und Architekturen, sondern auch eine tiefe Kenntnis der spezifischen Anforderungen der Bildsegmentierung selbst. Das Ziel dieses Kapitels ist es, einen umfassenden Überblick über den theoretischen Hintergrund und die praktische Anwendung neuronaler Netzwerke in der Bildsegmentierung zu geben, um eine fundierte Basis für die Diskussion der in diesem Projekt verwendeten Algorithmen und der erzielten Ergebnisse zu legen. Zunächst werden grundlegende Techniken und Typen der Segmentierung vorgestellt. Anschließend werden die Grundlagen neuronaler Netze erläutert, beginnend mit den Grundlagen eines einzelnen Neurons bis hin zu spezifischen Netzwerktypen wie den *Convolutional Neural Network*s, die vor allem in der Bildverarbeitung Anwendung finden. Das Kapitel befasst sich außerdem mit der Erläuterung von Maskenbildern, sowie mit Bildaugmentationstechniken. Zum Schluss werden spezielle KI-Verfahren für die Bildsegmentierung (Mask RCNN, U-NET und YOLOv8) sowie Bewertungsmetriken für die Beurteilung der Segmentierungsleistung erläutert.

3.1 Segmentierung

Die Bildsegmentierung bezieht sich auf eine Technik, bei der das Bild in verschiedene Objekte aufgeteilt werden. Diese Objekte sind zusammenhängende Regionen in Form von Pixeln [YP22].

Die Technik der Bildsegmentierung vereinfacht die Identifizierung unterschiedlicher Bildbereiche durch die Zuweisung von *Labels* zu jedem Pixel. Diese Abgrenzung der einzelnen Bildsegmente basiert auf den Schlüsseleigenschaften Farbe, Intensität und Textur [YP22]. Einfach ausgedrückt, zielt die Segmentierung darauf ab, ein digitales Bild in Gruppen von Pixeln aufzuteilen, die ähnliche visuelle Merkmale aufweisen. Dies ermöglicht es, die Konturen der abgebildeten Objekte zu bestimmen [SKSA10]. Nachstehend werden verschiedene Segmentierungstechniken und -Typen kurz erläutert.

3.1.1 Segmentierungstechniken

Es gibt fünf gängige Techniken der Bildsegmentierung:

Threshold-based segmentation 16

Die Idee hinter dieser Methode ist einfach: Die Pixel, die zu den Objekten gehören, die wir im Bild erkennen wollen (also im Vordergrund), unterscheiden sich

stark von denen im Hintergrund. Deshalb hilft es, mit Schwellenwerten zu arbeiten, um diese beiden Bereiche voneinander zu trennen. In [aSNS10] sind bestimmte Vorverarbeitungs- und Nachverarbeitungstechniken, die für die Schwellwertsegmentierung erforderlich sind beschrieben. Die Anwendung von Schwellenwerten [aSNS10] wird folgendermaßen beschrieben:

Abbildung 16: Ein Beispiel für *Threshold-Based* Segmentierung [Wik14]

$$T = T[x, y, p(x, y), f(x, y)]$$

Hierbei steht T für den Schwellenwert. Die Position des für den Schwellenwert relevanten Punktes ist durch die Koordinaten x,y gegeben. Die Funktionen $p(x, y)$ und $f(x, y)$ stehen für die Werte der Graustufenpixel. Ein Bild, das nach dem Schwellenwertverfahren bearbeitet wurde, also das Schwellenwertbild $g(x, y)$, lässt sich wie folgt bestimmen:

$$g(x,y) = \begin{cases} 1 & \text{if } f(x,y) > T \\ 0 & \text{if } f(x,y) \leq T \end{cases}$$

Edge-based image segmentation

Bei der kantenbasierten Segmentierung (siehe Abbildung 17) wird die Position einer Kante entweder durch das Vorhandensein eines Extremums in der ersten Ableitung oder durch eine Nullstelle in der zweiten Ableitung bestimmt. Die kantenbasierte Segmentierung ermöglicht eine präzisere Abgrenzung von Objekten, da sie auf der Erfassung sprunghafte Helligkeitsänderungen im Bild basiert. Durch die Identifizierung und Verbindung dieser Kanten können klare Umrisse der Objekte definiert werden und die gewünschten Regionen im Bild herausgearbeitet werden [SKSA10]. Somit stellt die kantenbasierte Segmentierung einen effektiven Ansatz zur Vermeidung von Größenverfälschungen bei der Segmentierung von Objekten dar, ohne auf komplexe Schwellenwertschemata zurückzugreifen [SKSA10].

Abbildung 17: Lena und Canny Edge basiertes Bild [ADAA16]

Region-based image segmentation

Die Segmentierung von Bildern in Regionen basiert darauf, ähnliche Bildteile in unterschiedliche Bereiche zu unterteilen, wodurch diese Bereiche direkt identifiziert werden können. Dabei werden benachbarte Bildpixel aufgrund bestimmter Bildcharakteristika zu Regionen zusammengefasst [SKSA10]. In diesem Teil werden verschiedene Ansätze der regionenbasierten Segmentierung beleuchtet. Die Methoden der regionenbasierten Segmentierung lassen sich hauptsächlich in zwei Gruppen einteilen: das Wachstum von Regionen (*region growing segmentation*) und die Algorithmen für das Teilen und Verschmelzen von Regionen (*region splitting-merging algorithm*) [SKSA10].

Der Algorithmus für Regionenwachstum startet mit vordefinierten *seed points* und erweitert diese Regionen durch das Hinzufügen benachbarter Pixel, die bestimmte Ähnlichkeitskriterien erfüllen. Die Auswahl der *seed points* und die Kriterien für die ähnlichkeit sind entscheidend für die Effektivität dieses Ansatzes. Im Gegensatz dazu basiert der Regionen-Splitting-Merging-Algorithmus auf einem iterativen Verfahren, das von oben nach unten arbeitet: Es beginnt mit dem gesamten Bild als einer Einheit, teilt dieses in kleinere Quadrate und verschmilzt anschließend homogene Bereiche wieder miteinander. Dieser Prozess wird so lange wiederholt, bis keine weiteren ähnlichen Bereiche mehr zusammengeführt werden können [SKSA10].

Clustering-based image segmentation

In der Bildsegmentierung durch *Clustering* werden Objekte oder Muster so klassifiziert, dass die Elemente innerhalb einer Gruppe einander ähnlicher sind als jene aus verschiedenen Gruppen. Diverse *Clustering*-Methoden, wie das harte und das unscharfe *Clustering*, wurden hierbei eingesetzt, wobei jede Methode ihre spezifi-

schen Charakteristika aufweist [MSMB13]. Bildsegmentierung, die auf *Clustering*-Methoden beruht, umfasst Techniken wie das k-Means-Clustering, das *Mountain-Clustering* und das Subtraktive *Clustering*.

Ein verbessertes Verfahren des k-Means-*Clustering* adressiert die Begrenzungen des ursprünglichen Ansatzes, indem es Anfangszentroide auf systematische Weise festlegt. Durch die Auswahl von Startpunkten, die zueinander die geringste Distanz aufweisen, verbessert diese Methode signifikant die Effizienz des k-Means-*Clustering* [PR13].

Der Subtraktive-*Clustering*-Algorithmus, eine Weiterentwicklung des *Mountain-Clustering*, nutzt die Dichte der umliegenden Datenpunkte zur Clusterbildung. In Kombination mit dem k-Means-Clustering-Algorithmus ermöglicht dieses Verfahren die Segmentierung von Bildern in eine bestimmte Anzahl von Clustern. Ein nachfolgend angewendeter Medianfilter hilft dabei, das Bildrauschen zu reduzieren [NKY15].

Ein weiterer Ansatz, der Human Mental Search (HMS)-Algorithmus, basiert auf einem populationsbasierten Mechanismus und verwendet das k-Means-*Clustering*, um effektiv Cluster zu bilden. Dieser Ansatz zeichnet sich durch eine hohe Robustheit und eine schnellere Konvergenz im Vergleich zu konventionellen *Clustering*-Methoden aus [SHG19].

Artificial neural network-based segmentation

In einem künstlichen neuronalen Netzwerk entspricht jedes Neuron einem Pixel eines Bildes. Das Bild wird auf das neuronale Netzwerk abgebildet. Das Bild in Form eines neuronalen Netzwerks wird mithilfe von Trainingsdaten trainiert, und dann werden Verbindungen zwischen Neuronen, d.h. Pixeln, gefunden. Anschließend werden neue Bilder aus dem trainierten Bild segmentiert [YP22]. Eine detailliertere Erläuterung dieses Ansatzes und seiner Anwendungen wird in Abschnitt 3.5 vorgestellt.

3.1.2 Segmentierungsytypen

Die Bildsegmentierungstechniken lassen sich grob in zwei Hauptkategorien einteilen: Instanzsegmentierung und semantische Segmentierung, abhängig von den spezifischen Anforderungen der Trennungsaufgabe (siehe Abbildung 18).

Bei der semantischen Segmentierung werden Objekte derselben Kategorie im Segmentierungsergebnis mit derselben Farbe gekennzeichnet. Verschiedene Farben weisen unterschiedliche Kategorien auf, sodass unterschiedliche Kategorien im Bild unterschieden werden können. [ASdS22].

Bei der Instanzsegmentierung handelt es sich um eine Kombination aus semantischer Segmentierung und Objekterkennung. Die einzelnen Objekte werden anhand eines Bildes identifiziert. Man kann nicht nur die Kategorien festlegen, sondern jedes einzelne Objekt separat markieren [ASdS22].

Abbildung 18: Instanz- und semantische Segmentierung von Berkeley-Bild #323016 [ASdS22]

3.2 Neuronale Netze

Dieses Kapitel zielt darauf ab, die grundlegenden Bausteine und Strukturen neuronaler Netze zu erläutern, ihre Entwicklung zu skizzieren und spezifische Netzwerkarchitekturen, die in dieser Arbeit verwendet werden, zu beschreiben.

3.2.1 Neuron

Ein Neuron dient in einem künstlichen neuronalen Netz als grundlegende Recheneinheit für die Verarbeitung von Daten. Es akzeptiert die Eingabenwerte, macht Berechnungen und liefert einen Ausgangswert. Die wesentliche Idee ist, dass das Neuron während des Trainings lernt, Eingaben auf der Grundlage der im Verlauf des Trainings vorgenommenen Anpassungen sinnvoll zu interpretieren und zu verarbeiten.

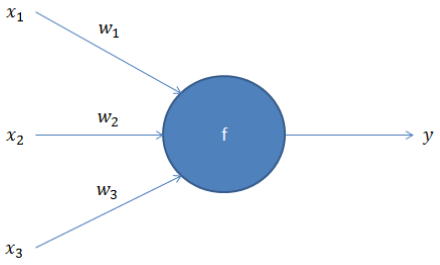


Abbildung 19: Ein einfaches Neuron [Fro19]

Die Operation eines Neurons wie es in Abbildung 19 dargestellt ist, lässt sich mathematisch wie folgt zusammenfassen:

$$y = \sum_{i=1}^{n} w_i x_i + b$$

Zuerst berechnet das Neuron die gewichtete Summe seiner Eingänge. Hierbei ist x_i ist der i-te Eingangswert, w_i das entsprechende Gewicht, und b der Bias des Neurons. n ist die Anzahl der Eingänge.

Anschließend wendet das Neuron eine Aktivierungsfunktion f auf diese Summe an, um den Ausgang y zu bestimmen:

$$a = f(x)$$

Die Aktivierungsfunktion ist üblicherweise eine nicht-lineare Funktion. Die am häufigsten verwendete Aktivierungsfunktion ist die Rektifizierte Lineare Einheit (ReLU: *Rectified Linear Unit*) (siehe Abbildung 20), die mathematisch wie folgt ausgedrückt werden kann:

$$f(x) = max(0, x)$$

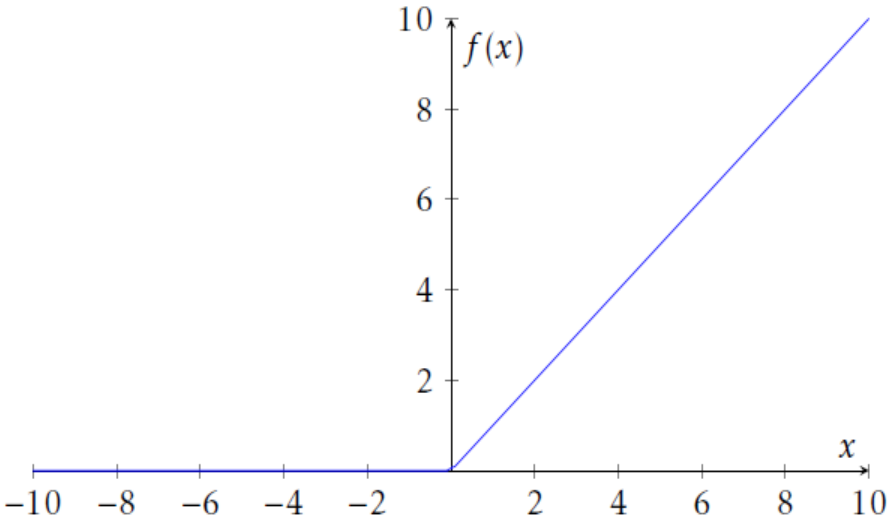


Abbildung 20: ReLU-Funktion [Sei18]

Diese mathematische Beschreibung bildet die Grundlage dafür, wie künstliche Neuronen in einem neuronalen Netzwerk Informationen verarbeiten und lernen, komplexe Aufgaben durch Anpassung ihrer Gewichte und Biases zu lösen.

3.2.2 Künstliche neuronale Netze

Künstliche neuronale Netze (KNN) sind Rechenmodelle, die aus Neuronen bestehen, die in Schichten angeordnet sind. Ein einfaches neuronales Netz ist in der Abbildung 21 dargestellt. Wie daraus erkennbar ist, steht jedes Neuron einer vorangegangenen Schicht in Verbindung mit sämtlichen Neuronen der folgenden Schicht.

Diese Konfiguration stellt die einfachste Form dar, die ein neuronales Netzwerk annehmen kann, und wird als vollvernetztes neuronales Netzwerk (*Fully Connected Layer*) bezeichnet. Es besteht aus:

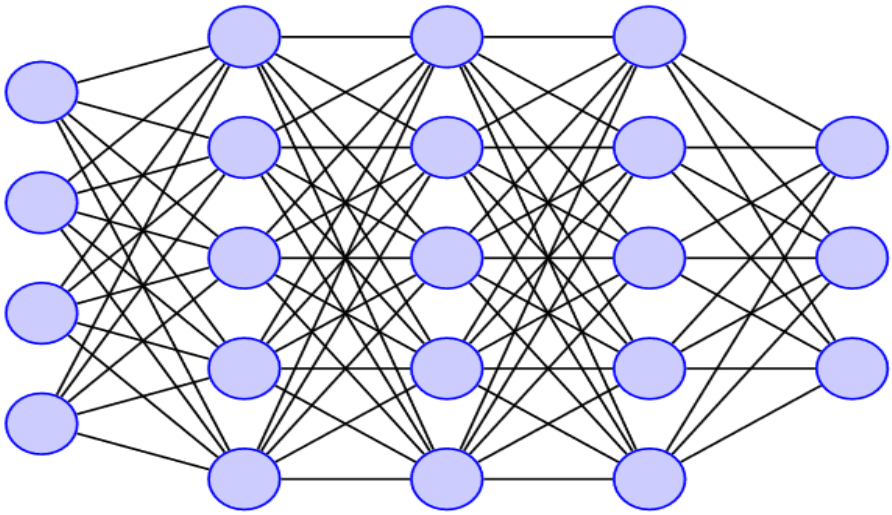


Abbildung 21: Ein Künstliches Neuronales Netz [Neu24]

Eingabeschicht (Input Layer): Nimmt die zu verarbeitenden Daten auf.

Verborgene Schichten (*Hidden Layers*): Führen Berechnungen durch, die auf den Eingangsdaten und den Verbindungsgewichten basieren. Die Komplexität eines neuronalen Netzes hängt oft von der Anzahl und Größe der verborgenen Schichten ab. Der Name *Hidden layer* rührt daher, dass ihre Ausgaben außerhalb des Netzwerks nicht sichtbar sind.

Ausgabeschicht (*Output Layer*): Gibt das Ergebnis der Verarbeitung durch das Netz aus.

Durch die Berechnung der gewichteten Summe der Eingänge und die Anwendung einer Aktivierungsfunktion kann das Netzwerk komplexe Funktionen approximieren. Dies wird erreicht, indem es die Gewichte und die Biases während des Trainingsprozesses anpasst.

