



OTTO VON GUERICKE
UNIVERSITÄT
MAGDEBURG

INF

FAKULTÄT FÜR
INFORMATIK

Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik

Verwendung von Deep Learning Techniken zur
Entwicklung einer grafisch basierten
Steckplatzanalyse-App für Serverschränke

Bachelorarbeit

Autor:

Martin Weiss

Betreuende:

Prof. Dr. rer. nat. Frank Ortmeier

M.Sc. Konstantin Kirchheim

Institut für Intelligente Kooperierende Systeme (IKS)
Chair of Software Engineering

Magdeburg, 18.07.2022

Weiss, Martin:

Verwendung von Deep Learning Techniken zur Entwicklung einer grafisch basierten Steckplatzanalyse-App für Serverschränke

Bachelorarbeit, Otto-von-Guericke-Universität Magdeburg, 2022.

Inhaltsangabe

In einer Welt des exponentiellen Datenwachstums sind die Begriffe *ubiquitous computing* oder *persuasive computing* sowohl für Industrie als auch für Verkehr, Gesundheit und Gesellschaft nicht mehr wegzudenken. Für die Vernetzung der stetig wachsenden Anzahl smarter Geräte, den Wunsch des autonomen Fahrens oder bei der Schaffung eines *Meta-versums*, dessen Potenzial sich in der aktuell andauernden COVID-19-Pandemie zeigt, sind gewisse Anforderungen, die durch den Telekommunikationssektor erfüllt werden müssen, unabdingbar. Darunter sind vordergründig das Verfügen über flächendeckend hohe Bandbreiten und das Gewährleisten geringer Latenzen bei der Datenübertragung von Relevanz. Der Ausbau des Mobilfunksystems der 5. Generation stellt für die Erreichung dieser Anforderungen eine Schlüsselkomponente dar. Dieser bezieht den Einsatz von verteilten Mini-Rechenzentren mit ein, welche die Rechenleistung näher an den physischen Standort der Nutzer:innen bringen soll.

Daraus resultiert ein enormer Verwaltungsaufwand bei der Überprüfung und Instandhaltung der Servertechnik, den es mithilfe der Digitalisierung zu minimieren gilt.

In dieser Bachelorarbeit soll ein prototypisches Verfahren entworfen werden, welches die automatische Analyse eines Serverschranks per Smartphone-Kamera ermöglicht und somit eine effiziente und einheitliche Dokumentationsmöglichkeit untersucht. Dafür sollen gegenwärtige *Deep Learning*-Objekterkennungsalgorithmen verglichen und hinsichtlich der Problemstellung eingesetzt werden. Abschließend wird das entwickelte Verfahren durch eine Evaluierung auf dessen Korrektheit und Genauigkeit überprüft. Basierend darauf lässt sich die Verwendbarkeit des Verfahrens einschätzen und einen Anstoß für mögliche Weiterentwicklungen geben.

Danksagung

An dieser Stelle möchte ich mich bei den Kollegen und Kolleginnen der Firma *highstreet technologies* bedanken, die es mir ermöglicht haben, ein für mich sehr interessantes Thema zu bearbeiten. Neben der finanziellen Förderung waren vor allem die bereitgestellte und zum Teil extra angeschaffte Hardware sowie die durchgehende Hilfsbereitschaft eine große Unterstützung.

Ein weiterer Dank gilt Prof. Dr. rer. nat. Frank Ortmeier und Konstantin Kirchheim für die Begutachtung und die zuverlässige Betreuung meiner Bachelorarbeit. Sowohl bei spontanen Fragen als auch bei geplanten Besprechungen habe ich aufgrund der durchgängigen Auskunftsbereitschaft, wichtige Anmerkungen und Hinweise für die Bearbeitung bekommen.

Letztlich möchte ich noch ein gesondertes Dankeschön an meine Familie und an meine Freundin richten, die mir in einer turbulenten Zeit eine sehr große Stütze waren, *sind* und **bleiben**.

Inhaltsverzeichnis

Abbildungsverzeichnis	ix
Tabellenverzeichnis	xi
Quelltext- und Algorithmusverzeichnis	xiii
Abkürzungsverzeichnis	xv
1 Einführung	1
2 Grundlagen und wissenschaftliche Einordnung	5
2.1 Computer Vision	5
2.1.1 Das digitale Bild	5
2.1.2 Operationen auf digitale Bilder	6
2.1.3 Die Teilaufgaben der Bilderkennung	7
2.2 Deep Learning	9
2.2.1 Künstliche Neuronale Netze	9
2.2.2 Training	11
2.2.3 Hyperparameter	13
2.2.4 Generalisierung	15
2.2.5 Evaluierung eines Deep Learning Verfahrens	16
2.3 Convolutional Neural Networks	19
2.3.1 Funktionsweise	19
2.3.2 Faltung, Kreuzkorrelation	20
2.3.3 Convolutional Layer	21
2.3.4 Pooling	22
2.3.5 Bedeutende CNN-Architekturen	22
2.4 Transfer Learning	24
2.4.1 Merkmalsextraktion	24
2.4.2 Feintuning	24
3 Stand der Technik in der Objekterkennung	25
3.1 Object Detection Challenges	25
3.1.1 VOC	25
3.1.2 ILSVRC	26
3.1.3 MS-COCO	26
3.1.4 OID	26
3.2 Zweistufige Detektoren	27
3.2.1 R-CNN	27
3.2.2 Fast R-CNN	28
3.2.3 Faster R-CNN	30

3.2.4	Mask R-CNN	32
3.3	Einstufige Detektoren	34
3.3.1	YOLO	34
3.3.2	Weitere einstufige Detektoren	37
4	Konzept	39
4.1	Analyse der Zielstellung	39
4.1.1	Rack, Maße und Komponenten	39
4.1.2	Exemplarische Steckplatzanalyse	41
4.2	Lösungsstrategie	41
4.2.1	Steckplatzerkennung	42
4.2.2	Postprocessing der Steckplatzerkennung	43
4.2.3	Micro-TSS-App-Integration	46
5	Umsetzung	47
5.1	Implementierung der Steckplatzerkennung	47
5.1.1	Datensatzerstellung	47
5.1.2	Framework und Hardware	52
5.1.3	Implementierung des Mask-RCNN-Detektors	52
5.1.4	Implementierung des YOLOv5-Detektors	59
5.2	Implementierung des Postprocessings	63
5.3	Einbindung in die Micro-TSS-App	64
5.3.1	Server	65
5.3.2	Client	67
6	Evaluierung	69
6.1	Daten der Testszenarien	69
6.1.1	Datenakquise	69
6.1.2	Analyse der Evaluierungsdaten	71
6.2	Evaluierungsstrategie	72
6.2.1	Vorgehen	72
6.2.2	Evaluierungsmetriken	73
6.2.3	Gütekriterium eines Evaluierungsergebnisses	76
6.2.4	Umsetzung der Evaluierung	76
6.2.5	Parameter der Evaluierung	77
6.3	Evaluierungsergebnisse	77
6.3.1	Ergebnisse basierend auf den ursprünglichen Parametern	77
6.3.2	Einfluss der Konfidenz-Score-Schwellenwerte	79
6.3.3	Einfluss der PPMR-Konstante	80
6.3.4	Einfluss des Freiflächen-Schwellenwerts	80
6.4	Auswertung der Ergebnisse und mögliche Fehlerquellen	81
7	Zusammenfassung und weitere Forschungsmöglichkeiten	85
A	Anhang	87
	Literaturverzeichnis	95

Abbildungsverzeichnis

2.1	Beispiele für unterschiedliche Bildmatrizen der Dimension 8×8	6
2.2	Grundlegende Transformationen	7
2.3	Resultate unterschiedlicher Bilderkennungsaufgaben	7
2.4	Das Perzeptron	10
2.5	Darstellung eines allgemeinen r -schichtigen Perzeptrons	11
2.6	Bedeutende Aktivierungsfunktionen	14
2.7	Visualisierung einer <i>Intersection over Union</i> Berechnung	17
2.8	Faltung an einem Beispiel	21
2.9	Max-Pooling	22
2.10	Architektur des LeNet-5	23
2.11	Der Residual Block	23
3.1	Beispiele aus den Objekterkennungs-Challenges	27
3.2	Die R-CNN Architektur	28
3.4	Ablauf des RoI Poolings	29
3.3	RoI projiziert auf eine Merkmalskarte	29
3.5	Die Fast R-CNN Architektur	30
3.7	Die Faster R-CNN Architektur	31
3.6	Ankerboxen des RPNs	31
3.8	R-CNN, Fast R-CNN und Faster R-CNN im Vergleich	32
3.9	Ablauf des RoI-Aligns	33
3.10	Die Mask R-CNN Architektur	33
3.11	Der YOLO Erkennungsprozess	35
3.12	Aufbau des YOLO-CNNs	36
3.13	YOLO im Vergleich mit den R-CNN-Varianten	37
4.1	Visualisierung der Maße eines Steckplatzes	40
4.2	Eine exemplarische Steckplatzanalyse	41
5.1	Darstellung der Benutzeroberfläche von <i>labelme</i>	49
5.2	Das Resultat der perspektivischen Transformation	50
5.3	Analyse des Datensatzes	52
5.4	Augmentationen für das Training von Mask R-CNN am Beispiel	54
5.5	Metriken und adaptive Lernrate im Verlauf des Mask R-CNN Trainings	57
5.6	Die letzten zehn Epochen des Mask R-CNN Trainings	57
5.7	Metriken während des 300-epöchigen Mask R-CNN Trainings	58
5.8	Augmentationen für das Training von YOLOv5 am Beispiel	61
5.9	Adaptive Lernraten für YOLOv5	62
5.10	Ausgabe der Fehlerfunktionen im Verlauf des YOLOv5 Trainings	62
5.11	Mean Average Precision von YOLOv5 im Trainingsverlauf	62
5.12	Algorithmischen Teilschritte der Steckplatzanalyse	64

5.13	Bildschirmfotos aus der Micro-TSS-APP	68
6.1	Rackauslastung der Test-Szenarien	71
6.2	Häufigkeitsverteilung der Steckplatztypen im Evaluierungsdatensatz . . .	72
A.1	Steckplatzanalyse-System	87
A.2	Bildschirmaufnahme einer Evaluierungs-Steckplatzanalyse	92
A.3	Visualisierung einer fehlerhaften Verzerrung der Seitenverhältnisse	94
A.4	Häufigkeitsverteilung der überlagerten Steckplätze	94

Tabellenverzeichnis

3.1	Faster R-CNN im Vergleich zu Mask R-CNN auf die MS-COCO-Objekterkennungs-Challenge	33
5.1	Darstellung der im Datensatz annotierten Klassen.	49
5.2	Die annotierten Klassen des vorverarbeiteten Datensatzes	51
5.3	Die annotierten Klassen des vorverarbeiteten Datensatzes inklusive dessen Häufigkeit	52
5.4	Die besten Trainingsepochen mit Mask R-CNN	58
5.5	Die beste Trainingsepoche mit YOLOv5	61
5.6	Tabellarische Darstellung einer Steckplatzanalyse	65
6.1	Die Evaluierungsergebnisse basierend auf den ursprünglichen Parametern	78
6.2	Der Einfluss des Konfidenz-Score-Schwellenwerts auf die Evaluierungsergebnisse	80
6.3	Evaluierungsergebnisse basierend auf einem alternativen perspektivischen Transformationsverfahren	82
A.1	Tabellarische Darstellung eines Evaluierungsergebnisses	92
A.2	Der Einfluss der PPMR-Konstante auf die Evaluierungsergebnisse	93
A.3	Der Einfluss des Freiflächen-Schwellenwerts auf die Evaluierungsergebnisse	93

Quelltext- und Algorithmusverzeichnis

5.1	JSON-Schema für die Anfragen ReST-API-Endpunkts	65
5.2	JSON-Schema für die Antworten des ReST-API-Endpunkts	66
A.3	getEnlargedPredictionsSorted	87
A.1	getOptimizedPredictions	88
A.2	getSteckplatzAnalyse	89
A.4	createEmptySlot	90
A.5	createPredictedSlot	90

Abkürzungsverzeichnis

MEC *Multi-Access Edge Computing*

KI *Künstliche Intelligenz*

KNN *künstliches neuronales Netz*

MLP *multilayer perceptron*

AP *Average Precision*

TP *True positive*

TN *True negative*

FP *False positive*

FN *False negative*

IoU *Intersection over Union*

VOC *PASCAL Visual Object Class Challenge*

ILSVRC *ImageNet Large Scale Visual Recognition Challenge*

mAP *mean Average Precision*

COCO *Common Objects in Context*

SGD *stochastic gradient descent*

CNN *Convolutional Neural Network*

RoI *Region of Interest*

RPN *Region Proposal Network*

FPN *Feature Pyramid Network*

RU *Rack Unit*

GPU *Graphics Processing Unit*

PPMR *Pixel per Metric Ratio*

JSON *JavaScript Object Notation*

HTTP *Hypertext Transfer Protocol*

API *Application Programming Interface*

1 Einführung

Dem zukünftigen Markt für *Multi-Access Edge Computing* (MEC) wird ein immenses Potenzial vorausgesagt. Laut einer im Jahr 2020 veröffentlichten Marktforschungsanalyse des Unternehmens *Frost & Sullivan* [Sta22, Val20] soll der MEC-Markt im Jahr 2024 einen Umsatz von 7.23 Milliarden US-Dollar erzielen und somit den Umsatz von 2019 mehr als verhundertfachen.

Dieses Wachstum soll sich aus der Kopplung des MEC's mit dem Mobilfunknetz der 5. Generation und den daraus resultierenden innovativen Möglichkeiten ergeben. MEC beschreibt einen Teilbereich des *Edge Computings*, wobei die Netzwerktechnologie von Mobilfunkbetreibern näher an den Kunden verlagert wird. Dadurch sollen hohe Bandbreiten und geringe Latenzen bei der Datenübertragung gesichert werden [SIDQ17], wodurch sich Techniken wie das vernetzte autonome Fahren, *Smart Manufacturing* oder eine immersive, latenzfreie *Mixed-Reality* realisierbar machen.

Um die Vorteile des MEC's zu nutzen, sollen eine Vielzahl von Mini-Rechenzentren am Rand des Netzwerkes aufgebaut werden. Dieser verteilte Ausbau resultiert in einen enormen Verwaltungs- und Dokumentationsaufwand. Zu jedem Serverschrank der Mini-Rechenzentren muss dessen Hardware und Auslastung durch geschultes Fachpersonal dokumentiert werden, um bei Netzwerkänderungen oder Hardware-Ausfällen entsprechende Anpassungen planbar zu machen. Dabei müssen die Dokumentationen einheitlich ausgeführt und konsistent abgespeichert werden, um eine Vergleichbarkeit dieser zu gewährleisten. Bislang existiert jedoch weder ein einheitlicher Standard noch eine Software, welche den Prozess der Dokumentationserstellung der MEC-Technik vereinfacht. Bisherige Methoden zur automatisierten Überwachung von Servertechnik stützen sich entweder auf fest installierte Kameras [PHP10, EY21] oder montierte Robotersysteme [Oga21] und sind aufgrund des verteilten Aufbaus beim MEC nicht anwendbar. Daraus entstehen hohe Kosten und eine lange Auftragsdauer für die manuellen Dokumentationen durch das Fachpersonal. Diese Faktoren erschweren und verlangsamen den Ausbau des Mobilfunknetzes.

So kommt es, dass im vom Bundesministerium für Bildung und Forschung geförderten Forschungsprojekts *OTB-5G+¹*, neben der einzusetzenden 5G-Technologie, auch ein intelligentes Netzwerkmanagement erforscht werden soll.

Im Rahmen dieses Projekts wurde vom Unternehmen *highstreet technologies²* eine Netzwerk-Management-Software entwickelt, welche unter anderem die Dokumentation von Mobilfunksendestationen per Smartphone-App unterstützt und damit eine einfache und konsistente Dokumentation der Standorte erlaubt.

In Zusammenarbeit mit *highstreet technologies* soll die Funktionalität der App durch diese Bachelorarbeit erweitert werden. Dabei soll ein prototypisches Verfahren entwickelt werden, welches eine automatische Analyse einer Serverschrank-Bestückung per Foto ermöglicht und somit eine Vereinfachungsmöglichkeit für das Problem der teuren und zeitaufwendigen Dokumentationen der MEC-Serverschränke schafft. Dieses Verfahren soll belegte und

¹<https://www.forschung-it-sicherheit-kommunikationssysteme.de/projekte/otb-5g>

²<https://www.highstreet-technologies.com/>

freie Steckplätze eines Serverschranks identifizieren und die Größe der darin montierten Komponenten ermitteln. Aus diesen Informationen kann ein einheitlicher Serverzustand bestimmt werden, der für die Planung von Netzwerkanpassungen notwendig ist.

Da die Problemstellung das Erkennen und Lokalisieren von Servertechnik in Bildern impliziert, werden in dieser Arbeit aktuelle Objekterkennungsstrategien vorgestellt. Aus den Erkenntnissen der Forschung der letzten Jahre geht hervor, dass auf Deep Learning basierende Techniken die zuverlässigsten Ergebnisse bei der Objekterkennung erzielen. Jene Verfahren werden in dieser Arbeit verglichen und gemäß der Zielstellung umgesetzt. Da sich der Ausbau des MEC's zum Erstellungszeitpunkt der Arbeit in dessen Anfangsphase befindet, liegen für die Entwicklung eines auf *Deep Learning* basierenden Verfahrens nicht genügend Daten vor. Aus diesem Grund soll die Machbarkeitsanalyse anhand von herkömmlichen Serverschränken ausgeführt werden. Eine erfolgreiche Umsetzung dieser Problemstellung kann die Grundlage für die Anpassung eines Verfahrens für MEC-Systeme liefern.

Zielstellung der Arbeit

Wie aus der Einführung hervorgeht, soll durch diese Bachelorarbeit ein prototypisches Verfahren entwickelt werden, welches eine automatische Auslastungsanalyse eines Serverschranks (engl. *Rack*) per Foto ermöglicht. Dieses Verfahren ist dabei in die von *highstreet technologies* entwickelte *Micro-TSS-App* einzubinden und das Ergebnis der Analyse darin zu visualisieren.

Die Analyse soll für jeden Steckplatz eines Racks dessen aktuelle Auslastung ermitteln, demnach feststellen, ob ein Steckplatz durch eine Komponente belegt ist.

Gleichzeitig soll für belegte Steckplätze ermittelt werden, wie groß die darin verbauten Komponenten sind, um somit Informationen über die Anzahl von freien und belegten Steckplätzen eines Serverschranks zu erhalten. Dies impliziert, dass das Verfahren ebenfalls die Gesamtanzahl von möglichen Steckplätzen in einem Serverschrank identifizieren muss. Damit eventuelle Anpassung und Veränderungen am Rack geplant werden können, sollen die erkannten Komponenten und freien Steckplätze außerdem der exakten Steckplatzposition im Rack zugeordnet werden.

Da jene Analyse die Auslastung und Eigenschaften jedes Steckplatzes bestimmt, wird diese in dieser Arbeit als „Steckplatzanalyse“ bezeichnet.

Die Steckplatzanalyse muss dabei nicht in Echt-Zeit ablaufen, da der Nutzen der App vorwiegend in der konsistenten und einheitlichen Datenerfassung liegt. Wichtig für den Einsatz einer automatischen Steckplatzanalyse ist dessen Korrektheit und Genauigkeit.

Eine Analyse gilt als korrekt, wenn für jeden Steckplatz des Racks die richtige Auslastung klassifiziert wird und die ermittelten Größen des Racks und seiner Komponenten mit den tatsächlichen Größen übereinstimmen.

Das entwickelte Verfahren ist abschließend anhand von Real-Welt-Test-Szenarien zu evaluieren. Dabei sollen die Abweichungen der Analyse-Ergebnisse zu den tatsächlichen Auslastungszuständen der Test-Szenarien bestimmt werden.

Ziel des durch die Arbeit entwickelten Prototyps ist, dass die Klassifizierung in weniger als 1% der Fälle einen falschen Auslastungszustand eines Steckplatzes voraussagt. Ferner dürfen die prognostizierten Positionen sowie Komponenten- und Rackgrößen durchschnittlich nicht mehr als 0.05 Einheiten vom tatsächlichen Ergebnis abweichen, da-

mit die Verwendbarkeit der analysierten Steckplatzinformationen gewährleistet werden kann.

Struktur der Arbeit

Der Aufbau der Arbeit richtet sich nach folgendem Schema: In [Kapitel 2](#) wird eine wissenschaftliche Einordnung der grafisch orientierten Aufgabe der Bachelorarbeit vorgenommen. Zusätzlich werden darin die Grundlagen des *Deep Learnings* beschrieben, welche für das Verständnis der in dieser Arbeit verwendeten Objekterkennungs-Algorithmen vonnöten ist. Anschließend werden im [Kapitel 3](#) die wichtigsten Objekterkennungs-Algorithmen vorgestellt und miteinander verglichen.

In [Kapitel 4](#) wird die Problemstellung der Arbeit detailliert untersucht und mittels der Informationen aus den vorangegangenen Kapiteln eine umfangreiche Lösungsstrategie definiert. Im darauffolgenden [Kapitel 5](#) wird die genaue Umsetzung jener Strategie beschrieben.

Darauffolgend wird in [Kapitel 6](#) die Evaluierung der implementierten Lösungsstrategie auf die Real-Welt-Test-Szenarien vorgenommen und dessen Ergebnisse hinsichtlich der Zielstellung bewertet.

Abschließend wird in [Kapitel 7](#) eine Zusammenfassung der Ergebnisse dieser Arbeit gegeben und gleichzeitig auf weitere Forschungs- und Verbesserungsmöglichkeiten hingewiesen.

2 Grundlagen und wissenschaftliche Einordnung

Um die Lösungsstrategie für eine erfolgreiche Objekterkennung zu verstehen und wissenschaftlich einordnen zu können, werden in diesem Kapitel die entsprechenden Grundlagen vorgestellt. Zu Beginn wird im [Abschnitt 2.1](#) die Wissenschaft der Computer Vision beschrieben, die neben weiteren Anwendungsfeldern auch die Aufgabe der Objekterkennung beinhaltet. Anschließend wird das grundlegende Werkzeug dieser Arbeit vorgestellt, der Bereich *Deep Learning* ([Abschnitt 2.2](#)). Darauffolgend werden die wichtigsten Begriffe und die Funktionsweise von *Convolutional Neural Networks* ([Abschnitt 2.3](#)) erläutert, welche für grafisch basierte *Deep-Learning*-Aufgaben von Bedeutung sind. Damit Ergebnisse aus vorherigen Arbeiten genutzt und die Entwicklungszeit verkürzt werden kann, wird in [Abschnitt 2.4](#) die Technik *Transfer Learning*, präsentiert.

2.1 Computer Vision

Computer Vision ist laut Prieße [[Pri15](#), S. V ff] ein Themengebiet der Computervisualistik. Es beschäftigt sich neben der reinen Erfassung und Verarbeitung von Bildern, mit dem Problem diese maschinell zu analysieren und zu interpretieren.

Demnach ist die Aufgabe einer bildbasierten Steckplatzanalyse in dieses Themengebiet einzuordnen.

Bevor die Aufgabe jedoch einer speziellen Bilderkennungsaufgabe aus der *Computer Vision* zugeordnet werden kann, muss zunächst erläutert werden, wie digitale Informationen überhaupt von Computern verarbeitet und dargestellt werden.

2.1.1 Das digitale Bild

In der Regel werden digitale Bilder als rechteckförmige Rasterbilder interpretiert und sind somit als mathematische Matrix darstellbar. Eine Bildmatrix $\mathbf{A}_{M \times N} = (a_{m,n})$ mit den Elementen $a_{m,n}$ in M Zeilen und N Spalten und den ganzzahligen Indizes $m \in \{1, 2, \dots, M\}$ und $n \in \{1, 2, \dots, N\}$ enthält für jedes Indexpaar (m, n) einen spezifischen Wert [[Wer21](#), S. 10]. Dieser Wert gibt an, wie der jeweilige Bildpunkt (Pixel) dargestellt wird.

Schwarz-Weiß- und Grauwertbilder

Die Bildpunkte eines Binärbildes (auch als Schwarz-Weiß-Bild bezeichnet) bestehen entweder aus dem Wert 0, für Schwarz, oder 1 für Weiß (siehe [Abbildung 2.1a](#)).

Die Pixel eines Grauwertbildes enthalten hingegen Zahlen aus $\{0, 1, \dots, 255\}$ (siehe [Abbildung 2.1b](#)). Die jeweiligen Werte bestimmen dabei die Intensität oder Helligkeit des Pixels. Ein Pixel mit dem Wert 0 wird somit schwarz und eines mit dem Wert 255 weiß dargestellt. Aufgrund der 256 unterschiedlichen Helligkeitsstufen werden Grauwertbilder auch als Graustufenbild bezeichnet [[Wer21](#), S. 12].

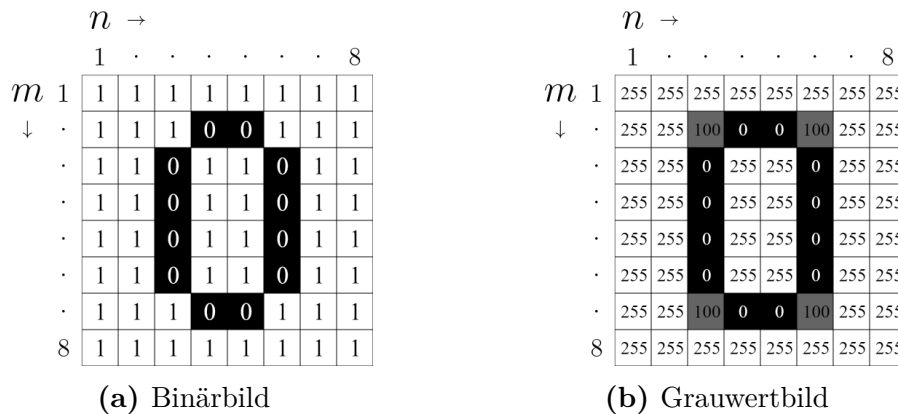


Abbildung 2.1: Beispiele für unterschiedliche Bildmatrizen der Dimension 8×8

Farbbilder

Farbige Bilder verwenden im Grunde das gleiche Rasterbildsystem, wie Binär- oder Grauwertbilder. Der Unterschied ist, dass jeder Pixel aus mehreren Farbkomponenten besteht und mittels additiver Farbmischung zu einem bestimmten Farbeindruck wird [[Wer21](#), S. 14]. In den meisten Farbsystemen werden dafür drei Farbkomponenten in einem Zahlentripel verwendet.

Eines der Relevantesten ist das *RGB*-System, welches sich aus den Primärfarben Rot, Grün und Blau zusammensetzt. Hierbei wird jedem Pixel ein ganzzahliges Zahlentripel (R, G, B), standardmäßig mit Werten zwischen 0 und 255, zugeordnet. Je höher der Wert, desto stärker ist die jeweilige Farbintensität. Beispielsweise wird ein roter Bildpunkt als (255, 0, 0) dargestellt und ein weißer als (255, 255, 255).

Werner nennt außerdem das *HSV*-System als relevantes Farbsystem in der Bildverarbeitung [[Wer21](#), S. 14]. Dieses setzt sich aus den Werten Farbton *H* (engl. *hue*), Farbsättigung *S* (engl. *saturation*) und Farbwert *V* (engl. *value*) zusammen. Je nach Vorhaben kann zwischen diesen Farbsystemen gewechselt werden.

In dieser Arbeit wird, wenn nicht, näher beschrieben, das *RGB*-Farbsystem verwendet.

2.1.2 Operationen auf digitale Bilder

Da digitale Bilder als Rasterinformationen verarbeitet werden, lassen sich auch Matrix-Rechenoperationen analog zu [Abbildung 2.2](#) auf diese anwenden. Dabei ist primär die

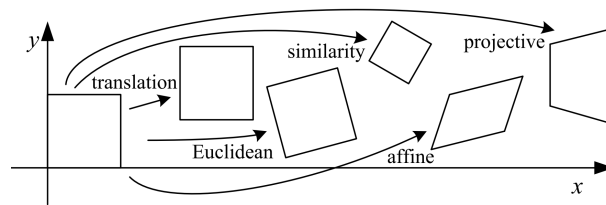


Abbildung 2.2: Das Ergebnis unterschiedlicher Matrix-Transformationen aus [Sze22, S. 33]

perspektivische Transformation (auch *projektive Transformation* bezeichnet) [Sze22, S. 33 ff] für diese Arbeit von Relevanz.

Das Ziel dieser Operation ist, Punkte aus der dreidimensionalen Welt auf die zweidimensionale Bildebene abzubilden. Laut Tomaszewska [Tom13] lassen sich dadurch Längeninformatoren aus einem Bild gewinnen, was hinsichtlich der Zielstellung der Arbeit von Bedeutung ist.

Im Kapitel 12 des Buchs „*Learning OpenCV 3*“ [Kae16] wird im Abschnitt „*Affine and Perspective Transformations*“, das in dieser Arbeit verwendete Verfahren näher erläutert.

2.1.3 Die Teilaufgaben der Bilderkennung

Computer Vision beinhaltet das Lösen von unterschiedlichsten Problemen. Dazu gehören auch Bilderkennungsaufgaben. Diese handeln vom Analysieren der Bildinhalte und haben das Ziel, Objekte und Gegenstände zu identifizieren.

Szeliski teilt die Bilderkennung in Instanzerkennung (engl. *instance recognition*) und generische Objekterkennung (engl. *generic object recognition*) auf [Sze22, S. 276]. In beiden Fällen sollen Objekte aus mehreren Blickwinkeln und trotz Überlagerungen erkannt werden. Die Instanzerkennung beschäftigt sich mit dem Wiedererkennen von eindeutigen 2D- oder 3D-Objekten, wohingegen generische Objekterkennung, jegliche Instanzen einer bestimmten Objektklasse ausfindig macht.

Die Erkennungsaufgaben können wiederum in einzelne Untergruppen unterteilt werden. Um, die im Verlauf der Arbeit beschriebenen Lösungstechniken in einen Kontext setzen zu können, werden zunächst die bedeutsamsten Untergruppen der Bilderkennung erklärt und in [Abbildung 2.3](#) exemplarisch visualisiert.

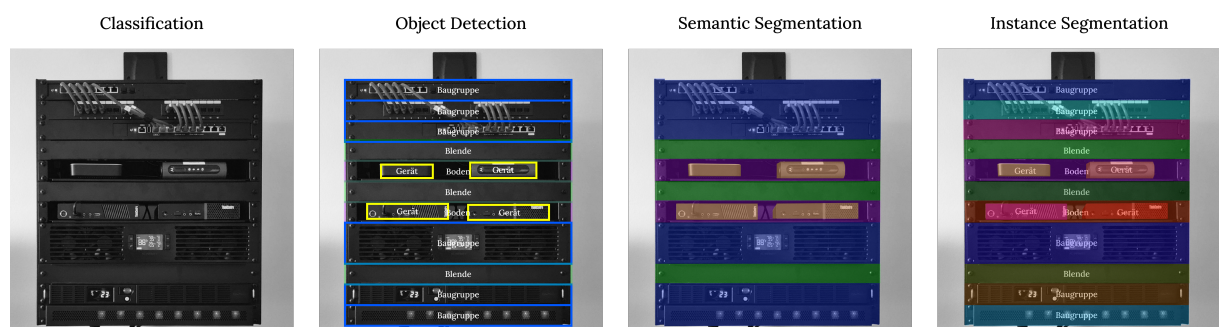


Abbildung 2.3: Die Resultate unterschiedlicher Bilderkennungsaufgaben an einem Beispielbild

Image Classification

Bei der Bildklassifizierung (engl. *image classification*) werden Bilder einer Kategorie zugeordnet [Sze22, S. 278]. Die Klassifizierungsalgorithmen erkennen vereinfacht formuliert, worin das Hauptaugenmerk des Bildes liegt. Dem Bild aus [Abbildung 2.3](#) könnte etwa die Kategorie „Serverschrank“ zugeordnet werden.

Object Detection

Die Objekterkennung (engl. *object detection*) hat das Ziel, unterschiedliche Objekte in einem Bild zu lokalisieren und gleichzeitig zu klassifizieren. Den wahrgenommenen Objekten wird ein rechteckiger Begrenzungsrahmen (engl. *bounding box*) und ein Klassenname (engl. *label*) zugeordnet [Sze22, S. 265]. Ersteres gibt Aufschluss über die Position und Begrenzung des Gegenstandes, siehe [Abbildung 2.3](#). Der Klassenname beinhaltet hingegen den Typus des Objekts. Die Objekterkennung ist die für diese Arbeit relevante Untergruppe, da es die Komponenten eines Serverschranks zu lokalisieren und zu klassifizieren gilt.

Segmentation

Im Gegensatz zu dem in der Objekterkennung verwendeten Begrenzungsrahmens, werden die Gegenstände bei Segmentierungsaufgaben (engl. *segmentation tasks*), pixelgenau, voneinander abgegrenzt [Sze22, S. 307]. Die Identifizierung ist demnach deutlich präziser. Szeliski nennt unterschiedliche Segmentierungsaufgaben, wie *semantic segmentation*, *instance segmentation* und *panoptic segmentation* [Sze22, S. 307].

- Bei der *semantic segmentation* werden Objekte, die der gleichen Klasse angehören, zusammengefasst dargestellt. Es wird also nicht zwischen ihnen unterschieden, erkennbar in [Abbildung 2.3](#).
- Die *instance segmentation* unterscheidet, im Gegensatz zu der *semantic segmentation*, jedes Objekt voneinander. Auch wenn die Objekte der gleichen Klasse angehören. ([Abbildung 2.3](#))
- Die *panoptic segmentation* ordnet jedem Pixel einen Klassennamen zu. Das bedeutet, es werden neben Objekten auch irrelevante „Dinge“, wie Hintergrund oder Boden erkannt. Wichtig ist, dass die erkannten Objekte auch in ihre einzelnen Instanzen unterteilt werden. Vergleichbar mit der *instance segmentation*.

Eine solch pixelgenaue Abgrenzung von Objekten ist für diese Arbeit jedoch nicht zwingend notwendig, da die rechteckigen Begrenzungsrahmen der Objekterkennung zur Positionsbestimmung der Steckplätze ausreichen.

2.2 Deep Learning

Nachdem im [Abschnitt 2.1](#) die Computer Vision eingeleitet und bedeutende Aufgaben vorgestellt worden, ist es unumgänglich dessen fundamentales Werkzeug, *Deep Learning*, vorzustellen.

Goodfellow et al. [[IYA16](#), S. 8] beschreiben *Deep Learning* als eine Art des maschinellen Lernens und demnach laut Ertel [[Ert16](#), S. 3] als einen Teilbereich der Künstlichen Intelligenz.

Ertel definiert Künstliche Intelligenz (KI) mittels eines Zitats nach Elaine Rich:

„*Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better.*“ [[Ert16](#), S. 2]

Demnach erforscht der Bereich, wie sich die Fähigkeiten des Menschen durch Computer imitierten lassen. Zu diesen Fähigkeiten zählt auch das adaptive Verhalten und somit die Lernfähigkeit des Menschen, welche durch maschinelles Lernen imitiert werden soll.

Für das maschinelle Lernen bezieht sich Ertel auf ein Zitat von Tom Mitchell:

„*Machine Learning is the study of computer algorithms that improve automatically through experience.*“ [[Ert16](#), S. 194].

Algorithmen des maschinellen Lernens (engl. *machine learning*) basieren demnach nicht auf modellierten Lösungswegen und erlernen lediglich durch mehrfaches Wiederholen und einem entsprechenden Gütekriterium des Versuchs, wie sie ihre Aufgabe anhand der gegebenen Daten erfüllen können [[Ert16](#), S. 194].

Es werden mehrere Strategien erforscht, um diese Fähigkeit zu erreichen. Für diese Arbeit ist explizit das Erstellen und Verwenden von künstlichen neuronalen Netzen relevant. Der Term *Deep Learning*, bezieht sich dabei nur auf besondere Aufbauvarianten solcher künstlicher neuronaler Netze.

2.2.1 Künstliche Neuronale Netze

Künstliche Neuronale Netze (KNNs) sind „informationsverarbeitende Systeme“ [[Kru15](#), S. 7] die sich an dem Aufbau von Gehirnen orientieren. Dabei liegt der Fokus speziell auf der Funktionsweise des Nervensystems.

Grundlegend besteht das Nervensystem aus einer großen Anzahl von Neuronen, die auf eine bestimmte Art miteinander verbunden sind. Durch diese Verbindung wird eine gerichtete Informationsübertragung von Aktivierungssignalen ermöglicht. Sollten die übertragenen Signale ausreichend erregende Impulse beinhalten und nicht durch hemmende Impulse ausgeglichen werden, so aktiviert sich das Neuron und sendet Signale an die daran verbundenen Neuronen [[Kru15](#), S. 7,13].

Es gibt zahlreiche Methoden, wie die Funktion eines Nervensystems nachempfunden werden kann. Der bekannteste Ansatz stellt die Verwendung von mehrschichtigen Perzeptren (engl. *multilayer perceptrons* (MLPs)) dar [[Kru15](#), 43].

Solch ein MLP kennzeichnet sich laut Kruse durch die Verwendung von Perzeptren, die streng in einzelne Schichten angeordnet sind, aus [[Kru15](#), S. 43]. Der Begriff *Deep Learning* bezieht sich darauf, dass ein mehrschichtiges, also tiefes KNN, trainiert wird [[IYA16](#), S. 165].

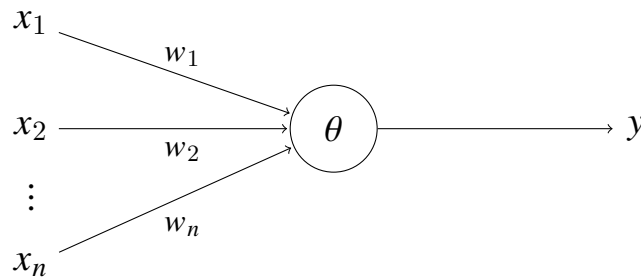


Abbildung 2.4: Eigene Darstellung eines einzelnen Perzeptrons nach [Kru15], mit den Eingaben $x_1 \dots x_n$, den jeweiligen Gewichten $w_1 \dots w_n$ und der Ausgabe y .

Perzeptron

Kruse et al. [Kru15, S. 13] definieren ein einzelnes Perzeptron (auch Schwellenwertelement bezeichnet) als Verarbeitungseinheit für reelle Zahlen mit n Signaleingängen $x_1, \dots, x_n$ und einem Signalausgang y . Die Einheit selbst, enthält einen Schwellenwert θ und jedem Eingang x_i wird ein individuelles Gewicht w_i zugeordnet. Die Ausgabe y errechnet sich aus der Aktivierungsfunktion

$$y = \begin{cases} 1, & \text{falls } \sum_{i=1}^n w_i x_i \geq \theta, \\ 0, & \text{sonst.} \end{cases} \quad (2.1)$$

Grafisch ließe sich solch ein einzelnes Perzeptron, wie in [Abbildung 2.4](#) darstellen.

Im Kontext des Nervensystems stellt das Perzeptron ein einzelnes Neuron mit seinen Ein- und Ausgängen dar. Jenes wird aktiv, wenn es ausreichend erregende Impulse erhält. Das Neuron gilt, gemäß der Aktivierungsfunktion aus [Gleichung 2.1](#), als ausreichend erregt, wenn die Summe des Produkts, der jeweiligen Eingangs- und Gewichtswerte, größer oder gleich dem Wert, Theta, des Schwellenwertelements ist.

Wenn ein bestimmter Eingang für das Neuron von großer Bedeutung ist, so bekommt das angehörige Gewicht, einen höheren Wert zugewiesen. Das hat zur Folge, dass sich das Neuron durch diesen Eingang leichter erregen lässt, da das erhöhte Produkt von Eingangswert und Gewicht, zu einer schnelleren Schwellenwertüberschreitung beiträgt.

Mehrschichtige Perzeptren

Wie bereits angeführt, besteht der Aufbau eines [MLP](#)'s aus mehreren Schichten, welche eine individuelle Anzahl an Neuronen enthalten. Der Aufbau, der im [MLP](#) verwendeten Neuronen ähneln dabei dem Aufbau eines einzelnen Perzeptrons. Der Unterschied ist hingegen, dass diese auch andere Aktivierungsfunktionen, als die vorgestellte Stufenfunktion (siehe [Gleichung 2.1](#)), verwenden können. [Kru15, S. 43]

Grundlegend beinhaltet ein [MLP](#) eine Eingabe- und eine Ausgabeschicht. Zwischen diesen befinden sich entweder keine, eine oder mehrere versteckte Schichten (engl. *hidden layers*). Die Neuronen einer jeweiligen Schicht sind immer mit allen Neuronen der darauffolgenden Schicht verbunden [Kru15, S. 43 ff]. Dieser Aufbau wird auch als „vollständige Vernetzung“ (engl. *fully connected*) bezeichnet [Wer21, S. 354]. Aufgrund der festgelegten

Verbindungsrichtung, die keine Schleifen oder rückwärtigen Verbindungen beinhaltet, gehören **MLPs** zu den „vorwärtsbetriebenen Netzen“ (engl. *feed forward networks*) [Kru15, S. 35]. Der allgemeine Aufbau eines r -schichtigen Perzeptrons ist in **Abbildung 2.4** erkennbar.

