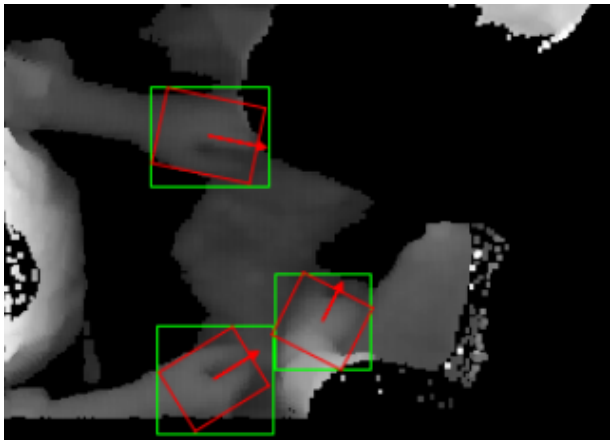
Universität Bremen

Handerkennung und Bestimmung ihrer Orientierung in Tiefenbildaufnahmen im Bereich der Chirurgie



Bachelorarbeit zur Erlangung des Akademischen Grades
„Bachelor of Science“ (B.Sc.)

Clement Phung

Matrikelnummer:	4447757
Erster Gutachter:	Prof. Dr. Zachmann
Zweiter Gutachter:	Prof. Dr. Maneth
Abgabedatum:	20.10.2020

E R K L Ä R U N G

Ich versichere, den Bachelor-Report oder den von mir zu verantwortenden Teil einer Gruppenarbeit*) ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

*) Bei einer Gruppenarbeit muss die individuelle Leistung deutlich abgrenzbar und bewertbar sein und den Anforderungen entsprechen.

Bremen, den _____

_____
(Unterschrift)

Inhaltsverzeichnis

1. Einleitung

Moderne Operationssäle sind mit Operationsleuchten ausgestattet, welche den Chirurgen bei der Operation durch das Ausleuchten des Operationsgebiet, dem sogenannten Situs, unterstützen. 40000 Lux (lx) ist eine optimale Beleuchtungsintensität des Situs im Bereich der Bauchchirurgie [16]. Um diese Lichtintensität zu erreichen, werden in der Regel die Operationslampen oberhalb des Situs manuell von dem OP-Personal der Operation ausgerichtet. Dies führt zu kleineren Unterbrechungen der Operation und verlängert nicht nur die gesamte Operationsdauer, sondern stört auch die Konzentration des Personals.

Als Lösungsansatz entwickelten Teuber et al. 2015 einen Algorithmus zur kontinuierlichen Berechnung der optimalen Ausrichtung von OP-Lampen [38]. Dabei wird eine Tiefenbildkamera des Typs Microsoft Kinect v2 verwendet, welche die Eingabe in Form von Tiefenbildern für den Algorithmus generiert. In Abb. 1 ist exemplarisch eine Aufahme der Tiefenbildkamera während einer Operation zu sehen. Eine Umsetzung des Lösungsansatzes birgt in der Realität jedoch zwei neue Probleme. Zum einen ist das Ausrichten der Lampen nicht zu jedem Zeitpunkt sinnvoll. Durch eine Änderung der Positionen der Lampen, ändert sich auch die Lage des Schattens im Situs. Dies hat einen direkten Einfluss auf die Tiefenwahrnehmung der Chirurgen, welche in empfindlichen Phase der OP nicht gestört werden sollte [21]. Zum anderen ist der Fokus des Chirurgen nicht zwingend an einer festen Position gebunden. Die autonomen OP-Lampen müssen sich dementsprechend anpassen und den neuen Fokus des Chirurgen anvisieren.

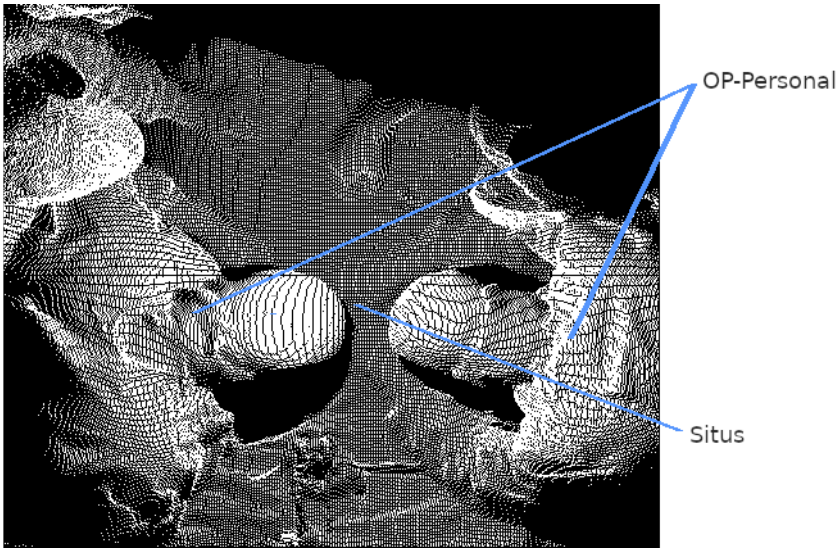


Abbildung 1: Tiefenaufnahme in Form einer Punktwolke aus der oberen Perspektive. In der Mitte befindet sich der Situs, rechts und links ist das OP-Personal zu sehen.

Als Lösung für diese neuen Probleme wird die Erkennung der Hände in den Tiefenbildern und die Bestimmung ihrer horizontalen Ausrichtung vorgeschlagen. Über diese zusätzlichen Informationen der Hände sind Rückschlüsse über den Zeitraum aussagbar, in welcher der Chirurg konzentriert am Situs arbeitet. Die Wahrscheinlichkeit liegt höher, dass sich der Chirurg in einer konzentrierten Phase befindet, wenn sich die Hände in der unmittelbaren Nähe des Situs befinden. Für das zweite Problem kann nur eine Teillösung vorgeschlagen werden, da keine Daten über die tatsächlichen Fokuspunkte des Personals vorliegen. Aus den erkannten Positions- und Ausrichtungsdaten kann der Punkt direkt vor der Hand als möglicher Fokus des Operierenden bestimmt werden. Ohne einer Evaluation mit realen Daten kann nur eine mögliche Realisierung, wie in Abschnitt 6.2 dargestellt, vorgeschlagen werden.

Diese Arbeit setzt sich mit einer Umsetzung dieser (Teil-)Lösung im Bezug zu den oben genannten Problemen auseinander.

1.1. Ziel der Arbeit

Ziel dieser Arbeit ist es, die bisher aufgenommenen Tiefenbildvideos der Operation zu verwenden, um damit ein *Machine Learning Modell* zu entwickeln, welches Hände in Tiefenbildern im Bereich der Chirurgie erkennt. Zusätzlich soll zu jeder erkannten Hand die Ausrichtung prognostiziert werden.

Dieser Vorgang setzt sich aus mehreren Schritten zusammen. Üblicherweise wird im Bereich des maschinellen Lernens aus den Videodaten eine angemessene Untermenge von Bildern extrahiert, die als Trainingsdaten verwendet werden. Im Anschluss folgt ein Datenvorverarbeitungsprozess, welcher die Bilder in einen trainingsbereiten Zustand versetzt. Der nächste Schritt ist das Definieren von Detektionsmodellen, bei dem der Aufbau der Modelle festgelegt wird. Unter Verwendung der Trainingsdaten wird das Modell trainiert und im Nachhinein evaluiert. Dieser Schritt wird für andere Konfigurationen und Modelle wiederholt, bis ein Modell unter einer bestimmten Konfiguration sich für die Situation eignet.

1.2. Anforderung

Das resultierende Modell soll keinen genauen Vorgaben unterliegen, dennoch sind folgende Anforderungen an das Modell gestellt:

- **Nur Tiefendaten werden verwendet**: Aufgrund der hellen Beleuchtung durch die OP-Lampen ist der Bereich in der Nähe des Situs sehr stark ausgeleuchtet und

auf den RGB-Aufnahmen kaum erkennbar. Wichtige Merkmale für die Erkennung gehen verloren und bieten daher keine Verwendung. Daher wird die Erkennung der Hände und die Bestimmung ihrer Ausrichtung nur auf den Tiefendaten durchgeführt. Näheres wird in Abschnitt 3.1.2 erläutert.

- **Ausreichende Genauigkeit**: Die Vorhersage des Modells soll möglichst genau der Realität entsprechen. Die Bestimmung der Ausrichtung soll der tatsächlichen Ausrichtung so gering wie möglich abweichen. Dennoch ist in unserem Fall eine Abweichung der Ausrichtung von etwa $\leq 30°$ noch ausreichend.

- **Kompatibilität mit C++**: Das Programm für den Algorithmus der automatischen Ausrichtung der OP-Lampen ist in der Programmiersprache C++ implementiert. Das resultierende Modell soll als Komponente in dem Programm die Berechnung erweitern und muss dementsprechend eine Schnittstelle zu C++ bereitstellen. Die Verwendung einer bestimmten Programmiersprache ist hingegen bei dem Definieren und Trainieren eines Modelles nicht vorgegeben.

- **Vorhersage in Echtzeit**: Die Frequenz der Tiefenbildkamera beträgt 30 Hertz (hz). Die benötigte Zeit für das Treffen der Vorhersage, auch Inferenz genannt, muss die Zeit zwischen zwei Bildern unterbieten. Im Falle einer Anwendung des Modells im Programm ASuLa kommt hinzu, dass eine Vorhersagedauer des Modells den Verzug bis zur tatsächlichen Positionierung der Lampen beeinflusst. Die Ausführungszeit sollte im besten Fall nicht bemerkbar verlangsamt werden.

1.3. Aufbau der Arbeit

Das folgende Kapitel 2 gibt einen Überblick über den aktuellen Stand im Bereich der Erkennung von Händen und die Bestimmung der Ausrichtung von Objekten. Ebenfalls werden die bestehenden Ansätze auf die Kompatibilität zu den aufgestellten Anforderungen geprüft. In dem anschließenden Kapitel 3 werden die Grundlagen zum Verständnis der Situation dargelegt. Dazu zählt neben der Funktionsweise von neuronalen Netzen auch der Aufbau der vorliegenden Daten. Danach folgt in Kapitel 4 mit der Implementierung des Konzepts, welche darauffolgend im Abschnitt 5 ausgewertet wird. Zum Schluss werden im Fazit und Ausblick (Abschnitt 6) die Erkenntnisse dargestellt und auf bestehenden Forschungsbedarf aufmerksam gemacht.

2. Stand der Technik

Das Problem dieser Arbeit lässt sich in zwei Themengebiete aufteilen. Zum einen die Erkennung von Händen und zum anderen die Bestimmung der Ausrichtung. Diese werden in der Regel auch in der Literatur unabhängig voneinander betrachtet.

2.1. Handerkennung in Aufnahmen

Die Technik der Handerkennung im *Computer Vision* Bereich ist ein weit verbreitetes Thema, denn diese gewinnt durch mögliche Einsatzgebiete der Gestensteuerung im Bezug zur Mensch-Technik-Interaktion immer mehr an Bedeutung.

Die Problemstellung beinhaltet, dass trotz der zur Verfügung stehenden RGB-Daten, eine Erkennung der Hände nur auf den Tiefendaten möglich ist. Das liegt, wie in Kapitel 3.1.2 beschrieben, an der Reflektion der OP-Lampen. Im Allgemeinen bleibt das Problem unabhängig der Nutzung der RGB-Daten dieselbe. Anders als bei RGB-Aufnahmen wird bei den Tiefendaten nur ein Farbkanal verwendet. Demzufolge entfallen mögliche Ansätze, die auf die Farbunterschiede der Hand beruhen.

Häufig werden in modernen Ansätzen eine spezielle Variante der *Convolutional Neural Networks* (CNN, Näheres wird in Kapitel 3.2.2 erläutert.) verwendet [30, 6, 24]. Trotz der vergleichsweise hohen Genauigkeit der vorgeschlagenen Modelle, wie beispielsweise dem *Multiple Scale Region-based Fully Convolutional Networks* (MS-RFCN, Average Precision: 75,1%) [24], sind die meisten Modelle nicht in der Lage die Vorhersagen in Echtzeit (30 *hz*) zu treffen. Dies liegt vor allem an dem aufwendigen Prozess in der Spekulation über die Positionen eines möglichen Objektes.

Durch den breiten Anwendungsbereich der *Hand Pose Estimation*, der Bestimmung einer Handhaltung, sind einige Methoden zur Handerkennung in den Tiefendaten vorgestellt worden. Ähnlich zu den Methoden basierend auf Segmentierung in RGB-Bildern werden häufig Tiefenunterschiede zwischen den Händen und dem Hintergrund ausgenutzt [31, 43]. Das Segmentieren wird in einigen Fällen mit dem Tragen von schwarzen Armbändern unterstützt, die beim Segmentieren der Hände hilft [18, 33]. In diesem Fall sind die Hände größtenteils sehr nah vor dem Hintergrund, sodass diese Ansätze nicht vollständig zur Geltung kommen könnten. Des Weiteren würde das Tragen von speziellen Armbändern das OP-Personal bei der Arbeit einschränken und stellt daher keinen guten Ansatz für das Problem dar.

In Echtzeiterkennungssystemen ist die Nutzung von *Random-Decision-Forests* der bevorzugte Weg [19, 31]. Diese sind im Vergleich zu CNNs deutlich schneller in der Ausführung, hängen aber in der Genauigkeit den CNNs hinterher [19].

Dies ändert sich mit der Vorstellung effizienteren Objektdetektoren basierend auf CNNs, welche bei nahezu gleicher Genauigkeit nur ein Bruchteil der Zeit benötigen. Ähnlich zu unserer Thematik war das Ziel in der Arbeit von Yang et al. ebenfalls Hände auf RGB-Aufnahmen zu finden und eine Ausrichtung vorzuschlagen [44]. 2018 wurde von den Autoren ein Modell auf der Basis eines *Single Shot Detectors* (SSD) [27] mit einem *MobileNet-Feature-Extractor* [12] definiert, welche eine AP von $83,2\%$ erreicht. Dabei wurde mit zugänglicher Hardware (Nvidia Titan X) eine Laufzeit von nur $7,2ms$ pro Erkennung benötigt. Mit dieser niedrigen Laufzeit ist in diesem Fall eine Umsetzung unter Echtzeitanforderung realistisch.

2.2. Bestimmung der Ausrichtung

In der Bestimmung der Ausrichtung ist keine gängige Methode verbreitet. Einige Untersuchungen haben ergeben, dass eine Vorhersage einer direkten Repräsentation des Winkels zu mangelhaften Ergebnissen führen [44]. Im Zuge dessen sehen die vorgeschlagenen Methoden vielfältig aus, sind unterschiedlich komplex umzusetzen und bieten verschiedene Vor- und Nachteile.

Einen empfehlenswerten Vorschlag stellen Yang et al. vor [44]. Die Methode basiert auf einer Regression von Längen der Vektoren einer Vektordarstellung. Hände in diesem System werden wie in Abb. 2 sichtbar durch zwei orthogonale Vektoren dargestellt.

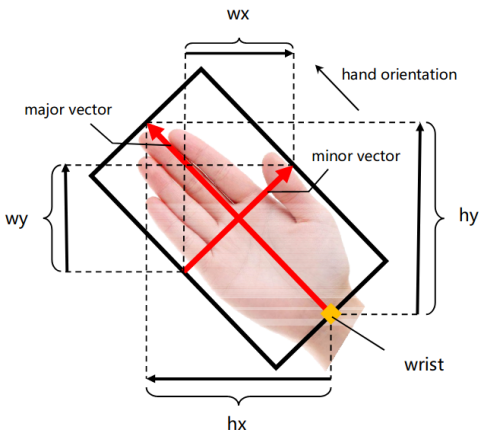


Abbildung 2: Darstellung des Vektoren basierten Systems zur Darstellung von Händen [44]. Die vier Größen wx, wy, hx und hy können auch negative Werte annehmen und decken somit alle möglichen Richtungen ab.

Der Hauptvektor (*major vector*) liegt entlang des Handgelenks in Richtung der Fingerspitzen und der Nebenvektor (*minor vector*) steht dazu senkrecht in Richtung des Daumens. Die Vektoren werden vorhergesagt, indem die horizontalen Komponenten hx und wx, beziehungsweise die vertikalen Komponenten hy und wy des Vektors vorhergesagt und paarweise miteinander addiert werden. Der Schnittpunkt der Vektoren gleicht dem Mittelpunkt der detektierten Box. Die Methode beweist sich in den Ergebnissen der Arbeit als zuverlässig. $63,36\%$ der vorhergesagten Richtungen werden mit einer Abweichung von $\leq 10°$ (analog $85,18\%$ bei $\leq 20°$ und $90,23\%$ bei $\leq 30°$) zu den Referenzdaten vorhergesagt.

