

Masterarbeit

Benchmarking von Deep-Learning-Algorithmen zur Detektion von Objekten im Kontext der Intralogistik

Maximilian Otten

20. Juli 2020

Gutachter:

Prof. Dr. Jakob Rehof

Dr.-Ing. Oliver Urbann

Technische Universität Dortmund
Fakultät für Informatik
Software Engineering (LS-14)
<http://ls14-www.cs.tu-dortmund.de>

Inhaltsverzeichnis

Abkürzungsverzeichnis	v
1 Vorbemerkungen	1
1.1 Motivation und Hintergrund	1
1.2 Zielsetzung und Abgrenzung	2
1.3 Aufbau der Arbeit	2
2 Anforderungsanalyse	5
2.1 Formulierung der Anforderungen an das Benchmark	5
2.2 Rahmenbedingungen	7
2.2.1 Datenquelle	8
2.2.2 Spezifikation der verwendeten Hardware	8
2.2.3 Spezifikation der verwendeten Software	9
2.3 Zusammenfassung der Anforderungen und Rahmenbedingungen	10
3 Grundlagen	13
3.1 Themeneinordnung	13
3.1.1 Begrifflichkeiten der Artificial Intelligence	14
3.1.2 Deep Learning-Algorithmen der Bildverarbeitung	15
3.2 Grundlagen Machine Learning	16
3.2.1 Artifical Neural Networks	16
3.2.2 Generalisierung und Overfitting	19
3.3 Grundlagen Deep Learning	20
3.3.1 Convolutional Neural Networks	20
3.3.2 Typen von Layer in Convolutional Neural Networks	21
3.4 Architektur von Object Detection-Algorithmen	26
3.5 Evaluierungsmetriken von Object Detection-Algorithmen	30
4 Methode und Datensatz	37
4.1 Einordnung CRISP-DM	37
4.2 Aufbau der Benchmark-Datensätze	43

5 Vorauswahl der Benchmark-Algorithmen	51
5.1 Ermittlung der Grundgesamtheit	51
5.2 Aufbereitung der Grundgesamtheit	53
5.3 Filterung der Grundgesamtheit	55
5.3.1 Erarbeitung eines Konzepts zur Filterung	55
5.3.2 Durchführung der Filterung	58
5.3.3 Resultat der Filterung	62
6 Algorithmische Konzepte der Benchmark-Verfahren	63
6.1 Konzepte der Backbone-Architekturen	63
6.1.1 VGG	64
6.1.2 ResNet	64
6.1.3 ResNeXt	65
6.1.4 Darknet-53	66
6.1.5 FPN	66
6.2 Konzepte der Object Detection-Algorithmen	67
6.2.1 HTC	67
6.2.2 RFBNet	70
6.2.3 ATSS	73
6.2.4 YOLO-v3 ASFF	76
7 Modellierung der Benchmark-Algorithmen	79
7.1 Ermittlung zusätzlicher Einflussfaktoren	79
7.1.1 Einflussfaktoren der aktuellen Literatur	80
7.1.2 Einflussfaktoren im Benchmark	81
7.2 Hyperparameteroptimierung und Modellierung	83
7.2.1 Erarbeitung des Konzepts zur Hyperparameteroptimierung	83
7.2.2 Durchführung der Hyperparameteroptimierung	86
8 Auswertung der Ergebnisse	97
8.1 Erarbeitung des Evaluationskonzepts	97
8.1.1 Spezifizierung der Benchmark-Metriken	97
8.1.2 Spezifizierung der Laufzeitmessung	99
8.2 Gegenüberstellung der Metriken	100
8.3 Gesamtheitliche Betrachtung	104
9 Schlussbetrachtung	109
9.1 Zusammenfassung	109
9.2 Ausblick	110

A Algorithmen	113
B Tabelle der Grundgesamtheit an Object Detection-Algorithmen	115
C Trainingsverläufe der Benchmark-Verfahren	125
D Verlauf der k-Fold Cross Validation der Benchmark-Verfahren	129
E Detaillierte Metriken der Benchmark-Verfahren	135
Abbildungsverzeichnis	139
Tabellenverzeichnis	140
Algorithmenverzeichnis	143
Literaturverzeichnis	159
Eidesstattliche Versicherung	159

INHALTSVERZEICHNIS

Abkürzungsverzeichnis

ANN	Artificial Neural Network
AP	Average Precision
ASFF	Adatively Spatial Feature Fusion
ATSS	Adaptive Training Sample Selection
CNN	Convolutional Neural Network
CP	Condition Positive
CRISP-DM	CRoss Industry Standard Process for Data Mining
FCOS	Fully Convolutional One-Stage Object Detection
FLOPs	Floating Point Operations
FP	False Positive
FPN	Feature Pyramid Network
FPS	Frames Per Second
HTC	Hybrid Task Cascade
IOU	Intersection over Union
LR	Learning Rate
MS	Multi-Scale
NMS	Non Maximum Suppression

Abkürzungsverzeichnis

PF	Pixelfläche
PoE	Power over Ethernet
PR	Precision
R-CNN	Region-CNN
RC	Recall
ResNet	Residual Network
RFBNet	Receptive Field Block Net
ROI	Region of Interest
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SSD	Single Shot MultiBox Detector
TP	True Positive
VGG	Visual Geometry Group
YOLO	You Only Look Once

Kapitel 1

Vorbemerkungen

1.1 Motivation und Hintergrund

Prozesse in der Logistik werden fortwährend verbessert und auf neue Anforderungen angepasst. Parallel erhalten immer mehr Automatisierungs- und Robotiksysteme Einzug. Entscheidungen, wie und wo optimiert werden kann, beruhen auf der Analyse der bestehenden Produktionsabläufe. Businessverantwortliche der Intralogistik streben nach analytischen Methoden, auf deren Grundlage eine Vergrößerung der Transparenz von Lagerprozessen erzielt wird. Für die vorliegende Arbeit ist dies im Kontext eines Lagersystems der Kontraktlogistik zu betrachten. Notwendig für die Transparenz von Lagerprozessen ist die genaue Ermittlung des Warenflusses sowie der Bewegungsprofile der im Lager befindlichen Objekte. Damit ist ein Abgleich zwischen den im Lagerverwaltungssystem geplanten Prozessen und den tatsächlich abgewickelten Prozessen möglich. Bisherige Methoden setzen meist auf Beacon-basierte Lösungen [84]. Dabei muss aktiv in die bestehenden Lagerprozesse eingegriffen werden, um Mitarbeiter mit den notwendigen Geräten auszustatten. Eine Alternative bieten bilddatengestützte Algorithmen aus dem Forschungsfeld der Bildverarbeitung. Durch die leichtgewichtige Integration von Kameras in die bestehenden Lagersysteme ist ein großes Potential gegeben. Für die Ermittlung der Bewegungsprofile gilt es, die im Lager befindlichen Objekte über die Zeit zu verfolgen. Die resultierenden Pfade können anschließend in einem weiteren Schritt zur Analyse und Optimierung der Prozesse herangezogen werden. Methoden, die eine automatisierte Verfolgung von multiplen Objektinstanzen ermöglichen, werden im Kontext der Arbeit unter der englischen Bezeichnung *Object Tracking*-Algorithmen verwendet. Zur Realisierung der Verfolgung müssen die Objektinstanzen im Vorfeld automatisiert detektiert werden. Dies wird in der englischen Literatur mit *Object Detection* bezeichnet. Eine detaillierte Erläuterung der Funktionsweise und Basiseigenschaften der Algorithmen beider Themenfelder erfolgt im Verlauf der Arbeit. Da in diesem Zusammenhang die Genauigkeit der Verfolgung stark von der Detek-

tionsgenauigkeit und der damit verbundenen Laufzeit beeinflusst wird, besteht der Bedarf eines gezielten Vergleichs aktueller State-of-the-Art-Lösungen.

1.2 Zielsetzung und Abgrenzung

Die optimalen Ergebnisse erzielen Object Tracking-Algorithmen, wenn Bilder in einer hohen Frequenz verarbeitet und die darin enthalten Objektinstanzen mit einer hohen Genauigkeit detektiert werden. Dafür werden Object Detection-Algorithmen herangezogen, die seit einigen Jahren von *Deep Learning*-basierten Verfahren dominiert werden. Allgemein weisen diese einen Trade-Off zwischen der Detektionsgenauigkeit und der damit verbunden Laufzeit auf. Gegenstand der Arbeit ist es daher ein Benchmarking durchzuführen, welches die aktuellen Object Detection-Algorithmen bezüglich des Konfliktes zwischen Laufzeit und Genauigkeit vergleicht. Im Kontext der Arbeit ist dieser als Speed-Accuracy Trade-Off bezeichnet. Dafür gilt es, zunächst einen domänen spezifischen Datensatz aufzubauen, der die Szenarien aus den Lagerprozessen im Kontext der Intralogistik weitestgehend abdeckt. Die verwendeten Bilddaten werden im Rahmen eines Kooperationsprojekts zwischen dem Fraunhofer-Institut für Materialfluss und Logistik IML [31] und zwei weiteren Projekt-partnern erhoben. Auf deren Grundlage ist eine Modellierung für eine ausgewählte Menge von Verfahren durchzuführen. Die Vorauswahl hat so zu erfolgen, dass der Speed-Accuracy Trade-Off mit entsprechenden Verfahren abgedeckt ist. Anhand der aus dem Benchmark gewonnenen Erkenntnisse wird eine Entscheidungsbasis zur Verfügung gestellt, aus der die Wahl eines für Object Tracking geeigneten Verfahrens zur Detektion von Objektinstanzen hervorgeht. Durch die Ermittlung der bestehenden State-of-the-Art-Lösungen für Object Detection-Algorithmen wird sicher gestellt, dass auf Grundlage der Entscheidungsbasis eine Einschätzung des Potentials einer bilddatengestützten Analyse von Lagerprozessen der Intralogistik gegeben wird, welche dem aktuellen Stand der Forschung entspricht.

1.3 Aufbau der Arbeit

Ein Großteil der zugrunde liegenden Literatur für die Arbeit liegt in englischer Sprache vor. Daher werden nach der Einführung von Fachbegriffen die englischen Bezeichnungen verwendet. Eine Ausnahme bilden Bezeichner, für die sich im deutschsprachigen Raum eine feste Definition durchgesetzt hat. Hier sind unter anderem die *Klassifikation* und die *Bildverarbeitung* aufzuführen.

Zu Beginn wird in Kapitel 2 geschildert, unter welchen Anforderungen und Rahmenbedingungen das Benchmarking durchzuführen ist. Dafür erfolgt zunächst eine detaillierte Betrachtung der Eigenschaften, die ein Object Detection-Algorithmus zur Kopplung an ein Object Tracking-Algorithmen zu erfüllen hat. Darauf folgt die Beschreibung der verwendeten Datenquelle sowie der eingesetzten Hard- und Software. Anschließend werden

in Kapitel 3 die Grundlagen der Deep Learning-basierten Methoden zur Object Detection geschildert. Hierfür werden in einer Themeneinordnung die Begrifflichkeiten Artificial Intelligence, Machine Learning und Deep Learning differenziert sowie die Object Detection in den Kontext anderer Verfahren des Forschungsfeldes der Bildverarbeitung gestellt. Aufbauend darauf werden die, für die Arbeit, notwendigen Konzepte des Machine Learning und des Deep Learning vorgestellt. Abschließend wird auf die generelle Architektur von Deep Learning-basierten Object Detection-Algorithmen eingegangen und Evaluierungsmaßnahmen aufgezeigt, die eine Bewertung der Genauigkeit von ermittelten Detektionen zulassen. Danach wird die Methodik und der Aufbau des Datensatzes in Kapitel 4 erläutert, auf deren Grundlage die Modellierungen der Algorithmen im Benchmark beruhen. Zur Ermittlung der State-of-the-Art Object Detection-Algorithmen wird in Kapitel 5 eine Grundgesamtheit aufgebaut. Diese bildet den Ausgangspunkt für die Selektion von Verfahren, die durch Anwendung von spezifizierten Filterregeln für das Benchmark ausgewählt werden. Darauf folgt in Kapitel 6 die Erläuterung der algorithmischen Konzepte der selektierten Verfahren. In Kapitel 7 findet die Modellierung der Verfahren unter Anwendung einer Hyperparameteroptimierung statt. Auf deren Grundlage wird eine k-Fold Cross Validation durchgeführt, deren Ergebnis in Kapitel 8 diskutiert wird. Abschließend erfolgt eine zusammenfassende Schlussbetrachtung in Kapitel 9, die insbesondere einen Ausblick auf die gewonnenen Erkenntnisse für die Intralogistik eingeht. Zudem wird aufgezeigt, welche weiteren Schritte notwendig sind, um Object Detection-Algorithmen erfolgreich in einem Produktivbetrieb einzusetzen.

Kapitel 2

Anforderungsanalyse

Im folgenden Kapitel werden zunächst die Anforderungen formuliert, welche an die eingesetzten Object Detection-Algorithmen im Benchmark gestellt werden. Dabei wird insbesondere darauf eingegangen, welche Eigenschaften gefordert werden, um eine erfolgreiche Integration in einen Object Tracking-Algorithmus zu erlauben. Anschließend werden die Rahmenbedingungen aufgezeigt, unter denen die Implementierung und Modellierung der Object Detection-Algorithmen im Benchmark statt findet. Zunächst erfolgt die Deklaration der Bilddatenquelle, anschließend werden die eingesetzte Hard- und Software spezifiziert. Abschließend werden die ermittelten Anforderungen und Rahmenbedingungen zusammenfassend festgehalten.

2.1 Formulierung der Anforderungen an das Benchmark

Wie eingangs in Kapitel 1 erläutert, zielt die vorliegende Arbeit darauf ab, ein Benchmarking durchzuführen, welches aktuelle State-of-the-Art Deep Learning-basierte Object Detection-Algorithmen vergleicht. Dabei ist insbesondere auf den Trade-Off zwischen der Detektionsgenauigkeit und der damit verbunden Laufzeit einzugehen. Hintergrund ist die Bereitstellung einer fundierten Entscheidungsbasis zur Auswahl eines für Object Tracking geeigneten Verfahrens. Das gekoppelte System aus Object Detection und Object Tracking ist die Basis, um eine bilddatengeschützte Prozessanalyse im Kontext der Intralogistik zu ermöglichen.

Object Detection-Algorithmen realisieren eine automatisierte Klassifizierung und Lokalisierung von Instanzen vordefinierter Objektklassen in digitalen Bildern in Form von Rechtecken, in der Literatur als Bounding-Box bezeichnet. Abbildung 2.1 illustriert dieses Vorgehen am Beispiel einer automatisierten Detektion von fahrerlosen Transportsystemen (engl. Automated Guided Vehicle).

Für eine Prozessanalyse müssen die erkannten Objektinstanzen über die Zeit verfolgt werden. Dies ist realisiert durch die Zuordnung einer ID zu jeder Instanz eines erkannt-

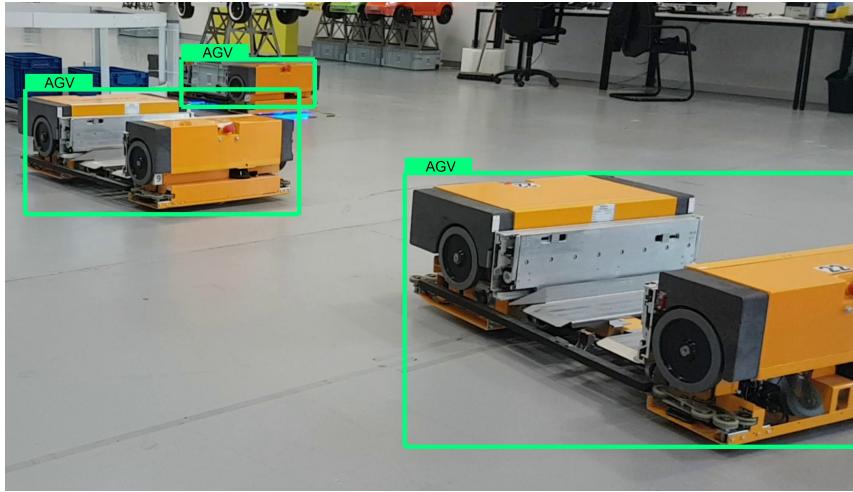


Abbildung 2.1: Visualisierung der Ausgabe eines Object Detection-Algorithmus. Entnommen aus [30].

ten Objektes. Die ID ist für einen gegebenen Datenstrom aus Bildern eindeutig. Object Tracking-Algorithmen lösen diese Problemstellung. Genauer lässt sich die Vorgehensweise formulieren als: Zuweisung der gleichen Tracking-ID t zu einer Bounding-Box $b_{k,n}$ und Bounding-Box $b_{m,n+1}$ in zwei aufeinander folgenden Bildern i_n und i_{n+1} , unter der Bedingung, dass $b_{k,n}$ und $b_{m,n+1}$ zur gleichen Instanz eines Objektes gehören. Dabei gilt $b_{k,n} \in B_n$ und $b_{m,n+1} \in B_{n+1}$ mit B_n und B_{n+1} als Mengen aller Bounding-Boxen der Bilder i_n und i_{n+1} , sowie $i_n, i_{n+1} \in I$ mit I als Menge aller Bilder eines gegebenen Datenstroms. Zudem gilt $t \in T$ als Menge aller Tracking-IDs, sowie $k \in \{1, \dots, |B_n|\}$, $m \in \{1, \dots, |B_{n+1}|\}$ und $n \in \{1, \dots, |I|\}$.