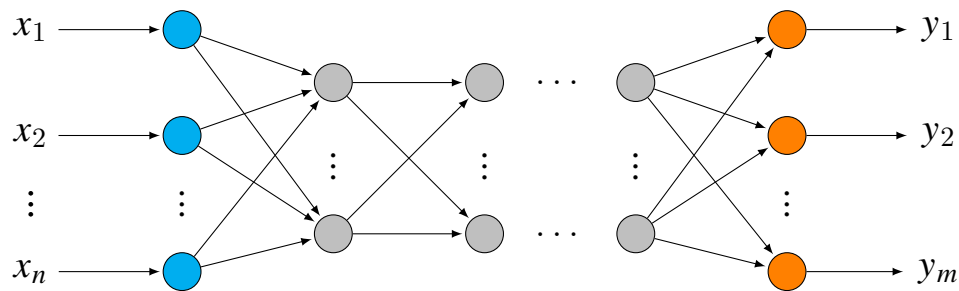


Abbildung 2.5: Eigene Darstellung eines allgemeinen r -schichtigen Perzeptrons nach [Kru15]. Die blauen Knoten stellen die Neuronen der Eingabeschicht und die Orangen, die Neuronen der Ausgabeschicht dar. Die grauen Knoten gehören den Neuronen der $r - 2$ versteckten Schichten an.

2.2.2 Training

Damit neuronale Netze die Fähigkeit erhalten, komplexe Aufgaben zu lösen, müssen diese trainiert werden. Dies bedeutet, dass anhand eines Lernverfahrens und eines Datensatzes, die Gewichte und Schwellenwerte des **KNN**'s angepasst werden.

Es gibt zahlreiche Lernverfahren, um ein **KNN** auf eine Aufgabe zu trainieren. Jene lassen sich laut Goodfellow et al. [IYA16, S. 102] grob in überwachte (engl. *supervised*) und unüberwachte (engl. *unsupervised*) Verfahren kategorisieren.

Grundsätzlich lernen **KNNs** anhand eines großen Datensatzes, wobei die einzelnen Merkmale (engl. *features*) jedes Elements betrachtet werden. Am Beispiel der Bilderkennung, stellen diese Merkmale, normalerweise die Werte jedes einzelnen Pixels dar. [IYA16, S. 97] Die Unterscheidung der Lernverfahren richtet sich dabei nach der Art des Datensatzes, welchen die Algorithmen während ihres Trainingsprozesses verwenden.

Überwachte Lernverfahren

Bei Datensätzen für überwachte Lernverfahren enthalten die Elemente, einen für sie zugewiesenen Klassenname (engl. *label*). Dieses *Label* definiert die gewünschte Ausgabe, die das **KNN** für die jeweilige Eingabe liefern soll. [IYA16, S. 103].

Solche Lernverfahren werden für verschiedenste Aufgabentypen eingesetzt [IYA16, S.98 ff]. Für diese Arbeit stehen überwachte Verfahren mit den Aufgaben, Klassifikation und Regression im Fokus.

- Bei einer Klassifikationsaufgabe ordnet der Algorithmus der Eingabe eine oder mehrere Klassennamen aus einer festgelegten Auswahl zu. Diese Zuordnung wird

häufig durch einen Wahrscheinlichkeitsvektor dargestellt. Die Bildklassifizierung aus [Abschnitt 2.1.3](#), gehört demnach solch Aufgabentyp an [\[IYA16, S. 98\]](#).

- Algorithmen für Regressionsaufgabe prognostizieren eine numerische Vorhersage für eine Eingabe [\[IYA16, S. 99\]](#). Etwa können Modelle, die auf vorherige Wetterdaten trainiert wurden, eine Wettervorhersage für ein ausgewähltes Datum treffen [\[HWX14\]](#).

Unüberwachte Lernverfahren

Datensätze für unüberwachte Lernverfahren enthalten hingegen keine Label. Dadurch können solche Algorithmen nur mittels der Merkmale, bestimmte Eigenschaften, Abläufe und Strukturen im Datensatz erkennen [\[IYA16, S. 103\]](#).

Unüberwachte Lernverfahren finden z. B. bei einigen Segmentierungsalgorithmen Anwendung [\[Sze22, S. 205\]](#).

Ablauf des Trainings mit überwachten Lernverfahren

Da in dieser Arbeit überwachte Lernverfahren zum Einsatz kommen, wird folglich eine grobe Grundlage gegeben, wie eine Anpassung der Gewichts- und Schwellenwerte, anhand eines überwachten Lernverfahrens, ausgeführt werden kann.

Bei solch einem Training werden die gegenwärtigen Ausgabewerte des [KNN's](#), für die Datensatzelemente, in mehreren Epochen mit den Werten der jeweiligen Labels verglichen. Dabei soll der Fehler zwischen Ausgabe und *Label* minimiert werden. Demnach beschreibt das Training ein Optimierungsproblem. Jenes können [KNNs](#) unter anderem mithilfe des Gradientenabstiegsverfahrens lösen [\[Kru15, S.58\]](#). Bei diesem Verfahren, wird der Gradient einer festgelegten Fehlerfunktion, für die aktuelle Ausgabe aller Datensatzelemente berechnet [\[Sze22, S. 223,226\]](#). Anhand des Datensatzes wird dann entschieden, in welche Richtung die aktuellen Gewichts- und Schwellenwerte angepasst werden müssen, um den errechneten Fehler zu minimieren [\[Kru15, S. 58 ff.\]](#). Wie stark diese Anpassung vorgenommen wird, hängt von einer festgelegten Lernrate ab [\[Kru15, S.24\]](#). Ein Verfahren, das solche Änderungen, Schicht für Schicht berechnet, ist das der Fehler-Rückübertragung (engl. *backpropagation*) [\[Kru15, S. 62\]](#).

Dieser Vorgang wird so lange wiederholt, bis ein festgelegtes Abbruchkriterium erreicht wird.

Da es sehr ressourcenlastig ist, die Anpassung basierend auf den Ausgaben aller Datensatzelemente auszuführen, ist es hilfreich, eine Epoche in einzelne Iterationen zu unterteilen [\[Ket17, S. 116\]](#). Die Änderungen an den Gewichts- und Schwellenwerten, werden dadurch bereits nach jeder Iteration, also einer kleineren Anzahl von Vergleichen, ausgeführt.

Der stochastische Gradientenabstieg (engl. *stochastic gradient descent (SGD)*) ist ein derartiges Optimierungsverfahren [\[Ket17, S. 113\]](#), das solch eine Unterteilung vornimmt. Hierbei wird der Gradient der Fehlerfunktion, anhand einer Teilmenge des Datensatzes berechnet. Diese Teilmenge kann aus einem oder mehreren, zufällig ausgewählten, Trainingsbeispielen bestehen. Beinhaltet diese Teilmenge mehrere Elemente, wird sie von Hardt

et al. [HRS16] als *mini batch* bezeichnet. Die Größe des *mini batch* definieren sie dabei als *mini batchsize*.

2.2.3 Hyperparameter

Der Ablauf und das Ergebnis des Trainings hängen von vorher festgelegten Hyperparametern ab [IYA16, S. 118], welche zum Teil bereits in den vorangegangenen Abschnitten erwähnt wurden.

Jene Parameter werden durch das Training nicht angelernt oder verändert und sind auch nicht Teil des fertigen Modells. Sie werden lediglich zum Erreichen des fertigen Modells genutzt.

Beispielsweise stellen die folgenden Werte Hyperparameter dar:

- Die Anzahl der versteckten Schichten, die im KNN enthalten sein sollen.
- Die Wahl des Optimierungsverfahrens, z. B. Gradientenabstieg oder SGD.
- Die ausgewählte Lernrate beim Optimierungsverfahren.
- *mini batchsize*

Aktivierungsfunktionen

Neben der Sprungfunktion Gleichung 2.1, werden überwiegend andere Aktivierungsfunktionen gewählt [Kru15, S. 44]. Insbesondere nicht lineare Funktionen sind für die Berechnungsfähigkeit eines KNN's von Vorteil.

Ein MLP, das nur lineare Aktivierungsfunktionen verwendet, ließe sich laut Kruse et al. [Kru15, S. 47 ff] immer auf ein zweischichtiges MLP reduzieren. Dadurch ist die Berechnungsfähigkeit begrenzt und es sollten nicht lineare Funktionen, für das Lösen von komplexen Aufgaben verwendet werden [Kru15, S. 49]. Die Auswahl der Aktivierungsfunktionen beschreibt einen weiteren Hyperparameter.

In den aktuellen Modellen gehört die *Rectified Linear Unit (ReLU)*-Funktion, zu einer der meistgenutzten Aktivierungsfunktionen [Sze22, S. 217]. Mathematisch wird diese in Gleichung 2.2 [Ket17, S. 30] beschrieben. Dessen grafische Darstellung ist in Abbildung 2.6a zu erkennen. Die ReLU-Funktion wird neben weiteren Aktivierungsfunktionen, auch von den in dieser Arbeit eingesetzten KNNs verwendet.

$$f(x) = \max(0, x) \quad (2.2)$$

Bei Klassifikationsaufgaben enthält die Ausgabeschicht häufig eine gesonderte Aktivierungsfunktion, um das Ergebnis als Wahrscheinlichkeitsvektor darzustellen.

Für binäre Klassifikationsaufgaben wird dafür meist ein einzelnes Neuron mit einer **Sigmoid-Funktion** (Gleichung 2.3) verwendet [Ket17, S.28].

Die Ausgabe eines solchen Neurons liefert einen *Konfidenz-Wert* (engl. *Confidence-Score*) zwischen 0 und 1, erkennbar in der Abbildung 2.6b. Jener *Konfidenz-Wert* gibt die Zugehörigkeitswahrscheinlichkeit der Eingabe, für eine der beiden Klassen 0 und 1 an.

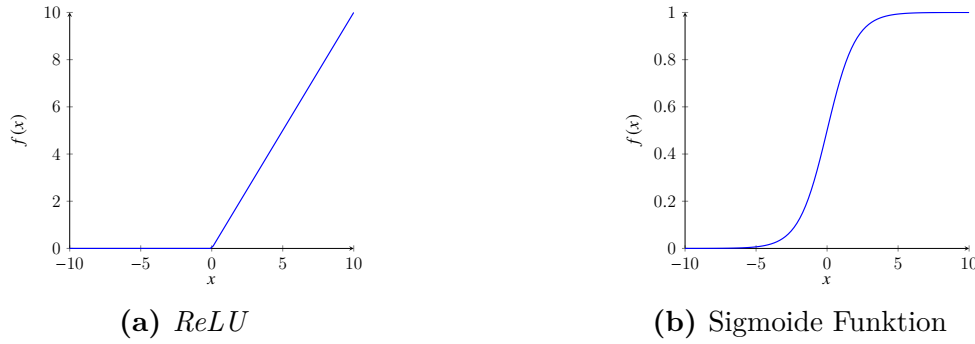


Abbildung 2.6: Plot der Aktivierungsfunktionen Sigmoid und *ReLU*

$$f(x) = \frac{1}{1 + e^{-x}} \quad (2.3)$$

Für eine mehrklassige Differenzierung wird typischerweise die **Softmax-Schicht** als Ausgabe genutzt. Jene erhält die Ergebnisse aller Neuronen der vorherigen Schicht und normalisiert die Werte, sodass die Summe aller Ausgänge der Neuronen 1 beträgt (siehe Gleichung 2.4 [Wer21, S. 356]). Jedes Neuron der Ausgabeschicht gibt demnach einen normalisierten *Konfidenz-Wert* für eine bestimmte Klasse an [Ket17, S. 29].

$$f(z_i) = \frac{e^{z_i}}{\sum_{n=1}^N e^{z_n}}, \quad (2.4)$$

$f(z_i)$ definiert die Ausgabe des i -ten Neurons mit der Eingabe z_i . N ist die Menge aller in der *Softmax-Schicht* enthaltenen Neuronen [Wer21, S. 356].

Fehlerfunktionen

Damit der Gradient und die entsprechende Schrittrichtung beim Training berechnet werden kann, benötigt es eine Funktion, die den Fehler der Ausgaben bestimmt. Die Wahl der Fehlerfunktion (engl. *loss function*) stellt ebenfalls einen Hyperparameter dar. Anschließend sind zwei grundlegende Fehlerfunktionen für Klassifikationsaufgaben aus [Ket17, S. 24 ff] dargestellt.

Je nach Literatur wird die Fehlerfunktion, je nach Aufgabe auch mit *objective function*, oder Kostenfunktion engl. *cost function* beschrieben [IYA16, S. 80]. In dieser Arbeit wird diese Unterteilung jedoch nicht vorgenommen, weswegen einheitlich der Term Fehlerfunktion für jene Funktion verwendet wird.

Binary Cross Entropy Die Fehlerwerte für binäre Klassifikationsprobleme werden in der Regel mit der binären Kreuzentropie (engl. *Binary Cross Entropy*) **BCE** aus Gleichung 2.5 [Ket17, S. 24 ff] bestimmt. Solch binäres Klassifikationsmodell könnte etwa die Aufgabe haben, Hunde und Katzen auf Bildern voneinander zu differenzieren. Bei der Problemstellung, ob ein Bild einen Hund enthält, würde das Modell, mit einer Sigmoid-Aktivierungsfunktion in der Ausgabeschicht, einen Wert zwischen 0 und 1 liefern. Dieser Wert gibt an, wie

wahrscheinlich es ist, dass das Eingabebild einen Hund enthält.

Die *binary cross entropy* misst den Fehler der Zuordnungswahrscheinlichkeit und liefert vorwiegend bei falschen Zuordnungen mit einem hohen *Konfidenz-Wert* große Fehlerwerte.

$$BCE = - \sum_{i=1}^n y_i \log(f(x_i, \theta)) + (1 - y_i) \log(1 - f(x_i, \theta)), \quad (2.5)$$

wobei n die Anzahl der Datensatzelemente und x_i ein entsprechendes Datensatzelement darstellt. $f(x_i, \theta)$ beschreibt den *Konfidenz-Wert* des erstellten Modells auf die Eingabe x_i , basierend auf θ , als dessen gegenwärtige Parameter (Gewichts- und Schwellenwerte). y_i definiert die gewünschte Ausgabe oder das *Label*, für das i -te Datensatzelement aus $y \in \{0, 1\}$.

Cross Entropy Die Kreuzentropie CE lässt sich nicht nur für binäre Ausgaben berechnen, sondern auch für Klassifikationsaufgaben mit mehreren Klassen, siehe [Gleichung 2.6](#) [Ket17, S. 25 ff]. Dabei wird anstatt der Sigmoid-Aktivierungsfunktion die *Softmax-Schicht* verwendet.

$$CE = - \sum_{i=1}^n y_i \log f(x_i, \theta), \quad (2.6)$$

in dem Fall gilt $y \in \{0, 1, \dots, k\}$, mit k als Anzahl der zu identifizierenden Klassen.

2.2.4 Generalisierung

Das Ziel der Steckplatzanalyse-App ist es, neben den im Datensatz vorhanden Racks, auch die Bestückung anderer Racks zu erkennen. Dafür muss das [KNN](#) lernen zu „generalisieren“.

Die Generalisierung thematisiert demnach, die Fähigkeit des Modells, unbekannte Daten zu erkennen [Ket17, S. 9 ff].

Um die Generalisierung des Modells zu überprüfen, sollte der Datensatz in Trainings-, Validierungs- und Testdaten (engl. *train*, *validation*, *test*) aufgeteilt werden [Ket17, S. 219 ff]. Diese Aufteilung findet bereits vor dem Training des [KNN](#)'s statt.

- Der Trainingsdatensatz beinhaltet die Daten, auf denen das [KNN](#) tatsächlich trainiert wird. Die Gewichtsadjustierungen werden demnach basierend auf diesem Datensatz vorgenommen.
- Der Validierungsdatensatz wird genutzt, um das Modell auf Daten zu testen, die nicht im Trainingsdatensatz enthalten sind. Das erlaubt das direkte Evaluieren des Modells und hilft bei der Bewertung von Hyperparameterveränderungen.
- Mithilfe des Testdatensatzes, soll ebenfalls gemessen werden, wie das [KNN](#) auf ungesene Daten reagiert. Der Unterschied zum Validierungsdatensatz ist jedoch, dass jener Datensatz erst verwendet wird, wenn das Modell auf die Trainings- und Validierungsdaten eine zufriedenstellende Leistung erreicht. Mittels dieser Testdaten soll verhindert werden, dass die Entwickelnden durch die bearbeiteten Daten beeinflusst werden und dementsprechende Veränderung am [KNN](#) vornehmen.

Overfitting und Underfitting

Durch die Gegenüberstellung der Leistung des **KNN**'s, auf die Trainings- und Validierungsdaten, wird deutlich, welche Techniken angewendet werden können, um eine bessere Generalisierung zu erreichen.

Das Ziel des Netzes soll sein, dass sich die Performance auf die Validierungsdaten kaum von der Performance auf die Trainingsdaten unterscheidet und dabei gleichzeitig einen gewissen Qualitätswert überschreitet.

Wird dies nicht erreicht, lässt sich der Vergleich auf zwei Muster überprüfen.

Arbeitet das Netz mit den Trainingsdaten erheblich besser, als mit den Validierungsdaten, ist ein *Übertraining* (engl. *overfitting*) zu erkennen.

Dieses entsteht, wenn sich das **KNN** zu sehr an den Trainingsdatensatz anpasst [Kru15, S. 77].

Unterscheiden sich die Validierungs- und Testergebnisse hingegen kaum und erreichen dabei keine gute Performance, so lässt sich ein *Untertraining* (engl. *underfitting*) erkennen [Ket17, S. 220].

2.2.5 Evaluierung eines Deep Learning Verfahrens

Nachdem die Fundamente des *Deep Learnings* dargestellt wurden, ist es notwendig, dessen Mess- und Bewertbarkeit zu beschreiben.

Damit ermittelt werden kann, ob eine Aufgabe als erfüllt gilt, lässt sich ein **KNN** mittels unterschiedlichster Metriken bewerten [Ket17, S. 215].

Im Folgenden werden Bewertungsstrategien für Objekterkennungsaufgaben aus [PNd20] vorgestellt, welche für diese Arbeit besonders von Bedeutung sind.

Prämissen der Metrikdefinierung

Um die für die Objekterkennung meist genutzten Bewertungstechniken verständlich erklären zu können, definieren Padilla et al. zunächst die folgenden Prämissen [PNd20].

- *True positive* (**TP**) beschreibt die Menge richtig erkannter Begrenzungsrahmen.
- *False positive* (**FP**) bezieht sich auf die Erkennungen eines nicht existierenden Objekts oder auf eine nicht ausreichend genaue Begrenzungen.
- *False negative* (**FN**) kennzeichnet Objekte, welche nicht erkannt wurden.
- *True negative* (**TN**) definiert einen Begrenzungsrahmen, der korrekterweise nicht erkannt wurde.¹

Für diese Definitionen muss festgelegt werden, was als richtig erkannter Begrenzungsrahmen gilt. Dafür wird häufig der *Intersection over Union* (**IoU**)-Wert verwendet.

¹Findet in der Objekterkennung keine Anwendung, da sich in einem Bild unendlich viele Begrenzungen erstellen lassen, die nicht erkannt werden sollen

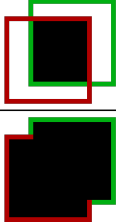
$$IoU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{Diagramm}}{\text{Diagramm}}$$


Abbildung 2.7: Eigene Darstellung einer **IoU** Berechnung nach [PNd20].

Sei B_p der vom **KNN** erkannte Begrenzungsrahmen und B_{gt} der gewünschte (*ground-truth*) Rahmen, so berechnet sich der **IoU**-Wert wie folgt:

$$IoU = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})} \quad (2.7)$$

area beschreibt hierbei den Flächeninhalt der jeweiligen Begrenzungsrahmen.

Übersteigt der **IoU**-Wert einen vorher definierten Schwellenwert, so gilt das Objekt als richtig erkannt [PNd20]. Ein grafisches Beispiel für solch eine **IoU**-Berechnung ist in **Abbildung 2.7** dargestellt.

Qualitätsmetriken

Mittels dieser Definitionen lassen sich die Metriken *Precision* P und *Recall* R beschreiben, welche in der Objekterkennung mehrfach Anwendung finden [PNd20].

$$P = \frac{TP}{TP + FP} \quad (2.8)$$

$$R = \frac{TP}{TP + FN} \quad (2.9)$$

Precision bezeichnet den Prozentsatz, wie viele der erkannten Objekte wirklich relevant sind. *Recall* bezieht sich hingegen darauf, wie viele von den zu erkennenden Objekte letztlich identifiziert wurden.

Basierend auf allen, im Validierungsdatensatz erkannten Objekten einer Klasse, lässt sich die Korrespondenz von *Precision* und *Recall* für verschiedene *Konfidenz*-Werte in einem *Precision-Recall-Diagramm* visualisieren.

Ein gutes **KNN** zeichnet sich dabei durch hoch bleibender *Precision* bei steigenden *Recall*-Werten aus. Kurzum bedeutet das, dass bei absteigenden *Konfidenz*-Werten und gleichbleibendem **IoU**-Schwellenwert keine Erhöhung der **FP**-Erkennungen ausgelöst werden soll.

Demnach kann sich die Qualität des Algorithmus mittels des Flächeninhalts unter der *Precision-Recall-Kurve* für einen **IoU**-Schwellenwert definieren lassen. Die entsprechende Metrik wird als *Average Precision* AP bezeichnet.

Aufgrund des häufig auftretenden „Zickzackverhaltens“ der *Precision-Recall-Kurve*, ist eine genaue Flächeninhaltsberechnung nur schwer ausführbar. Weshalb unterschiedliche Interpolationsverfahren angewendet werden, um das „Zickzackverhalten“ zu entfernen.

Eines davon ist die *11-point Interpolation*, siehe **Gleichung 2.10** und **Gleichung 2.11**. Hierfür wird der Durchschnitt aus den maximalen Präzisionswerten zugehörig aus 11 gleichmäßig

verteilten Recall-Werten $\{0, 0.1, \dots, 1\}$ gebildet.

$$AP_{11} = \frac{1}{11} \sum_{R \in \{0, 0.1, \dots, 1\}} P_{\text{interp}}(R) \quad (2.10)$$

$$P_{\text{interp}}(R) = \max_{\tilde{R}: \tilde{R} \geq R} P(\tilde{R}) \quad (2.11)$$

Anstatt den zum *Recall* gehörigen *Precision*-Wert, für die Durchschnittsberechnung zu verwenden, wird mit dem interpolierten Wert aus P_{interp} gearbeitet. Dieser entspricht dem maximal auftretenden *Precision*-Wert höherer *Recall*-Werte.

Eine genauere Methode stellt das Interpolieren über alle *Recall*-Werte dar [PNd20]. Siehe Gleichung 2.12 und Gleichung 2.13.

$$AP_{\text{all}} = \sum_n (R_{n+1} - R_n) P_{\text{interp}}(R_{n+1}) \quad (2.12)$$

$$P_{\text{interp}}(R_{n+1}) = \max_{\tilde{R}: \tilde{R} \geq R_{n+1}} P(\tilde{R}) \quad (2.13)$$

Hierbei entsteht der *Average Precision* (AP)-Wert aus den interpolierten *Precision*-Werten aller *Recall*-Werte.

Die genannten Metriken beziehen sich nur auf die Qualitätsbewertung einer zu erkennenden Objektklasse.

Damit die Bewertung jedoch, alle im Datensatz vorhandenen Klassen einbezieht, wird die *mean Average Precision* (mAP)-Metrik verwendet.

Jene beschreibt den Durchschnitt der AP-Werte jeder im Datensatz enthaltenen Klasse. Die mAP ist beschrieben als

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^N AP_i, \quad (2.14)$$

wobei AP_i , die AP der i -ten Klasse, aller N im Datensatz enthaltenen Klassen beschreibt.

Je nach gewähltem IoU-Schwellenwert und AP-Interpolierungsverfahren entstehen unterschiedliche Ergebnisse für die AP- und mAP-Metriken. Deshalb sollte laut Padilla et al. festgehalten werden, mit welcher Technik gearbeitet wird, um eine Vergleichbarkeit zu garantieren [PNd20].

In dieser Arbeit bezieht sich die mAP standardmäßig auf einen IoU-Schwellenwert von 0.5 mit dem AP_{all} -Interpolierungsverfahren. Das hat den Grund, dass die in dieser Arbeit vorgestellten Objekterkennungsalgorithmen, sich häufig auf die im Unterabschnitt 3.1.1 beschriebene *PASCAL Visual Object Class Challenge* beziehen. Jene verwendet für die Evaluierung eine mAP basierend auf dem IoU-Schwellenwert von 0.5 und benutzt seit 2010 das AP_{all} -Interpolierungsverfahren [EvW⁺10, S. 314].

Bei der Umsetzung des Konzepts der Arbeit wird sich hingegen auch auf die mAP@0.5:0.95 bezogen. Diese Metrik wird in der im Verlauf der Arbeit vorgestellten *Common Objects in Context Objekterkennungs-Challenge* zur Evaluierung verwendet. Nach Padilla et al. [PNd20] bezieht diese Metrik zehn mAP-Werte basierend auf den IoU-Schwellenwerten von 0.5 bis 0.95 in einer Schrittweite von 0.5 mit ein und berechnet sich demnach durch Gleichung 2.15.

$$mAP_{@0.5:0.95} = \frac{mAP_{0.5} + mAP_{0.55} + \dots + mAP_{0.95}}{10} \quad (2.15)$$

2.3 Convolutional Neural Networks

Für das *Deep Learning* mittels grafischer oder generell, rasterartig aufgebauten Daten, werden *Convolutional Neural Networks* (CNNs) präferiert [IYA16, S. 326 ff].

Es wäre in der Theorie zwar auch möglich, MLPs oder andere Netzstrukturen für solche Aufgaben zu verwenden, jedoch würde daraus ein hoher Berechnungsaufwand resultieren, da aufgrund der Vollverbundenheit der MLPs, jedes im Datum enthaltene Pixel, durch jedes einzelne Neuron übertragen werden müsste.

Ein weiteres Problem des MLP's ist, dass die Rasterinformationen in einen eindimensionalen Vektor umgewandelt werden müssen. Dadurch gehen räumliche Informationen verloren. Resultierend daraus benötigt ein MLP für jede Positionsänderung, der zu erkennenden Objekte im Bild, einzelne Gewichte und Schwellenwerte. Das erschwert nicht nur das Training, sondern auch die Datensatzerstellung [Wer21, S. 412 ff].

Um den Problemen entgegenzuwirken, kommt die andersartige Netzstruktur der CNNs zum Einsatz, dessen Ansatz der Bildsignalverarbeitung des menschlichen Gehirns ähnelt [Wer21, IYA16, S.413 ff, 358 ff].

Vereinfacht ausgedrückt erkennt das Gehirn zunächst, Kanten, Ecken oder andere einfache Strukturen in den Bildsignalen. Daraufhin werden Merkmale auf Basis der erkannten Strukturen identifiziert. Diese Merkmale werden anschließend bekannten Objekten zugeordnet.

Solch schritt-basierte Vorgehen wird in den CNNs umgesetzt.

2.3.1 Funktionsweise

Laut Goodfellow et al. [IYA16, S.332] werden in einem CNN *sparse interactions*, anstatt der *vollständigen Vernetzung* verwendet. Dadurch sinkt der Berechnungsaufwand erheblich, denn *sparse interactions* sorgen dafür, dass die Neuronen einer Schicht, nur mit einem Teil der Neuronen der nächsten Schicht verbunden sind.

Das impliziert wiederum, dass tiefere Schichten mehr Verbindungen zu den ursprünglichen Neuronen der Eingabeschicht aufweisen, ähnlich wie bei einer Baumstruktur, in welcher die Wurzel über mehrere Äste eine Verbindung zu den einzelnen Blättern besitzt. Diese „Sichtweite“, also die Anzahl an verbundenen Neuronen der Eingabeschicht, wird auch als *rezeptives Feld* (engl. *receptive field*) bezeichnet.

Damit ein Objekt an unterschiedlichen Positionen im Bild wiedererkannt werden kann, adaptiert ein CNN, anhand der im [Unterabschnitt 2.3.2](#) vorgestellten Kreuzkorrelation, die einzelnen Verarbeitungsschritte des Gehirns.

Das bedeutet, dass in den ersten Schichten zunächst Kanten und Ecken im Bild erfasst werden, welche in tieferen Schichten komplexen Merkmalen zugeordnet werden.

Die Kreuzkorrelation wird mittels *Parameter Sharing* [IYA16, S. 331ff], auf mehrere Teile der Eingabedaten angewandt, wodurch ein CNN die Eigenschaft der *Translation Equivariance* erhält.

Jene Eigenschaft besagt, dass bei einer Verschiebung der Eingabe, sich auch die Ausgabe in der gleichen Art und Weise verändert [IYA16, S. 334], was dem CNN, eine positionsunabhängige Objekterkennung ermöglicht.

2.3.2 Faltung, Kreuzkorrelation

Im *Machine-Learning*-Bereich wird die Kreuzkorrelation auch als Faltung (engl. *convolution*) bezeichnet, wenngleich es einen minimalen Unterschied zwischen einer Faltungsoperation² und einer Kreuzkorrelation gibt [IYA16, S. 329]. Da diese Arbeit im Machine-Learning-Bereich einzuordnen ist, wird im Folgenden auch der Begriff Faltung, für jene Operation verwendet.

Ein CNN zeichnet sich generell durch die Anwendung von Faltungsoperationen in mindestens einer seiner Schichten aus.

Die im *Machine-Learning* angewendete Faltungsoperation auf zweidimensionale Bilder, lässt sich mittels eines Filters (engl. *kernel*) K und dem Eingabebild I , ausführen.

Das Ergebnis der Faltung gibt an, wie sehr die Eingabedaten dem Filter, an der entsprechenden Position ähneln.

Solch Filter könnte z. B. das Gauß-Filter³ sein, welcher in der Bildbearbeitung als Weichzeichner genutzt wird.

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n) \quad (2.16)$$

Das Filter ist in einer Matrixform darstellbar und „gleitet“ [Wer21, S. 93] über die einzelnen Rasterelemente, des Eingabebildes.

Dabei wird die Gleichung 2.16 [IYA16, S. 329] für jedes im Ausgabe-Bild S enthaltene Pixel angewendet.

Das bedeutet, das Rasterelement, mit den Indizes i, j , wird mittels der Summe aus der Multiplikation jeder Spalte und Zeile des *Kernels* mit den entsprechenden Pixeln des Eingabebildes berechnet (Skalarprodukt). Die beim „Gleiten“ ausgeführte Schrittweite (engl. *stride*) stellt, genau wie die Dimension des Filters, einen veränderbaren Hyperparameter dar und hat einen Einfluss auf das Ergebnis der Faltung.

Aufgrund der im Ausgabebild hervorgehobenen Merkmale wird dieses auch als Merkmalskarte (engl. *feature map*) bezeichnet [IYA16, S. 328].

Bei einer *valid convolution* wird das Filter nur auf die im Eingabebild korrespondierenden Pixel ausgeführt. Das resultiert, je nach *stride* und Dimension des *Kernels*, in einem Ausgabebild mit weniger Pixeln als im Eingabebild [Mic19, S. 10]. Sollten sehr viele Faltungen mit einem großen *Kernel* hintereinander ausgeführt werden, würden die *feature maps* zu klein werden, weshalb die Technik *Padding* zum Einsatz kommt.

Diese fügt dem Eingabebild, basierend auf der benötigten Faltungsstrategie, Pixel an den Rändern hinzu. So existiert neben der *valid convolution*, beispielsweise auch die *same convolution*, welche der Eingabe ausreichend Pixel hinzufügt, sodass die Dimension des

²Bei der tatsächlichen Faltung, wird das Filter K , aus Gleichung 2.16 spiegelverkehrt ausgewertet [Mic19, S. 10].

³Siehe [Wer21, S. 116]

$$\begin{array}{c} I \\ \begin{array}{|c|c|c|c|c|c|} \hline 0 & 22 & 0 & 12 & 0 & 0 \\ \hline 0 & 255 & 42 & 200 & 23 & 0 \\ \hline 0 & 250 & 60 & 212 & 42 & 0 \\ \hline 0 & 200 & 180 & 242 & 120 & 0 \\ \hline 0 & 12 & 31 & 200 & 11 & 0 \\ \hline 0 & 0 & 0 & 140 & 0 & 120 \\ \hline \end{array} \end{array} \quad * \quad \begin{array}{c} K \\ \begin{array}{|c|c|c|} \hline -1 & -1 & -1 \\ \hline -1 & 8 & -1 \\ \hline -1 & -1 & -1 \\ \hline \end{array} \end{array} = \begin{array}{c} S(i, j) \\ \begin{array}{|c|c|c|c|} \hline 961 & -305 & 711 & -127 \\ \hline 735 & -384 & 516 & -145 \\ \hline 558 & 367 & 498 & 305 \\ \hline -171 & -237 & 576 & -265 \\ \hline \end{array} \end{array}$$

$$x' = 256 \cdot \frac{x - x_{min}}{x_{max} - x_{min}} \quad (2.17)$$

Abbildung 2.8: *Valid Convolution* mittels eines 3x3 Filters K und einem *stride* von 1, am Beispiel eines 6x6 Eingabebildes I .

Eingabebildes der Dimension des Ausgabebildes entspricht. Häufig wird den hinzugefügten Pixeln der Wert 0 zugewiesen, jenes bezeichnet ein *zero padding* [IYA16, S. 343].

In [Abbildung 2.8](#) ist eine Beispielerrechnung einer *valid convolution* vorgeführt. Im Ausgabebild S ist zu erkennen, wie das Filter die Kanten der Eingabe hervorhebt, indem jeweils der zentrale Bereich des Filters als Verstärkung und dessen umliegende Bereiche als Dämpfung des Ausgabewerts dienen. Dabei liegt das unterste, weitest rechts positionierte Rasterelement, mit dem Wert 120, aufgrund des nicht vorhandenen *Paddings* nie im Zentrum des Kernels und ist somit nicht in der Merkmalskarte erkennbar.

Des Weiteren enthält die Ausgabe Rasterelemente mit Werten über- und unterhalb der 256 möglichen Helligkeitswerte eines Grauwertbildes. Die dargestellten Helligkeitsstufen des Rasters entsprechen demnach nicht den darin enthaltenen Werten, sondern den durch [Gleichung 2.17](#) berechneten normalisierten Werten.

Für ein [CNN](#) ist so eine Normalisierung nicht notwendig, da keine Visualisierung stattfindet.

Wenn die Eingabe des [CNN](#)'s mehrere Kanäle enthält, wie bei Bildern im RGB-Farbsystem, so wird die Faltung auf jedem Kanal separat mit eigenen Gewichten ausgeführt und dessen Ergebnis mit denen der anderen Kanäle aufsummiert [Ket17, S. 74].

2.3.3 Convolutional Layer

Durch die Faltung lassen sich folglich Kanten, Muster und Merkmale in einem Bild hervorheben.

Das erlaubt dem [CNN](#) basierend auf den Ergebnissen der Filtermasken zu arbeiten. Anstatt jedem Pixel ein Gewicht zuzuordnen, werden in einer Faltungs-Schicht (engl. *Convolutional Layer*) eines [CNN](#)'s mehrere Filter einer festgelegten Größe erlernt. Das impliziert wiederum, dass in einem *Convolutional Layer*⁴ die jeweiligen Filtermasken trainiert werden. Demnach erlernt das [CNN](#), welche *feature maps* für die Erkennung eines bestimmten Objekts von großer Relevanz sind und passt die Filter dementsprechend an.

Ein typisches *Convolutional Layer* lässt sich laut Goodfellow et al. [IYA16, S. 335 ff] in drei Stufen aufteilen.

⁴Das *Convolutional Layer* ist eine Schicht eines [KNN](#)'s, welche Faltungsoperationen anstatt normaler Gewicht und Schwellenwertberechnungen ausführt. [Mic19, S. 10]

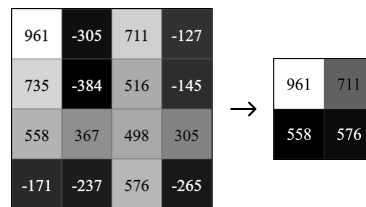


Abbildung 2.9: *Max-Pooling* mit einem 2×2 Filter und *stride* von 2, am Beispiel des Ausgabebildes S aus [Abbildung 2.8](#). Die Anpassung der Helligkeitswerte hat erneut durch die [Gleichung 2.17](#) stattgefunden.

In der Ersten führt die Schicht die entsprechenden Faltungen durch. Im Anschluss werden die Ausgaben mittels einer nicht linearen Aktivierungsfunktion, wie der *ReLU-Funktion*, detektiert. Goodfellow et al. bezeichnen diese Stufe als *detector stage*. Abschließend wird eine *Poolingfunktion* angewendet, um die Ausgabe der Schicht weiter zu verändern.

2.3.4 Pooling

Eine *Poolingfunktion* verändert die Ausgabe an einer gewissen Position, basierend auf einer statistischen Zusammenfassung naheliegender Werte [[IYA16](#), S. 335].

Dadurch wird die Ausgabe herunterskaliert und deren relevanten Strukturen gehen nicht verloren, weswegen in einigen Quellen der Begriff *Subsampling* für das *Pooling* verwendet wird [[Ket17](#), S.71]. Daraus ergibt sich zum einen der Vorteil, dass sich die Berechnungskomplexität des Netzes reduziert und zum anderen eine verbesserte Approximation einer Translationsinvarianz erreicht wird [[IYA16](#), S. 336 ff.].

Solch Translationsinvarianz bedeutet, dass auch bei leicht veränderten Eingaben mittels einer Translation, die Ausgabe der *Poolingfunktion*, unverändert bleibt. Goodfellow et al. [[IYA16](#), S. 336] beschreiben diese Eigenschaft als hilfreich, wenn das Vorhandensein eines Merkmals bedeutsamer ist, als dessen Position.

Max-Pooling stellt neben anderen Funktionen, wie *Average-Pooling* oder L^2 -*Pooling*, solch eine *Poolingfunktion* dar [[IYA16](#), S. 335]. Jene bezieht sich auf die maximalen Werte rechteckiger Teilbereiche des Ausgabebildes, siehe [Abbildung 2.9](#).

2.3.5 Bedeutende CNN-Architekturen

LeNet-5

Lecun et al. [[LBBH98](#)] entwickelten, mit dem in [Abbildung 2.10](#) dargestellten *KNN*, eines der ersten erfolgreichen *CNN*'s, welches der automatischen Zeichenerkennung (engl. *optical character recognition*) dient.

Dieses Beispiel zeigt, dass ein *CNN* mit dem Aufbau eines vollverbundenen, klassischen *MLP*'s gekoppelt werden kann. So werden bei der Bilderkennung die *Convolutional Layer* für die Erkennung der Merkmale eines Bildes verwendet und die klassischen Neuronen eines *MLP*'s zur Klassifizierung dieser genutzt.

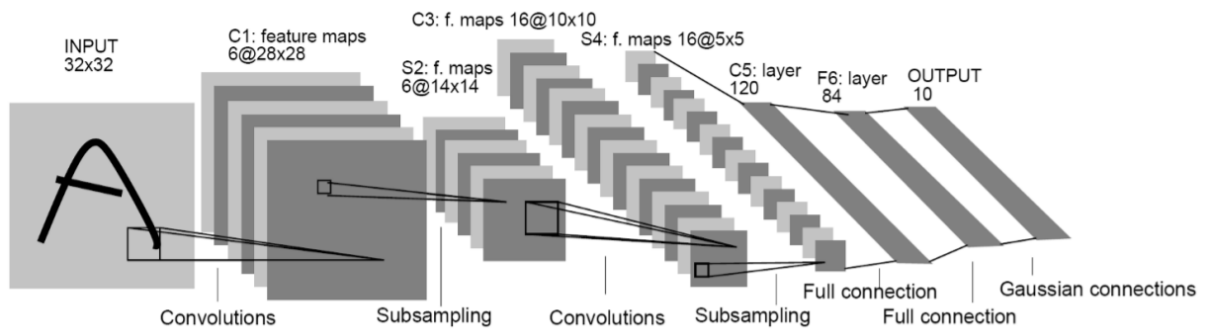
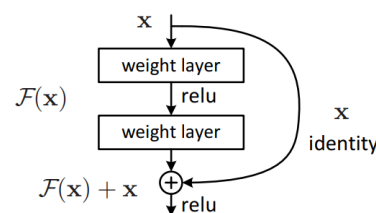


Abbildung 2.10: Architektur des LeNet-5 aus [LBBH98]

ResNet

Für die Bilderkennung sind primär die von He et al. publizierten *Deep Residual Networks* (ResNet) [HZRS16] von Bedeutung. Diese werden auch bei der Entwicklung von Objekterkennungsalgorithmen benutzt, worauf in Kapitel 3 detaillierter eingegangen wird. Grundlegend zeichnet sich *ResNet* durch den Einsatz von *residual blocks* aus. Diese leiten die Aktivierungen eines Neurons einer Schicht, im Gegensatz zu den reinen *feed forward networks*, nicht nur an die Neuronen der nächsten Schicht, sondern auch an Neuronen in tieferen Schichten weiter. Solch ein *residual block* ist in Abbildung 2.11 visualisiert. Diese Weiterleitung stellt eine Lösung des *vanishing-gradient-problems*⁵ dar und erleichtert dadurch das Training von tiefen KNNs.

Abbildung 2.11: Visualisierung eines Beispiel-*residual blocks* aus [HZRS16]

WideResNet

Wide Residual Networks [ZK16] adaptieren den Aufbau des *ResNets*. Die Besonderheit der *WideResNets* liegt in der Verwendung von einer höheren Anzahl an Merkmalskarten pro Schicht, wodurch sich die Tiefe des Netzes verringern lässt. Dies führt hauptsächlich zu einer Verkürzung der Trainingsdauer.

ResNeXt

Die von Xie et al. veröffentlichte *ResNeXt*-Architektur [SRP⁺17] verändert den Aufbau des *residual blocks* und fügt diesem die *split-transform-merge*-Strategie aus [SLJ⁺15] hinzu. Dabei werden verzweigte Verbindungen innerhalb eines *Blocks* angewandt. Die Anzahl

⁵Das *vanishing-gradient-problem* beschreibt das Verschwinden von Gradienteninformationen bei der Fehler-Rückübertragung in früheren Schichten eines KNN's [BSF94].

der Verzweigungen *Cardinality*, stellt dabei neben der Tiefe und Breite einen neuen [Hyperparameter](#) des Netzes dar, dessen Vergrößerung einen positiven Einfluss auf die Leistung des Netzwerks hat.