In dem Vorgehen, auf Satellitenbilder Schiffe zu erfassen und deren Ausrichtung zu bestimmen, wird in einer anderen Arbeit ebenfalls mithilfe der Regression eine *Oriented Bounding Box* (OBB, Definition in Kapitel 4.2.3) vorhergesagt [45]. Mit dieser Darstellung alleine ist die Ausrichtung des Objektes nicht eindeutig, da jede der vier Seiten der OBB in die Ausrichtung des Objektes zeigen könnte. Die Kanten werden aufsteigend gegen den Uhrzeigersinn gelabelt und verwandeln in dieser Weise die Bestimmung der Orientierung in eine Klassifikation zwischen den vier Kanten. Diese Methode konkateniert mehrere neuronale Netze, sodass eine Erkennung in Echtzeit nicht möglich erscheint.

Der Ansatz, die Seite zu klassifizieren, weckt den Gedanken, die Bestimmung der Orientierung nicht separat in einem nachfolgendem Schritt vorzunehmen, sondern gleich mit dem Erkennungsprozess zu verbinden. Ähnlich zu einem Kompass wird die Ausrichtung der Hand in einzelne Klassen, die eine bestimmte Spanne einer Ausrichtung darstellen, aufgeteilt. Diese Methode ist immer mit einer kleinen Abweichung behaftet. Doch in diesem Anwendungsfall ist das kein Ausschlusskriterium, denn eine Abweichung bei der Bestimmung der Ausrichtung ist zu einem gewissen Grad akzeptabel. In Abb. 3 ist zu sehen, wie die Methode im Bezug zu diesem Problem aussieht.

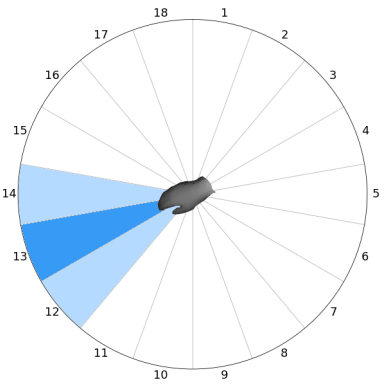


Abbildung 3: Bestimmung der Ausrichtung durch Klassifikation. Neben der *Groundtruth* (dunkelblau) liegt die akzeptable Abweichung (hellblau).

Einen ähnlichen Ansatz beschreiben Hara et al. in ihrer Arbeit, in der die Richtung von gedrehten Objekten bestimmt werden [10]. Die Spanne, die eine Klasse abdecken soll, muss dabei geringer als die maximal akzeptierte Abweichung sein. Andernfalls ist eine Vorhersage in allen Fällen zu ungenau. In Abschnitt 1.1 wird die maximal akzeptierte Abweichung auf $\leq 30°$ festgelegt. Daher ist es sinnvoll, den Definitionsbereich in 20° große Abschnitte (somit 18 Klassen) aufzuteilen, unter der Annahme, dass sich eine vorhergesagte Ausrichtung in der Mitte einer Spanne befindet. So ist auch eine Missklassifikation von einer nebenliegenden Klasse noch im hinnehmbaren Bereich. Dank der Existenz schneller Architekturen der CNNs stellt die Echtzeitfähigkeit kein Hindernis für den Ansatz der Klassifikation dar, wie es zum Beispiel im Modell basierend auf Regression der Vektoren gezeigt ist.

2.3. Eigener Lösungsansatz

Bei der Detektion von Händen fällt die Wahl auf die Verwendung von CNNs. Diese bieten durch Veröffentlichungen schneller Architekturen sowohl eine hohe Genauigkeit in der Detektion, als auch die Möglichkeit der Echtzeiterkennunng.

In der Bestimmung der Ausrichtung ist weder das Modell basierend auf Regression der Vektoren [44], als auch das Modell basierend auf der Klassifikation der Ausrichtung (nach Abb. 3) die bessere Methode. Daher werden in dieser Arbeit beide Methoden umgesetzt. Im Anschluss werden beide Methoden evaluiert und in Betracht der aufgestellten Anforderungen miteinander verglichen. Im weiteren Verlauf dieser Arbeit wird das auf Regression basierende Modell auch Regressionsmodell genannt. Analog dazu wird das andere Modell Klassifikationsmodell genannt. Bevor die Modelle jedoch definiert und trainiert werden, müssen zu aller erst die Trainingsdaten vorverarbeitet werden.

3. Grundlagen

In diesem Kapitel wird der Fokus auf die Grundlagen des oben genannten Problems und dessen Lösung gelegt. Darunter fallen die Generierung und der Aufbau der vorliegenden Daten. Weiterhin deckt dieses Kapitel den fundamentalen Aufbau von *Convolutional Neural Networks* und der Architektur des *Single Shot Detectors* ab, welche für die Erkennung von Händen und die Bestimmung der Orientierung zuständig ist.

3.1. Datenerhebung und Aufbau der Daten

3.1.1. Microsoft Kinect v2

Für die Aufnahme der Tiefendaten wurde eine Microsoft Kinect v2 verwendet. Gegenüber anderen handelsüblichen Modellen sticht die Kinect v2 mit einer höheren Auflösung von 512 $px\times$ 424 px heraus. Die Kamera arbeitet mit dem Laufzeitverfahren, welches der Kamera ermöglicht, Distanzen zwischen der Kamera und Objekten im Bereich von $0,5\,m - 4,5\,m$ zu messen. Dazu stößt die, wie in Abb. 4 zu sehende Infrarotquelle, Infrarot-Blitze aus, welche von Objekten reflektiert werden. Individuell wird für jedes Pixel die Zeit aufgezeichnet, welche die Strahlen von der Quelle zurück bis zum Infrarotsensor benötigen. Intern wird die Dauer und somit die Distanz durch die Größe der Phasenverschiebung zwischen dem gesendeten und empfangenen wellenartigem Signal berechnet [9].

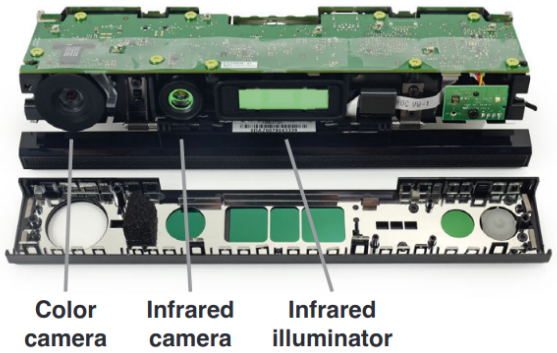
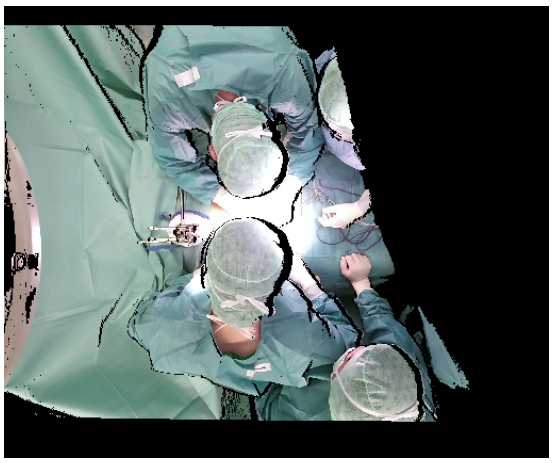


Abbildung 4: Aufbau der Kinect v2. Links sitzt die Farbkamera (Color camera) und rechts davon der Infrarotsensor (Infrared camera) und die Infrarotquelle (Infrared illuminator) [9].

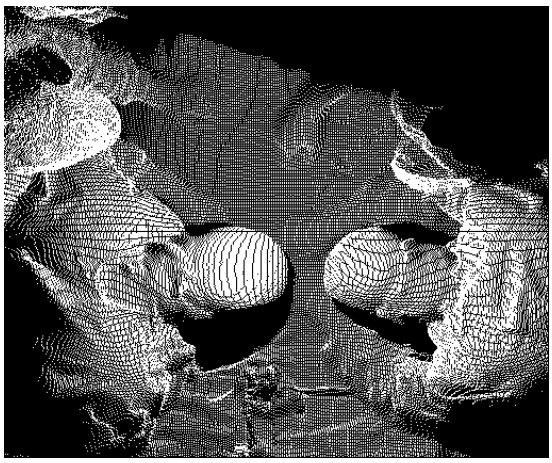
Simultan zu der Aufzeichnung der Tiefendaten nimmt die Farbkamera RGB-Daten bis zu einer Auflösung von $1920\,px \times 1080\,px$ auf. Beide Aufnahmetypen können nach einigen Nachbearbeitungsschritten kombiniert werden und resultieren in einer kolorierten Punktwolke der Auflösung $512\,px \times 424\,px$ [23]. Im März 2020 wurde der Nachfolger der Kinect v2, die Azure Kinect veröffentlicht. Diese ersetzt zwar das verwendete Modell im OP-Saal, benutzt aber das verwendete Format zur Aufzeichnung der vorliegenden Daten.

3.1.2. Vorliegende Daten

Für die Erstellung des Detektionsmodells stehen 23 Aufnahmen von einer durchgeführten Operationen bereit, welche öffentlich durch Teuber et al. bereit gestellt werden [39]. Alle Aufnahmen sind dabei aus einer Sicht, ähnlich einer Vogelperspektive, aufgenommen. Jede Aufnahme beinhaltet maximal 27000 Bilder, welche bei einer Bildrate von 30 fps zusammen ca. 15 Minuten ergeben. In der Aufnahmedatei sind die Aufnahmen in Form der kolorierten Punktwolke, wie in Abb. 5a zu sehen, in der Auflösung von $512\,px \times 424\,px$ gespeichert. Aus technischen Gründen des Programms ASuLa wird das Sichtfeld modifiziert. Nachdem ein einfacher Rauschfilter angewendet wird, wird das Bild um 90° gegen den Uhrzeigersinn gedreht und im Anschluss um die vertikale Achse gespiegelt. Des Weiteren wird in das Bild hineingezoomt, sodass überflüssige Informationen weit außerhalb des Situs ausgeschnitten werden. Zusätzlich werden die RGB-Daten der kolorierten Punktwolke entfernt. Das Ergebnis dieser Transformation ist in Abb. 5b abgebildet.



(a) Gefärbte Punktwolke einer Aufnahme



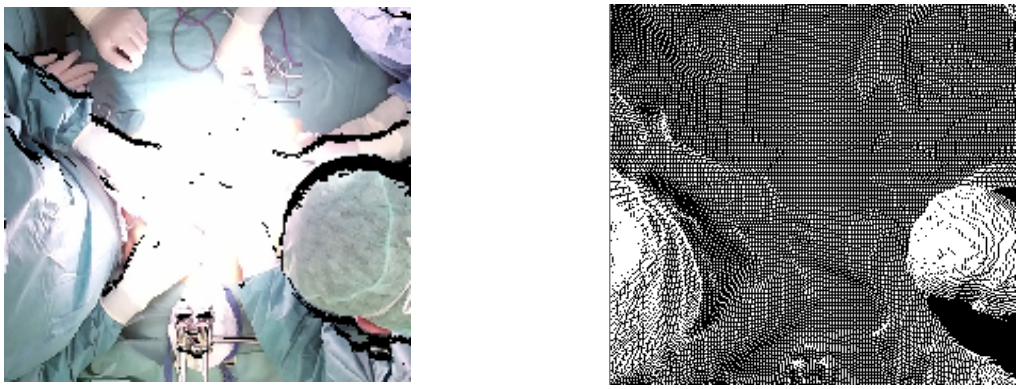
(b) Abbild der Punktwolke

Abbildung 5: Beide Bilder zeigen den gleichen Zeitpunkt. Rechts ist das Ergebnis der Transformation abgebildet.

In der Mitte der Aufnahme ist der Situs in der Bauchregion des Patienten zu sehen, welcher sich auf einer Höhe von ca. $1,1\,m$ befindet. Rechts und links neben dem Situs steht das Personal. Dieser spezifische Aufbau ändert sich in den Aufnahmen nicht und es ist für zukünftige Aufnahmen anzunehmen, dass dieser gleich bleibt.

Im Verlauf dieser Arbeit werden die Begriffe Tiefenbild und Abbild der Punktwolke der Einfachheit halber gleichgesetzt.

Die RGB-Aufnahmen der Operation bieten keinen Mehrwert für die Erkennung, denn durch die direkte Beleuchtung der OP-Lampen ist eine deutliche Überbelichtung im Bereich rund um den Situs zu erkennen. In der Abb. 6 ist die gleiche Situation zum Vergleich beider Modi dargestellt. In den Tiefendaten sind die Hände der Operierenden in der Nähe des Situs klar zu erkennen, während die gleichen Hände in den RGB-Daten kaum bis gar nicht von der Umgebung zu unterscheiden sind. Viele wichtige Features für die Erkennung gehen durch die Überbelichtung verloren.



(a) Überbelichtung in Farbaufnahmen

(b) Situsbereich im Tiefenbild

Abbildung 6: Vergleich der Sichtbarkeit der Hände im Situsbereich zwischen der colorierten Punktwolke und der Tiefenaufnahme.

Während der Operation kann es vorkommen, dass das Personal die Lampe in das Sichtfeld der Kamera verschiebt. Infolgedessen sind die Aufnahmen in dem Zeitraum teilweise oder vollständig abgedeckt, wie es in Abb. 7 der Fall ist. Etwa die Hälfte des Aufnahmematerials ist von einer Verdeckung des Sichtfeldes betroffen.

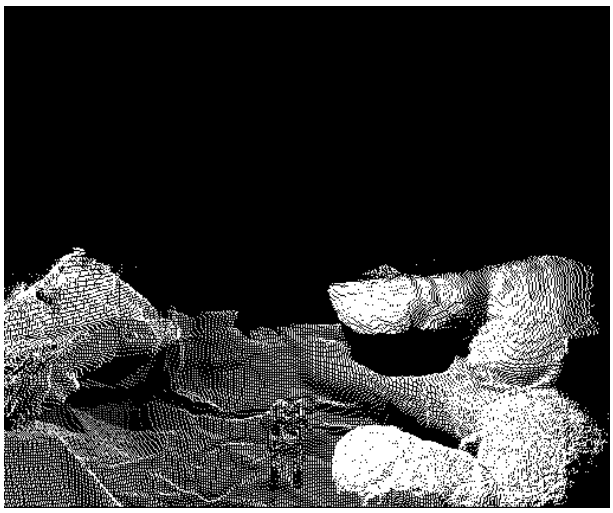


Abbildung 7: Starke Einschränkung des Sichtfeldes durch die OP-Beleuchtung.

3.2. Modell zur Erkennung der Hände

3.2.1. Künstliches Neuronales Netz

Ein künstliches Neuronales Netz (NN) ist ein Modell zum Lösen von Regressions- oder Klassifikationsaufgaben. Das Modell besteht aus künstlichen Neuronen, die gruppierte Schichten bilden. Diese werden, wie in Abb. 8 dargestellt, angereiht und miteinander verknüpft. Alle Verbindungen zwischen den Neuronen, außer die Verbindungen ausgehend der *input layer*, sind mit einem Gewicht w versehen. Beim Trainieren wird eine Eingabe x durch den *Input Layer*, die nachfolgenden *Hidden Layers* und dem *Output Layer* geleitet.

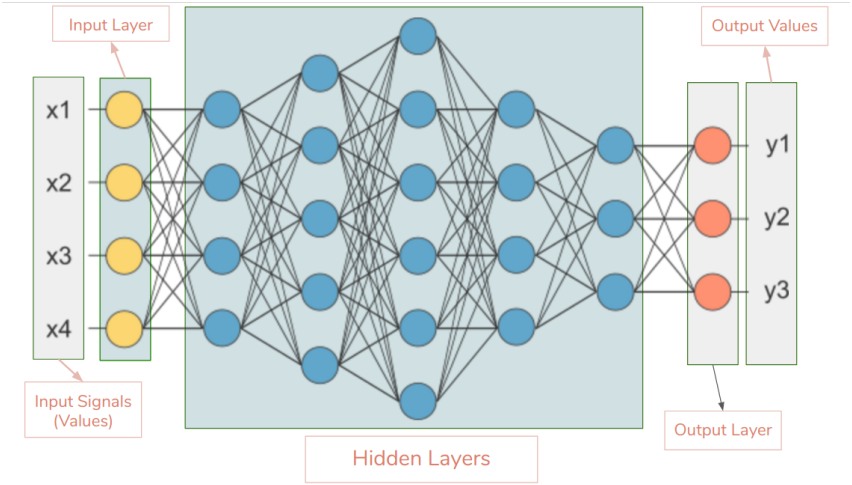


Abbildung 8: Aufbau eines Neuronalen Netzes mit fünf *Hidden Layers*. Vollständig vernetze Schichten werden auch *Fully Connected Layers* genannt [28].