Aus der Zuordnung der Tracking-ID zu einer Objektinstanz (einer Bounding-Box) kann in einem weiteren Schritt ein eindeutiger Pfad bestimmt und auf Grundlage dessen ein Bewegungsprofil für jedes in der Lagerhalle befindliche Objekt hergeleitet werden. Basierend auf den Bewegungsprofilen der Objekte lassen sich wichtige Kennzahlen zur Bestimmung der Wirtschaftlichkeit der im Lager ablaufenden Prozesse berechnen. Die Güte dieser Prozessanalysen ist gebunden an die Genauigkeit der Bestimmung des Bewegungsprofils, beruhend auf den Ergebnissen des Object Tracking-Algorithmus. Die Genauigkeit des Object Tracking-Algorithmus ist wiederum maßgeblich an zwei Faktoren gebunden:

F1: Dem Abstand δ zwischen den Positionen $p_{k,n}$ und $p_{m,n+1}$, der Instanz mit Tracking-ID t , welche durch die Bounding-Boxen $b_{k,n}$ und $b_{m,n+1}$ repräsentiert sind:

$$\delta(P_{k,n}, P_{m,n+1}) = |P_{m,n+1} - P_{k,n}| \quad (2.1)$$

F2: Der Genauigkeit der Detektion der Objektinstanzen anhand ihrer Bounding-Box, die von einem Object Detection-Algorithmus ermittelt wird

Der Faktor $F1$ wird stark von der Laufzeit des zugrunde liegenden Object Detection-Algorithmus beeinflusst. Je kleiner dessen Laufzeit ist, desto höher ist die Frequenz, mit der der Object Detection-Algorithmus Bilder aus einem gegebenen Datenstrom verarbeiten kann. Potentiell gilt: Je höher die Bildfrequenz, desto kleiner ist das δ zwischen den Positionen $p_{k,n}$ und $p_{m,n+1}$ einer Instanz. Dies beruht auf dem verringerten Zeitintervall, in dem eine Bewegung der Instanz erfolgen kann. Für die Berechnungsgrundlagen des Object Tracking-Algorithmus ergibt sich für ein kleineres δ eine erleichterte Zuordnung der Tracking-ID. Dies resultiert wiederum in einer Verbesserung der Tracking-Performance.

Aktuelle State-of-the-Art Algorithmen der Bildverarbeitung basieren auf Konzepten des Deep Learning[56]. Daher werden Deep Learning-basierte Object Detection-Algorithmen eingesetzt, um die für das Object Tracking notwendigen Bounding-Boxen bereit zu stellen. Ein für die geschilderte Problematik wichtiger Einflussfaktor von Object Detection-Algorithmen ist der Konflikt zwischen der Detektionsgenauigkeit von Bounding-Boxen und der damit verbundenen Laufzeit: Hohe Laufzeit \rightarrow hohe Genauigkeit vs niedrige Laufzeit \rightarrow niedrige Genauigkeit. Dieser Konflikt wurde bereits näher in [52] untersucht und dort unter der Bezeichnung Speed-Accuracy Trade-Off aufgeführt. Diese wird nachfolgend zur Beschreibung des Konflikts verwendet. Er prägt sich in einem konkurrierendem Optimierungsproblem aus, welches einen direkten Effekt auf den Tracking-Algorithmus hat. Für das Tracking wird der Konflikt formuliert als: Hohe Bildfrequenz \rightarrow niedriges δ (hohe Tracking-Performance) vs niedrige Bildfrequenz \rightarrow hohes δ (niedrige Tracking-Performance).

Dies motiviert die Erstellung einer fundierten Entscheidungsbasis zur Auswahl eines für Object Tracking geeigneten Verfahrens. Es werden aktuelle State-of-the-Art Object Detection-Algorithmen bezüglich des Speed-Accuracy Trade-Off untersucht. Da die genauen Anforderungen an die Bildrate des Object Tracking-Algorithmus nicht bekannt sind, werden Object Detection-Algorithmen aus verschiedenen Laufzeit-Gruppen betrachtet. Eine detaillierte Beschreibung der zur Auswahl stehenden Architekturen, eine Betrachtung und Erarbeitung weiterer Einflussfaktoren auf den Speed-Accuracy Trade-Off, sowie die Definition der zur Bewertung herangezogenen Metriken, mit anschließender Auswertung, findet in den Kapiteln 5 und 8 statt.

2.2 Rahmenbedingungen

In diesem Kapitel werden zunächst die Kameras vorgestellt, die als Quelle der Bilddaten verwendet wird. Darauf folgt die Beschreibung der Hardware, welche für die Ausführung der Object Detection-Algorithmen eingesetzt wird. Zuletzt wird auf die aktuellen Frameworks eingegangen, die für die Implementierung von Object Detection-Algorithmen zur Verfügung stehen.

2.2.1 Datenquelle

Die für das Benchmark verwendeten Bilddaten sind beschränkt auf Prozessabläufe in Regalgängen eines Lagers der Intralogistik. Die Bilddaten sind im Rahmen eines Projektes vom Fraunhofer IML und zwei externen Unternehmen erhoben worden, dürfen aufgrund einer Geheimhaltungsvereinbarung (engl. non-disclosure agreement (NDA)) zwischen allen Projektpartnern jedoch nicht in der Arbeit veröffentlicht werden. In Kapitel 4.1 werden daher Beispielbilder der Original-Szenarien aufgezeigt, die aus einer Forschungshalle des Fraunhofer IML stammen. Die im Benchmark ermittelten Ergebnisse beruhen jedoch auf den originalen Bilddaten aus dem Projekt.

Für die Durchführung der bilddatengestützten Analyse werden Prozessdaten aus zwei Regalgängen betrachtet. Pro Regalgang sind jeweils zwei Kameras installiert (siehe Abbildung 2.2). Somit können die Prozesse aus beiden Blickwinkeln (Anfang und Ende des Regalgangs) analysiert werden. Für die Aufnahme der Bilder werden Industriekameras eingesetzt, die für einen Dauerbetrieb in der Lagerhalle ausgelegt sind. Im Projekt kommt das Kamera-Modell *Basler acA1440-73gc* in Kombination mit dem Objektiv *Basler Lens C125-0818-5M-P f8mm* zum Einsatz. Die exakten Spezifikationen beider Komponenten sind in [6, 7] aufgeführt. Ein wichtiger Bestandteil ist eine Power over Ethernet (PoE)-Schnittstelle, welche es ermöglicht, die Kameras simultan mit Strom zu versorgen und die Bilddaten an die im nachfolgenden Unterkapitel vorgestellte Workstation zu übertragen. Die Kamera liefert standardmäßig Bilder in der Auflösung 1440×1080 Pixel, welche mit einem vom Hersteller zur Verfügung gestellten Python-SDK mit maximal 73 *Frames Per Second (FPS)* ausgelesen werden können. Jede Kamera ist über einen eigenen Port an die Workstation angebunden, womit eine Bandbreite von $1GBit/s$ garantiert wird. Insgesamt gilt es, vier Bilddatenströme zu verarbeiten. Hierfür wurde ein als Multithreading organisiertes Softwaremodul entworfen, welches eine parallele Verarbeitung aller Bilddatenströme sicherstellt. Dieses wird sowohl zur Aufnahme der Bilder der Benchmark-Datensätze (siehe Kapitel 4.2), als auch zur Ausführung der Object Detection-Algorithmen eingesetzt.

2.2.2 Spezifikation der verwendeten Hardware

Im Rahmen des maschinellen Lernens (engl. Machine Learning) bezeichnet das *Training* den Prozess zur Erstellung eines algorithmischen Modells [33] und die *Inferenz* das Schlussfolgern basierend auf einem bereits trainierten Modell [33, 25], demnach die Ausführung eines Modells. Für die Inferenz von Deep Learning-Algorithmen im Kontext der Bildverarbeitung gibt es verschiedene Herangehensweisen. Diese sind in Abbildung 2.3 veranschaulicht und lassen zwei Bereiche einteilen: Dezentrale und zentrale Inferenz. Bei der dezentralen Inferenz werden die Algorithmen direkt auf dem Gerät (weiße Kamera) selbst ausgeführt (Microcontroller oder Embedded Devices) und nur das Ergebnis an eine zentrale Instanz (Rechteck) weitergeleitet. Hingegen werden bei einem zentralen Ansatz die Daten an ei-

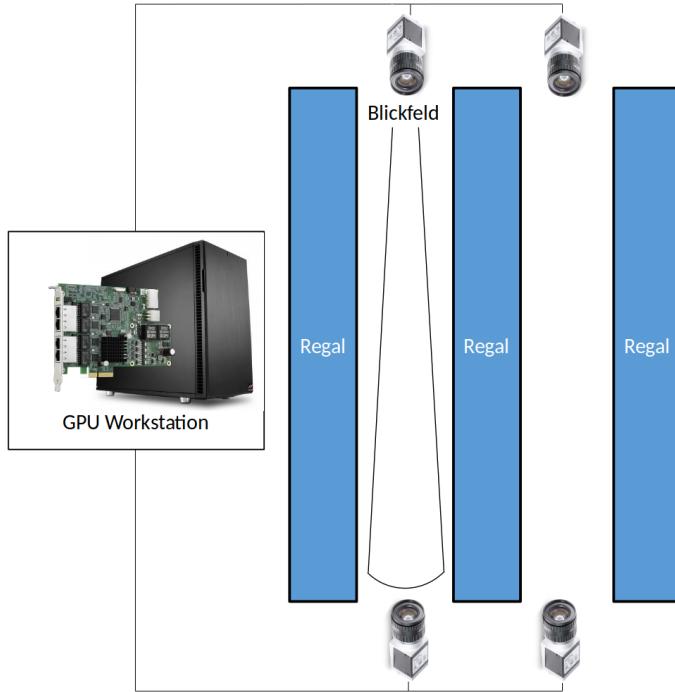


Abbildung 2.2: Hardware Setup in der Lagerhalle. Bilder entnommen aus [134, 78, 2].

ne zentrale Instanz übermittelt und die Berechnung damit auf Workstations oder Server ausgelagert. Je nach Rechenleistung der Verfahren können unterschiedliche Daten verarbeitet werden. Die Verwendung von GPUs bietet in diesem Kontext einen großen Zuwachs an Rechenleistung. Im Rahmen des Projekts bei Fraunhofer IML kommt eine Workstation zum Einsatz, die als zentrale Instanz fungiert. Ausgestattet mit einer *NVIDIA Geforce RTX 2080 Ti* GPU, können die Deep Learning basierten Object Detection-Algorithmen performant ausgeführt werden. Grundlage sind die von NVIDIA bereitgestellten Frameworks CUDA [89] und cuDNN [88].

Über eine in die Workstation integrierte PCI Express Karte mit der Bezeichnung *AD-LINK's PCIe-GIE74 GigE Vision PoE+* werden die zuvor beschriebenen Kameras an die Workstation angebunden. Diese verfügt über vier Netzwerkports mit PoE Technologie. Damit ist das simultane Übertragen von Bildern und die Stromversorgung gewährleistet. Abbildung 2.2 veranschaulicht den schematischen Aufbau des Gesamtsystems in der Lagerhalle.

2.2.3 Spezifikation der verwendeten Software

Zum Aufbau der fundierten Entscheidungsbasis gilt es, aktuelle State-of-the-Art Object Detection-Algorithmen bezüglich des Speed-Accuracy Trade-Off zu vergleichen. Da der Vergleich und nicht die Implementierung im Vordergrund steht, wird für die Anwendung eines Object Detection-Algorithmus im Benchmark die Verfügbarkeit einer Open-Source Imple-

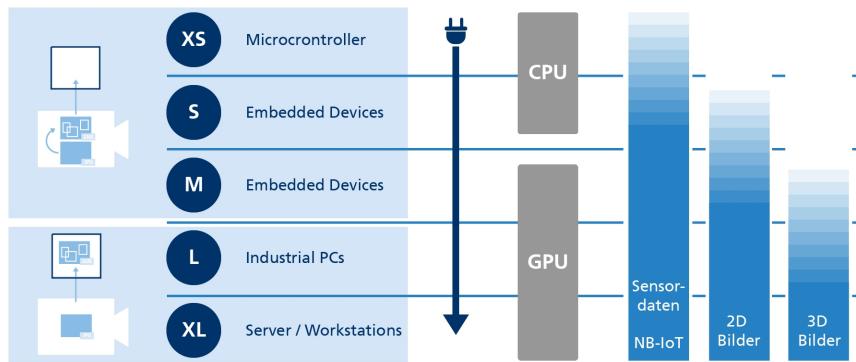


Abbildung 2.3: Dezentrale vs zentrale Inferenz auf unterschiedlichen Plattformen. Bearbeitete Version aus [147]

mentierung vorausgesetzt. Unterschiedliche Verfahren verwenden unterschiedliche Open-Source Deep Learning Frameworks. Aktuell wird vorwiegend die Programmiersprache Python genutzt. Aus Analysen in [45, 44] gehen Tensorflow [1] und Pytorch [100] als populärste Deep Learning Frameworks hervor. Diese sind hauptsächlich für die Programmiersprache Python ausgelegt, bieten jedoch auch Schnittstellen zu anderen Programmiersprachen. Weitere Deep Learning Frameworks sind unter anderem Caffe [55] (dessen Nachfolgeversion in Pytorch integriert wurde), MXNet [136], Theano [71] und ONNX [137]. Ein populärer Aufsatz auf Tensorflow ist Keras [21]. In Kapitel 5.3 wird die Auswahl der im Benchmark zu vergleichenden Object Detection-Algorithmen getroffen. Um einen einheitlichen Vergleich der Algorithmen im Benchmark zu garantieren wird, gekoppelt an die Auswahl, ein einheitlicher Rahmen an Frameworks bestimmt. Da die Inferenz auf einer GPU eine Verwendung der Frameworks CUDA und cuDNN voraussetzt, sind diese in die Spezifikation eines gesamtheitlichen Deep Learning Frameworks für das Benchmark mit einzubeziehen.

2.3 Zusammenfassung der Anforderungen und Rahmenbedingungen

Basierend auf den in Kapitel 2.1 und 2.2 geschilderten Anforderungen und Rahmenbedingungen, werden die Grundanforderungen an die im Benchmark zu vergleichenden Object Detection-Algorithmen in Tabelle 2.1 zusammengefasst. In den folgenden Kapiteln werden zunächst alle notwendigen theoretischen Grundlagen in Kapitel 3 erläutert. Anschließend wird in Kapitel 4 eine Leitlinie zur algorithmischen Modellierung vorgestellt und geschildert, wie diese im Kontext der Arbeit angewandt wird. Auf Grundlage des so erlangten Hintergrundwissens, werden die in Tabelle 2.1 aufgeführten Anforderungen konkretisiert.

2.3. ZUSAMMENFASSUNG DER ANFORDERUNGEN UND RAHMENBEDINGUNGEN

Anschließend findet, unter Einhaltung der final definierten Anforderungen, eine Selektion von Object Detection-Algorithmen statt, die es in Benchmark zu betrachten gilt.

Nr	Beschreibung
1	Hohe Detektionsgenauigkeit auf Basis von zu definierenden Metriken
2	Einhaltung von zu definierenden Laufzeitbestimmungen
3	Verfügbarkeit einer Open-Source Implementierung

Tabelle 2.1: Grundanforderungen an das Benchmarking.

Kapitel 3

Grundlagen

In Kapitel 1 wird der Einsatz von Bilddaten zur Analyse von Lagerprozessen im Kontext der Intralogistik motiviert. Diese bieten das Potential, eine höhere Transparenz der Arbeitsabläufe zu schaffen. Grundlage dafür sind Algorithmen aus dem Bereich der Bildverarbeitung, die eine automatisierte Detektion von Objekten ermöglichen. Diese sind spezifiziert als Object Detection-Algorithmen. Seit 2014 werden diese von Verfahren aus dem Forschungsfeld des Deep Learning dominiert [160, 166].