2.4 Transfer Learning

Aus den vorherigen Abschnitten lässt sich erkennen, wie unentbehrlich aufgabenspezifische Daten für einen auf *Deep Learning* basierenden Algorithmus sind. Bei einer nicht ausreichend vorhandenen Menge an Trainingsdaten oder bei einem Datensatz mit schlechter Qualität ist es, auch mit einem komplexen [KNN](#) kaum möglich, gute Ergebnisse und eine zufriedenstellende Generalisierungsfähigkeit zu erreichen.

Um einen Mangel an Daten auszugleichen, kann laut Nogueira et al. zum einen, die Technik *Data Augmentation* eingesetzt werden [[NPd16](#)]. Diese zielt darauf ab, den Trainingsdatensatz künstlich durch ausgewählte Transformationen zu vergrößern [[NPd16](#)]. Jedoch ist dies allein, bei kleinen Datensätzen, häufig nicht ausreichend, um ein *Overfitting* zu verhindern [[NPd16](#)].

Eine weitere Technik zum Ausgleich eines Datenmangels stellt das *Transfer Learning* dar. Dieses beschreibt nach Pan et al. [[PY10](#)], die Übertragung des gelernten Wissens eines Aufgabenbereichs auf einen Anderen.

Bezogen auf ein [CNN](#) bedeutet das, dass die vortrainierten Gewichte, Schwellenwerte und Filter eines Netzes, auf eine andere Aufgabe übertragen werden können.

Für das *Transfer Learning* von [CNNs](#) unterscheiden Nogueira et al. dabei zwischen den folgenden Strategien:

2.4.1 Merkmalsextraktion

Bei der Merkmalsextraktion (engl. *feature extraction*) wird die letzte vollverbundene Schicht eines vortrainierten [CNN](#)'s an die neue Aufgabe angepasst [[NPd16](#)]. Anschließend wird allein jene Ausgabeschicht auf den Datensatz trainiert. Die vorherigen Parameter des Netzes bleiben dabei unverändert und produzieren die Merkmale basierend auf dem Training des ursprünglichen Datensatzes. Durch das *Transfer Learning* mittels *feature extraction* erlernt das [CNN](#) jene Merkmale für das Lösen der neuen Aufgabe zu verwenden.

2.4.2 Feintuning

Beim Feintuning (engl. *fine-tuning*) wird, wie bei der *feature extraction*, die Ausgabeschicht an die neue Aufgabe angepasst. Jedoch werden neben den Neuronen der letzten Schicht auch andere ausgewählte Gewichte trainiert. Beim Feintuning eines [CNN](#)'s kann es etwa hilfreich sein, nur die tieferen Schichten des [CNN](#)'s zu trainieren, da in den ersten Schichten eher Kanten und Ecken im Bild erkannt werden [[NPd16](#)]. Die Erkennung der Kanten und Ecken ist für viele Computer-Vision-Aufgaben von Bedeutung und muss demnach nicht verändert werden. Die tieferen Schichten, sind hingegen deutlich spezifischer und entscheiden zwischen Merkmalen, welche für den neuen Aufgabenbereich nicht immer von Bedeutung sind.

3 Stand der Technik in der Objekterkennung

Nachdem im [Kapitel 2](#) die Aufgaben der Computer Vision und die Grundlagen des *Deep Learnings* mit den darin enthaltenen Techniken erläutert wurden, werden in diesem Kapitel derzeitige Lösungsstrategien für die Objekterkennung erklärt. Dafür werden im [Abschnitt 3.1](#), zunächst essenzielle Challenges vorgestellt, an denen die Performance von erforschten Objekterkennungs-Strategien standardmäßig bemessen und evaluiert wird. Daraufhin werden die bekanntesten zweistufigen- ([Abschnitt 3.2](#)) und einstufigen- ([Abschnitt 3.3](#)) Deep-Learning-Objekt-Detektoren vorgestellt, welche für die Umsetzung der Zielstellung dieser Arbeit von großer Relevanz sind.

Die Herausforderung der Objekterkennung besteht darin, dass eine unbekannte Anzahl von Objekten mit unterschiedlichen Seitenverhältnissen (engl. *Aspect Ratios*) in einem Bild klassifiziert und lokalisiert werden sollen. Eine einfache Verbindung der *Convolutional Layer* mit vollverbundenen Schichten, welche die Begrenzungsrahmen und eine Klassifizierung des Bildes prognostizieren, ist aufgrund der unbekannten Objektanzahl nicht ausreichend.

So kommt es, dass für die Lösung dieser Problemstellung spezielle Techniken entwickelt wurden, die sich nach Zou et al. [[ZSGY19](#)] in zweistufige- und einstufige Verfahren differenzieren lassen.

3.1 Object Detection Challenges

Seit einigen Jahren werden zahlreiche Datensätze veröffentlicht, an denen Objekterkennungs-Verfahren trainiert und evaluiert werden können. Diese öffentliche Handhabung erlaubt es der Forschung, neue Ansätze zu entwickeln und diese basierend auf den Ergebnissen der Datensätze miteinander zu vergleichen. Folglich werden die wichtigsten solcher Challenges benannt, um im Verlauf des Kapitels eine Vergleichbarkeit der Verfahren zu ermöglichen und die Auswahl der für die Arbeit relevanten Techniken zu begründen.

In [Abbildung 3.1](#) sind Beispielbilder aus den einzelnen Challenges, inklusive der dazugehörigen Begrenzungsrahmen und Klassennamen visualisiert.

3.1.1 VOC

Die *PASCAL Visual Object Class Challenges* (VOC)¹ [[EvW⁺10](#), [EEv⁺15](#)] gehören laut Zou et al. [[ZSGY19](#)] zu den ersten wichtigen Wettbewerben der Computer Vision. Jene beinhalten die Aufgaben: *image classification*, *object detection*, *semantic segmentation*

¹<http://host.robots.ox.ac.uk/pascal/VOC/>

und *action classification*².

Nach Zou et al. sind VOC07 und VOC12 die meistgenutzten Datensätze für die Objekterkennung. Der Datensatz aus 2007 beinhaltet circa 5000 Trainingsbilder, mit insgesamt mehr als 12000 zu erkennenden Objekten aus 20 verschiedenen Klassen. Der Wettbewerb aus 2012 vergrößert den Bestand auf 11540 Trainingsbilder und insgesamt 27450 annotierte³ Objekte.

Die neueren Veröffentlichungen beziehen sich, laut Zou et al., jedoch eher auf andere, größere, Challenges, wie ILSVRC und MS-COCO.

3.1.2 ILSVRC

Die *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC)⁴ [RDS⁺15] hat laut Zou et al. den Stand der Technik der generischen Objekterkennung vorangetrieben. Die ILSVRC fand in den einzelnen Jahren zwischen 2010 und 2017 jährlich statt und beinhaltet unter anderem Bilder aus dem ImageNet-Datensatz [DDS⁺09].

Der ILSVRC-Objekterkennungsdatensatz aus 2014 involviert beispielsweise 200 Objektklassen in über 476000 Bildern, mit insgesamt mehr als 534000 annotierten Objekten.

3.1.3 MS-COCO

Zou et al. beschreiben den *Microsoft-Common Objects in Context* (COCO)-Datensatz⁵ [LMB⁺14] als bisher herausforderndsten Datensatz. Jener beinhaltet mit 80 unterschiedlichen Objektklassen zwar weniger als die ILSVRC. Dafür mit über 897000, jedoch mehr annotierte Objekte, in 164000 Bildern.

Die Objekte sind dabei für die *instance segmentation* annotiert und enthalten neben dem rechteckigen Begrenzungsrahmen, auch Informationen für Segmentierungsmasken, welches das Objekt pixelgenau eingrenzt [LMB⁺21a].

Der Grund für die erhöhte Schwierigkeit der Challenge liegt zum einen an vielen dicht aneinander liegenden Objekte und zum anderen an einer zahlreichen Menge kleiner Objekte, dessen Flächeninhalt weniger als 1% des Bildes ausmacht.

Diese Eigenschaften bringen die Aufgaben der Challenge näher an „Echt-Welt-Szenarien“ und stellen laut Zou et al. dadurch den heutigen Standard für die Qualitätsbewertung eines Bilderkennungsalgorithmus dar.

3.1.4 OID

Die *Open Images Detection* (OID) Challenge⁶ [KDA⁺17] beinhaltet bei Weitem mehr Bilder und Objekte als die bisher erwähnten Datensätze und stellt damit den derzeit

²Den Objekten wird basierend auf den Bildinformationen eine Handlung prognostiziert.

³Ein annotiertes Objekt beschreibt ein Objekt, welches im Datensatz für überwachte Lernverfahren ein *Label* beinhaltet. Bei der Objekterkennung besteht dieses *Label* aus dem Klassennamen und den Positionsbeschreibungen des Begrenzungsrahmens.

⁴<https://image-net.org/challenges/LSVRC/>

⁵<https://cocodataset.org/#home>

⁶<https://storage.googleapis.com/openimages/web/factsfigures.html>

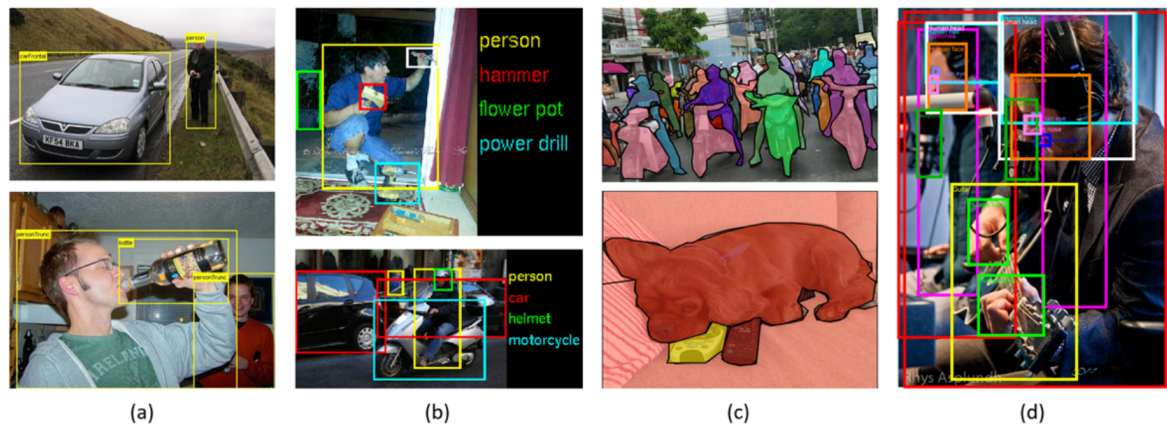


Abbildung 3.1: Bilder- und *Label*-Beispiele von [ZSGY19] aus den einzelnen Challenges (a) VOC07, (b) ILSVRC, (c) MS-COCO und (d) OID.

größten annotierten Datensatz dar.

Genauer involviert dieser 1.9 Millionen Bilder mit 16 Millionen annotierten Objekten aus 600 verschiedenen Klassen, allein für die *Object Detection* [KDA⁺17].

3.2 Zweistufige Detektoren

Um das Objekterkennungsproblem zu lösen, teilen zweistufige Objektdetektoren den Erkennungsprozess in zwei Schritte auf. Im ersten Schritt werden *Regions of Interest* (RoIs) (auch als *Region Proposals* bezeichnet) vorgeschlagen, das sind Regionsvorschläge eines Bildes, welche ein potenzielles Objekt beinhalten könnten. Im zweiten Schritt werden diese Regionen klassifiziert und dessen Position verfeinert [DZW20].

Aufgrund des mehrstufigen Prozesses besitzen die zweistufigen Detektoren laut Du et al. [DZW20] Defizite in der Inferenzdauer⁷ gegenüber den einstufigen Detektoren. Sie zeichnen sich jedoch durch höhere Genauigkeiten aus.

3.2.1 R-CNN

Der im Jahr 2014 von Girshick et al. publizierte regionenbasierte CNN-Ansatz, *R-CNN* [GDDM14], erstellt die Objektvorhersagen basierend auf den Ausgaben eines CNN's für algorithmisch generierte RoIs.

Dafür generiert das Modell im ersten Schritt bis zu 2000 RoIs mittels eines *Selective Search*-Algorithmus aus [UvGS13].

Anschließend wird jede Region durch affine Transformationen in ein quadratisches Bild einer festgelegten Größe transformiert. Dabei richtet sich die Größe nach den Vorgaben der Eingabeschicht des gewählten CNN's. Infolgedessen wird für jede dieser transformierten

⁷Inferenz beschreibt das Anwenden eines Modells auf Eingabedaten.

Regionen ein Merkmalsvektor durch das *backbone*⁸ extrahiert.

Daraufhin wird der vom *backbone* extrahierte Merkmalsvektor jeder Region in eine *Support Vector Machine (SVM)* [Nob06] weitergegeben. Die *SVM* trifft dabei eine Vorhersage, ob die Region ein Objekt beinhaltet und zu welcher Klasse dieses gehört. Neben dieser Klassifizierung werden außerdem die Koordinaten des rechteckigen Begrenzungsrahmens durch eine Regression (engl. *Bounding Box Regression*) verfeinert. Der vollständige Ablauf des Erkennungsprozesses ist in [Abbildung 3.2](#) visualisiert.

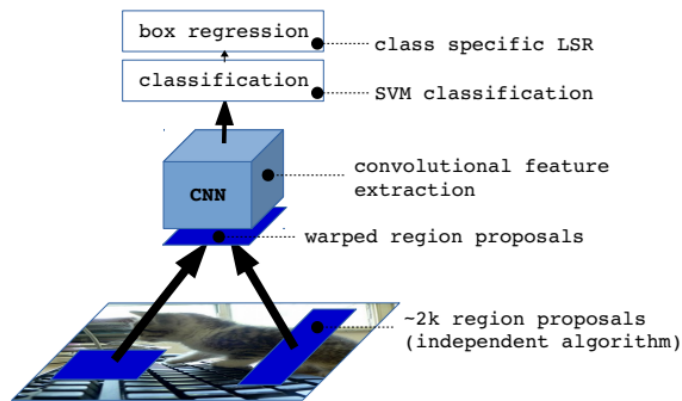


Abbildung 3.2: Visualisierung der *R-CNN* Architektur. Quelle: [Gal17]

Der *R-CNN*-Ansatz verbesserte die *mAP* auf die *VOC* aus 2012 auf einen Wert von 53.4% und war damit mehr als 30% genauer als vorherige Methoden. Dafür wurde eine modifizierte Variante des *AlexNet* [KSH12] als *backbone* eingesetzt und das Modell anhand der Trainingsdaten in einem dreischrittigen Verfahren trainiert.

Im ersten Schritt wurde ein Feintuning des CNN's ausgeführt, im zweiten Schritt die *SVM* trainiert und im dritten Schritt wurde die *Bounding-Box Regression* erlernt.

Trotz dieser Erfolge besitzt der *R-CNN*-Detektor einige Schwachstellen [Gir15]. Dazu gehört zum einen ein hoher Trainingsaufwand, der aus dem dreischrittigen Trainingsverfahren resultiert. Zum Anderen benötigt die Vorhersage eines Bildes durchschnittlich 47 Sekunden (gemessen auf einer Nvidia K40 *Graphics Processing Unit (GPU)* [Gir15]) und ist somit an lange Wartezeiten verknüpft. Ein Grund dafür ist, dass für jede der bis zu 2000 *RoIs* eine Merkmalsextraktion durch das *backbone* ausgeführt wird.

3.2.2 Fast R-CNN

2015 veröffentlichte Girshick mit *Fast R-CNN* [Gir15] eine verbesserte Version des *R-CNN*-Detektors.

Die Funktionsweise ist dabei ähnlich zum *R-CNN*-Detektor.

Zunächst werden die *RoIs* algorithmisch mittels *Selective Search* generiert. Anders als bei *R-CNN* dient bei diesem Ansatz jedoch das gesamte Bild als Eingabe des *backbones*. Demnach werden einmalig Merkmalskarten aus dem Bild extrahiert, anstatt mehrmals für jede *RoI*. Die generierten *RoIs* werden dann auf die extrahierten Merkmalskarten projiziert,

⁸Die für die Merkmalsextraktion verwendeten CNNs der Detektoren werden standardmäßig als *backbone* [ZSGY19] definiert.

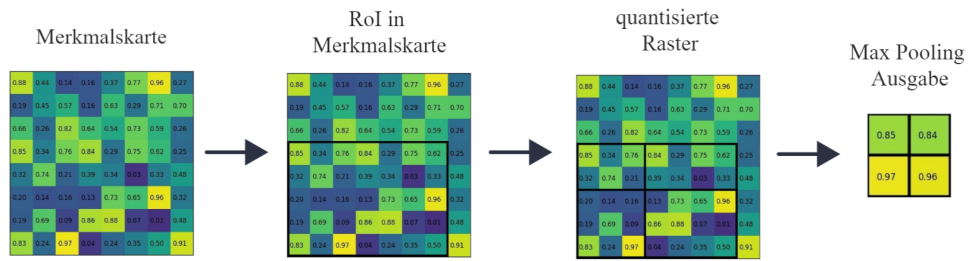


Abbildung 3.4: Ablauf des *RoI Poolings* auf eine Merkmalskarte der Größe 8×8 , mit einer 7×5 großen *RoI* und einer gepoolten *RoI* der Größe 2×2 . Eigene Darstellung nach: [Moh20]

siehe [Abbildung 3.3](#). Das bedeutet, dass die ursprüngliche Koordinaten, werden basierend auf der Größe der Merkmalskarte angepasst.



Abbildung 3.3: Auf eine Merkmalskarte projizierte *RoI*. Quelle: [Moh20]

Anschließend wird durch die *RoI Pooling*-Schicht für jede projizierte *RoI* ein Merkmalsvektor festgelegter Größe aus der Merkmalskarte extrahiert.

Diese *RoI Pooling*-Schicht verwendet *Max Pooling*, um die Merkmale der unterschiedlich großen Regionen in eine Merkmalskarte festgelegter Größe $H \times W$ zu konvertieren. Die Parameter H und W gehören zu den Hyperparametern des *Fast R-CNN* Ansatzes.

Genauer werden beim *RoI Pooling* die $h \times w$ großen Regionen in ein $H \times W$ Raster mit einer quantisierten Rastergröße⁹ von $h/H \times w/W$ dividiert. Anschließend werden mittels *Max Pooling* die einzelnen Ausgabewerte jedes Rasterelements ermittelt, erkennbar in [Abbildung 3.4](#).

Dadurch, dass die Merkmalskarten mit *Fast R-CNN* nur einmalig generiert werden, verkürzt sich die Inferenzdauer deutlich gegenüber dem *R-CNN*, erkennbar in [Abbildung 3.8b](#). Bei genauer Betrachtung der [Abbildung 3.8b](#) wird deutlich, dass die algorithmische Generierung der Regionsvorschläge, den Flaschenhals bei der Inferenzdauer des *Fast R-CNN*-Detektors darstellt

Neben dem generellen Aufbau des Detektors wurde ebenfalls der Trainingsablauf verändert. Im Gegensatz zum dreischrittigen Verfahren verwendet *Fast R-CNN* einen *multi-task loss*. Das bedeutet, dass sich die Fehlerfunktion aus den Fehlerfunktionen der Teilaufgaben der Klassifikation und der Regression des Begrenzungsrahmens zusammensetzt.

Durch diese Veränderungen erreicht *Fast R-CNN* auch erhebliche Verbesserungen in der Trainingsdauer. Des Weiteren verbessert *Fast R-CNN* die *mAP* auf die *VOC* aus 2012 auf einen Wert von 65.7% und liefert damit 23% genauere Vorhersagen als der *R-CNN*-Ansatz.

⁹Die Rastergrößen werden auf die nächste natürliche Zahl quantisiert, sollte aus der Division ein Dezimalwert resultieren. Dies ist in der [Abbildung 3.4](#) gut zu erkennen.

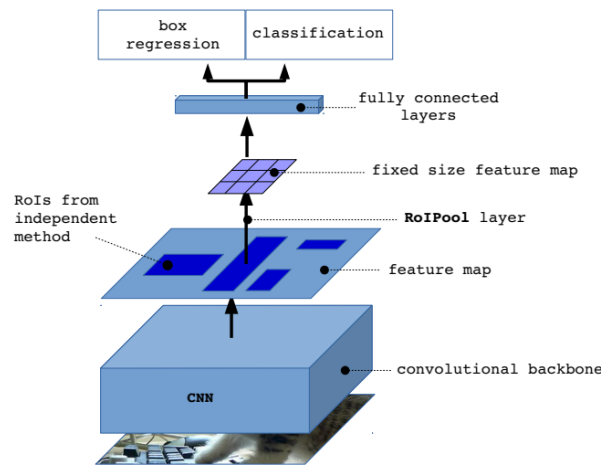


Abbildung 3.5: Visualisierung der *Fast R-CNN* Architektur. Quelle [Gal17]

3.2.3 Faster R-CNN

Ren et al. veröffentlichten mit *Faster R-CNN* [RHGS15] einen Detektor, dessen Funktionsweise auf der des *Fast R-CNN*-Detektors basiert, jedoch den darin enthaltenen Flaschenhals entfernt.

Das Problem des *Fast R-CNN*-Detektors liegt in der Generierung der RoIs. Der *Selective Search* Algorithmus ist mit einer Berechnungsdauer von 1 bis 2 Sekunden [RHGS15] deutlich zeitintensiver als der eigentliche Erkennungsprozess des *Fast R-CNN*-Detektors, siehe Abbildung 3.8b.

Um diesem Geschwindigkeitsproblem zu entgehen, verwendet *Faster R-CNN* ein KNN, welches Regionsvorschläge prognostiziert.

Dieses KNN wird als *Region Proposal Network* (RPN) bezeichnet. Das RPN prognostiziert die Vorschläge basierend auf den Merkmalskarten, welche aus der Eingabe des Bildes in das *backbone* resultieren.

Die einzelnen Vorschläge bestehen dabei aus einem rechteckigen Begrenzungsrahmen und einem *Objectness Score*. Dieser Wert liefert eine Wahrscheinlichkeit, ob es sich bei der jeweiligen Region um ein Objekt einer der zu klassifizierenden Klassen handelt oder ob die Region als Hintergrund gewertet werden kann.

Für die Prognose gleitet ein kleines Netzwerk als *Sliding Window* über die extrahierten Merkmalskarten des *backbones*. Jedes dieser Fenster wird in zwei vollverbundene Schichten eingegeben. Wobei eine Schicht für die *Bounding Box Regression* (*reg*) zuständig ist und eine für die Klassifikation der Region (*cls*). Die Klassifikation gibt an, ob es sich bei der Region um ein Objekt oder Hintergrund handelt und unterscheidet demnach in zwei Klassen.

Zu jedem möglichen Fenster werden unterschiedlich große Regionsvorschläge der maximalen Anzahl k prognostiziert. Die Mittelpunkte der möglichen Fensterpositionen werden als Anker bezeichnet und die daraus resultierenden k Regionen als *Ankerboxen* (engl. *anchor boxes*).

Für jeden Anker trifft, dass RPN eine $4k$ kodierte Ausgabe durch die *reg*-Schicht, welche die Koordinaten der k *Ankerboxen* beinhaltet. Gleichzeitig erhält jeder Anker durch die *cls*-Schicht eine $2k$ kodierte Ausgabe für die zweiklassige Klassifizierung der k *Ankerboxen*, erkennbar in Abbildung 3.6.

In der Veröffentlichung [RHGS15] prognostiziert das RPN drei unterschiedlich skalierte

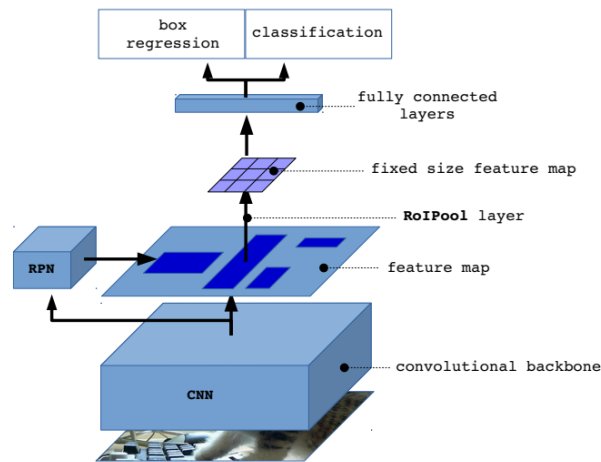


Abbildung 3.7: Visualisierung der *Faster R-CNN* Architektur. Quelle: [Gal17]

Ankerboxen, in jeweils drei verschiedenen Seitenverhältnissen, was in einer maximalen Anzahl $k = 9$ resultiert.

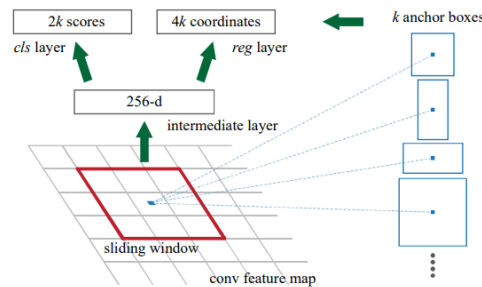
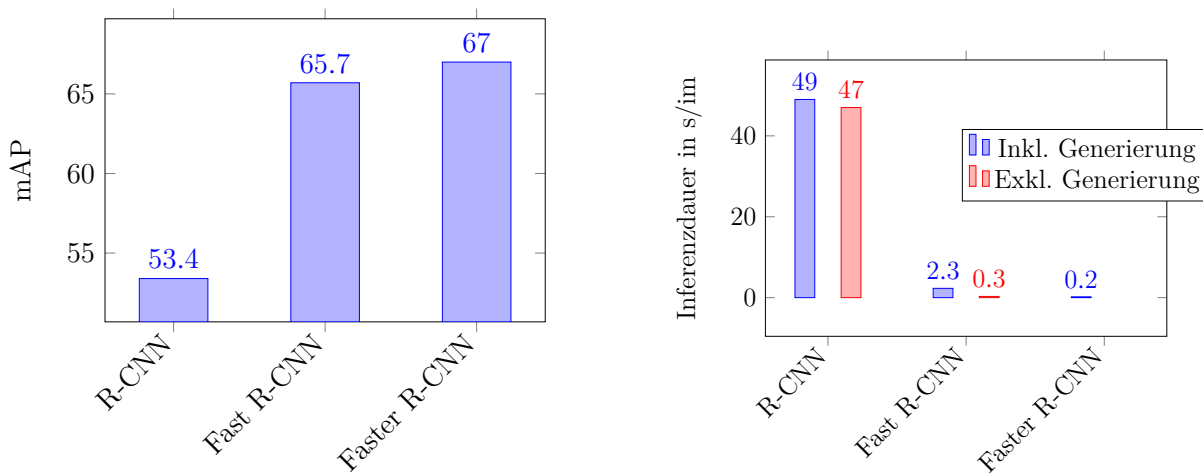


Abbildung 3.6: Visualisierung der *Ankerbox-Generierung* des RPN aus [RHGS15]

Bevor der *Faster R-CNN*-Detektor die Ausgaben des RPN's weiterverarbeitet, werden die Ausgaben nachbearbeitet. RoIs, welche über das Bild hinausragen, werden entsprechend der Bildgröße abgeschnitten. Gleichzeitig werden zu kleine Regionen entfernt. Ferner werden redundante RoIs mittels *Non-Maximum Suppression* herausgefiltert [RHGS15]. Identisch zum *Fast R-CNN*-Detektor werden die gefilterten Regionsvorschläge mittels *RoI Pooling* in eine festgelegte Größe konvertiert und anschließend durch vollverbundene Schichten und einer *Softmax-Schicht* klassifiziert und dessen Begrenzungsrahmen durch *Bounding Box Regression* verfeinert. Der gesamte beschriebene Ablauf ist in [Abbildung 3.7](#) verdeutlicht. Die Verwendung des RPN's resultiert in einer wesentlichen Verkürzung der Inferenzdauer und ermöglicht die Verwendung von *Faster R-CNN* in Echtzeit Anwendungen, bei Bildraten zwischen 5 und 17 Bildern pro Sekunde¹⁰, je nach gewähltem *backbone*. Gleichzeitig verbessert die Verwendung des RPN's die Präzision des Detektors, da die Generierung, der RoIs durch die Trainings verbessert und an den Datensatz angepasst werden kann.

Für die VOC aus 2012 erreicht der *Faster R-CNN*-Detektor eine mAP von 67% und verbessert die Genauigkeit von *Fast R-CNN* um 2%. In [Abbildung 3.8](#) ist ein Vergleich der Präzision und Inferenzdauer der *R-CNN*-Varianten dargestellt.

¹⁰Die Angaben beziehen sich auf Tabelle 5 der *Faster R-CNN*-Veröffentlichung [RHGS15].



(a) Vergleich der **mAP** zwischen *R-CNN*, *Fast R-CNN* und *Faster R-CNN* auf die **VOC** aus 2012.

(b) Vergleich der Inferenzdauer zwischen *R-CNN*, *Fast R-CNN* und *Faster R-CNN* in Sekunden pro Bild. Die Zeitangaben in Blau beinhalten die Berechnung der Regionsvorschläge und beziehen sich auf die Werte aus [Gir15] und [RHGS15]

Abbildung 3.8: Vergleich der Präzision und Inferenzdauer zwischen *R-CNN*, *Fast R-CNN* und *Faster R-CNN* basierend auf den Werten aus [Gir15] und [RHGS15].

3.2.4 Mask R-CNN

Die Besonderheit des im Jahr 2017 publizierten *Mask R-CNN*-Ansatzes [HGDG17] liegt darin, dass dieser Detektor neben den Begrenzungsrahmen und Klassen der Objekte, Masken für eine *instance segmentation* prognostiziert.

Dafür erweitert *Mask R-CNN* den *Faster R-CNN*-Detektor um einen weiteren Zweig, welcher parallel zur Objekterkennung eine Objektmaske vorhersagt, siehe [Abbildung 3.10](#). Die erste Stufe des *Faster R-CNN*-Detektors wird vollständig übernommen. Diese extrahiert Merkmalskarten, basierend auf dem Eingabebild, mittels des gewählten *backbones* und erstellt Regionsvorschläge durch das **RPN**.

Die darauffolgende *RoI Pooling*-Schicht wird bei *Mask R-CNN* durch eine *RoI Align*-Schicht ersetzt. Diese verhindert, dass pixelgenaue Informationen der Regionen, durch das *RoI Pooling* und der darin enthaltenen quantisierten Projektion und Division verloren gehen. Dafür wird in der *RoI Align*-Schicht eine bilineare Interpolation verwendet, um die **RoI** auf eine festgelegte Größe zu transformieren. Der *RoI Align*-Prozess ist in der originalen Veröffentlichung [HGDG17] detailliert beschrieben und dessen Ablauf in [Abbildung 3.9](#) visualisiert. Die transformierten **RoIs** werden, äquivalent zum *Faster R-CNN*-Detektor, in vollverbundene Schichten weitergeleitet und durch eine *Softmax-Schicht* in $K + 1$ Klassen klassifiziert und dessen Begrenzungsrahmen durch eine *Bounding Box Regression* verfeinert. Parallel dazu wird ein kleines *Fully Convolutional Network (FCN)* auf die transformierten **RoIs** angewandt. Dieses *FCN* aus [LSD15] ermittelt die pixelgenauen Segmentierungsmasken. Durch das Hinzufügen der *RoI Align*-Schicht ist der *Mask R-CNN*-Detektor präziser als die vorherigen regionenbasierten **CNN**-Detektoren. Dies wird bei dem Vergleich der **mAP** von *Faster R-CNN* und *Mask R-CNN* auf den *test-dev* Evaluierungsdatensatz

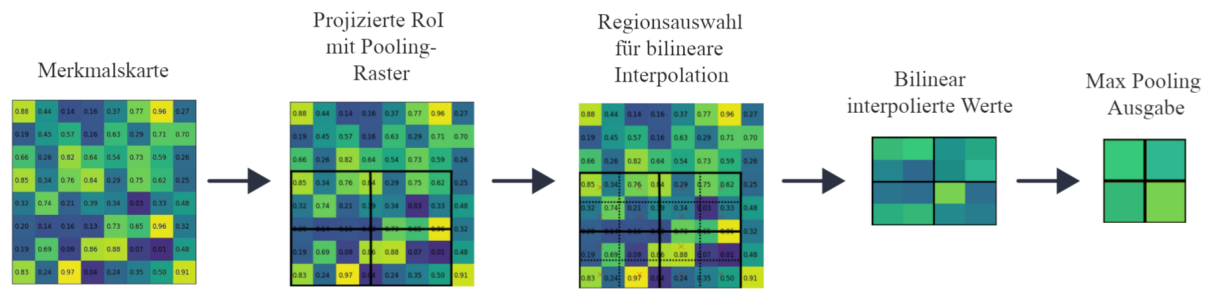


Abbildung 3.9: Visualisierung des *RoI Align*-Prozesses. Eigene Darstellung nach: [Gal17]

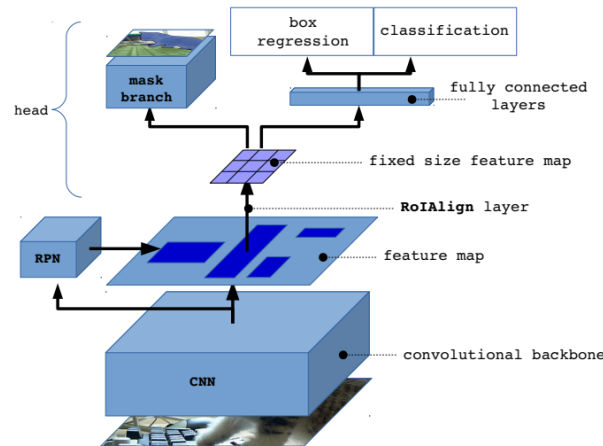


Abbildung 3.10: Visualisierung der *Mask R-CNN* Architektur. Quelle: [Gal17]

[LMB⁺21b] der *Common Objects in Context Object Detection Challenge* in Tabelle 3.1 verdeutlicht.

Für diese Evaluierung wurde das gesamte *Mask R-CNN* Modell trainiert, jedoch nur die Ausgaben der Klassifizierung und der Begrenzungsrahmen berücksichtigt.

Bei der Objekterkennung verwenden He et al. [HGDG17] die CNN-Architekturen *ResNet-101*, sowie *ResNeXt-101* kombiniert mit einem *Feature Pyramid Network* (FPN) aus [LDG⁺17] als *backbone* für den *Mask R-CNN*-Detektor.

Das FPN ermöglicht das Erkennen von Objekten in unterschiedlichen Skalierungen. Dafür werden mehrere Schichten von Merkmalskarten in unterschiedlichen Dimensionen als eine Merkmalspyramide (engl. *Feature Pyramid*) generiert. Die detaillierte Funktionsweise des

Detektor	Pooling	Backbone	mAP mit IoU=0.5
Faster R-CNN	RoIPool	ResNet-101-FPN	59.1
Faster R-CNN	RoIAlign	ResNet-101-FPN	59.6
Mask R-CNN	RoIAlign	ResNet-101-FPN	60.3
Mask R-CNN	RoIAlign	ResNeXt-101-FPN	62.3

Tabelle 3.1: Vergleich zwischen *Faster R-CNN* und *Mask R-CNN* auf den *test-dev* Evaluierungsdatensatz aus COCO. Die Werte basieren auf den Angaben aus [HGDG17].

FPN's ist in [LDG⁺17] beschrieben.

Die Benutzung eines *ResNet-FPN backbone* führt bei *Mask R-CNN* laut He et al. [HGDG17] zu „*excellent gains in both accuracy and speed*“, weshalb jenes auch in dieser Arbeit verwendet wird.

Beim Vergleich vom *Faster R-CNN* mit *Mask R-CNN* in Tabelle 3.1 wird deutlich, welchen Einfluss die *RoI Align*-Schicht auf die Präzision des Modells hat. Trotz gleichem *backbone* erzielt *Faster R-CNN*, allein mit dem Austausch der *RoI Pooling*-Schicht durch die *RoI Align*-Schicht, eine Steigerung der *mAP* um 0.5 Punkte.

Mask R-CNN performt bei gleichem *backbone* abermals 0.5 Punkte besser als *Faster R-CNN* mit *RoI Align*. Laut He et al. [HGDG17] liegt dies an den Vorteilen des *multi-task loss* beim Training des *Mask R-CNNs*. Das Einbeziehen des Fehlerwertes für die Ausgabe der Segmentierungsmasken in der Fehlerfunktion zwingt das Netz, pixelgenaue Angaben zu treffen und das spiegelt sich dementsprechend in der Präzision des Netzes wider.

Die höchsten *mAP*-Werte wurden mit *Mask R-CNN* und dem *ResNeXt-101-FPN* als *backbone* erzielt. Das ausgetauschte *backbone* verbesserte die *mAP* vom *Mask R-CNN* um 2 Punkte und ist somit 3.2 Punkte besser als das *Faster R-CNN* mit dem *ResNet-101-FPN backbone*.

Der Nachteil des *Mask R-CNN*-Detektors gegenüber dem *Faster R-CNN* liegt in der leicht erhöhten Berechnungskomplexität, die nach He et al. [HGDG17] aus der Berechnung der Segmentierungsmasken resultiert.

3.3 Einstufige Detektoren

Einstufige Detektoren führen den Erkennungsprozess in einem Schritt aus und beinhalten, anders als die zweistufigen Detektoren, keinen Schritt für die Generierung von Regionsvorschlägen.

3.3.1 YOLO

Im Gegensatz zu den *R-CNN*-Varianten behandelt der von Redmon et al. im Jahr 2015 veröffentlichte Detektor *You Only Look Once (YOLO)* [RDGF16] die Objekterkennung als eine einzelne Regressionsaufgabe.

YOLO prognostiziert mehrere Begrenzungsrahmen mit dazugehörigen Klassifikationswahrscheinlichkeiten anhand eines einzelnen *CNN*'s.

Das *CNN* verwendet dabei die Merkmale des gesamten Eingabebildes, um die Begrenzungsrahmen vorherzusagen [RDGF16].

Dafür dividiert *YOLO* das Eingabebild in ein Raster der Größe $S \times S$. Jedem erkannten Begrenzungsrahmen wird basierend auf dessen Mittelpunkt eines der $S \times S$ Rasterelemente zugeordnet.

Des Weiteren generiert *YOLO B* unterschiedliche Begrenzungsrahmen für jedes einzelne Rasterelement.

Parallel dazu wird jedem Begrenzungsrahmen ein *Konfidenz-Wert* zugewiesen. Dieser gibt an, wie wahrscheinlich es ist, dass der vorhergesagte Rahmen ein Objekt enthält. Gleichzeitig bezieht der *Konfidenz-Wert* eine Schätzung der Genauigkeit des Rahmens für ein enthaltenes Objekt ein.

Jeder prognostizierte Begrenzungsrahmen besteht neben dem *Konfidenz-Wert* aus den vier

weiteren Werten: x, y, w, h .

(x, y) beschreiben die Koordinaten des Rahmenmittelpunkts, relativ zu den Begrenzungen des zugeordneten Rasterelements.

w definiert die Breite und h die Höhe des Rahmens, relativ zur Größe des Eingabebildes. Jedes Rasterelement, welches Begrenzungsrahmen enthält, in denen ein Objekt vermutet wird, prognostiziert außerdem C bedingte Klassenwahrscheinlichkeiten. Für jedes Rasterelement wird dies jedoch höchstens ein mal berechnet, unabhängig der Anzahl an Rahmen B .

Bei der Inferenz werden die bedingten Klassenwahrscheinlichkeiten mit den individuellen *Konfidenz-Werten* der Begrenzungsrahmen multipliziert. Daraus resultieren klassenspezifische *Konfidenz-Werte* für jeden Begrenzungsrahmen.

Die Vorhersagen werden als Tensor der Form $S \times S \times (B * 5 + C)$ kodiert und lassen sich abschließend durch einen Schwellenwert für die klassenspezifischen *Konfidenz-Werte* und *Non-Maximum Suppression* [Gir15] filtern.

In [Abbildung 3.11](#) ist der Ablauf des Objekterkennungsprozesses mit dem *YOLO*-Modell visualisiert.