3.2.3 Convolutional Neural Network

*Convolutional Neural Network*s (CNNs) sind eine Klasse künstlicher neuronaler Netzwerke (KNNs), die speziell für die Verarbeitung von Bilddaten konzipiert wurden. Diese Netzwerke bestehen auch aus Neuronen, die sich über mehrere Schichten erstrecken aber speziell darauf ausgelegt sind, die komplexen Strukturen innerhalb

von Bildern effizient zu erkennen und zu interpretieren. Der Schlüssel zu ihrer Leistungsfähigkeit liegt in der Art und Weise, wie sie die räumlichen Hierarchien von Bildern durch ihre einzigartige Architektur erfassen, die eine direkte Verarbeitung von rohen Bildpixeln ermöglicht [ON15].

Der hauptsächliche Unterschied zwischen CNNs und traditionellen KNNs liegt in ihrer primären Nutzung im Bereich der Mustererkennung innerhalb von Bildern. CNNs ermöglichen es, bildspezifische Merkmale direkt in die Architektur einzubetten, was sie besonders effizient für bildzentrierte Aufgaben macht und gleichzeitig den für das Modell erforderlichen Parametrisierungsaufwand verringert. Ein signifikantes Problem traditioneller KNNs besteht darin, dass sie mit der Rechenkomplexität, die für die Verarbeitung von Bilddaten erforderlich ist, oft an ihre Grenzen stoßen. Während Benchmark-Datensätze für maschinelles Lernen wie die MNIST-Datenbank handschriftlicher Ziffern mit ihrer geringen Bildgröße von 28×28 für die meisten KNNs geeignet sind, zeigt sich die Herausforderung bei größeren, farbigen Bildern mit einer Eingangsgröße von 64×64 deutlich. Hier steigt die Anzahl der Gewichte pro Neuron im ersten versteckten Layer erheblich auf 12.288. Diese Zunahme der Gewichte und die Notwendigkeit eines deutlich größeren Netzwerks verdeutlichen die Limitationen traditioneller Modelle im Umgang mit komplexen Bilddaten [ON15].

Drei verschiedene Schichttypen – Faltungsschichten (*Convolutional Layers*), *Pooling*-Schichten und vollständig verbundene Schichten (*Fully Connected Layers*) – bilden die Grundlage von CNNs [ON15].

Faltungsschicht (*Convolution Layer*)22: Diese Schicht nutzt Filterkerne (Faltungskerne), die über das Bild verschoben werden. An jeder Position wird ein Ausgabewert berechnet, der von der Position des Kernels abhängig ist. Die Parameter der Filterkerne werden während des Trainingsprozesses gelernt. Konvolutionsschichten in neuronalen Netzen helfen, die Modellkomplexität durch die Anpassung spezifischer Hyperparameter zu reduzieren: *Depth* , *Stride* und *Padding*. Die Tiefe des Ausgabevolumens wird durch die Anzahl der Neuronen in der Schicht festgelegt, und eine Anpassung dieser Tiefe kann sowohl die Anzahl der Neuronen im Netzwerk verringern als auch die Performanz des Modells beeinflussen. Die Schrittweite (*Stride*) ist ein weiterer Parameter, der dazu genutzt werden kann, die raumliche Auflösung des Outputs zu verringern und gleichzeitig ein Subsampling zu bewirken. *Padding* wird eingesetzt, um den Eingangsdaten an den Rändern Nullen hinzuzufügen. Dies stellt sicher, dass der Output auch bei grösseren Kernen die Größe des Inputs beibehält [ON15].

***Pooling-Schicht* 23:** Das Ziel dieser Schicht ist es, die Menge der Informationen zu reduzieren und dabei die wichtigsten Merkmale zu erhalten. Das gängigste Verfahren hierfür ist das *Max-Pooling*, bei dem aus einem Bereich des Kernels (üblicherweise 2×2 mit einer Schrittweite von 2) der maximale Wert extrahiert und weitergeleitet wird. Dies skaliert die Aktivierungsmap auf 25% der ursprünglichen Größe herunter, während das Tiefenvolumen auf seiner Standardgröße beibehalten wird. Neben

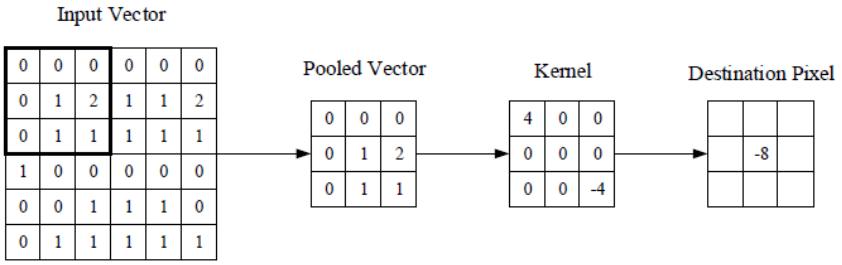


Abbildung 22: Eine visuelle Darstellung einer Faltungsschicht [ON15]

Max-Pooling existieren auch andere *Pooling*-Verfahren, wie *Average-Pooling*, bei dem der Durchschnittswert eines Bereichs weitergegeben wird [ON15] [SMB10].

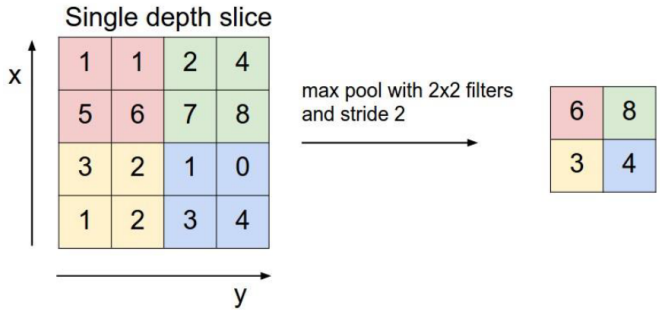


Abbildung 23: *Max-Pooling* [Pet17]

Durch die Kombination von *Convolution*- und *Pooling*-Schichten in verschiedenen Abfolgen und Tiefen können CNNs komplexe Hierarchien von Merkmalen in den Daten lernen. Dies ermöglicht es den Netzwerken, von einfachen Kanten und Texturen bis hin zu komplexen Objekten in Bildern zu generalisieren. Diese Architektur macht CNNs besonders mächtig für Aufgaben der Bilderkennung, Bildklassifizierung und auch für komplexere Probleme wie die Segmentierung von Bildinhalten oder Objekterkennung in Echtzeit. [ON15]

Die Architektur eines CNN ergibt sich aus dem Stapeln dieser Schichten. Eine solche vereinfachte Architektur ist in Abbildung 24 gezeigt.

Die Funktionsweise des genannten Beispiel-CNNs gliedert sich in vier Hauptbereiche [ON15]:

1. Analog zu anderen KNN-Formen behält die Eingabeschicht die Bildpixelwerte.

2. In der Faltungsschicht wird der Output der Neuronen ermittelt, die an lokale Regionen des Inputs angebunden sind, indem das Skalarprodukt zwischen ihren Gewichten und dem entsprechenden Eingangsregion berechnet wird. Die

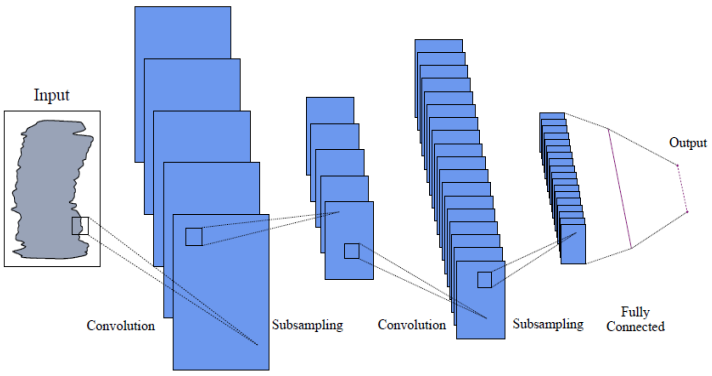


Abbildung 24: Ein typisches *Convolutional Neural Network* [Sei18]

Rektifizierte Lineare Einheit (kurz ReLu) hat das Ziel, eine elementweise Aktivierungsfunktion wie die Sigmoid-Funktion auf den Ausgang der vorherigen Schichtaktivierung anzuwenden.

3. Durch die *Pooling*-Schicht erfolgt eine Reduzierung der räumlichen Dimensionalität des Eingangs mittels Downsampling, was zur weiteren Verringerung der Parameteranzahl innerhalb dieser Aktivierung führt.

4. Die Aufgaben der vollständig verbundenen Schichten ähneln denen in Standard-KNNs, wobei sie darauf abzielen, aus den Aktivierungen Klassenscores zu generieren, die für die Klassifikation genutzt werden. Es wird zudem empfohlen, ReLu zwischen diesen Schichten einzusetzen, um die Leistungsstärke zu optimieren.

Durch diese einfache Methode der Transformation können CNNs die ursprüngliche Eingabeschicht Schicht für Schicht unter Verwendung von *Convolution*- und *Downsampling*-Techniken transformieren, um Klassenscores für Klassifizierungs- und Regressionszwecke zu erzeugen.

Es ist jedoch wichtig zu beachten, dass das bloße Verständnis der Gesamtarchitektur einer CNN-Architektur nicht ausreicht. Die Erstellung und Optimierung dieser Modelle können einige Zeit in Anspruch nehmen und ziemlich verwirrend sein.

3.3 Das Maskenbild

Ein Maskenbild, auch oft in der Literatur *Ground Truth* genannt wird und in Abbildung 25 dargestellt ist, ist eine Maske über dem ROI (*Region of Interest*), bei der die Pixel, die das Objekt enthalten, als wahr und überall sonst als falsch markiert sind.

Abbildung 25: Links: Normalenbild; Rechts: Aus der dazugehörigen Annotationsdatei extrahiertes Maskenbild

3.4 Image Augmentation

Bild Augmentieren (*Image augmentation*) ist ein Verfahren zur Steigerung der Leistungsfähigkeit von maschinellen Lernmodellen, insbesondere wenn nur begrenzte Datensätze zur Verfügung stehen. Wie in Abbildung 26 zu sehen ist, erhöht diese Methode die Vielfalt, indem sie künstlich einen kleinen Datensatz erweitert und neue Bilder generiert, die modifizierte Versionen der Originaldaten sind. Dieses Projekt widmet sich elementaren Transformationen wie Drehung, Spiegelung und Verzerrung von Bildern, um verschiedene Perspektiven und Ausrichtungen zu simulieren.

Das Ziel der Image Augmentation ist es, eine Diversität in den Trainings- und Validierungsdaten zu erstellen. Diese Diversität ist der Schlüssel dazu, Modelle zu trainieren, die sich gut auf neue, ungesehene Daten verallgemeinern lassen. Indem das Modell mit verschiedenen Situationen und Modifikationen der Originalbilder präsentiert wird, lernt es, die Muster effektiver zu identifizieren. Dieser Ansatz kann helfen, *Overfitting* zu verhindern und kann zu einer größeren Genauigkeit führen.

3.5 KI-Verfahren für Bildsegmentierung

Dieses Kapitel widmet sich speziell den KI-Verfahren, die benutzt wurden, um diese Arbeit durchzuführen. Dabei wird eine detaillierte Untersuchung dreier bedeutender Modelle durchgeführt, die in der aktuellen Forschung und industriellen Anwendung weit verbreitet sind.

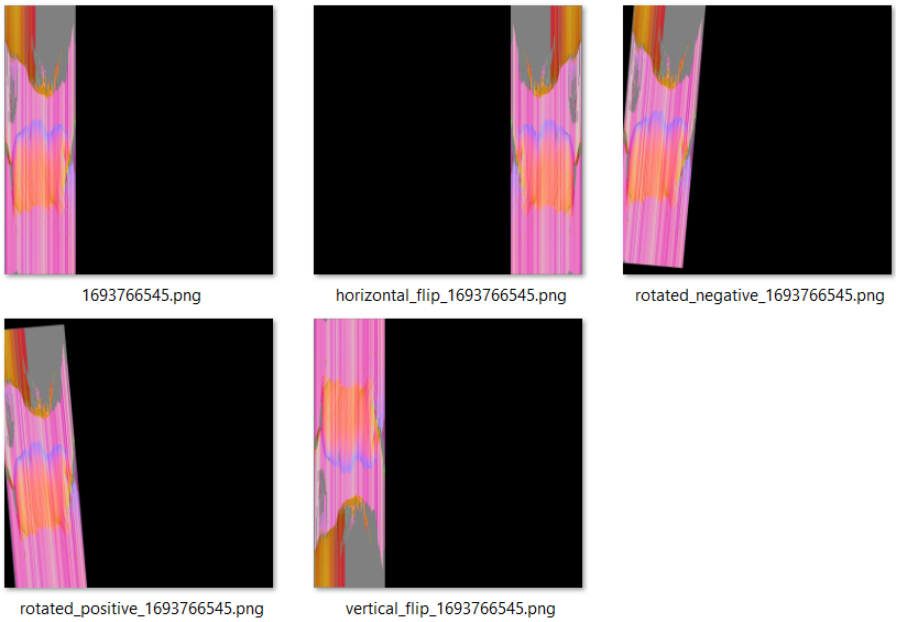


Abbildung 26: Beispiel einer Image Augmentation

3.5.1 Mask RCNN

Im Jahr 2017 wurde Mask RCNN [HGDG18] als Weiterentwicklung von Faster-RCNN [RHGS17] eingeführt. Faster RCNN arbeitet in zwei Stufen: Die erste Stufe ist ein *Region Proposal Network* (RPN), das potenzielle Objektbegrenzungsrahmen (*object bounding boxes*) vorschlägt. In der zweiten Stufe, die im Grunde genommen Fast RCNN [Gir15] entspricht, werden aus jedem vorgeschlagenen Begrenzungsrahmen unter Verwendung von RoIPool Merkmale extrahiert, die dann für die Klassifizierung und die Regression der Begrenzungsrahmen verwendet werden, wobei die Merkmale beider Stufen für eine effizientere Inferenz geteilt werden können. Mask RCNN baut auf dem Objekterkennungsmodell Faster RCNN auf, indem es einen zusätzlichen Zweig einführt, der neben den bestehenden Zweigen für die Objekterkennung und die Regression der Begrenzungsrahmen auch die Vorhersage von Segmentierungsmasken ermöglicht [HGDG18] (siehe Abbildung 27).

Das Mask RCNN-Modell umfasst mehrere Teile, wie in der folgenden Abbildung dargestellt ist (siehe Abbildung 28):

Feature Extraction Network, FEN (Netzwerk zur Merkmalsextraktion): Dieser Teil basiert auf dem Zusammenspiel des ResNet 101-Neural Networks [HZRS15] und des *Feature Pyramid Networks* [LDG+17]. Seine Hauptaufgabe ist die Berechnung von Faltungsmerkmalen in verschiedenen Maßstäben über das gesamte Bild hinweg. Das FPN (siehe Abbildung 29) verwendet eine von oben nach unten gerichtete Architektur mit Querverbindungen, um aus einem Eingabebild mit einheitlicher Größe eine Merkmalspyramide innerhalb des Netzwerks zu generieren [LDG+17].

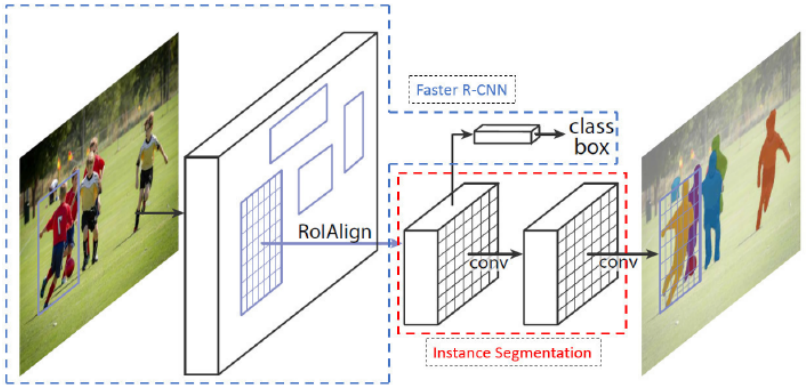


Abbildung 27: Schema der Mask RCNN-Architektur [HGDG18]

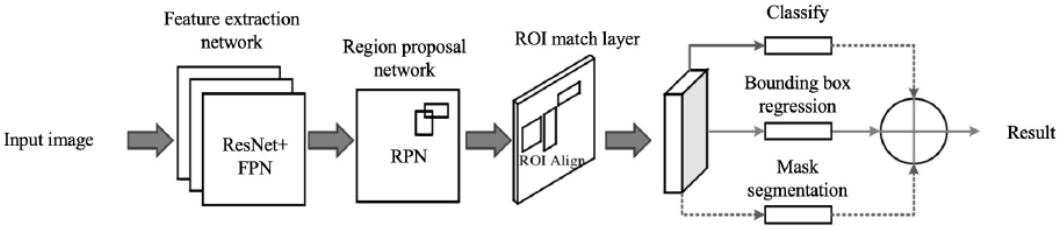


Abbildung 28: Struktur des Mask RCNN-Netzwerks für Instanzsegmentierung [Cha24]

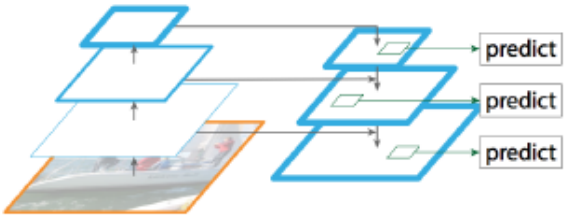


Abbildung 29: Das FPN Schema, source [LDG+17]

Das *Region Proposal Network* (RPN) [RHGS17] wird genutzt, um die relevanten Bereiche eines Bildes zu identifizieren, indem es rechteckige Vorschlagsbereiche generiert. Diese Bereiche werden durch das Anwenden einer gleitenden Faltung (Sliding Convolution) auf Merkmalsserien bestimmt, die aus dem Netzwerk zur Merkmalsextraktion stammen. Anschließend werden die vorgeschlagenen Regionen mittels eines Klassifizierungssystems ohne vordefinierte Kategorien erstellt. Das Netzwerk, welches als Zweiklassen-Klassifizierer fungiert, entscheidet, ob in den vorgeschlagenen Regionen Objekte von Interesse vorhanden sind. Es berechnet die Koordinaten des Zentrumpunkts, die Länge und Breite des Eingabebildes, die den Bereich des Interesses (ROI) jeder Region entsprechen, und legt die Position der Objektbegrenzungsrahmen fest.