Zu Beginn des Kapitels findet eine Einordnung der, im Kontext von Deep Learning stehenden Themenfelder, sowie eine Gegenüberstellung Deep Learning basierter Algorithmen der Bildverarbeitung statt. Darauf folgen Kapitel zur Erläuterung der Grundlagen für Deep Learning basierte Object Detection-Algorithmen. Hierfür werden die jeweiligen Konzepte aus den Themengebieten Machine Learning und Deep Learning aufgeführt. Danach erfolgt eine Erläuterung der verschiedenen Herangehensweisen bei der Architektur von Object Detection-Algorithmen. Zuletzt wird auf die Metriken eingegangen, welche für die Evaluation aktueller State-of-the-Art Verfahren herangezogen werden.

Der Fokus dieses Kapitels liegt auf der Schilderung grundlegender Prinzipien und Methodiken, um ein einheitliches Verständnis zur Thematik von Deep Learning basierten Object Detection-Algorithmen zu schaffen. Für tiefergehende theoretische Konzepte wird jeweils auf die Originalliteratur verwiesen.

3.1 Themeneinordnung

Nachfolgend findet zunächst eine Abgrenzung der Begrifflichkeiten *Artificial Intelligence*, *Machine Learning* und *Deep Learning* statt. Anschließend werden die für die Arbeit relevanten Algorithmen aus dem Bereich der Bildverarbeitung differenziert.

3.1.1 Begrifflichkeiten der Artificial Intelligence

Deep Learning ist eine Teildisziplin des Forschungsfeldes des Maschinellen Lernens (engl. Machine Learning), welche wiederum eine Teildisziplin der Künstlichen Intelligenz (engl. Artificial Intelligence) ist. Abbildung 3.1 illustriert die Abgrenzung der Begrifflichkeiten. Artificial Intelligence befasst sich mit der Entwicklung intelligenter Entitäten [117], insbesondere Computersystemen, die menschliche Intelligenz anhand von Lernmethoden, schlussfolgerndem Denken und Selbstadaptation reproduzieren [59]. Eine allgemeine Definition für Artificial Intelligence kann jedoch nicht formuliert werden. Historisch sind diesbezüglich verschiedene Ansätze verfolgt worden, die sich nach [117] in die vier Kategorien *Think Like Humans*, *Act Like Humans*, *Think Rationally* und *Act Rationally* gliedern lassen. Ein Ansatz zur Definition von Artificial Intelligence, im Zusammenhang mit menschlichem Handeln, wird mit dem Turing Test [141] assoziiert. Ein Computer besteht den Test, wenn ein menschlicher Vermittler nach dem stellen verschiedener Fragen nicht sagen kann, ob die Antworten von einer Person oder einem Computer gegeben worden sind.

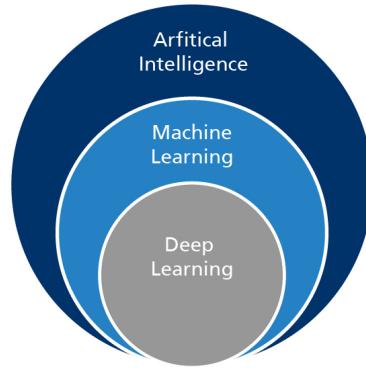


Abbildung 3.1: Abgrenzung künstliche Intelligenz und Deep Learning. Bearbeitete Version aus [147].

Zur erfolgreichen Entwicklung einer computergestützten künstlichen Intelligenz, welche diese Anforderung erfüllt, sind nach [117] Methoden aus unterschiedlichen wissenschaftlichen Disziplinen notwendig, unter anderem Methoden des Machine Learning. Dieses Themenfeld behandelt die Entwicklung von Algorithmen, die sich automatisiert auf Basis von Erfahrung verbessern [80] und so adaptiv auf neue Begebenheiten angepasst werden können, um Muster zu detektieren und zu extrapolieren [59]. Ferner lieferte Tom Mitchell 1997 die Definition: „*A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E*“ [81]. Die Erfahrung wird dabei über so genannte Trainingsdaten oder auch *Ground-Truth-Daten* abgebildet und der Lernprozess als Training bezeichnet [34]. Machine Learning-Algorithmen werden nach dem Grad an Supervision, der während des Trainings notwendig ist, in die Teilbereiche Supervised Learning, Unsu-

pervised Learning, Semi-Supervised Learning und Reinforcement Learning aufgeteilt [34]. Jeder Teilbereich behandelt wiederum verschiedene Klassen von algorithmischen Problemstellungen. Das Konzept des Deep Learning ist in allen Teilbereichen vertreten. Nach [62] lässt sich Deep Learning wie folgt definieren: „*Deep learning allows computational models that are composed of multiple processing Layer to learn representations of data with multiple levels of abstraction.*“ Bei Deep Learning-basierten Object Detection-Algorithmen handelt es sich um Algorithmen, die nach dem Prinzip des Supervised Learning vorgehen. Für Supervised Learning-Algorithmen sind in den Trainingsdaten, neben den Eingabedaten des entsprechenden Tasks, die zu erwartenden Lösungen zu inkludieren. Die Lösungen werden als Annotation bezeichnet [34]. Wie in Kapitel 2.1 beschrieben sind im Fall der Object Detection aus gegebenen Eingabebildern Instanzen vordefinierter Objektklassen zu extrahieren. Auf die genaue Funktionsweise wird in Kapitel 3.4 eingegangen. Eine detaillierte Beschreibung des Formats der Trainingsdaten für Object Detection-Algorithmen wird in Kapitel 4.2 gegeben.

3.1.2 Deep Learning-Algorithmen der Bildverarbeitung

Für die Arbeit gilt es zwischen den folgenden Typen von Deep Learning-basierten Algorithmen der Bildverarbeitung zu differenzieren: *Klassifikation* (engl. Classification), *Objektdetektion* (engl. Object Detection), *Instanzsegmentierung* (engl. Instance Segmentation) und *Semantische Segmentierung* (engl. Semantic Segmentation). Da sich die deutschen Begriffe Instanzsegmentierung und Semantische Segmentierung etabliert haben, werden diese im Kontext der Arbeit verwendet. Jeder der Algorithmen erhält als Eingabe ein digitales Bild. Bei der Klassifikation wird dem gegebenen Eingabebild ein Klassenlabel aus einer vorgegeben Menge von Klassenlabels zugeordnet. Die Object Detection befasst sich mit der Lokalisierung und Klassifizierung aller Instanzen der vorgegeben Objektklassen in Form von Rechtecken. Algorithmen, die eine Semantische Segmentierung realisieren, ermöglichen eine Zuordnung von einem Klassenlabel zu jedem Pixel. Algorithmen der Instanzsegmentierung sind eine Kombination aus Object Detection und semantischer Segmentierung. Anstelle von Rechtecken gilt es, Polygone zu ermitteln, aus der eine pixelgenaue Zuordnung der Klassenlabel für die resultierende Maske hervorgeht [77]. Abbildung 3.2 stellt die Resultate der verschiedenen Algorithmen gegenüber. Die farblichen Markierungen kennzeichnen, was vom jeweiligen Algorithmus erkannt wird. Zudem sind die ermittelten Klassenlabel angegeben.

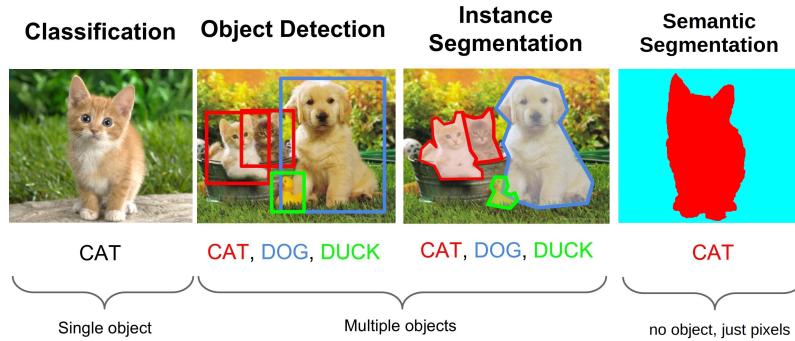


Abbildung 3.2: Gegenüberstellung Deep Learning-basierter Algorithmen der Bildverarbeitung. Bearbeitete Version aus [130, 94].

3.2 Grundlagen Machine Learning

Dieses Kapitel dient der Erläuterung der für Deep Learning notwendigen Grundlagen aus der Obermenge von Methoden des Machine Learning. Zunächst werden alle relevanten Konzepte aus dem Themenfeld der künstlichen Neuronalen Netze (engl. Artificial Neural Networks) vorgestellt. Anschließend erfolgt die Erläuterung einer von Machine Learning-Algorithmen geforderten wichtigen Eigenschaft: Die *Generalisierung*.

3.2.1 Artifical Neural Networks

Artifical Neural Networks bilden die Grundlage für alle Deep Learning basierten Algorithmen [41]. Nachfolgend wird zunächst deren generelle Architektur geschildert. Darauf folgt die Spezifikation wichtiger Hyperparameter. Abschließend wird das Konzept des Transfer Learning geschildert.

Architektur von Artifical Neural Networks

Das *Artificial Neural Network (ANN)* ist ein mathematisches Konstrukt, welches den elektrochemischen Vorgängen zwischen den in einem Netzwerk verbunden Gehirnzellen, den Neuronen, nachempfunden sind [117, 101]. Das Berechnungsmodell eines einzelnen künstlichen Neurons (engl. *artificial neuron*) ist in Abbildung 3.3 veranschaulicht. Es ist definiert über eine Transfer- und eine Aktivierungsfunktion. Die Transferfunktion realisiert eine Linearkombination der eingehenden Verbindungen x_i und ihren Gewichten w_{ij} . Als Analogie lassen sich die gewichteten Verbindungen mit einem Langzeit-Informationsspeicher vergleichen [101]. Zudem spezifiziert die Größe der Gewichte die Stärke der Verbindung zwischen den Neuronen. Die Ausgabe der Transferfunktion dient in einem weiteren Schritt als Eingabe der Aktivierungsfunktion. Hierbei handelt es sich um nicht lineare Funktionen, welche eine entscheidende Basiseigenschaft eines künstlichen neuronalen Netzwerks sicherstellt: Die Approximation beliebiger nicht linearer Funktionen. Um im Falle von niedrigen

Eingabewerten die Aktivierung von mindestens weniger Neuronen zu garantieren, wird dem Produkt der Transferfunktion ein additiver Term, der Bias b_i , hinzugefügt [101]. Zum Aufbau von ANN-Architekturen werden einzelne Neuronen in einem Verbund in einzelnen Schichten (engl. Layer) angeordnet, die miteinander verbunden sind. Die Ausgabewerte der Aktivierungsfunktion sind in diesem Fall die Eingabewerte der nachfolgenden Layer. In einem ANN bestimmt die Art der Verbindung und die Anzahl der Neuronen und Layer dessen Verhalten.

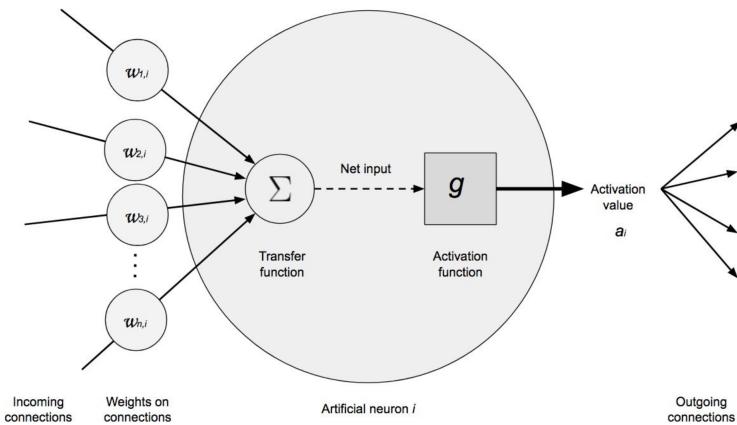


Abbildung 3.3: Berechnungsmodell eines einzelnen künstlichen Neurons. Entnommen aus [101].

Die einfachste Form eines ANN ist das Feedforward Multilayer Neural Network. Es besteht aus einem Input Layer, mehreren Hidden Layern und einem Output Layer [117, 101], welche in einem azyklischen Graph verbunden sind. Abbildung 3.4 illustriert den geschilderten Aufbau. Der Input Layer erhält die Eingabedaten, welche anschließend an die Hidden Layer weiter propagiert werden. Abschließend wird auf Basis des Output Layers das Ergebnis zur Verfügung gestellt. Jedes Neuron eines Layers ist dabei mit jedem Neuron des vorherigen Layers verbunden. In der Literatur wird dies als *Fully-Connected* Layer bezeichnet. Die Neuronen der einzelnen Layer erhalten als Eingabe lediglich die Ausgabe der Aktivierungsfunktionen des vorherigen Layers, was unter die Begrifflichkeit der Forward Propagation fällt. Zur Realisierung der Approximation einer definierten Zielfunktion (Ausgabe des Netzwerks) gilt es, die Gewichte und Bias der Neuronen im Netzwerk anzupassen. Hierfür wird der Lernalgorithmus *Backpropagation* [115] angewendet. Dazu gilt es eine Funktion zu definieren, die den Fehler (engl. loss) zwischen der Ausgabe des Netzwerks und der Abweichung von der zur Eingabe gehörigen Annotation angibt. Diese Funktion wird in der Literatur als *Loss-Funktion* bezeichnet. Der Backpropagation-Algorithmus realisiert anschließend unter Anwendung eines Gradientenabstiegs eine Minimierung der Loss-Funktion. Basierend auf dem Gradienten der Loss-Funktion werden die Gewichte w_{ij} und Bias b_i schrittweise entlang des Gradienten angepasst. Eine detaillierte mathematische Herleitung ist in [101] zu finden.

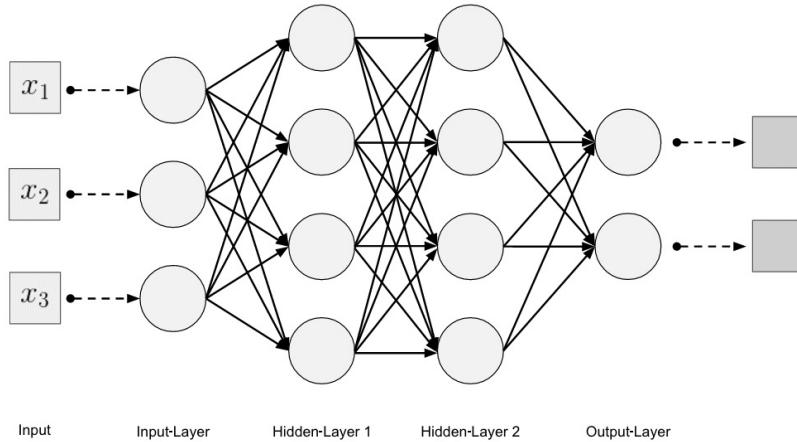


Abbildung 3.4: Architektur eines ANN. Bearbeitete Version aus [101].

Hyperparameter für das Training von Artifical Neural Networks

Der wichtigste für das Training neuronaler Netze relevante Hyperparameter ist die *Learning Rate (LR)*. Sie bestimmt, wie stark die Gewichte und Bias pro Iteration des Backpropagation-Algorithmus angepasst werden [101]. Bei der Wahl des Wertes für die LR gilt es folgende Eigenschaften zu berücksichtigen: Ein zu großer Wert erlaubt zwar eine schnellere Optimierung, kann jedoch dazu führen, ein lokales Minimum zu überspringen und damit Instabilitäten im Trainingsprozess hervorzurufen. Ein zu kleiner Wert führt zu ineffektiven Trainingsprozessen, aufgrund der damit verbundenen Steigerung der Trainingszeit [101]. Idealerweise wird zu Beginn des Trainings ein großer Wert gewählt, welcher auf Basis des Verlaufs des Optimierungsprozesses stetig bis hin zur Konvergenz der Loss-Funktion verringert wird. Für die Anwendung der Backpropagation stehen mehrere Methoden zur Verfügung, die als Optimierer bezeichnet werden. Eine der ersten und meist verwendeten Optimierer im Bereich der Object Detection ist *Stochastic Gradient Descent (SGD)* [57]. Gekoppelt an den Optimierer wird ein Schedulingverfahren ausgewählt. Dieses passt den Wert der LR auf Grundlage der Anzahl der *Epochen* an, die das Training bereits durchlaufen hat. Eine Epoche entspricht der Iteration über alle Beispiele im Trainingsdatensatz. Zudem wird von Schedulingverfahren zu Beginn eines Trainings oft das Konzept der *Warm-Up-Epochen* angewandt. Hierfür wird die LR für die Anzahl an Warm-Up Epochen auf einen kleinen Wert gesetzt, um eine feinjustierte Grundinitialisierung der Gewichte vorzunehmen, ehe mit dem eigentlichen Scheduling der LR fortgefahrene wird. Die Trainingsbeispiele, die für einen einzelnen Trainingsschritt innerhalb einer Epoche verwendet werden, werden als Batch bezeichnet, die genaue Anzahl der Trainingsbeispiele über die *Batch Size* spezifiziert. Der Wert des Gradienten, der vom Optimierer zur Anpassung der Gewichte verwendet wird, ergibt sich als Mittel über die Gradienten für die Trainingsbeispiele aus einer Batch. Des Weiteren hängt von der Batch Size die Anzahl der Trainings-

schritte pro Epoche ab, da pro Epoche jedes Trainingsbeispiel nur einmal verwendet wird. Demnach ergeben sich aus einer größeren Batch Size weniger Trainingsschritte pro Epoche und umgekehrt.