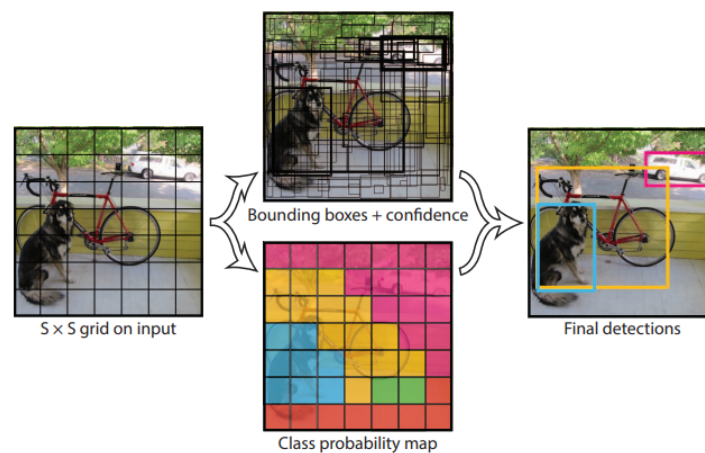
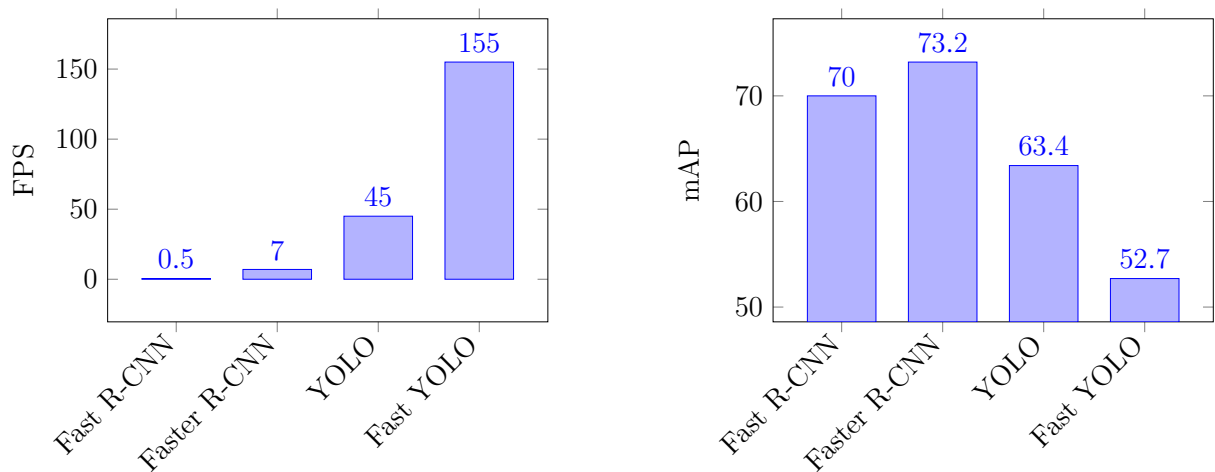


Abbildung 3.11: Visualisierung des *YOLO* Objekterkennungsprozesses. Je dicker der Rahmen eines Begrenzungsrahmens im obersten Teilbild, desto höher ist dessen *Konfidenz-Wert*. Im äußerst rechten Bild wurden die Vorhersagen bezüglich eines Schwellenwerts gefiltert, sodass nur die relevanten Begrenzungsrahmen dargestellt werden. Quelle: [RDGF16]

Für die Evaluierung des *YOLO*-Ansatzes auf die *VOC* verwendeten Redmon et al. [RDGF16] die Parameter $S = 7$, $B = 2$ und $C = 20$, da die *VOC* 20 Klassen beinhaltet. Daraus resultiert ein Tensor der Form $7 \times 7 \times 30$.

In [Abbildung 3.12](#) ist die Architektur des *YOLO*-Modells dargestellt. Das Modell beinhaltet 24 *Convolutional Layer*, welche anschließend in zwei vollverbundene Schichten übergehen.



(a) Vergleich der berechenbaren Bilder pro Sekunde (*FPS*) zwischen *Fast R-CNN*, *Faster R-CNN* mit einem VGG-16 [SZ14] backbone, *YOLO* und *Fast YOLO* auf die VOC aus 2012.

(b) Vergleich der Präzision zwischen *Fast R-CNN*, *Faster R-CNN* mit einem VGG-16 [SZ14] backbone, *YOLO* und *Fast YOLO* auf die VOC aus 2012. Der Trainingsdatensatz beinhaltete dabei zusätzlich Daten aus der VOC aus 2007.

Abbildung 3.13: Vergleich der berechenbaren Bilder pro Sekunde und Präzision zwischen *YOLO*- und *R-CNN*-Varianten basierend auf den Werten aus [RDGF16].

somit zu einer geringeren Inferenzdauer bei gleichbleibender Genauigkeit führen. Ein Vergleich der verschiedenen *YOLO*-Versionen und dessen *CNN*-Architekturen ist in [NE22] aufgeführt.

In [DT20] wird die Leistung von *YOLOv3* mit der Leistung von *Mask R-CNN*-Architektur an einem eigens erstellten Datensatz aus Kühlschranksfotos gegenübergestellt. Daraus geht hervor, dass die Inferenzdauer mit *YOLOv3* deutlich kürzer war, jedoch die beste Genauigkeit durch *Mask R-CNN* erreicht wurde.

Laut Jocher [GAA⁺22] wurden mit dem besten *YOLOv5*-Modell, *YOLOv5x6*, hingegen bessere mAP-Werte auf den COCO-Datensatz als mit *Mask R-CNN* erzielt. *YOLOv5x6* erreichte dabei eine mAP von 72.7, bei einem IoU-Schwellenwert von 0.5. Demnach wurde der Wert von *Mask R-CNN* mit über 10 Punkten übertroffen (siehe Tabelle 3.1). Jedoch existiert bislang keine wissenschaftliche Veröffentlichung zu der *Open-Source-Variante YOLOv5*, welche diese Werte belegt.

3.3.2 Weitere einstufige Detektoren

Neben den *YOLO*-Versionen existieren unter anderem durch den *Single-Shot-Detektor (SSD)* [LAE⁺16], dem *RetinaNet* [LGG⁺17] oder dem *EfficientDet* [TPL20] noch weitere einstufige Objektdetektoren.

Da mit *YOLOv5* in [ZY22], [KKKK⁺22] oder [LH21] jedoch bessere mAP-Werte auf spezialisierte Datensätze, als mit den weiteren einstufigen Detektoren, erreicht wurden, werden diese Varianten in dieser Arbeit lediglich aufgeführt und nicht näher erläutert.

4 Konzept

In diesem Kapitel wird in [Abschnitt 4.1](#) die Zielstellung der Arbeit analysiert. Dafür werden im [Unterabschnitt 4.1.1](#) die Maße und Komponenten eines Racks beschrieben, sowie eine exemplarische Steckplatzanalyse visualisiert ([Unterabschnitt 4.1.2](#)). Anschließend wird im [Abschnitt 4.2](#) eine Lösungsstrategie anhand der Informationen aus den vorangegangenen Kapiteln formuliert.

4.1 Analyse der Zielstellung

Die Zielstellung der Arbeit liegt darin, eine auf *Deep Learning* basierende Steckplatzanalyse zu entwickeln, welche in der *Micro-TSS-App* von *highstreet technologies* einzubinden ist. Dabei soll die Analyse die Steckplatzauslastung eines Serverschranks automatisch anhand eines Fotos ermitteln können.

Jene Steckplatzauslastung beinhaltet Informationen über die Anzahl, Größe und Steckplatzposition der im Serverschrank montierten Komponenten sowie eine Auskunft über nicht belegte Steckplatzpositionen.

Die gewonnenen Informationen sollen in der *Micro-TSS-App* visualisiert werden.

Die Analyse muss dabei nicht in Echtzeit berechenbar sein, jedoch eine hohe Genauigkeit bei der Unterscheidung zwischen belegten und freien Steckplätzen aufweisen.

4.1.1 Rack, Maße und Komponenten

Bevor eine Lösungsstrategie entworfen werden kann, ist es zunächst notwendig, den Aufbau eines Rack zu betrachten. Dabei soll dargelegt werden, wie sich ein Steckplatz definieren lässt.

Nach Faber [[vF21](#), S. 123] dient ein Rack der Aufbewahrung von technischen Komponenten. „Je nach Zweck bzw. untergebrachter Technik wird auch zwischen Server-Racks, Netzwerk-Racks und Stromversorgungs- und Stromverteilungs-Racks unterschieden.“ [[vF21](#), S. S. 123] Jene Unterscheidung wird in dieser Arbeit nicht berücksichtigt, da die Steckplatzanalyse nicht die untergebrachte Technik klassifizieren muss, sondern nur die Auslastung durch die montierten Komponenten.

Laut Faber sind die meisten Serverkomponenten für den Einbau in ein „19-Zoll-Rack“ vorgesehen. [[vF21](#), S. 123]. Diese Racks werden ebenfalls im Rechenzentrum von *highstreet technologies* verwendet, weswegen sich das in dieser Arbeit entwickelte Verfahren auf „19-Zoll-Racks“ bezieht.

Jene Racks weisen ein 19-Zoll (482,6 mm) breites „Einbauchassis“ auf. In dieses lassen

sich Komponenten wie *Rack-Server* oder *Blade-Server* einbauen.

Die Außenmaße des Racks können dabei deutlich breiter als das darin enthaltene „Einbauchassis“ ausfallen. Faber nennt etwa Breiten von 60 oder 80 cm, sowie Tiefen zwischen 60 und 120 cm.

Die Höhe eines Racks orientiert sich an dessen verwendbaren Innenmaß und wird in *Höheneinheiten* (engl. *Rack Units* (**RUs**)) angegeben. Eine *Höheneinheit* entspricht einer Größe von 1.75 Zoll (44.45 mm).

Nach Faber befinden sich in einem Rechenzentrum üblicherweise 42-**RU** hohe Racks.

Der in dieser Arbeit verwendete Begriff: „Steckplatz“, beschreibt die möglichen Einbaupositionen einer 1-**RU** hohen Komponente in einem Rack. Folglich beinhaltet ein 42-**RU** hohes Rack 42 Steckplätze.

Die Maße eines *19-Zoll-Racks* und dessen Komponenten sind für eine gute Kompatibilität genormt und richten sich beispielsweise nach den Normen der Reihen *DIN 41494* und *DIN EN 60297-3*.

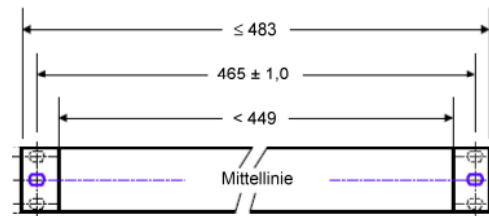


Abbildung 4.1: Visualisierung der Maße eines 1-**RU** hohen Steckplatzes aus *DIN EN 60297-3* [DIN].

Faber nennt die Komponenten: *Rack-Server* und *Blade-Server*.

- Ein *Rack-Server* ist „ein für den Einsatz in Rechenzentren optimiertes Computersystem (Server), das waagrecht in einen Serverschrank (Rack) eingeschoben wird.“ [vF21, S. 123]
- Ein *Blade-Server* lässt sich als „ein für den Einsatz in Rechenzentren optimiertes Computersystem (Server), das senkrecht zunächst in einen Baugruppenträger (Blade Center oder Blade Enclosure) eingeschoben wird,“ [vF21, S. 124] beschreiben.

Da die Unterscheidung dieser Komponenten für die Steckplatzanalyse nicht notwendig ist, werden diese folgend zusammengefasst als *Baugruppe* definiert. Genauer wird in dieser Arbeit jegliche Art von elektronischer Komponente, welche am „Einbauchassis“ montiert ist als *Baugruppe* definiert.

Neben den Baugruppen können Racks außerdem die Komponenten: *Blenden*, sowie *Fachböden* beinhalten:

- *Blenden* werden auf freie Steckplätze montiert und sollen laut [Ras07] die Kühlung des Racks verbessern.
- *Fachböden*¹ sind im Rack montierte Böden. Diese bieten eine Abstellfläche für Elektronik, welche sich nicht am „Einbauchassis“ des Racks befestigen lässt.

¹<https://www.serverschrank24.de/zubehor/fachboden/>

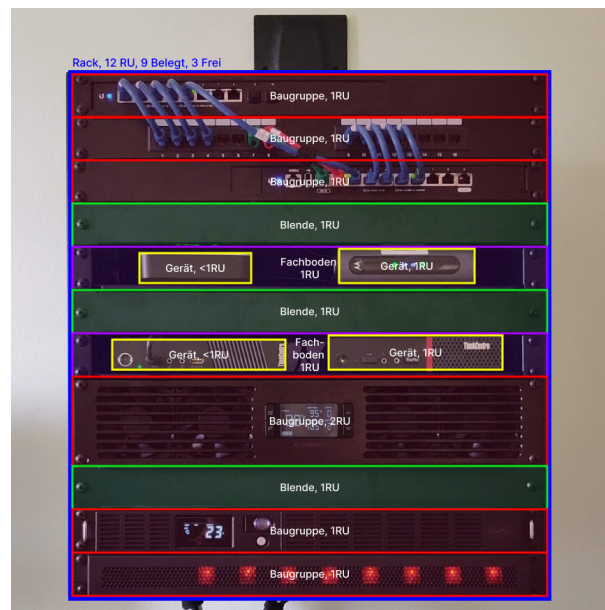


Abbildung 4.2: Visualisierung einer exemplarischen Steckplatzanalyse anhand eines Rack-Fotos

4.1.2 Exemplarische Steckplatzanalyse

In [Abbildung 4.2](#) ist eine exemplarische Steckplatzanalyse visualisiert.

Das zugrundeliegende Rack hat eine Höhe von 12 [RU](#), von welchem 9 Steckplätze belegt und 3 frei sind.

Die Steckplätze 1 - 3 werden von jeweils einer 1-[RU](#) hohen Baugruppe belegt. Im Steckplatz 4 befindet sich eine 1-[RU](#) hohe *Blende*, welche als leerer Steckplatz definiert werden kann. Im fünften Steckplatz ist ein *Fachboden* eingesetzt. Dieser *Fachboden* beinhaltet 2 Geräte, welche die durch den *Fachboden* belegte Höhe im „Einbauchassis“ nicht überschreitet. Die Geräte besitzen demnach eine Höhe von 1 [RU](#) oder kleiner. Steckplatz 6 beinhaltet erneut eine 1-[RU](#) hohe *Blende*. Unterhalb dieser *Blende* ist ein *Fachboden* der Höhe 1-[RU](#) montiert, welcher ebenfalls zwei Geräte enthält, welche die belegte Höhe des *Fachbodens* nicht überschreiten. Im Steckplatz 8 - 9 ist eine 2-[RU](#) hohe Baugruppe befestigt. Darunter befindet sich nochmals eine 1-[RU](#) hohe *Blende*. In den Steckplätzen 11 - 12 sind abschließend jeweils zwei 1-[RU](#) hohe Baugruppen montiert.

4.2 Lösungsstrategie

In [\[Son18\]](#) wurde von A. Soni bereits ein auf *Deep Learning* basierendes Serverschrank-Überwachungssystem mittels *Faster R-CNN* [\[RHGS15\]](#) publiziert. Dieser Ansatz zielt jedoch, analog zu den in der Einleitung erwähnten Überwachungsmethodiken aus [\[PHP10, EY21, Oga21\]](#) auf die Inspektion des LED-Status der einzelnen Serverkomponenten ab und ermöglicht dadurch nicht die in dieser Arbeit erforderliche Bestimmung der Serverschrankkapazität.

Die Problemstellung erfordert das Erkennen von belegten und freien Steckplätzen eines Racks, inklusive der Komponenten- und Rack-Größe, was allein durch eine Erkennung der LEDs nicht erreicht werden kann.

Folglich wurde eine spezielle, dreiteilige Lösungsstrategie entworfen, um eine Steckplatzanalyse gemäß der Problemstellung zu erreichen. Diese separiert die Problemstellung in die Teilaufgaben: [Steckplatzerkennung](#), [Postprocessing](#) und [Micro-TSS-App-Integration](#). Die Teilaufgabe Steckplatzerkennung beinhaltet die reine Objekterkennung. Durch diese sollen die einzelnen Rackelemente erkannt und klassifiziert werden.

Anschließend gilt es, die für die Steckplatzanalyse relevante Informationen aus den Ergebnissen der Objekterkennung zu gewinnen. Dieses Problem soll die zweite Teilaufgabe lösen. Die letzte Teilaufgabe befasst sich mit dem Einbinden der Steckplatzanalyse in die *Micro-TSS-App* und der damit verbundenen Software-Architekturerstellung.

4.2.1 Steckplatzerkennung

Die Kernaufgabe dieser Arbeit liegt in dem *deep-learning-basierten* Erkennen der belegten und freien Steckplätze eines Racks. Dabei gilt es, die Komponenten des Schrankes im Foto zu lokalisieren und gleichzeitig zu klassifizieren. Folglich wird ein Objekterkennungsalgorithmus benötigt.

Die Lokalisierung ist essenziell, um die jeweilige Position der Komponenten im Rack festzustellen. Die Klassifizierung ermittelt hingegen, den Typ des Steckplatzes. Für die Objekterkennung mittels *Deep Learning* wird ein Datensatz benötigt, der für ein [überwachtes Lernverfahren](#) gemäß der Zielstellung annotiert ist.

Da zum Zeitpunkt der Erstellung der Arbeit kein öffentlicher Datensatz für die entsprechende Problematik zur Verfügung steht, liegt eine Herausforderung der Arbeit in der Erstellung eines solchen Datensatzes.

Das impliziert, dass zunächst ein Bestand an Rack-Fotos aufgebaut werden muss. Dafür sollen Fotos im Rechenzentrum von *highstreet technologies* aufgenommen und Fotos aus dem Internet heruntergeladen werden.

Nachdem ein Datenbestand an Rack-Fotos aufgebaut wurde, muss dieser für die [überwachten Lernverfahren](#) annotiert werden. Demnach muss für jeden Steckplatz, der im Datenbestand enthaltenen Racks, ein Begrenzungsrahmen definiert werden, welcher einer Klasse zugehörig ist.

Grundlegend müssend die enthaltenen Komponenten und freien Steckplätze eines Racks binär als **frei** (engl. *empty*) oder **belegt** (engl. *occupied*) klassifiziert werden.

Wie in [Abbildung 4.2](#) zu erkennen, gibt es jedoch Szenarien, in welcher die zweiklassige Differenzierung nicht ausreichend ist. Neben den *Baugruppen* können auch *Fachböden* im Rack moniert sein.

Ein *Fachboden*, auf dem kein *Gerät* abgestellt ist, soll als freier Steckplatz klassifiziert werden. Ferner muss festgestellt werden, ob ein abgestelltes Gerät, die durch den *Fachboden* belegten Höheneinheiten überschreitet. In solch einem Szenario richtet sich die Steckplatzauslastung nach der Höhe des abgestellten Geräts.

Damit die Analyse ausreichend Informationen für die Errechnung der Steckplatzauslastung enthält, muss die Erkennung einen *Fachboden* von einem herkömmlichen belegten Steckplatz unterscheiden. Ferner gilt es, die Begrenzungen der auf dem *Fachboden* abgestellten Objekte zu erkennen, sodass die Steckplatzauslastung des Racks errechnet werden kann. Damit sich die Größe des Racks ermitteln lässt, müssen die Begrenzungen der „Einbauchassis“ im Datensatz beinhaltet sein. Dies erlaubt außerdem die perspektivische Transformation der Bilder. Wodurch Rackfotos, welche in einem suboptimalen Neigungs-

winkel aufgenommen wurden, für das *Deep Learning* optimiert werden können. Gleichzeitig wird das Foto entsprechend den Maßen des „Einbauchassis“ zugeschnitten. Diese künstliche Wiederherstellung des „Einbauchassis-Rechtecks“ soll zum einen zu verbesserten Objekterkennungsergebnissen führen und zum anderen eine präzisere Berechnung der Rackauslastung durch das *Postprocessing* ermöglichen. Zusätzlich soll durch das Beinhaltenden der „Einbauchassis-Begrenzungen“ im Datensatz eine Klassifikation von Rack-Fotos erfolgen, die eine Steckplatzanalyse demnach nur erlaubt, wenn ein Rack auf dem zu analysierenden Foto enthalten ist.

Nachdem der Datensatz erstellt und annotiert wurde, gilt es Objektdetektoren aus [Kapitel 3](#) zu wählen und auf diesen zu trainieren. Dabei soll jeweils ein zweistufiger und ein einstufiger Objektdetektor trainiert werden.

Da die Zielstellung der Arbeit eine sehr hohe Genauigkeit der Analyse erfordert und die Inferenzdauer nicht von Bedeutung ist, wird in dieser Arbeit der *Mask R-CNN*-Ansatz als zweistufiger Detektor gewählt, da jener laut den Evaluierungen der *Object Detection Challenges* die höchsten Genauigkeiten der zweistufigen Objektdetektoren erzielt.

Das Training soll dabei mit unterschiedlich gewählten [Hyperparametern](#) ausgeführt werden, um die höchstmögliche Genauigkeit zu erzielen. Dies beinhaltet den Vergleich von verschiedenen *backbones* für das *Mask R-CNN*-Modell, sowie die Verwendung von unterschiedlichen Lernraten und Optimierungsfunktionen.

Zusätzlich soll mit *YOLOv5* ein einstufiger Objektdetektor auf den Datensatz trainiert werden. Dadurch soll ermittelt werden, ob sich die Unterschiede der Genauigkeit zwischen den ein- und zweistufigen Objektdetektoren auch in dieser Arbeit feststellen lassen.

Damit die Genauigkeit und Generalisierungsfähigkeit der trainierten Modelle verglichen werden kann, soll der Datensatz in *train* und *validation* aufgeteilt werden. Die Evaluierung auf den Validierungsdatensatz soll zum Einstellen der Hyperparameter verwendet werden, um somit die bestmögliche Steckplatzerkennung trainieren zu können.

Im [Abschnitt 3.1](#) ist erkennbar, welchen Umfang aktuell anerkannte Objekterkennungsdatensätze haben können. Solch eine große Anzahl an Bildern lässt sich für den in dieser Arbeit erstellten Datensatz nicht aufbringen. Folglich ist eine weitere Herausforderung festzustellen, ob die ausgewählten Objekterkennungsalgorithmen in der Lage sind, auf Basis der geringen Menge an Daten, ausreichend gute Präzision bei der Steckplatzerkennung zu erzielen.

Aus [Abschnitt 2.4](#) geht hervor, dass die Techniken *Transfer Learning* und *Data Augmentation* verwendet werden können, um einen Mangel an Trainingsdaten auszugleichen. Demnach gilt es diese Techniken auch in dieser Arbeit zu verwenden. Um eine optimale Generalisierungsfähigkeit bei der Objekterkennung zu erreichen, sollen freie Steckplatzflächen² nicht vom Objektdetektor erkannt werden. Die Optik dieser Flächen unterscheidet sich je nach Rack und Foto sehr stark voneinander, wodurch dessen Merkmale nur schwer mit der geringen Anzahl an Daten erlernbar sind.

Folglich müssen diese Freiflächen mittels des *Postprocessings* identifiziert werden.

4.2.2 Postprocessing der Steckplatzerkennung

Damit die Vorhersagen der Objekterkennungsalgorithmen der Zielstellung entsprechen, muss eine Nachbearbeitung (engl. *postprocessing*) der Steckplatzerkennungsergebnisse

²Freie Steckplätze in denen weder eine *Blende*, noch ein *Fachboden* ohne darauf abgestelltem Gerät montiert ist.

ausgeführt werden.

Die Herausforderung liegt dabei darin, aus den erkannten Begrenzungsrahmen, die genauen Steckplatzpositionen und die Größen der erkannten Komponenten im Serverschrank zu berechnen. Des Weiteren sollen dadurch freie Steckplatzflächen des Racks identifiziert werden.

Da der Objektdetektor auch fehlerhaft oder ungenau identifizierte Objekte enthalten kann, sollen die Analyseergebnisse zunächst entsprechend der Zielstellung optimiert und gefiltert werden. Durch den einheitlichen Aufbau der Serverschränke gibt es Ergebnisse, welche in der realen Welt auszuschließen sind. Folglich lässt sich ein Algorithmus definieren, der diese eindeutig fehlerhaften oder ungenauen Objektvorhersagen optimiert.

Zu Beginn soll dieser mittels eines Schwellenwerts die Vorhersagen mit einem zu geringen *Konfidenz-Wert* herausfiltern. Daraufhin sollen Vorhersagen bearbeitet werden, welche sich zu stark überschneiden. In der realen Welt ist es nicht möglich, dass sich die Begrenzungen der Objekte überlagern, außer bei *Fachböden* und den darauf abgestellten Geräten.

Für diese Überprüfung soll die Breite der Begrenzungsrahmen auf 100% der Bildbreite vergrößert werden, da lediglich die vertikale Steckplatzposition und die Höhe von Geräten für die Steckplatzauslastung relevant ist. Die vergrößerten Vorhersagen gelten als überschritten, wenn der geometrische Schwerpunkt einer Vorhersage innerhalb der Begrenzungen einer anderen Vorhersage liegt.

Je nach prognostizierter Klasse sollen die Überschneidungen mittels imperativer Programmierung unterschiedlich nachbearbeitet werden. Zuerst soll überprüft werden, ob es sich bei den Überschneidungen um Objekte der gleichen Klasse handelt. In solch einem Fall sollen die Begrenzungen der Objekte vereinigt werden. Das soll verhindern, dass der Detektor kleinere Objekte innerhalb größeren, komplex-aussehenden Baugruppen erkennt. Gleichzeitig werden dadurch Geräte, die sich auf einem *Fachboden* befinden, zusammengefasst, wodurch sich die Bestimmung der Auslastung vereinfacht.

Beinhaltet die überschneidende Vorhersage hingegen ein Objekt der Klasse *Blende*, soll das Objekt mit dem geringeren *Konfidenz-Wert* aussortiert werden. Dies soll bei der Unterscheidung zwischen *Blendern* und ähnlich aussehenden *Baugruppen* oder *Fachböden* helfen.

Befindet sich letztlich ein Objekt innerhalb eines Objekts der Klasse *Baugruppe*, welches noch nicht durch den *Konfidenz-Schwellenwert* herausgefiltert wurde, soll dieses aussortiert werden. Dies soll nochmals verhindern, dass der Detektor kleinere Objekte innerhalb größeren, komplex-aussehenden Baugruppen erkennt.

Nachdem die Analyseergebnisse gefiltert und optimiert worden sind, müssen daraus die Informationen für die Steckplatzauslastung gewonnen werden. Da die Breite des „Einbauchassis“ normiert ist und die Analyseergebnisse sich auf perspektivisch transformierte Bilder beziehen, lässt sich dafür das Pixel-Per-Meter-Verhältnis (engl. *Pixel per Metric Ratio* (**PPMR**)) des Fotos bestimmen. Mittels dieses Verhältnisses lassen sich die Höheneinheiten eines erkannten Objekts errechnen. Dieses **PPMR** wurde bereits in anderen Arbeiten, wie [KKT⁺21] oder [MK17], zur Bemessung von Objekten verwendet.

Ferner können die Maße des auf das „Einbauchassis“ zugeschnittenen Bildes und das **PPMR** verwendet werden, um die Steckplatzposition eines erkannten Objekts zu ermitteln. Das **PPMR** berechnet sich mittels [Gleichung 4.1](#), worin die Pixel-Breite des auf das „Einbauchassis“ zugeschnittenen Bildes W durch die tatsächliche Breite eines „Einbauchassis“

dividiert wird³. Die Pixelgröße einer Höheneinheit lässt sich mittels des *PPMR*'s wie in [Gleichung 4.2](#) als *Pixel per Rack Unit Ratio (PPRUR)* berechnen.

$$PPMR = \frac{W}{0,4826} \quad (4.1)$$

$$PPRUR = PPMR \times 0,04445 \quad (4.2)$$

Um kleinere Ungenauigkeiten bei den Begrenzungen des „Einbauchassis“ auszubessern, wird das Pixel-Per-RU-Verhältnis basierend auf der Höhe des „Einbauchassis“ angepasst. Da ein korrekt begrenztes „Einbauchassis“ eine ganzzahlige *RU*-Höhe besitzt, lässt sich das *PPRUR* mittels [Gleichung 4.3](#) optimieren, wobei H die Höhe des Bildes und $\lfloor \frac{H}{PPRUR} \rfloor$ die gerundete *RU*-Höhe des Racks beschreibt.

$$PPRUR_{opt} = \frac{H}{\lfloor \frac{H}{PPRUR} \rfloor} \quad (4.3)$$

Basierend auf dem optimierten $PPRUR_{opt}$ lässt sich durch [Gleichung 4.4](#) für jedes erkannte Objekt die genormte Höhe ermitteln.

Um freie Steckplätze zu identifizieren, soll für jedes Objekt überprüft werden, wie groß der Abstand zum oberhalb liegenden Objekt⁴ ist. Überschreitet der Abstand einen Schwellenwert, soll diese Fläche als Objekt eines freien Steckplatzes klassifiziert werden.

Nachdem für alle Objekte die *RU*-Höhe berechnet wurde, lässt sich die Position des Steckplatzes aus den *RU*-Informationen gewinnen.

Da nach dem *Postprocessing* das gesamte Bild in Objekte der Klasse frei oder belegt klassifiziert worden ist, kann die Ermittlung der Steckplatzposition immer basierend auf der Position des oberhalb liegenden Objekts erfolgen. Diese Steckplatzposition besteht aus der Anfangs- und Endposition des erkannten Objekts. Dabei wird dem am höchsten liegenden Objekt die Startposition 0 zugewiesen. Die Endposition bezeichnet hingegen die Addition der *RU*-Höhe des Objekts auf die Startposition. Darauffolgend erhält das nächst höhere Objekt die Endposition des vorherigen Objekts als dessen Startposition.

$$RU_{obj} = \left\lfloor \frac{H_{Box}}{PPRUR_{opt}} \right\rfloor \quad (4.4)$$

Die Voraussetzung für diese Berechnungen ist, dass die Begrenzung des „Einbauchassis“ präzise und fehlerfrei identifiziert worden ist. Deshalb soll diese Eingrenzung in dieser Arbeit durch die Nutzer:innen der App ausgeführt werden.

Anschließend müssen die Ergebnisse analog zur Zielstellung in ein einheitliches Datenschema gebracht werden. Dieses Datenschema muss dabei auch auf die *Micro-TSS-App* übertragbar sein.

Da jede Steckplatzposition eines Racks der Höhe n durch das *Postprocessing* und der Steckplatzerkennung in dessen Objektklasse, binärem Auslastungszustand, *RU*-Höhe und Position analysiert worden ist, soll der Auslastungszustand eines Racks einheitlich in einer tabellarischen Datenstruktur mit n Zeilen und den Spalten für die jeweiligen Informationen dargestellt werden.

³0.4826 Meter entsprechen 19 Zoll

⁴Befindet sich kein Objekt oberhalb des Objekts, wird der Abstand zur oberen Kante des Bildes verwendet.

4.2.3 Micro-TSS-App-Integration

Eine weitere Aufgabe dieser Arbeit ist es, die entwickelte Steckplatzanalyse in die *Micro-TSS-App* zu integrieren. Dabei ist lediglich das Erstellen der Schnittstelle zwischen Nutzer:in und der Steckplatzanalyse von Relevanz. Weitere Funktionalitäten, wie ein automatischer Vergleich der aktuellen Bestückung mit gewünschten oder vorherigen Datenbeständen, sowie eine Speicherungsmöglichkeit des Bestückungszustandes sind nicht Teil dieser Arbeit. Mittels des durch das *Postprocessing* kreierten Datenschemas, können jene Funktionalitäten jedoch implementierbar gemacht werden.

Für die Steckplatzanalyse-App gilt es eine Kamera- oder Fotoauswahlfunktionalität zu entwerfen. Durch diese Funktionalität soll das zu analysierende Foto ausgewählt werden können. Außerdem muss eine Möglichkeit gegeben werden, die Begrenzungen des „Einbauchassis“ mit einem Viereck zu markieren, um die perspektivische Transformation der Rack-Fotos zu ermöglichen.

Zuzüglich muss das Ergebnis der Analyse in der App visualisiert werden. Diese Visualisierung soll die erkannten freien und belegten Steckplätze, sowie dessen Typ und Größe anzeigen.

Da Serverräume häufig schmale Gänge besitzen, kann nicht immer davon ausgegangen werden, dass ein Serverschrank allein durch ein einzelnes Bild aufgenommen werden kann. Folglich soll die App die Analyse von mehreren Teilfotos erlauben, welche zusammenhängend die Steckplatzanalyse des gesamten Racks produzieren. Dafür ist es notwendig, dass die Nutzer:innen mit der Begrenzungsauswahl jeweils die zu analysierenden Teile des Racks markieren.

Damit die Begrenzungsauswahl durch die App-Nutzer:innen korrekt ausgeführt wird, sollen diese vor der ersten Anwendung eine Erklärung zur Festlegung der Punkte des Eingrenzungsvierecks und zur richtigen Sortierung der Fotos bekommen.

Da die Steckplatzanalyse nicht in Echtzeit Ergebnisse liefern muss, soll diese auf einem Server ausgeführt werden. Daraus resultiert die bestmögliche Kompatibilität der App, da nicht auf bestimmte Hardware-Voraussetzungen des Smartphones geachtet werden muss. Demnach gilt es eine Serverschnittstelle zu entwickeln, welche die zu analysierenden Fotos empfängt, die Steckplätze auf den perspektivisch transformierten Fotos erkennt und die Ergebnisse des *Postprocessings* an die *Micro-TSS-App* weiterleitet.

5 Umsetzung

In diesem Kapitel soll die Umsetzung der im [Kapitel 4](#) erstellten Lösungsstrategie erläutert werden. Im [Abschnitt 5.1](#) sind die Teilschritte für das Erreichen der *deep learning* basierten Steckplatzerkennung beschrieben. Im [Abschnitt 5.2](#) wird auf die Umsetzung des *Postprocessings* eingegangen. Abschließend wird in [Abschnitt 5.3](#) die Integration der Steckplatzanalyse mit der *Micro-TSS-App* erläutert. Zusätzlich befindet sich auf dem beiliegenden USB-Stick ein Verzeichnis¹ mit dem für die Arbeit wesentlichen Programmcode.

5.1 Implementierung der Steckplatzerkennung

Dieser Abschnitt widmet sich der Umsetzung der *deep learning* basierten Objekterkennung. Dabei werden neben dem Training der Objektdetektoren und der damit verbundenen Wahl der *Hyperparameter* (siehe [Unterabschnitt 5.1.3](#), [Unterabschnitt 5.1.4](#)), auch die Teilprozesse der *Datensatzerstellung* inklusive dessen *Analyse* beschrieben. In [Unterabschnitt 5.1.2](#) wird zusätzlich die genutzte Hard- und Software vorgestellt.

5.1.1 Datensatzerstellung

Wie aus [Abschnitt 4.2](#) hervorgeht, muss für das Training der Objektdetektoren zunächst ein Rack-Foto-Datensatz erstellt werden.

Die Erstellung umfasst das [Beschaffen](#) und konsistente [annotieren](#) der Rack-Fotos. Zusätzlich wird die [Vorverarbeitung der Daten](#) erläutert, welche für die Berechnungen des *Postprocessings* essenziell ist. Letztlich wird in eine [Analyse des Datensatzes](#) vorgenommen und dessen Größe sowie eine Häufigkeitsverteilung der einzelnen Klassen visualisiert.

Beschaffung der Rack-Fotos

Für die Beschaffung der Rack-Fotos kamen verschiedene Quellen und Strategien zum Einsatz.

Zum einen wurden Fotos mittels eines *Samsung Galaxy S21+*² und einem *iPhone XS*³ manuell aus den Rechenzentren von *hightstreet technologies* aufgenommen.

Dabei wurden die Fotos in verschiedenen Perspektiven, Winkeln und Kameraeinstellungen aufgezeichnet. Die unterschiedlichen Kameraeinstellungen beziehen sich dabei auf die

¹MD5-Hash 97930926a13dea7c5ff9cf709e02cfa7

²<https://www.samsung.com/de/smartphones/galaxy-s21-5g/>

³https://support.apple.com/kb/SP779?locale=de_DE

Auswahl eines Tele- oder Weitwinkelobjektivs.

Insgesamt wurden 120 Fotos von 7 unterschiedlichen Racks in Vorder- und Rückseite im Rechenzentrum von *highstreet technologies* aufgezeichnet.

Neben den vom Smartphone aufgezeichneten Fotos wurden mehrere Bilder aus dem Internet heruntergeladen.

Dazu wurden neben der manuellen Stichwort- und fotobasierten *Google-Bildersuche*⁴ die Downloadskripts für die folgenden Plattformen verwendet:

- Mittels der Pythonbibliothek *imgurpython* [JS13] wurden Fotos von *Imgur*⁵ basierend auf den Stichworten: *homelab*, *server* und *serverrack* heruntergeladen.
- Das Skript von Vasile [Luc22] lud Fotos des *Subreddits homelab* von *Reddit*⁶ herunter.
- Durch das von Yicong entwickelte Skript [Jay22] wurden Bilder zu den Stichworten: *homelab* und *serverrack* über die *Google-Bildersuche* heruntergeladen.

Daraus resultierte zunächst ein Datenbestand von 5033 Bildern. Durch das Pythonpaket *imagededup* [JLJT19] wurden mittels des *perceptual hashings* [JGH15] Duplikate des Bestands entfernt. Abschließend wurden Bilder, welche trotz der Stichworte, keine Serverschränke enthielten, manuell aussortiert.

Daraus resultierten letztlich 495 verwendbare Bilder aus dem Internet und ein gesamter Datenbestand von 615 Rack-Fotos.

Auswahl der Annotierungssoftware

Bei der Auswahl der Annotierungssoftware waren primär zwei Faktoren entscheidend.

Zum einen musste die Software in der Lage sein, Segmentierungsinformationen abzuspeichern, da *Mask R-CNN* für das Training einen für *instance segmentation* annotierten Datensatz benötigt.

Zum Anderen durfte die Software nur lokal auf einem Ubuntu- oder Windowssystem ausgeführt werden. Dadurch sollte vermieden werden, dass die aufgezeichneten Fotos aus den Rechenzentren auf fremden Servern gespeichert werden.

Basierend auf diesen Kriterien wurde die Open-Source-Software *labelme* [Wad] gewählt, dessen Benutzeroberfläche in [Abbildung 5.1](#) visualisiert ist. *labelme* speichert die Begrenzungs- und Segmentierungsannotationen eines Fotos in einer gleichnamigen *JavaScript Object Notation* (JSON)-Datei ab.

Annotierungsstrategie

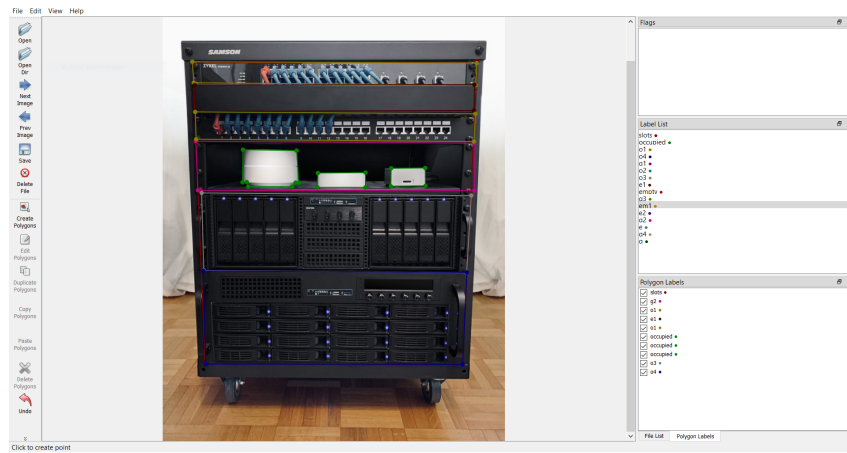
Damit ein möglichst fehlerfreier Datensatz für die überwachten Lernverfahren erstellt werden konnte, wurde eine Annotierungsstrategie erstellt. Diese sollte für ein konsistentes Annotierungsverhalten sorgen.

Nach dieser Strategie waren die Begrenzungen der im Foto enthaltenen Racks mit einem vier-punktigen Polygon zu markieren, welches der Klasse **slots** zugehörig ist. Die vier Punkte des Polygons, sollten dabei die Eckpunkte des „Einbauchassis“ festhalten. Die

⁴<https://www.google.de/imghp?hl=de>

⁵<https://imgur.com/>

⁶<https://www.reddit.com/>

Abbildung 5.1: Darstellung der Benutzeroberfläche von *labelme*

Label	Bedeutung
<i>slots</i>	Begrenzungen des „Einbauchassis“
<i>occupied</i>	elektronisches Gerät, welches auf einem Fachboden abgestellt ist
<i>occupied</i>	Baugruppe einer nicht eindeutig feststellbaren Größe
o_i	Baugruppe der RU-Höhe $i = \{1, \dots, 4\}$
g	Fachboden einer nicht feststellbaren Größe
g_i	Fachboden der RU-Höhe $i = \{1, \dots, 4\}$
e	Blende einer nicht feststellbaren Größe
e_i	Blende der RU-Höhe $i = \{1, 2\}$

Tabelle 5.1: Darstellung der im Datensatz annotierten Klassen.

Eckpunkte von „Einbauchassis“, welche nicht vollständig im Foto enthalten waren, sollten sich dabei am letzten, vollständig sichtbaren Steckplatz orientieren.

Neben den Begrenzungen des Racks waren die einzelnen *Baugruppen*, *Blenden*, *Fachböden* und die darauf abgestellten Geräte jeweils durch ein Polygon einzugrenzen. Damit möglichst viele Informationen in den Annotationen enthalten blieben, sollten die einzelnen Klassen basierend auf dessen RU-Höhe unterteilt werden. Die einzelnen Klassen und dessen Bedeutung sind in Tabelle 5.1 dargestellt.

Vorverarbeitung der Daten

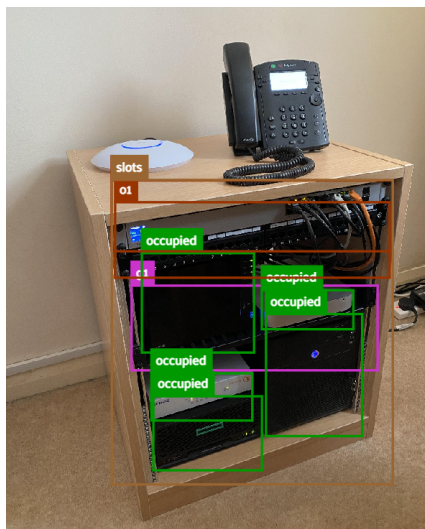
Zunächst wurde der Datensatz bestehend aus den Labelinformationen und Fotos in einen Trainings- und Validierungsdatensatz aufgeteilt. Dabei wurden 85% für das Training und 15% des Datenbestandes zur Validierung genutzt. Genauer wurde für die Aufteilung die Funktion `train_test_split`⁷ der Bibliothek *Scikit-learn* verwendet.

Um den Datensatz auf das Training vorzubereiten, wurden die *JSON*-Dateien der Fotos, in einer einzelnen *JSON*-Datei im Format des Datensatzes der *COCO*-Objekterkennungschallenge zusammengefasst. Dafür wurde das Skript `labelme2coco`⁸ aus *labelme* angewandt.