Der ROI-Align-Teil, eine Weiterentwicklung gegenüber der vorherigen *ROI-Pooling*-Methode, verbessert die Genauigkeit der Maskenform und ist eine Kernoperation, um eine kleine Merkmalskarte aus jedem Bereich des Interesses (ROI) zu extrahieren. Die Methode des ROI-Align behebt eine wesentliche Einschränkung des *ROI-Pooling*, das in Mask RCNN [HGDG18] verwendet wird.

Das Hauptproblem, das durch ROI-Align im Vergleich zum *ROI-Pooling* gelöst wird, ist der Verlust der räumlichen Genauigkeit. Bei der *ROI-Pooling*-Methode kann der Prozess der Quantisierung des Interessenbereichs in ein festes Raster aufgrund von Rundungen während des Quantisierungsprozesses zu einer Fehlausrichtung zwischen den ursprünglichen Bildmerkmalen und den zusammengefassten Merkmalen führen. Diese Fehlausrichtung kann Ungenauigkeiten bei der Objektlokalisierung und -segmentierung zur Folge haben [HGDG18].

ROI-Align löst dieses Problem, indem es bilineare Interpolation verwendet, um die exakten Werte der Merkmale an den Bruchteilkoordinaten zu berechnen, anstatt den ROI in diskrete Behälter zu quantisieren. Durch die Bewahrung des räumlichen Layouts der Merkmale innerhalb jedes ROI ermöglicht ROI-Align genauere Merkmalsdarstellungen für die nachfolgende Verarbeitung, was zu einer verbesserten Leistung bei der Objektlokalisierung und -segmentierung führt [HGDG18].

Im abschließenden Teil, bekannt als branch layer, finden drei spezifische Vorgänge statt: die Klassifizierung, die Regression des Begrenzungsrahmens und die Generierung der Maskenkontur. Durch den Einsatz des Klassifizierungszweigs und des Regressionszweigs für den Begrenzungsrahmen werden sowohl die Positionskoordinaten als auch die entsprechenden Kategorien für die vorgeschlagenen Begrenzungsrahmen festgelegt. Zum Abschluss prognostiziert der Segmentierungszweig, unter Verwendung des Fully Convolutional Networks (FCN), die binären Masken für sämtliche im Bild identifizierten Objekte, was zur Erstellung des segmentierten Bildes führt.

3.5.2 U-NET

In diesm Abschnitt werden wir uns mit U-NET befassen, einer renommierten Architektur (siehe Abbildung 31) im Bereich der Bildsegmentierung. U-NET unter-

scheidet sich von anderen beliebten Architekturen wie ResNet, die für Bildklassifizierungsaufgaben konzipiert sind. Bei der Bildklassifizierung wird einem gesamten Bild ein Klassenlabel zugewiesen, während U-NET die semantische Segmentierung ermöglicht, bei der jedem Pixel ein Klassenlabel für eine detaillierte, pixelgenaue Segmentierung zugewiesen wird. Die folgende Abbildung 30 illustriert das Segmentierungsergebnis (zyanfarbene Maske) mit manueller Annotation (gelber Rand), unter Verwendung eines U-NET [RFB15]:

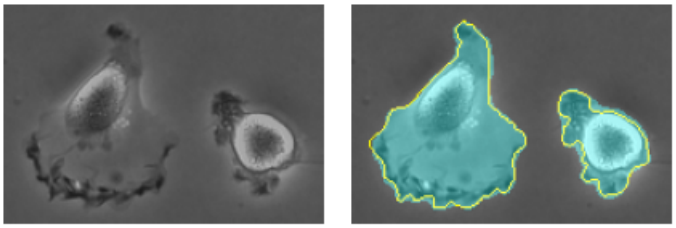


Abbildung 30: U-NET Segmentierung einer Zelle aus dem ‚ISBI cell tracking challenge. [RFB15]

Ursprünglich wurde U-NET 2015 von Olaf Ronneberger für die medizinische Bildsegmentierung vorgeschlagen und hat sich insbesondere bei Aufgaben wie der Zellsegmentierung als bemerkenswert erfolgreich erwiesen. Die Architektur selbst wird als U-förmig beschrieben, daher der Name U-NET. Sie verfügt über einen Encoder (kontrahierender Pfad) und einen Decoder (expansiver Pfad). Der Encoder besteht aus konvolutionellen und *Max-Pooling*-Schichten, um Bildmerkmale zu erfassen, während der Decoder Up-Sampling-Schichten und konvolutionelle Schichten umfasst, um die Segmentierung aus diesen Merkmalen zu rekonstruieren.

Ein charakteristisches Merkmal von U-NET sind seine Skip-Verbindungen, die dazu beitragen, Lokalisierungsinformationen zu bewahren, indem Merkmale aus frühen und späteren Schichten zusammengeführt werden. Diese Struktur hat sich bei Bildsegmentierungsaufgaben als sehr effektiv erwiesen [RFB15].

Um ein noch klareres Verständnis der Funktionsweise von U-NET zu vermitteln, betrachten wir, wie das Netzwerk ein Bild verarbeitet. Der Prozess beginnt im Downsampling-Teil mit einem Graustufen-Eingangsbild der Größe 572x572 Pixel. Dieses Bild durchläuft verschiedene Faltungsschichten (Convolutional Layers), die wichtige Bildmerkmale extrahieren. Diese Merkmale werden anschließend durch *Max-Pooling* mit einem *Stride* von 2 herunterskaliert, wobei die Größe der Merkmalkarte (*feature map*) schrittweise reduziert und die Anzahl der Merkmalkanäle (*feature channels*) jeweils verdoppelt wird, um die Tiefe und die Fähigkeit zur Merkmalserkennung zu erhöhen [RFB15].

Im *Upsampling*-Teil werden diese Merkmale hochskaliert und mit Merkmalen aus früheren Schichten zusammengeführt, wobei die gleiche Auflösung beibehalten wird, um eine effektive Integration der Merkmale zu gewährleisten. Dieser Prozess wiederholt sich, wobei die *feature map* schrittweise vergrößert und die Details ver-

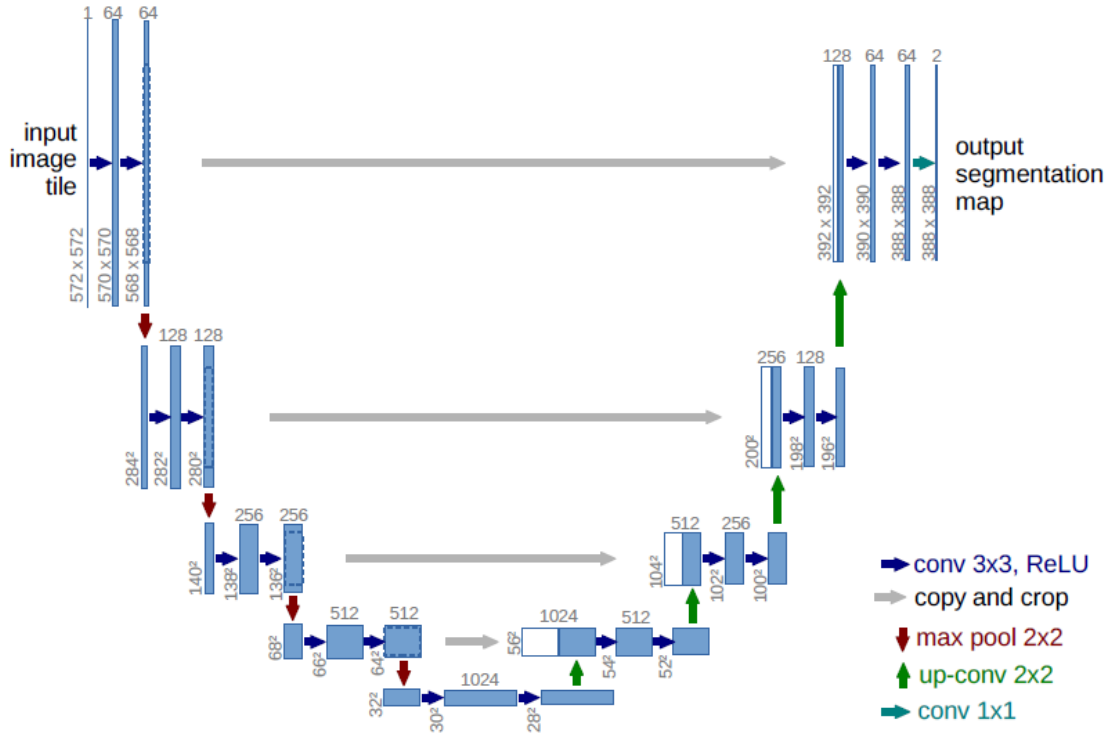


Abbildung 31: U-NET-Architektur (Beispiel für 33x33 Pixel in der niedrigsten Auflösung). Jedes blaue Feld entspricht einer *multi-channel feature map*. Die Anzahl der Kanäle ist oben auf dem Feld angegeben. Die x-y-Größe ist an der unteren linken Kante des Feldes angegeben. Weiße Felder stellen kopierte *feature maps* dar. Die Pfeile bezeichnen die verschiedenen Operationen [RFB15].

feinert werden, bis die endgültige Segmentierungskarte erstellt wird. Der Ausgang besteht typischerweise aus mehreren Klassen, wie Vorder- und Hintergrund, die in der medizinischen Bildgebung Zellen und Nicht-Zellen darstellen könnten [RFB15].

Der Ausgang des U-NET besteht typischerweise aus mehreren Klassen, wie beispielsweise Vorder- und Hintergrund, die in der medizinischen Bildgebung zum Differenzieren zwischen Zellen und Nicht-Zellen verwendet werden könnten. Es ist möglich, dass U-NET in spezialisierten Anwendungen auch mehrere spezifische Zelltypen unterscheiden kann, abhängig von der Trainingseinrichtung und den Zielparametern [RFB15].

über die gesamte Architektur hinweg verwenden die meisten Faltungsschichten einen 3x3-Kernel, um eine effektive und präzise Feature-Extraktion zu gewährleisten. Die letzte Schicht verwendet einen 1x1-Kernel, der das Ergebnis auf die gewünschte Anzahl von Klassen abbildet, was eine klare und scharfe Definition der verschiedenen Segmente ermöglicht. Zusätzliche Komponenten wie Batch-Normalisierung und ReLU-Aktivierungsfunktionen folgen den konvolutionellen Schichten. Diese Elemente helfen, die neuronale Aktivierung zu standardisieren und zu optimieren, was insgesamt die Leistung des Netzwerks verbessert und zu genauerer Segmentierung führt [RFB15].

3.5.3 YOLO

Im folgenden Abschnitt werden wir den YOLO-Algorithmus vorstellen. YOLO kurz für (*You Only Look Once*) ist ein weit verbreiteter Algorithmus zur Objekterkennung, der erstmals im Jahr 2015 veröffentlicht wurde (siehe Abbildung 32)[RDG+16]. Seit seiner ersten Veröffentlichung hat YOLO zahlreiche *Updates* und Verbesserungen erfahren, die es zu einem leistungsstarken Werkzeug in der Bildverarbeitung gemacht haben [JCQ23a]. Im Jahr 2023 wurde die kommerzielle Version YOLOv8 veröffentlicht. Es wurde auf diese Version entschieden, weil sie leichter zu installieren und bis dato state-of-art in YOLO Algorithmen zur Segmentierung ist. Da jedoch bislang keine wissenschaftliche Publikation zu YOLOv8 verfügbar ist, werde ich mich darauf konzentrieren, den YOLO-Algorithmus in seiner allgemeinen Form zu erläutern.

Die meisten Objekterkennungsalgorithmen bestehen aus mehreren Komponenten, die zusammenwirken, um Objekte in Bildern zu identifizieren. Im Beispiel von Faster RCNN zur Objektdetektion beginnt der Prozess mit der Generierung von Bereichsvorschlägen, bei denen potenzielle *Bounding Boxes*, innerhalb eines Bildes erstellt werden. Anschließend wird einen Klassifikator auf diese vorgeschlagenen *Bounding Boxes* angewendet, um festzustellen, welche davon tatsächlich Objekte enthalten. Nach der Klassifizierung erfolgt, basierend auf der Präsenz anderer Objekte in der Szene, ein Post-Processing-Schritt, in dem die *Bounding Boxes* verfeinert, doppelte Erkennungen eliminiert und die *Bounding Boxes* neu bewertet werden. Diese drei Hauptkomponenten – Bereichsvorschläge, Klassifizierung und Post-Processing – sind entscheidend für die Funktionsweise und Effizienz von Objekter-

kennungsalgorithmen [RDG+16].

Der YOLO-Algorithmus integriert die verschiedenen Komponente der Objekterkennung in ein einziges neuronales Netzwerk. Zunächst wird die Größe des Eingabebildes auf die erforderlichen Dimensionen reduziert. Anschließend wird ein einzelnes konvolutionelles neuronales Netzwerk (CNN) auf das Bild angewendet. Schließlich werden die resultierenden Erkennungen anhand der Konfidenzwerte des Modells gefiltert, um die Zuverlässigkeit der Erkennung zu gewährleisten [RDG+16].

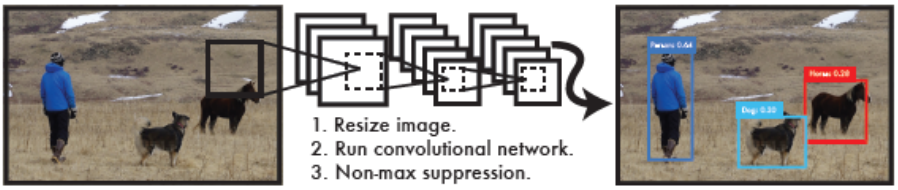


Abbildung 32: YOLO Detektionsystem [RDG+16]

In Details wird zunächst das Bild durch YOLO in ein Raster aufgeteilt (siehe Abbildung 33). Jede dieser Zellen prognostiziert mehrere Begrenzungsrahmen (*Bounding Boxes*) sowie die dazugehörigen Konfidenzwerte (*Confidences*). Ein Konfidenzwert gibt an, wie wahrscheinlich es ist, dass ein *Bounding Box* ein relevantes Objekt enthält [RDG+16].

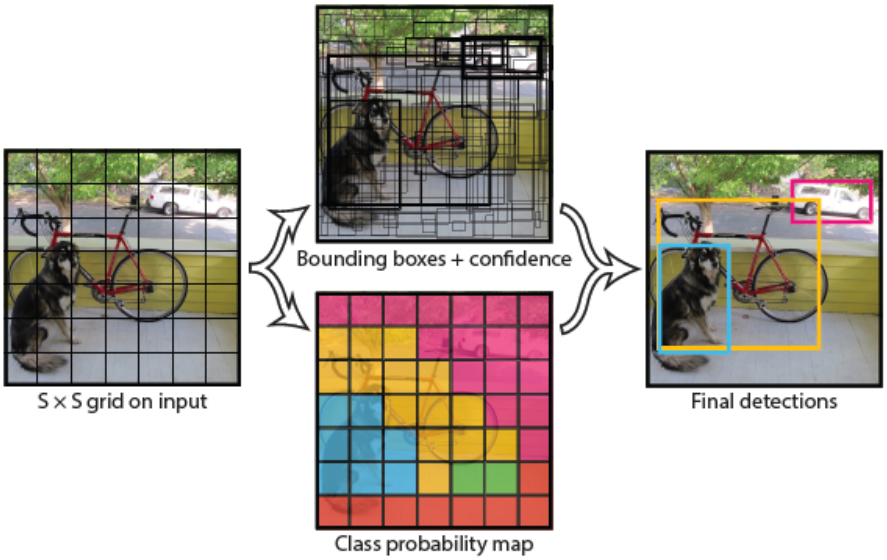


Abbildung 33: YOLO Algorithmus bildlich dargestellt [RDG+16]

Zusätzlich zu den Konfidenzwerte macht der Algorithmus Klassenvorhersagen, die jedem erkannten Objekt innerhalb einer Zelle eine Wahrscheinlichkeitsvertei-

lung über mögliche Kategorien zuweisen. Diese Erkennungsprozesse werden durch ein *Convolutional Neural Network* (CNN) ermöglicht, das die Eingabebilder analysiert. Das CNN durchläuft verschiedene Schichten, die das Bild schrittweise komprimieren und dabei immer abstraktere Merkmale extrahieren [RDG+16].