In Kapitel 6 findet eine detaillierte Beschreibung der algorithmischen Prinzipien der für das Benchmarking ausgewählten Object Detection-Algorithmen statt. Im Zuge dessen wird geschildert, welche Hyperparameter diese verwenden. Dabei werden insbesondere die eingesetzten LR-Schedulingverfahren näher erläutert.

Transfer Learning im Kontext von Artifical Neural Networks

Nach [90] wird *Transfer Learning* definiert als: Verbesserung des Lernprozesses eines neuen Tasks, basierend auf transferiertem Wissen eines in Relation stehenden, bereits gelernten Tasks. Im Kontext der ANNs wird Transfer Learning angewendet, um zu Beginn eines Lernprozesses eine Initialisierung der Gewichte auf Basis einer zuvor trainierten ANN-Architektur vorzunehmen [5]. Damit verfügt die neue ANN-Architektur von Beginn an über Operatoren, die zur Realisierung des gegebenen Tasks eingesetzt werden können. Je näher die Domänen beider Architekturen in Relation zueinander stehen, desto mehr profitiert die neue Architektur von dem Einsatz bereits gelernter Operatoren. Zudem wird der Lernprozess beschleunigt, da weniger Trainingsbeispiele zum Anlernen des neuen Tasks notwendig sind [95].

3.2.2 Generalisierung und Overfitting

Ein zentrale Eigenschaft, die bei der Modellierung von Machine Learning-Algorithmen gefordert wird, ist eine gute Performance auf Daten, die nicht Teil der Trainingsbeispiele waren. Diese Eigenschaft wird als *Generalisierung* definiert [62]. Um die Generalisierung eines Modells zu messen, wird der zu Grunde liegende Datensatz an Beispielen in einen Trainings- und einen Testdatensatz aufgesplittet. Das Modell approximiert zunächst die Zielfunktion auf Basis der Trainigsdaten. Anschließend werden Metriken auf dem Trainings- und Testdatensatz evaluiert und die Fähigkeit der Generalisierung anhand der Diskrepanz zwischen den jeweiligen Metriken bestimmt. Weist ein Modell niedrige Metriken für beide Datensätze auf, wird dies als *Underfitting* bezeichnet [127, 14]. In diesem Fall ist die Komplexität der Models zu gering, um die Zielfunktion approximieren zu können. Ist jedoch die Modellkomplexität zu groß, oder wird zu lange auf den gleichen Daten optimiert, besteht die Gefahr des *Overfittings*. Das Modell wird zu sehr an die Trainingsbeispiele angepasst und modelliert zusätzlich das Rauschen, welches dem Trainingsdatensatz unterliegt. Hohe Metriken im Trainingsdatensatz, jedoch niedrige Testmetriken, sind ein Anzeichen für Overfitting [127, 14]. Die Eigenschaften der Generalisierung ist in diesem Fall verletzt. Abbildung 3.5 illustriert die verschiedenen Ausprägungen eines Fits. Gezeigt wird die gelernte Funktion eines Klassifikationsalgorithmus der Datenpunkte.

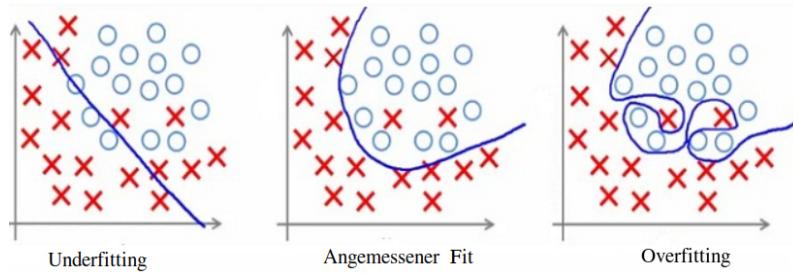


Abbildung 3.5: Underfitting und Overfitting im Vergleich. Bearbeitete Version aus [8].

Bei einem Underfitting ist keine eindeutige Trennung der beiden Klassen zu erkennen. Ein Overfitting führt zu einer zu exakten Trennung. Wird die gelernte Funktion einer Klassifikation auf eine andere Verteilung angewandt, ist eine niedrige Klassifikationsgüte zu erwarten. Bei einem angemessenen Fit ist neben einer hohen Klassifikationsgüte auf den Trainingsdaten eine hohe Klassifikationsgüte auf den Testdaten zu erwarten. Die Performance auf den Testdaten ist dabei gebunden an den Grad der Generalisierung. Tiefergehende Informationen zur Thematik des Under- und Overfittings sind in [117, 101] zu finden. In Kapitel 4.2 wird die Methodik der k-Fold Cross Validation eingeführt, um eine Messung der Generalisierung im Benchmark zu erlauben.

3.3 Grundlagen Deep Learning

Deep Learning-basierte Object Detection-Algorithmen beruhen auf einem erweiterten Konzept des ANN: Dem *Convolutional Neural Network (CNN)*. Im nachfolgenden Kapitel erfolgt zunächst eine Beschreibung des Aufbaus und der Funktionsweise von Convolutional Neural Networks. Anschließend erfolgt eine Beschreibung der verschiedenen Typen von Layern, die in Convolutional Neural Networks vertreten sind.

3.3.1 Convolutional Neural Networks

Ein CNN ist eine spezielle Form eines ANN. Der Unterschied beruht auf der zugrunde liegenden Operation in der Transferfunktion. CNNs nutzen eine Faltung (engl. convolution) anstelle einer Matrixmultiplikation in mindestens einem ihrer Layer [39]. Es existieren viele Varianten von Architekturen für CNNs. Alle Deep Learning-basierten Object Detection-Algorithmen sind auf Basis von CNNs aufgebaut. Im folgenden, sowie im Gesamtkontext der Arbeit, liegt der Fokus auf CNN-Architekturen aus dem Bereich der Bildverarbeitung.

Die biologische Inspiration stammt vom visuellen Cortex in der Großhirnrinde [101]. Er ermöglicht die visuelle Wahrnehmung, welche sich in einen mehrschrittigen Prozess gliedert. Zunächst werden von einzelnen Zellen Grundstrukturen, beispielsweise Kanten, erkannt. Darauf aufbauend können komplexe Zusammenhänge durch die Kombination der

Grundstrukturen erkannt werden [101]. Die Architektur von CNNs richtet sich nach dem gleichen Prinzip [39]. Abbildung 3.6 veranschaulicht das Vorgehen, geschildert am Beispiel einer Bildklassifikation. Einzelne Layer identifizieren zunächst Kanten auf Basis der Eingabepixel. Diese Kanten können in einem nächsten Schritt verwendet werden, um erweiterte Konturen zu erkennen. Darauf aufbauend können einzelne Körperteile extrahiert werden, die in einem finalen Schritt die Zuordnung zu einer Objektklasse erlauben.

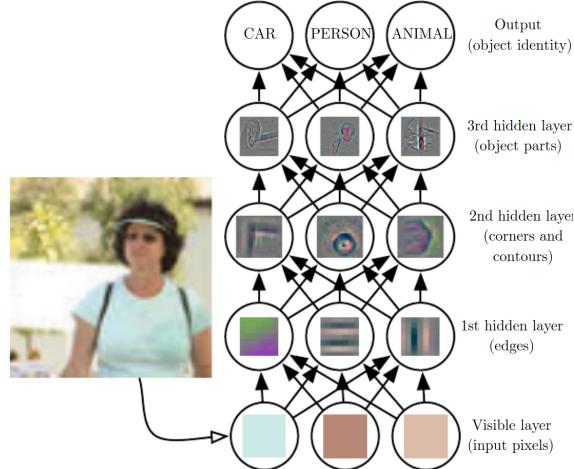


Abbildung 3.6: Funktionsweise eines CNNs. Entnommen aus [39].

3.3.2 Typen von Layer in Convolutional Neural Networks

CNN-Architekturen sind aus mehreren Layer-Typen zusammengesetzt. Die Namensgebung des Forschungsbereichs Deep Learning beruht auf der tiefen (engl. deep) Verkettung dieser Layer. Im nächsten Unterkapitel werden die Basisfunktionalitäten der einzelnen Layer-Typen diskutiert. Tiefergehende theoretische Konzepte sind in [117, 39, 101] zu finden.

Convolutional Layer

Convolutional Layer bilden den Grundbaustein eines CNNs [101]. Wie zuvor angesprochen, stammt der Bezeichner von der zugrunde liegenden mathematischen Operation, der Faltung (engl. convolution), ab. In einem CNN ist diese realisiert durch die Faltung der Daten aus einem Input Layer mit den Filtern eines Convolutional Layers. Bei den Filtern handelt es sich um quadratische Matrizen, deren Größe als Parameter des Convolutional Layers zu spezifizieren ist. Die Operation der Faltung ist ein Skalarprodukt, welches auf einem Teilbereich der Eingabedaten und dem Filter definiert ist. Die Ausgabe, welche durch Anwendung des Prinzips des Sliding-Window auf Basis der Faltung zwischen Eingabedaten und den Filtern generiert wird, wird als *Feature Map* bezeichnet. Der Zusammenschluss aller Feature Maps eines Convolutional Layers ist als *Feature Volume* definiert. Es ist üblich, das Sliding-Window in Schritten von einem Pixel über die Eingabe zu iterieren, was in der

englischen Literatur einem so genannten *Stride* der Größe eins entspricht. Nach Anwendung der Faltung steht jeder Wert in der Output-Feature Map für den Grad der Aktivierung eines (geometrischen) Features, welches durch den jeweiligen Filter repräsentiert ist. Als Aktivierungsfunktion wird meistens die Rectified Linear Unit (ReLU)-Funktion verwendet, die in Form eines zusätzlichen Layers an jeden Convolutional Layer konkateniert ist. Abbildung 3.7 veranschaulicht das Vorgehen eines Convolutional Layer am Beispiel eines 3×3 Filters, angewendet auf eine 3D-Matrix. Das Resultat auf der rechten Seite ergibt sich aus der jeweiligen Faltung zwischen Filter und dem entsprechenden Bereich im Input Layer (hell rot mit dunkel rot, hellgrün mit dunkelgrün, hellblau mit dunkelblau), sowie der anschließenden Addition aller Teilergebnisse.

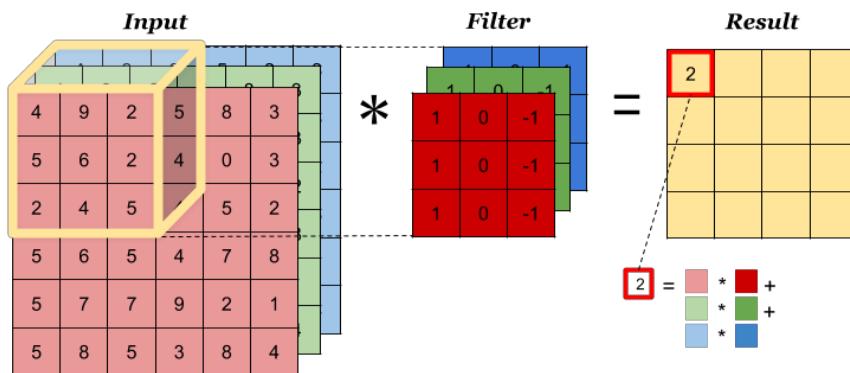


Abbildung 3.7: Prinzip eines Convolutional Layers. Entnommen aus [103].

Ein im Zusammenhang von Convolutional Layer wichtiges Konzept ist das Receptive Field. Dieses ist definiert als: Bereich des Eingaberaums, mit dem die Features eines Filters oder auch *Kernel* verbunden sind und ist charakterisiert über den Mittelpunkt und die Größe dieses Bereichs [154]. Je näher ein Eingabepixel am Mittelpunkt des Receptive Field lokalisiert ist, desto mehr trägt er zur Berechnung des entsprechenden Features bei. Damit wird nach [49] neben der Betrachtung eines bestimmten Bereiches des Eingaberaums, ein exponentieller Fokus auf den Mittelpunkt dieses Bereichs gelegt.

Pooling Layer

Pooling Layer werden dazu genutzt, die Datendimension zu reduzieren und damit gleichermaßen die Anzahl an Parametern und Rechenoperationen im Netzwerk zu verringern. Hierfür wird ähnlich zum Convolutional Layer ein Filter eingesetzt. Im Gegensatz zum Skalarprodukt wird im Fall des Pooling üblicherweise der Maximum Operator verwendet. Es sind jedoch auch andere Operatoren wie beispielsweise der Mittelwert möglich. Die Maximum Operation bestimmt den maximalen Wert des Ausschnitts aus der Eingabe Feature Map. Zur Realisierung der Reduzierung der Dimension wird ein Stride der Größe zwei eingesetzt, um den Filter über die Eingabe Feature-Map zu bewegen. Abbildung 3.8 veranschaulicht dieses Vorgehen.

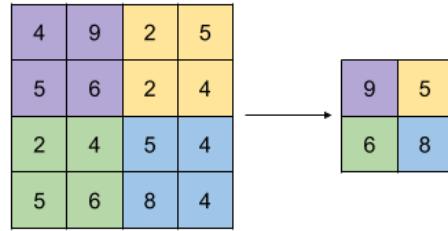


Abbildung 3.8: Funktionsweise eines Pooling Layers. Entnommen aus [103].

Region of Interest Pooling Layer

Der *Region of Interest (ROI) Pooling Layer* nutzt ähnlich zum Pooling Layer einen Maximum Operator, um eine Eingabe Feature Map in eine vordefinierte Größe runter zu skalieren. Die Namensgebung beruht auf der Anwendung des ROI Pooling Layer, um Features interessanter Bereiche (engl. regions of interest) einer gegebenen Feature Map für die Weiterverarbeitung bereitzustellen. Neben der Input Feature Map erwartet der ROI Pooling Layer eine $N \times 4$ Matrix, mit N als Anzahl aller ROI, für die ein Pooling durchzuführen ist und einen 1×4 Vektor zu Spezifizierung der Koordinaten der ROI. Abbildung 3.9 illustriert das Vorgehen der Runterskalierung am Beispiel einer ROI der Größe 7×5 in eine Matrix der Größe 2×2 .

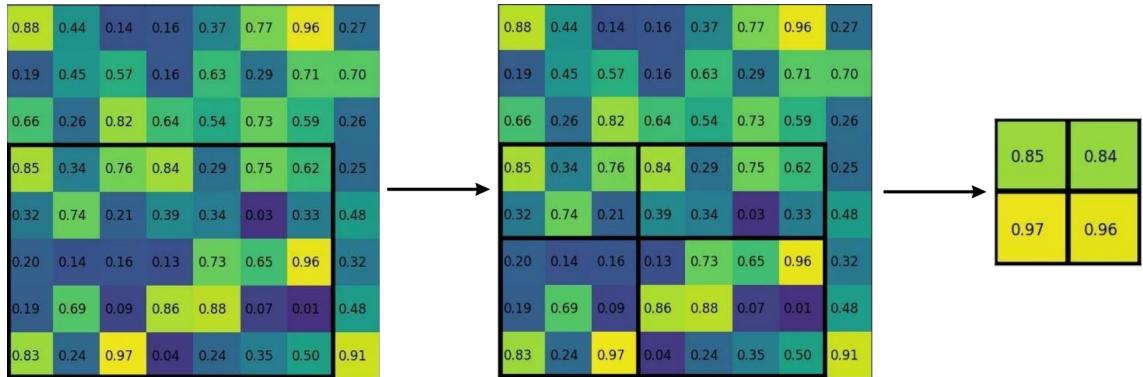


Abbildung 3.9: Funktionsweise eines ROI Pooling Layer. Entnommen aus [43].