Die Ausgabe jedes Neurons einer Schicht wird mit dem Gewicht der ausgehenden Verbindung multipliziert, miteinander summiert und dem neuen Neuron übergeben. Das Neuron ist mit einer Aktivierungsfunktion ausgestattet, welche nicht-Linearität in das Netz einführt. Mithilfe dieser Funktion wird bestimmt, welche Neuronen das Signal weiterleiten und somit wichtig für die Vorhersage sind. Jedes Neuron verfügt über einen *Bias*, um die Ausgaben in einen bestimmten Wertebereich zu verschieben. Das Durchleiten, auch *Forward Propagation* genannt, berechnet in dieser Weise eine Ausgabe y_{pred}. Ähnlich zu der linearen Regression wird im Anschluss aus der Vorhersage y_{pred} und der erwarteten Ausgabe y_{gt} ein Verlust L berechnet. Dieser Verlust stellt die Abweichung der Vorhersage dar und wird aus der festgelegten Verlustfunktion $Loss(y_{pred}, y_{gt})$ kalkuliert. Oft besteht die Eingabe aus einer Menge der Größe ≥ 2 Dateneinheiten, auch *Batch* genannt.

Zuletzt folgt mit der *Backpropagation* der Lernprozess des Netzes. Diese passt die Gewichte des NN schichtweise beginnend von dem *Output Layer* an. Das Ziel des Vorgangs

ist die Minimierung des *Losses*. Vergleichbar ist dies mit der Suche nach einem globalen Minimum einer Polynomfunktion, in welcher man den kompletten Verlauf der Kurve nicht sieht. Schrittweise bewegt man sich entlang des Gefälles (*Gradient Descent*) um ein Minimum zu erreichen. Die Richtung der Schritte wird durch die Ableitung der *Loss*-Funktion bestimmt. Die Gewichte werden nun mit einem Produkt aus der Ableitung des *Losses* und der *Learning Rate* subtrahiert. Die *Learning Rate* bestimmt, wie einflussreich die Aktualisierung der Gewichte ausfällt. In der Regel wird dieser mit steigender Anzahl an Trainingsvorgängen verringert um sich immer mehr einem Minimum zu nähern.

Eine besondere Lernstrategie ist die Nutzung von *Transfer Learning*. Diese Technik strebt eine Beschleunigung des Lernprozesses an, indem vorher trainiertes Wissen in einen neuen Trainingsprozess mit einem anderen Datensatz verwendet wird [29]. Im Bezug zu neuronalen Netzen kann *Transfer Learning* mit dem Initialisieren der Gewichte eines anderen Netzes umgesetzt werden [2]. Für die Objekterkennung bedeutet dies die Nutzung von Gewichten, welche vorher auf andere Bilder trainiert werden. Somit kann auf allgemeines Wissen in der Objekterkennung, wie die Erkennung von einfachen Strukturen wie Kanten oder ähnliches zurückgegriffen werden und muss nicht erneut trainiert werden.

3.2.2. Convolutional Neural Network

Als spezielle Variante der NN wurde das *Convolutional Neural Network* (CNN) veröffentlicht, welche Aufgaben im *Machine Vision* Bereich effektiv lösen kann. Einer der Hauptfeatures der CNNs gegenüber anderen künstlichen NN ist die Fähigkeit räumliche Korrelationen in den Daten zu erfassen und zu verwenden [20].

In Abb. 9 ist ein typischer Aufbau eines CNNs zu sehen. Die Architektur besteht aus einer Abwechslung aus *Convolutional* und *Pooling* Schichten. Am Ende der Architektur sind üblicherweise eine oder mehrere vollständig vernetzte Schichten angesetzt[20].

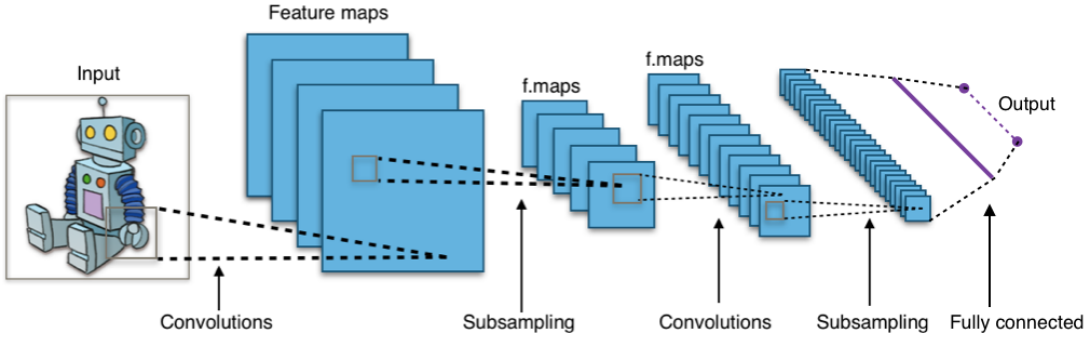


Abbildung 9: Allgemeiner Aufbau eines CNNs. Die *Feature-Maps* nehmen in der Regel immer eine kleinere Auflösung an [42].

Der *Convolutional Layer* ist aus einer Reihe von *Convolutional Kernels* (Filtern) der Größe $w \times h$ aufgebaut, wobei jedes Neuron des CNNs als eine Zelle des *Convolutional Kernels* fungiert. Die Filter teilen die Aufnahme in kleine Bereiche, sogenannte rezeptive Felder, auf. Ähnlich wie beim Lesen eines Textes wird Stück für Stück, Zeile für Zeile der Filter auf dem Bild angewendet. In der Objekterkennung berechnet der Kernel aus den festen Gewichten des rezeptiven Feldes und dem Bildinhalt des Sichtfensters eine Ergebnismatrix. Jede Zelle der Ergebnismatrix ist dabei nur mit dem rezeptiven Feld der vorherigen Schicht verbunden. Anschaulich ist dieser Vorgang in Abb. 10 dargestellt. Die Schrittweite der Verschiebung (*Stride*) und die Kernelgröße kann schichtweise bei der Berechnung der Ergebnismatrix variieren, sodass die Dimensionen der Ergebnismatrix kleiner als die der Eingabematrix ausfallen können. Oft wird die Schrittweite auf 1 und ein *Padding* festgelegt, um die Dimensionen der Matrix unverändert zu lassen. Das *Padding* beschreibt das Verhalten des Filters an den Grenzen der Matrix [3]. In der unten gezeigten Abbildung wird beispielsweise die Matrix um Nullwerte erweitert (*Zero-Padding*).

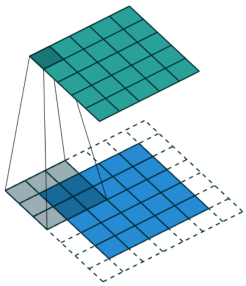


Abbildung 10: Vorgang einer Faltung mit einer Kernelgröße von 3×3 . Der graue Bereich stellt das *Padding* dar [7].

Auf eine oder mehrere (häufig zwei) *Convolutional Layers* folgt ein *Pooling Layer*. Ähnlich zu der Arbeitsweise der *Convolutional Layers* fasst die *Pooling Layer* die erzeugten *Feature-Maps* zusammen und gibt nur die relevantesten Informationen weiter. Dadurch reduziert sich die Anzahl der Parameter des Netzes und ermöglicht eine abstraktere Repräsentation des Inhaltes [3]. Häufig wird eine *Max Pooling Layer* verwendet, welche, wie in Abb. 11 zu sehen, das Maximum des Kernels errechnet.

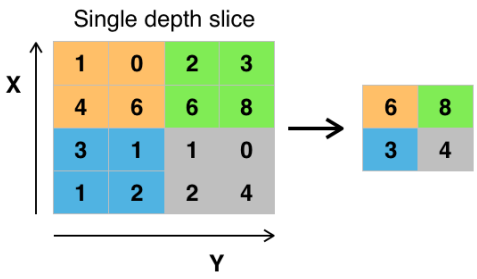


Abbildung 11: Ergebnis der *Max Pooling* Operation mit der Filtergröße 2×2 und dem *Stride* 2 [41].

Die letzten Schichten des CNNs bestehen aus einem oder mehreren *Fully Connected Layers*, welche die Klassifizierung realisiert und damit die Vorhersage trifft.

Die Eingabe eines CNNs ist ein Element der Dimension $n \times w \times h \times c$. Hierbei steht n für die Anzahl der Bilder, w und h für die Breite und Höhe des Bildes und c für die Anzahl an Farbkanälen. Oft wird die Dimension in der Variante $w \times h$ angegeben, da n variieren kann und c in der Regel bei der Objekterkennung drei beträgt (RGB). Da in diesem Fall nur ein Kanal genutzt wird, werden die Eingaben um zwei weitere Kanäle erweitert. Die Bilder werden dadurch kompatibel zu den implementierten CNNs und ermöglichen die Verwendung von vortrainierten Gewichten.

3.2.3. SSD: Single Shot Detector

Üblicherweise arbeiten moderne Objektdetektoren nach dem Ablauf: Spekulationen über *Bounding Boxes* von Objekten aufstellen, erneut Features für die gefunden Objekte extrahieren und die gefundenen Objekte klassifizieren. Diese Methode ist sehr aufwendig in der Berechnung und für Echtzeitanwendungen schlichtweg zu langsam (Vergleich moderner Architekturen in Abb. 21). In dieser Arbeit wird hingegen der *Single Shot Detector* (SSD) verwendet, welcher dem typischen Ablauf nicht folgt [27]. Nach Abb. 12 besteht das Netz aus dem *Base Network*, in diesem Fall VGG-16 [35] und sechs weiteren *Convolutional Layers*. Das *Base Network* liefert, wie in Kapitel 3.2.2 beschrieben, abstrakte Features des Bildes.

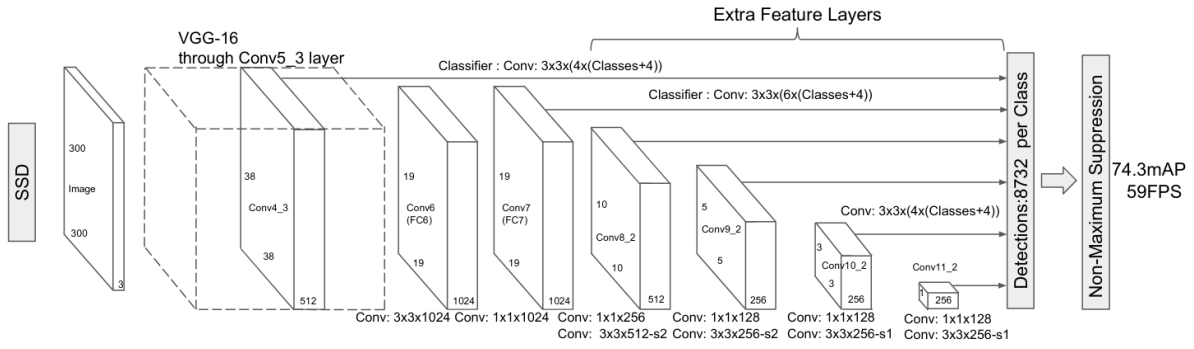


Abbildung 12: Aufbau des SSD [27].

Anders als die üblichen Objektdetektoren wird bei dem SSD eine bestimmte Menge an *Default Bounding Boxes* festgelegt. Anhand diesen wird die Abweichung der Position und Form zu dem tatsächlichen Objekt prognostiziert. Die hinzugefügten *Convolutional Layers* werden in immer kleinere Dimensionen gefaltet, um Objekte in verschiedenen Größen vorherzusagen. Jedes dieser hinzugefügten *Convolutional Layers* ist in der Lage Objekte zu

detektieren. Da benachbarte *Default Bounding Boxes* oft ähnliche Vorhersagen tätigen, wird ein Objekt oft mehrmals erkannt. Mittels *Non-Maximum Supression* werden vorhergesagte *Bounding Boxes* entfernt. Diese berechnet die *Intersection-over-Union* (definiert in Abschnitt 5) unter den detektierten *bounding boxes* und entfernt diese, wenn ein bestimmter Schwellenwert überschritten wird.

Beim Trainieren des SSDs werden Informationen über *(True) Negatives* benötigt, welche darstellen, wie ein gesuchtes Objekt nicht aussieht. Diese helfen bei der Minimierung der *False Positive*-Rate bei der Detektion. Der SSD generiert diese automatisch aus Bereichen der Bilder, welche nicht von den Labels der *Groundtruth* abgedeckt werden. Mittels des *Hard Negative Mining* wird gezielt eine passenden Menge von *Negatives* gewählt [27]. Für das Training werden daher keine expliziten Bilder benötigt, welche die Rolle als *Negatives* annehmen.

4. Umsetzung

4.1. Ablauf und verwendete Versionen

Der folgende Abschnitt beschreibt die Umsetzung in chronologischer Reihenfolge. Zu Beginn werden aus den vorliegenden Daten angemessene Datensätze extrahiert. Im Anschluss folgt für jeden Datensatz die Datenvorverarbeitung. Danach werden die Modelle definiert und unter verschiedenen Konfigurationen trainiert. Ein Vergleich der Ergebnisse hebt die resultierende Konfiguration hervor. Im Folgenden werden die einzelnen Schritte erläutert.

Für die Definition und das Training beider Modelle wird Python 3.7 verwendet. Bei der Datenaufbereitung kommt die Bildbearbeitungs-Bibliothek OpenCV [17] zum Einsatz. Das Modell basierend auf Klassifikation vewendet Tensorflow 2.3 [1] als Deep-Learning-Framework, hingegen das Modell basierend auf die Regression der Orientierung Pytorch 1.6.0 [32] verwendet.

4.2. Datenaufbereitung

4.2.1. Datenextraktion

Wie in Abschnitt 3.1.2 beschrieben, liegen die Tiefendaten im Videoformat vor. Da das Training und die Erkennung in der Regel auf einzelnen Bildern ausgeführt wird, müssen diese aus den Videos extrahiert werden. Im vorgegebenen Programm ASuLa lässt sich unkompliziert ein Speichermechanismus für die Bilder implementieren. Unter Angabe eines validen Speicherpfades und der Wertanpassung eines geeigneten Bildformates können die Bilder automatisch gespeichert werden. In unserem Fall wurde das Bildformat PNG verwendet, um Komprimierungsverluste zu verhindern.

In der Regel profitieren trainierte Modelle von einer großen Anzahl an qualitativ guten Trainingsdaten, um besonders viele Fälle abzudecken. In diesem Fall wird aufgrund des hohen Aufwandes beim Labeln die Datensatzgröße auf eine Größe zwischen 1000 und 1500 Bildern eingeschränkt.

Damit die Bilder nicht nur einen kleinen Abschnitt der Aufnahme repräsentieren, wird eine Zeitspanne festgelegt, in welcher maximal ein Bild gespeichert wird. Als vertretbare Zeitspanne wird hier die Differenz von $1\,s$ gewählt. Diese Zeitspanne ist nicht zu groß, um zu wenig Bilder zu generieren, aber groß genug, um nicht zu viele ähnliche Bilder zu erhalten. Eine Sekunde ergibt bei einer Bildwiederholungsrate von $30\,fps$ und einer Anzahl von höchstens 27000 Bildern eine Menge von maximal 900 Bildern pro Aufnahmesequenz.

Viele dieser Bilder entfallen als mögliche Trainigsdaten, da auf diesen keine Hände zu erkennen sind oder die OP-Lampen wichtige Bereiche des Bildes verdecken. Um die Bilder herauszufiltern, bei denen die OP-Lampe ein Teil der Sicht verdeckt, werden die Aufnahmen manuell inspiziert. Im Nachhinein werden die Bilder, welche in den Zeiträumen liegen, aus dem Prozess ausgeschlossen. Die Bilder ohne sichtbare Hände werden beim Annotierungsprozess nicht bearbeitet und werden somit ebenfalls ausgeschlossen.

4.2.2. Datenvorverarbeitung

Die Datenvorverarbeitung umfasst mehrere voneinander unabhängige Schritte. Diese haben das Ziel die Qualität der Daten zu erhöhen, um das Training besonders effektiv zu gestalten.

Der Hauptfokus der Erkennung liegt auf dem situsnahen Bereich. Bereiche außerhalb des Situs sind für die Erkennung überflüssig und bieten durch ihre zusätzlichen Informationen eine irrelevante Last beim Lernen von wichtigen Features. Wie in Kapitel 3.1.2 beschrieben, stellt die vorliegende Punktwolke schon einen Ausschnitt der ursprünglichen Sicht des Tiefenbildes dar. Dieser ausgeschnittene Bereich deckt, wie in Abb. 5b zu sehen, noch einen großen Bereich ab, der nicht für die Erkennung in der Nähe des Situs relevant ist. Daher ist eine weitere Einschränkung der Sicht sinnvoll für das Training. Wie ein Rahmen wird der Randbereich des Bildes schwarz eingefärbt. Eine Breite von $100px$ erscheint für die Breite des Rahmens als angemessen, wie es in Abb. 13 sichtbar ist.