⁷https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

⁸https://github.com/wkentaro/labelme/blob/main/examples/instance_segmentation/labelme2coco.py

Gemäß dem Konzept wurden die Fotos und Labelinformationen durch eine perspektivische Transformation, basierend auf den Annotationen der „Einbauchassis“, optimiert. Dabei wurden ausschließlich Fotos verwendet, welche mindestens ein „Einbauchassis“ enthielten. Zu jedem „Einbauchassis“ wurde ein neues perspektivisch transformiertes Foto erzeugt. Gleichzeitig wurde eine neue JSON-Datei erstellt, welche die perspektivisch transformierten Annotierungsinformationen enthielt. Für die Implementierung wurde eine Anleitung⁹ zur perspektivischen Transformation durch die Bibliothek *Open-cv*¹⁰ verfolgt. Mit dem Python-Paket *Imantics* [Bro19] wurden die Annotierungsinformationen an die perspektivisch transformierten Fotos angepasst. In *Abbildung 5.2* ist das Ergebnis einer perspektivischen Transformation mit dessen Annotierungsinformationen visualisiert. Darin ist erkennbar, dass die Begrenzungsrahmen der Komponenten, nach der perspektivischen Transformation deutlich präziser sind.



(a) Annotiertes Foto ohne perspektivischer Transformation



(b) Annotiertes Foto nach Anwendung der perspektivischen Transformation

Abbildung 5.2: Visualisierung einer perspektivischen Transformation basierend auf den Eckpunkten des „Einbauchassis“ eines Racks

Des Weiteren wurden die Annotationen der Lösungsstrategie angepasst. Die Klassen wurden demnach ohne Höheninformationen zusammengefasst, da diese sich aus den Größen der Begrenzungsrahmen errechnen lassen und für das Modell nur schwer erlernbar sind. Da *Baugruppen* einer nicht feststellbaren Größe beim Annotieren nicht von auf einem *Fachboden* abgestellten Geräten differenziert wurden, ließen sich Objekte dieser Klasse nicht ohne Weiteres einer *Baugruppe* oder einem abgestellten Gerät zuordnen. Um ein erneutes Annotieren zu vermeiden, wurde festgelegt, dass Objekte der Klasse **occupied**, welche mehr als 80% der Breite des „Einbauchassis“ einnehmen, als *Baugruppe* zu klassifizieren sind. Das folgt daraus, dass die auf einem *Fachboden* platzierten Geräte meist deutlich schmäler als montierbare Baugruppen sind.

Die daraus resultierenden Klassen und dessen Bedeutung sind in *Tabelle 5.2* dargestellt.

⁹<https://pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/>

¹⁰<https://opencv.org/>

Label	Bedeutung
slots	Begrenzungen des „Einbauchassis“
device	Baugruppe
blend	Blende
level	Fachboden
occupied	elektronisches Gerät, welches auf einem Fachboden abgestellt ist

Tabelle 5.2: Darstellung der im vorverarbeiteten Datensatz annotierten Klassen.

Analyse des Datensatzes

Im Folgenden wird der Datensatz analysiert, um einen Überblick über die Diversität der Rackfotos zu bekommen.

Dafür wird lediglich der für das Training relevante, also der vorverarbeitete Datensatz, mit den perspektivisch transformierten Fotos betrachtet.

Da nicht alle der ursprünglich 615 Fotos ein ersichtliches „Einbauchassis“ enthielten, wurde der Datensatz auf eine Größe von 541 Fotos verkleinert. 455 Fotos davon im Trainingsdatensatz und 86 im Testdatensatz.

Die 541 Fotos weisen dabei unterschiedliche Größen auf. Die durchschnittliche Bildgröße beträgt 2422×1625 Pixel (Höhe $\times$ Breite). Das größte Bild des Datensatzes entspricht der Größe 3486×4407 Pixel. Das kleinste Bild ist wiederum 66×122 Pixel groß.

Die im Datensatz enthaltenen Racks haben insgesamt 40 verschiedene RU-Höhen. Dies resultiert daraus, dass die Racks häufig nicht vollständig im Foto enthalten sind und demnach, entsprechend der Annotierungsstrategie, am letzten vollständig enthaltenen Steckplatz begrenzt und durch die perspektivische Transformation abgeschnitten wurden. Die durchschnittliche Rackgröße beträgt 18RU.

Insgesamt enthält der Datensatz 6076 annotierte Objekte, 541 „Einbauchassis“ und 5535 darin enthaltene Elemente. Die jeweiligen Häufigkeiten der Objektklassen sind in der [Tabelle 5.3](#) ersichtlich und in [Abbildung 5.3](#) visualisiert.

Dabei ist erkennbar, dass 3690 Objekte der Klasse **device** angehören. Mit 2543 Objekten stellen 1RU *Baugruppen* die am häufigst auftretenden Objekte dar. Ferner wurden 871 *Blenden* annotiert, wovon 91% eine Größe von 1RU aufweisen. Außerdem sind 328 *Fachböden* und 646 Geräte, welche auf diesen abgestellt sind, im Datensatz enthalten.

Es wird deutlich, dass der Datensatz zum Großteil aus 1RU Objekten und dabei vorwiegend aus *Baugruppen* oder *Blenden* besteht. Deswegen wird vermutet, dass die besten Erkennungsergebnisse bei Racks, welche mit 1RU *Baugruppen* oder *Blenden* bestückt sind, erzielt werden können.

Ferner enthält der Datensatz nur wenig *Fachböden* und noch weniger *Fachböden* einer anderen Größe, als 1RU. Demnach wird es dem Detektor vermutlich schwerfallen, eine Erkennung dieser zu generalisieren.

Label	ursprüngliches Label	Häufigkeit
slots	slots	541
device	o1	2543
device	o2	795
device	o3	68
device	o4	114
device	occupied	170
occupied	occupied	646
level	g1	177
level	g2	97
level	g3	22
level	g4	9
level	g	23
blend	e1	789
blend	e2	48
blend	e	34

Tabelle 5.3: Darstellung der im vorverarbeiteten Datensatz annotierten Klassen inklusive dessen Häufigkeit.

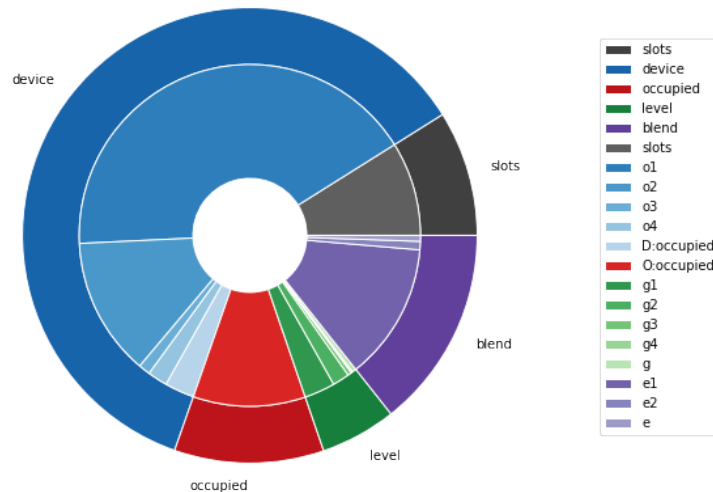


Abbildung 5.3: Verschachteltes Kreisdiagramm zur Darstellung der Klassenhäufigkeitsverteilung.

5.1.2 Framework und Hardware

Für die Implementierung der Steckplatzerkennung wurde die Python Bibliothek *PyTorch*¹¹ gewählt. *PyTorch* bietet laut Paszke et al. eine gute Benutzerfreundlichkeit und behält trotzdem eine hohe Berechnungsgeschwindigkeit bei [PGM⁺19]. Dafür erlaubt die Bibliothek die Unterstützung durch Hardwarebeschleunigung mittels GPUs.

Des Weiteren beinhaltet *PyTorch* die Bibliothek *torchvision* [MR10], welche ein vor implementiertes *Mask RCNN*-Modell und die Auswahl verschiedener vortrainierter *backbones* anbietet. Ferner wurde die Open-Source-Implementierung von *YOLOv5* [GAA⁺22] mittels *PyTorch* erstellt.

Um die Hardwarebeschleunigung von *PyTorch* zu nutzen, wurde über den Cloud-Computing-Anbieter *Scaleway*¹² eine GPU-Instanz für das Training gemietet. Dadurch ließ sich das Training auf einer *TeslaP100*¹³ des Herstellers *NVIDIA*, mit einem dediziertem Arbeitsspeicher von 16 Gigabyte ausführen.

Die erstellte GPU-Instanz beinhaltet automatisch eine für das *machine learning* optimierte *Anaconda*¹⁴ Umgebung. Diese umfasst unter anderem *Python 3.8.5* und *PyTorch 1.11*.

5.1.3 Implementierung des Mask-RCNN-Detektors

Im Folgenden wird die Implementierung des Mask-RCNN-Detektors erläutert.

¹¹<https://pytorch.org/>

¹²<https://www.scaleway.com/>

¹³<https://www.nvidia.com/de-de/data-center/tesla-p100/>

¹⁴<https://www.anaconda.com/>

Einlesen der Daten und Data Augmentation

PyTorch verlangt für das Training auf einem eigenen Datensatz eine entsprechende Python Klasse, welche das Einlesen der Fotos mit den dazugehörigen Annotationen bewerkstelligt. Diese Datensatz-Klasse muss die Funktionen: `__init__`, `__len__` und `__getitem__` beinhalten. Diese wurden mithilfe der Anleitungen der *PyTorch*-Dokumentation [PyT22b] sowie mit einer Anleitung, welche auf *Kaggle*¹⁵ veröffentlicht wurde [pdo22], implementiert.

Zunächst wurden die Bilder auf eine einheitliche Größe gebracht. Dafür wurde die maximale Seite des Bildes auf 800 Pixel skaliert und das ursprüngliche Seitenverhältnis beibehalten. Um letztlich ein 800×800 großes Bild zu erhalten, wurde ein *zero padding* angewandt. Ferner wurden die Pixelwerte des Bildes für bessere Erlernbarkeit mittels der jeweiligen Verfahren aus [PS16] zwischen 0 und 1 normalisiert sowie standardisiert. Anschließend wurde mit der Python-Bibliothek *Albumentations* [BIK⁺20] *Data-Augmentation* auf die Fotos und Annotationen ausgeführt. Dabei wurden folgende Augmentationen nacheinander zu unterschiedlichen Wahrscheinlichkeiten auf die Bilder angewandt:

- Horizontale Spiegelung mit einer Wahrscheinlichkeit von 50%
- Einen der folgenden Weichzeichner mit einer Wahrscheinlichkeit von 50%
 - **Blur** (Bild wird mit einem *Kernel* zufälliger Größe weichgezeichnet)
 - **MotionBlur** (Bewegungsunschärfe wird mit einem Kernel zufälliger Größe auf das Bild angewendet)
 - **MedianBlur** (Bild wird durch einen *Median-Filter* weichgezeichnet)
- Eine der folgenden Kontrastanpassungen mit einer Wahrscheinlichkeit von 50%
 - **CLAHE** (Auf das Bild wird kontrastlimitiertes adaptiertes Histogrammebenen [PAA⁺87] angewendet)
 - **RandomBrightnessContrast** (Helligkeit und Kontrast des Bildes wird zufällig verändert)
- Eine der folgenden Farbveränderungsmethodiken mit einer Wahrscheinlichkeit von 50%
 - **ChannelShuffle** (Die Farbkanäle des Bildes werden zufällig miteinander vertauscht)
 - **ColorJitter** (Verändert zufällig Helligkeit, Kontrast und Sättigung des Bildes)

Die Augmentationen sollten verschiedene „Real-Welt-Szenarien“ abbilden und resultierend daraus den Datensatz diversifizieren.

Es wurde sich gegen zufälliges *Zooming* oder *Cropping* entschieden, da diese kein „Real-Welt-Szenario“ bedienen. Die zu analysierenden Fotos sollen immer das vollständige „Einbauchassis“ enthalten, was durch das *Zooming* oder *Cropping* verloren ginge.

Ferner wurden keine vertikalen Spiegelungen auf die Bilder ausgeführt, da diese bei *Fachboden* auf denen sich abgestellte Geräte befinden, unrealistische Daten generieren würden.

¹⁵<https://www.kaggle.com/>

Die horizontale Spiegelung hat wiederum keinen Einfluss auf den Realismus der Daten und wurde dementsprechend verwendet, um mehr Daten zu generieren. Die Weichzeichner, sowie Kontrast- und Farbveränderungen sollen die Generalisierungsfähigkeit des Detektors verbessern, da Objekte auch in Fotos unterschiedlicher Konditionen, wie Aufnahmen in einer schwächeren Belichtung oder mit einem suboptimalen Fokus, zu erkennen sind.

In [Abbildung 5.4](#) sind verschiedene Ergebnisse durch die Anwendung der *Data-Augmentation* visualisiert.



Abbildung 5.4: Visualisierung der Augmentationen für das Training von *Mask R-CNN* durch die „Augmentations-Pipeline“ auf das Rack aus der [exemplarischen Steckplatzanalyse](#).

Abschließend wurden die augmentierten Bilder und Annotationen in die von *PyTorch* geforderten Tensoren konvertiert.

Implementierung des Modells

Für die Umsetzung des *Mask R-CNN*-Detektors wurde die vorimplementierte Version von *PyTorch* verwendet [[PyT22a](#)]. Diese ermöglicht die Auswahl eines *backbones*. Entsprechend He et al. [[HGDG17](#)] werden in dieser Arbeit ebenfalls auf *ResNet* basierende *FPNs* als *backbone* eingesetzt, um das bestmögliche Erkennen von Objekten in unterschiedlichen Skalierungen zu ermöglichen. Genauer wurden die folgenden von *PyTorch* implementierten *FPN*-Versionen für das Training gewählt:

- *ResNet50*, *ResNet101*, *ResNet152*
- *WideResNet101*
- *ResNeXt101*

Um die Trainingsdauer zu verkürzen und den Mangel an Daten auszugleichen, wurden die durch *PyTorch* angebotenen, vortrainierten Gewichte und Schwellenwerte auf den *ImageNet-Datensatz* für diese *CNNs* verwendet.

Training

Für das Training wurden die von *torchvision* vorgefertigten Trainingsskripts genutzt und angepasst.

Zunächst wurde ein Feintuning der verschiedenen *Mask R-CNN*-Modelle, die aus den fünf *backbones* entstanden, verglichen.

Genauer wurden zunächst die Parameter der von He et al. [HZRS16] bezeichneten *Layer*: „conv5_x“, „conv4_x“ und „conv3_x“, demnach die tieferen Schichten des *CNN*’s, mittels *Feintuning* trainiert.

Dadurch, dass die Parameter der *backbones* aus dem Training auf den *ImageNet-Datensatz* resultieren, wurde vermutet, dass das alleinige Training auf die *Merkmalsextraktionen* nicht ausreichend ist, da sich die Objekte des *ImageNet-Datensatzes* stark von dem *Rack-Datensatz* unterscheiden.

Das Feintuning wurde basierend auf den folgenden Hyperparametern ausgeführt.

- Optimierungsverfahren: *SGD*
- Fehlerfunktionen: *multi-task loss* aus der Summe der folgenden Fehlerfunktionen, siehe [HGDG17]:
 - *Cross Entropy* für den Klassifizierungsfehler
 - *Smooth L1 Loss*¹⁶ [Gir15] für den Fehler bei der Regression der Begrenzungsrahmen und *RPN*-Vorhersagen
 - *Binary Cross Entropy with Logits*¹⁷ für den *Objectness Score*- und Segmentierungsfehler.
- Lernrate: 0.01
- *Nesterov-Momentum*: 0.9
- *weight decay*: 0.0001
- *mini batchsize*: 4
- Epochenanzahl: 30

Da sowohl Girshick et al. in [Gir15] als auch Ren et al. in [RHGS15] *SGD* als Optimierungsverfahren wählten, wurde jenes Verfahren auch zunächst in dieser Arbeit mit einer *mini batchsize* von 4 angewandt. Größere *Batch*-Größen konnten aufgrund der Bildgröße nicht von der *GPU* eingelesen werden.

Der *multi-task loss* wurde automatisch durch das von *torchvision* implementierte *Mask R-CNN* errechnet.

Für die Parameteranpassungen wurde sich zunächst für eine Lernrate von 0.01 mit einem *Nesterov-Momentum* [SMDH13] von 0.9 entschieden.

Dieses *Nesterov-Momentum* (auch als *Nesterov accelerated Momentum*) bezeichnet, stellt eine verbesserte Version des einfachen Momentums [Qia99] beim *SGD* dar, welches die Schrittrichtung basierend auf vorher berechneten Gradienten anpasst. Das *Nesterov accelerated Momentum* berechnet den Gradienten und die daraus folgende Schrittrichtung

¹⁶Laut Girshick et al. [Gir15] eine robuste Version der L_1 -Fehlerfunktion, welche auch als mittlerer absoluter Fehler (engl. *mean absolute error (MAE)*) bezeichnet wird.

¹⁷Dabei wird die *Binary Cross Entropy* mit der *Sigmoidfunktion* kombiniert.

basierend auf einem durch das Momentum erreichten Schrittes und ist dadurch laut Sutskever et al. [SMDH13] in der Lage bessere Optimierungsergebnisse zu erzielen.

Zusätzlich wurde die Lernrate während des Trainings angepasst, um schneller, bessere Ergebnisse während des Trainings zu erzielen. Dafür wurde die Python Bibliothek *pytorch-gradual-warmup-lr* [Ild19] benutzt, welche das *Gradual Warmup* aus [GDG⁺17] umsetzt. Die daraus resultierende adaptive Lernrate ist in [Abbildung 5.5d](#) visualisiert. Außerdem wurde ein *weight decay* [IYA16, S. 117] von 0.0001 gewählt, jenes soll dabei helfen *Overfitting* zu verhindern, indem es große Gewichtswerte bei der *backpropagation* bestraft und in die Fehlerfunktion einbezieht.

Bei den ersten Trainingsiterationen wurde deutlich, dass die berechneten Fehlergradienten das Minimum übersprangen und unendlich große Fehlerwerte produzierten. Jenes Verhalten wird auch als *exploding gradient problem* bezeichnet [IYA16, S. 286].

Eine Möglichkeit dieses Problem zu lösen, wäre die Lernrate zu verringern, jedoch würde dies in einem längeren Training resultieren. Außerdem erhöht es die Wahrscheinlichkeit, dass das Modell in einem nicht optimalen, lokalen Minimum stecken bleiben könnte. Eine weitere Möglichkeit dieses Problem zu lösen stellt das Verwenden von *gradient clipping* [IYA16, S. 408 ff] dar. Dadurch wird der errechnete Gradient mit dem aus der [Gleichung 5.2](#) errechneten Koeffizienten multipliziert, um zu große Lernschritte zu verhindern. Dafür werden alle Gradienten quadriert, aufsummiert und dessen Quadratwurzel bestimmt. Das genaue Vorgehen wurde aus einer *GitHub-Repository* [JDD17] einer *Faster R-CNN* Implementierung übernommen, bei welcher C den Wert 10 erhielt.

$$T = \sqrt{\sum_{i=1}^n grad_i^2} \quad (5.1)$$

$$grad_{coeff} = \frac{C}{\max(T, C)} \quad (5.2)$$

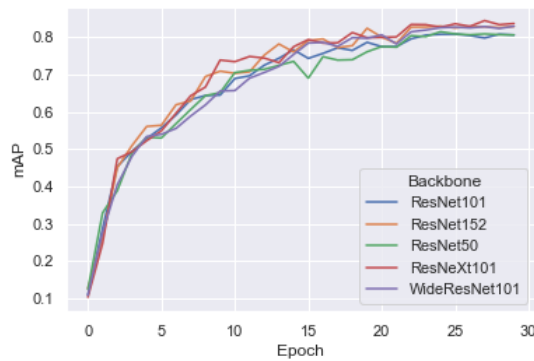
Während des Trainings wurde das Modell nach jeder Epoche auf den Validierungsdatensatz getestet und dessen [mAP](#) ermittelt. Erzielte das Modell nach einer Epoche bessere [mAP](#)-Werte, wurden die Gewichts- und Schwellenwerte abgespeichert, sodass letztlich das Modell, welches die beste Performance auf den Validierungsdatensatz aufwies, abgespeichert wurde. In [Abbildung 5.5](#) sind die durchschnittlichen Fehlerwerte der einzelnen Fehlerfunktionen, sowie die erzielten [mAP](#) auf den Validierungsdatensatz im Verlauf des Trainingsprozesses visualisiert.

In [Abbildung 5.5b](#) wird deutlich, dass das *Mask R-CNN* mit *ResNeXt101* als *backbone* die Höchste [mAP](#) auf den Validierungsdatensatz erzielen konnte. Demnach wurde jenes Modell für weitere Hyperparameteranpassungen verwendet.

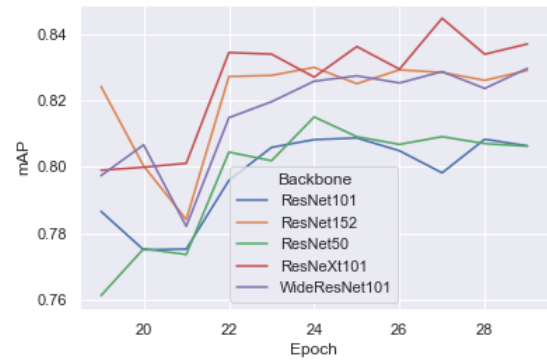
Diese Anpassungen beinhalteten das Verändern der initialen Lernrate auf 0.05 oder 0.001. Jene Durchläufe erzielten jedoch schlechtere [mAP](#)-Werte. Zusätzlich wurde überprüft, welchen Einfluss die Anzahl der zu trainierenden Parameter auf die [mAP](#) hat. Durch die Einbeziehung der Parameter aus der von He et al. [HZRS16] bezeichneten Schicht *conv2_x* konnten noch bessere [mAP](#)-Werte erreicht werden, siehe [Abbildung 5.6](#).

Anschließend wurde mit *AdamW* [IF19] ein Durchlauf mit einem anderen Optimierungsverfahren ausgeführt. Die [mAP](#) die durch [SGD](#) erzielt wurde, übertraf jedoch das Ergebnis dieses Durchlaufs.

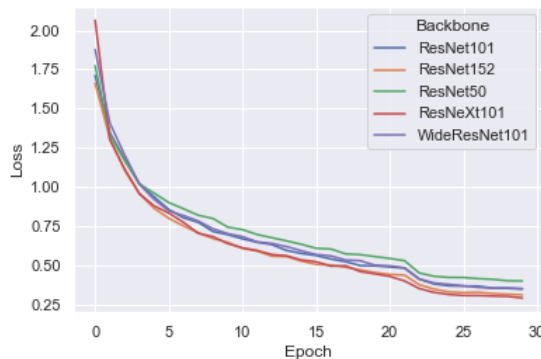
Abschließend wurden die bislang optimalen Hyperparameter verwendet, um ein Modell in 300 Epochen mit einer angepassten adaptiven Lernrate feinzutunen. Die adaptive Lernrate



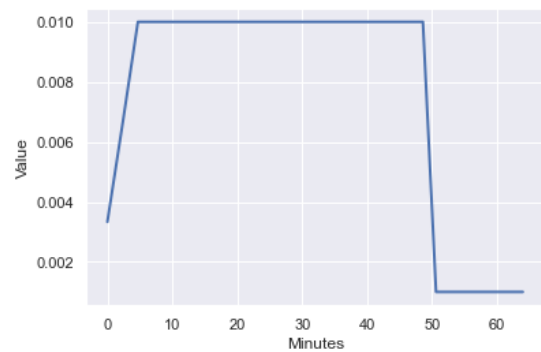
(a) mAP im Verlauf des Trainings.



(b) mAP im Verlauf der letzten 10 Epochen des Trainings.



(c) Die durchschnittlichen Fehlerwerte der einzelnen Fehlerfunktionen im Verlauf des Trainings.



(d) Die adaptive Lernrate im Verlauf des Trainingsprozesses.

Abbildung 5.5: Visualisierung der adaptiven Lernrate und ein Vergleich der erzielten *mean Average Precision* und durchschnittlichen Fehlerwerte der Fehlerfunktion im Verlauf des Trainings der *Mask R-CNN* Modelle mit unterschiedlichen *backbones*.

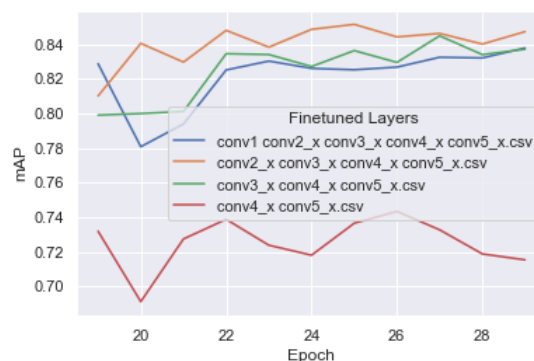
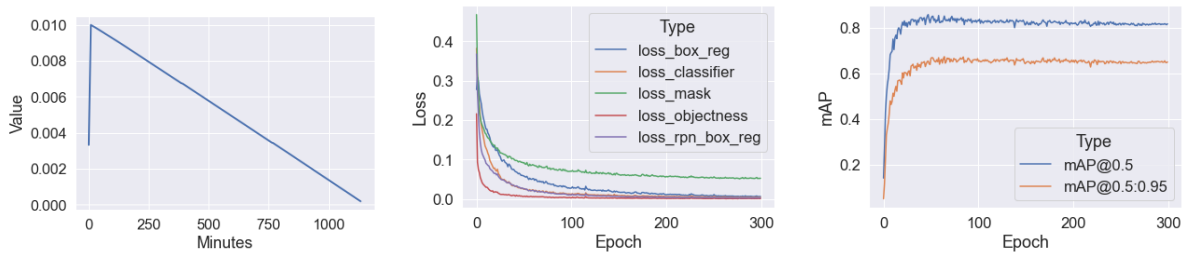


Abbildung 5.6: Vergleich der *mean Average Precision* während der letzten zehn Epochen basierend auf der Auswahl der zu trainierenden Parametern des *Mask R-CNN*-Modells mit *ResNeXt101* als *backbone*.



(a) Adaptive Lernrate im Verlauf des Trainings. (b) Fehlerwerte der einzelnen Fehlerfunktionen im Verlauf des Trainings. (c) Visualisierung der *mean Average Precision* auf den Validierungsdatensatz basierend auf unterschiedliche *IoU*-Schwellenwerte.

Abbildung 5.7: Visualisierung der Lernrate, Fehlerwerte und *mean Average Precision* im Verlauf des 300 epöchigen Durchlaufs des *Mask R-CNN* Feintunings.

Run	mAP@0.5	mAP@0.5:0.95	Score	Epoche
30_Epochen	0.851430	0.655208	0.674830	26
300_Epochen	0.851392	0.673105	0.690934	65

Tabelle 5.4: Gegenüberstellung der besten Trainingsepochen des *Mask R-CNN*-Modells mit *ResNeXt101* als *backbone* und den vier trainierbaren Schichten, basierend auf dem festgelegten Gütekriterium.

dieses Durchlaufs ist in [Abbildung 5.7a](#) visualisiert und dessen Inspiration stammt aus der *YOLOv5* Implementierung, auf welche in [Unterabschnitt 5.1.4](#) detaillierter eingegangen wird.

Um Zugriff auf die besten Modell-Parameter zu bekommen, wurde in diesem Durchlauf erneut nach jeder Epoche, die *mAP* auf den Validierungsdatensatz ermittelt. Zuzüglich wurde nach jeder Epoche die *mAP*@0.5:0.95 auf den Validierungsdatensatz bestimmt.

Die Güte des Modells wurde durch [Gleichung 5.3](#) festgelegt. Die Inspiration des Gütekriteriums stammt ebenfalls aus der *YOLOv5* Implementierung.

$$Score = (mAP_{0.5} \times 0, 1) + (mAP_{@0.5:0.95} \times 0, 9) \quad (5.3)$$

In [Abbildung 5.7](#) sind neben der adaptiven Lernrate auch die Fehlerwerte resultierend aus den Fehlerfunktionen im Verlauf des Trainings sowie die *mAP*, basierend auf unterschiedlichen *IoU*-Schwellenwerten, visualisiert.

Die besten Ergebnisse des 30 epöchigen Durchlaufs werden in [Tabelle 5.4](#) mit den Ergebnissen der besten Epoche des 300 epöchigen Durchlaufs gegenübergestellt. Darin ist ersichtlich, dass die beste Epoche des 300 epöchigen Durchlaufs bereits in Epoche 65 erreicht wurde. Im weiteren Verlauf begann das Modell ein leichtes *Overfitting* auszulösen. Trotzdem der Fehler auf den Trainingsdatensatz weiter minimiert wurde, verschlechterte sich die *mAP* auf den Validierungsdatensatz siehe [Abbildung 5.7](#).

5.1.4 Implementierung des YOLOv5-Detektors

Für die Umsetzung des *YOLOv5*-Detektors wurde die Open-Source-Implementierung [GAA⁺22] übernommen.

Diese beinhaltet neben unterschiedlichen *YOLOv5*-Modellen auch vorgefertigte Trainings- und Evaluierungsskripts.

Anpassen des Rack-Datensatzes

Damit das Modell durch das Trainingsskript auf den Rack-Datensatz trainiert werden konnte, musste dieser angepasst werden.

Das Trainingsskript setzt eine bestimmte Verzeichnisstruktur für den Datensatz voraus. Dieser musste die Verzeichnisse *images* und *labels* enthalten. Im *images*-Verzeichnis sollten die Bilder des Datensatzes abgelegt werden.

Um den Vorgaben des *YOLOv5x6*-Modells zu entsprechen, wurden jene mittels *zero-padding* auf eine quadratische Größe von 1280×1280 Pixel skaliert.

Das *labels*-Verzeichnis sollte zu jedem Bild des *images*-Verzeichnis eine gleichnamige *.txt*-Datei enthalten. Im Gegensatz zum *Mask R-CNN*-Detektor benötigt *YOLO* die Begrenzungsrahmen im *x, y, w, h* Format mit normalisierten Werten von 0 bis 1. *x* und *y* mussten sich auf die Koordinaten des Rahmenmittelpunkts beziehen, *w* und *h* wiederum die Breite und Höhe des Rahmens beschreiben. Die Normalisierung der Werte *x* und *w* sollte aus der Division durch die Breite des Bildes gewonnen werden. Die Normalisierung der Werte *y* und *h* wiederum aus der Division durch die Bildhöhe.

Die entsprechende *.txt*-Datei eines Fotos musste für jedes annotierte Objekt eine Zeile in der Form *Klasse, x, y, w, h* beinhalten.

Ferner sollten die Bild- und Textdateien in Unterverzeichnisse für die Unterteilung des Datensatzes in *Train* und *Validation* abgelegt werden.

YOLOv5x6

Die *YOLOv5* Implementierung bietet verschiedene *YOLOv5* Versionen, inklusive vortrainierter Gewichts- und Schwellenwerte an. Die Gewichts- und Schwellenwerte wurden aus dem Training auf den *COCO*-Datensatz gewonnen.

Für diese Arbeit wurde die Version *YOLOv5x6* für das Training gewählt, da diese laut Jocher et al. [GAA⁺22] die höchste *mAP* auf den *COCO*-Datensatz aufweist.

Training

Für das Training wurde das vorgefertigte Trainingsskript mit dessen Standardkonfiguration genutzt.

Demnach wurden die folgenden Hyperparameter verwendet:

- Optimierungsverfahren: *SGD*
- Fehlerfunktionen:

- *Binary Cross Entropy with Logits Loss* für den Klassen- und *Objectness Score*-Fehler.
- *Complete IoU Loss*[ZWL⁺20] für den Fehler der Begrenzungsrahmen.
- Lernrate: 0,01
- *Nesterov-Momentum*: 0,937
- *weight decay*: 0,0005
- *mini batchsize*: 3 (höchstmögliche Größe wurde automatisch durch das Skript festgestellt)
- Epochenanzahl: 300

Das Trainingsskript beinhaltet ebenfalls vorgefertigte Augmentationen, welche unter anderem mit der *Albumentations*-Bibliothek [BIK⁺20] implementiert wurden. Dadurch wurden die folgenden Augmentationen ausgeführt:

- Mit *Albumentations* ausgeführte Augmentationen, jeweils mit einer Wahrscheinlichkeit von 1%
 - **Blur** (Bild wird mit einem Kernel zufälliger Größe weichgezeichnet)
 - **MedianBlur** (Bild wird durch einen *Median-Filter* weichgezeichnet)
 - **ToGray** (Bild wird in ein Graustufenbild umgewandelt)
 - **CLAHE** (Auf das Bild wird kontrastlimitiertes adaptiertes Histogrammebenen [PAA⁺87] angewendet)
 - **RandomBrightnessContrast** (Helligkeit und Kontrast des Bildes wird zufällig verändert)
- Anderweitig implementierte Augmentationen
 - **HSV** (Bild wird in das *HSV-Farbsystem* konvertiert und dessen Kanäle anschließend zufällig verändert)
 - Horizontale Spiegelung mit einer Wahrscheinlichkeit von 50%

In [Abbildung 5.8](#) sind verschiedene Ergebnisse unter Anwendung der vorgefertigten „Augmentations-Pipeline“ visualisiert.

Epoche	mAP@0.5	mAP@0.5:0.95	Score
271	0.84947	0.716721	0.729996

Tabelle 5.5: Darstellung der besten Trainingsepoche des *YOLOv5x6*-Modells bemessen am Gütekriterium.



Abbildung 5.8: Visualisierung der Augmentationen für das Training von *YOLO* durch die „Augmentations-Pipeline“ auf das Rack der exemplarischen Steckplatzanalyse

Für die Gewichts- und Schwellenwertänderungen wurde eine adaptive Lernrate eingesetzt, siehe [Abbildung 5.9](#). Genauer verwendet das Trainingskript, je nach Parametertyp, unterschiedliche Lernraten für die Anpassung der Werte.

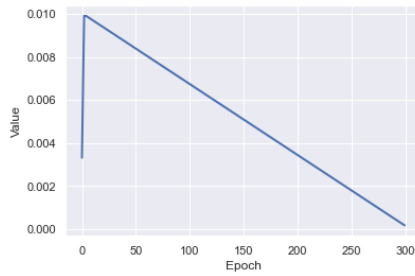
- **lr0:** für Gewichtswerte bei welchen *weight decay* in die Gewichts Anpassung einbezogen wird
- **lr1:** für Gewichtswerte bei welchen kein *weight decay* einbezogen wird (*Batch-Normalization*¹⁸[[IYA16](#), S. 313 ff]-Gewichte)
- **lr2:** für die Anpassung der Schwellenwerte

Ähnlich zum Training des *Mask R-CNNs* wurde das Modell während des Trainings nach jeder Epoche evaluiert und dessen *Precision*, *Recall* und [mAP](#) auf den Validierungsdatensatz berechnet. Das Modell, welches während des Trainings die besten Evaluierungsergebnisse lieferte, wurde abgespeichert und somit ein *Overfitting* auf den Trainingsdatensatz minimiert. Die Güte des Evaluierungsergebnisses wurde mittels [Gleichung 5.4](#) bestimmt.

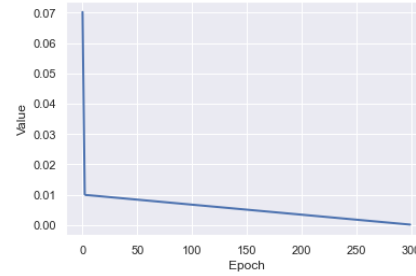
$$Score = (mAP_{0.5} \times 0,1) + (mAP_{@0.5:0.95} \times 0,9) \quad (5.4)$$

Die durch das Training erzielte Minimierung der Fehlerfunktionen auf den Trainings- und Validierungsdatensatz sind in [Abbildung 5.10](#) und die erreichten Genauigkeitswerte in [Abbildung 5.11](#) visualisiert. Die beste Epoche des Trainings ist in [Tabelle 5.5](#) dargestellt.

¹⁸*Batch-Normalization* optimiert das Training von tiefen *KNNs*, in dem es die Ausgaben der Neuronen in einem Netz normalisiert.

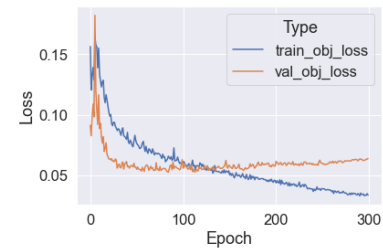
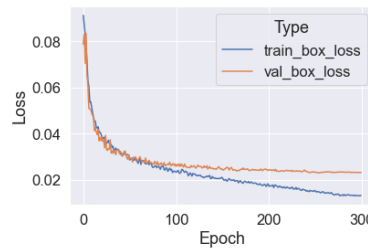
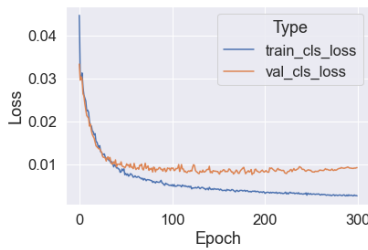


(a) Lernraten **lr0** und **lr1** im Verlauf des Trainings



(b) Lernraten **lr2** im Verlauf des Trainings

Abbildung 5.9: Visualisierung der Anpassung der Lernraten während des Trainingsprozesses vom YOLOv5x6-Modell.



(a) Fehlerwerte resultierend aus der Fehlerfunktion für die Klassifizierung. (b) Fehlerwerte resultierend aus der Fehlerfunktion für die *Bounding Box Regression*. (c) Fehlerwerte resultierend aus der Fehlerfunktion für den *Objectness Score*.

Abbildung 5.10: Visualisierung der Fehlerwerte der einzelnen Fehlerfunktionen auf den Validierungs- und Trainingsdatensatz während des Trainingsprozesses vom YOLOv5x6-Modell.

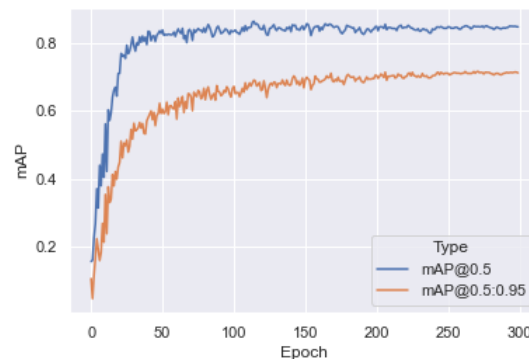


Abbildung 5.11: Visualisierung der **mAP@0.5** und **mAP@0.5:0.95** des YOLOv5x6-Modells auf den Validierungsdatensatz im Verlauf des Trainings.

5.2 Implementierung des Postprocessings

In diesem Abschnitt wird die Umsetzung des *Postprocessings* der Objekterkennung aus [Unterabschnitt 4.2.2](#) beschrieben und dessen Resultate visualisiert. Die dafür entwickelten Algorithmen sind als Pseudocode im [Anhang](#) hinterlegt.

Das *Postprocessing* beinhaltet zum einen die Optimierung und Filterung der erkannten Vorhersagen, welches mittels [Algorithmus A.1](#) erreicht wird. Um die Überprüfung auf Objekt-Überschneidungen entsprechend des [Konzepts](#) zu ermöglichen, werden die Vorhersagen durch [Algorithmus A.3](#), bezogen auf einen Schwellenwert für *Konfidenz-Werte* gefiltert, auf 100% der Bildbreite vergrößert und basierend auf dessen oberen y-Koordinate des Begrenzungsrahmens aufsteigend sortiert. Dafür wird das von *Python* implementierte Sortiervorgehen *TimSort* [[AJNP18](#)] genutzt. Des Weiteren werden Objektvorhersagen der Klasse *Slots* herausgefiltert, da die Erkennung eines solchen Objekts lediglich zur Validierung eines Rack-Fotos dienen sollte.

Die optimierten Vorhersagen aus [Algorithmus A.1](#) werden im [Algorithmus A.2](#) genutzt, um die eigentliche Steckplatzanalyse auszuführen. Analog zum [Konzept](#) werden die jeweiligen Steckplatzpositionen und normierten Größen der vorhergesagten Objekte mittels der [PPMR](#) ermittelt. Zusätzlich werden gänzlich freie Positionen, welche der Objektdetektor nicht erkennen sollte, als freie Steckplatzposition inklusive normierter Größe in die Analyse aufgenommen. [Algorithmus A.4](#) beschreibt, aus welchen Informationen ein solch nachträglich erstelltes Objekt besteht. Der letztendliche binäre Auslastungszustand wird anhand der Objektklasse klassifiziert. Dabei sollten *Fachböden*, die kein darauf abgestelltes Gerät enthalten als freier Steckplatz klassifiziert werden. Jene Informationen wurden durch [Algorithmus A.5](#) gewonnen und in einem einheitlichen Schema zurückgegeben.

Abschließend galt es, die aus dem [Algorithmus A.2](#) resultierenden Daten der Steckplatzanalyse in einem adäquaten Datenschema darstellbar zu machen.

Die Ausgabe *A* des Algorithmus hat die Form:

$n \times [bbox, label, ru, isEmpty, startPos, endPos]$, wobei *n* die Anzahl der analysierten Steckplatzobjekte beschreibt. Jene Ausgabe erlaubt die Möglichkeit, die Daten der Steckplatzauslastung in einer Tabelle darzustellen, welche aus der Endposition des letzten erkannten Steckplatzobjekts, demnach aus $A_{n-1}.endPos$, Zeilen mit den Spalten:

$[Position, istFrei, RU, Typ]$ besteht. Der Zeilenindex beschreibt die exakte Steckplatzposition in der Spalte *Position* und die Zeilenanzahl die Höhe des analysierten Racks.

Jedes Objekt aus *A* wird in die Zeilen der Tabelle eingeordnet, bei welchen die Startposition kleiner gleich dem Zeilenindex und die Endposition größer als der Zeilenindex ist. Die jeweiligen Spalteninformationen *istFrei*, *RU* und *Typ* werden aus den Informationen des Objekts übernommen. *A* beinhaltet, außer für *Fachböden* und den darauf abgestellten Geräten, für jede Zeile nur ein Objekt. Bei mehreren Objekten pro Steckplatz resultieren die Informationen aus dem Objekt, welches als Erstes in die Tabelle übernommen wird, da dieses einen höheren Startpunkt besitzt und die weiteren Objekte des Steckplatzes nach Anwendung der Algorithmen nur innerhalb des ersten Objekts enthalten sein können. In der Spalte *Typ* werden die Informationen bei mehreren Objekten pro Steckplatz mit einem Komma voneinander abgetrennt.