In der Endphase des Algorithmus erzeugt das Netzwerk einen Tensor, der Koordinaten der *Bounding Boxes*, Klassenwahrscheinlichkeiten und Konfidenzwerte enthält. Um die Genauigkeit zu erhöhen, verwirft der Algorithmus alle Vorhersagen, deren Konfidenzwerte unter einem vorgegebenen Schwellenwert liegen. Dies hilft, falsch positive Ergebnisse zu reduzieren [RDG+16].

Ein weiteres wichtiges Merkmal von YOLO ist die Anwendung der Non-Maximum Suppression (NMS), die überlappungen von Begrenzungsrahmen behandelt. Wenn mehrere Rahmen für dasselbe Objekt prognostiziert werden, behält YOLO nur den Rahmen mit dem höchsten Vertrauenswert bei [RDG+16].

Darüber hinaus kann YOLO in seiner neuesten Version (YOLO v8) auch die Instanz Segmentierung durchführen. Diese Funktion prognostiziert die genauen Pixel eines Objekts. Dies macht YOLO zu einem vielseitigen Werkzeug, das nicht nur in typischen Szenarien der Objekterkennung Anwendung findet, sondern auch in wissenschaftlichen Bereichen, wo detaillierte und präzise Bildanalysen entscheidend sind.

3.6 Bewertungsmetriken für die Bildsegmentierung

Dieses Kapitel konzentriert sich auf die wichtigsten Metriken, die in diser Forschung verwendet werden, um die Qualität der Bildsegmentierungser-gebnisse zu messen. Es werden zwei zentrale Metriken vorgestellt, die in der Evaluierung von Bildsegmentierungsalgorithmen weit verbreitet sind.

3.6.1 Jaccard-Index

Jaccard-Index, auch bekannt als *Intersection over Union* (IoU), ist eine Bewertungsmetrik, die in der Bildverarbeitung zur Quantifizierung der überlappung zwischen Zielmasken und Vorhersagemasken verwendet wird. IoU misst das Verhältnis der Schnittmenge (die gemeinsamen Pixel) von Zielmaske und Vorhersagemaske zur Vereinigungsmenge beider Masken (siehe Abbildung 34)[RTG+19]:

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} =$$

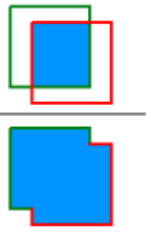


Abbildung 34: *Intersection over Union* [PNdS20]

Mathematisch wird IoU mit der folgenden Formel berechnet:

$$\text{IoU} = \frac{\text{Target} \cap \text{Prediction}}{\text{Target} \cup \text{Prediction}}$$

Diese Metrik wird normalerweise bei mehrklassigen Detektionen separat für jede Klasse berechnet, und anschließend wird ein Durchschnittswert gebildet, um den globalen durchschnittlichen IoU-Wert zu erhalten [Jor21].

3.6.2 Average Precision

Die *Average Precision* (AP) ist eine weitere Schlüsselmetrik zur Bewertung der Leistung von Modellen in der der Bildsegmentierung. AP liefert ein Maß für die Qualität der Segmentierung für eine einzige Klasse, indem sie die Fläche unter der *Precision-Recall*-Kurve berechnet. Diese Kurve zeigt den Trade-off zwischen *Precision* (Präzision) und *Recall* (Trefferquote) für unterschiedliche Schwellenwerte der Vorhersagewahrscheinlichkeit [PNdS20].

Die *Precision-Recall*-Kurve wird durch Plotting von *Precision*-Werten gegen *Recall*-Werte für verschiedene Schwellenwerte erzeugt. Die *Precision* ist definiert als das Verhältnis von wahren Positiven zu der Summe aus wahren Positiven und falschen Positiven, während *Recall* das Verhältnis von wahren Positiven zu der Summe aus wahren Positiven und falschen Negativen ist. Mathematisch lassen sich diese Beziehungen wie folgt ausdrücken:

1. **Precision** gibt an, welcher Anteil der vom Modell als positiv klassifizierten Vorhersagen tatsächlich korrekt ist. Sie wird mit der Formel

   $$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

   berechnet, wobei "True Positives"die korrekt als positiv identifizierten Vorhersagen sind und "False Positives"die fälschlicherweise als positiv klassifizierten Vorhersagen [PNdS20].

2. **Recall** misst, wie viele der tatsächlichen positiven Fälle vom Modell korrekt als solche erkannt wurden. Die Berechnung erfolgt mittels

   $$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

   , wobei "False Negatives"jene Fälle sind, die das Modell fälschlicherweise nicht als positiv erkannt hat [PNdS20].

AP wird dann als die Fläche unter dieser Kurve berechnet, was ein aggregiertes Maß für die Modellleistung über alle möglichen Schwellenwerte darstellt. Für Probleme mit nur einer Klasse, wie in Ihrer Anfrage, bietet die AP eine direkte Einschätzung der Modellfähigkeit zur korrekten Identifikation und Segmentierung der Zielklasse [PNdS20].

AP ist besonders nützlich in Szenarien, wo die Balance zwischen *Precision* und *Recall* von Bedeutung ist, und bietet einen Einblick in, wie gut das Modell in der Lage ist, korrekte Vorhersagen zu machen, ohne von der Anzahl der Klassen beeinflusst zu werden. Für Segmentierungsaufgaben mit nur einer Klasse liefert AP somit eine klare und fokussierte Bewertung der Modellleistung [Jor21].

4 Umsetzung

Das vorliegende Kapitel beschreibt die Umsetzung der Methoden, die im Abschnitt 3.5 zur Segmentierung von Schweißnähten erläutert wurden. Zunächst wird die Datensammlung und -annotation erläutert, gefolgt von den angewendeten Vorverarbeitungsmethoden. Im Anschluss wird die Durchführung des Bildaugmentierungsprozesses beschrieben. Der Trainingsprozess der Modelle wird detailliert beschrieben, einschließlich der Trainingsstrategie, der Verbesserung des Netzwerkcodes und der praktischen Durchführung der Trainings. Diese methodischen Schritte bilden die Grundlage für die spätere Bewertung und den Vergleich der Segmentierungsergebnisse der drei Modelle.

4.1 Datensammlung und Annotation der Bilder

Datensammlung

Die initiale Phase meines Forschungsprojekts konzentrierte sich auf die Identifikation und Sammlung von Pseudo-3D-Bilddaten, die für das Training eines neuronalen Netzwerks zur Detektion von Schweißnahtgrenzen von großer Bedeutung sind. Diese Daten wurden mittels Exports aus der **VIROwsi**-Software im .zip-Format erzeugt. In diesen Exports sind alle Informationen enthalten, die jede einzelne Naht betrifft, wie die eingelernten Nahtgrenzen. Diese Exports können in der **VIROwsi**-Software zur Weiterverarbeitung wieder importiert werden.

Ein kritischer Aspekt der Datenauswahl war die Verfügbarkeit einer bereits trainierten Detektion der Schweißnahtgrenzen in den Daten. Dies vereinfacht die spätere Annotation, da dann nicht neue Polygone erlernt werden müssen, sondern lediglich die Koordinaten der Punkte korrigiert werden müssen. Ein anderer Aspekt war die Diversität der gesammelten Bilddaten. Die Varianz der Daten, im Sinne von Variationen in den Schweißnahttypen, -positionen und -eigenschaften, wurde höher bewertet als die Quantität der Bilder pro Naht. Diese Strategie diente dazu, das Trainingsmodell gegenüber einem breiten Spektrum von Szenarien robust zu machen, um seine Generalisierungsfähigkeit zu verbessern.

Die folgenden Tabellen bieten eine detaillierte Aufschlüsselung nach Produkt, mit spezifischen Nahtnummern und der Anzahl der Bilder pro Naht für jedes Teil der Datensammlung:

- **Pseudo3D-Daten vor der Erweiterung:**

Tabelle 1, Tabelle 2 und Tabelle 3.

- **Pseudo3D-Daten zum Erweitern der bestehenden Daten:**

Tabelle 4, Tabelle 5, Tabelle 6, Tabelle 7 und Tabelle 8.

Nahtnummer	Anzahl der Bilder pro Naht
1	16
2	16
3	16
4	16
5	16
6	16
8	16
9	16
10	16
11	16
13	16
14	16
15	16
16	16
21	16
22	16
23	16
24	16
25	16
30	16
31	16
37	16
38	16

Tabelle 1: Übersicht der Pseudo 3D Bilddaten vom Produkt: Hilfsrahmen_VW316_VorgabenVW

Nahtnummer	Anzahl der Bilder pro Naht
308	3
309	3
3011	3

Tabelle 2: Übersicht der Pseudo 3D Bilddaten vom Produkt: OP40_UZB

Nahtnummer	Anzahl der Bilder pro Naht
308	3
309	3
3011	3
401	10
402	10
403	10
404	10
405	10
406	10
407	10
408	10
409	10
413	10
419	10
420	10

Tabelle 3: Übersicht der Pseudo 3D Bilddaten vom Produkt: BT1

Nahtnummer	Anzahl der Bilder pro Naht
1	16
6	15
8	16
9	16
22	16
23	16
24	16

Tabelle 4: Übersicht der Pseudo 3D Bilddaten vom Produkt: D5A8

Nahtnummer	Anzahl der Bilder pro Naht
1	2
3	9
4	9
5	12
6	12

Tabelle 5: Übersicht der Pseudo 3D Bilddaten vom Produkt: Kamenz_Lasernaht

Nahtnummer	Anzahl der Bilder pro Naht
1	15

Tabelle 6: Übersicht der Pseudo 3D Bilddaten vom Produkt: Nissan

Nahtnummer	Anzahl der Bilder pro Naht
1	4
2	4
3	4

Tabelle 7: Übersicht der Pseudo 3D Bilddaten vom Produkt: PB300V

Nahtnummer	Anzahl der Bilder pro Naht
1	15

Tabelle 8: Übersicht der Pseudo 3D Bilddaten vom Produkt: PB310_ST2150

Technische Erweiterungen und Integration

Nachdem die Bilddaten zusammengestellt waren, wurden in der **VIROwsi**-Software bestehende Methoden angepasst und weiterentwickelt, um diese Daten effizient zu verarbeiten. Kernpunkte dieser Optimierung waren die Erzeugung und lokale Speicherung von Normalen- und Layer-Bildern, basierend auf Pseudo-3D-Daten, sowie die Erstellung von Annotationsdateien mithilfe der erweiterten Methode zur Detektion von Nahtgrenzen. Die Anpassung der Methode *ViMethPseudo3dToNormalImage* ermöglichte es, dass die erzeugten Normalenbilder nun in einem spezifischen lokalen Verzeichnis gespeichert werden und dieselben Bezeichnungen wie ihre zugehörigen Annotationsdateien tragen. In ähnlicher Weise wurde die Methode **ViMeth102605PrepareForUnwind** modifiziert, um Layer direkt aus den Pseudo-3D-Daten zu extrahieren und diese unter identischen Namen wie die Annotationsdateien lokal zu speichern. Darüber hinaus habe ich die Nahtgrenzen-Detektionsmethode erweitert, um die Koordinaten in JSON-Dateien im LabelMe-Format zu exportieren. Die Implementierung dieser erweiterten Funktionen erfolgte in C++ innerhalb der Entwicklungsumgebung Microsoft Visual Studio 2017. Das folgende C++ Code-Listing demonstriert die Erweiterung zur Speicherung von Nahtgrenzenkoordinaten im LabelMe-Format:

Listing 1: Main Code

```
std::vector<Point> topPoints;
std::vector<Point> bottomPoints;
for (int i = regionStart; i <= regionEnd; ++i) {
  topPoints.push_back({static_cast<double>(i) , static_cast<
     double>(topBorder[i])});
  bottomPoints.push_back({ static_cast<double>(i) ,
     static_cast<double>(bottomBorder[i]) });
}

double tolerance = 1;
std::vector<Point> simplifiedPointsTop;
```

```
douglasPeucker(topPoints, tolerance, 0, topPoints.size() - 1,
    simplifiedPointsTop);

std::vector<Point> simplifiedPointsBottom;
douglasPeucker(bottomPoints, tolerance, 0, bottomPoints.size
   () - 1, simplifiedPointsBottom);


saveWeldLimits4Labelme(outputPath, imgName,
   simplifiedPointsBottom, simplifiedPointsTop, pseudo->
   GetScanWidth(), pseudo->GetNumberOfValidScans());
```

- **Definition der Point-Struktur:**

Listing 2: Point-Struktur

```
struct Point {
  double x, y;

  bool operator==(const Point& other) const {
    return x == other.x && y == other.y;
  }
};
```

Diese Struktur repräsentiert einen 2D-Punkt im Raum, wobei x und y die Koordinaten des Punktes sind. Ein überladungsoperator == wird definiert, um die Gleichheit zweier Punkte zu überprüfen, was in verschiedenen Algorithmen zur Datenverarbeitung verwendet wird.

- **Berechnung des senkrechten Abstands:**

Listing 3: Berechnung des senkrechten Abstands

```
double perpendicularDistance(const Point& p, const Point&
   start, const Point& end) {

double num = std::abs((end.y - start.y) * p.x - (end.x -
   start.x) * p.y + end.x * start.y - end.y * start.x);

double den = std::sqrt((end.y - start.y) * (end.y - start.y)
   + (end.x - start.x) * (end.x - start.x));
```

```
  return num / den;
}
```

Diese Funktion berechnet den senkrechten Abstand eines Punktes zu einem Linienabschnitt, was ein kritischer Schritt im Douglas-Peucker-Algorithmus zur Reduzierung der Punkteanzahl in einer geometrischen Form ist.

- **Douglas-Peucker-Algorithmus:**

Listing 4: Douglas-Peucker-Algorithmus

```
void douglasPeucker(const std::vector<Point>& points, double
   tolerance, size_t start, size_t end, std::vector<Point>&
   result) {
  if (end <= start + 1)
    return;

  double maxDistance = 0;
  size_t index = 0;

  for (size_t i = start + 1; i < end; ++i) {
    double distance = perpendicularDistance(points[i], points
       [start], points[end]);
    if (distance > maxDistance) {
      maxDistance = distance;
      index = i;
    }
  }

  if (maxDistance > tolerance) {
    // Recursively simplify the two sub-segments
    douglasPeucker(points, tolerance, start, index, result);
    douglasPeucker(points, tolerance, index, end, result);
  }
  else {
    // Keep the endpoints within tolerance
    result.push_back(points[start]);
    result.push_back(points[end]);
  }
}
```

Der Douglas-Peucker-Algorithmus ist ein Verfahren zur Reduzierung der Anzahl von Punkten in einer Kurve, indem Punkte, die einen minimalen Einfluss auf die Gesamtform haben, entfernt werden, um die Datenverarbeitungseffizienz zu verbessern.

- **Speichern von Nahtgrenzen im LabelMe-Format:**

Die folgende Funktion konvertiert die verarbeiteten Daten in das JSON-Format von LabelMe. Die Funktion passt die Koordinaten der Nahtgrenzen an, kombiniert die Punkte von Unter- und Obergrenze und speichert das Ergebnis in einer JSON-Datei. Dies erleichtert die Integration der Daten in Annotationstools und unterstützt die visuelle Datenanalyse.