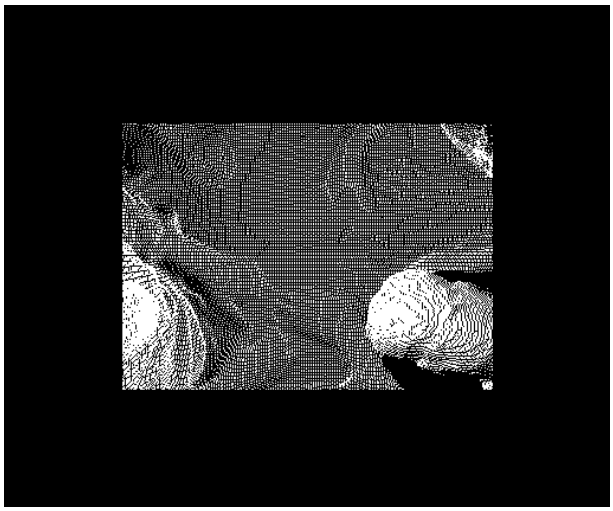


Abbildung 13: Der Bereich weit außerhalb des Situs wird ausgeblendet. Für folgende Abbildungen wird der schwarze Rahmen der Übersicht halber nicht angezeigt.

Aus der Vorrichtung des Operationssaals ist bekannt, dass der Situs sich in den Aufnahmen und in realen Umständen auf einer Höhe von ungefähr $1,10\,m$ befindet. Die Bereiche weit unterhalb oder oberhalb dieser Höhe sind nicht situsnah und sind für die Erkennung daher nicht relevant. Hände tief im Situs sind für den Anwendungsfall wichtig und daher wird die untere Grenze auf $1,05\,m$ gesetzt. Da $30\,cm$ eine angemessene Differenz in der Höhe ist, wird der situsnahe Bereich auf $1,05\,m - 1,35\,m$ festgelegt. Kombiniert ergibt sich, wie in Abb. 14 gezeigt, ein Bereich, der im dreidimensionalen Raum als situsnah festgelegt wird und den Hauptfokus bei der Erkennung abbildet.

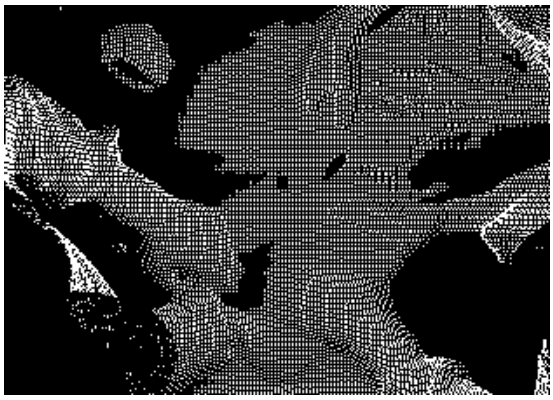


Abbildung 14: Tiefenbild im Bereich $1,05\,m - 1,35\,m$.

Die Oberfläche von Gegenständen im Abbild der Punktwolke ist in einem Helligkeitsverlauf wiedergegeben. Desto heller ein Bereich, desto näher ist dieser an der Kamera. Andersherum ist ein Bereich dunkler, wenn der Bereich weiter von der Kamera entfernt ist. Durch die vergleichsweise kleine Spanne von $30cm$ sind die Helligkeitsunterschiede verschieden naher Objekte kaum wahrnehmbar. Nahezu alle Hände erscheinen in sehr ähnlichen Helligkeitswerten wie der Hintergrund.

Als die vermutlich gängigste Lösung des Problems bietet sich eine Normalisierung der Daten an. Normalisieren bildet einen gewünschten Wertebereich auf den neuen Wertebereich zwischen $[0, 1]$ ab. In unserem Fall werden die Höhenunterschiede maßstabsgetreu vergrößert, was die Hände stärker hervorhebt. Die Normalisierung an einen Punkt berechnet sich durch:

$$h(x) = \frac{f(x) - x_{min}}{x_{max} - x_{min}} \tag{4.1}$$

Dabei entspricht $f(x)$ das unveränderte bzw. $h(x)$ das normalisierte Bild an der Position x. Die Untergrenze x_{min} beträgt $1,05\,m$, analog dazu beträgt die Obergrenze x_{max} $1,35\,m$.

Nach der Normalisierung sind kleinere Abstände besser zu erkennen. Die Hände sind nun durch die größeren Helligkeitsunterschiede einfacher von Hintergrund unterscheidbar, wie es in Abb. 15 zu sehen ist.

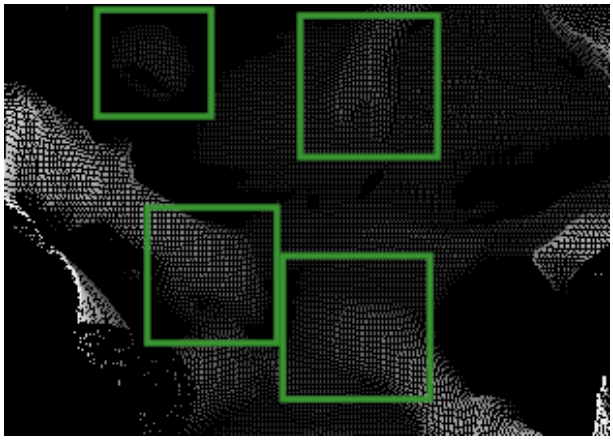


Abbildung 15: Die Hände erscheinen durch die Normalisierung konstrastreicher. Die Hände sind in grün umrandet.

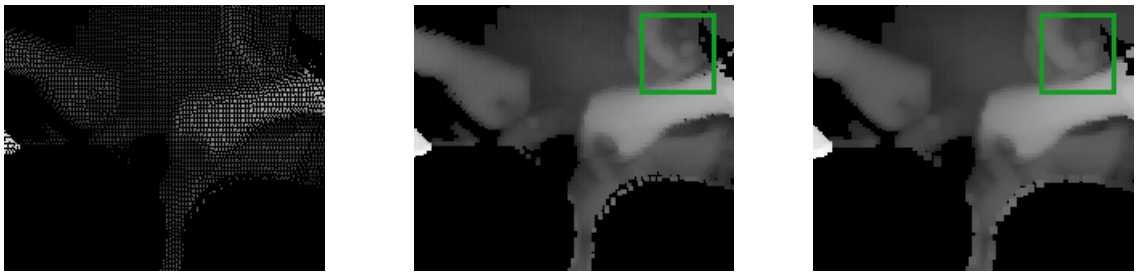
Im Gegensatz zu Objekten in RGB-Aufnahmen sind Objekte in der Punktwolke in einer punktartigen Struktur gehüllt. Auf Pixelebene betrachtet wirken die Oberflächen durch die starken Kontraste nicht zusammenhängend. Ebenso sind die Kanten der Objekte durch die Kontraste schwer auf Pixelebene zu erkennen. Das ist für die Erkennung nicht optimal, denn für die Erkennung von Händen sind gleichmäßige Oberflächen und klare Kanten von großer Bedeutung. Eine beliebte Methode um die Oberflächen zu glätten ist die Verwendung eines Rauschfilters. Nach einigen Experimenten mit dem bilateralen Filter [40] ergibt sich jedoch das Ergebnis, dass das bilaterale Filtern nicht in der Lage ist, die mosaikartige Struktur zu glätten.

Eine bessere Methode der unreinen Struktur entgegenzuwirken bietet die Dilatation. Die Dilatation gehört zu den morphologischen Operationen und ermöglicht die visuelle Ausdehnung von Formen in Bildern. Formal lautet eine mögliche Definition nach Tambe et al. folgendermaßen [37]:

$$X \oplus B = \{x + b : x \in X \,\&\, b \in B\} \tag{4.2}$$

Jedes Element des Bildes X wird um das strukturierende Element b erweitert. Das strukturierende Element ist eine Menge, welche in Verbindung mit morphologischen Operatoren Aufschluss über die Interaktion gibt [36]. In unserem Fall besteht das strukturierende Element aus einem Quadrat mit variabler Größe.

In Abb. 16 sind ausgeführte Dilatationen mit strukturierenden Elementen unterschiedlicher Größe abgebildet. In diesem Fall beträgt eine geeignete Größe 3×3. Ist das strukturierende Element kleiner als diese Größe, ist die unreine Struktur noch sichtbar. Anders herum werden bei höheren Größen feine Strukturen an der Hand verzerrt, wie es in Abb. 16c im Vergleich zu Abb. 16b eingezeichnet ist.



(a) Elementgröße: 1×1 (b) Elementgröße: 3×3 (c) Elementgröße: 5×5

Abbildung 16: Unterschiedliche Größen des strukturierenden Elements. 1×1 bildet das neutrale Element.

4.2.3. Auswahl der Daten

Für ein gutes Ergebnis ist die Qualität der Daten von großer Bedeutung. Darunter zählt vor allem eine angemessene Wahl der Trainingsdaten. Idealerweise zeigen einzelne Bilder keine Mängel auf und sind sauber gelabelt. Bilder, welche durch OP-Lampen verdeckt sind (beschrieben in Abschnitt 3.1.2), gelten als mangelhaft und werden daher entfernt. Denn im Falle einer Umsetzung der autonomen OP-Lampen ist eine Verdeckung der Kamera durch ihre Konstruktion ausgeschlossen. Daher wird etwa die Hälfte der Aufnahmezeit, welche eine deutliche Verdeckung der Sicht durch die Lampen aufweist, entfernt. Sichteinschränkungen aufgrund der Köpfe werden auch nach einer Realisierung noch vorkommen und bleiben aufgrund dessen in den Daten enthalten.

Eine Aufnahme deckt einen Zeitraum von maximal 15 Minuten ab und repräsentiert daher keine ganze OP. Infolgedessen wird für jede Aufnahme zufällig eine feste Anzahl von 120 Bildern gewählt, um besonders viele verschiedene Szenarien abzudecken. Dies ist eine angemessene Menge, da diese bei zehn verwendbaren Aufnahmesequenzen eine Anzahl von 1200 Bildern ergibt.

Diese Bilder müssen anschließend entsprechend annotiert werden, um als Referenzdaten für das Training zu fungieren. Die gängige Methode im Bereich der Objekterkennung ist das Annotieren in Form einer *Axis Aligned Bounding Box* (AABB). Dies ist ein minimal überdeckendes Rechteck, deren Seiten parallel zu den Achsen des Systems sind. Der Nachteil an diesem Annotierungsformat ist die fehlende Möglichkeit, Informationen über die Orientierung zu speichern.

Der AABB steht als Erweiterung die *Oriented Bounding Box* (OBB) gegenüber. Der Unterschied zu den AABB liegt darin, dass die Seiten des OBB nicht parallel zu den im System definierten Achsen sein müssen und somit ermöglicht, die Ausrichtung der Objekte zu speichern. Da viele Anwendungsfälle die Punktsymmetrie der OBB ausnutzen, wird die Ausrichtung in der Regel im Intervall $(-90°; 0°]$ angegeben. In unserem Fall ist allerdings die Ausrichtung im Intervall $[0°; 360°)$ erforderlich, daher wird eigenhändig ein bestehendes Annotierungstool modifiziert. Das resultierende Format gleicht dem Tupel

$$ann_{obb} = (x, y, w, h, a, ax, ay, bx, by, cx, cy, dx, dy) \tag{4.3}$$

bei dem x und y die Koordinaten des Mittelpunkts, w und h die Breite und Höhe, a die Ausrichtung in Radiant(rad) und ax, bx, cx, dx bzw. ay, by, cy, dy die Koordinaten der Eckpunkte darstellen.

Zusätzlich die Eckkoordinaten der OBB zu speichern, ermöglicht es eine grobe Abschätzung der AABB zu berechnen. Doch diese ist zu ungenau, wie es in Abb. 17 dargestellt ist. Für das Klassifikationsmodell musste deswegen das Labeln der Aufnahmen wiederholt werden. Die Rotation hingegen wurde mit der Formel 4.4 in die dementsprechende Klasse konvertiert, bei dem a der Rotation in rad entspricht. Somit müssen keine Aufnahmen mehrmals im Bezug zur Ausrichtung gelabelt werden und weisen darüber hinaus fairerweise dieselben Ungenauigkeiten in der Rotation auf.

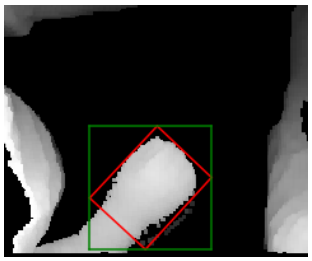


Abbildung 17: Transformation einer OBB (rot) zu der Abschätzung einer AABB (grün).

$$cls(a) = (a \; div \; (\frac{1}{9}\pi)) + 1 \tag{4.4}$$

In der Regel werden die Daten beim Trainingsprozess in drei verschiedene Datensätze aufgeteilt: *train*, *validation* und *test*. Beim Trainieren wird das Modell unter einer bestimmten Konfiguration auf dem *train* Datensatz trainiert. Das trainierte Modell wird mithilfe des *validation* Datensatzes ausgewertet und unter verbesserten Hyperparameter erneut trainiert. Sobald das Modell endgültig trainiert wurde, wird es auf dem bisher ungesehenen *test* Datensatz ausgewertet. Dadurch erlangt das Modell keine Vorteile durch den Aufbau der anderen Datensätze. Bei beiden Modellen sind die Datensätze identisch.

4.2.4. Analyse und Augmentation der Daten

Ein Überblick über den *train* Datensatz ermöglicht uns zu erkennen, ob sich der Datensatz für das Training eignet. Von den 1200 Bildern sind auf 290 Bildern keine Hände zu erkennen und werden weggelassen. Wie in Kapitel 3.2.3 beschrieben sind diese für das Training nicht relevant. Der Rest wird in die drei Datensätze aufgeteilt: *train* (607), *validation* (153) und *test* (150).

Im Trainingsdatensatz sind 1407 Hände gelabelt worden, das entspricht einem Durchschnitt von $\approx 2,32$ Händen bei einem Median von zwei Händen pro Bild. Diese befinden sich, wie in der Heatmap in Abb. 18 abgebildet, vergleichsweise weit gestreut auf dem ganzen Sichtfeld. Ab und zu sind kleinere Ansammlungen, insbesondere in der unmittelbaren Nähe des Situs, zu sehen.

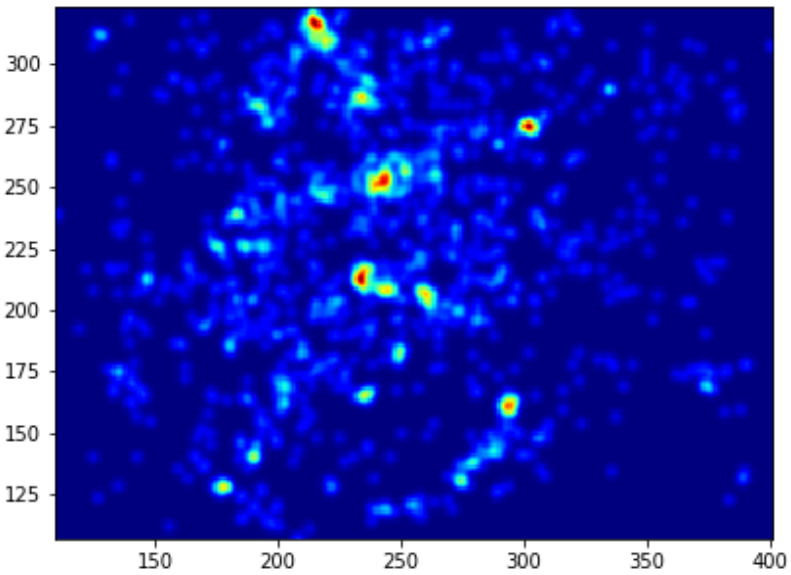


Abbildung 18: Heatmap der Positionen des *train* Datensatzes. Man beachte, dass die schwarzen Rahmen abgeschnitten sind.

Eine Verteilung über der Ausrichtung der Hände ist in Abb. 19 abgebildet. Besonders auffällig ist die hohe Anzahl an Händen in dem Intervall [45°; 135°] und die knappe Anzahl in dem Intervall [315°; 45°]. Generell ist die Anzahl der Hände in den horizontalen Bereichen höher als die Anzahl in den vertikalen Ausrichtungen. Dies ist auf den Aufbau des OP-Saals zurückzuführen, weil das Personal nur auf der rechten und linken Seite des Situs stehen können. Die oben genannten Erkenntnisse treffen im Allgemeinen auch auf die anderen Datensätze zu, schließlich wurden diese zufällig zugeteilt.