Fully Connected Layer

Die Neuronen eines *Fully Connected Layers* sind zu jedem Neuron des vorherigen Layers verbunden, so wie es in den Hidden Layern regulärer Neuronaler Netze der Fall ist [130]. Somit werden anstelle der Filter wieder Matrix Multiplikationen, gefolgt von einem Bias Offset, eingesetzt. Die meist vertretende Architektur von CNNs folgt folgendem Schema [130]:

INPUT → [[CONV → RELU] * N → POOL?] * M → [FC → RELU] * K → FC

Dabei stehen N, M und K für die Anzahl des jeweiligen Blocks. CONV, POOL und FC sind Abkürzungen für die zuvor beschriebenen Typen von Layern: Convolutional Layer, Pooling Layer und Fully Connected Layer. Insgesamt ist demnach eine beliebige Verkettung der verschiedenen Typen von Layern möglich.

Softmax Layer

Zur Realisierung einer Klassifikation unter Anwendung eines CNNs wird am Ende der Architektur ein *Softmax Layer* verwendet. Dieser wird meist an einen Fully Connected Layer konkateniert. Der Rückgabewert ist in der Literatur definiert als Wahrscheinlichkeitsverteilung sich gegenseitig ausschließender Klassen in Form eines Vektor $V^{1 \times N}$ [101]. Hierbei ist N die Anzahl aller Klassen. Jedes Element a_i des Vektors spezifiziert die Wahrscheinlichkeit der entsprechenden Klassen-ID. Es gilt $a_i \in [0, 1]$. Für das Beispiel der Bildklassifizierung wird dem Eingabebild die Klassen-ID zugeordnet, welche die höchste Klassenwahrscheinlichkeit aufweist. Im Kontext der Object Detection wird auch der Begriff der *Klassen-Scores* verwendet.

Dilated Convolutional Layer

Im Zuge der Realisierung einer automatisierten Detektion von Objekten spielt die Unterscheidung von Vorder- und Hintergrund eine wichtige Rolle. Daher wurde in [153] ein zusätzlicher Parameter für Convolutional Layer eingeführt: Die *Dilation*. Hierfür wird das Konzept des Stride auf den Kernel eines Convolutional Layers angewendet. Dies resultiert in einem vergrößerten Receptive Field bei gleichbleibenden Berechnungskosten [104]. Mit einem größeren Receptive Field ist nach [139] eine bessere Unterscheidung von Vorder- und Hintergrund möglich, welche die Genauigkeit der Detektion von Objekten begünstigt. Abbildung 3.11 stellt den Aufbau einer Convolutional und einer Dilated Convolution gegenüber. Beide erhalten eine Input Feature Map und berechnen anhand ihres Kernel einen Ausgabewert für die Output Feature Map. Abbildung 3.10a illustriert die Anwendung eines normalen 3×3 Kernel einer Convolutional. Abbildung 3.11b zeigt die Anwendung eines 3×3 Kernel mit einer Dilation der Größe eins. Beide verwenden einen Stride der Größe zwei.

Deformable Convolutional Layer

Aufgrund der festen geometrischen Strukturen, denen Convolutional Layer unterliegen, sind diese inhärent in der Menge an geometrischen Transformationen begrenzt, welche unter Anwendung von Convolutional Layern modelliert werden können [24]. Dadurch ergeben

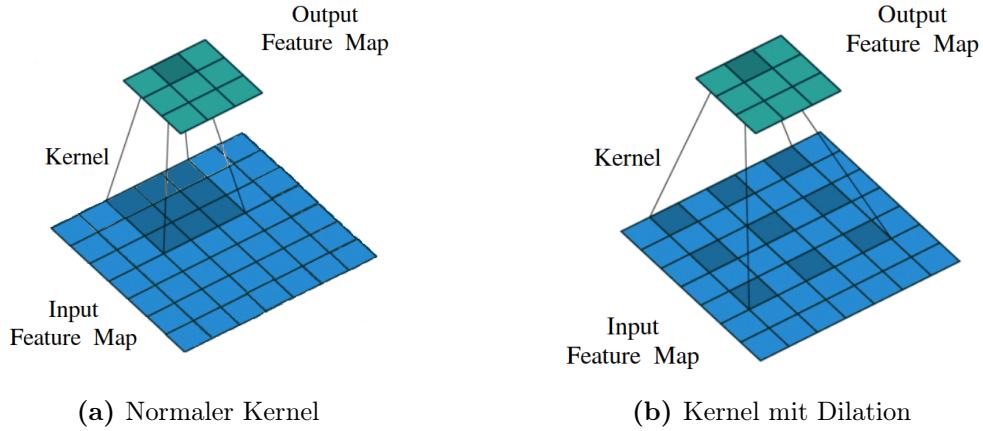


Abbildung 3.10: Gegenüberstellung eines normalen Kernel und eines Kernel mit Dilatation. Bearbeitete Version aus [139].

sich beispielsweise Probleme bei der Detektion von Objekten, die verdeckt oder nicht in ihrer ursprünglichen Form im Eingabebild abgebildet sind. Daher wurde in [24] das Konzept der *Deformable Convolutional Layer* (dt. deformierbar) eingeführt. Diese erweitern die Kapazität an geometrischen Transformationen, die von Convolutional Layern geleistet werden können. Dafür werden, neben den Gewichten des Kernels, Offsets für die einzelnen Einträge des Kernels gelernt. Anhand der Offsets wird das Receptive Field des Convolutional Layers verändert. Im Zuge dessen können deformierte Bereiche der Eingabe für die Ausführung des Convolution-Operators betrachtet werden. Verglichen mit Dilated Convolutional Layern, wird der Dilation-Paramter auf jeden Wert des Kernels angewendet [140]. Die Offsets werden dabei über die Feature Maps vorheriger Convolutional Layer gelernt. Damit ist die Deformierung dicht gebunden an die lokalen Eingabe Features dieser Layer. Aufgrund dieses Aufbaus ist eine einfache Integration von Deformable Convolutional Layern in bestehende CNN-Architekturen möglich. Abbildung 3.11 stellt den Aufbau einer Convolutional und einer Deformable Convolution gegenüber. Beide erhalten eine Input Feature Map und berechnen anhand ihres Kernel einen Ausgabewert für die Output Feature Map. Abbildung 3.11a illustriert die Anwendung eines normalen 3×3 Kernel einer Convolutional. Abbildung 3.11b zeigt die zusätzliche Offset-Matrix der Deformable Convolution. Anhand der Offsets wird der 3×3 Kernel auf entsprechend transponierten Eingabepixeln angewendet.

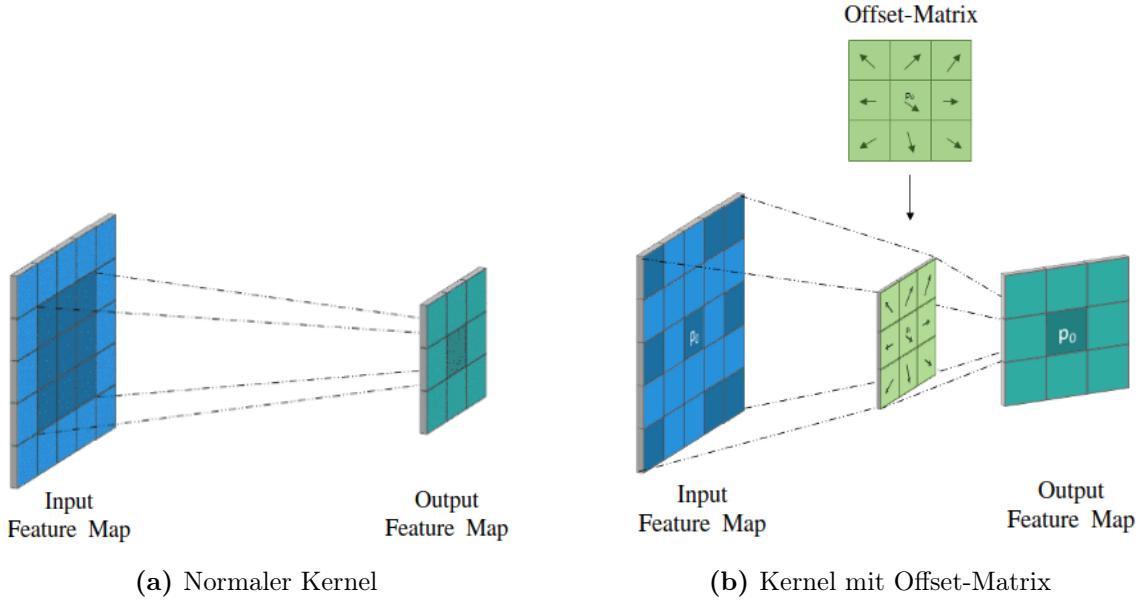


Abbildung 3.11: Gegenüberstellung eines normalen Kernel und eines Kernel mit Offsets. Bearbeitete Version aus [107].

3.4 Architektur von Object Detection-Algorithmen

Wie in Kapitel 3.1.2 geschildert, befasst sich die Object Detection mit der automatisierten Lokalisierung und Klassifizierung von Instanzen von vorgegebener Objektklassen in einem digitalen Bild [160]. Ausgabe der Object Detection-Algorithmen ist eine Liste der erkannten Objekte in Form von Rechtecken, welche als *Bounding-Boxen* bezeichnet werden. Die Bounding-Boxen besitzen drei Attribute: Klassen-ID, Klassen-Score (in der Literatur auch Klassenwahrscheinlichkeit) und ein 4D-Array zur Speicherung der Pixel-Koordinaten, bezogen auf die Position der Bounding-Boxen in Referenz zum Eingabebild.

Bei Object Detection-Algorithmen wird zwischen zwei Varianten der Herangehensweise unterschieden: *Two-Stage* Verfahren und *One-Stage* Verfahren [56]. In [160] werden Two-Stage Verfahren mit Region Proposal Based Frameworks und One-Stage Verfahren mit Regression / Classification Based Frameworks bezeichnet. Grundlage für beide Verfahren sind visuelle Features, die von einer *Backbone-Architektur* oder auch *Feature-Extraktor* aus dem gegebenen Eingabebild extrahiert werden. Die Features ermöglichen eine robuste Repräsentation und semantische Zuordnung der zu detektierenden Objekte [160]. Hierfür werden meist CNN-Architekturen herangezogen, die aus dem Bereich der Bildklassifizierung stammen [56]. Nach dem Entfernen der Schichten, die für die eigentliche Klassifizierung verantwortlich sind, wird die resultierende CNN-Architektur an den Object Detection-Algorithmus konkateniert. Der Object Detection-Algorithmus selbst wird dabei als *Meta-Architektur* bezeichnet. Die Backbone-Architekturen stellen die für die weiteren Berechnungen notwendigen visuellen Features in Form einer Feature Map bereit. Die Qua-

lität der auf diese Weise zur Verfügung gestellten visuellen Features ist maßgeblich für die Performance der Object Detection-Algorithmen verantwortlich [56]. Gängige Backbone-Architekturen sind: VGG [124], MobileNet [50], Inception [132], ResNet [47] und ResNeXt [150]. Abbildung 3.12 zeigt das Beispiel eines CNNs zur Klassifikation und den Bereich, welcher für Object Detection-Algorithmen zu entfernen ist.

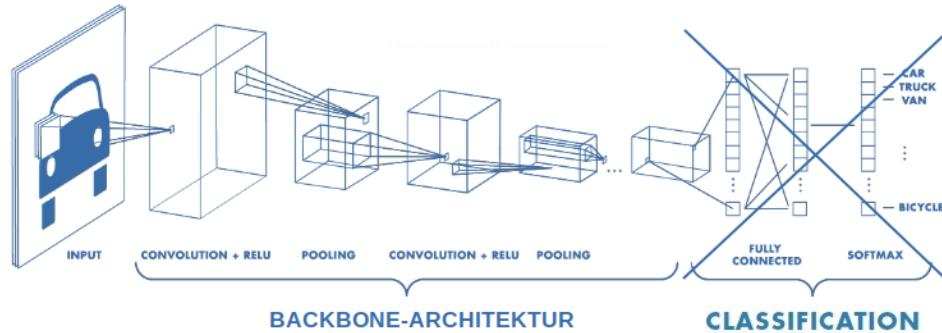


Abbildung 3.12: CNN zur Bildklassifikation. Bearbeitete Version aus [19].

Das Prinzip hinter Two-Stage Verfahren ist vergleichbar mit der menschlichen Art der Wahrnehmung: Nach dem Scannen eines großen Bereichs (des vollständigen Szenarios), werden relevante Bereiche identifiziert und fokussiert [160]. Dies entspricht dem ersten Schritt der Two-Stage Verfahren, dem Region-Proposal-Schritt, aus dem so genannte *Kandidaten-Boxen* hervorgehen. Da Objekte in einer großen Variabilität betreffend ihrer Größe, ihres Seitenverhältnisses und ihrer Position im Bild auftreten können, wird in einem mehrstufigen Prozess das gesamte Eingabebild gescannt und alle relevanten Bereiche als Kandidaten-Boxen ausgegeben. Diese sind analog zu Bounding-Boxen kodiert über ein 4D-Array zur Speicherung der Bildkoordinaten. Eine Angabe der Klassen-ID und Klassen-Score erfolgt jedoch nicht. Mit diesem Vorgehen ist einer hohen Rechenaufwand verbunden. Wird die Suche jedoch auf eine feste Anzahl von Masken beschränkt, reduziert sich die Qualität der Kandidaten-Boxen. Dies zieht gleichermaßen eine Reduzierung der Genauigkeit bei der Detektion von Objekten mit sich. Daher wurden mit der Zeit starke Optimierungen an Algorithmen zur Generierung der Kandidaten-Boxen vorgenommen [119, 38, 37, 112]. Das Konzept der *Anchor-Box* hat maßgeblich zur Verbesserung der Algorithmen beigetragen. Anstelle des Sliding-Windows werden dafür an vordefinierten Positionen Boxen verschiedener Größe und Skalierung generiert und als Eingabe für den Region-Proposal-Schritt verwendet. Die Anwendung des Konzepts der Anchor-Box beruht zudem auf einer Auslagerung des Region-Proposal-Schritt in ein eigenes CNN, welches mit *Region Proposal Network (RPN)* bezeichnet wird.

Anschließend werden in einem zweiten Schritt die generierten Kandidaten-Boxen klassifiziert und die exakten Koordinaten der Bounding-Boxen mittels Regressionsverfahren bestimmt. Aus der Klassifikation gehen die Klassen-Scores für jede Bounding-Box hervor. Die Berechnungen der Klassifikation und Bounding-Box-Regression beruhen maßgeblich

auf den visuellen Features, die von der Backbone-Architektur zur Verfügung gestellt werden. Diese werden unter Anwendung eines ROI Pooling Layers für jede Kandidaten-Box aus der Output-Feature Map der Backbone-Architektur extrahiert [56]. Durch die Aufteilung in zwei getrennte Schritte erreichen Two-Stage Verfahren eine hohe Genauigkeit bezogen auf die Lokalisierung und Klassifizierung der Objekte. Bedingt durch den erhöhten Grad an Komplexität, weisen Two-Stage Verfahren jedoch gleichermaßen eine erhöhte Laufzeit auf [56]. Aktuell ist der Basisalgorithmus, welcher für alle Entwicklungen im Bereich der Two-Stage Verfahren herangezogen wird, das *Faster Region-CNN (R-CNN)* [112]. Bei One-Stage Verfahren entfällt der separate Schritt der Kandidaten-Box Generierung. Anstelle des ROI Pooling Layers wird die Output-Feature Map der Backbone-Architektur direkt zur Weiterverarbeitung verwendet [74, 108]. Damit wird eine globale Regression und Klassifizierung realisiert. Es findet eine direkte Zuordnung der Eingabepixel (Input-Feature Map) zu Bounding-Box-Koordinaten und Klassen-IDs statt. Vorreiter ist die *You Only Look Once (YOLO)*-Architektur [108]. Diese nutzt ähnlich zu Faster R-CNN das Konzept der Anchor-Box. Hierfür wird zunächst die finale Feature Map in ein Raster eingeteilt, dessen Zellen als Anchor-Box behandelt werden. Die Mittelpunkte der Rasterzellen dienen als potentielle Ausgangspunkte der Bounding-Box-Regression und Klassifikation. Aktuell gängige Methoden [109, 110, 74] verwenden mehrere Anchor-Boxen in verschiedenen Größen und Skalierungen. Auf Basis der Messung der Schnittmenge zwischen den Anchor- und Ground-Truth-Boxen wird anschließend zwischen positiven und negativen Anchor-Boxen unterschieden, um die Trainingsbeispiele für die Klassifikation zu definieren. Die negativen Anchor-Boxen werden je nach Architektur entweder einer Hintergrund Klasse zugeordnet oder mit einem entsprechenden Faktor runtergewichtet. In der Bounding-Box-Regression werden negative Anchor-Boxen nicht für das Training berücksichtigt.