Listing 5: Speichern von Nahtgrenzen im LabelMe Format

```
void saveWeldLimits4Labelme(std::string outputPath, std::
    string imgName, std::vector<Point>& bottomBorder, std::
    vector<Point>& topBorder, int scanHeight, int scanWidth) {

  bottomBorder.erase(std::unique(bottomBorder.begin(),
      bottomBorder.end()), bottomBorder.end());
  topBorder.erase(std::unique(topBorder.begin(), topBorder.
      end()), topBorder.end());


  std::reverse(topBorder.begin(), topBorder.end());

  std::vector<Point> combinedPoints(bottomBorder);

  combinedPoints.insert(combinedPoints.end(), topBorder.begin
      (), topBorder.end());


  // Erstelle den JSON-String manuell
  std::string json_str = "{\n";
  json_str += "  \"version\": \"5.3.0\",\n";
  json_str += "  \"flags\": {},\n";
  json_str += "  \"shapes\": [\n";
  json_str += "    {\n";
  json_str += "      \"label\": \"weld\",\n";
  json_str += "      \"points\": [\n";

  for (size_t i = 0; i < combinedPoints.size(); ++i) {
    json_str += "        [\n";
```

```
      json_str += "␣␣␣␣␣␣␣␣␣␣" + std::to_string(combinedPoints[
         i].x) + ",\n";
      json_str += "␣␣␣␣␣␣␣␣␣␣" + std::to_string(scanHeight -
         combinedPoints[i].y) + "\n";
      json_str += "␣␣␣␣␣␣␣␣]";
      if (i < combinedPoints.size() - 1) {
        json_str += ",";
      }
      json_str += "\n";
    }

    json_str += "␣␣␣␣␣␣],\n";
    json_str += "␣␣␣␣␣␣\"group_id\":␣null,\n";
    json_str += "␣␣␣␣␣␣\"description\":␣null,\n";
    json_str += "␣␣␣␣␣␣\"shape_type\":␣\"polygon\",\n";
    json_str += "␣␣␣␣␣␣\"flags\":␣{}\n";
    json_str += "␣␣␣␣}\n";
    json_str += "␣␣],\n";
    json_str += "␣␣\"imagePath\":␣\"" + imgName + ".png\",\n";
    json_str += "␣␣\"imageData\":␣null,\n";
    json_str += "␣␣\"imageHeight\":␣" + std::to_string(
       scanHeight) + ",\n";
    json_str += "␣␣\"imageWidth\":␣" + std::to_string(
       scanHeight) + "\n";
    json_str += "}\n";

    // Speichere den JSON-String in einer Datei
    std::ofstream outfile(outputPath + imgName + ".json");
    if (outfile.is_open()) {
      outfile << json_str;
      outfile.close();
      std::cout << "JSON␣wurde␣erfolgreich␣in␣output.json␣
         gespeichert.\n";
    }
    else {
      std::cerr << "Fehler␣beim␣Oeffnen␣der␣Datei␣zum␣Speichern
         ␣des␣JSONs.\n";
    }
}
```

Annotation der Bilddaten

Die anschließende Phase der Datenaufbereitung und technische Erweiterungen

der **VIROwsi** umfasste die detaillierte Annotation der Bilddaten mittels LabelMe, eine Aufgabe, die sich durch ihre Komplexität und ihren zeitlichen Aufwand auszeichnete. Die sorgfältige manuelle Korrektur und überprüfung jeder Bildannotation war entscheidend, um die Genauigkeit und Zuverlässigkeit der Trainings- und Testdaten zu gewährleisten. Besondere Herausforderungen ergaben sich aus der Notwendigkeit, die annotierten Schweißnahtgrenzen mit den Original-Pseudo3D-Bildern abzugleichen, besonders in Fällen, in denen die Grenzen auf den Normalen- oder Layer-Bildern nicht eindeutig identifizierbar waren. Diese gründliche Arbeit erforderte nicht nur ein tiefes Verständnis der zugrundeliegenden Bildmerkmale, sondern auch Fachwissen, um die Konsistenz und Genauigkeit der Annotationen sicherzustellen.

4.2 Vorverarbeitungsmethoden

Die Phase der Vorverarbeitung spielt eine kritische Rolle in der Vorbereitung der Bilddaten für das Training der neuronalen Netze zur Segmentierung. Aufgrund der spezifischen Anforderungen dieser Netze und der Eigenheiten der gesammelten Daten ist eine direkte Verwendung der annotierten Bilddaten nicht möglich. Verschiedene Python-Skripte wurden entwickelt, um die Daten entsprechend zu konvertieren und zu modifizieren. Diese Skripte ermöglichen eine effiziente Anpassung der Daten und gewährleisten, dass sie in einem optimalen Format für das Training der neuronalen Netze vorliegen.

Aktualisierung von Bilddaten und Bildbreite ($updateImageDataAndWidth.py$)

Das Skript $updateImageDataAndWidth.py$ bearbeitet JSON-Dateien in einem spezifizierten Verzeichnis und dessen Unterordnern. Es entfernt die Bilddaten *imageData* aus den JSON-Dateien und setzt die Bildbreite *imageHeight* auf 1248 Pixel. Diese Anpassung ist notwendig, um einerseits die Konsistenz der Bildbreite im gesamten Datensatz zu gewährleisten und andererseits die Größe der Annotationsdateien zu reduzieren, indem überflüssige Bilddaten eliminiert werden. Allerdings der Hauptgrund für das Entfernen der Bilddaten ist die Vermeidung von Problemen, die entstehen können, wenn der Speicherort oder Pfad der Bilder geändert wird. Solche änderungen führten bisher dazu, dass die Annotationssoftware LabelMe die entsprechenden Annotationsdateien nicht mehr korrekt laden konnte.

Bilder mit Padding versehen ($padding\_images.py$)

Das Skript $padding\_images.py$ passt die Größe der Bilder an, indem es bei Bedarf schwarzes Padding auf der rechten Seite hinzufügt. Dieser Schritt ist notwendig, um alle Bilder auf eine einheitliche Größe zu bringen, ohne ihr Seitenverhältnis zu verändern. Dadurch wird sichergestellt, dass alle Bilder dieselben Dimensionen für das Training des Netzwerks aufweisen, was Bildverzerrungen vermeidet. Solche Verzerrungen könnten sonst die Merkmale im Bild künstlich verändern und die

Leistung des Modells beeinträchtigen.

Bildgrößenanpassung ($resize.py$)

Mit dem Skript $resize.py$ werden die Bilder auf die Größe 512x512 skaliert und die entsprechenden Annotationsdaten aktualisiert, um der neuen Bildgröße zu entsprechen. Diese Anpassung ist erforderlich, um eine einheitliche Eingangsgröße für das Netzwerk sicherzustellen, was eine Voraussetzung für das effiziente Training des Modells ist. Die Angleichung der Bildgrößen verbessert sowohl die Berechnungseffizienz als auch den Speicherverbrauch während des Trainingsprozesses. Für eine gesteigerte Berechnungseffizienz werden oft mehrere Bilder simultan in einem Batch verarbeitet.

Extraktion des blauen Kanals ($extractBlueChannel.py$)

Das Skript $extractBlueChannel.py$ bearbeitet Bilder im *input_folder*, indem es den blauen Kanal aus jedem Bild extrahiert, die ursprüngliche Ordnerstruktur beibehält und die extrahierten blauen Kanalbilder im $output\_folder$ speichert. Die Konzentration auf den blauen Kanal beruht auf den besonderen visuellen Merkmalen der Schweißnahtgrenzen, die sich im blauen Kanal deutlicher vom Hintergrund abheben als in vollfarbigen Bildern. Diese Vorgehensweise hat mir bei der Annotation der Bilddaten, bei denen die Nahtgrenzen schwer erkennbar waren, erheblich geholfen und ich musste seltener in der **VIROwsi** nach den korrekten Nahtgrenzen suchen.

Korrektur des Bildpfadformats ($correctjpg2png.py$)

Mit $correctjpg2png.py$ werden die ïmagePathFormate in den LabelMe JSON-Annotationsdateien von PNG zu JPG aktualisiert. Diese Korrektur ist notwendig, um Inkonsistenzen in den Dateiformaten innerhalb des Datensatzes zu bereinigen und eine einheitliche Verwendung von Bildformaten zu gewährleisten.

Konvertierung zu PNG ($convert2png.py$)

Das letzte Skript $convert2png.py$ konvertiert Bilder in das PNG-Format und speichert die konvertierten Bilder im angegebenen Ausgabeverzeichnis. Diese Umwandlung trägt dazu bei, eine einheitliche Dateiformatierung über den gesamten Datensatz hinweg sicherzustellen, was für die Verarbeitung und das Training des neuronalen Netzwerks wichtig ist.

Diese Skripte bilden zusammen einen umfassenden Vorverarbeitungsworkflow, der sicherstellt, dass die Bilddaten in einem für das Training optimalen Zustand vorliegen. Durch die Automatisierung dieser Prozesse wird nicht nur die Effizienz der Datenvorbereitung erhöht, sondern auch die Konsistenz und Zuverlässigkeit der Trainingsdaten verbessert.

Datensatz aufteilen($createValidationSet.py$)

Das Skript $createValidationSet.py$ ist unerlässlich für das Training von Mask R-CNN-Modellen, da es zufällig Daten aus dem gesamten Set auswählt und diese automatisch in Trainings- und Validierungssets aufteilt (70% Training, 30% Validierung). Dies gewährleistet eine repräsentative und unvoreingenommene Auswahl an Daten für beide Sets. Die manuelle Durchführung dieser Schritte wäre nicht nur zeitaufwändig, sondern auch anfällig für Fehler und könnte zu einer unausgewogenen Datenaufteilung führen.

4.3 Augmentieren der Bilder

Die Augmentation der Bilder ist ein essenzieller Schritt, um die Vielfalt der Trainingsdaten zu erhöhen und die neuronalen Netzwerke robuster gegenüber Variationen in den Bilddaten zu machen. Da in den vorhandenen Netzwerk-Codes von YOLOv8 und U-Net im Internet keine passende Augmentationstools verfügbar waren, war es notwendig, ein eigenes Tool in Python zu entwickeln. Der Prozess in diesem selbstentwickelten Tool gliedert sich in die folgenden Schritte: Zunächst werden aus den ursprünglichen JSON-Dateien Masken der Bilder erzeugt. Danach werden diese Masken zusammen mit den Originalbildern auf die gleiche Weise augmentiert, wobei die Zuordnung auf Basis ihrer Namen erfolgt. Anschließend nutzt das Tool die generierten augmentierten Masken, um die korrespondierenden JSON-Dateien zu erzeugen, die dann den augmentierten Originalbildern zugeordnet werden. Abschließend werden die Bilder umbenannt und ihre entsprechenden JSON-Dateien aktualisiert, um sicherzustellen, dass die Dateinamen in den JSON-Dateien mit den umbenannten Bilddateinamen übereinstimmen.

Von LabelMe zu Masken ($labelme2mask.py$)

Das Skript $labelme2mask.py$ verarbeitet JSON-Dateien aus einem spezifizierten Bilderordner und generiert entsprechende Masken in einer Unterordnerstruktur. Diese Masken stellen eine visuelle Repräsentation der in den JSON-Dateien annotierten Bereiche dar und sind ein kritischer Zwischenschritt für die Augmentation, da sie zusammen mit den Originalbildern augmentiert werden.

Bildaugmentation ($augment.py$)

Das $augment.py$ Skript führt die Bildaugmentation für zwei Bildersets durch und speichert die augmentierten Bilder. Das Skript umfasst verschiedene Augmentationstechniken wie horizontale und vertikale Spiegelungen, Rotationen und Verzerrungen. Nach der Ausführung dieses Skripts liegen die augmentierten Bilder in den angegebenen Ausgabeordnern vor, organisiert nach der Art der angewendeten Augmentation 35. Diese augmentierten Bilder und Masken bilden die Grundlage für eine erweiterte und diversifizierte Trainingsdatenbasis.

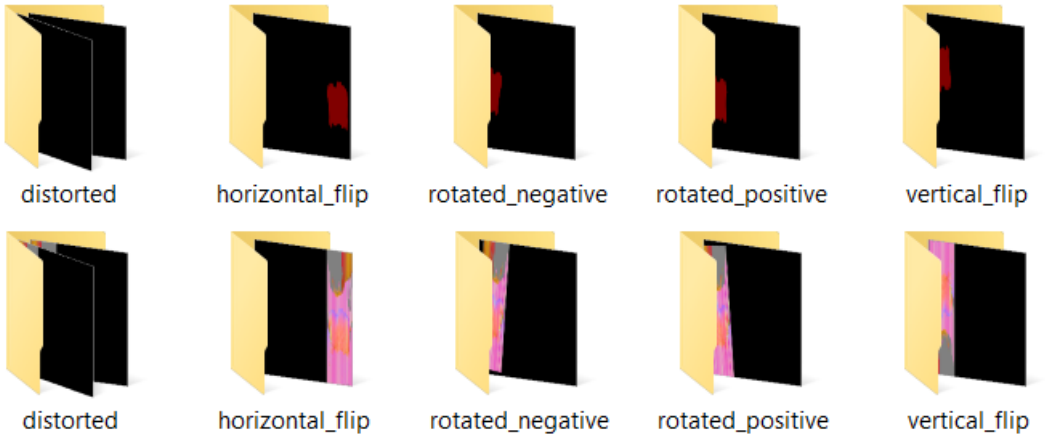


Abbildung 35: Beispiele für augmentierte Original- und Maskenbilder

Das Skript verwendet folgende Augmentationstechniken:

1. **Horizontaler Flip (Spiegelung)**: Das Bild wird horizontal gespiegelt.
2. **Vertikaler Flip (Spiegelung)**: Das Bild wird vertikal gespiegelt.
3. **Rotation nach positiv**: Das Bild wird um einen positiven Winkel von 5 Grad gedreht.
4. **Rotation nach negativ**: Das Bild wird um einen negativen Winkel von 5 Grad gedreht.
5. **Verzerrung (*Distortion*)**: Das Bild wird zufällig perspektivisch verzerrt, wobei die Anzahl der Verzerrungen (zwischen 3 und 5) und die Stärke der Verzerrung (eine zufällige Zahl zwischen 0,01 und 0,05) variieren.

Von Masken zu LabelMe ($mask2labelme.py$)

Mit dem $mask2labelme.py$ Skript werden aus den Bildern korrespondierende LabelMe JSON-Annotationsdateien erstellt. Dieser Schritt ist entscheidend, um die augmentierten Masken in ein Format zu überführen, das für das Training des neuronalen Netzwerks nutzbar ist. Es wird eine Menge von JSON-Annotationsdateien erzeugt, die den Maskenbildern in dem spezifizierten Verzeichnis entsprechen.

Umbenennung der Dateien ($rename.py$)

Das $rename.py$-Skript benennt Bilder um und aktualisiert deren zugehörige JSON-Dateien, um sicherzustellen, dass die Dateinamen in den JSON-Dateien den umbenannten Bilddateinamen entsprechen. Hinweis: Vor diesem Schritt sollten die

JSON-Dateien von augmented_Masks\1 nach augmented_Images\1 kopiert werden. Die Umbenennung wird durchgeführt, weil die Bildnamen nach der Augmentation den Originalnamen plus den Augmentationstyp enthalten. Durch die Umbenennung erhalten die Dateien Namen nur einen Zeitstempel, wodurch die Namen kürzer und ohne direkten Verweis auf den Augmentationstyp gestaltet werden. Dies erleichtert eine eindeutige Zuordnung zwischen den augmentierten Bildern und ihren Annotationsdateien, indem es die Namenslänge reduziert und gleichzeitig die übersichtlichkeit erhöht.

Diese Pipeline zur Bildaugmentation stellt sicher, dass die für das Training des neuronalen Netzwerks verwendeten Daten nicht nur vielfältig, sondern auch genau annotiert sind. Durch die systematische Erzeugung augmentierter Bilder und ihrer zugehörigen Annotationen wird die Generalisierungsfähigkeit des Netzwerks verbessert, indem es auf eine breitere Palette von Bilddaten trainiert wird.

4.4 Training

Das Training neuronaler Netzwerke ist ein kritischer Prozess in der Entwicklung von KI-Systemen, da es direkt die Leistung und Effizienz der Modelle beeinflusst. Dieses Kapitel beschäftigt sich mit den verschiedenen Aspekten des Trainingsprozesses, von der Strategieplanung bis zur praktischen Durchführung.

4.4.1 Trainingsstrategie

Um den komplexen Anforderungen der Segmentierung von Schweißnähten gerecht zu werden und den Prozess so effizient wie möglich zu gestalten, habe ich ein detailliertes Experimentier- und Trainingskonzept entwickelt. Dieses Konzept basiert auf den drei neuronalen Netzwerkmodellen – **YOLOv8**, **Mask-RCNN** und **U-Net** –, die im Unterkapitel 3.5 vorgestellt wurden. Jedes dieser Modelle weist einzigartige Stärken für bestimmte Segmentierungsaufgaben auf. Um deren Potenzial vollständig auszuschöpfen, war eine präzise und methodische Herangehensweise an das Training notwendig.

Erster Schritt in der Trainingsstrategie ist die Erstellung von unterschiedlichen Datenszenarien. Zu diesem Zweck erstellte ich zehn unterschiedliche Datendecks, die in ihrer Zusammensetzung variierten, um eine breite Palette an Szenarien abzudecken. Diese Decks, die weiter unten beschrieben sind, unterscheiden sich hinsichtlich Bildformate (Layer-Bilder vs. Normalen-Bilder und Farbbilder vs. Graubilder), Erweiterung durch neue Daten und der Anwendung von Bildaugmentieren. Dieser vielfältige Datensatz sollte sicherstellen, dass die Modelle unter verschiedensten Bedingungen getestet und optimiert werden konnten, um ihre Anwendbarkeit in realen Szenarien zu maximieren.

1. **Deck01**: Verwendet Layer-Images.
2. **Deck02**: Verwendet Layer-Images, augmentiert.