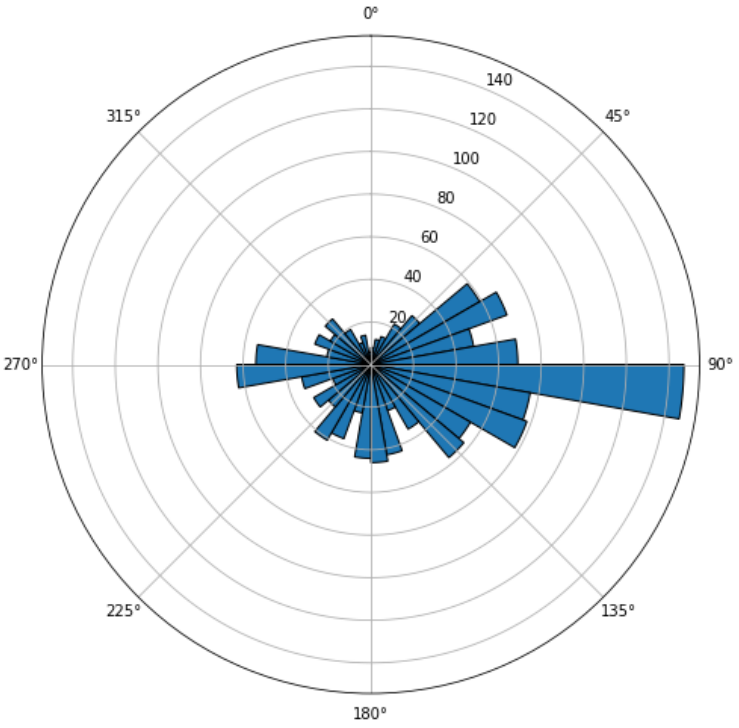
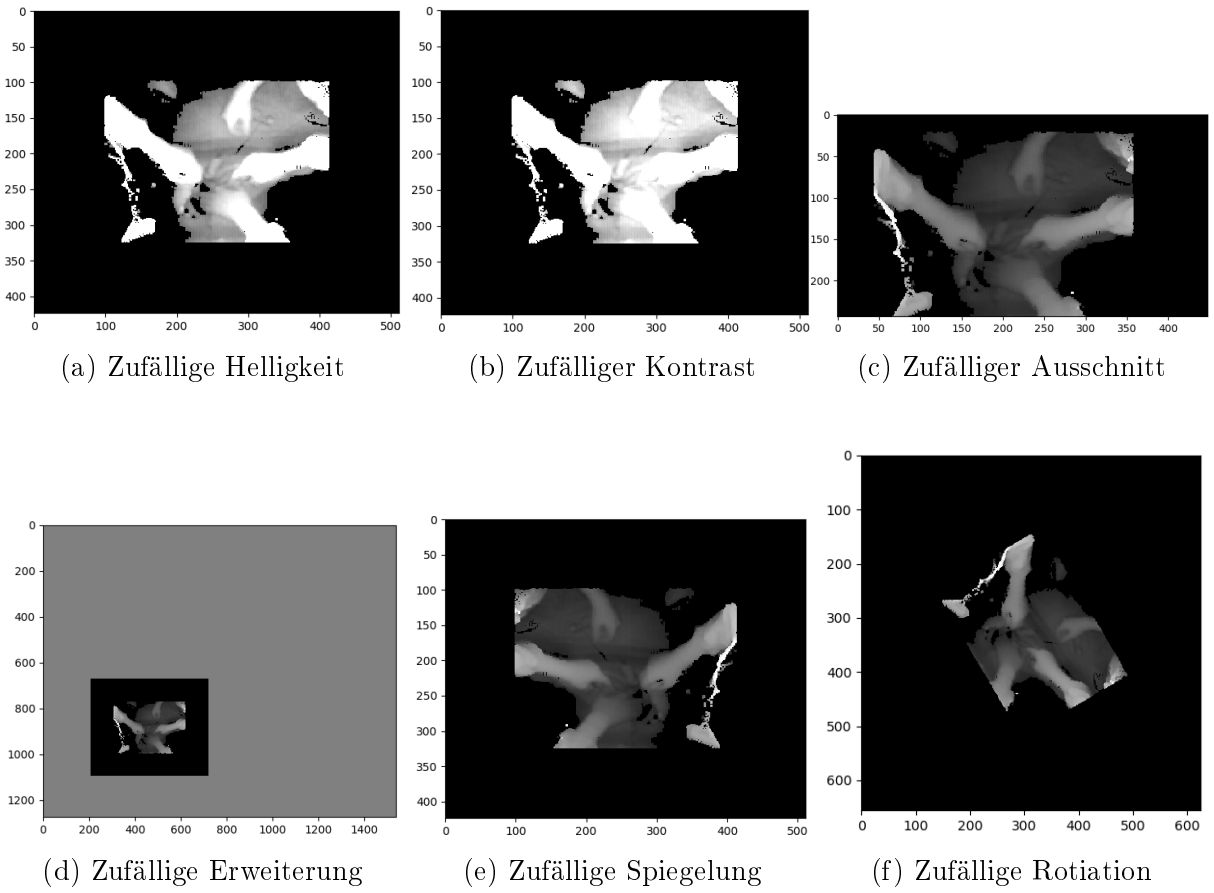


Abbildung 19: Verteilung in der Ausrichtung der Hände. Ein Balken deckt eine Spanne von 10° ab.

Die unausgeglichene Verteilung der Ausrichtungen der Hände stellt ein Problem in der Erkennung dar. Denn die nach oben gerichtete Hände sind durch die Trainingsdaten nicht ausreichend abgedeckt. Dies sorgt in der Anwendung für eine schlechte Leistung im Bezug zu nördlich gerichteten Händen.

Die Datensätze befinden sich im Anhang in digitaler Form auf dem beigelegtem Speichermedium.

Als weitverbreitetes Verfahren wird die Transformation der Daten, auch Augmentation genannt, verwendet. Diese Methode erweitert künstlich den Datensatz durch einer zufälligen Kombination einfacher Bildtransformation, um *Overfitting* zu vermindern [22]. Oft wird dieser Schritt, wie auch in unseren beiden Modellen, kurz nach dem Hineinladen in das Netz durchgeführt. Unter den Transformationen sind für RGB-Aufnahmen neben den formändernden, auch photometrische Transformationen bezüglich der Farben enthalten. Diese werden nicht benutzt, da die Tiefendaten nicht von solchen Störfaktoren betroffen werden können. In der Abb. 20 sind alle Ergebnisse der Transformationen dargestellt.



(a) Zufällige Helligkeit (b) Zufälliger Kontrast (c) Zufälliger Ausschnitt

(d) Zufällige Erweiterung (e) Zufällige Spiegelung (f) Zufällige Rotiation

Abbildung 20: Mögliche Operationen der Transformation.

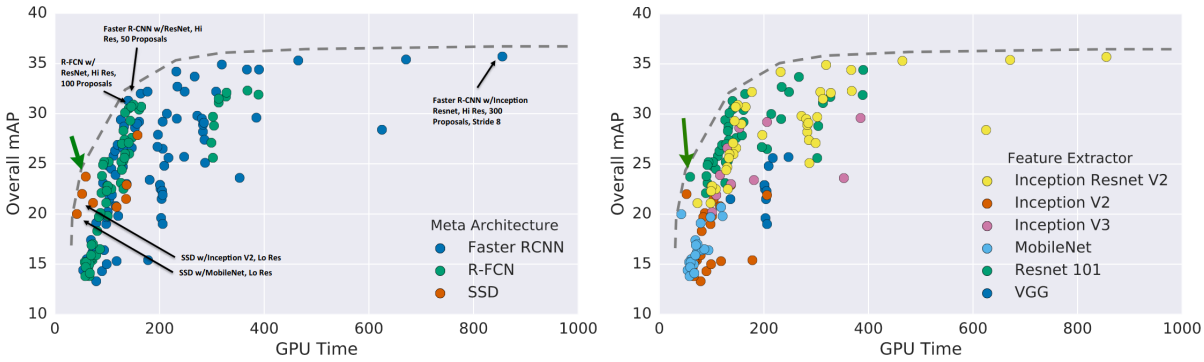
Bei der Rotation bzw. Spiegelung der Aufnahmen stimmt die Klasse der Hände nicht mehr mit der gelabelten Klasse überein. Diese werden je nach Transformation angepasst.

4.3. Definition der Modelle

Dieses Kapitel erläutert den Vorgang der Definition von den Modellen. Für das Klassifikationsmodell wird die Implementierung mittels der *Tensorflow Object Detection API* (ODAPI) verwendet [14]. Diese stellt eine textuelle Schnittstelle bereit, mit der sich besonders einfach Detektionsmodelle konfigurieren lassen. Im Gegensatz dazu ist die Implementierung des Regressionsmodell von Yang et al. [44] online[1] zur Verfügung gestellt.

4.3.1. Klassifikationsmodell

Die Wahl einer passenden Architektur für das Klassifikationsmodell erweist sich aber durch die hohe Anzahl an verschiedenen Architekturen in Kombination mit den *Feature-Extractorn* als schwierig. Daher wird die Wahl einer Subarchitektur der Übersicht halber in zwei Kriterien beurteilt: Genauigkeit und Echtzeitfähigkeit. Huang et al. vergleichen in ihrer Arbeit die von der ODAPI bereitgestellten Modelle auf diese beiden Merkmale [15]. Die drei Metaarchitekturen *Faster Region-based Convolutional Neural Networks* (Faster R-CNN) [34], *Region-based fully Convolutional Networks* (R-FCN) [5] und *Single Shot Detector* (SSD) [27] werden unter gleichen Bedingungen getestet. Eine Metaarchitektur stellt eine eigene Klasse von Architekturen dar, welche viele weitere Architekturen in dem Bereich beeinflusst. Obwohl jedes dieser Netzwerkarchitekturen mit einem bestimmten *Feature-Extractor* vorgestellt wurde, werden in dem Vergleich die Metaarchitekturen ebenfalls mit anderen *Feature-Extractorn* getestet um den Einfluss bestimmter Feature-Extractorn auf das Ergebnis zu verhindern. Das Resultat dieser Arbeit ist in Abb. 21 zu sehen. Die Genauigkeit dieser Modelle ist in *mean Average Precision* (mAP) angegeben, welche in Kapitel 5 definiert ist.



(a) Im Bezug der Metaarchitekturen (b) Im Bezug der Feature-Extraktoren

Abbildung 21: Ergebnis des Vergleichs zwischen den Metaarchitekturen. Der grüne Pfeil signalisiert das bevorzugte Modell [15].

[1] Online unter: https://github.com/yangli18/hand_detection

In Form einer grauen gestrichelten Linie sind in Abb. 21 Punkte einer „Optimalitätsgrenze“ dargelegt, bei welcher eine höhere Genauigkeit nur durch eine Einbüßung der Ausführungsgeschwindigkeit erreichbar ist [15]. Desto näher sich ein Modell an der Optimalitätsgrenze zu der gleichen Laufzeit (GPU Time) befindet, desto höher ist die mögliche Leistung in der Detektion ausgeschöpft. Unter Rücksichtnahme der Anforderung der Echtzeitfähigkeit passen nur Detektoren der SSD Klasse in das passende Schema (Aufbau und Funktionsweise in Abschnitt 3.2.3 beschrieben). Insbesondere das Modell „SSD w/Resnet 101“ (in Abb. 21 eingezeichnet) fällt durch die Nähe an der Optimalitätsgrenze und der niedrigen Zeit auf. Bei diesem Detektionsmodell handelt es sich um ein SSD mit dem *Feature-Extractor* Resnet 101 [11]. Zu diesem speziellen Netz bietet die ODAPI eine weitere Version an, welches pro Vorhersage nur die 50 wahrscheinlichsten Positionen einer Hand in Betracht zieht. Im Vergleich zu dem Modell „SSD w/Resnet 101“ ist die Genauigkeit marginal geringer, dennoch profitiert das Netz von einer noch niedrigeren Inferenzzeit. Andere Modelle, wie die Gruppe der Faster R-CNN, erzielen zwar eine höhere Genauigkeit, sind aber schlichtweg durch die hohe Ausführungszeit für Echtzeitanwendungen ungeeignet und kommen für diese Problemstellung nicht in Frage.

Vor dem Trainieren müssen die Daten in ein passendes Format gebracht werden. Mithilfe einer individualisierten Version eines Skripts, welche in der ODAPI enthalten ist, werden die Bilder samt ihrer Annotation in eine „tfrecord“ Datei exportiert. Zuletzt folgt die Konfiguration des Trainings durch die textuelle Schnittstelle. Diese beinhaltet neben allen wichtigen Parametern des Trainings auch organisatorische Einstellungsfelder wie die Pfade der Datensätze. Im Anhang befindet sich die Konfigurationsdatei in digitaler Version.

4.3.2. Regressionsmodell

Dem Klassifkationsmodell gegenüber steht das Regressionsmodell nach Yang et al. [44]. Dieses Modell ist, wie in Abb. 22 erkennbar, stark an einem SSD angelehnt und nutzt ein MobileNet als *Feature-Extractor*. Die Erkennung von Händen ist ähnlich aufgebaut wie beim Klassfikationsmodell, jedoch wird eine erkannte Hand nur einer einzigen Klasse zugeordnet.

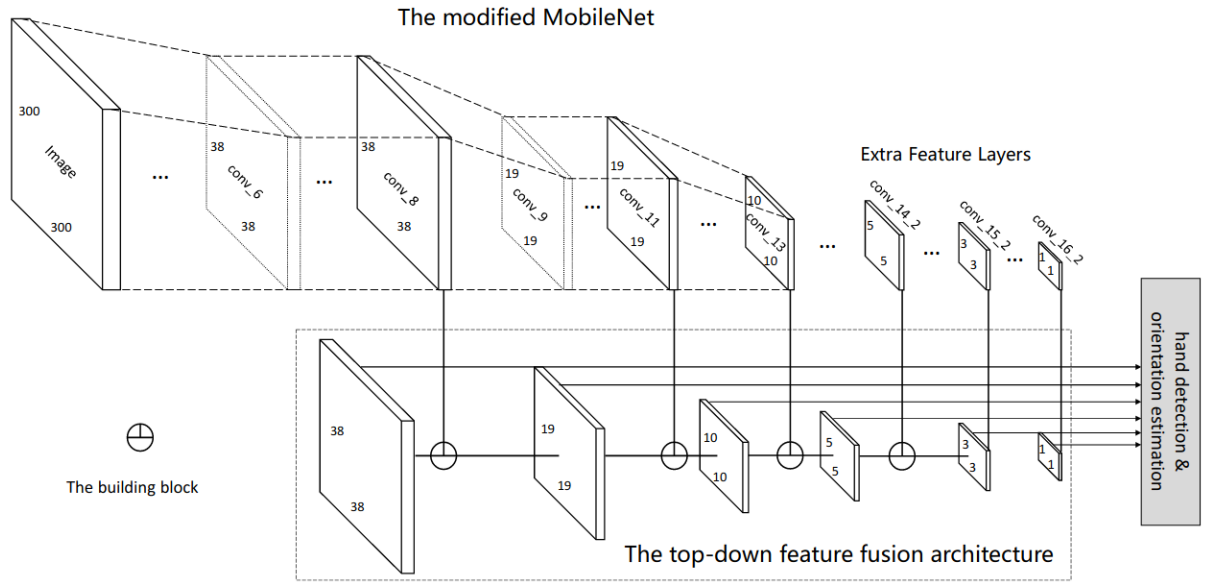


Abbildung 22: Aufbau des Refgressionsmodell [44].

Im Gegensatz zum üblichen Aufbau des MobileNets, knüpfen in dem Netz nach der Schicht *conv_13* neun weitere *Convolutional Layers* an, die die Auflösung der *Feature-Maps* etappenweise verringert. Anders als der originale Aufbau des SSDs (Abb. 12), werden die *Feature-Maps* nach den Schichten *conv_8, 11, 13, 14_2, 15_2, 16_2* mithilfe von *Building Blocks* mit den *Feature-Maps* des Detektors verbunden. Diese *Top-Down Feature Fusion Architecture* aktualisiert die *Feature-Maps* der SSD stetig mit neuen Kontextinformationen des Bildes [44].

Die veröffentlichte Implementierung unterstützt aufgrund Kompatibilitätsgründen nicht die aktuelle Version von PyTorch (1.6.0) und Python (3.8). Durch einige nicht nennenswerte Modifikationen lässt sich dies beheben. Des Weiteren wurde die Datenverwaltung aufgrund der individuellen Struktur der Labels angepasst.

Das Netz nimmt Bilder in einer festen Dimension von 300×300 als Eingabe auf, Bilder einer anderen Auflösung werden als erstes auf die Größe angepasst. Diese Anpassung wirkt sich positiv auf die Laufzeit pro Vorhersage aus [27]. Dabei bleibt die Leistung in der Erkennung über die unterschiedliche Auflösungen von Eingaben nahezu identisch.