Durch den verringerten Grad an Komplexität ergeben sich deutlich kürzere Laufzeiten verglichen mit Two-Stage Verfahren, jedoch ergeben sich gleichermaßen Einbußen bezüglich der Detektionsgenauigkeit. Das prinzipielle Vorgehen beider Varianten ist in Abbildung 3.13 veranschaulicht. Die einzelnen Blöcke repräsentieren die verschiedenen Layer in der CNN-Architektur, sowie den jeweiligen Pfad der *Klassifikation* (cls) und *Bounding-Box-Regression* bzw. Lokalisierung (loc). Gegenübergestellt wird der jeweilige Bereich der Backbone- und der Meta-Architektur.

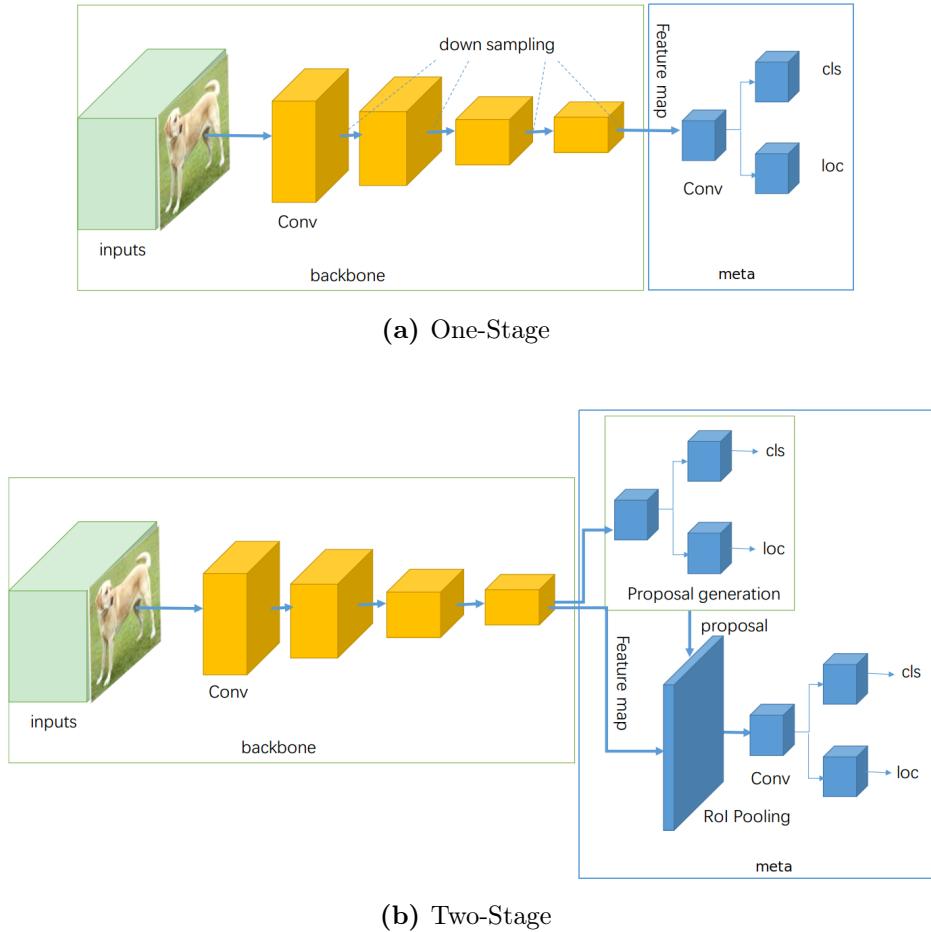


Abbildung 3.13: Gegenüberstellung der Architekturen von One- und Two-Stage Verfahren. Bearbeitete Version aus [56].

Für beide Verfahren findet eine Nachbereitung (post-processing) statt. Mittels dem Greedy-Algorithmus *Non Maximum Suppression (NMS)* (siehe Anhang Algorithmus 3) werden die aus den Object Detection-Algorithmus resultierenden Bounding-Boxen gefiltert [160]. Grundlage für die Filterung ist neben der Klassen-Scores die Überschneidung zwischen Bounding-Boxen. Die Überschneidung ist definiert über die in Formel 3.1 gegebenen Metrik, der *Intersection over Union (IOU)* - Schnittmenge unter Vereinigung. Zunächst wird für jede Objektinstanz eine Bounding-Box mit der höchsten Klassen-Score bestimmt. Anschließend werden alle Bounding-Boxen, die eine zu große Überlappung aufweisen, gefiltert. Unter Angabe eines Grenzwerts T_{nms} wird spezifiziert, wie stark sich die Bounding-Boxen überlappen dürfen.

Intersection-Over-Union (IOU)

$$IOU = \frac{Flaeche(Box_a \cap Box_b)}{Flaeche(Box_a \cup Box_b)} \quad (3.1)$$

Abbildung 3.14 veranschaulicht das Ergebnis vor und nach Anwendung des NMS-Algorithmus. Das detaillierte Vorgehen ist in Algorithmus 3 (siehe Anhang) geschildert. Dort wird näher darauf eingegangen, wie die IOU-Metrik und der Grenzwert T_{nms} verwendet werden. Als Resultat werden alle Bounding-Boxen ausgegeben, die für eine detektierte Objektinstanz die maximale Klassen-Score und keine Überlappung aufweisen.

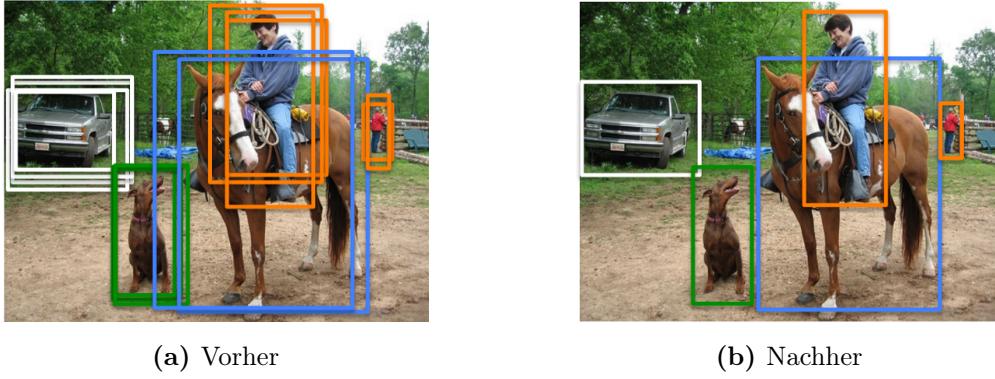


Abbildung 3.14: Ergebnis vor und nach Anwendung des NMS-Algorithmus. Bearbeitete Version aus [11].

3.5 Evaluierungsmetriken von Object Detection-Algorithmen

Benchmark-Datensätze dienen dem einheitlichen Vergleich neu entwickelter Verfahren. Hierfür gibt es im Bereich der Object Detection verschiedene Open-Source Bilddatensätze. Diese nehmen zum Einen die große Hürde, Trainingsdaten zu annotieren, zum Anderen sind für jeden Datensatz Metriken definiert, die eine einheitliche Evaluierung der Verfahren erlauben. Für Object Detection-Algorithmen sind der *MS COCO-Datensatz* [69] und der *Pascal VOC-Datensatz* [121, 27] die populärsten Beispiele [56]. Der Pascal VOC-Datensatz (2012 [28]) umfasst 20 Objektklassen, für die 11.000 Bilder und Annotation im Trainingsdatensatz vorliegen. Für den MS COCO test-dev Datensatz sind dies 115.000 Bilder und Annotationen für 80 Objektklassen [70]. Des Weiteren spielt der ImageNet-Datensatz [116] eine wichtige Rolle für Object Detection-Algorithmen. Dieser wird als Grundlage für das Training der in Kapitel 3.4 beschriebenen Backbone-Architekturen eingesetzt. Zudem wird der ImageNet-Datensatz als Grundlage zur Ermittlung der State-of-the-Art Klassifikationsalgorithmen in der Large Scale Visual Recognition Challenge (ILSVRC) [129] herangezogen. Mit etwa 1.200.000 Bildern und 1.000 Objektklassen werden viele Bereiche des öffentlichen Lebens abgedeckt [129]. Durch ein initiales Training auf dem ImageNet-Datensatz kann den Backbone-Architekturen damit ein großes Spektrum an visuellen Fea-

tures angelernt werden. Da die Performance bezüglich der Detektionsgenauigkeit von Object Detection-Algorithmen maßgeblich von der Qualität dieser visuellen Features abhängt [56], ist eine auf dem ImageNet-Datensatz initialisierte Backbone-Architektur von großer Bedeutung. Um die Detektionsgenauigkeit von Objekten messbar zu machen, wurden für die Evaluierung von Object Detection-Algorithmen, die auf dem MS COCO- und Pascal VOC-Datensatz trainiert wurden, eine Reihe von Metriken eingeführt. Diese werden in den nachfolgenden Abschnitten definiert.

True Positive, False Positive und Condition Positive

Die *True Positive (TP)* sind definiert als die absolute Anzahl an Datenpunkten, die als positiv vorhergesagt worden sind und tatsächlich positiv sind [82]. Die *False Positive (FP)* sind definiert als die absolute Anzahl an Datenpunkten, die als positiv vorhergesagt worden sind, aber negativ sind [82]. Bezogen auf die Object Detection entspricht dies denjenigen Bounding-Boxen, die erkannt wurden und einen gegebenen Grenzwert für die IOU-Metrik (Formel 3.1) überschreiten bzw. unterschreiten. Bei den Daten, die zur Evaluierung herangezogen werden, wird im Allgemeinen der Ausdruck der Ground-Truth-Daten verwendet. Die IOU-Metrik wird dabei auf Grundlage der erkannten Bounding-Boxen B und den Ground-Truth-Bounding-Boxen G berechnet. Das exakte Vorgehen wird durch Algorithmus 1 definiert.

Algorithmus 1: Bestimmung der True Positive und False Positive.

Bearbeitete Version aus [116].

Eingabe:

$B = \{(b_j, s_j)\}_{j=1}^M$, Liste der detektierten Bounding-Boxen mit zugehörigen Klassen-Scores (-Wahrscheinlichkeiten)

G : Liste der Ground-Truth-Bounding-Boxen

T_{iou} : Grenzwert für die IOU-Metrik

T_{score} : Grenzwert für die Klassen-Scores

Ausgabe:

Z : Binärvektor der angibt, ob eine vorhergesagte Bounding-Box ein TP oder FP ist

```

1 Sei  $U = G$  als Menge der Boxen die kein Match aufweisen
2 Sortiere  $B$  in absteigender Reihenfolge nach  $s_j$ 
3 for  $j \in \{1, \dots, |B|\}$  do
4   if  $s_i < T_{score}$  then
5      $B = B \setminus \{(b_j, s_j)\}$ 
6      $z_j = 0$ 
7   end
8 end
9 for  $i \in \{1, \dots, |B|\}$  do
10    $C = \{\}$ 
11   for  $k \in \{1, \dots, |U|\}$  do
12     if  $IOU(u_k, b_i) \geq T_{iou}$  then
13        $C = C \cup u_k$ , Box hat passende Überschneidung mit Ground-Truth-Box
14     end
15   end
16   if  $C \neq \emptyset$  then
17     Mit  $k^* = argmax_{\{k: u_k \in C\}} IOU(u_k, b_i)$ 
18     Setze  $U = U \setminus u_{k^*}$ 
19     Setze  $z_i = 1$ , da es sich um ein True Positive handelt
20   else
21     Setze  $z_i = 0$ , da es sich um ein False Positive handelt
22   end
23 end
24 return  $Z$ 

```

Für nachfolgende Betrachtungen gilt $T_{score} = 0.0$. In Kapitel 6 wird für jeden Algorithmus des Benchmarks ein konkreter Wert für T_{score} bestimmt.

Beruhend auf Z aus Algorithmus 1 werden die TP und FP definieren als:

$$TP = \sum_{z_j \in Z, z_j=1} 1 \quad (3.2)$$

$$FP = \sum_{z_j \in Z, z_j=0} 1 \quad (3.3)$$

Die *Condition Positive (CP)* sind definiert als absolute Anzahl der Ground-Truth-Bounding-Boxen im Datensatz [116], demnach die Anzahl an Bounding-Boxen der Menge G aus Algorithmus 1.

$$CP = |G| \quad (3.4)$$

Precision

Die *Precision (PR)* misst den Grad der Genauigkeit als Anteil der TP gegenüber aller vom Algorithmus detektierten Objekte [82, 53, 116].

$$PR = \frac{TP}{TP + FP} \quad (3.5)$$

Recall

Der *Recall (RC)* gibt den Anteil der Bounding-Boxen an, die als TP detektiert wurden und tatsächlich positiv sind [82]. Demnach spezifiziert als Anteil der TP gegenüber der Anzahl von Ground-Truth-Bounding-Boxen.

$$RC = \frac{TP}{CP} \quad (3.6)$$

Average Precision

Die *Average Precision (AP)*-Metrik (Formel 3.8) misst die Detektionsgenauigkeit von Object Detection-Algorithmen an. Um die Berechnung zu veranschaulichen, wird ein Beispiel aus [53] verwendet. Ausgangspunkt sind die berechneten PR- und RC-Metriken eines Experiments E , in dem es um die Detektion von Äpfeln in einem gegebenen Bilddatensatz geht. Aus dem Experiment resultieren die in Tabelle 3.1 aufgeführten Metriken.

$PR_{measured}$	1.0	1.0	0.67	0.5	0.4	0.5	0.57	0.5	0.44	0.5
$RC_{measured}$	0.2	0.4	0.4	0.4	0.4	0.6	0.8	0.8	0.8	1.0

Tabelle 3.1: Gemessene PR- und RC-Metrik für Experiment E . Bearbeitete Version aus [53].

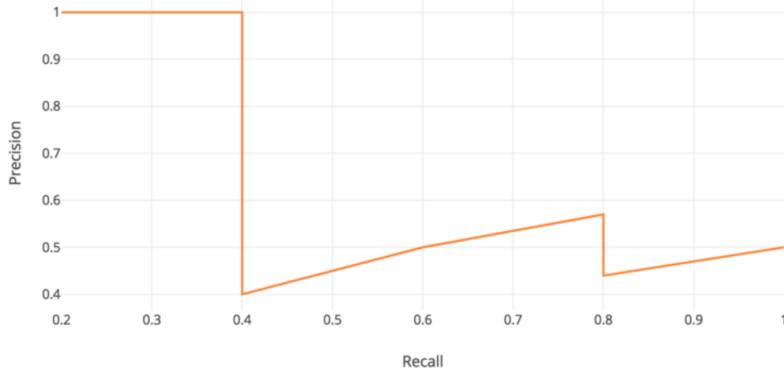


Abbildung 3.15: PR-RC-Kurve p bestimmt auf Grundlage der Daten in Tabelle 3.1. Entnommen aus [53].

Aus diesen Daten wird die PR-RC-Kurve p bestimmt, die in Abbildung 3.15 dargestellt ist. Die AP-Metrik ist das Integral $\int_0^1 p(r)dr$. Um die Berechnung zu vereinfachen, wird p in eine Treppenfunktion p_{interp} transformiert. Diese ist definiert über die Formel 3.7. Dann ist die AP-Metrik durch die Formel 3.8 approximiert, wobei R eine Zerlegung der x-Achse ist. In E wird $R = \{0, 0.1, 0.2, \dots, 1\}$ gesetzt. p_{interp} ist in Abbildung 3.16 grün dargestellt und die Werte der Zerlegung als rote Punkte gekennzeichnet. Für das Experiment sind für alle r aus R die Werte für p_{interp} in Tabelle 3.2 aufgeführt.

$$p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r}) \quad (3.7)$$

$$AP = \frac{1}{|R|} \sum_{r \in R} p_{interp}(r) \quad (3.8)$$

$p_{interp}(r)$	1.0	1.0	1.0	1.0	1.0	0.57	0.57	0.57	0.57	0.5	0.5
r	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0

Tabelle 3.2: Werte für $p_{interp}(r)$ mit $r \in R$ für Experiment E . Bearbeitete Version aus [53].

Für den Pascal VOC-Datensatz wird ebenfalls $R = \{0, 0.1, 0.2, \dots, 1\}$ gewählt. Zudem gilt $T_{iou}=0.5$ aus Algorithmus 1 zur Bestimmung von Z , welches die Grundlage zur Bestimmung der TP und FP bildet (siehe Formel 3.2 und 3.3) [27].