3. **Deck03**: Verwendet farbige Normalen-Bilder.
4. **Deck04**: Verwendet farbige Normalen-Bilder, augmentiert.
5. **Deck05**: Verwendet farbige Normalen-Bilder, erweitert.
6. **Deck06**: Verwendet farbige Normalen-Bilder, erweitert und augmentiert.
7. **Deck07**: Verwendet Graue Normalen-Bilder.
8. **Deck08**: Verwendet Graue Normalen-Bilder, augmentiert.
9. **Deck09**: Verwendet Graue Normalen-Bilder, erweitert.
10. **Deck10**: Verwendet Graue Normalen-Bilder, erweitert und augmentiert.

Decks	Anzahl der Bilder
Deck01	497
Deck02	3719
Deck03	497
Deck04	3722
Deck05	694
Deck06	5200
Deck07	497
Deck08	3719
Deck09	694
Deck10	5200

Tabelle 9: Anzahl der Bilder pro Datendeck

Modelspezifische Strategien

- **YOLOv8:**

In diesem Training wurden die vortrainierten Modelle YOLOv8x-seg, YOLOv8m-seg und YOLOv8n-seg verwendet, die sich durch unterschiedliche Größen und Parameteranzahlen auszeichnen. YOLOv8x-seg ist ein großes Modell mit 71 Millionen Parametern, YOLOv8m-seg ist ein mittelgroßes Segmentierungsmodell mit über 27 Millionen Parametern und YOLOv8n-seg ein kleineres Modell mit 3 Millionen Parametern. Die erstellten Modelle wurden mit 10 Datendecks trainiert, wobei die Anzahl der Bilder pro Datendeck variabel ist und in Tabelle 9 dargestellt wird. Die annotierten Dateien im .json-Format wurden für das Training in das .txt-Format konvertiert, wobei das Werkzeug *json2yolo* verwendet wurde, das über pip installiert wurde.Die Trainings- und Validierungsbilder wurden nach einem Verhältnis von 80 % für das Training und 20 % für die Validierung und das Testen zufällig ausgewählt.

Für die Modelle, die für ihre schnelle Verarbeitungsgeschwindigkeit bekannt sind, wurde der Fokus darauf gelegt, das Gleichgewicht zwischen Verarbeitungsgeschwindigkeit und Genauigkeit zu optimieren. Hierfür wurden die vortrainierten Modelle (yolov8n-seg.pt, yolov8m-seg.pt, yolov8x-seg.pt) in Trainingsläufen mit jeweils 300 Epochen und einer Batch-Größe von 8 unter Verwendung der Standard-Hyperparameter eingesetzt. Diese Strategie zielte darauf ab, die Performance der Modelle in Echtzeitanwendungen zu bewerten.

Training	Pre-Trained Modell	Epochen	Batch-Größe
Train_0	yolov8n-seg.pt	300	8
Train_1	yolov8m-seg.pt	300	8
Train_2	yolov8x-seg.pt	300	8

Tabelle 10: YOLOv8 Trainingsplan

- **Mask R-CNN:**

Das Training des Mask-R-CNN-Modells wurde grundlegend mit den default Parametern ausgeführt. Die Hauptänderung besteht in der Anpassung der steps_per_epoch mit experimentellen Werten von 10, 50 und 100, um die optimale Anzahl von Trainingsschritten pro Epoche zu bestimmen. Die Anzahl der Epochen wurde konstant auf 300 festgelegt, während die zuvor im Code integrierten Augmentierungsmethoden nicht verwendet wurden. Diese Anpassungen erlauben eine präzisere Steuerung des Trainingsprozesses.

Training	Pre-Trained Modell	Lernrate	Epochen	Steps per Epoche
Train_0	mask_rcnn_coco.h5	0.001	300	10
Train_1	mask_rcnn_coco.h5	0.001	300	50
Train_2	mask_rcnn_coco.h5	0.001	300	100

Tabelle 11: Mask R-CNN Trainingsplan

- **U-Net:**

U-Net, besonders geeignet für die detaillierte Segmentierung medizinischer Bilder, wurde mit einer variierenden Anzahl von Epochen und unterschiedlichen Einstellungen für die Initialisierung und Dropout-Raten trainiert. Diese Diversifizierung in den Trainingsparametern und das Integrieren vom frühen Stoppen sollte das Modell robuster gegenüber Overfitting machen und seine Generalisierungsfähigkeit über verschiedene Bildtypen hinweg verbessern.

Allgemeine Einstellungen für alle Trainingsläufe:

1. Anzahl der Klassen (NUM_CLASSES) = 2
2. Eingabegröße (INPUT_SHAPE) = [512, 512, 3] (Höhe, Breite, Kanäle)
3. Batch-Größe (BATCH_SIZE) = 8
4. VAL_SUBSPLITS = 4

Training	Initialisierung	Dropout	Epochen
Train_0	0.02	0.5	100
Train_1	0.02	0.5	300
Train_2	0.001	0.1	300

Tabelle 12: U-NET Trainingsplan

4.4.2 Verbesserung der Netzwerke Code

Für Mask-RCNN [Sin] wurden verschiedene Maßnahmen ergriffen, um das Training effizienter zu gestalten, Overfitting zu vermeiden und den Speicher zu schonen, da jedes Modell nach jeder Epoche gespeichert wird. Dazu gehörten die Implementierung von *EarlyStopping*, der das Training abbricht, wenn das Modell nach 20 Epochen nicht mehr besser wird, und *CleanCheck*, der immer nur die Modelle der letzten 20 Epochen behält. Außerdem wurde *LossHistory* integriert, welches den Verlauf von loss und val_loss in Abhängigkeit von den Epochen in einer CSV-Datei speichert, um eine detaillierte Analyse des Trainings zu ermöglichen.

ähnlich wurden auch bei der Implementierung von U-Net [Zen] spezifische Anpassungen vorgenommen, darunter das Hinzufügen des *EarlyStopping*-Verfahrens sowie die Integration einer Option zur Speicherung des besten Modells. Diese Maßnahmen trugen dazu bei, das Training zu optimieren und die Leistung des Modells zu verbessern.

Für YOLOv8 [JCQ23b] waren bereits einige Funktionen standardmäßig implementiert, wie das *EarlyStopping* mit einem voreingestellten Patience von 50. Zusätzlich zur automatischen Speicherung des aktuellen besten Modells bietet YOLOv8 auch die Möglichkeit, das Training an der Stelle fortzusetzen, an der es unterbrochen wurde. Diese Funktionen tragen dazu bei, die Robustheit des Trainingsprozesses zu erhöhen und potenzielle Datenverluste zu minimieren.

Durch diese Anpassungen und Verbesserungen im Code wurden die drei Netzwerkarchitekturen erfolgreich für die Aufgabe der Schweißnaht-Detektion angepasst und optimiert.

4.4.3 Ausführung der Trainings

Um die Modelle für Mask R-CNN, U-Net und YOLOv8 auf Schweißnähte zu trainieren, befinden sich die für diese Arbeit verwendeten Datensätze, der angepasste

Quellcode unter [Yam24]. Weitere Einzelheiten sind im verlinkten *Repository* und den zugehörigen Dokumentationen zu finden.

Allgemein werden die folgenden Schritte ausgeführt, um die Netzwerke zu trainieren:

1. **Umgebung einrichten**
2. **Trainings- und Validierungsdaten erstellen**
3. **Annotationen konvertieren**
4. **Modell trainieren**

5 Ergebnisse

In diesem Kapitel werden die Leistungen von Mask R-CNN, U-Net und YOLO bewertet. Es wird die Genauigkeit und Schnelligkeit der Modelle bewertet, gemessen in *Intersection over Union* (IoU), *Average Precision* (AP) und den Segmentierungszeiten. Die Ergebnisse der Messung werden in Balkendiagrammen veranschaulicht. Diese Diagramme vergleichen einzeln drei Konfigurationen (train0, train1 und train2) für jedes Datendeck, wie im Unterkapitel 4.4.1 beschrieben wurde. Durch diesen Vergleich verschiedener Bildkonfigurationen, einschließlich Farb- und Grauwertbilder sowie augmentierter und erweiterter Daten, zeigt diese Untersuchung auf, wie verschiedene Konfigurationen die Performanz beeinflussen. Die gewonnenen Einsichten tragen zum Verständnis der Effektivität jeder Konfiguration bei und betont, wie wichtig eine sorgfältige Auswahl der Datenverarbeitung für den Erfolg in der praktischen Anwendung ist.

5.1 Mask R-CNN

Die IoU-Werte in Abbildung 36 weisen eine generelle Steigerung von Deck01 bis Deck10 auf, was darauf hindeutet, dass sich die Leistung des Modells durch eine zunehmende Datenoptimierung positiv entwickelt. Die Datensätze, die farbige Bilder (Deck03 bis Deck06) enthalten, weisen im Vergleich zu denen, die Grauwertbilder (Deck07 bis Deck10) enthalten, minimal höhere IoU-Werte auf, was die Wichtigkeit von Farbinformationen zur besseren Segmentierung hindeutet. Die Feststellung, dass Decks mit erweiterten Daten (Deck02, Deck04, Deck06, Deck08, Deck10) häufig höhere IoU-Werte zeigen, bestätigt auch die positive Wirkung des Datenaugmentierens auf die Leitung und Generalisierungsfähigkeit des Modells.

In Abbildung 37 lässt sich ein ähnliches Ergebnis für die AP-Werte wie aus den IoU-Werten sehen.

Wie aus der Abbildung 38 zu entnehmen ist, werden in der Gegenüberstellung der Segmentierungszeit deutliche Schwierigkeiten sichtbar. Farbige Bilder erfordern im Durchschnitt längere Segmentierungszeiten, was auf eine höhere Rechenlast hindeutet. Auch augmentierte und erweiterte Datensätze neigen zu längeren Segmentierungszeiten, was auf die vergrößerte Komplexität und den Umfang der zu verarbeitenden Daten zurückgeführt werden kann. Die unterschiedlichen Parameterkonfigurationen der Modelle (train0, train1 und train2) zeigen keinen signifikanten Einfluss auf die Leistung.

In Abbildung 39 sind unterschiedliche Testbeispiele zu sehen, die die verschiedenen Segmentationsergebnisse in unterschiedlichen Szenarien illustrieren. Links ist das Ergebnis für Deck02 dargestellt, in der Mitte für Deck06 und rechts für Deck10.

5.2 U-NET

Wie in Abbildung 40 zu sehen ist, verbessern sich die IoU-Werte generell mit der Einführung vom Augmentieren und Erweiterung der Daten. Decks, die augmen-

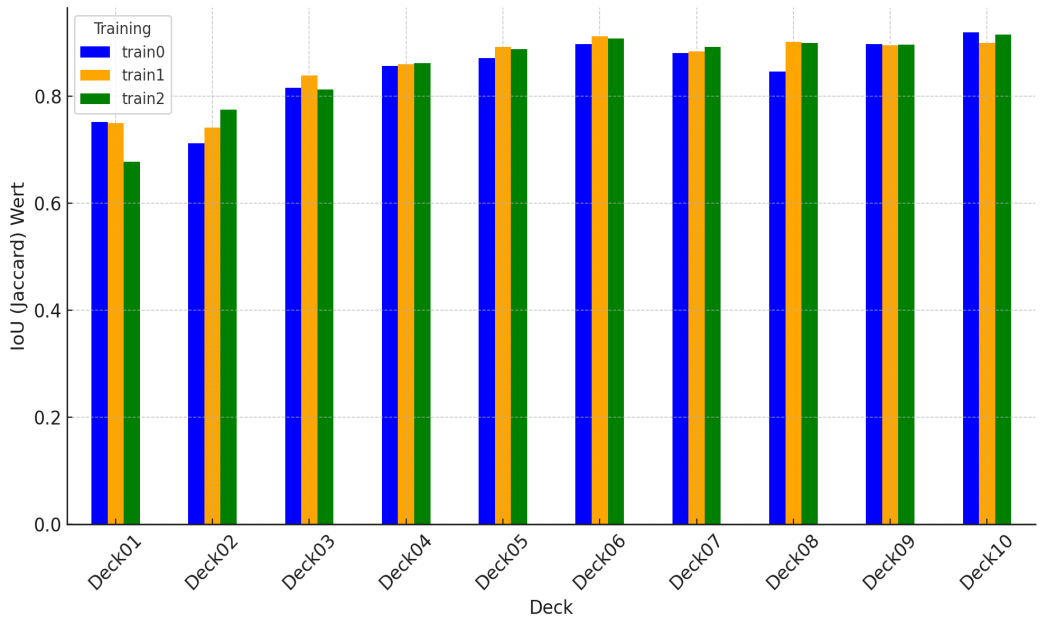


Abbildung 36: Mask R-CNN IoU Werte für jedes Deck und Training

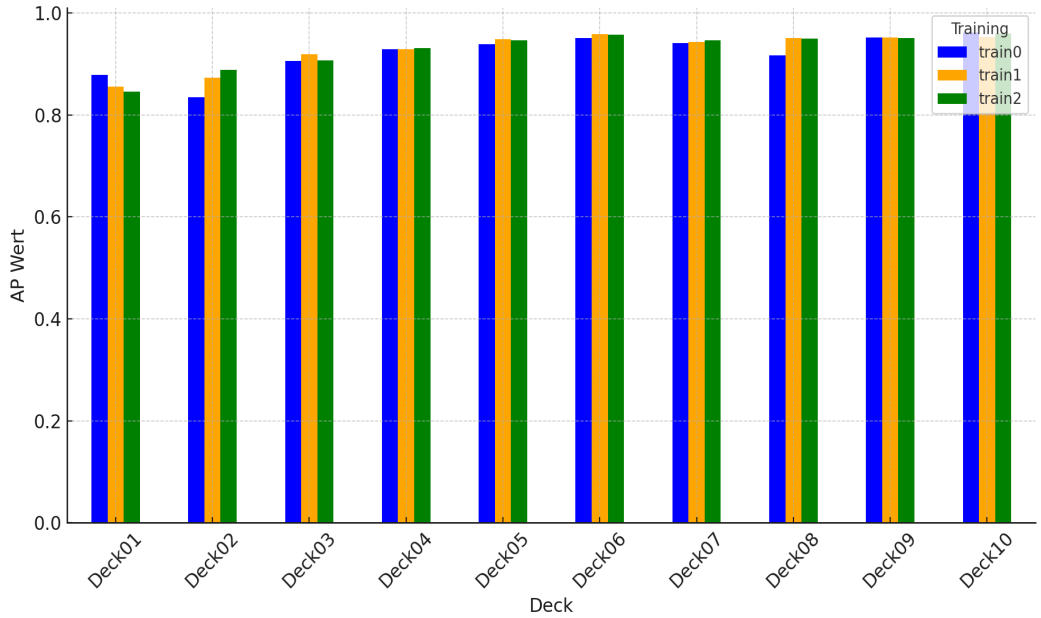


Abbildung 37: Mask R-CNN AP-Werte für jedes Deck und Training

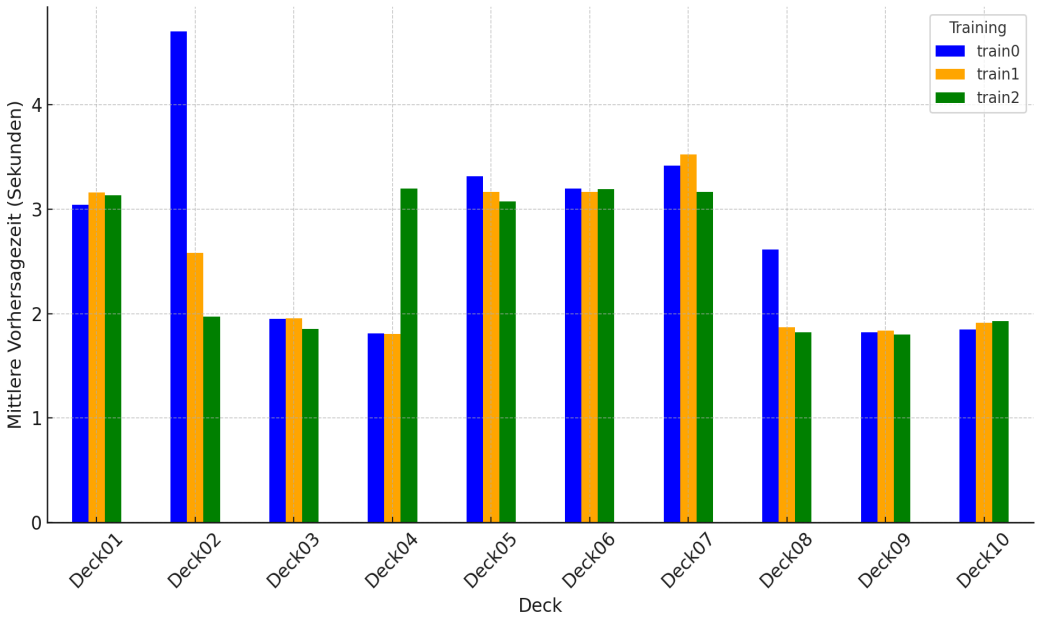


Abbildung 38: Mask RCNN Mittlere Segmentierungszeit Werte für jedes Deck und Training

tierte oder erweiterte Daten nutzen, verzeichnen einen Anstieg der IoU-Werte. Die Analyse legt zudem nahe, dass farbige Bilder, minimal, wertvollere Informationen für die Segmentierung liefern als Grauwertbilder. Decks, die sowohl Augmentieren als auch Erweiterungen einsetzen, zeigen die höchsten IoU-Werte. Die Layer-Bilder zeigen generell die schlechtesten IoU Werte.