In [Abbildung 5.12a](#) ist das Ergebnis der reinen Steckplatzerkennung durch den *Mask R-CNN*-Detektor an einem Beispiel visualisiert. In [Abbildung 5.12b](#) wird hingegen das



(a) Objekterkennungsergebnisse ohne festgelegten *Konfidenz*-Schwellenwert. (b) Optimierte Objektvorhersagen durch [Algorithmus A.1](#) mit den beschriebenen Schwellenwerten. (c) Vollständige Steckplatzanalyse durch [Algorithmus A.2](#) mit den beschriebenen Schwellenwerten.

Abbildung 5.12: Visualisierung der algorithmischen Teilschritte der Steckplatzanalyse mittels des *Mask-RCNN*-Detektors mit einem *Konfidenz*-Schwellenwert von 0.6 und einem Freiflächen-Schwellenwert (*empty-Threshold*) von 0.5 anhand eines Beispielbildes aus dem Validierungsdatensatz.

optimierte Ergebnis durch [Algorithmus A.1](#) dargestellt. Letztlich ist in [Abbildung 5.12c](#) das Ergebnis der gesamten Steckplatzanalyse des Beispielracks visualisiert und in [Tabelle 5.6](#) als Tabellenstruktur dargestellt.

5.3 Einbindung in die Micro-TSS-App

Für die Umsetzung des Konzepts wurde ein *Client-Server-Modell* [[CHM⁺07](#)] implementiert. Dieses besteht aus der *Micro-TSS-App* als *Client* und einem mittels dem Framework *Flask*¹⁹ entwickeltem *Web-Server*.

Unter [Abbildung A.1](#) im Anhang ist die Software-Architektur als ein *C4 Container Diagram* [[VIGHGP20](#)] dargestellt.

Der *Client* erlaubt den Benutzer:innen die Fotos für die Steckplatzanalyse auszuwählen und die daraus resultierenden Ergebnisse in der Benutzeroberfläche visualisiert zu bekommen. Der *Server* hingegen bekommt die Anfragen des *Clients* und führt die entsprechenden Steckplatzanalysen mithilfe des trainierten Objekt-Detektors und dem *Postprocessings* aus und leitet die Ergebnisse letztlich an den *Client* weiter.

¹⁹<https://palletsprojects.com/p/flask/>

Position	istFrei	RU	Typ
0	True	1	empty
1	False	1	occupied, level
2	False	1	device
3	True	1	blend
4	False	1	device
5	True	1	empty
6	False	6	occupied
7	False	6	occupied
8	False	6	occupied
9	False	6	occupied
10	False	6	occupied, level
11	False	6	occupied, level
12	True	2	empty
13	True	2	empty

Tabelle 5.6: Darstellung des Steckplatzanalyseergebnisses mit einem *Konfidenz*-Schwellenwert von 0.6 und einem Freiflächen-Schwellenwert (*emptyThresh*) von 0.5 am Beispiel des Bildes aus [Abbildung 5.12](#)

5.3.1 Server

Für die Umsetzung des *Servers* wurde das *Python-Framework Flask* gewählt. Zum einen ermöglicht *Flask* die einfache Einbindung des Objekt-Detektors inklusive des darauffolgenden Programmcodes für das *Postprocessing* aus [Abschnitt 5.2](#). Zum anderen wird in den *PyTorch*- und *YOLOv5* Dokumentationen [[Avi22](#), [GSR22](#)] eine Anleitung zum Entwurf eines *Flask-Web-Servers* für die Inferenz angeboten.

Auf Basis dieser wurde ein *Representational State Transfer-Application Programming Interface (ReST-API)* [[FT02](#)] implementiert, welches die vom *Client* gesendeten Fotos und Polygoninformationen empfängt und die Objekterkennung ausführt. Dabei enthält das *Application Programming Interface (API)* zwei Endpunkte für die Steckplatzanalyse. Jeweils einen für jeden Objektdetektor (*Mask R-CNN* und *YOLOv5*). Diese Endpunkte erwarteten einen *JSON-HTTP-Post-Request*, welches „Schlüssel-Wert-Paare“ [[Ple17](#)] analog zum Beispiel aus [Listing 5.1](#) enthält.

Listing 5.1: JSON-Schema für die Anfragen ReST-API-Endpunkts

```

0 {
1     "images": ["b64String"],
2     "polypoints": [
3         [[307.0, 842.0],
4          [350.0, 3778.0],
5          [2813.0, 3818.0],
6          [2822.0, 809.0]
7         ]],
8     "draw": [true],
9     "filenames": ["20220207_121328.jpg"]
10 }
```

Für den Schlüssel *images* wird ein *Array* aus den *n* zu analysierenden Fotos als einzelne *Base64* [Jos03] kodierte *Strings* erwartet.

Für *polypoints* hingegen ein *Array* aus *n* zweidimensionalen *Arrays* in einer *String*-Darstellung. Das zweidimensionale *Array* enthält die vier Punkte des Polygons, welches das „Einbauchassis“ eines Fotos markieren soll. Die Punkte werden innerhalb des *Arrays* als *Array* aus x- und y-Koordinate dargestellt. Dessen Reihenfolge ist für den *Server* irrelevant, da diese vor der perspektivischen Transformation automatisch angepasst wird. Für den Schlüssel *draw* wird ein *Array* aus *n* *Boolean*-Werten verlangt. Je nach Wahrheitswert wird vom *Server* eine Visualisierung der Steckplatzanalyse des jeweiligen Fotos zurückgegeben.

Abschließend enthält *filenames* ein *Array* aus den *n* Dateinamen der zu analysierenden Fotos.

Die enthaltenen Fotos werden identisch zur [Vorverarbeitung der Trainingsbilder](#) basierend auf den Polygoninformationen perspektivisch transformiert und durch den jeweiligen Objekt-Detektor und dem [Postprocessing](#) analysiert.

Steckplatzanalysen, welche aus mehreren Teilen bestehen, fassen die jeweiligen Analysen der Teilfotos zusammen. Dafür werden die Steckplatzpositionen der Objekte mit der Endposition des letzten Objekts des vorherigen Teilfotos addiert. Die Steckplatzpositionen der Objekte des ersten Fotos bleiben dabei unverändert.

Des Weiteren werden Freiflächen, welche in das nächste Teilfoto übergehen, vereinigt und dessen *RU*-Höhe an die vereinigte Größe angepasst.

Anschließend werden die zusammengefassten Analysen als *JSON* kodiert, analog zum Beispiel aus [Listing 5.2](#), zum *Client* übertragen.

Listing 5.2: JSON-Schema für die Antworten des ReST-API-Endpunkts

```

0 {
1     "drawing": ["b64String"],
2     "rack": {
3         "empty": 1,
4         "occupied": 1,
5         "predictions": [{
6             "Position": 0,
7             "RU": 1,
8             "Typ": "device",
9             "istFrei": false
10        }, {
11            "Position": 1,
12            "RU": 1,
13            "Typ": "blend",
14            "istFrei": true}],
15        "rack_height": 2
16    }
17 }
```

Für jedes Teilfoto, für welches der Schlüssel *draw* aus der Anfrage wahr ist, wird ein visualisiertes Ergebnis als *Base64* kodierter *String* in einem *Array*, unter dem Schlüssel *drawing* zurückgegeben. Der Schlüssel *rack* beinhaltet die zusammengefasste Steckplatzauslastung mit den Informationen *empty*, *occupied* und *rack_height*, welche die Anzahl der

belegten und freien Steckplätze sowie die gesamte Rack-Höhe beschreiben. Zusätzlich ist unter *predictions*, die Steckplatzauslastung, inklusive der Informationen zu den einzelnen Komponenten, analog zur tabellarischen Darstellung aus [Tabelle 5.6](#), als ein Array aus *JSON*-Objekten enthalten.

5.3.2 Client

Die ursprüngliche Version der *Micro-TSS-App* wurde mittels *Android Studio*²⁰ und *Java Version 11* nativ für Android-Smartphones ab Version 8 entwickelt. Für diese Arbeit wurden die Funktionalitäten der App gemäß den Anforderungen des aufgestellten Konzepts erweitert.

Benutzeroberfläche

Die Benutzeroberfläche der App besteht aus vier Hauptbildschirmen, welche sich durch eine Navigationsleiste am unteren Bildschirmrand aufrufen lassen. Im Bildschirm für administrative Aufgaben wurde ein Button für die Steckplatzanalyse hinzugefügt. Dieser Button leitet die Benutzer:innen zum Bildschirm für die Steckplatzanalyse weiter.

Dort wird den Benutzer:innen ein Button für das Aufnehmen und Auswählen von Fotos, sowie ein Button zur Ausführung der Steckplatzanalyse dargestellt, siehe [Abbildung 5.13a](#). Bei erstmaliger Benutzung wird eine Hilfestellung zur Benutzung der Steckplatzanalyse-App gegeben, um eine bessere Anwenderfreundlichkeit zu garantieren. Um diese zu implementieren wurde die Bibliothek *ShowcaseView* [\[Cur16\]](#) verwendet.

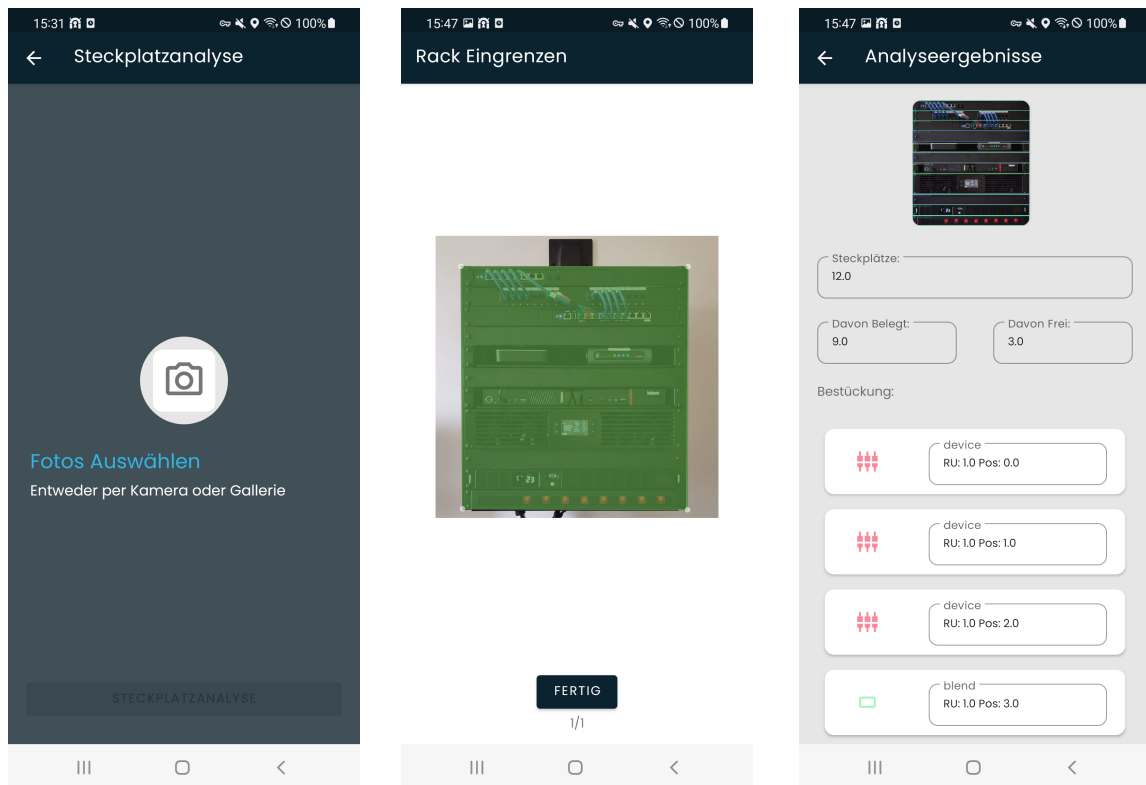
Die App erlaubt den Benutzenden die Auswahl, ob die Steckplatzanalyse basierend auf Aufnahmen der Smartphone-Kamera oder basierend auf Fotos, welche auf dem Smartphone gespeichert oder per Cloud zugänglich sind, ausgeführt werden soll.

Nachdem die zu analysierenden Fotos ausgewählt wurden, öffnet sich für jedes Foto ein Bildschirm zur Festlegung der Begrenzungen des „Einbauchassis“, ersichtlich in [Abbildung 5.13b](#). Dafür wird mittels der Open-Source-Bibliothek *floorplanner* [\[Sha21\]](#) ein vierpunktiges Polygon auf das Foto projiziert. Um fortzufahren, muss der Nutzer die Punkte des Polygons an den Begrenzungen des „Einbauchassis“ ausrichten.

Anschließend werden die zu analysierenden Fotos in dem vorherigen Bildschirm für die Steckplatzanalyse angezeigt und können analysiert werden.

Nachdem die Steckplatzanalyse vom *Server* erfolgreich ausgeführt wurde, wird ein Button angezeigt, der die Benutzenden auf einen Bildschirm weiterleitet, der das Ergebnis visualisiert. Auf diesem Ergebnisbildschirm wird ein Foto mit den erkannten Begrenzungsrahmen angezeigt, siehe [Abbildung 5.13c](#). Zusätzlich geben Textfelder die Größe des analysierten Racks und die Anzahl der darin enthaltenen freien und belegten Steckplätze an. Des Weiteren werden die erkannten Objekte inklusive der Spalteninformationen analog zur [Tabelle 5.6](#) in einer Liste visualisiert.

²⁰<https://developer.android.com/studio>



(a) Fotoauswahl-Bildschirm

(b) Eingrenzung des „Einbauchassis“

(c) Steckplatzanalyseergebnis

Abbildung 5.13: Bildschirmfotos aus der *Micro-TSS-App*

Datenübertragung

Die zu analysierenden Fotos und Polygoninformationen wurden über einen **Hypertext Transfer Protocol (HTTP) POST-Request** an den *Server* übertragen.

Dafür wurde der *HTTP-Client Retrofit* [SFI18] verwendet, da jener bereits für andere Aufgaben in der App genutzt wurde.

Die für die Steckplatzanalyse benötigten Daten wurden durch die *Java* Bibliothek *GSON* [Ple17] in die vom *Server* verlangte **JSON**-Darstellung konvertiert. Dabei wurden die ausgewählten Fotos zunächst in einzelne *Base64* [Jos03] *Strings* kodiert.

Die Ergebnisse wurden als **JSON** empfangen und durch *GSON* in *Java*-Objekte konvertiert. Dabei wurden die vom *Server* übermittelten Fotos der visualisierten Steckplatzanalyse ebenfalls in einer *Base64*-Codierung übertragen und zur Verwendung in der App in eine *Bitmap* also **Rastergrafik** umgewandelt.

6 Evaluierung

In diesem Kapitel wird die Überprüfung der Funktionsfähigkeit und Genauigkeit des implementierten Verfahrens erläutert.

Dafür wurde die Lösungsstrategie in „Real-Welt-Szenarien“ getestet und dessen Korrektheit kontrolliert.

Jene Szenarien wurde mittels eines weiteren Datenbestands von Rack-Fotos geschaffen, welcher in [Abschnitt 6.1](#) näher beschrieben ist.

Damit eine Bewertung des Prototyps vorgenommen werden konnte, wurde eine Strategie für die Evaluierung der Steckplatzanalyseergebnisse auf den Evaluierungsdatensatz entworfen. Diese ist in [Abschnitt 6.2](#) beschrieben. Jener Abschnitt beinhaltet neben dem Erklären des Vorgehens auch das Erläutern verschiedener Metriken, welche die Genauigkeit der Ergebnisse in unterschiedlichen Gesichtspunkten bemessen. Ferner wird in [Unterabschnitt 6.2.3](#) ein Gütekriterium bestimmt, welches die Evaluierungsergebnisse miteinander vergleichbar macht. Abschließend werden in diesem Abschnitt die Umsetzung und die unterschiedlichen Parameter der Evaluierung erläutert.

In [Abschnitt 6.3](#) werden die Ergebnisse der Evaluierung dargestellt und der Einfluss von Parameterveränderungen auf diese erörtert. Abschließend werden darin mögliche Fehlerquellen bei der Steckplatzanalyse identifiziert und die Ergebnisse hinsichtlich der Zielstellung der Arbeit diskutiert.

6.1 Daten der Testszenarien

Um das Verfahren auf Funktionsfähigkeit und Genauigkeit überprüfen zu können, mussten Daten erhoben werden, an welchen sich die Ergebnisse der Steckplatzanalyse validieren ließen.

Jene Daten sollten mehrere exemplarische Anwendungsszenarien der App simulieren und demnach aus Fotos verschiedener Racks bestehen. Da das Verfahren eine Analyse basierend auf einem oder mehreren Fotos erlaubt, sollte ein Rack je nach Aufnahmemöglichkeit mittels einem oder mehreren Fotos im Datensatz abgebildet werden.

6.1.1 Datenakquise

Um die Generalisierungsfähigkeit des Verfahrens überprüfen zu können, sollten Racks analysiert werden, welche nicht im bisherigen Trainings- und Validierungsdatensatz vorhanden waren. Dafür wurden Fotos im Rechenzentrum des *Fraunhofer-Instituts für Nachrichtentechnik, Heinrich-Hertz-Institut*¹ aufgenommen. Darin ließen sich sechs 47RU hohe

¹<https://www.hhi.fraunhofer.de/>

19-Zoll-Racks von dessen Vorder- und Rückseite fotografieren.

Zuzüglich wurden erneut Fotos im Rechenzentrum von *highstreet technologies* aufgenommen. Dabei wurden fünf 43RU hohe 19-Zoll-Racks von dessen Vorder- und Rückseite fotografiert. Entgegen der Racks des *Fraunhofer-Instituts* waren jene Racks bereits im Trainings- und Validierungsdatensatz vorhanden, wobei jedoch drei der fünf Racks einen veränderten Bestückungszustand aufwiesen.

Für die Evaluierung wurden die einzelnen Steckplätze der Racks mittels der Software *labelme* [Wad] annotiert. Die Annotierungsstrategie wurde jedoch an die Steckplatzanalyse angepasst. Im Gegensatz zum Trainings- und Validierungsdatensatz wurden anstatt der montierten Komponenten, die einzelnen Steckplätze getrennt voneinander nach folgendem Schema annotiert:

Steckplatznummer, Objektnummer, istFrei, RU, Typ

- Steckplatznummer: Position des Steckplatzes (0 bis RU-Höhe des Racks - 1, von oben nach unten)
- Objektnummer: Objektnummer des sich im Steckplatz befindenden Objekts (von oben nach unten)
- istFrei: boolescher Wert, der beschreibt, ob es sich um einen freien Steckplatz handelt
- RU: RU-Höhe des sich im Steckplatz befindenden Objekts
- Typ: Klasse des sich im Steckplatz befindenden Objekts (Freifläche, Blende, Baugruppe, Gerät, Fachboden)

Zusätzlich wurden in jedem Foto die Begrenzungen des „Einbauchassis“ markiert.

Mittels eines *Jupyter Notebooks* [RPG17] wurden die durch *labelme* gespeicherten Annotationsdateien verarbeitet und in einer **csv**-Datei für das jeweilige Rack und dessen fotografierte Seite abgespeichert. Dabei wurden die Annotationsinformationen von Racks, welche anhand von mehreren Fotos abgebildet wurden, in jener Datei zusammengefasst. Die **csv**-Datei enthielt zusätzlich zu den Annotationen, die Informationen über den Speicherpfad der zugrundeliegenden Fotos einer Annotation sowie die fotografierte Seite und die ID² des Racks.

Außerdem wurden die annotierten Begrenzungsrahmen der Steckplätze durch die perspektivische Transformation basierend auf den Begrenzungen des „Einbauchassis“ angepasst. Es wurde sich bewusst dagegen entschieden, die Annotationen auf die bereits perspektivisch transformierten Fotos auszuführen, um einen Vergleich zwischen verschiedenen perspektivischen Transformationsverfahren zu ermöglichen.

²Die ID der Racks richtete sich nach Aufnahmeort und Zeitpunkt. Das erste Rack, welches im Rechenzentrum des *Fraunhofer-Instituts* aufgenommen wurde, erhielt etwa die ID **hhi_Rack0**.

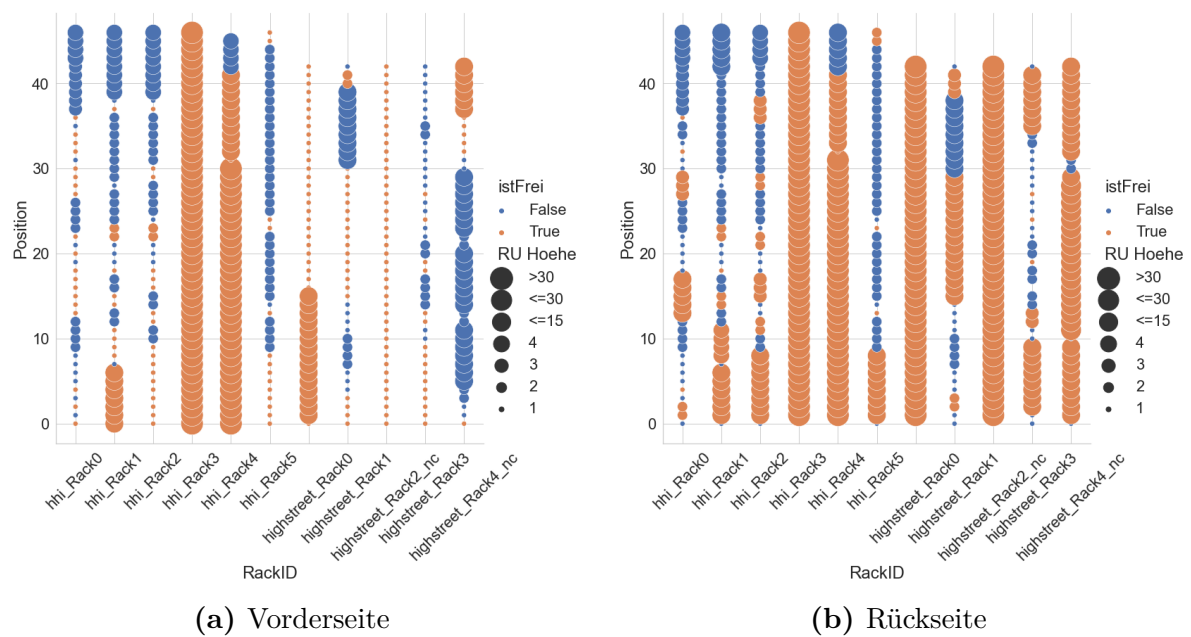


Abbildung 6.1: Visualisierung der einzelnen Steckplatzpositionen des Evaluierungsdatensatzes in Relation zur RU-Höhe und Auslastung des jeweiligen Steckplatzes

6.1.2 Analyse der Evaluierungsdaten

Insgesamt beinhaltet der Evaluierungsdatensatz 11 Racks mit 993³ Steckplätzen, welche mittels 52 Fotos abgebildet wurden. Dabei zeigten 29 Fotos die Vorderseite und 23 Fotos die Rückseite der Racks. Vereinzelt wurden sowohl durch ein einzelnes, als auch durch mehrere Fotos abgebildet und annotiert. Daraus resultierten 1087 annotierte Steckplätze, wovon 639 frei und 448 belegt waren.

In [Abbildung 6.1](#) sind die annotierten Steckplatzposition der einzelnen Racks in Relation zu dessen Größe und Auslastung visualisiert.

Darin ist ersichtlich, dass mehrere Racks nahezu vollständig aus freien Steckplätzen bestehen. Jene freien Steckplätze unterscheiden sich stark in dessen RU-Höhe, da die RU-Höhe von Freiflächen aus dem Abstand bis zum nächsten belegten Steckplatz resultiert, wohingegen die RU-Höhe einer *Blende* sich an der Größe jener *Blende* orientiert.

Wenngleich mehrere leere Racks im Datensatz vorhanden sind, ist das Verhältnis von freien zu belegten Steckplätzen mit ca. 3 : 2 nahezu ausgeglichen, was in [Abbildung 6.2](#) ersichtlich ist. Aus der Abbildung wird außerdem deutlich, dass die Objekttypen *Gerät* und *Fachboden* im Datensatz deutlich unterrepräsentiert sind, wodurch keine genaue Bewertung hinsichtlich der Erkennung dieser „Spezialfälle“ vorgenommen werden kann.

³Auf dem Foto der Vorderseite des Racks **hhi_Rack4** war der oberste Steckplatz nicht vollständig vorhanden, weswegen dieser nicht analysiert werden konnte.

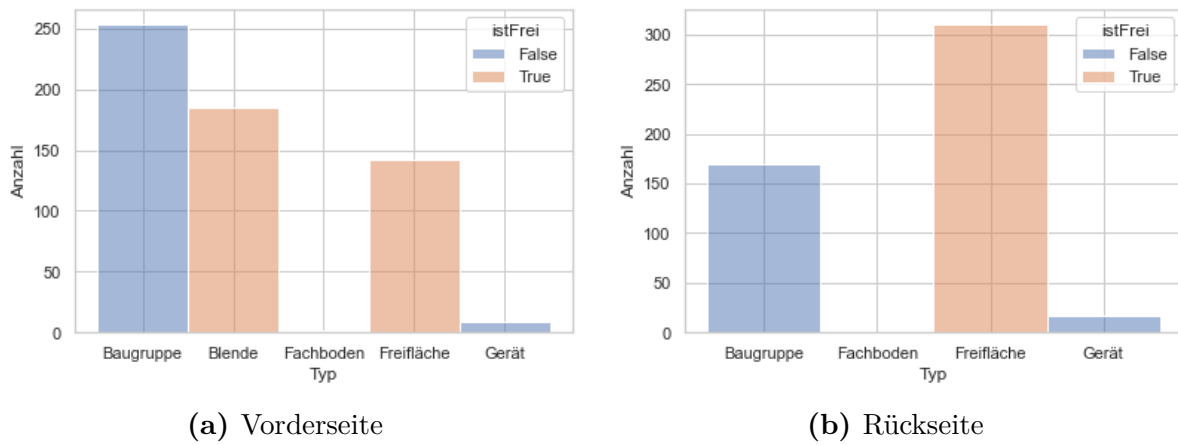


Abbildung 6.2: Häufigkeitsverteilung der Steckplatztypen im Evaluierungsdatensatz

6.2 Evaluierungsstrategie

In diesem Abschnitt wird verdeutlicht, welches [Vorgehen](#) verwendet wurde, um die Ergebnisse der Steckplatzanalyse auf einen Evaluierungsdatensatz einheitlich bewertbar zu machen. Außerdem werden in [Unterabschnitt 6.2.2](#) Metriken definiert, durch welche die Genauigkeit der Steckplatzanalyse in unterschiedlichen Gesichtspunkten bemessen werden kann.

Ziel der Evaluierung ist es, zu jedem Rack aus den Testszenarien, Informationen über die Korrektheit der durch die Steckplatzanalyse gewonnenen Steckplatzauslastung zu gewinnen.

Entsprechend der Zielstellung beinhaltet jene Steckplatzauslastung Informationen über die Anzahl, Größe und Position der im Serverschrank montierten Komponenten und freien Steckplatzpositionen.

Demnach muss für jeden Steckplatz überprüft werden, ob das jeweilige Ergebnis der Steckplatzanalyse mit dem tatsächlichen Zustand des Steckplatzes übereinstimmt. Dies umfasst zum einen das Kontrollieren der binären Klassifikation der aktuellen Auslastung, als frei oder belegt und zum anderen den Vergleich der Position und Größe der Vorhersage, mit der tatsächlichen Position und Größe der im Rack montierten Komponente oder Freifläche.

6.2.1 Vorgehen

Die Herausforderung dieser Evaluierung bestand darin, den Ergebnissen der Steckplatzanalyse die korrekten tatsächlichen Steckplatzzustände aus dem Evaluierungsdatensatz zuzuordnen.

Das in [Tabelle 5.6](#) dargestellte Ergebnis beinhaltet zwar die Steckplatzposition der einzelnen Vorhersagen, jedoch basiert diese auf dem *Postprocessing* und ist somit Teil der Korrektheitskontrolle und kann nicht als eindeutiger Indikator für die Zuordnung dienen.

Folglich wurden die Informationen der Steckplatzanalyse für die Evaluierung angepasst. Zuzüglich der bisherigen Informationen wurden die Koordinaten der Begrenzungsrahmen der jeweiligen Objekte, sowie der Dateiname des zugrundeliegenden analysierten Fotos in das auszugebende Ergebnis der Steckplatzanalyse aufgenommen.

Dadurch ließ sich ein Identifikator bestimmen, wodurch den Vorhersagen der jeweilige tatsächliche Zustand zugeordnet werden konnte.

Dieser Identifikator bezieht sich auf den [IoU](#)-Wert.

Genauer wurde bei der Evaluierung jedem annotierten Steckplatz das Objekt der Steckplatzanalyse zugeordnet, welches den größten [IoU](#)-Wert zu diesem Steckplatz aufwies. Da die Steckplatzanalyse nahezu jeden Bereich des Bildes als entweder freien oder belegten Steckplatz klassifiziert, kann davon ausgegangen werden, dass zu jedem annotierten Steckplatz ein vorhergesagtes Objekt existiert, welches einen positiven [IoU](#)-Wert produziert⁴. Dabei musste beachtet werden, dass sich die Steckplatzanalyse in manchen Fällen aus mehreren Bildern zusammensetzt, sodass die Koordinaten der Begrenzungsrahmen für die [IoU](#) Berechnung angepasst wurden. Die y-Koordinaten der Begrenzungsrahmen wurden dafür mit der Höhe der vorherigen Bilder aufsummiert und somit eine eindeutige [IoU](#) Identifizierung ermöglicht.

Dadurch ließ sich zu jedem Steckplatz und Objekt⁵ des Evaluierungsdatensatzes ein Objekt aus der Vorhersage zuordnen und auf Korrektheit der folgenden Eigenschaften überprüfen:

- Binäre Klassifikation (freier oder belegter Steckplatz)
- Größe in [RU](#)
- Steckplatzposition

Ferner wurde die vorhergesagte Rack-Höhe mit der tatsächlichen Höhe sowie die Anzahl der erkannten Objekte mit der tatsächlichen Objektanzahl verglichen.

6.2.2 Evaluierungsmetriken

Um eine Bewertung und Interpretation der Resultate der Steckplatzanalyse zu ermöglichen, wurden die folgenden Metriken definiert:

Durchschnittliche Genauigkeit der binären Klassifikation

Die Ergebnisse der binären Klassifikation werden mittels der durchschnittlichen *Genauigkeit* evaluiert.

Dafür wird bei allen Vergleichen erhoben, ob es sich um eine [TP](#), [TN](#), [FP](#) oder [FN](#) Klassifikation handelt. Richtig erkannte freie Steckplätze wurden als [TP](#) und falsche als [FP](#) bewertet.

⁴Lediglich Bereiche, welche kleiner als der Freiflächen-Schwellenwert sind, wurden nicht klassifiziert. Bei der Durchführung der Evaluierung trat ein solches Ereignis nur bei einem einzigen Steckplatz im Datensatz auf, welcher als falsch klassifizierter Steckplatz in die Bewertung einging.

⁵Die Begrenzungsrahmen der Steckplätze, welche dieselbe Objektnummer aufwiesen, ließen sich vereinigen und dienten somit als Identifikator zur Objektzuordnung.

Richtig identifizierte belegte Steckplätze wurden hingegen als **TN** und falsche als **FN** definiert. Für jede Steckplatzanalyse eines Racks lässt sich dadurch die Genauigkeit (engl. *Accuracy*) mittels [Gleichung 6.1](#) [[Sze22](#), S. 354] definieren.

$$acc = \frac{TP + TN}{TP + FP + TN + FN} \quad (6.1)$$

Die durchschnittliche Genauigkeit $\overline{acc}$ berechnet sich demnach aus dem arithmetischen Mittel aller Genauigkeitswerte der n Steckplatzanalysen für die im Evaluierungsdatensatz enthaltenen Racks, siehe [Gleichung 6.2](#). Jene Metrik gibt an, wie viel Prozent der Steckplätze jedes Racks durchschnittlich korrekt als frei oder belegt klassifiziert wurden⁶.

$$\overline{acc} = \frac{1}{n} \sum_{i=1}^n acc_i \quad (6.2)$$

Median der mittleren absoluten Abweichung der RU-Berechnung der Objekte

Im Gegensatz zur binären Klassifikation stellt die Erkennung der **RU**-Größe eine numerische Ausgabe dar, wodurch sich Metriken definieren lassen, welche anstatt der reinen Korrektheit des Ergebnisses, dessen Abweichung zum tatsächlichen Ergebnis bemessen.

Demnach wurde zu jedem Objekt der einzelnen Racks des Evaluierungsdatensatzes, das Objekt mit dem größten **IoU**-Wert aus der vorhergesagten Steckplatzanalyse entnommen und verglichen. Dadurch ließ sich für jedes Rack die mittlere absolute Abweichung (engl. *mean absolute error*) der **RU**-Vorhersagen mittels [Gleichung 6.3](#) [[CD14](#)] bestimmen. Dabei beschreibt y die tatsächliche und $\hat{y}$ die vorhergesagte **RU**-Höhe eines Objekts. m bezieht sich wiederum auf die Anzahl der angestellten Vergleiche.

$$mae = \frac{1}{m} \sum_{i=1}^m |\hat{y} - y|_i \quad (6.3)$$

Jene mae wurde für jedes der n im Evaluierungsdatensatz enthaltenen Racks bestimmt und dessen Median durch

$$\widetilde{mae} = \begin{cases} mae_{\frac{m+1}{2}} & m \text{ ungerade} \\ \frac{1}{2} \left(mae_{\frac{m}{2}} + mae_{\frac{m}{2}+1} \right) & m \text{ gerade} \end{cases} \quad (6.4)$$

berechnet. $\widetilde{mae}_{RU}$ gibt demnach den Zentralwert der mittleren absoluten Abweichung für die **RU**-Höhen-Bestimmung der Objekte eines Racks an.

Es wurde sich bewusst gegen die Verwendung des arithmetischen Mittels entschieden, da die Abweichung der **RU** in manchen Fällen eine große Varianz aufweisen kann. Würde beispielsweise im **hhi_Rack3** (siehe [Abbildung 6.1](#)), welches lediglich aus einer Freifläche der Größe **47RU** besteht, fälschlicherweise eine 1 **RU** Baugruppe im Zentrum des Racks

⁶Gleiches könnte auch durch das Kumulieren aller binären Klassifikationsergebnisse und der [Gleichung 6.1](#) erreicht werden, jedoch wurden aufgrund der besseren Nachvollziehbarkeit die einzelnen Genauigkeiten und dessen Durchschnitt im Anschluss bestimmt.

vorhergesagt, so bestünde die Steckplatzanalyse aus den drei Objekten: 21RU Freifläche, 1 RU Baugruppe und 21RU Freifläche. Gemäß der Evaluierungsstrategie, würde das Objekt mit dem größten IoU-Wert für den Vergleich verwendet werden, demnach eine der Freiflächen. Dies würde wiederum in einer mittleren absoluten Abweichung von 26 resultieren. Jener Wert würde die durchschnittliche Leistung des Verfahrens verzerren, da eine solch hohe Abweichung lediglich aus einem falschen Objekt und der daraus folgenden fehlerhaften RU-Berechnung resultiert.

Median der mittleren absoluten Abweichung der Steckplatzpositionen

Äquivalent zum Median der mittleren absoluten Abweichung der RU, wurde die mae und $\widehat{mae}$ aus Gleichung 6.3 und Gleichung 6.4 für die Abweichung der Steckplatzpositionen als mae_{POS} verwendet.

Hierbei beschreibt $\widehat{y}$ in Gleichung 6.3 die vorhergesagte Startposition und y die tatsächliche Startposition eines im Rack enthaltenen Objekts.

Die Endposition des Objekts wurde dabei nicht betrachtet, da diese von der RU-Höhe abhängt, dessen Abweichung bereits durch die $\widehat{mae}_{RU}$ berechnet wird.

Somit gibt die $\widehat{mae}_{POS}$ den Median der mittleren absoluten Abweichung der Startposition der im Rack enthaltenen Objekte an.

Die Wahl des Medians gegenüber dem arithmetischen Mittel resultiert ebenfalls aus der möglichen großen Varianz der Abweichung. Wird etwa dem obersten Steckplatz eines Racks eine falsche RU-Höhe prognostiziert, so addiert sich jener Fehler auf alle weiteren Steckplatzpositionen des Racks, da diese von jener RU Prognose abhängen.

Mittlere absolute Abweichung der Rack Höhe

Die Rack Höhe stellt ebenfalls eine numerische Ausgabe der Steckplatzanalyse dar, wodurch sich auch hier die mittlere absolute Abweichung bestimmen lässt.

Hierbei wurde die Abweichung der vorhergesagten Rack Höhe der m Racks des Evaluierungsdatensatzes $\widehat{y}$ zur tatsächlichen Rack Höhe y mittels Gleichung 6.3 bestimmt.

mae_{RACK} definiert folglich die mittlere absolute Abweichung der vorhergesagten Rack Höhen.

Mittlere absolute Abweichung der Komponentenanzahl

Analog zur Rack Höhe lässt sich ebenfalls die mittlere absolute Abweichung der Anzahl der vorhergesagten Komponenten mae_{ANZ} mittels Gleichung 6.3 bestimmen. Hierbei beschreibt $\widehat{y}$ die vorhergesagte und y die tatsächliche Objektanzahl.

6.2.3 Gütekriterium eines Evaluierungsergebnisses

Um die Ergebnisse der Evaluierung besser miteinander vergleichen zu können, wurde das folgende Gütekriterium *error*, resultierend aus den vorangegangenen Metriken, definiert.

$$error = (1 - \overline{acc}) \times (\overline{mae}_{ru} + \overline{mae}_{POS} + mae_{RACK} + mae_{ANZ}) \quad (6.5)$$

Dieses Gütekriterium multipliziert den Fehler der binären Klassifikation mit der Summe der Abweichungsmetriken. Dadurch soll den durchschnittlich besten Ergebnissen der geringste *error* zugewiesen werden.

Jenes Gütekriterium dient lediglich dem einfacheren Vergleich der Ergebnisse. Eine genaue Bewertung eines Ergebnisses lässt sich nur unter der einzelnen Betrachtung aller Metriken erzielen.

6.2.4 Umsetzung der Evaluierung

Die jeweiligen Vergleiche und die Berechnung der Metriken ließ sich automatisiert ausführen.

Dafür wurden die nötigen Evaluierungsdaten in einem *Jupyter Notebook* eingelesen.

Diese enthielten neben den annotierten Steckplätzen, ebenfalls die vier Koordinaten des Vierecks zur Begrenzung des „Einbauchassis“. Des Weiteren waren die Speicherpfade der zugrundeliegenden Bilder in der jeweiligen Annotationsdatei vorhanden.

Dadurch ließ sich für jedes annotierte Rack eine Anfrage aus einem oder mehreren Bildern an den *Server* senden und dessen Steckplatzanalyseergebnisse direkt im *Jupyter Notebook* verarbeiten.

Dabei wurden die Ergebnisse der jeweiligen Racks in einzelnen **csv**-Dateien abgespeichert und anhand der vorgestellten Metriken bemessen.

Abschließend wurde eine zusammenfassende **csv**-Datei erstellt, welche die durchschnittlichen Ergebnisse der *acc*, *mae_{RU}*, *mae_{POS}* errechnet und mittlere absolute Abweichungen der Rack Höhe *mae_{RACK}* und Komponentenanzahl *mae_{ANZ}* des gesamten Datensatzes bestimmt.

Der implementierte *Server-Code* wurde auf einer virtuellen Maschine im Netzwerk von *highstreet technologies* installiert und ausgeführt.

Jene Umsetzung validiert jedoch nicht die Funktionalität der App, weswegen die abgespeicherten Ergebnisse stichprobenartig auf Übereinstimmung zu den Ergebnissen aus der App überprüft worden sind. Dafür wurden die aufgenommenen Fotos in der *Micro-TSS-App* für die Analyse ausgewählt und die Begrenzungen des „Einbauchassis“ bestimmt.

Dabei wurde das identische Smartphone⁷ verwendet, mit welchem die Fotos für die Evaluierung aufgenommen wurden. Es ließ sich feststellen, dass die Evaluierungsergebnisse mit den Ergebnissen durch die App übereinstimmen, weswegen eine Bewertung der Evaluierungsergebnisse auch eine Bewertung der Genauigkeit und Korrektheit der Steckplatzanalyse in der App beschreibt. Der Vollständigkeit halber befindet sich im [Abschnitt A](#) des Anhangs eine genauere Beschreibung dieser Überprüfung.

⁷ Samsung Galaxy S21+ 5G.

6.2.5 Parameter der Evaluierung

Die Ergebnisse der Steckplatzanalyse sind abhängig von vorher festgelegten Parametern. Dazu gehört der *Konfidenz-Score*-Schwellenwert, welcher unsichere Vorhersagen herausfiltern soll, der Freiflächen-Schwellenwert sowie eine Konstante, welche die Breite des „Einbauchassis“ definiert und somit die [Berechnung, der PPMR](#) ermöglicht.

Zunächst wurde die Evaluierung mit den folgenden Parametern ausgeführt:

- *Konfidenz-Score*-Schwellenwert:
 - *Mask R-CNN*: 0.9
 - *YOLOv5*: 0.5
- Freiflächen-Schwellenwert: 0.5
- [PPMR](#)-Konstante: 482.6mm

Jene Parameter wurden bei der Implementierung gewählt und sollten ebenfalls bei der Evaluierung eingesetzt werden.