4.4. Training

Dieser Abschnitt beschreibt den Trainingsprozess beider Modelle. Aus technischen Gründen steht zum Trainieren nur ein persönlicher Rechner zur Verfügung, welcher nicht viel Leistung bietet (Nvidia GTX 1060 6Gb). Infolgedessen sind in Kombination mit den längeren Trainingszeiten (ca. 6h - 16h) nur wenige Trainingskonfigurationen umsetzbar.

Beide Modelle werden unter der gleichen Strategie trainiert. Wie in Kapitel 3.2.2 beschrieben, werden vortrainierte Gewichte (auf dem COCO Datensatz trainiert [26]) für die Initialisierung der Verbindungen des Netzes verwendet. Die Netze werden im Anschluss auf unserem Datensatz weiter trainiert.

Da die optimale Trainingsdauer nicht bekannt ist, werden für beide Modelle vorerst eine große Schrittanzahl ausgewählt. Ein Schritt beschreibt den Aufwand bei dem ein Batch durch das Netz geleitet und die Gewichte der Neuronen angepasst werden. Eine gängige Methode, um die optimale Schrittanzahl zu bestimmen, ist die Suche nach dem Zeitpunkt, an welcher der *Loss* sich nicht mehr durch weiteres Training verringert. Werden weniger Schritte durchlaufen, kommt es zu *Underfitting* und anders herum führt zu langes Training zu ungewolltem *Overfitting*. Während des Trainings wurde in regelmäßigen Abschnitten die Leistung des Netzes anhand des *validation*-Datensatzes getestet.

Die Ungewissheit ist ebenfalls in der optimalen Größe des *train*-Datensatzes wieder erkennbar. In unserem Fall besteht dieser Datensatz aus 607 gelabelten Bildern. Zu dem Datensatz wurden 100 gelabelte Bilder hinzugefügt und entfernt um den Einfluss der Größe des Datensatzes zu vergleichen. Die resultierende Modelle setzen sich aus den Werten zusammen, die die höchste Genauigkeit erreichen.

Tabelle 1: Trainingskonfigurationen beider Modelltypen

Parameter	Klassifikationsmodell	Regressionsmodell
Batchgröße	4	16
Schrittanzahl	50000	120000
Lernrate	0,013333	0,001
Aktivierungsfunktion	RELU_6	RELU_6
Verlustfunktion	Klassifikation: Focal Loss Lokalisierung: Smooth L1 Loss	Kombination der Losskomponenten

4.4.1. Klassifikationsmodell

Das Klassifikationsmodell wurde mit der vordefinierten Konfiguration des Modells trainiert. Die genaue Initialisierung der Parameter des Modells und des Trainings sind in Anhang in digitaler Form dargestellt. Die Batchgröße, welche die Menge an Bildern in einem Schritt darstellt, reduziert sich aufgrund der Limitierung der Hardware auf 4. Der Trainingsprozess startet unter der Konfiguration in Tabelle 1 mit dem Ziel 100000 Schritte auszuführen. Nach 50000 Schritten musste der Trainingsvorgang aufgrund zu hohem Zeitaufwand (16h) abgebrochen werden. Für weitere Experimente wird die Schrittanzahl 12000 gewählt.

Der Verlust wird gleichgewichtet aus dem Klassifikation-*Loss* und dem Lokalisierung-*Loss* berechnet. Ersteres wird mit dem *Weighted Sigmoid Focal Loss* und der andere mit dem *Smooth L1 Loss* berechnet. *Weighted Sigmoid Focal Loss* legt bei der Berechnung des *Losses* den Fokus mehr auf die Detektionen als auf die *Negatives* [25]. Ohne den Fokus würden alle *bounding boxes* der festgelegten Menge (erläutert in Abschnitt 3.2.3) den *Loss* beeinflussen.

4.4.2. Regressionsmodell

Da unser Modell sich hauptsächlich an den der Implementierung nach Yang et al. [44] orientiert und die Autoren beim Trainieren eine ähnliche Hardwarekonfiguration benutzen um gute Ergebnisse zu erzielen, werden die Hyperparameter übernommen. Die Trainingskonfigurationen sind in Tabelle 1 aufgelistet. Obwohl eine feste Schrittanzahl vorgegeben war, wird analog zu dem Klassifikationsmodell das Modell unter verschiedenen Schrittanzahlen und Größen des Trainingsdatensatzes geprüft, da unser Datensatz eine andere Größe und Diversifikation aufweist.

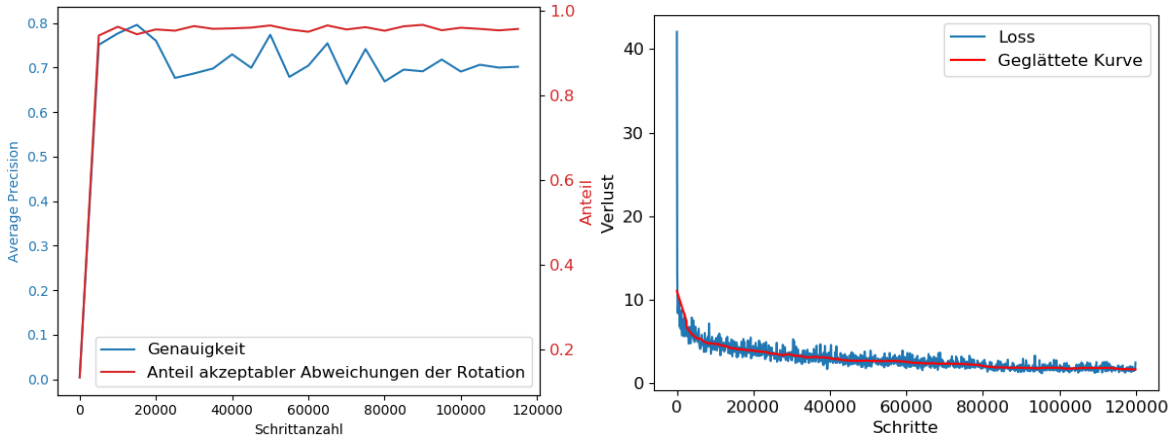
Wie in der Tabelle 1 zu sehen, wird der Loss aus mehreren einzelnen Loss-Komponenten berechnet. Das Training besteht aus der Minimierung der Funktion [44]:

$$L = \frac{1}{N}(L_{cls} + L_{loc} + L_{proj} + L_{wrist}) \tag{4.5}$$

Für N verschiedene Hände besteht der allgemeine Loss L aus dem *Loss* der Konfidenz L_{cls} und der Loss der Lokalisierung L_{loc} einer Vorhersage. Den Verlust im Bezug zur Regression der Vektoren wird vereinfacht über die Abweichung der Längen der Vektoren L_{proj} und der Position des Handgelenks L_{wrist} berechnet.

4.4.3. Resultierende Modelle

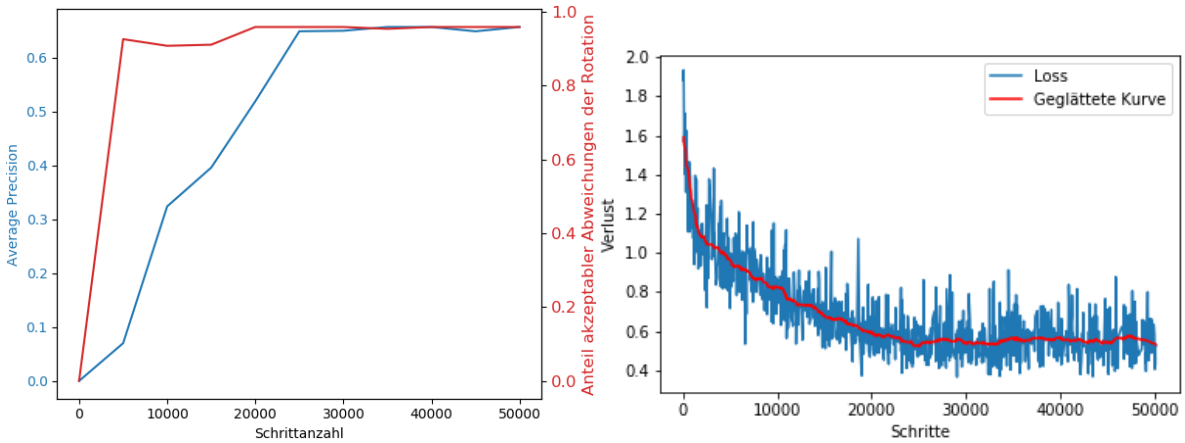
Wie oben erwähnt werden die verschiedenen Konfigurationen auf dem *validation*-Datensatz ausgewertet und verglichen. In den Abb. 23 ist zu sehen, dass sich für das Regressionsmodell die Schrittanzahl 15000 anbietet. Denn ab dem Punkt ist keine starke Abflachung des *Losses* zu beobachten und die Genauigkeit in der Detektion und Bestimmung der Rotation sind zu der Schrittanzahl hoch.



(a) Evaluation auf dem *validation* Datensatz (b) Verlustfunktion

Abbildung 23: Trainingsverlauf und Validationsergebnisse im Bezug zur Schrittanzahl des Regressionsmodells.

Analog dazu zeigt Abb. 24 die Ergebnisse des Klassifikationsmodell im Bezug zu der Schrittanzahl. Ähnlich zu den oben genannten Kriterien trifft als passende Schrittanzahl 25000 zu.



(a) Evaluation auf dem *validation* Datensatz (b) Verlustfunktion

Abbildung 24: Trainingsverlauf und Validationsergebnisse im Bezug zur Schrittanzahl des Klassifikationsmodells.

Der Vergleich zu der Datengröße ist in Abb. 25 abgebildet. Mit steigender Anzahl an Aufnahmen im *train* Datensatz wächst auch die Genauigkeit der Detektion unter Nutzung des Regressionsmodells. Anders ist es bei dem Klassifikationsmodell zu beobachten. Dort ist keine Zunahme der Genauigkeit mit wachsender Größe des Datensatzes zu erkennen. Es ist vorstellbar, dass das Regressionsmodell von einem noch größeren Datensatz profitiert. Doch aufgrund des hohen Aufwandes beim Labeln wird der Datensatz auf die eben genannte Größe beschränkt. Für die Evaluation auf dem *test* Datensatz wird das Regressionsmodell unter der Konfiguration in Tabelle 1 mit einer Schrittanzahl von 15000 und einer Datensatzgröße von 707 Bildern verwendet. Analog dazu wird das Klassifikationsmodell mit einer Datensatgröße von 607 Bildern und Schrittanzahl von 25000 verwendet. Im Anhang befinden sich in digitaler Form die resultierenden Modelle.

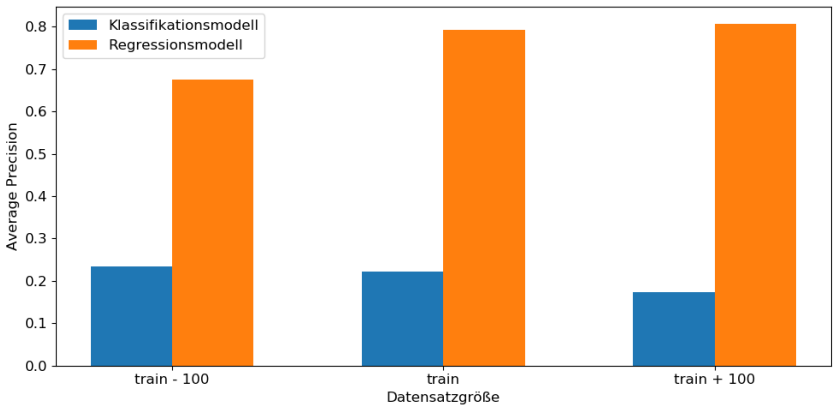


Abbildung 25: Vergleich beider Modelle auf verschiedene Datensatzgrößen. Der *train* Datensatz umfasst 607 Bilder.

5. Evaluation

In diesem Kapitel werden die resultierenden Modelle bezüglich der Anforderungen aus Kapitel 1.2 verglichen und bewertet. Dafür werden folgende Evaluationsgrößen ausgewählt:

- Leistung in der Detektion von Händen
- Genauigkeit in der Bestimmung der Ausrichtung
- benötigte Zeit der Inferenz

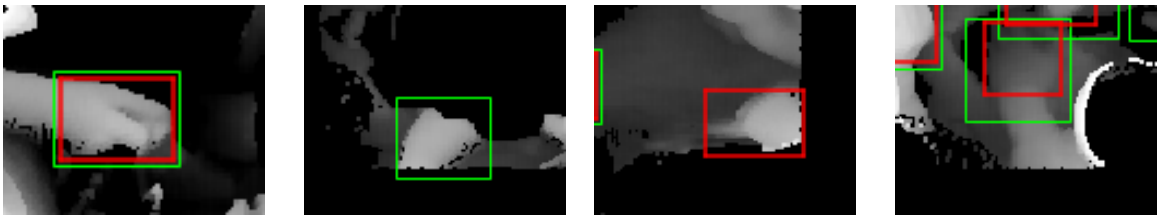
Für jede Messgröße wird außerdem die Ursache und das Ausmaß des Ergebnisses untersucht. Zum Schluss wird auf Basis der Erkenntnisse eine Entscheidung zwischen den Modellen getroffen und die dadurch entstandenen Gedanken diskutiert.

5.1. Detektion der Hände

Im Bereich der Objekterkennung wird die Leistung mit dem *mean Average Precision* (mAP) gemessen [8]. Dieser Wert ist im Bereich $[0; 1]$, wobei ein höherer Wert im allgemeinen Sinne eine bessere Erkennung hinsichtlich der *Precision* und dem *Recall* bedeutet. Um den Wert zu berechnen wird für jedes Bild aus dem *test* Datensatz eine Vorhersage getroffen. Aus den Flächen einer Vorhersage B und der Referenzdaten G wird die *Intersection-over-Union* (IoU) berechnet. Der Index ist definiert durch:

$$IoU(B, G) = \frac{B \cap G}{B \cup G} \tag{5.1}$$

Zusätzlich zur IoU wird ein Schwellenwert festgelegt, ab dem eine Vorhersage mit den Referenzdaten als übereinstimmend gilt. In der Regel (und auch in diesem Fall) wird ein Schwellwert von 50% gewählt. Wenn neben einer IoU $\geq 50\%$ auch die Klasse übereinstimmt, handelt es sich um ein *True positive* (TP), also einer Vorhersage, die der Groundtruth entspricht. Als *False Positive* (FP) werden die Fälle betrachtet, bei denen eine IoU unter dem Schwellenwert zu finden ist. Bei der mehrfachen Erkennung einer Hand, bei dem alle eine IoU von $> 50\%$ aufweisen, gilt nur eine AABB als TP. Alle anderen Detektionen werden als FP eingeordnet. Zuletzt kann es vorkommen, dass ein Objekt gar nicht erkannt wird oder das Label nicht richtig zugewiesen wird. Beide Fälle zählen zu den *False Negatives* (FN). In unserer Situation werden die Klassen nicht auf Übereinstimmung geprüft, da die Klassen bei dem Klassifikationsmodell nur die Ausrichtung betreffen. In Abb. 26 sind exemplarisch alle Fälle im Bezug zu diesem Problem zu sehen.



(a) True Positive (b) False Positive (c) False Negative (d) IoU $< 0{,}5$

Abbildung 26: Vorhersagen (grün) im Vergleich zu der *Groundtruth* (rot). Berücksichtigung von *True Negatives* (TN) ist in der Objekterkennung irrelevant.

Aus der Verteilung der Kategorien lassen sich die Metriken *Precision* und *Recall* berechnen. Eine hohe *Precision* sagt aus, dass es sich bei einer Detektion sehr wahrscheinlich um ein TP handelt. Der *Recall* stellt den Anteil der vorhergesagten TP zu allen existierenden TP in der *Groundtruth* dar.

Aus der Sortierung der Detektionen nach ihrer Konfidenz ist es möglich eine *Precision/Recall*-Kurve zu erstellen. Im Anschluss wird eine streng monoton fallende Verison der Kurve berechnet, die die *Precision*-Werte auf das Maximum von sich und aller nachfolgenden Werten zuordnet. In Abb. 27 ist die *Precision/Recall*-Kurve für einen konkreten Fall unserer Arbeit zu sehen. Die *Average Precision* (AP) ist die Fläche unterhalb des Graphen und wird durch die Integration der Kurve berechnet. Ein großer Flächeninhalt und dementsprechend eine hohe AP zeigt eine durchgängig, hohe *Precision* und *Recall* in der Erkennung.