Bei den AP-Werten in Abbildung 41 zeigt sich ein ähnlicher Trend.

Die Beobachtungen aus Abbildung 42 zur Segmentierungszeit sind etwas weniger eindeutig. Die Zeiten variieren zwar über die verschiedenen Decks, jedoch ohne eine klare Korrelation zu bestimmten Bearbeitungsmethoden wie Augmentieren oder Erweiterung. Interessanterweise zeigen sich keine signifikanten Unterschiede in den Segmentierungszeiten zwischen Decks mit farbigen und grauwertigen Bildern, was darauf schließen lässt, dass das U-Net Modell beide Bildformate ähnlich effizient verarbeitet. Ebenfalls bemerkenswert ist, dass augmentierte Datensätze keine merklichen Verzögerungen in der Verarbeitung verursachen, was darauf hindeutet, dass das Modell die zusätzliche Komplexität und Informationsvielfalt ohne negative Auswirkungen auf die Geschwindigkeit bewältigen kann.

In Abbildung 43 sind unterschiedliche Testbeispiele zu sehen, die die verschiedenen Segmentationsergebnisse in unterschiedlichen Szenarien illustrieren. Links ist das Ergebnis für Deck02 dargestellt, in der Mitte für Deck06 und rechts für Deck10.

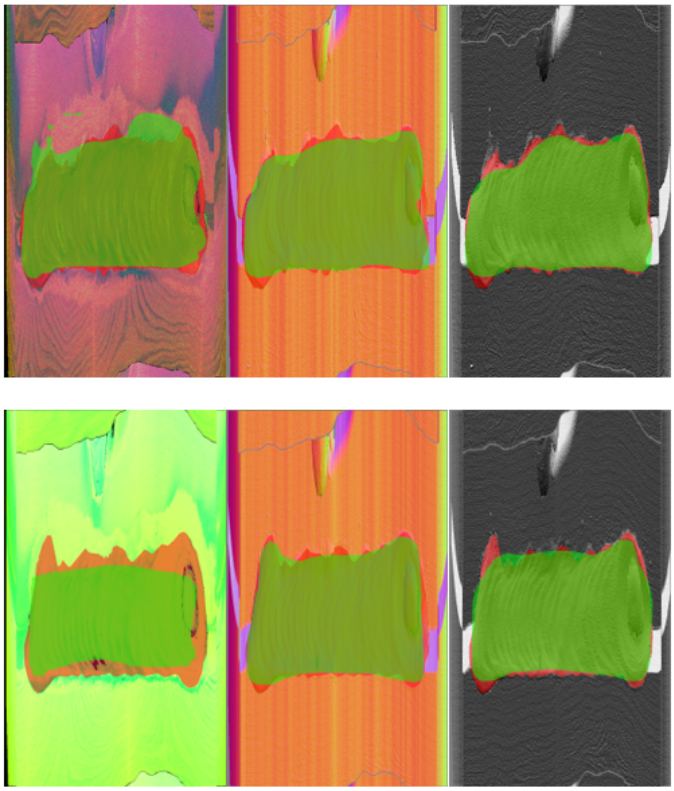


Abbildung 39: Beispiele aus Mask R-CNN Ergebnisse

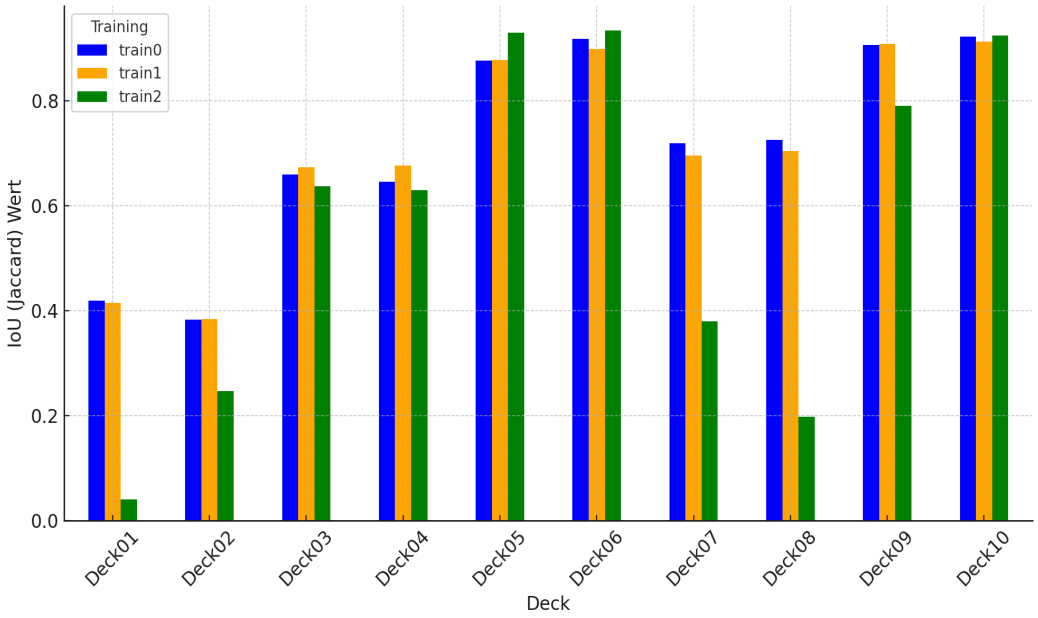


Abbildung 40: U-NET IoU Werte für jedes Deck und Training

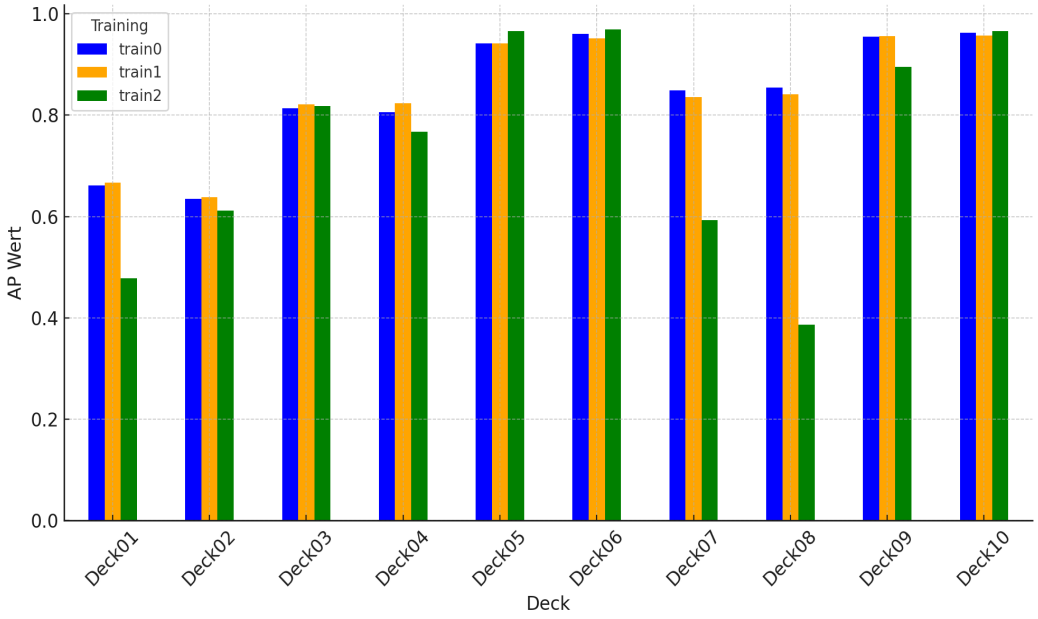


Abbildung 41: U-NET AP Werte für jedes Deck und Training

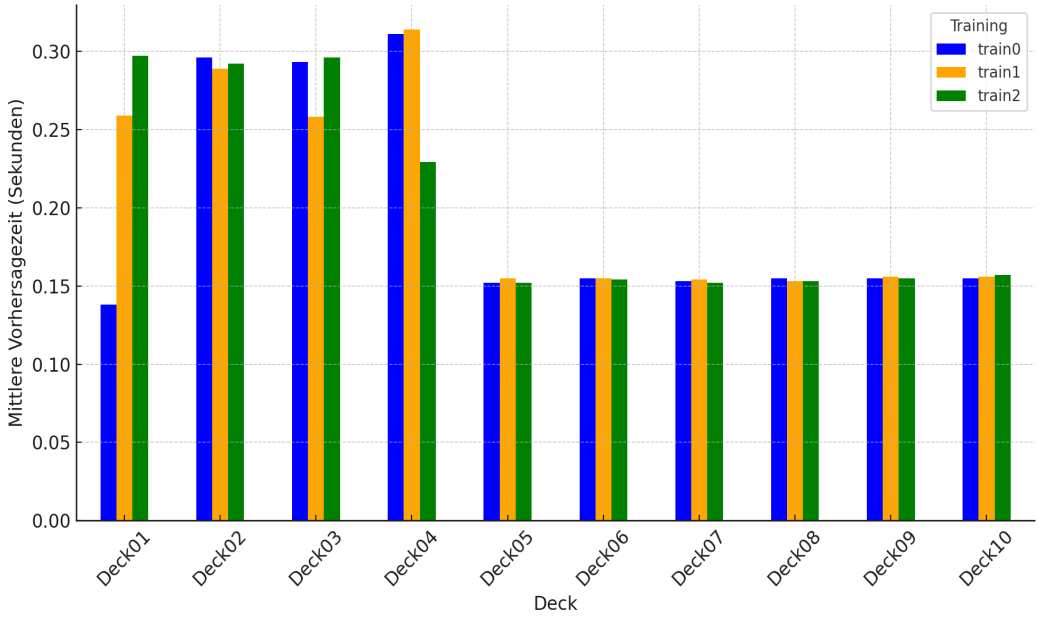


Abbildung 42: U-NET Mittlere Segmentierungszeit Werte für jedes Deck und Training

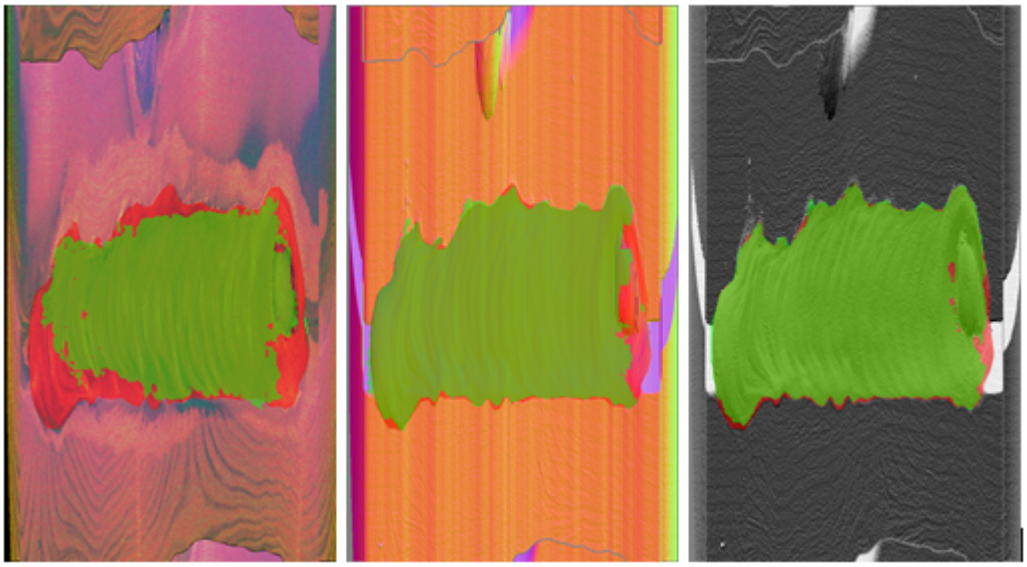


Abbildung 43: Beispiele aus U-Net Ergebnisse

5.3 YOLOv8

Die Daten in Abbildung 44 verdeutlichen die Steigerung der IoU-Werte bei der Verwendung von erweiterten und augmentierten Bildkonfigurationen, besonders ersichtlich in Deck05, Deck06, Deck09 und Deck10. Die Augmentation scheint in Kombination mit den erweiterten Daten die Leistung weiter zu steigern. Dies steht im Kontrast zu Layer-Images von Deck01 und Deck02, deren niedrigere IoU-Werte darauf hinweisen, dass dieser Bilder-Typ bei YOLOv8 zu Genauigkeitseinbußen führt.

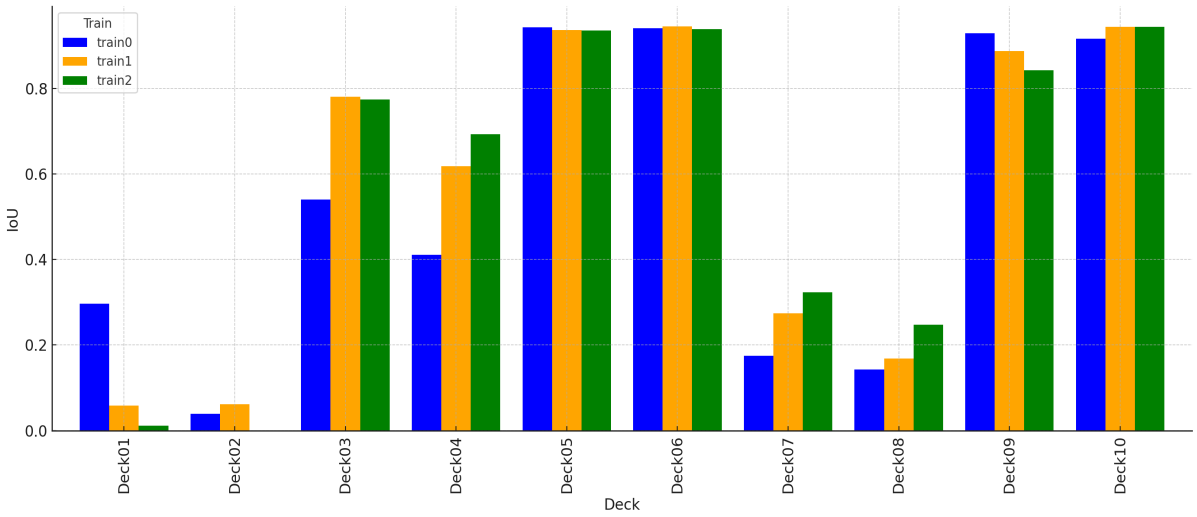


Abbildung 44: YOLOv8 IoU Werte für jedes Deck und Training

Im Hinblick auf die AP-Werte werden ähnliche Trends beobachtet 45.

Gemäß Abbildung 46 bleiben die Segmentierungszeiten unabhängig von Bildkonfigurationen oder der Verwendung von augmentierten oder erweiterten Daten stets stabil. Jedoch zeigt sich, wie erwartet, eine Abhängigkeit von dem während des Trainings verwendeten vortrainierten Modell. Das Training mit dem vortrainierten Modell yolov8n-seg.pt erreicht die schnellste Segmentierung, während yolov8m-seg.pt mittlere Geschwindigkeit aufweist und yolov8x-seg.pt am langsamsten ist. Trotzdem überschreitet die Segmentierungszeit aller Modelle nicht 600 Millisekunden.

Die folgende Abbildung 47 zeigt die Segmentierung desselben Bildes aus Deck6 und Deck10 mit verschiedenen Modellen von YOLOv8. Links wurde ein Modell verwendet, das mit yolov8n-seg.pt trainiert wurde (train0), in der Mitte mit yolov8m-seg.pt (train1) und ganz rechts mit yolov8x-seg.pt (train2). Dabei repräsentiert die grüne Farbe den Segmentierungsbereich und die rote Farbe den Ground-Truth-Bereich. Trotz des Wechsels der vortrainierten Modelle zeigte sich keine deutliche Verbesserung in der Genauigkeit der trainierten Modelle.