Im Anschluss wurde überprüft, welchen Einfluss eine Änderung der Parameter auf die Ergebnisse hat. Dabei erhielten die *Konfidenz-Score*-Schwellenwerte in verschiedenen Durchläufen die Werte: 0.5, 0.6, 0.9, 0.95 und 0.97.

Des Weiteren wurden 465mm als alternative [PPMR](#)-Konstante gewählt, da diese nach [\[DIN\]](#) den Abstand des Montierungsbereichs des „Einbauchassis“ angibt (siehe [Abbildung 4.1](#)).

Abschließend wurde anhand des besten Ergebnisses⁸ der Einfluss des Freiflächen-Schwellenwerts überprüft. Dieser erhielt die alternativen Werte: 0.25 und 0.75.

6.3 Evaluierungsergebnisse

Im Folgenden werden die Ergebnisse der Evaluierung beschrieben und hinsichtlich Parameterveränderungen diskutiert.

6.3.1 Ergebnisse basierend auf den ursprünglichen Parametern

In [Tabelle 6.1](#) sind die Ergebnisse der Evaluierung mit den ursprünglichen Parametern dargestellt.

Darin wird zunächst deutlich, dass das Verfahren, gemessen am Gütekriterium, bessere Ergebnisse bei den Racks aus dem Rechenzentrum von *highstreet technologies*⁹ als bei den Racks aus dem Rechenzentrum des *Fraunhofer-Instituts für Nachrichtentechnik, Heinrich-Hertz-Institut*¹⁰ erzielt. Dabei weisen vorwiegend die prognostizierten Steckplatzpositionen ($\overline{mae}_{pos}$) der Rückseite der Racks aus dem *hhi* größere Fehlerwerte auf.

⁸Gemessen am Gütekriterium aus [Gleichung 6.5](#).

⁹Im weiteren Verlauf als *highstreet* bezeichnet.

¹⁰Für eine bessere Lesbarkeit wird das *Fraunhofer-Instituts für Nachrichtentechnik, Heinrich-Hertz-Institut* im folgenden durch *hhi* abgekürzt.

Daten	Seite	Model	$\overline{acc}$	$\overline{mae}_{RU}$	$\overline{mae}_{POS}$	mae_{RACK}	mae_{ANZ}	$error$
hhi	Back	Mask R-CNN@0.9	0.791297	1.869565	20.104814	2.333333	2.833333	5.664410
hhi	Front	Mask R-CNN@0.9	0.786945	2.307692	1.598485	2.500000	7.000000	2.856254
hhi	Back	YOLOv5@0.5	0.893913	0.514752	20.811335	1.833333	1.000000	2.563012
hhi	Front	YOLOv5@0.5	0.941489	0.594697	1.194444	3.875000	3.625000	0.543514
highstreet	Back	Mask R-CNN@0.9	0.831672	3.500000	1.280000	3.200000	4.200000	2.050233
highstreet	Front	Mask R-CNN@0.9	0.920930	0.129032	0.529412	2.800000	2.400000	0.463226
highstreet	Back	YOLOv5@0.5	0.915282	1.000000	0.166667	4.400000	0.600000	0.522425
highstreet	Front	YOLOv5@0.5	0.920930	0.052632	1.000000	2.800000	2.200000	0.478580

Tabelle 6.1: Darstellung der Evaluierungsergebnisse basierend auf den ursprünglichen Parametern. Unterschieden wird dabei nach dem gewählten Datensatz sowie ob die Fotos darin die Vorder- oder Rückseite des Racks abbilden.

Da die weiteren $\overline{mae}_{POS}$ -Werte jedoch signifikant geringer als die bei der Rückseite des *hhi*-Datensatzes sind, wird vermutet, dass die Berechnung der Steckplatzposition im Wesentlichen der richtigen Logik folgt, jedoch in jenem Teil-Datensatz häufiger ein Fehler innerhalb der Analyse der oberen Steckplatzpositionen auftrat.

Bei einer Nichtbetrachtung jenes „Ausreißer“-Werts zeigt sich, dass die Vorhersagen auf den Rückseiten der *hhi*-Racks geringere Abweichung produzieren als auf dessen Vorderseite. Der höhere $error$ resultiert lediglich aus $\overline{mae}_{POS}$ und der ca. 5% schlechteren binären Klassifizierungsgenauigkeit bei der Verwendung von *YOLOv5*.

Ein Grund, warum die weiteren Abweichungen auf der Rückseite der *hhi*-Racks geringer ausfielen, lässt sich anhand der Analyse des Evaluierungsdatsatzes nicht feststellen. Bei der Analyse des Trainingsdatensatzes wurde aufgrund der Häufigkeitsverteilung der Komponenten für das Training die Vermutung aufgestellt, dass die Objekterkennung am besten für 1 RU hohe Objekte funktionieren wird. Da die Vorderseiten der *hhi*-Racks jedoch mehr 1 RU Objekte als die Rückseiten enthält, lässt sich diese Vermutung weder bestätigen noch für die Begründung der besseren Ergebnisse auf den *hhi*-Rückseiten verwenden.

Demnach wird angenommen, dass die *hhi*-Rückseiten stärker den *highstreet*-Rückseiten ähneln, als die *hhi*-Vorderseiten den *highstreet*-Vorderseiten. Da viele der Komponenten der *highstreet*-Racks im Trainingsdatensatz vorhanden waren, würde dies die Präzision der vorhergesagten Begrenzungsrahmen für ähnlich aussehende Komponenten verbessern. Jene Verbesserung der Präzision würde sich ebenfalls in der Verbesserung der Abweichungsmetriken $\overline{mae}_{RU}$ und mae_{RACK} widerspiegeln, da diese auf den Begrenzungsrahmen basieren. Die schlechtere $\overline{acc}$ mit *YOLOv5* (Rückseite: 0.89, Vorderseite: 0.94) ließe sich wiederum damit begründen, dass der Objektdetektor mehr Komponenten fälschlicherweise als Freifläche identifiziert hat, da diese von Kabeln überdeckt waren, welche auf der Vorderseite weniger häufig vorkommen. Dies kann auch zutreffen, trotzdem die mae_{ANZ} auf den Rückseiten (2.833333, 1.0) geringer ist, da die hohen mae_{ANZ} der Vorderseiten (7.0, 3.625) allein daraus resultieren können, dass mehrere Komponenten, aufgrund der schlechteren Präzision der Begrenzungsrahmen, fälschlicherweise als eine einzelne Komponente oder umgekehrt umrahmt wurden. Dies würde zu einer Verschlechterung der mae_{ANZ} führen, jedoch nicht zwangsweise zu einer Verschlechterung der $\overline{acc}$.

Für den *highstreet*-Datensatz zeigt sich wiederum, dass zumeist die Vorderseiten genauer analysiert wurden. Dies festigt die für die *hhi*-Racks aufgestellte Vermutung, da die Objektdetektoren sowohl für ähnlich aussehende Vorderseiten, als auch für ähnlich aussehende

Rückseiten trainiert wurden und somit keinen Präzisionsvorteil aufgrund der Generalisierungsfähigkeit erhielten. Die Genauigkeitsunterschiede ließen sich demnach auch in diesem Datensatz durch das Kabelvorkommen auf den Rückseiten begründen.

Entgegen der Erwartung, dass zweistufige Objektdetektoren genauer als einstufige Detektoren sind, weist *YOLOv5* als Objektdetektor zumeist geringere Fehlerwerte als der *Mask R-CNN* Detektor auf. Dies kann zum einen aus der generellen Performance des Modells resultieren oder aber an suboptimal gewählten *Konfidenz-Score*-Schwellenwerten. Zweiteres würde die generell höheren Abweichungen der mae_{ANZ} für die auf *Mask R-CNN* basierende Steckplatzanalyse erklären. Ein zu hoher *Konfidenz-Score*-Schwellenwert kann dazu führen, dass mehrere Objekte in der Steckplatzanalyse nicht erkannt und als einzelne Freifläche mit in das Analyseergebnis aufgenommen werden. Dadurch könnten die hohen Abweichungen von 7.0 für die Vorderseite der Rack-Fotos aus dem *hhi* oder 4.2 für die fotografierten Rückseiten der Racks aus dem Rechenzentrum von *highstreet* entstehen. Dies lässt sich durch eine Evaluierung auf alternative *Konfidenz-Score*-Schwellenwerte überprüfen.

6.3.2 Einfluss der Konfidenz-Score-Schwellenwerte

Für die Evaluierung wurden die in [Unterabschnitt 6.2.5](#) beschriebenen alternativen *Konfidenz-Score*-Schwellenwerte gewählt.

Dabei zeigte sich, dass sich die Performance der auf *YOLOv5* basierenden Steckplatzanalyse mit einem *Konfidenz-Score*-Schwellenwert von 0.6 minimal für die *hhi*-Rückseiten verbessern ließ.

Die Performance der auf *Mask R-CNN* basierenden Steckplatzanalyse konnte hingegen durch die Reduzierung auf den *Konfidenz-Score*-Schwellenwert von 0.5 signifikant verbessert werden¹¹. Jene verbesserten Ergebnisse sind in [Tabelle 6.2](#) dargestellt.

Vor allem die Ergebnisse auf die *hhi*-Racks ließen sich dadurch verbessern, wohingegen die Ergebnisse auf die bekannten Rückseiten der *highstreet*-Racks minimal verschlechtert wurden. Das spricht für die Vermutung, dass der vorherige Schwellenwert zu fälschlich identifizierten Freiflächen geführt hat. Gleichzeitig lässt dies erneut auf eine mangelnde Generalisierungsfähigkeit schließen, da die unbekannten Komponenten aus dem *hhi* nur mit geringen *Konfidenz-Score*-Schwellenwerten erkannt werden konnten. Die für den Objektdetektor bekannten Komponenten von *highstreet* wurden dadurch vermutlich von Objekten mit geringen *Konfidenz-Scores* überlagert, was aufgrund des *Postprocessings* zu den ungenaueren Vorhersagen führt.

Die Parameteranpassung führte jedoch nicht dazu, dass *Mask R-CNN* bessere Ergebnisse als *YOLOv5* erzielte. Bei einem Vergleich der *mAP* auf den Validierungsdatensatz im Verlauf des Trainings der Objektdetektoren in [Tabelle 5.5](#) und [Tabelle 5.4](#), deutete sich bereits an, dass *YOLOv5* auf diesen Datensatz bessere Ergebnisse erzielen kann.

Dies könnte zum einen an der generellen Modell-Architektur und den unterschiedlichen Trainingskripts liegen, zum anderen basieren die vortrainierten Modell-Parameter jeweils auf unterschiedlichen Datensätzen. Die *YOLOv5*-Parameter stammen aus dem Training auf

¹¹Gemessen anhand der geringsten Summe des Gütekriteriums für alle Ergebnisse mit dem gleichen *Konfidenz-Score*-Schwellenwert und Modell.

den Datensatz der laut Zou et al. [ZSGY19] herausforderndsten Objekterkennungschallenge **COCO**. Es ist möglich, dass dadurch eine bessere Ausgangssituation für das *Transfer Learning* auf den Rack-Datensatz geschaffen wurde.

Daten	Seite	Model	$\overline{acc}$	$\overline{mae}_{RU}$	$\overline{mae}_{POS}$	mae_{RACK}	mae_{ANZ}	$error$
hhi	Back	Mask R-CNN@0.5	0.808806	1.326087	20.409161	2.166667	2.333333	5.016018
hhi	Front	Mask R-CNN@0.5	0.837477	0.502444	1.320707	2.125000	4.375000	1.352705
hhi	Back	YOLOv5@0.6	0.893913	0.601708	20.702640	1.833333	1.000000	2.560706
hhi	Front	YOLOv5@0.6	0.941489	0.594697	1.194444	3.875000	3.625000	0.543514
highstreet	Back	Mask R-CNN@0.5	0.799114	4.333333	1.280000	3.200000	6.000000	2.975790
highstreet	Front	Mask R-CNN@0.5	0.911628	0.129032	0.529412	2.400000	2.400000	0.482374
highstreet	Back	YOLOv5@0.6	0.915282	1.000000	0.166667	4.400000	0.600000	0.522425
highstreet	Front	YOLOv5@0.6	0.920930	0.052632	1.000000	2.800000	2.200000	0.478580

Tabelle 6.2: Darstellung der Evaluierungsergebnisse basierend auf einem *Konfidenz-Score*-Schwellenwert von 0.5 für *Mask R-CNN* und 0.6 für *YOLOv5*.

6.3.3 Einfluss der PPMR-Konstante

Im weiteren Verlauf der Evaluierung wurden $465mm$ als alternative Konstante für die **PPMR** Berechnung geprüft.

Da sich die annotierten Begrenzungen des „Einbauchassis“ im Evaluierungsdatensatz jedoch am äußersten Rand jener „Chassis“, demnach an einer Breite von 19-Zoll (482.6 mm) orientierten, wurde erwartet, dass jene Parameteränderung zu einer Verschlechterung der Ergebnisse führen würde.

Im Anhang in [Tabelle A.2](#) sind der Vollständigkeit halber die Ergebnisse unter Einbeziehung jener Parameterveränderungen dargestellt. Verglichen mit [Tabelle 6.2](#) ließ sich die Erwartung bestätigen, dass die Veränderung der **PPMR**-Konstante zu einer Verschlechterung der Analysegenauigkeit führt. Dadurch lässt sich eine erneute Bestätigung finden, dass die Berechnungen basierend auf das **PPMR** der richtigen Logik folgen, da andernfalls bessere Analyseergebnisse produziert worden wären.

6.3.4 Einfluss des Freiflächen-Schwellenwerts

Die bisherigen Ergebnisse basieren auf einem Freiflächen-Schwellenwert von 0.5.

Jener Schwellenwert gibt an, wie groß die Freifläche oberhalb einer Komponente sein muss, um in der Steckplatzanalyse als Freifläche klassifiziert zu werden.

Dieser Schwellenwert ist hauptsächlich bei einer ungenauen Identifizierung der Begrenzungen eines Steckplatzes von Bedeutung. Wurde der prognostizierte Begrenzungsrahmen beispielsweise oberhalb der tatsächlichen Objektkante angesetzt, so würde ein zu hoher Schwellenwert dazu führen, dass eine eventuelle oberhalb liegende Freifläche nicht mit in die erkannten Objekte der Steckplatzanalyse aufgenommen wird. Ferner kann ein zu niedrig gewählter Schwellenwert dazu führen, dass Freiflächen fälschlicherweise mit in die erkannten Objekte aufgenommen werden, sobald die Begrenzungsrahmen unterhalb der tatsächlichen Kante angesetzt werden.

Die jeweiligen Ergebnisse sind ebenfalls im Anhang verfügbar (siehe [Tabelle A.3](#)). Darin zeigte sich, dass der Freiflächen-Schwellenwert von 0.25 keine Veränderung an den bisherigen Ergebnissen bewirkte. Der Freiflächen-Schwellenwert von 0.75 resultierte jedoch in schlechteren Genauigkeitsergebnissen.

6.4 Auswertung der Ergebnisse und mögliche Fehlerquellen

Durch die Evaluierung wurde ersichtlich, dass die entwickelte Steckplatzanalyse auch bei einer optimalen Bestimmung der Parameter¹² und der Wahl des besten *Objektdetektors* (*YOLOv5*) nicht die geforderte Genauigkeit der Zielstellung erzielt.

Die Analyse klassifiziert im besten Fall¹³, in 5.9% der Fälle einen falschen Auslastungszustand und scheitert damit deutlich an den geforderten 1%. Des Weiteren wurde in keinem der Datensätze ansatzweise eine geringere Abweichung als 0.05 Einheiten erzielt.

Neben den bisher angeführten Vermutungen, welche sich auf die Generalisierungsfähigkeit der Objektdetektoren und der Kabelvorkommen als Hindernisse bezieht, lassen sich zwei weitere Fehlerquellen identifizieren.

Eine Fehlerquelle, die zu höheren Abweichungen führt, könnte in ungenauen *PPMR*-Berechnungen liegen. Während der Evaluierung wurde festgestellt, dass die implementierte perspektivische Transformation zu leichten Veränderungen des Seitenverhältnisses der Rack-Fotos führt. Jenes transformierte Seitenverhältnis richtet sich nach der jeweiligen breitesten und längsten Gerade des markierten Vierecks. Dies bildet, je nach Grad der Kameraneigung, nicht immer das korrekte Seitenverhältnis des „Einbauchassis“ ab, was wiederum zu einem falschen *PPRUR* führt. Dies spiegelt sich in den Abweichungen $\widetilde{mae}_{ru}$ und mae_{RACK} wider. Da in den *PPMR*-Berechnungen jedoch immer auf die nächstliegende ganze Zahl gerundet wird, zeigt sich dieser Fehler besonders bei großen Objekten.

Aus [Abbildung 6.1](#) geht hervor, dass die *highstreet*-Rückseiten deutlich größere Objekte als deren Vorderseite enthalten, was wiederum an der unterschiedlichen Behandlung von *Blenden* und Freiflächen liegt. Das hat zur Folge, dass die *highstreet*-Rückseiten in [Tabelle 6.2](#) größere Abweichungen für $\widetilde{mae}_{ru}$ und mae_{RACK} produzieren, als die *highstreet*-Vorderseiten. Im Jahr 2007 publizierten Zhang und He [[ZL07](#)] einen Ansatz, der die Veränderungen der Seitenverhältnisse bei einer perspektivischen Transformation minimieren soll. Jener Ansatz wurde im Zuge der Evaluierung für diese Arbeit übernommen, um zu überprüfen, ob sich dadurch eine Verringerung der Abweichung erreichen ließ. In [Tabelle 6.3](#) sind die daraus resultierenden Ergebnisse dargestellt. Diese Anpassung brachten jedoch keine eindeutige Verbesserung der Genauigkeit mit sich.

Die Analysen der Rückseiten der *highstreet*-Racks enthielten zwar geringere Abweichungen für $\widetilde{mae}_{ru}$ und mae_{RACK} , jedoch verschlechterte sich dadurch die binäre Klassifizierungsgenauigkeit und demnach auch die Korrektheit der Analyse.

Der von Zhang und He publizierte Ansatz produzierte in bestimmten Szenarien sogar deutlich verzerrtere Seitenverhältnisse als der vorherige Ansatz. Ein Beispiel dafür wurde im Anhang unter [Abbildung A.3](#) visualisiert. Durch solche Szenarien wurde die Abweichung der Steckplatzanalyse in einigen Fällen verschlechtert (siehe $\widetilde{mae}_{ru}$ und mae_{RACK}

¹²Konfidenz-Score-Schwellenwert von 0.6, Freiflächen-Schwellenwert von 0.5 und *PPMR* Konstante von 482.6mm.

¹³Vorderansicht der *hhi*-Racks

Daten	Seite	Model	$\overline{acc}$	$\overline{mae}_{RU}$	$\overline{mae}_{POS}$	mae_{RACK}	mae_{ANZ}	$error$
hhi	Back	Mask R-CNN@0.5	0.752290	1.593944	19.800466	1.833333	2.000000	6.249160
hhi	Front	Mask R-CNN@0.5	0.858753	0.806452	1.613636	4.000000	4.125000	1.489457
hhi	Back	YOLOv5@0.6	0.900931	0.395963	21.212733	0.500000	1.166667	2.305870
hhi	Front	YOLOv5@0.6	0.933511	0.644265	1.297980	5.875000	3.125000	0.727543
highstreet	Back	Mask R-CNN@0.5	0.799114	3.833333	1.600000	2.800000	5.600000	2.778922
highstreet	Front	Mask R-CNN@0.5	0.865116	0.129032	0.705882	1.200000	2.600000	0.625175
highstreet	Back	YOLOv5@0.6	0.939203	0.500000	0.166667	2.000000	1.000000	0.222924
highstreet	Front	YOLOv5@0.6	0.916279	0.078947	0.764706	1.000000	2.400000	0.355283

Tabelle 6.3: Darstellung der Evaluierungsergebnisse basierend auf 482.6mm als [PPMR](#)-Konstante, 0.5 als Freiflächen-Schwellenwert, 0.5 als *Konfidenz-Score*-Schwellenwert für *Mask R-CNN*, 0.6 als *Konfidenz-Score*-Schwellenwert für *YOLOv5* und dem von Zhang und He entwickelten perspektivischen Transformationsverfahrens.

der *hhi*-Vorderseiten).

Da sich der Grund für solch fehlerhafte Verzerrung nicht identifizieren ließ, bleibt der bisherige perspektivische Transformations-Ansatz für die weitere prototypische Verwendung der App bestehen. Es ist jedoch zu empfehlen, die Fotos für die Steckplatzanalyse in dem geringst möglichen Neigungswinkel aufzunehmen, um den Fehler des Seitenverhältnisses zu minimieren.

Jener Fehler könnte verhindert werden, wenn die [RU](#)-Höhe des „Einbauchassis“ vor der Analyse bekannt wäre. Dadurch ließe sich das *PPRUR* mittels $\frac{H}{RU}$ bestimmen, wobei *RU* die [RU](#)-Höhe und *H* die Pixel-Höhe des durch die perspektivische Transformation zugeschnittenen Bildes beschreibt. Jedoch würde dies voraussetzen, dass die Benutzenden das Rack eigenständig inspizieren, wodurch der Automatismus der Steckplatzanalyse verloren ginge.

Jener Automatismus wurde ohnehin schon, durch die verpflichtende Markierung des „Einbauchassis“, eingeschränkt und sollte für eine gute Nutzbarkeit der App nicht weiter verschlechtert werden.

Eine weitere Fehlerquelle zeigte sich während des Annotierens des Evaluierungsdatensatzes. Dabei wurde festgestellt, dass sich nicht alle im Rack verbauten Komponenten, in einer einheitlichen Tiefe befinden (Das genaue Verhältnis wurde im Anhang unter [Abbildung A.4](#) visualisiert). Das führt dazu, dass Komponenten, die tiefer im Rack montiert sind, von darüberliegenden, zur Kamera näher montierten Komponenten, teilweise oder gar völlig überdeckt werden, sobald die Fotos mit einem suboptimalen Neigungswinkel aufgenommen werden.

In solch einem Fall lassen sich die Begrenzungen der Komponenten nicht allein mithilfe des Bildes ermitteln, was eine korrekte bildbasierte Steckplatzanalyse unmöglich macht. Dies resultiert wiederum in schlechteren Evaluierungsergebnissen für alle definierten Metriken. Demnach müsste die Voraussetzung für die Anwendung der Steckplatzanalyse-App lauten, dass die zu analysierenden Racks ausschließlich Komponenten in einer einheitlich montierten Tiefe beinhalten. Andernfalls müsste bei der Aufnahme der einzelnen Fotos eines Racks genau darauf geachtet werden, dass jede Komponente vollständig sichtbar ist¹⁴.

¹⁴Der Begrenzungsrahmen einer entfernten Komponente wäre in solchem Szenario kleiner als das *PPRUR*. Bei einem optimal bestimmten *PPRUR*, sollte dies aufgrund der ganzzahligen Rundungen im *Postpro-*

Zweiteres würde jedoch zu einem erheblichen Mehraufwand bei der Dokumentation führen und somit die eigentliche Motivation der automatischen Steckplatzanalyse nichtig machen.

Der entwickelte Prototyp ist demnach aufgrund der hohen Abweichungen und unzureichenden Genauigkeit bei der binären Auslastungsklassifizierung für einen tatsächlichen Einsatz der Auslastungsdokumentationen aktuell ungeeignet. Basierend auf den aufgestellten Vermutungen und den zu verhindernden Fehlerquellen sollte sich die Steckplatzanalyse-App jedoch für weniger komplexen Szenarien, welche die Voraussetzung für eine gute Steckplatzanalyse erfüllen¹⁵, für Demonstrationszwecke verwenden lassen.

Solch eine Demonstration wurde für eine Steckplatzanalyse auf das Foto der exemplarischen Steckplatzanalyse aus [Kapitel 4](#), welches aus dem Validierungsdatensatz stammt, aufgezeichnet und veröffentlicht¹⁶. Diese soll dabei helfen, einen Einblick über die Nutzbarkeit der *Micro-TSS-App* zu bekommen. Gleichzeitig ist darin ersichtlich, dass das Ergebnis der Steckplatzanalyse identisch zur beschriebenen [exemplarischen Analyse](#) ist und demnach korrekte Ergebnisse erzielt, trotzdem die im Trainingsdatensatz unterrepräsentierten *Fachböden* und *Geräte* im Rack vorhanden waren.

Bevor die Steckplatzanalyse-App jedoch grundlegend für Racks, die den Voraussetzungen entsprechen, eingesetzt werden kann, müsste überprüft werden, ob sich die geforderte Zielstellung für diese eingeschränkten Szenarien erreichen lässt. Des Weiteren sollte ein entsprechender Evaluierungsdatensatz genügend *Fachböden* und *Geräte* beinhalten, um eine Bewertung jener Funktionalität zu ermöglichen. Da sich jedoch nicht genügend solcher vorausgesetzten Racks im Evaluierungsdatensatz dieser Arbeit befinden, kann jene Bewertung nur in Zukunft vorgenommen werden.

Sollte sich die Zielstellung in solch einem eingeschränkten Datensatz erreichen lassen und sollten die [MEC-Racks](#) diese Voraussetzungen grundlegend erfüllen, ließe sich das entwickelte Verfahren nach wenigen Anpassungen für die einheitliche und automatische Dokumentation von [MEC-Racks](#) verwenden.

Weitere Anmerkungen

Es sollte erwähnt werden, dass die hier aufgeführte Bewertung der Evaluierungsergebnisse lediglich auf 11 Racks resultiert, was nicht ausreichend ist, um eine generalisierte Einschätzung über die Genauigkeit und Korrektheit des entwickelten Verfahrens geben zu können. Des Weiteren setzt das Verfahren voraus, dass die Server-Komponenten in den dafür vorgesehenen Steckplatzpositionen verbaut sind, eine Montierung zwischen zwei Steckplätzen beispielsweise zwischen 0 und 1 würde aufgrund der im *Postprocessing* gerundeten [RU-Höhen](#), Positionen und Freiflächen inkorrekte Ergebnisse produzieren.

Außerdem zeigte sich bei der Evaluierung des Prototyps, dass die vom Objektdetektor erlernte Klasse *Slots* nicht für die Klassifizierung eines Rack-Fotos genutzt werden kann. Da weder der Trainings- noch der Validierungsdatensatz Bilder enthielt, welche kein Rack abbildeten, lernte das Modell nicht zwischen einem Rack und einem irrelevanten Hintergrund zu unterscheiden.

cessing, jedoch keine fehlerhaften Auswirkungen haben.

¹⁵Das heißt, alle Komponenten befinden sich in einer einheitlichen Tiefe und die Kamera wurde für das Foto parallel zum Rack ausgerichtet.

¹⁶Dieses ist unter <https://cloud.ovgu.de/s/Ny7RN7zKGLQ4HRP> mit dem Passwort **DemoVideo** erreichbar. MD5-Hash: 5807d4f8e09393468866e4420591dc7a

Ferner ließ sich eine suboptimale *User Experience* bei der Steckplatzanalyse von mehreren Teilfotos in der App feststellen. Diese Teilfotos sollten das Rack durch die mehrfachen Eingrenzung des „Einbauchassis“ abbilden. Da diese Eingrenzungen jedoch nacheinander gefordert waren, musste sich die exakte Position der vorherigen Begrenzungen gemerkt werden, dies zeigt sich bei der Betrachtung des zweiten Demovideos¹⁷. In zukünftigen App-Versionen sollte das markierte Viereck auch im Nachhinein für die Nutzenden ersichtlich und veränderbar sein, um eine bessere Benutzung der App zu ermöglichen.

¹⁷<https://cloud.ovgu.de/s/6BWt7FnrzLspJei> mit dem Passwort **DemoVideo**. MD5-Hash: 95906a6ff27860dd3d8b88f3b606aac8

7 Zusammenfassung und weitere Forschungsmöglichkeiten

In dieser Bachelorarbeit wurde ein Prototyp einer auf *Deep Learning* basierenden Steckplatzanalyse-App für Serverschränke entwickelt.

Es wurde gezeigt, dass sich mittels aktueller Objekterkennungs-Modelle in Verbindung mit den Techniken *Transfer Learning* und *Data Augmentation* eine Objekterkennung von Serverkomponenten erlernen ließ, trotzdem dafür nur ein kleiner Datensatz vorhanden war. Dabei stellte sich *YOLOv5* [GAA⁺22] als der beste Objektdetektor für die hiesige Problemstellung heraus. Da die Generalisierungsfähigkeit der Detektoren jedoch als begrenzt einzuschätzen ist, ist ein weiteres Training auf einen größeren Datensatz zu empfehlen. Hierfür müssten zum einen weitere Daten akquiriert werden. Zum anderen ließ sich der Datensatz durch die genauen Annotationen der bisherigen Daten auch synthetisch vergrößern. Dafür müssten die annotierten Objekte ausgeschnitten und in verschiedensten Permutationen in die Racks projiziert werden.

Die Objekterkennungsergebnisse der Detektoren wurden mittels eigens für diese Arbeit definierten Algorithmen verarbeitet und die Informationen der Steckplatzanalyse daraus entnommen.

Abschließend wurden die Genauigkeit und Korrektheit dieser Informationen basierend auf mehreren Test-Szenarien evaluiert. Dabei konnte gezeigt werden, dass das entworfene Verfahren grundsätzlich der richtigen Logik folgt. Jedoch war der entwickelte Prototyp letztlich nicht in der Lage, den geforderten Genauigkeits- und Korrektheitszielen dieser Arbeit zu entsprechen.

Zurückzuführen ist diese Tatsache zum einen auf die Verzerrung der Seitenverhältnisse durch die perspektivische Transformation der Rack-Fotos. Weiterhin trugen eine zu große Kameraneigung und unterschiedlich tief montierte Komponenten, die eine perspektivische Überlagerung zur Folge hatten, zur erwähnten Feststellung bei. Auch eine Anpassung des perspektivischen Transformationsverfahrens brachte keine Verbesserung der Evaluierungsergebnisse mit sich.

Demnach wird für eine optimale Nutzung des Prototyps empfohlen, dass die zu analysierenden Fotos zum einen nahezu keine Neigung zu den Racks aufweisen sollten. Zum anderen dürfen die Racks ausschließlich Komponenten einer einheitlichen montierten Tiefe besitzen, um eine bildbasierte Steckplatzanalyse zu ermöglichen.

Damit die entwickelte Steckplatzanalyse-App in Echt-Welt-Szenarien für den MEC-Ausbau eingesetzt werden kann, muss zusätzlich sichergestellt werden, dass der Ansatz für einen größeren, optimalen Evaluierungsdatensatz die Genauigkeitsanforderungen erreicht. Ebenso ist es von Relevanz, dass die MEC-Racks konsistent die Voraussetzung für eine optimale Steckplatzanalyse erfüllen.

Des Weiteren müssten die Funktionalitäten der App und des Servers angepasst werden, da der Prototyp lediglich das Ausführen von Steckplatzanalysen ermöglicht. Sowohl eine konsistente Speicherung der Rack-Auslastung als auch ein Vergleich mit vorherigen Zu-

ständen bleiben bislang aus.

Außerdem wäre es wünschenswert, den Prozess des manuellen Markierens der „Einbauchassis“ zu entfernen, um somit eine einfachere und schnellere Steckplatzanalyse zu schaffen. Dies könnte gelingen, indem ein weiteres [KNN](#) auf einen großen Datensatz trainiert wird, welches die Eckpunkte der „Einbauchassis“ präzise voraussagt. Diese Voraussagen müssten jedoch auf einem Video basieren, damit die Steckplatzanalyse auch für Racks, welche sich nicht durch ein einziges Foto aufzeichnen lassen, ermöglicht wird.

Um der App weitere Funktionalitäten hinzuzufügen, könnte an den verbauten Komponenten ein eindeutiger *QR*-Code angebracht werden, der zusätzliche Informationen zu der jeweiligen Komponente enthalten könnte. Dabei müsste jedoch eruiert werden, inwiefern ein solches durch *QR*-Codes unterstütztes Verfahren von gehäuftem Kabelvorkommen im Rack gestört werden würde.

Alles in allem wurde durch das entwickelte Konzept eine automatische Dokumentationsmöglichkeit geschaffen, die, wenn überhaupt, nur basierend auf mehreren Voraussetzungen funktionieren kann. Demnach stellt das Ergebnis der Arbeit keine voll umfassende Grundlage für die Lösung der zeitaufwendigen und teuren Dokumentationen beim [MEC](#)-Ausbau dar. Jedoch kann das Ergebnis der Arbeit verwendet werden, um die Schwierigkeiten einer bildbasierten Steckplatzanalyse herauszustellen. Außerdem könnte das Verfahren in der Lage sein, durch einige Anpassungen und Verbesserungen für den Einsatz eines eingeschränkteren Anwendungsfalls zu dienen. Beispielsweise könnte der Prototyp für spezifische Racks eines einzelnen Herstellers oder Netzbetreibers, welche die Voraussetzungen einer optimalen Steckplatzanalyse erfüllen, in Betracht gezogen werden. Dafür müsste jedoch erst durch eine weitere Überprüfung bewiesen werden, dass das Verfahren für die eingeschränkten Szenarien funktioniert.

A Anhang

Pseudocode und Software-Architektur

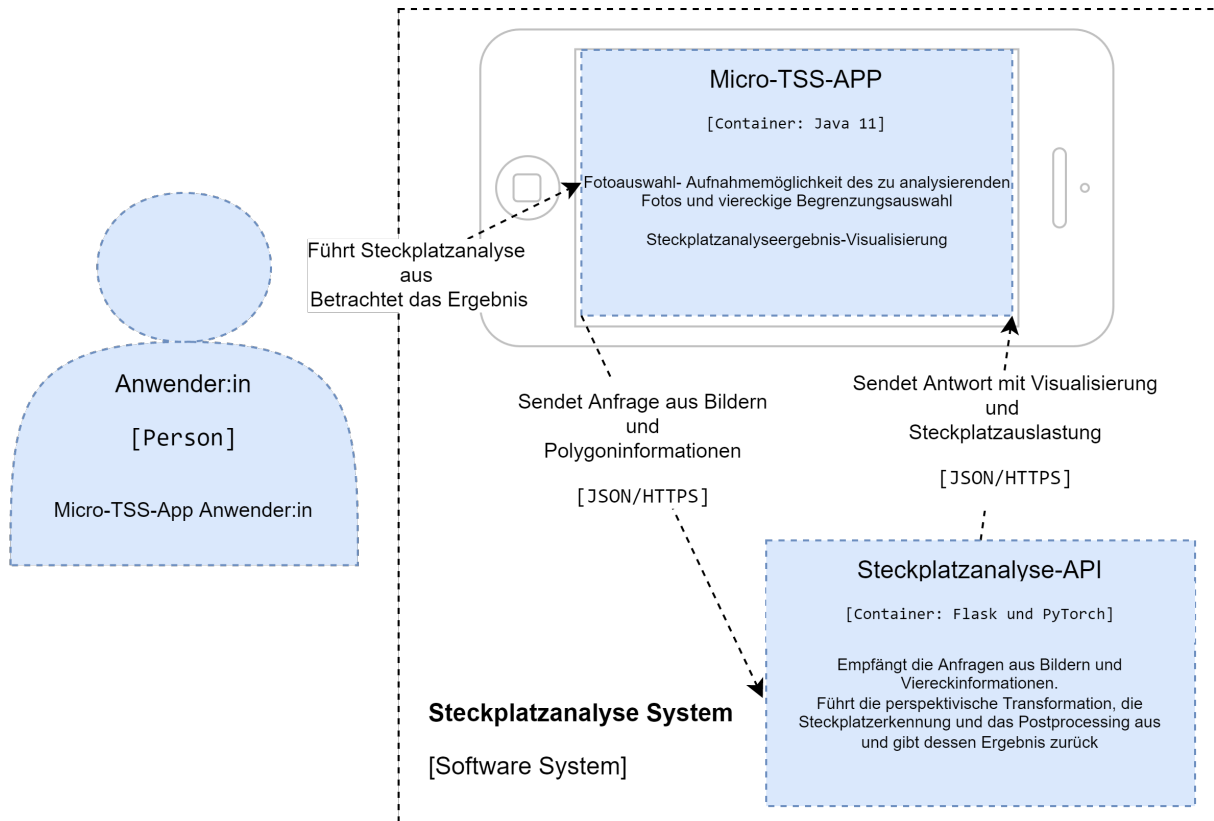


Abbildung A.1: Steckplatzanalyse-System als *C4 Container Diagram* [VIGHGP20]

Algorithmus A.3: getEnlargedPredictionsSorted

input : Liste $preds$ aus n Vorhersagen in der Form $n \times [bbox, label, score]$. $confThresh$ als Schwellenwert für die Filterung der Vorhersagen basierend auf den *Confidence*-Werten. w als Breite des dazugehörigen Bildes.

output : Eine sortierte Liste von gefilterten, auf die Bildbreite angepassten Vorhersagen $preds$, ohne die Vorhersagen des Einbauchassis.

$preds \leftarrow \{p \in preds : p.conf \geq confThresh \wedge p.label \neq slots\};$

foreach p *in* $preds$ **do**

$p.bbox.xmin \leftarrow 0;$

$p.bbox.xmax \leftarrow w;$

end

$preds \leftarrow \text{timSort}(preds, \text{key} = \text{lambda } p: p.bbox.ymin)$

Algorithmus A.1: getOptimizedPredictions

input : Liste *preds* aus *n* Vorhersagen der Form $n \times [bbox, label, score]$. *confThresh* als Schwellenwert für die Filterung der *Confidence*-Werte und *pprur* zur Berechnung der RU. *w* als Breite des dazugehörigen Bildes.

output: *optimized* als eine optimierte Liste der Vorhersagen.

preds $\leftarrow$ getEnlargedPredictionsSorted(*preds*, *confThresh*, *w*);

optimized $\leftarrow \emptyset$; *usedPredIds* $\leftarrow \emptyset$;

for *i* $\leftarrow 0$ to *n* **do**

if *i* $\in$ *usedPredIds* **then**

continue;

end

p $\leftarrow$ *preds*[*i*];

 // Überprüfe für jede nicht bereits optimierte Vorhersage, ob es mit anderen Vorhersagen Überschneidungen gibt.

for *j* $\leftarrow i + 1$ to *n* **do**

pNext $\leftarrow$ *preds*[*j*];

 // Überprüfe, ob der geometrische Schwerpunkt von *p* innerhalb einer anderen Vorhersage *pNext* liegt oder umgekehrt.

if *pNext.bbox.centroid.within*(*p.bbox*) $\vee$ *p.bbox.centroid.within*(*pNext.bbox*) **then**
 // Wenn die Klassen der sich überschneidenden Vorhersagen identisch sind, dann vereinige diese.

if *p.label* = *pNext.label* **then**

p.box.ymin $\leftarrow$ *min*(*p.box.ymin*, *pNext.box.ymin*);

p.box.ymax $\leftarrow$ *max*(*p.box.ymax*, *pNext.box.ymax*);

p.conf $\leftarrow$ (*p.conf* + *pNext.conf*)/2;

p.ru $\leftarrow$ *round*((*p.box.ymax* - *p.box.ymin*)/*pprur*);

usedPredIds.insert(*j*);

end

 // Wenn eine der Vorhersagen als Blende klassifiziert wurde, sortiere die Vorhersage mit geringerem Confidence-Wert aus.

else if *p.label* = *Blende* $\vee$ *pNext.label* = *Blende* **then**

if *pNext.conf* > *p.conf* **then**

p $\leftarrow$ *pNext*;

end

usedPredIds.insert(*j*);

end

 // Sortiere die Vorhersage, welche keine Baugruppe ist aus.

else if *p.label* = *Baugruppe* **then**

usedPredIds.insert(*j*);

end

else if *pNext.label* = *Baugruppe* **then**

p $\leftarrow$ *pNext*; *usedPredIds.insert*(*j*);

end

else

break;

end

end

optimized.insert(*p*);

end

Algorithmus A.2: getSteckplatzAnalyse

input : Liste *preds* von *n* Vorhersagen in der Form $n \times \{bbox, label, score\}$. Schwellenwert *confThresh* durch den Vorhersagen mit zu geringem *Confidence-Wert* herausgefiltert werden. *emptyThresh* Schwellenwert legt fest, wie viel RU eine freie Fläche groß sein muss, um als freier Steckplatz klassifiziert zu werden. *w* als Breite und *h* als Höhe des dazugehörigen Bildes.

output : analyse als sortierte Liste von *m* optimierten Vorhersagen in der Form $m \times \{bbox, label, ru, isEmpty, startPos, endPos\}$.

```

sp ← 0; spPX ← 0; // Aktuelle Startposition in Nummer und Pixel
pprur ← h/round(h/(w*44.45/0.4826)); // optimiertes PPRUR
analyse ← ∅; usedPredIds ← ∅;
preds ← getOptimizedPredictions(preds, confThresh, pprur);
for i ← 0 to n do
    p ← preds[i]; p.ru ← round((p.bbox.ymax - p.bbox.ymin)/pprur);
    spaceAbove ← (p.bbox.ymin - spPX)/pprur; // Pixelabstand der BBox nach oben
    if spaceAbove > emptyThresh then // Füge leeren Steckplatz hinzu
        analyse.insert(createEmptySlot([0, spPX, w, p.bbox.ymin], round(spaceAbove), sp));
        sp ← analyse.last().endPos;
        spPX ← analyse.last().bbox.ymax;
    end
    if i ∉ usedPredIds then
        p.startPos ← sp;
        tmpBox ← [0, spPX, w, p.bbox.ymax]; // Temporäre BBox für Überschneidungen
        predsToAdjust ← ∅; predsToAdjust.insert(p);
        for j ← i + 1 to n do
            pNext ← preds[j];
            if tmpBox.centroid.within(pNext.bbox) ∨ pNext.bbox.centroid.within(tmpBox)
            then
                tmpBox.ymax ← max(tmpBox.ymax, pNext.bbox.ymax);
                predsToAdjust.insert(pNext); usedPredIds.insert(j);
            end
        end
        end
        for j ← 0 to |predsToAdjust| do
            pNew ← createPredictedSlot(predsToAdjust, tmpBox, p.startPos, j, pprur);
            sp ← max(sp, pNew.endPos); analyse.insert(pNew);
        end
        spPX ← tmpBox.ymax;
    end
end
spaceBelow ← (h - spPX)/pprur; // Pixelabstand nach unten
if spaceBelow > emptyThresh then
    analyse.insert(createEmptySlot([0, spPX, w, h], round(spaceBelow), sp));
end

```

Algorithmus A.4: createEmptySlot

input : Begrenzungsrahmen $bbox$ in Form $[xmin, ymin, xmin, xmax]$, Normierte Höhe als $ru \in \mathbf{N}$, Startposition als $sp \in \mathbf{N}$.
output : Freier Steckplatz emSlot in der Form $\{bbox, label, ru, isEmpty, startPos, endPos\}$.
 emSlot. $bbox \leftarrow bbox$;
 emSlot. $label \leftarrow \text{Frei}$;
 emSlot. $isEmpty \leftarrow \text{true}$;
 emSlot. $startPos \leftarrow sp$;
 emSlot. $endPos \leftarrow \text{emSlot.startPos} + \text{emSlot.ru}$;

Algorithmus A.5: createPredictedSlot

input : Liste $preds$ von n Vorhersagen in der Form $n \times bbox, label, score, ru, startPos$, temporärer Begrenzungsrahmen $tmpBBox$ der Form $[xmin, ymin, xmax, ymax]$, Aktuelle Startposition als $sp \in \mathbf{N}$, Index $i \in \mathbf{N}$ des zu erstellenden Steckplatzes aus $preds$, Pixel per Rack Unit Ratio $pprur \in \mathbf{R}$.
output : Steckplatz pNew in der Form $\{bbox, label, ru, isEmpty, startPos, endPos\}$.
 pNew $\leftarrow preds[i]$;
 if $i > 0$ then // pNew ist überschschnittene Vorhersage die unterhalb sp beginnt
 distFromSp $\leftarrow \text{pNew.bbox.ymin} - \text{tmpBBox.ymin}$;
 tmpBoxPxHeight $\leftarrow \text{tmpBBox.ymax} - \text{tmpBBox.ymin}$;
 distRatio $\leftarrow \text{distFromSp} / \text{tmpBoxPxHeight}$;
 pNew.startPos $\leftarrow \text{round}(sp + (\text{distRatio} * \text{round}(\text{tmpBoxPxHeight} / \text{pprur})))$;
 end
 pNew.bbox.ymax $\leftarrow \text{tmpBBox.ymax}$; // tmpBBox beinhaltet entweder pNew.bbox oder den vereinigten Begrenzungsrahmen aus Fachböden und Geräten
 pNew.ru $\leftarrow \text{round}((\text{pNew.ymax} - \text{pNew.bbox.ymin}) / \text{pprur})$;
 pNew.endPos $\leftarrow \text{pNew.startPos} + \text{pNew.ru}$;
 switch pNew.label do
 case Baugruppe do pNew.isEmpty $\leftarrow \text{false}$;
 case Gerät do pNew.isEmpty $\leftarrow \text{false}$;
 case Blende do pNew.isEmpty $\leftarrow \text{true}$;
 case Fachboden do
 if $n > 1$ then // Fachboden war an Überschneidung beteiligt
 pNew.isEmpty $\leftarrow \text{false}$;
 else
 pNew.isEmpty $\leftarrow \text{true}$;
 end
 end
 end
 end

Vergleich zwischen Evaluierungs- und App-Ergebnissen

Da die Evaluierung automatisch und unabhängig von der *Mirco-TSS-App* ausgeführt wurde, wurden die Evaluierungsergebnisse stichprobenartigen mit den Ergebnissen resultierend aus der App verglichen. Dies sollte sicherstellen, dass die gewonnenen Evaluierungsergebnisse auch bei Verwendung der App zutreffen.