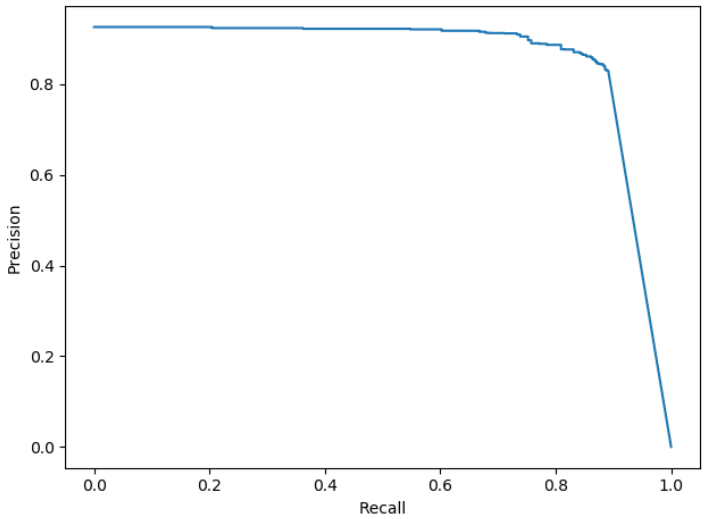


Abbildung 27: *Precision/Recall*-Kurve des Regressionsmodell auf dem *test* Datensatz

Bei der Auswertung auf dem *test*-Datensatz kommt das Klassifikationsmodell auf eine AP von 69,9%, während das Regressionsmodell auf eine AP von 83,7% kommt. Das ist ein bemerkbarer Unterschied im Detektionsvermögen, wie in Abb. 28 exemplarisch zu sehen. Trotz des gleichen *confidence*-Schwellenwertes in der Erkennung (0,4), werden auf vielen Bildern mehr Hände durch das Regressionsmodell, als durch das Klassifikationsmodell erkannt. Von 374 manuell gelabelten Händen detektiert das Klassifikationsmodell nur 152 Hände, im Gegensatz dazu sind es bei dem Regressionsmodell 331. Eine Minderung des *confidence*-Schwellenwertes beim Klassifikationsmodell verbessert nicht das Ergebnis, denn zwischen einer eindeutig erkannten Hand und einer Missklassifikation liegt meist keine Differenz >0,2.

(a) Detektion des Klassifikationsmodells

(b) Detektion des Regressionsmodells

Abbildung 28: Vergleich beider Modelle. Die Vorhersagen des Klassifikationsmodells haben das Format *Klasse: Confidence*.

Die deutlich niedrigeren Konfidenzwerte des Klassifikationsmodell liegen unter anderem an der hohen visuellen Ähnlichkeit benachbarter Klassen. Dies ist beispielhaft in Abb. 29 sichtbar. Um eine Hand werden mehrere benachbarte Klassen mit nahezu gleicher Konfidenz vorhergesagt. Allgemein zeige ein Objektdetektor Schwierigkeiten darin ähnliche Klassen zu unterscheiden [13]. Dies ist besonders der Fall, wenn die Ausrichtung der Hand in der Nähe einer benachbarten Klasse angrenzt.

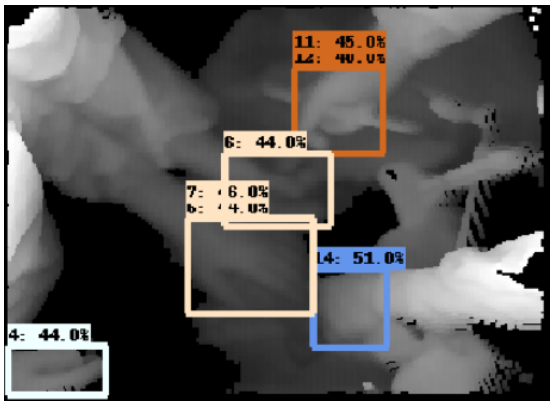


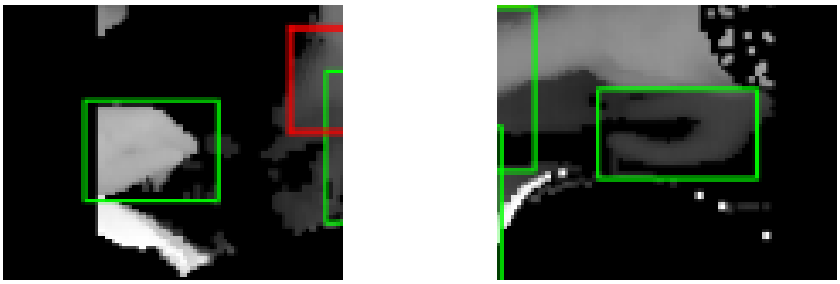
Abbildung 29: Das Regressionsmodell detektiert oft benachbarte Klassen.

Diese Erkenntnis lässt sich im *Recall* der Tabelle 2 wiederfinden. Dabei ist im Bezug zu unserer Situation der *Recall* schwer gewichtet. Denn nach Abschnitt 1 ist einer der Hauptgründe schließlich die Bestimmung von Phasen, in welcher keine Hände in der Nähe des Situs zu sehen sind.

Tabelle 2: Vergleich zwischen den Modellen hinsichtlich der Metriken.

Metrik	AP	Precision	Recall
Klassifikation	69,9%	0,956	0,406
Regression	83,7%	0,864	0,885

Dennoch liefert auch das Regressionsmodell keine idealen Ergebnisse. Dies begründet sich zum Teil durch die Inkonsistenz in der Annotation der Datensätze. Von den 374 gelabelten Händen werden zwar 331 Instanzen richtig erkannt, allerdings trifft das Modell auf dem *test*-Datensatz 52 weitere Vorhersagen, welche nicht den Labels entsprechen. Wie in Abb. 30 zu sehen, bilden viele dieser FP tatsächlich Hände ab, die im Annotationsprozess nicht gelabelt wurden. Oft sind auf den Bildern nur Teile von Händen zu sehen. Wenn ein überwiegender Anteil verdeckt ist, dann wird diese Hand bewusst nicht gelabelt, da zu wenige Informationen über die Ausrichtung gegeben sind. Dies ist aber, wie in der Abbildung gezeigt, von Fall zu Fall schwer zu beurteilen. Des Weiteren ist es auch nicht auszuschließen, dass Hände schlichtweg übersehen werden.



(a) Verdeckt durch den Rahmen (b) verdeckt durch andere Objekte

Abbildung 30: Vom Regressionsmodell erkannte Hände, welche nicht in der *Groundtruth* enthalten sind.

5.2. Bestimmung der Ausrichtung

In der Bestimmung der Ausrichtung ist eine Angabe eines Wertes, wie etwa die Durchschnittsabweichung nicht aussagekräftig genug. Beispielsweise ist es möglich, dass die vorhergesagten Richtungen stark um den Durschnitt streuen, sodass es nie zu einer stabilen Erkennung über einen Zeitraum kommt. Zusätzlich kommt hinzu, dass eine vorhergesagte Klasse des Klassifikationsmodells einen Bereich abdeckt und daher viele Metriken nicht ohne Annahmen berechnet werden können. Deutlich aufschlussreicher ist die Verteilung einzelner Abweichungsklassen. Als mögliche Aufteilung bieten sich die Bereiche $\leq 10°$, $10° < a \leq 20°$, $20° < a \leq 30°$, $30° < a \leq 90°$ und $> 90°$ an, bei dem a die Abweichung der Ausrichtung darstellt. Die erste Klasse stellt die Vorhersagen dar, welche als nahezu abweichungslos betrachtet werden können. Die zweite Klasse deckt den Anteil der Vorhersagen ab, deren Abweichung zwar bemerkbar ist, aber nicht als sonderlich abweichend wahrgenommen werden. Die Klasse $20° < a \leq 30°$ nimmt nun alle Vorhersagen dazu, welche nach den Anforderungen noch ausreichend genau sind. Zu der vorletzten Klasse gehören alle Vorhersagen, welche die richtige vertikale Komponente in Richtung der Hand besitzen. Alle anderen von dem Modell vorgeschlagenen Richtungen werden in die letzte Kategorie eingeordnet. Die Verteilung der Abweichungen ist in Tabelle 3 abgebildet.

Tabelle 3: Verteilung der Abweichungen in der Bestimmung der Rotation. Alle Vorhersagen müssen mit der Groundtruth eine IoU von über 0,5 haben.

Modelltyp	$a \leq 10°$	$10° < a \leq 20°$	$20° < a \leq 30°$	$30° < a \leq 90°$	$> 90°$
Klassifikation	68,42%(104)	-	29,61%(45)	0%(0)	1,97%(3)
Regression	70,39%(233)	22,66%(75)	4,53%(15)	2,11%(7)	0,30%(1)

Beim Klassifikationsmodell wird festgelegt, dass eine prognostizierte Ausrichtung sich immer in der Mitte der zugeordneten Spanne (Größe: 20°) befindet. So berechnet sich eine Abweichung von $\leq 10°$, auch wenn sich die tatsächliche Ausrichtung an den Grenzen der vorhergesagten Klasse befindet. Mit einer weiteren Klasse rechts und links neben der tatsächlichen Klasse ist der Bereich abgedeckt, welcher den akzeptierten Bereich $\leq 30°$ entspricht (in Abb. 3 dargestellt).

Im Gegensatz zu der Detektion verhalten sich beide Modelle zur Bestimmung der Orientierung sehr ähnlich. Beide Modelle bestimmen nahezu alle (Klassfikationsmodell: 98,03%, Regressionsmodell: 97,58%) ihnen vorgelegten Detektionen in der maximal akzeptierten Abweichung von $\leq 30°$. Der Großteil des Rests wurde von dem Regressionsmodell mit einer Abweichung $\leq 90°$ bestimmt. Das Klassfikationsmodell trifft dagegen einige Vorhersagen (1,97%), die noch weiter außerhalb liegen.

Zu beachten ist, dass sich die Angaben nur auf die Ausrichtung aus der Vogelperspektive beziehen. Wie in Abb. 31 zu sehen, wird die Ausrichtung invariant zur Rotation der eigenen Achse der Hand bestimmt. Es ist erkennbar, dass das Modell die Inkohärenz zwischen der Ausrichtung und der Rotation der eigenen Achse aus den Trainingsdaten erlernt.

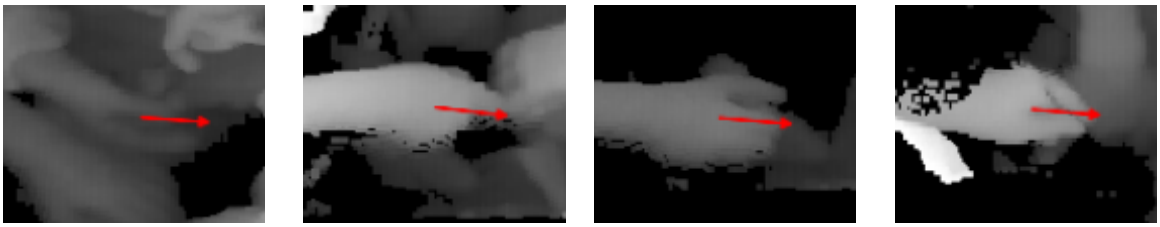


Abbildung 31: Vorhersagen der Ausrichtung im Bereich [94°; 97°]. Die Hände unterscheiden sich in der eigenen Achse.

Ähnlich zu der Situation bei der Detektion, ist zu bedenken, dass die Annotation nicht überall konsistent gehalten ist. In der Regel legt der Weg von der Mittelhand zum Zeigefinger die Richtung fest, dennoch ist es schwierig genau diese Ausrichtung festzulegen. So sind kleinere Abweichungen in der Orientierung beim Labeln kaum bemerkbar und kommen häufig vor.

Trotz der eben genannten Vorgabe, die Ausrichtung der Hände beim Labeln nach einem Prinzip zu bestimmen, kommt es regelmäßig vor, dass diese bewusst nicht eingehalten wird. Die Abb. 32 zeigt ein Ausschnitt aus einem Bild des *test*-Datensatzes, bei dem eine Person ein schmales Operationsutensil hält. Es lässt sich aus dem Kontext erahnen, dass der Fokus der Person in Richtung des Utensils positioniert ist und somit wurde dementsprechend die Ausrichtung gelabelt.

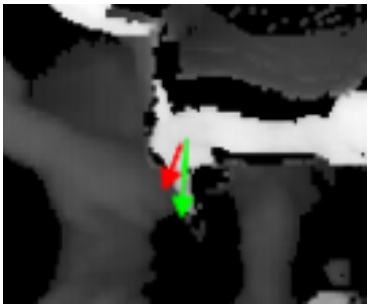


Abbildung 32: Die Groundtruth zeigt eindeutig in Richtung Süden (grüner Pfeil). Das Modell erkennt dies nicht richtig.

5.3. Dauer der Inferenz

Die letzte bedeutende Messgröße ist die benötigte Zeit für die Inferenz. Für das Vorhersagen auf den 150 Bildern des *test*-Datensatzes wird von dem Klassifikationsmodell eine Zeit von $t_{Klas} \approx 8,2998\,s$ und bei dem Regressionsmodell eine Zeit von nur $t_{Regr} \approx 1,4987\,s$ benötigt. Durchschnittlich ergibt das eine Zeit pro Erkennung von $\overline{t_{Klas}} \approx 55,3\,ms$ und $\overline{t_{Regr}} \approx 10\,ms$. Da sich die Auflösung der Bilder und die Topologie der neuronalen Netze nicht während der Erkennung ändern, ist anzunehmen, dass die Detektionszeit pro Bild nur sehr leicht variiert.

Ein Vergleich zwischen Werten ist nicht ganz aussagekräftig, denn das Klassifikationsmodell ist in der Struktur deutlich größer als das Regressionsmodell. Die ODAPI bietet das Modell „SSDw/Resnet50“ nur mit einer Eingabegröße von 640×640 an. Im Gegensatz dazu ist die Größe der Eingabe des Klassifikationsmodell 300×300. Die Eingabegröße beeinflusst deutlich die benötigte Laufzeit der Inferenz [27]. Des Weiteren ist der *Feature-Extractor* beider Modelle unterschiedlich. Der *MobileNet Feature-Extractor* generiert die *Feature-Maps* in einer niedrigeren Laufzeit als dem *Resnet50 Feature-Extractor*. Allerdings sind Objekterkennungssysteme, welche den *Resnet50 Feature-Extractor* benutzen, genauer in der Erkennung [4]. Es ist vorstellbar, dass sich die Laufzeiten ähneln würden, wenn die Eingabegröße und die Wahl des *Feature-Extracors* gleich wären. Schließlich sind beide Netze sehr ähnlich aufgebaut.

In diesem Fall ist die Verwendung des Regressionsmodells hinsichtlich der Laufzeit attraktiver. Doch bevor die Bilder als Eingabe in das Regressionsmodell hineingegeben werden, müssen diese auf eine Größe von 300×300 angepasst werden. Dieser Aufwand ist hingegen nicht bei dem Klassifikationsmodell notwendig und muss bei einer möglichen Integration in das Programm ASuLa mitberechnet werden. Pro Bild verläuft sich dieser Aufwand auf unbemerkbare $\overline{t_{res}} \approx 0,2\,ms$ und ist daher nicht der Berücksichtigung wert.

Das Klassifikationsmodell ist in diesem Fall nicht in der Lage während einer Aktualisierung des Bildes eine Vorhersage zu treffen. Zwar werden für die Laufzeittests keine leistungsstarke Hardware verwendet, dennoch ist zu bedenken, dass während eines Aktualisierungsschrittes des Programmes ASuLa weitere Berechnungen ausgeführt werden.

5.4. Folgerung der Ergebnisse

Die Kombination aus den oben gewonnenen Erkenntnissen und der Evaluation hebt zweifelsfrei die Verwendung des Regressionsmodells hervor. Im Zuge der Anforderung „Kompatibilität mit C++“ stellt Pytorch eine Konversion des Modells mithilfe des *Torch Scripts* zur Verfügung. Diese Repräsentation des Modells erlaubt durch den *Torch Scripts Compiler* die Verwendung in der Umgebung von C++.

Aus Testzwecken wird für einen fairen Vergleich zwischen den beiden Modellen der *Confidence*-Schwellenwert auf 0,4 gesetzt. Für das Regressionsmodell kann es hinsichtlich der *Precision* und *Recall* besser aussehen, wenn der Schwellwert angepasst wird. In Abb. 33 ist zu sehen, dass dies nicht der Fall ist, weil sich die Metriken im Bereich [0,3; 0,7] des Schwellwerts sehr linear verhalten. Da in dieser Situation der *Recall* höher als die *Precision* gewichtet ist, ist die Wahl des Wertes 0,4 bei dem Regressionsmodell gerechtfertigt.