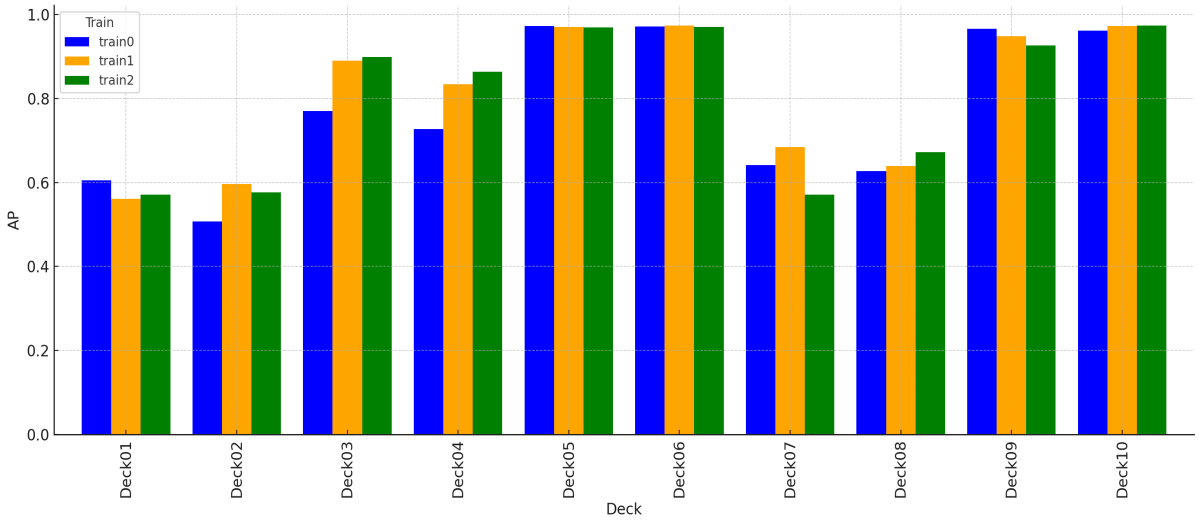


Abbildung 45: YOLOv8 AP Werte für jedes Deck und Training

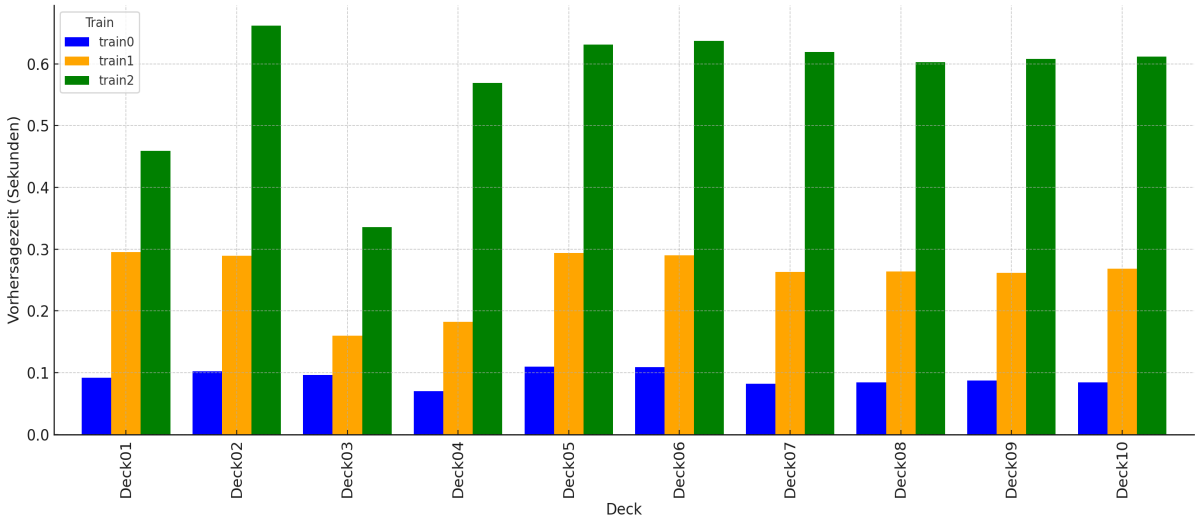


Abbildung 46: YOLOv8 mittlere Segmentierungszeit Werte für jedes Deck und Training

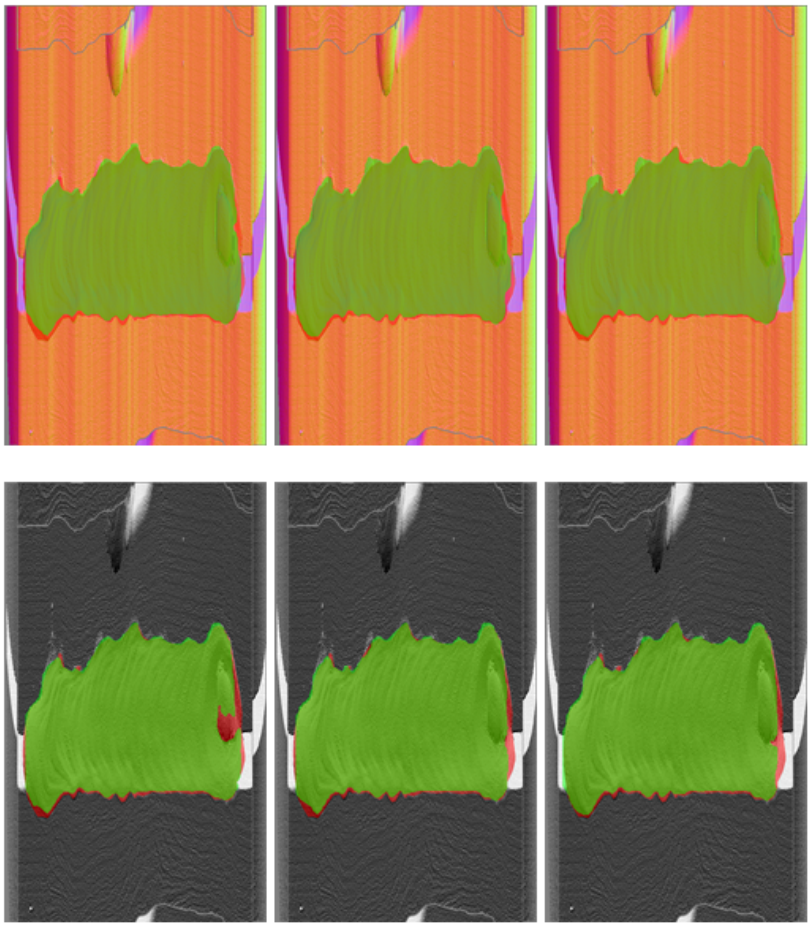


Abbildung 47: Beispiele aus YOLOv8 Ergebnisse

6 Schlussbetrachtung

6.1 Zusammenfassung der wichtigsten Ergebnisse

Die Evaluierung der Modelle liefert überzeugende Hinweise auf die kritische Rolle der Datenqualität und -vielfalt für die Segmentierungs- und Erkennungsleistung. Eine signifikante Verbesserung der *Intersection over Union* (IoU)-Werte durch Augmentieren und Erweiterung verdeutlicht, dass eine reichhaltige und qualitativ hochwertige Datengrundlage wichtig ist, um die Leistungsfähigkeit des Modells zu optimieren.

Wir konnten feststellen, dass Mask R-CNN die Nähte gut erkennen und eine durchschnittliche Genauigkeit von 91% gemessen in IoU erreicht werden kann, jedoch werden die Nähte Details (Konturen) nicht genau segmentiert. Darüber hinaus benötigt es, unabhängig von den für das Training verwendeten Daten oder Konfiguration, für die Segmentierung auf der CPU im Durchschnitt 2,5 Sekunden.

Zu U-Net lässt sich sagen, dass es ein leistungsfähiges Werkzeug für die Bildsegmentierung darstellt und eine präzise, pixelgenaue Klassifizierung bietet. Wir konnten damit eine durchschnittliche Genauigkeit von 93% gemessen in IoU erreichen. Besonders vorteilhaft ist es für Anwendungen, die eine detaillierte Segmentierung innerhalb von Bildern erfordern. Des Weiteren zeigt sich, dass U-Net trotz der Einführung komplexerer Datenstrukturen durch Augmentieren und Erweiterungen effizient skaliert. Dies deutet darauf hin, dass Verbesserungen der Modellgenauigkeit nicht zwangsläufig mit Einbußen bei der Segmentierungszeit einhergehen müssen. U-Net ermöglicht auf der CPU eine Segmentierung mit einer durchschnittlichen Zeit von 200 Millisekunden.

Die Beobachtungen zu den Ergebnissen von YOLOv8 deuten darauf hin, dass erweiterte und augmentierte Bildkonfigurationen die Genauigkeit verbessern können. Dabei spielt die Wahl des vortrainierten Modells eine wesentliche Rolle für die Segmentierungszeit. Beim Vergleich der Segmentierungsgenauigkeit unterschiedlicher YOLOv8-Modelle konnte trotz des Wechsels zwischen verschiedenen vortrainierten Modellen keine signifikante Verbesserung der Genauigkeit festgestellt werden. Zusammenfassend lässt sich feststellen, dass Schweißnähte unabhängig vom verwendeten vortrainierten Modell mit einer Genauigkeit von 94% erkannt werden können. Die optimale Segmentierungszeit von durchschnittlich 92 Millisekunden wird jedoch ausschließlich mit dem vortrainierten Modell yolov8n-seg.pt erreicht.

6.2 Beurteilung des Gesamterfolgs

Die Analyse der Leistung von Mask R-CNN, U-Net und YOLOv8 bei der Segmentierung von Schweißnähten offenbart verschiedene Stärken und Schwächen der einzelnen Modelle.

1. Alle drei Modelle sind in der Lage, Schweißnähte korrekt zu erkennen. Jedoch zeigt sich, dass Mask R-CNN keine Konkurrenz für U-Net und YOLOv8 darstellt, insbesondere hinsichtlich Genauigkeit und Segmentierungszeit.

2. YOLOv8, insbesondere trainiert mit dem vortrainierten Modell yolov8n-seg.pt, erzielt die beste Segmentierungszeit unter allen drei Modellen. Allerdings variiert die Genauigkeit nicht stark von U-Net.

Diese Bewertung des Gesamterfolgs verdeutlicht die jeweiligen Vor- und Nachteile der Modelle in Bezug auf die Segmentierung von Schweißnähten und bildet eine wichtige Grundlage für die Entscheidung über die optimale Auswahl eines Modells für spezifische Anwendungsfälle.

6.3 Ausblick auf zukünftige Arbeiten

Es besteht Potenzial zur Optimierung der Modelle YOLOv8 und U-Net, sowohl auf der Ebene der Modellarchitektur als auch in Bezug auf die Datensätze. Zukünftige Forschungsarbeiten könnten darauf abzielen, die Effizienz des Modells zu steigern, ohne Kompromisse bei der Leistung eingehen zu müssen.

Die Stabilität der Segmentierungszeiten bei U-Net und YOLOv8 über verschiedene Bildkonfigurationen hinweg deutet auf ihre robuste Leistungsfähigkeit hin, unabhängig von der Datenkomplexität. Dies spricht für ihre breite Anwendbarkeit in einer Vielzahl von Szenarien, was sie zu vielversprechenden Kandidaten für die Schweißnahtsegmentierung macht. Zukünftig könnten wir diese Modelle mit mehr Daten und Variationen trainieren, um zu prüfen, ob die Ergebnisse weiter verbessert werden können und ob die Segmentierungszeiten im Bereich von 100 bis 200 Millisekunden bleiben.

Literatur

[ADAA16] H. Al-Dmour and A. Al-Ani. A steganography embedding method based on edge identification and xor coding. *Expert systems with Applications*, 46:293–306, 2016.

[Ana23] Anaconda. Anaconda software distribution. https://docs.anaconda.com, 2023.

[ASdS22] Laura Antonelli, Valentina De Simone, and Daniela di Serafino. A view of computational models for image segmentation. *ANNALI DELL'UNIVERSITA' DI FERRARA*, 68:278–279, 09 2022.

[aSNS10] Al amri SS, Kalyankar NV, and Khamitkar SD. Image segmentation by using threshold techniques. *J Comput*, 2(2), 2010.

[Cha24] Yushuo Chang. Waste detection based on mask r-cnn. Master's thesis, Politecnico di Torino, Turin, Italy, 2024.

[Fro19] Nicola Fronzetti. Predictive neural network applications for insurance processes. Bachelor's degree internship report, Politecnico di Torino, Department of Control and Computer Engineering, 09 2019.

[Gir15] Ross Girshick. Fast r-cnn, 2015.

[HGDG18] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[JCQ23a] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8. https://github.com/ultralytics/ultralytics, 2023. Online; Zuletzt zugegriffen am 28.04.2024.

[JCQ23b] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Yolov8: Instance segmentation. https://docs.ultralytics.com/tasks/segment/ Online; Zuletzt zugegriffen am 28.04.2024.

[Jor21] Jeremy Jordan. Evaluating image segmentation models. https://www.jeremyjordan.me/evaluating-image-segmentation-models, 2021. Online; Zuletzt zugegriffen am 28.04.2024.

[LDG+17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017.

[MSMB13] Amrita Mohanty, Rajkumar Soundrapandiyan, Zameer Mir, and Puja Bardhan. Analysis of color images using cluster based segmentation techniques. *International Journal of Computer Applications*, 79:42–47, 10 2013.

[Neu24] Izaak Neutelings. Neural networks. https://tikz.net/neural_networks, 2024. Online; Zuletzt zugegriffen am 28.04.2024.

[NKY15] Dhanachandra N, Manglem K, and Chanu YJ. Image segmentation using k-means clustering algorithm and subtractive clustering algorithm. In *Eleventh international multi-conference on information processing-2015 (IMCIP-2015)*, 2015.

[NVF17] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. *NVIDIA GeForce RTX™ 2070 User Guide*, 2017. Online; Zuletzt zugegriffen am 28.04.2024.

[ON15] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *ArXiv e-prints*, pages 2–3, 11 2015.

[Pet17] Anne Petersen. Neuronale netze. `https://user.phil.hhu.de/~petersen/SoSe17_Teamprojekt/AR/neuronalenetze.html`, 2017. Online; Zuletzt zugegriffen am 28.04.2024.

[PNdS20] Rafael Padilla, Sergio L. Netto, , and Eduardo A. B. da Silva. A survey on performance metrics for object-detection algorithms. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242, 2020.

[PR13] Purohit P and Joshi R. A new efficient approach towards k-means clustering algorithm. *Int J Comput Appl*, 65(11), 2013.

[Pyk23] Kurtis Pykes. Keras vs tensorflow vs pytorch. https://www.datacamp.com/tutorial/pytorch-vs-tensorflow-vs-keras, 2023. Online; Zuletzt zugegriffen am 28.04.2024.

[Pyt24] Python. Python – the language of today and tomorrow. https://pythoninstitute.org/about-python, 2024. Online; Zuletzt zugegriffen am 28.04.2024.

[RDG+16] Joseph Redmon, Santosh Divvala, Ross Girshick, , and Ali Farhadi. You only look once: Unified, real-time object detection. https://arxiv.org/pdf/1506.02640.pdf, 2016.

[RFB15] O. Ronneberger, P. Fischer, , and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *ArXiv e-prints*, 2015.

[RHGS17] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *IEEE Transactions on Pattern Analysis & Machine Intelligence*, pages 1137–1149, 2017.

[RTG+19] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 658–666, Los Alamitos, CA, USA, jun 2019. IEEE Computer Society.

[Sei18] Stephan Seidel. Segmentierung von schweißnähten mittels selbstlernender algorithmen. Masterarbeit, Mathematik und Naturwissenschaften, Hochschule Darmstadt, 4 2018. Matr. Nr. 727969; Prüfer: Prof. Dr. Andreas Weinmann, Prof. Dr. Thomas Netzsch.

[SHG19] Mousavirad SJ, Ebrahimpour-Komleh H, and Schaefer G. Effective image clustering based on human mental search. *Appl Soft Comput J*, pages 209–220, 2019.

[Sin] Aarohi Singla. Mask-r-cnn-using-tensorflow2. https://github.com/AarohiSingla/Mask-R-CNN-using-Tensorflow2. Online; Zuletzt zugegriffen am 28.04.2024.

[SKSA10] Singh, K. K., Singh, and A. A study of image segmentation algorithms for different types of images. *International Journal of Computer Science Issues*, 7(5):1–2, 2010.

[SMB10] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis, editors, *Artificial Neural Networks – ICANN 2010*, pages 92–101. Springer Berlin Heidelberg, 2010.

[Tea23] WSI Team. Betriebsanleitung virowsi: Optische 3d-nahtprüfung. Version 3.8, 9 2023. Verfügbar bei Vitronic.

[Wad16] Kentaro Wada. labelme: Image polygonal annotation with python. https://github.com/labelmeai/labelme, 2016. Online; Zuletzt zugegriffen am 28.04.2024.

[Wik14] Wikipedia. Thresholding (image processing). `http://en.wikipedia.org/w/index.php?title=Thresholding_(image_processing)&oldid=606970852`, 2014. Online; Zuletzt zugegriffen am 28.04.2024.

[Yam24] Badr El Yamouni. Evaluation of segmentation methods based on neural networks for the detection of weld seam limits. `https://github.com/BadrElya/Schweissnaht_Segmentierung`, 2024. Online; Zuletzt zugegriffen am 28.04.2024.

[YP22] Riya Yadav and Manish Pandey. *Image Segmentation Techniques: A Survey*, pages 166–168. Springer Singapore, 01 2022.

[Zen] Catch Zeng. tensorflow-unet-labelme. https://github.com/CatchZeng/tensorflow-unet-labelme. Online; Zuletzt zugegriffen am 28.04.2024.