Für die Stichproben wurden die Racks **hhi_Rack0** und **hhi_Rack1** in dessen Vorder- und Rückseite gewählt. Diese wurden durch die auf *YOLOv5* basierende Steckplatzanalyse mit einem *Konfidenz-Schwellenwert* von 0.5, einer *PPMR*-Konstante von 482.6 mm und einem *Freiflächen*-Schwellenwert von 0.5 analysiert.

Das erste Rack wurde durch drei verschiedene Fotos auf der Vorderseite und durch zwei Fotos auf der Rückseite abgebildet. In [Abbildung A.2](#) ist die Steckplatzanalyse für die Vorderseite des ersten Racks mittels der App, mit dem jeweiligen Analyseergebnis der Evaluierung (siehe [Tabelle A.1](#)) gegenübergestellt. Das Bildschirmfoto zeigt dabei nur einen Teil des Bildschirms, da die Untereinanderreihung von 45 Steckplätzen nicht adäquat auf einer Seite darstellbar ist.

Im Vergleich des gesamten Bildschirmfotos mit der Tabelle zeigte sich, dass das in der App visualisierte Ergebnis, mit dem Ergebnis der Evaluierung übereinstimmt.

Dies ließ sich außerdem für die weiteren Stichproben feststellen.

Insgesamt wurden in den Stichproben neun Fotos analysiert. Dadurch wurden mehr als 17% des gesamten Evaluierungsdatensatzes abgedeckt. Da die Ergebnisse bei allen Stichproben mit den Evaluierungsergebnissen übereinstimmten, kann davon ausgegangen werden, dass die durch die Evaluierung ermittelten Genauigkeiten auch bei der Verwendung der *Micro-TSS-App* gelten.

Position	Objektnummer	istFrei	RU	Typ
0	0	True	1	empty
1	1	False	1	device
2	2	True	3	empty
3	2	True	3	empty
4	2	True	3	empty
5	3	False	1	device
6	4	False	1	device
7	5	False	1	device
8	6	False	1	device
9	7	False	2	device
10	7	False	2	device
11	8	False	2	device
12	8	False	2	device
13	9	True	1	blend
14	10	True	1	blend
15	11	True	1	blend
16	12	True	1	blend
17	13	True	1	blend
18	14	True	1	blend
19	15	False	1	device
20	16	True	1	empty
21	17	False	1	device
22	18	True	1	empty
23	19	False	2	device
24	19	False	2	device
25	20	False	2	device
26	20	False	2	device
27	21	True	1	blend
28	22	True	1	blend
29	23	True	1	blend
30	24	True	1	blend
31	25	True	1	empty
32	26	True	1	blend
33	27	True	1	blend
34	28	True	2	empty
35	28	True	2	empty
36	29	False	3	device
37	29	False	3	device
38	29	False	3	device
39	30	False	2	device
40	30	False	2	device
41	31	False	3	device
42	31	False	3	device
43	31	False	3	device

Tabelle A.1: Darstellung des Evaluierungsergebnisses vom Rack **hhi_Rack0** basierend auf der Steckplatzanalyse mit *YOLOv5*.

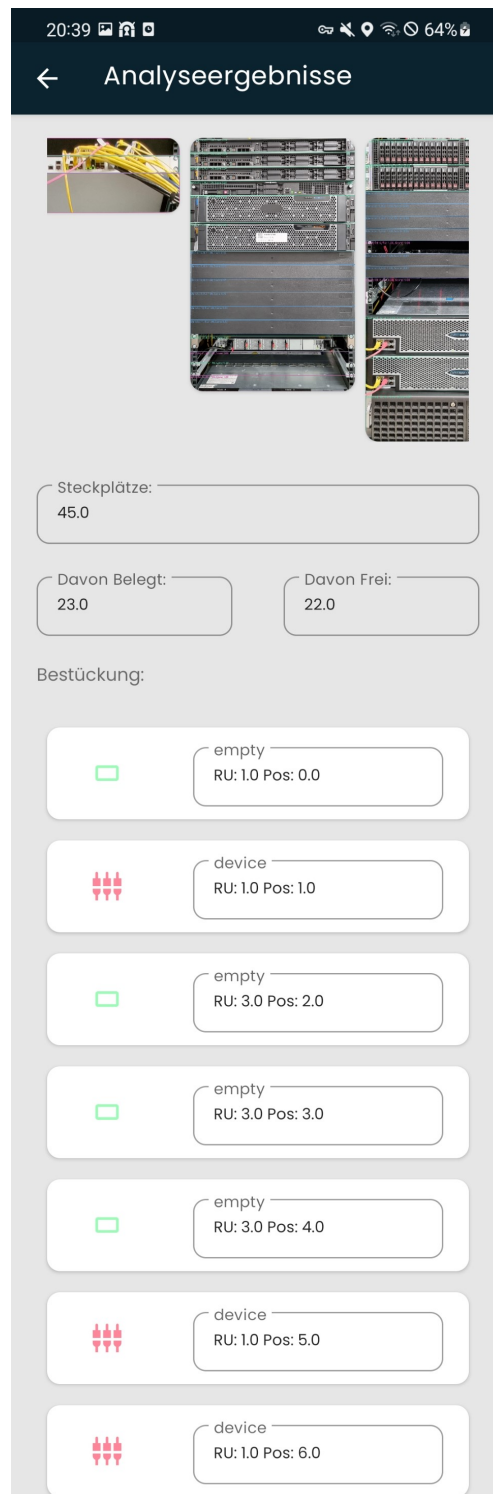


Abbildung A.2: Bildschirmaufnahme der Visualisierung der Steckplatzanalyse des Racks **hhi_Rack0**

Einfluss der Parameterveränderungen auf die Evaluierungsergebnisse

Daten	Seite	Model	$\overline{acc}$	$\overline{mae}_{RU}$	$\overline{mae}_{POS}$	mae_{RACK}	mae_{ANZ}	$error$
hhi	Back	Mask R-CNN@0.5	0.808806	1.183230	19.837733	3.833333	2.333333	5.198108
hhi	Front	Mask R-CNN@0.5	0.837477	0.578784	1.532828	3.625000	4.375000	1.643371
hhi	Back	YOLOv5@0.6	0.893913	0.601708	20.363354	3.000000	1.000000	2.648480
hhi	Front	YOLOv5@0.6	0.944149	0.640152	1.919872	4.875000	3.750000	0.624695
highstreet	Back	Mask R-CNN@0.5	0.799114	4.500000	1.280000	4.000000	6.000000	3.169980
highstreet	Front	Mask R-CNN@0.5	0.911628	0.129032	0.529412	2.600000	2.600000	0.517723
highstreet	Back	YOLOv5@0.6	0.915282	1.333333	0.500000	5.200000	0.600000	0.646678
highstreet	Front	YOLOv5@0.6	0.916279	0.054054	1.000000	3.000000	2.400000	0.540339

Tabelle A.2: Darstellung der Evaluierungsergebnisse basierend auf 465mm als PPMR-Konstante

Daten	Seite	Model	FFSWT	$\overline{acc}$	$\overline{mae}_{RU}$	$\overline{mae}_{POS}$	mae_{RACK}	mae_{ANZ}	$error$
hhi	Back	Mask R-CNN@0.5	0.25	0.808806	1.326087	20.409161	2.166667	2.333333	5.016018
hhi	Front	Mask R-CNN@0.5	0.25	0.837477	0.502444	1.320707	2.125000	4.375000	1.352705
hhi	Back	Mask R-CNN@0.5	0.50	0.808806	1.326087	20.409161	2.166667	2.333333	5.016018
hhi	Front	Mask R-CNN@0.5	0.50	0.837477	0.502444	1.320707	2.125000	4.375000	1.352705
hhi	Back	Mask R-CNN@0.5	0.75	0.808806	1.326087	20.409161	2.166667	2.333333	5.016018
hhi	Front	Mask R-CNN@0.5	0.75	0.837477	0.578784	1.532828	3.625000	4.375000	1.643371
hhi	Back	YOLOv5@0.5	0.25	0.893913	0.514752	20.811335	1.833333	1.000000	2.563012
hhi	Front	YOLOv5@0.5	0.25	0.941489	0.594697	1.194444	3.875000	3.625000	0.543514
hhi	Back	YOLOv5@0.5	0.50	0.893913	0.514752	20.811335	1.833333	1.000000	2.563012
hhi	Front	YOLOv5@0.5	0.50	0.941489	0.594697	1.194444	3.875000	3.625000	0.543514
hhi	Back	YOLOv5@0.5	0.75	0.893913	0.514752	20.811335	1.833333	1.000000	2.563012
hhi	Front	YOLOv5@0.5	0.75	0.944149	0.640152	1.919872	4.875000	3.750000	0.624695
highstreet	Back	Mask R-CNN@0.5	0.25	0.799114	4.333333	1.280000	3.200000	6.000000	2.975790
highstreet	Front	Mask R-CNN@0.5	0.25	0.911628	0.129032	0.529412	2.400000	2.400000	0.482374
highstreet	Back	Mask R-CNN@0.5	0.50	0.799114	4.333333	1.280000	3.200000	6.000000	2.975790
highstreet	Front	Mask R-CNN@0.5	0.50	0.911628	0.129032	0.529412	2.400000	2.400000	0.482374
highstreet	Back	Mask R-CNN@0.5	0.75	0.799114	4.333333	1.280000	3.200000	6.000000	2.975790
highstreet	Front	Mask R-CNN@0.5	0.75	0.911628	0.129032	0.529412	2.600000	2.600000	0.517723
highstreet	Back	YOLOv5@0.5	0.25	0.915282	1.000000	0.166667	4.400000	0.600000	0.522425
highstreet	Front	YOLOv5@0.5	0.25	0.920930	0.052632	1.000000	2.800000	2.200000	0.478580
highstreet	Back	YOLOv5@0.5	0.50	0.915282	1.000000	0.166667	4.400000	0.600000	0.522425
highstreet	Front	YOLOv5@0.5	0.50	0.920930	0.052632	1.000000	2.800000	2.200000	0.478580
highstreet	Back	YOLOv5@0.5	0.75	0.915282	1.000000	0.166667	4.400000	0.600000	0.522425
highstreet	Front	YOLOv5@0.5	0.75	0.916279	0.054054	1.000000	3.000000	2.400000	0.540339

Tabelle A.3: Darstellung der Evaluierungsergebnisse basierend auf den Freiflächen-Schwellenwerten (FFSWT) 0.25, 0.5 und 0.75.

Potenzielle Fehlerquellen

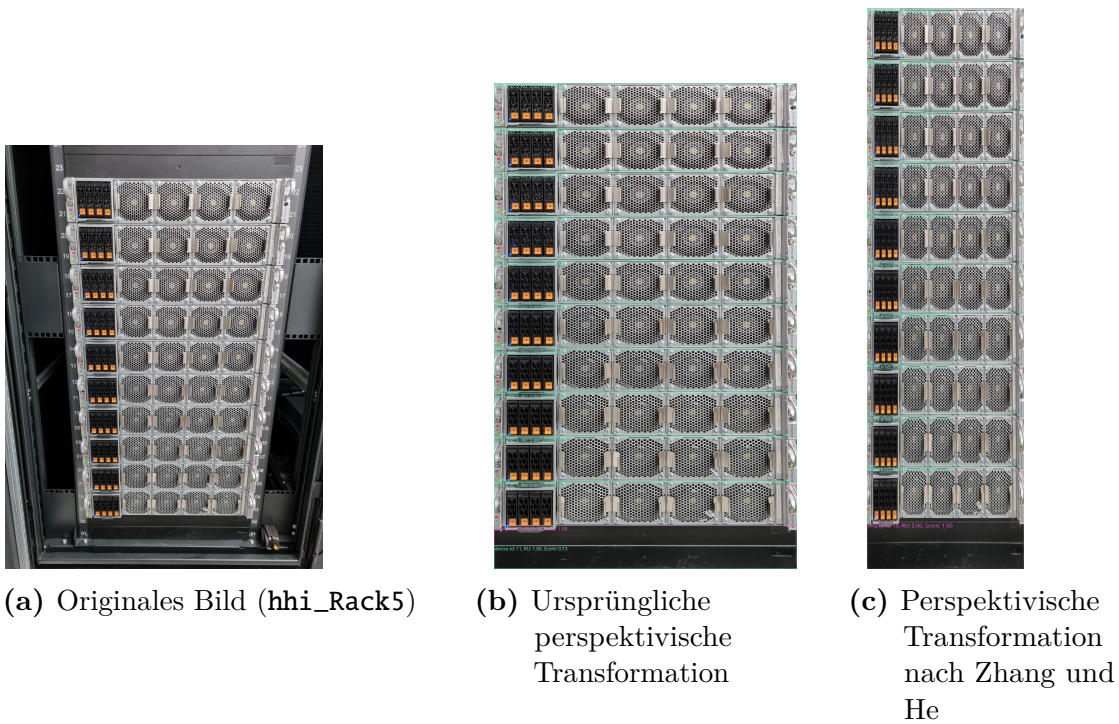


Abbildung A.3: Visualisierung einer fehlerhaften Verzerrung der Seitenverhältnisse durch den von Zhang und He [ZL07] veröffentlichten Ansatz.

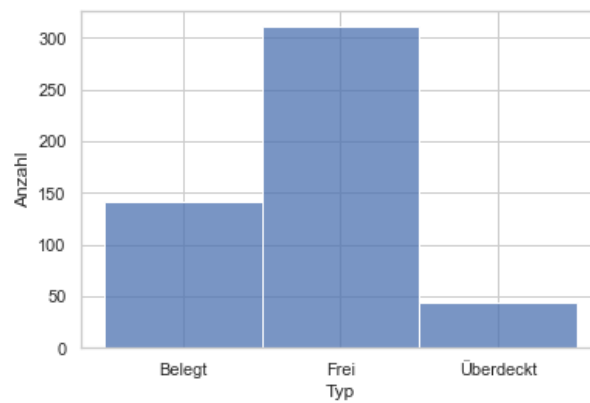


Abbildung A.4: Verhältnis von überlagerten Steckplätzen zu freien und belegten Steckplätzen.

Literaturverzeichnis

- [ACH20] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *ArXiv*, abs/2004.10934, 2020. (Zitiert auf Seite 36)
- [AJNP18] Nicolas Auger, Vincent Jugé, Cyril Nicaud, and Carine Pivoteau. On the worst-case complexity of timsort. *arXiv preprint arXiv:1805.08612*, 2018. (Zitiert auf Seite 63)
- [Avi22] Avinash Sajnashetty. Deploying pytorch in python via a rest api with flask — pytorch tutorials 1.11.0+cu102 documentation, 14.06.2022. URL: https://pytorch.org/tutorials/intermediate/flask_rest_api_tutorial.html. (Zitiert auf Seite 65)
- [BIK⁺20] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020. URL: <https://www.mdpi.com/2078-2489/11/2/125>, doi:10.3390/info11020125. (Zitiert auf den Seiten 53 und 60)
- [Bro19] Justin Brooks. imantics. <https://github.com/jsbrooks/imantics>, 2019. (Zitiert auf Seite 50)
- [BSF94] Bengio Y., Simard P., and Frasconi P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi:10.1109/72.279181. (Zitiert auf Seite 23)
- [CD14] T. Chai and R. R. Draxler. Root mean square error (rmse) or mean absolute error (mae)? – arguments against avoiding rmse in the literature. *Geoscientific Model Development*, 7(3):1247–1250, 2014. doi:10.5194/gmd-7-1247-2014. (Zitiert auf Seite 74)
- [CHM⁺07] M. J. Callaghan, J. Harkin, E. McColgan, T. M. McGinnity, and L. P. Maguire. Client–server architecture for collaborative remote experimentation. *Journal of Network and Computer Applications*, 30(4):1295–1308, 2007. doi:10.1016/j.jnca.2006.09.006. (Zitiert auf Seite 64)
- [Cur16] Alex Curran. Showcaseview. <https://github.com/amlcurran/ShowcaseView>, 2016. (Zitiert auf Seite 67)
- [DDS⁺09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Image-net: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, Piscataway, NJ, 2009. IEEE. doi:10.1109/CVPR.2009.5206848. (Zitiert auf Seite 26)

- [DIN] DIN EN 60297-3-106:2011-01, Bauweisen für elektronische Einrichtungen-Maße der 482,6-mm-(19-Zoll-)Bauweise- Teil 3-106: Adaptionsmaße für Baugruppenträger und Einschübe, geeignet für metrische Schränke oder Gestelle nach IEC 60917-2-1 (IEC 60297-3-106:2010); Deutsche Fassung EN 60297-3-106:2010. doi:10.31030/1723606. (Zitiert auf den Seiten 40 und 77)
- [DT20] M. G. Dorrer and A. E. Tolmacheva. Comparison of the yolov3 and mask r-cnn architectures' efficiency in the smart refrigerator's computer vision. *Journal of Physics: Conference Series*, 1679(4):042022, 2020. doi:10.1088/1742-6596/1679/4/042022. (Zitiert auf Seite 37)
- [DZW20] Lixuan Du, Rongyu Zhang, and Xiaotian Wang. Overview of two-stage object detection algorithms. *Journal of Physics: Conference Series*, 1544(1):012033, 2020. doi:10.1088/1742-6596/1544/1/012033. (Zitiert auf Seite 27)
- [EEv⁺15] Mark Everingham, S. M. Ali Eslami, Luc van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, 2015. doi:10.1007/s11263-014-0733-5. (Zitiert auf Seite 25)
- [Ert16] Wolfgang Ertel. *Grundkurs Künstliche Intelligenz*. Springer Fachmedien Wiesbaden, Wiesbaden, 2016. doi:10.1007/978-3-658-13549-2. (Zitiert auf Seite 9)
- [EvW⁺10] Mark Everingham, Luc van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010. doi:10.1007/s11263-009-0275-4. (Zitiert auf den Seiten 18 und 25)
- [EY21] Misheel Enkhbaatar and Tatsuya Yamazaki. Development of an automatic visual monitoring system for data center devices. In *2021 IEEE-EE 16th International Conference on Computer Sciences and Information Technologies (CSIT)*, pages 194–197, Piscataway, NJ, 2021. IEEE. doi:10.1109/CSIT52700.2021.9648588. (Zitiert auf den Seiten 1 und 41)
- [FT02] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, 2002. doi:10.1145/514183.514185. (Zitiert auf Seite 65)
- [GAA⁺22] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, TaoXie, Jiacong Fang, imyhxy, Kalen Michael, Lorna, Abhiram V, Diego Montes, Jebastin Nadar, Laughing, tkianai, yxNONG, Piotr Skalski, Zhiqiang Wang, Adam Hogan, Cristi Fati, Lorenzo Mammana, Alex-Wang, Deep Patel, Ding Yiwei, Felix You, Jan Hajek, Laurentiu Diaconu, and Mai Thanh Minh. ultralytics/yolov5: v6.1 - tensorrt, tensorflow edge tpu and openvino export and inference, 2022. doi:10.5281/ZENODO.6222936. (Zitiert auf den Seiten 36, 37, 52, 59, und 85)
- [Gal17] Silvio Galesso. Instance segmentation with mask r-cnn - seminar: Current works in computer vision. 2017. URL: <https://lmb.informatik.uni-freiburg>.

- [de/lectures/seminar_brox/seminar_ss17/maskrcnn_slides.pdf](#). (Zitiert auf den Seiten 28, 30, 31, und 33)
- [GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587. IEEE, 6/23/2014 - 6/28/2014. doi:10.1109/CVPR.2014.81. (Zitiert auf Seite 27)
- [GDG⁺17] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. (Zitiert auf Seite 56)
- [Gir15] Girshick Ross. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015. doi:10.1109/ICCV.2015.169. (Zitiert auf den Seiten 28, 32, 35, und 55)
- [GSR22] Glenn Jocher, Sudhanshu Singh, and Robin Cole. yolov5/readme.md at 24bea5e4b7ad369753f45e93b736d9205c37d20a ultralytics/yolov5, 15.06.2022. URL: https://github.com/ultralytics/yolov5/blob/24bea5e4b7ad369753f45e93b736d9205c37d20a/utils/flask_rest_api/README.md. (Zitiert auf Seite 65)
- [HGDG17] He Kaiming, Gkioxari Georgia, Dollár Piotr, and Girshick Ross. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017. doi:10.1109/ICCV.2017.322. (Zitiert auf den Seiten 32, 33, 34, 54, und 55)
- [HRS16] Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1225–1234, New York, New York, USA, 2016. PMLR. URL: <https://proceedings.mlr.press/v48/hardt16.html>. (Zitiert auf Seite 13)
- [HWX14] Tao Hong, Jason Wilson, and Jingrui Xie. Long term probabilistic load forecasting and normalization with hourly information. *IEEE Transactions on Smart Grid*, 5(1):456–462, 2014. doi:10.1109/TSG.2013.2274373. (Zitiert auf Seite 12)
- [HZRS16] He Kaiming, Zhang Xiangyu, Ren Shaoqing, and Sun Jian. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi:10.1109/CVPR.2016.90. (Zitiert auf den Seiten 23, 55, und 56)
- [IF19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. (Zitiert auf Seite 56)
- [Ild19] Ildoo Kim. pytorch-gradual-warmup-lr. <https://github.com/ildoonet/pytorch-gradual-warmup-lr>, 2019. (Zitiert auf Seite 56)

- [IYA16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. (Zitiert auf den Seiten 9, 11, 12, 13, 14, 19, 20, 21, 22, 56, und 61)
- [JA18] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *ArXiv*, abs/1804.02767, 2018. (Zitiert auf Seite 36)
- [Jay22] Jayden Oh Yicong. ohyicong/google-image-scraper: A library to scrap google images, 19.04.2022. URL: <https://github.com/ohyicong/Google-Image-Scraper>. (Zitiert auf Seite 48)
- [JGH15] Junlin Ouyang, Gouenou Coatrieux, and Huazhong Shu. Robust hashing for image authentication using quaternion discrete fourier transform and log-polar transform. *Digital Signal Processing*, 41:98–109, 2015. URL: <https://www.sciencedirect.com/science/article/pii/S1051200415000810>, doi:10.1016/j.dsp.2015.03.006. (Zitiert auf Seite 48)
- [JJDD17] Jianwei Yang, Jiasen Lu, Dhruv Batra, and Devi Parikh. A faster pytorch implementation of faster r-cnn. <https://github.com/jwyang/faster-rcnn.pytorch>, 2017. (Zitiert auf Seite 56)
- [JLJT19] Tanuj Jain, Christopher Lennan, Zubin John, and Dat Tran. Imagededup. <https://github.com/idealo/imagededup>, 2019. (Zitiert auf Seite 48)
- [Jos03] Simon Josefsson. Rfc3548: The base16, base32, and base64 data encodings, 2003. (Zitiert auf den Seiten 66 und 68)
- [JS13] Edwin Amsler Jasdev Singh, Jacob Greenleaf. imgurpython. <https://github.com/Imgur/imgurpython>, 2013. (Zitiert auf Seite 48)
- [Kae16] Adrian Kaehler. *Learning OpenCV 3 : computer vision in C++ with the OpenCV library*. O'Reilly Media, Sebastopol, CA, array edition, 2016. URL: <https://learning.oreilly.com/library/view/-/9781491937983/?ar>. (Zitiert auf Seite 7)
- [KDA⁺17] Ivan Krasin, Tom Duerig, Neil Alldrin, Vittorio Ferrari, Sami Abu-El-Haija, Alina Kuznetsova, Hassan Rom, Jasper Uijlings, Stefan Popov, Shahab Kamali, Matteo Mallocci, Jordi Pont-Tuset, Andreas Veit, Serge Belongie, Victor Gomes, Abhinav Gupta, Chen Sun, Gal Chechik, David Cai, Zheyun Feng, Dhyanesh Narayanan, and Kevin Murphy. Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://storage.googleapis.com/openimages/web/index.html*, 2017. (Zitiert auf den Seiten 26 und 27)
- [Ket17] Nikhil Ketkar. *Deep Learning with Python: A Hands-On Introduction*. Apress L. P, Berkeley, CA, 2017. (Zitiert auf den Seiten 12, 13, 14, 15, 16, 21, und 22)
- [KKKK⁺22] Elżbieta Kubera, Agnieszka Kubik-Komar, Paweł Kurasiński, Krystyna Piotrowska-Weryszko, and Magdalena Skrzypiec. Detection and recognition of pollen grains in multilabel microscopic images. *Sensors*, 22(7), 2022. URL: <https://www.mdpi.com/1424-8220/22/7/2690>, doi:10.3390/s22072690. (Zitiert auf Seite 37)

- [KKT⁺21] Chaladi V. N. Koushik, Rampilla V. N. Kamal, Channagiri Tarun, Koya Dinesh Teja, and Suneeth Manne. An efficient algorithm for prawn detection and length identification. In *Proceedings of International Conference on Computational Intelligence and Data Engineering*, pages 457–467, 2021. (Zitiert auf Seite 44)
- [Kru15] Rudolf Kruse. *Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze*. Springer eBook Collection. Springer Vieweg, Wiesbaden, 2. aufl. 2015 edition, 2015. doi:10.1007/978-3-658-10904-2. (Zitiert auf den Seiten 9, 10, 11, 12, 13, und 16)
- [KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc, 2012. URL: <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>. (Zitiert auf Seite 28)
- [LAE⁺16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer vision - ECCV 2016*, volume 9905 of *Lecture Notes in Computer Science*, pages 21–37. Springer, Cham, 2016. doi:10.1007/978-3-319-46448-0_2. (Zitiert auf Seite 37)
- [LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi:10.1109/5.726791. (Zitiert auf den Seiten 22 und 23)
- [LDG⁺17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. (Zitiert auf den Seiten 33 und 34)
- [LGG⁺17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017. (Zitiert auf Seite 37)
- [LH21] YangFan Li and Xiaoyu He. Covid-19 detection in chest radiograph based on yolo v5. In *2021 IEEE International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI)*, pages 344–347, 2021. doi:10.1109/CEI52496.2021.9574463. (Zitiert auf Seite 37)
- [LMB⁺14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. (Zitiert auf Seite 26)
- [LMB⁺21a] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick,

- and Piotr Dollár. Coco - common objects in context, 25.08.2021. URL: <https://cocodataset.org/#format-data>. (Zitiert auf Seite 26)
- [LMB⁺21b] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Coco - common objects in context, 25.08.2021. URL: <https://cocodataset.org/#detection-2017>. (Zitiert auf Seite 33)
- [LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. (Zitiert auf Seite 32)
- [Luc22] Lucian Vasile. Notyourguy/scraperr: Reddit image scraper made in python, 19.04.2022. URL: <https://github.com/NotYourGuy/scraperr>. (Zitiert auf Seite 48)
- [Mic19] Michael Kaas. Technischer bericht nr. 2019-01. 2019. URL: https://www.hs-niederrhein.de/fileadmin/dateien/FB03/Technische_Berichte/fb03-tb-2019-01.pdf. (Zitiert auf den Seiten 20 und 21)
- [MK17] Izadora Binti Mustaffa and Syawal Fikri Bin Mohd Khairul. Identification of fruit size and maturity through fruit images using opencv-python and raspberry pi. In *Proceedings of 2017 International Conference on Robotics, Automation and Sciences*, pages 1–3, Piscataway, NJ, 2017. IEEE Press. doi:10.1109/ICORAS.2017.8308068. (Zitiert auf Seite 44)
- [Moh20] Arun Mohan. A review on fast rcnn - datadriveninvestor. *DataDrivenInvestor*, 21.05.2020. URL: <https://medium.datadriveninvestor.com/review-on-fast-rcnn-202c9eadd23b>. (Zitiert auf Seite 29)
- [MR10] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In Alberto del Bimbo, editor, *Proceedings of the international conference on Multimedia*, ACM Conferences, page 1485, New York, NY, 2010. ACM. doi:10.1145/1873951.1874254. (Zitiert auf Seite 52)
- [NE22] Upesh Nepal and Hossein Eslamiat. Comparing yolov3, yolov4 and yolov5 for autonomous landing spot detection in faulty uavs. *Sensors (Basel, Switzerland)*, 22(2), 2022. doi:10.3390/s22020464. (Zitiert auf Seite 37)
- [Nob06] William S. Noble. What is a support vector machine? *Nature Biotechnology*, 24(12):1565–1567, 2006. URL: <https://www.nature.com/articles/nbt1206-1565>, doi:10.1038/nbt1206-1565. (Zitiert auf Seite 28)
- [NPd16] Keiller Nogueira, Otávio Penatti, and Jefersson dos Santos. Towards better exploiting convolutional neural networks for remote sensing scene classification. *Pattern Recognition*, 61, 2016. doi:10.1016/j.patcog.2016.07.001. (Zitiert auf Seite 24)
- [Oga21] Kohichi Ogawa. Monitoring method for devices of server racks using linear modules. In *2021 IEEE 10th Global Conference on Consumer Electronics (GCCE)*, pages 627–628, Piscataway, NJ, 2021. IEEE. doi:10.1109/GCCE53005.2021.9621807. (Zitiert auf den Seiten 1 und 41)

- [PAA⁺87] Stephen M. Pizer, E. Philip Amburn, John D. Austin, Robert Cromartie, Ari Geselowitz, Trey Greer, Bart ter Haar Romeny, John B. Zimmerman, and Karel Zuiderveld. Adaptive histogram equalization and its variations. *Computer Vision, Graphics, and Image Processing*, 39(3):355–368, 1987. doi:10.1016/S0734-189X(87)80186-X. (Zitiert auf den Seiten 53 und 60)
- [pdo22] pdochannel. Object detection fasterrcnn tutorial. *Kaggle*, 24.01.2022. URL: <https://www.kaggle.com/code/pdochannel/object-detection-fasterrcnn-tutorial>. (Zitiert auf Seite 53)
- [PGM⁺19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. (Zitiert auf Seite 52)
- [PHP10] Ki-Woong Park, Woomin Hwang, and Kyu Ho Park. Falconeye: Data center status extraction via vision transformation techniques. In Iqbal Ahamed, editor, *2010 IEEE 34th Annual Computer Software and Applications Conference workshops (COMPSAC-W 2010)*, pages 388–392, Piscataway, NJ, 2010. IEEE. doi:10.1109/COMPSACW.2010.73. (Zitiert auf den Seiten 1 und 41)
- [Ple17] Valentin Plenk. Dateiformate: Json. In Valentin Plenk, editor, *Angewandte Netzwerktechnik kompakt*, IT kompakt, pages 41–51. Springer Vieweg, Wiesbaden and Heidelberg, 2017. doi:10.1007/978-3-658-15904-7_3. (Zitiert auf den Seiten 65 und 68)
- [PNd20] Rafael Padilla, Sergio L. Netto, and Eduardo A. B. da Silva. A survey on performance metrics for object-detection algorithms. In Anselmo C. Paiva, editor, *Proceedings of the 2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242, Piscataway, NJ, 2020. IEEE. doi:10.1109/IWSSIP48289.2020.9145130. (Zitiert auf den Seiten 16, 17, und 18)
- [Pri15] Lutz Pries. *Computer Vision*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. doi:10.1007/978-3-662-45129-8. (Zitiert auf Seite 5)
- [PS16] Kuntal Kumar Pal and K. S. Sudeep. Preprocessing for image classification by convolutional neural networks. In *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 1778–1781, Piscataway, NJ, 2016. IEEE. doi:10.1109/RTEICT.2016.7808140. (Zitiert auf Seite 53)
- [PY10] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. doi:10.1109/TKDE.2009.191. (Zitiert auf Seite 24)
- [PyT22a] PyTorch. torchvision.models.detection.mask_rcnn — torchvision 0.12 documentation, 05.06.2022. URL: https://pytorch.org/vision/0.12/_modules/torchvision/models/detection/mask_rcnn.html. (Zitiert auf Seite 54)

- [PyT22b] PyTorch. Torchvision object detection finetuning tutorial — pytorch tutorials 1.11.0+cu102 documentation, 11.05.2022. URL: https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html. (Zitiert auf Seite 53)
- [Qia99] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999. doi:10.1016/S0893-6080(98)00116-6. (Zitiert auf Seite 55)
- [Ras07] Neil Rasmussen. Improving rack cooling performance using airflow management™ blanking panels. *White Paper*, 44:2007–2008, 2007. (Zitiert auf Seite 40)
- [RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788. IEEE, 2016. doi:10.1109/CVPR.2016.91. (Zitiert auf den Seiten 34, 35, 36, und 37)
- [RDS⁺15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. doi:10.1007/s11263-015-0816-y. (Zitiert auf Seite 26)
- [RF17] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *30th IEEE Conference on Computer Vision and Pattern Recognition*, pages 6517–6525, Piscataway, NJ, 2017. IEEE. doi:10.1109/CVPR.2017.690. (Zitiert auf Seite 36)
- [RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015. (Zitiert auf den Seiten 30, 31, 32, 41, und 55)
- [RPG17] Bernadette M. Randles, Irene V. Pasquetto, Milena S. Golshan, and Christine L. Borgman. Using the jupyter notebook as a tool for open science: An empirical study. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 1–2, 2017. doi:10.1109/JCDL.2017.7991618. (Zitiert auf Seite 70)
- [SFI18] Saputra Kurnia, Farhan Khalid, and Irvanizam Irvanizam. Analysis on the comparison of retrofit and volley libraries on android-based mosque application. In *2018 International Conference on Electrical Engineering and Informatics (ICELTICS)*, pages 117–121, 2018. doi:10.1109/ICELTICS.2018.8548881. (Zitiert auf Seite 68)
- [Sha21] Hamza Sharaf. floorplanner. <https://github.com/hamzasharuf/floorplanner>, 2021. (Zitiert auf Seite 67)
- [SIDQ17] Sonia Shahzadi, Muddesar Iqbal, Tasos Dagiuklas, and Zia Ul Qayyum. Multi-access edge computing: open issues, challenges and future perspectives. *Journal of Cloud Computing*, 6(1), 2017. doi:10.1186/s13677-017-0097-9. (Zitiert auf Seite 1)

- [SLJ⁺15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015)*, pages 1–9, Piscataway, NJ, 2015. IEEE. doi:10.1109/CVPR.2015.7298594. (Zitiert auf Seite 23)
- [SMDH13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 2013. PMLR. URL: <https://proceedings.mlr.press/v28/sutskever13.html>. (Zitiert auf den Seiten 55 und 56)
- [Son18] Ankit Soni. Data center monitoring using an improved faster regional convolutional neural network. *International Journal of Advanced Research in Electrical Electronics and Instrumentation Engineering*, 7:1849–1853, 2018. (Zitiert auf Seite 41)
- [SRP⁺17] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2017. (Zitiert auf Seite 23)
- [Sta22] Statista. Multi-access edge computing market revenue 2024 | statista, 01.07.2022. URL: <https://www.statista.com/statistics/1183528/multi-access-edge-computing-market-revenue-worldwide/>. (Zitiert auf Seite 1)
- [SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. (Zitiert auf Seite 37)
- [Sze22] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer eBook Collection. Springer International Publishing and Imprint Springer, Cham, 2nd ed. 2022 edition, 2022. doi:10.1007/978-3-030-34372-9. (Zitiert auf den Seiten 7, 8, 12, 13, und 74)
- [Tom13] E. Tomaszewska. The use of digital images for determining the coordinates of objects in the real plane with a flat four-point transformation. *Геодинаміка*, (2):16–18, 2013. (Zitiert auf Seite 7)
- [TPL20] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. (Zitiert auf Seite 37)
- [UvGS13] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013. doi:10.1007/s11263-013-0620-5. (Zitiert auf Seite 27)

- [Val20] Francesca Valente. 90% of industrial enterprises will utilize edge computing by 2022, finds frost & sullivan. *Frost & Sullivan*, 26.08.2020. URL: <https://www.frost.com/news/press-releases/90-of-industrial-enterprises-will-utilize-edge-computing-by-2022-finds-frost-sullivan/>. (Zitiert auf Seite 1)
- [vF21] Eberhard von Faber. It/tk-services und informationstechnologie. In *IT und IT-Sicherheit in Begriffen und Zusammenhängen*, Springer eBook Collection, pages 85–131. Springer Vieweg, Wiesbaden, 2021. doi:10.1007/978-3-658-33431-4_4. (Zitiert auf den Seiten 39 und 40)
- [VIGHGP20] Andrea Vázquez-Ingelmo, Alicia García-Holgado, and Francisco J. García-Peñalvo. C4 model in a software engineering subject to ease the comprehension of uml and the software. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 919–924, 2020. doi:10.1109/EDUCON45650.2020.9125335. (Zitiert auf den Seiten 64 und 87)
- [Wad] Wada, Kentaro. Labelme: Image polygonal annotation with python. URL: <https://github.com/wkentaro/labelme>, doi:10.5281/zenodo.5711226. (Zitiert auf den Seiten 48 und 70)
- [Wer21] Martin Werner. *Digitale Bildverarbeitung: Grundkurs mit neuronalen Netzen und MATLAB®-Praktikum*. Springer eBook Collection. Springer Vieweg, Wiesbaden, 2021. doi:10.1007/978-3-658-22185-0. (Zitiert auf den Seiten 5, 6, 10, 14, 19, und 20)
- [WLW⁺20] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2020. (Zitiert auf Seite 36)
- [ZK16] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. (Zitiert auf Seite 23)
- [ZL07] Zhengyou Zhang and Li-Wei He. Whiteboard scanning and image enhancement. *Digital Signal Processing*, 17(2):414–432, 2007. URL: <https://www.sciencedirect.com/science/article/pii/S1051200406000595>, doi:10.1016/j.dsp.2006.05.006. (Zitiert auf den Seiten 81 und 94)
- [ZSGY19] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *arXiv preprint arXiv:1905.05055*, 2019. (Zitiert auf den Seiten 25, 27, 28, und 80)
- [ZWL⁺20] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 12993–13000, 2020. (Zitiert auf Seite 60)
- [ZY22] Yanzhao Zhu and Wei Qi Yan. Traffic sign recognition based on deep learning. *Multimedia Tools and Applications*, 81(13):17779–17791, 2022. (Zitiert auf Seite 37)

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig und ausschließlich unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder einer anderen Prüfungsbehörde vorgelegt oder noch anderweitig veröffentlicht.

Unterschrift

Datum