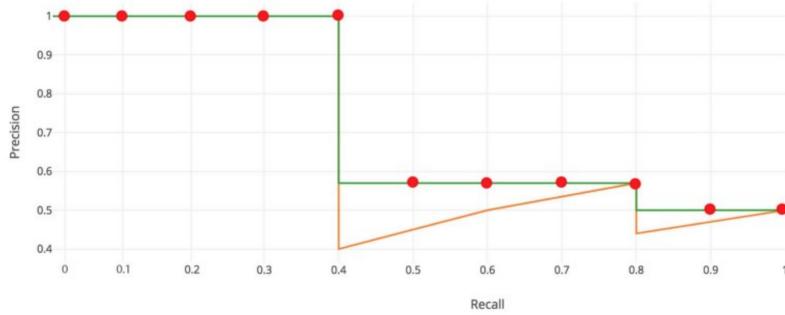


Abbildung 3.16: Illustration der Treppefunktion p_{interp} . Entnommen aus [53].

Für den MS COCO-Datensatz wird die Zerlegung verfeinert, sodass $R = \{0, 0.01, \dots, 1\}$ und $|R| = 101$ gilt. Der IOU-Grenzwert T_{iou} ist für den MS COCO-Datensatz in verschiedenen Stufen und Größenklassen festgelegt. Die Größenklassen s , m und l sind dabei bezogen auf die Größe der *Pixelfläche (PF)* der Bounding-Boxen. Im MS COCO-Datensatz weisen alle Bilder Formate ähnlich zur Auflösung von 640×480 auf. Dies erlaubt die Betrachtung der PF, ohne Skalierungen vornehmen zu müssen. Eine Übersicht aller im MS COCO-Datensatz verwendeten Metriken ist in Tabelle 3.3 gegeben. Die AP_{coco} -Metrik wird als Mittelwert der AP-Metriken bestimmt, die unter Anwendung der verschiedenen IOU-Grenzwerte in $T_{iou}^* = \{0.5, 0.55, \dots, 0.95\}$ berechnet werden. Anhand dessen wird eine gute Lokalisierung der erkannten Bounding-Boxen stärker gewichtet [70], da mit jedem größeren T_{iou} aus T_{iou}^* eine größere Schnittmenge zwischen den detektierten Bounding-Boxen und den Ground-Truth-Bounding-Boxen gefordert wird. Die Bezeichnung AP_{coco} -Metrik gilt nur im Rahmen der Arbeit. In der Literatur wird diese als mAP oder auch nur AP bezeichnet. Zur Betrachtung der Detektionsgenauigkeit für verschiedene Größenklassen werden die Metriken AP_s , AP_m und AP_l herangezogen. Diese folgen der Berechnung der AP_{coco} -Metrik. Dabei wird jeweils die entsprechende Teilmenge an detektierten Bounding-Boxen B_s , B_m und B_l zur Berechnung der jeweiligen AP_s -, AP_m - und AP_l -Metrik herangezogen. Die Spezifikation der Teilmengen ergeben sich aus dem in Tabelle 3.3 aufgeführten Wertebereich der PF.

Metrik	Definition
AP_{coco}	Gemittelte AP unter Verwendung von $T_{iou}^* = \{0.5, 0.55, \dots, 0.95\}$
$AP@0.5$	AP für IOU mit $T_{iou} = 0.5$
$AP@0.75$	AP für IOU mit $T_{iou} = 0.75$
AP_s	AP_{coco} für kleine Objekte, es gilt: $PF < 32^2$
AP_m	AP_{coco} für mittelgroße Objekte, es gilt: $32^2 < PF < 96^2$
AP_l	AP_{coco} für große Objekte, es gilt: $PF > 96^2$

Tabelle 3.3: Metriken des MS COCO-Datensatzes. Bearbeitete Version aus [70].

In PASCAL VOC 2010 wurde die Berechnung der AP-Metrik angepasst [99, 53]. Anstelle der vordefinierten Zerlegung wird eine neue Zerlegung $R^* = \{r_0, \dots, r_N\}$ anhand der Peaks der PR-RC-Kurve bestimmt. Dadurch ergibt sich eine bessere Approximation der PR-RC-Kurve. Für die Formel 3.7 und 3.8 ergeben sich folgende Änderungen:

$$p_{interp}(r_{n+1}) = \max_{\tilde{r} \geq r_{n+1}} p(\tilde{r}) \quad (3.9)$$

$$AP = \sum_{n=0}^N (r_{n+1} - r_n) \cdot PR_{interp}(r_{n+1}) \quad (3.10)$$

Abbildung 3.17 illustriert die Berechnung der adaptierten AP-Metrik. Zunächst wird p_{interp} für den r_2 bestimmt. Anschließend wird durch das Produkt $(r_2 - r_1) \cdot p_{interp}(r_2)$ die grau markierte Fläche bestimmt. Oben rechts ist die ermittelte Gesamtfläche abgebildet.

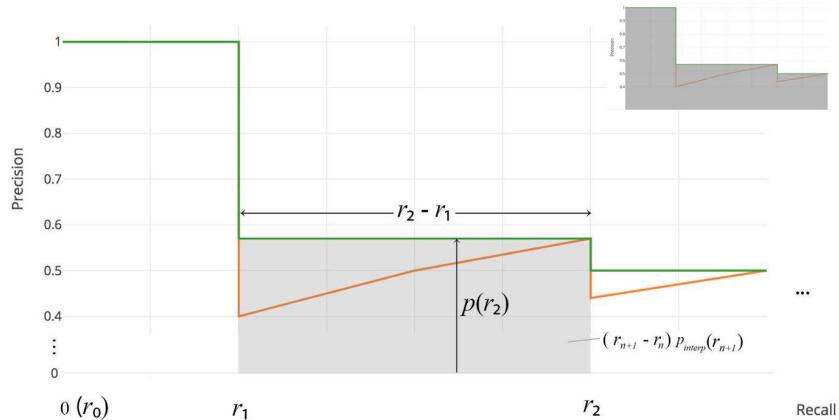


Abbildung 3.17: Graph zur Schilderung der Berechnung der adaptierten AP-Metrik. Entnommen aus [53].

Kapitel 4

Methode und Datensatz

4.1 Einordnung CRISP-DM

Die Vorgehensweise bei der Modellierung von Algorithmen im Bereich des Machine Learning ist am Schema des *CRoss Industry Standard Process for Data Mining (CRISP-DM)* [18] orientiert. Nach [83] ist CRISP-DM als Leitfaden zu verstehen. Der Kern des Schemas beruht auf sechs zyklischen Phasen, die Rücksprünge erlauben. Abbildung 4.1 veranschaulicht die einzelnen Phasenübergänge. Eine detaillierte Beschreibung einzelner durchzuführender Tasks innerhalb der Phasen von CRISP-DM ist in [18, 120, 128] zu finden. Der Fokus der Arbeit ist auf die Erstellung einer Entscheidungsbasis zum Vergleich von Object Detection-Algorithmen gelegt. Auf Details der Einzelschritte von CRISP-DM wird daher nicht eingegangen. Nachfolgend findet eine kurze Einordnung der Anwendung von CRISP-DM auf den in Kapitel 1 geschilderten Anwendungsfall statt. Für detailliertere Beschreibungen sind jeweils Referenzen angegeben, wo diese in der Arbeit zu finden sind.

Geschäftsverständnis

Das Ziel ist die Erhöhung der Transparenz in Lagerprozessen, die im Kontext der Intralogistik betrachtet werden. Hierfür gilt es, automatisiert Bewegungsprofile zu generieren. Auf deren Grundlage erfolgt ein Abgleich und eine Analyse von vordefinierten Business KPI (Key Performance Indicator). Zur Realisierung dieser Vorgabe werden Methoden aus dem Forschungsfeld der Bildverarbeitung eingesetzt. Die Basis bildet ein Object Tracking-Algorithmus, der aus einer Eigenentwicklung des Fraunhofer IML resultiert, sowie Bilddaten, die im Rahmen eines Kooperationsprojekts zwischen dem Fraunhofer IML [31] und zwei weiteren Projektpartnern erhoben werden. Der Object Tracking-Algorithmus erwartet als Eingabe Bounding-Boxen, welche von einem Object Detection-Algorithmus zu generieren sind. Der Kontext der zu detektierenden Objekte wird für die Arbeit auf Szenarien beschränkt, welche in Regalgängen der Lagersysteme abgewickelt werden. Objekte dieser Szenarien sind mittels Deep Learning-basierten Object Detection-Algorithmen automati-

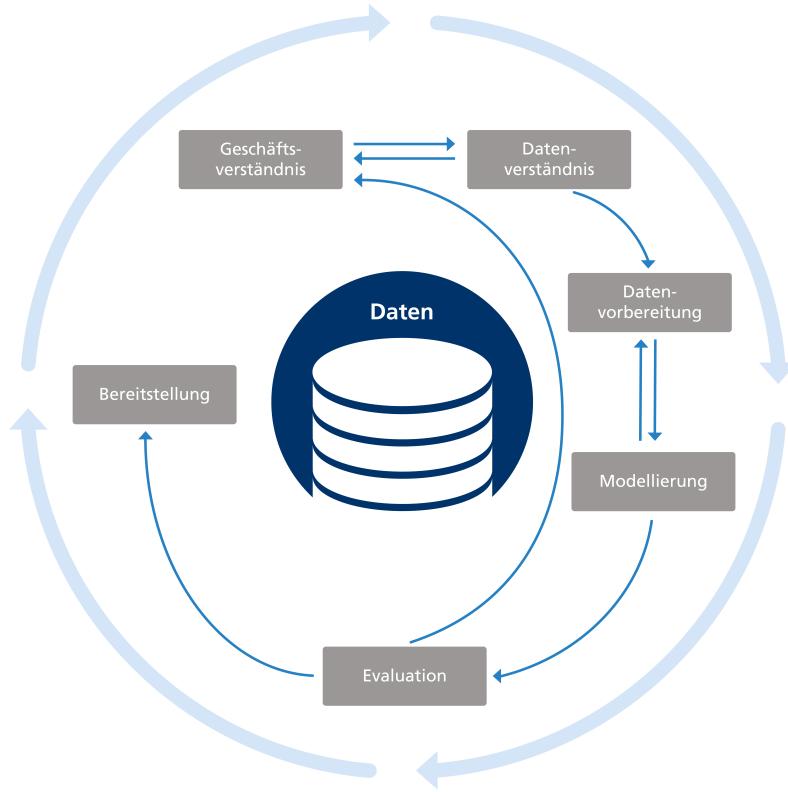


Abbildung 4.1: Übersicht der Phasen von CRISP-DM. Bearbeitete Version aus [18].

siert in digitalen Bildern der in Kapitel 2.2.1 beschriebenen Kameras zu detektieren und die Resultate dem Object Tracking-Algorithmus zur Verfügung zu stellen. Basierend auf den Ergebnissen des Object Tracking-Algorithmus werden die finalen Bewegungsprofile generiert. Die Qualität der Bewegungsprofile ist gebunden an zwei Faktoren, denen der Object Tracking-Algorithmus aufgrund der Kopplung mit einem Object Detection-Algorithmus unterliegt (siehe Kapitel 2.1). Darauf beruht die Motivation zur Erstellung einer detaillierten Gegenüberstellung verschiedener Verfahren in Form eines Benchmarks. Die im Benchmark verwendeten Verfahren gilt es unter Berücksichtigung der in Kapitel 2.1 definierten Anforderungen zu modellieren.

Datenverständnis

Aus Erläuterungen in Kapitel 3.1.1 geht die zentrale Rolle von Trainingsdaten beim Anlernen eines bestimmten Verhaltens nach Methoden des Supervised-Learnings hervor. Zur Realisierung der im Geschäftsverständnis gegebenen Problemstellung wird die Anforderung spezifiziert, einen Object Detection-Algorithmus einzusetzen. Damit besteht das Anlernen im gegebenen Kontext darin, in gegebenen Bildern Instanzen vordefinierter Objektklassen zu lokalisieren und klassifizieren. Die grundlegenden zu detektierenden Objektklassen, wurden unter Rücksprache mit den Verantwortlichen im Projekt bei Fraunhofer IML definiert

als: $K = \{Person, Palette, Hubwagen, Gabelstapler\}$. Diese gilt es, unter Betrachtung der Szenarien in Regalgängen der Lagersysteme mit entsprechenden Trainingsdaten abzudecken. Im Fall der Object Detection bestehen die Trainingsdaten aus digitalen Bildern sowie textuellen Annotationen, die für jedes Bild anzugeben sind [102]. Diese bilden die für das Training notwendigen Ground-Truth-Daten. Jede Annotation hat die unten aufgeführten Informationen zu kodieren [28].

Annotation:**• Auflösung:**

Auflösung des Bildes, gegeben über:

- Breite
- Höhe
- Tiefe (Anzahl der Farbkanäle)

• Liste von Objekten:

Liste der im Bild befindlichen Objekte. Dabei sind für jedes Objekt folgende Attribute anzugeben:

- Klassenlabel: Textueller Bezeichner: l_i
- Bounding-Box: Koordinaten des Rechtecks in Form von zwei Tuplen $(x_{min}, y_{min}), (x_{max}, y_{max})$
Diese definieren die Position der Instanz des definierten Klassenlabels in Bildkoordinaten als linke obere und rechte untere Ecke des Rechtecks.

Für den textuellen Bezeichner ist im Vorfeld des Trainings eine Zuordnung zu definieren, aus der eine ganzzahlige Klassen-ID für jeden Bezeichner hervorgeht. Dies ist notwendig, da während der Implementierung der Algorithmen nur numerische Werte verarbeitet werden.

Das Benchmark in Kapitel 5 und 7 ist als zweistufiger Prozess organisiert. Zunächst findet eine initiale Filterung aller zur Verfügung stehenden Object Detection-Algorithmen statt. Anschließend erfolgt die Modellierung der finalen Modelle. Für die Schritte werden zwei unterschiedliche Bilddatensätze herangezogen. Eine detaillierte Beschreibung der Datensätze erfolgt in Kapitel 4.2. Für weitere Beschreibungen werden die Datensätze mit $D1$ (initialer Datensatz) und $D2$ (Benchmark-Datensatz) bezeichnet. Zudem wird das Konzept der *k-Fold Cross Validation* geschildert und erläutert, wie dieses im Benchmark angewendet wird. Abbildung 4.2 visualisiert die in K spezifizierten Objektklassen. Die jeweiligen Ground-Truth-Bounding-Boxen sind farblich gekennzeichnet: Person in grün, Gabelstapler in blau, Palette in schwarz und Hubwagen in Weiß. Aufgrund des Geheimhaltungsvereinbarung stammen die Bilddaten aus einer Forschungshalle des Fraunhofer IML. Bei den

zugrunde liegenden Daten des Benchmarking handelt es sich jedoch um Daten aus dem Produktivbetrieb einer Lagerhalle eines Projektpartners des Fraunhofer IML.