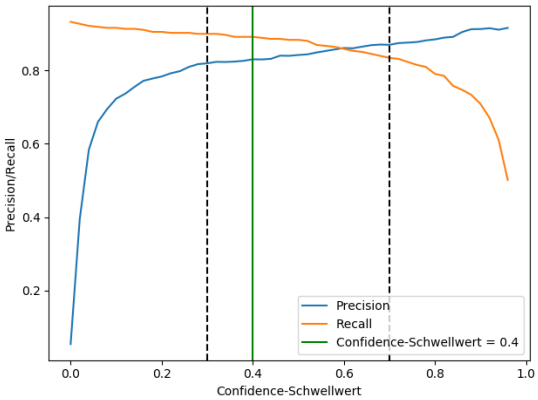


Abbildung 33: *Recall* und *Precision* in Relation zum *Confidence*-Schwellenwert. Die schwarzen gestrichelten Linien deuten die Werte 0,3 und 0,7 an.

Zu guter Letzt ist anzumerken, dass die gemessenen Werte nur unter den Annahmen der vorgegebenen Umgebung entstanden sind. Obwohl der *test*-Datensatz Bilder enthält, die das Modell vorher noch nicht gesehen hat, kann nicht garantiert werden, dass sich das Modell in realen Anwendungsfällen genau so leistungsfähig wie gemessen verhält. Die gegebenen Aufnahmen sind die Quelle des Trainings- und Testmaterial und decken nicht alle möglichen Szenarien im Bereich der Bauchchirurgie ab. Viele weitere Instrumente und andere Geräte sind nicht in den Aufnahmen enthalten und können beispielsweise eine Verwechslungsgefahr zu Händen darstellen. Des Weiteren ist das Modell durch die Trainingsdaten an individuelle Details, wie beispielsweise der Gestik des OP-Personals angepasst. Wie sich das Modell bei anderem OP-Personal verhält, lässt sich nur mit einer weiteren Untersuchung klären. Ein geänderter Aufbau des OP-Saals hingegen, wie etwa eine Rotation der Kamera oder ähnliches, stellt aufgrund der detailreichen Augmentation der Trainingsdaten kein Problem dar.

6. Fazit und Ausblick

6.1. Fazit

Die Ergebnisse dieser Arbeit stellen ein Detektionsmodell in Form eines CNNs vor, welches in Tiefenbildaufnahmen im Bereich der Chirurgie zuverlässig Hände erkennt und deren Ausrichtung bestimmt. Dabei erreicht das Modell die in Tabelle 4 dargestellte Leistung. Die Ausrichtung wird durch die Vorhersage der Längen von Vektoren einer Hand-Vektoren-Darstellung (zu sehen in Abb. 2) prognostiziert. Die benötigte Zeit für die Inferenz ist dabei niedrig und ermöglicht somit die Erkennung in Echtzeit.

Tabelle 4: Gemessene Ergebnisse des resultierenden Modells.

Messgröße	Wert
Precision	0,864
Recall	0,885
Average Precision (AP)	83,7%
Anteil akzeptabler Abweichung der Ausrichtungen	97,58%
Durchschnittliche Laufzeit	0,01s
confidence Schwellwert	0,4

Neben dem Aufbau und der Evaluation des Modells, steht auch vor allem die Frage, in welcher Form die Tiefenbildaufnahmen verwendet werden, im Fokus. Die Literatur umfasst nur Ansätze zu dem Thema Erkennung und Bestimmung der Orientierung von Objekten in RGB-Daten. Eine angepasste Datenvorverarbeitung ermöglicht dies, welche folgendermaßen aufgebaut ist. Zuerst werden überflüssige Bereiche ausgeschnitten und die übrig gebliebenen Bereiche normalisiert. Im Anschluss verwendet man Dilatation, um die unreine Struktur zu beseitigen. Zum Schluss bringt eine Augmentation der Daten mehr Vielfalt in den Datensatz.

Aus dem Vergleich zu einer anderen Methode zur Bestimmung der Orientierung basierend auf Klassifikation stellt sich heraus, dass das Modell mithilfe von Regression der Vektoren in jeder Messgröße besser abschneidet. Die Architektur des Regressionsmodells basiert nach Yang et al. auf einem SSD, der durch die *Top-Down Fusion Architecture* mit dem MobileNet *Feature-Extractor* verbunden ist [44]. Die Ergebnisse der Arbeit entsteht aus folgender Trainingskonfiguration:

- Batchgröße: 16
- Schrittanzahl: 15000
- Verlustfunktion: Gemittelter *Loss* der *Loss*-Komponenten (siehe Formel 4.5)
- Lernrate: 10^{-3}
- Aktivierungsfunktion: RELU_6

Aus dem Projekt wurde deutlich, dass bestehende Methoden der Erkennung auf RGB-Bildern auch auf Abbildungen von Punktwolken eingesetzt werden können. Dabei bieten CNNs dank schneller Architekturen sowohl eine hohe Genauigkeit, als auch eine niedrige Laufzeit. Auch die Technik *transfer learning* stellt sich als sehr effektives Verfahren heraus, obwohl die vortrainierten Gewichte auf RGB-Daten trainiert werden. Zurückblickend auf die gestellten Anforderungen aus dem Kapitel 1.2 ergibt sich das Detektionsmodell als eine angemessene Erweiterung des Programms ASuLa.

6.2. Ausblick

Bevor die Methode endgültig als Realisierung des Lösungsansatzes vorgeschlagen wird, lohnt sich eine Betrachtung entstandener Gedanken und weiterhin bestehenden Forschungsbedarf.

Aufgrund der Hardwarelimitierung kann das Klassifikationsmodell nicht optimal trainiert werden, was die Ursache für die mangelhafte Detektion von Händen sein kann. Ein erneuter Trainingsprozess des Klassifikationsmodell mit stärkerer Hardware kann mehr Aufschluss über die Tatsache geben, sodass eine weitere Untersuchung lohnenswert erscheint. Das Klassifikationsmodell zeigt überraschenderweise bei der Bestimmung der Orientierung auch unter den schlechten Umständen ein gutes Ergebnis. Daher kann in Kombination einer verlässlichen Detektion von Händen auch die Verwendung des Klassifikationsmodell infrage kommen. Gleichzeitig bietet sich für ein erneutes Training das Ausschneiden des Rahmens (siehe Abschnitt 4.2.2) und die Reduzierung der Farbkanäle des Netzes (siehe Abschnitt 3.2.2) an. Dadurch erleichtert sich das Trainieren der Modelle deutlich.

Während der Untersuchung entwickelten sich Ideen, in welcher Form das Modell genutzt werden kann. Wie in Kapitel 1 erwähnt, ist die offensichtliche Anwendung des Modells die Berechnung eines möglichen Fokusses des OP-Personals, welche die Lampen autonom ansteuern sollen. Ein möglicher Prototyp berechnet pro Detektion den Punkt über die Position und Ausrichtung der Hand. Ein naiver Ansatz der Berechnung ist in 6.1 definiert.

$$foc(orient, pos) = (x_{foc}(orient, pos), y_{foc}(orient, pos)) \tag{6.1}$$

wobei gilt:

$$x_{foc}(orient, pos) = 1,5(x(orient)) + x(pos) \tag{6.2}$$
$$y_{foc}(orient, pos) = 1,5(y(orient)) + y(pos) \tag{6.3}$$

Die Funktionen $x()$ und $y()$ bilden Vektoren und Punkte auf die x-, beziehungsweise y-Werte ab. Der Vektor *orient* stellt den Ausrichtungsvektor und *pos* die Handposition dar.

Dieser Algorithmus kann zusätzlich mit Heuristiken erweitert werden, um die Stabilität der Erkennung zu erhöhen. Beispielsweise kann eine maximale Differenz der Ausrichtung einer Hand auf nachfolgenden Bildern festgelegt werden. Dies verhindert starke Schwankungen in der Bestimmung der Ausrichtung. Als Erweiterung der oben genannten Idee ist neben der Erkennung der Position und Ausrichtung auch die Bestimmung der Haltung von Händen möglich. Mit dieser zusätzlichen Information kann zu jeder Hand auch die Signifikanz der Hand bestimmt werden. Zum Beispiel wird eine Hand besonders wichtig gewertet, wenn diese in der Nähe des Situs ein Skalpell hält. In Kombination mit einer Vorhersage des Fokuspunktes, ist es möglich nur den Punkt anzuvisieren, welche für das Modell am bedeutendsten gehalten wird. Ein möglicher Prototyp ist in Abb. 34 zu sehen.

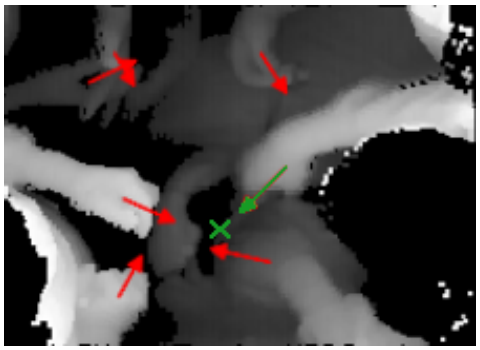


Abbildung 34: Möglicher Aufbau eines Prototyps. Die Hand mit dem grünen Pfeil hält ein Skalpell und ist deswegen besonders signifikant. Das grüne Kreuz stellt den berechneten Fokuspunkt dar.

Das Projekt hat gezeigt, wie aufwendig es ist, die Aufnahmen konsistent zu labeln. Kleinere Abweichungen in der Position und der Ausrichtung sind durchgängig vorhanden und beeinflussen damit die Qualität der Daten ein. Es ist erwartbar, das die Genauigkeit steigt, wenn die Qualität der Daten durch robustere Labels steigt, auch wenn dieser Unterschied vermutlich marginal ausfällt. Ein neuer Labelprozess kann mit Berücksichtigung der Genauigkeit auch mit einem diverseren Datensatz verbunden werden, welcher im Gegensatz zu dieser Situation unterschiedliche Operationen aufzeichnet. Eine größere Vielfalt der Daten deckt mehr mögliche Situationen ab und wirkt sich positiv auf die Beständigkeit der Erkennung aus.

7. Anhang

Im Anhang befinden sich in digitaler Form auf dem beigelegten Speichermedium:

- Die resultierenden Modelle
- Die Datensätze (*train, validation, test*)
- Die Konfigurationsdatei

A. Akronyme

AABB Axis-Aligned-Bounding-box

AP Average Precision

CNN Convolutional Neural Network

Faster R-CNN Faster Region-based Convolutional Neural Networks

FN False Negative

FP False Positive

MAP Mean Average Precision

MS-RFCN Multiple Scale Region-based Fully Convolutional Networks

NN Neuronales Netz

OBB Oriented Bounding Box

ODAPI Tensorflow Object Detection API

R-FCN Region-based fully Convolutional Networks

SSD Single Shot Detector

TN True Negative

TP True Positive

Abbildungsverzeichnis

Tabellenverzeichnis

B. Literatur

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software erhältlich unter tensorflow.org.

[2] I. Athanasiadis, P. Mousouliotis, and L. Petrou. A framework of transfer learning in object detection for embedded systems. *CoRR*, abs/1811.04863, 2018.

[3] R. Becker. Aufbau von convolutional neural networks (cnn). https://jaai.de/convolutional-neural-networks-cnn-aufbau-funktion-und-anwendungsgebiete-1691/, 2019. Zuletzt geprüft am 13.10.2020.

[4] S. Bianco, R. Cadene, L. Celona, and P. Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018.

[5] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 379–387. Curran Associates, Inc., 2016.

[6] X. Deng, Y. Zhang, S. Yang, P. Tan, L. Chang, Y. Yuan, and H. Wang. Joint hand detection and rotation estimation using cnn. In *IEEE Transactions on Image Processing*, pages 1888–1900. IEEE, 2018.

[7] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *ArXiv*, abs/1603.07285, 03 2016.

[8] M. Everingham, S. Eslami, L. Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111:98–136, 2015.

[9] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter, and R. Siegwart. Kinect v2 for mobile robot navigation: Evaluation and modeling. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 388–394, 2015.

[10] K. Hara, R. Vemulapalli, and R. Chellappa. Designing deep convolutional neural networks for continuous object orientation estimation. *ArXiv*, 1702.01499, 2017.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2016.

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[13] B. (https://stats.stackexchange.com/users/173082/ben). Do more object classes increase or decrease the accuracy of object detection. Onliune unter: https://stats.stackexchange.com/q/349105, 2018. zuletzt geprüft: 15.10.2020.

[14] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3296–3297, 2017.

[15] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3296–3297, 2017.

[16] International Electrotechnical Commission. Particular requirements for the basic safety andessential performance of surgical luminaires and luminaires for diagnosis. Technical Report 60601-2-41, IEC, 2009.

[17] Itseez. Open source computer vision library. https://github.com/itseez/opencv, 2015. Zuletzt geprüft am 17.10.2020.

[18] B. Kang, Y. Lee, and T. Nguyen. Efficient hand articulations tracking using adaptive hand model and depth map. In *International Symposium on Visual Computing*, pages 586–598, 2015.

[19] B. Kang, K.-H. Tan, H.-S. Tai, D. Tretter, and T. Nguyen. Hand segmentation for hand-object interaction from depth map. *CoRR*, 03 2016.

[20] A. Khan, A. Sohail, U. Zahoora, and A. Saeed. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, pages 5455–5516, 2020.

[21] A. Knulst, J. Dongen, M. Groenewegen, E. Kaptein, and J. Dankelman. The effect of shadows on performing stereo visual pointing tasks: Is shadow-free open surgery ideal? *LEUKOS: The Journal of the Illuminating Engineering Society of North America*, 8:111–122, 09 2013.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[23] E. Lachat, H. Macher, M.-A. Mittet, T. Landes, and P. Grussenmeyer. First experiences with kinect v2 sensor for close range 3d modelling. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5/W4, 2015.

[24] T. H. N. Le, K. G. Quach, C. Zhu, C. N. Duong, K. Luu, and M. Savvides. Robust hand detection and classification in vehicles and in the wild. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1203–1210. IEEE, 2017.

[25] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, 2017.

[26] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context. In *Computer Vision – ECCV 2014*, pages 740–755. Springer, Cham, 2014.

[27] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[28] H. Mate Labs. Everything you need to know about neural networks. Onliune unter: https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491, 2017. Zuletzt geprüft am 11.10.2020.

[29] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[30] P. Panteleris, I. Oikonomidis, and A. Argyros. Using a single rgb frame for real time 3d hand pose estimation in the wild. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 436–445. IEEE, 2018.

[31] M. Park, M. Hasan, J. Kim, and O. Chae. Hand detection and tracking using depth and color information. In *Proceedings of the International Conference on Image*

Processing, Computer Vision, and Pattern Recognition (IPCV), pages 779–786, 07 2012.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[33] C. Qian, X. Sun, Y. Wei, X. Tang, and J. Sun. Realtime and robust hand tracking from depth. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1106–1113, 2014.

[34] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[35] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[36] P. Soille. Erosion and dilatation. In *Morphological Image Analysis*, chapter 3, pages 63–103. Springer, Berlin and Heidelberg, 2004.

[37] S. Tambe, D. Kulhare, M. D. Nirmal, G. Prajapati, and M. Pune. Image processing (ip) through erosion and dilation methods. *Study and Implementation of Heterogeneous Protocols on Wearable Body Sensor Network in E-Healthcare Monitoring System*, Volume 3:285–289, 2008.

[38] J. Teuber, R. Weller, R. Kikinis, K.-J. Oldhafer, M. J. Lipp, and G. Zachmann. Optimized positioning of autonomous surgical lamps. In R. J. W. III and B. Fei, editors, *Medical Imaging 2017: Image-Guided Procedures, Robotic Interventions, and Modeling*, volume 10135, pages 279 – 289. International Society for Optics and Photonics, SPIE, 2017.

[39] J. Teuber, R. Weller, R. Kikinis, K.-J. Oldhafer, and G. Z. Michael J. Lipp. Autonomous surgical lamps. Technical report, Universität Bremen and Asklepios Klinik Barmbek, 2015.

[40] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 839–846, 1998.

[41] Wikimedia Commons. maxpooling with 2x2 filter and stride = 2. Onliune unter: https://upload.wikimedia.org/wikipedia/commons/e/e9/Max_pooling.png, 2015. Zuletzt geprüft am 11.10.2020.

[42] Wikimedia Commons. typical cnn architecture. Onliune unter: https://upload.wikimedia.org/wikipedia/commons/6/63/Typical_cnn.png, 2015. Zuletzt geprüft am 11.10.2020.

[43] W. Xu, I.-S. Lee, S.-H. Lee, B. Lu, and E.-J. Lee. Multiview-based hand posture recognition method based on point cloud. *KSII Transactions on Internet and Information Systems*, 9:2585–2598, 07 2015.

[44] L. Yang, Z. Qi, Z. Liu, S. Zhou, Y. Zhang, L. Hao, J. Wu, and L. Shi. A light cnn based method for hand detection and orientation estimation. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2050–2055. IEEE, 2018.

[45] X. Yang, H. Sun, X. Sun, M. Yan, Z. Guo, and K. Fu. Position detection and direction prediction for arbitrary-oriented ships via multitask rotation region convolutional neural network. *IEEE Access*, 6:50839–50849, 2018.