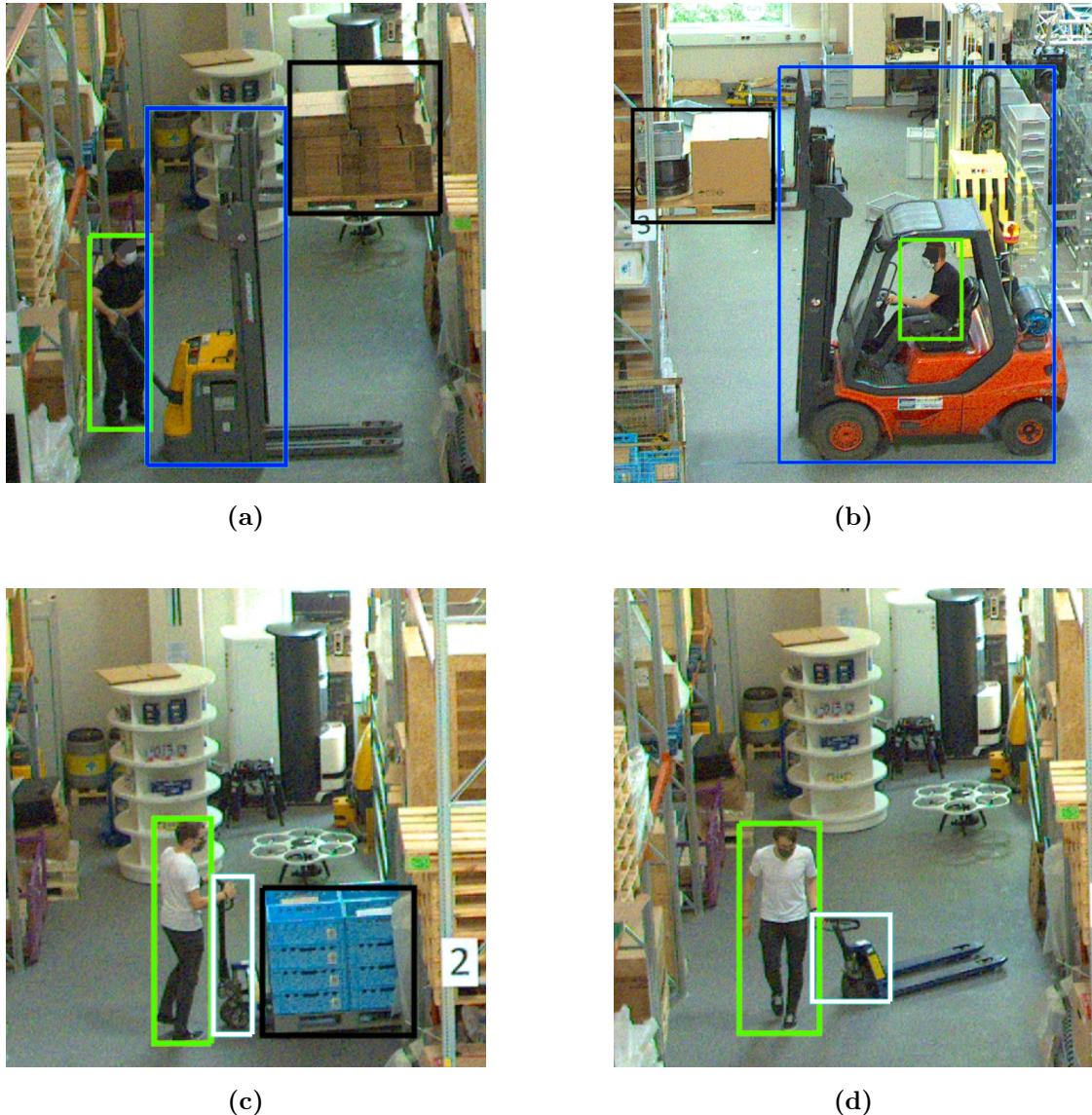


Abbildung 4.2: Gegenüberstellung der zu detektierenden Objektklassen.

Datenvorbereitung

Die Datenvorbereitung für das Training von Object Detection-Algorithmen beläuft sich primär auf die Erstellung der zuvor beschriebenen Annotationen. Annotiert werden die im Geschäftsverständnis definierten Objektklassen. Für das Erstellen der Annotationen können diverse Open-Source Tools herangezogen werden. Im Projekt des Fraunhofer IML wird das Computer Vision Annotation Tool (CVAT) verwendet [91]. Abbildung 4.3 ver-

anschaulicht das Vorgehen bei der Erstellung von Annotationen. Der Bereich innerhalb des roten Rechtecks zeigt das zu annotierende Bild mit bereits annotierten Objektinstanzen. Im Bereich des blauen Rechtecks ist eine Übersicht der erstellten Annotationen gegeben. Der Bereich innerhalb des gelben Rechtecks zeigt Instrumente, die zur Steuerung des Annotationsprozesses verwendet werden.

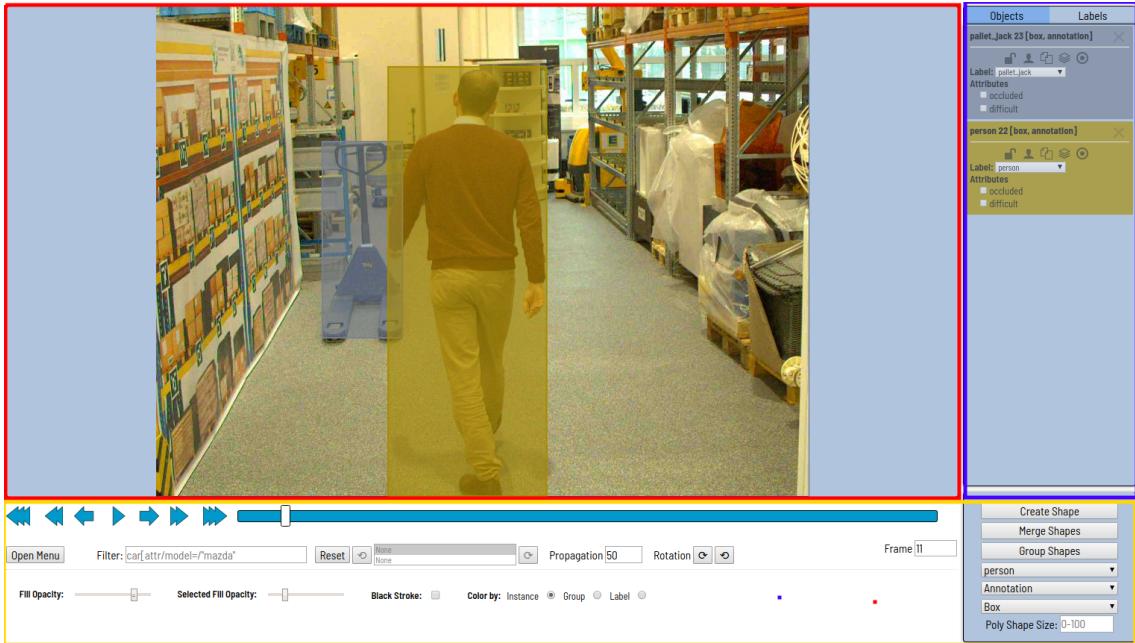


Abbildung 4.3: Erstellung von Annotationen mit CVAT. Entnommen aus [91].

Des Weiteren werden die Eingabebilder im Vorfeld des Trainings normiert und auf eine konfigurierte Auflösung skaliert. Zudem verwendet jeder Algorithmus eine zufällige Auswahl von *Augmentierungen* an. Dabei handelt es sich um eine künstliche Erweiterungen des Datensatzes. Die Bilder werden gezerrt, gedreht, gespiegelt oder mit Bildrauschen versehen. Mit dem Einsatz von Augmentierungen wird das Ziel verfolgt, die Generalisierung der trainierten Modelle zu steigern.

Um robust gegenüber Eingabebildern verschiedener Auflösung zu sein, werden im Kontext der Object Detection Eingabebilder in mehreren Skalen der Auflösung (engl. *Multi-Scale*) verarbeitet. Dies kann sowohl während des Trainingsprozesses (MS-Training) als auch während der Inferenz (MS-Inferenz) angewendet werden. Für ein MS-Training wird dafür in regelmäßigen Abständen die Eingabegröße der CNN-Architektur angepasst. Für eine MS-Inferenz werden die Object Detection-Algorithmen auf allen im Vorfeld der Ausführung definierten Skalen des Eingabebildes ausgeführt. Das finale Ergebnis wird anschließend über ein Voting der Einzelergebnisse bestimmt. In Kapitel 5 wird erläutert, in welcher Form ein MS-Training oder eine MS-Inferenz für Modellierung der Algorithmen im Benchmark angewendet wird.

Modellierung

In Kapitel 5.1 wird in einem ersten Schritt eine Grundgesamtheit von aktuell zur Verfügung stehenden State-of-the-Art Object Detection-Algorithmen aufgebaut. Von der Grundgesamtheit gilt es, für das Benchmarking eine Teilmenge von Algorithmen zu selektieren. Dafür wird unter Anderem ein initiales Training durchgeführt. Dieses zielt darauf ab, die Qualität und Validität der zur Verfügung stehenden Open-Source Implementierungen der Object Detection-Algorithmen zu überprüfen. Weitere Details sind den Erläuterungen in Kapitel 5.3 zu entnehmen. Nachdem eine Menge an Object Detection-Algorithmen bestimmt sowie deren Eignung überprüft worden ist, erfolgt die Modellierung der Object Detection-Algorithmen des Benchmarkings. Dafür gilt es zunächst eine Hyperparameter für die selektierten Object Detection-Algorithmen zu ermitteln. Anschließend erfolgt auf deren Basis eine k-Fold Cross Validation.

Die für das Training von CNNs wichtigen Hyperparameter werden in Kapitel 3.2 erläutert. In Kapitel 6 werden die algorithmischen Konzepte der selektierten Object Detection-Algorithmen für das Benchmarking geschildert. Eine Übersicht der zusätzlich relevanten Hyperparameter für die Architekturen des Benchmarks wird am Ende der jeweiligen Unterkapitel in Kapitel 6.2 gegeben. Des Weiteren wird für das Training der Object Detection-Algorithmen des Benchmarks zudem die Methodik des Transfer Learning (siehe Kapitel 3.2) angewendet. Details zur Durchführung der Hyperparameteroptimierung sowie der anschließenden k-Fold Cross Validation werden in Kapitel 7 diskutiert.

Zur Bewertung der Detektionsgenauigkeit werden die in Kapitel 3.5 definierten Metriken eingesetzt. Für das initiale Training wird lediglich die AP@0.5-Metrik angegeben, um eine grobe Einschätzung der Detektionsgenauigkeit der Algorithmen auf den Daten der Intralogistik zu erlauben.

Evaluation

Die Evaluierung der Modelle ist auf Basis einer k-Fold Cross Validation durchzuführen. Das Vorgehen hierzu wird in Kapitel 4.2 erläutert. In Kapitel 8 findet die finale Auswertung der Ergebnisse statt. An dieser Stelle erfolgt eine explizite Definition der im Benchmark verwendeten Metriken. Alle in der Arbeit angegebenen PR- RC- und AP-Metriken sind Prozentwerte. Daher werden die Ergebnisse basierend auf den Formeln in Kapitel 3.5 nach der Berechnung jeweils mit 100 multipliziert.

Bereitstellung

Die Bereitstellung legt fest, wie und wo die trainierten Object Detection-Algorithmen eingesetzt werden. Für den in Kapitel 1 geschilderten Anwendungsfall wird auf Basis der Ergebnisse des Benchmarks eine Empfehlung für die Auswahl eines Algorithmus geben. Diesen gilt es, zur Extraktion der für den Object Tracking-Algorithmus notwendigen Bounding-

Boxen in das Gesamtsystem zu integrieren. Auf Basis der Ausgabe des Object Tracking-Algorithmus werden Bewegungsprofile erstellt, auf deren Grundlage wiederum Business-KPIs bestimmt werden können.

4.2 Aufbau der Benchmark-Datensätze

Nachfolgend wird der Aufbau der Datensätze $D1$ und $D2$ beschrieben, die als Grundlage für das Training und die Evaluation der Modelle des Benchmarks in Kapitel 5 und 7 herangezogen werden. Die Daten des gegebenen Anwendungsfalls unterliegen dem Kontext von Lagerprozessen der Instralistik. Auf deren Grundlage ist eine automatisierte Identifizierung und Lokalisierung von Objekten durch Anwendung von Deep Learning basierten Object Detection-Algorithmen zu realisieren. Für den Aufbau eines Datensatzes gilt es demnach im Vorfeld zu analysieren, welche Szenarien in den Prozessen der Lagersysteme über entsprechende Trainingsbeispiele abzubilden sind. Für die Arbeit wird die Menge aller Szenarien beschränkt auf Arbeitsabläufe, die innerhalb von Regalgängen abgewickelt werden. Informationen zur Quelle der Daten werden in Kapitel 2.2.1 erläutert. Die zu detektierenden Objektklassen K werden im Datenverständnis (siehe Kapitel 4.1) definiert. Da keine Priorisierung der Objektklassen vorgenommen wird, gilt es für jede Objektklasse in etwa die gleiche Anzahl an Trainingsbeispielen in Form von Bildern und Annotationen zur Verfügung zu stellen.

In Kapitel 5 findet eine Vorfilterung von Object Detection-Algorithmen für das Benchmark statt. Dafür wurde zu Beginn der Arbeit ein initialer Datensatz $D1$ erstellt. Anhand dieses Datensatzes wird lediglich sichergestellt, dass der Programmiercode für das Training und die Inferenz der zur Verfügung stehenden Open-Source Implementierungen eines Object Detection-Algorithmus valide ist. Daher weist der Datensatz $D1$ eine sehr viel niedrigere Komplexität bezüglich der Abdeckung an Szenarien auf. Zudem standen zu Beginn der Arbeit nur eingeschränkt Daten zur Verfügung. Dies beruht auf den nachfolgend beschriebenen Begebenheiten. Innerhalb der Prozessabläufe existieren lange Zeitabschnitte, während denen ein stark reduziertes bis nicht vorhandenes Auftreten von Objektinstanzen zu beobachten ist. Um dennoch qualitative Trainingsdatensätze zu generieren, gilt es mit entsprechenden Heuristiken eine Filterung der aufgezeichneten Bilddaten vorzunehmen. Im Idealfall können dazu bereits trainierte Modelle von Object Detection-Algorithmus eingesetzt werden. Diese Möglichkeit stand jedoch zu Beginn der Arbeit nicht zur Verfügung. Daher wurde eine Heuristik zur Filterung herangezogen, die jedoch nur eingeschränkt bei der Filterung nicht relevanter Bildsequenzen unterstützt hat. Infolge dessen wurden nur begrenzt Daten generiert, welche anschließend manuell auf die Eignung zur Aufnahme in den initialen Datensatz überprüft wurden. Insgesamt umfasst der initiale Datensatz 1.400 Bilder und Annotationen von einer Kamera. Für das Training und die Validierung wurde ein einfacher Split im Verhältnis 4 : 1 vorgenommen. Dies folgt dem in [80] beschriebenen

Vorgehen. Es ist zu berücksichtigen, dass basierend auf den Modellierungen, die auf dem Datensatz $D1$ durchgeführt worden sind, nur eine Abschätzung der Eignung eines Object Detection-Algorithmus getroffen wird. Daher sind komplexere Validierungsmethoden zur Erhöhung der Generalisierung an dieser Stelle zu vernachlässigen. Kann auf dem initialen Datensatz für einen Algorithmus kein Modell trainiert werden, wird der Algorithmus nicht für das Benchmark betrachtet.

Für den Aufbau des Datensatzes $D2$, der als Grundlage für das Training und die Validierung der Modelle im Benchmark eingesetzt worden ist, wurden tiefergehende Zusammenhänge betrachtet. Für eine qualitative Modellierung gilt es eine hohe Generalisierung zu gewährleisten. Demnach sind möglichst viele der Szenarien innerhalb der Prozesse in Regalgängen mit entsprechenden Trainingsdaten abzudecken, um die resultierenden Modelle der Object Detection-Algorithmen im Produktivbetrieb einsetzen zu können. Dafür wurden zunächst die Tageszeiten analysiert, an denen der Großteil der Prozesse abgewickelt wird. Aus Aussagen des Projektpartners geht hervor, dass eine Priorisierung des Zeitabschnitts von 07:00 Uhr bis 00:00 Uhr angenommen werden kann. Daher wird bei der Auswahl von Trainingsdaten der Fokus auf diesen Zeitraum gelegt. Für den Zeitabschnitt von 00:00 Uhr bis 07:00 Uhr werden dementsprechend weniger Trainingsdaten selektiert. Ein weiterer Faktor, der im Zusammenhang einer bilddatengestützten Modellierung beachtet werden muss, ist die Verfälschung von Bildern. Damit gemeint sind Bildeffekte wie beispielsweise Rauschen, Über- / Unterbelichtung oder Verschwommenheit. Diese Effekte haben einen negativen Einfluss auf die Verarbeitung von Bildern im Gesamtkontext der Bildverarbeitung. In [48] werden diesbezüglich vier Objektklassen von Verfälschungen herausgestellt, die unter dem Einsatz von Augmentierungs-Algorithmen künstlich nachgeahmt werden. Anhand der Algorithmen wird der ImageNet-Datensatz künstlich um Bilder erweitert, welche die zuvor beschriebenen Effekte aufweisen. Dadurch wird die Robustheit der auf dem ImageNet-Datensatz trainierten Deep Learning Klassifikationsalgorithmen gesteigert. Für den gegebenen Anwendungsfall ergibt sich lediglich eine Betrachtung verschiedener Lichtverhältnisse, unter denen die Bilder mit den in Kapitel 2.2.1 beschriebenen Kameras aufgezeichnet worden sind. Diese ergeben sich aus einer unterschiedlich starken Sonneneinstrahlung im Lagerhaus. Andere Verfälschungen wie unklare, verrauschte oder verpixelte Bilder sind aufgrund einer entsprechenden Konfiguration der Kameras nicht in den Daten vertreten. Um dennoch unerwarteten Eventualitäten entgegen zu wirken, verwendet jeder Object Detection-Algorithmus eine Bildnormalisierung sowie die in Kapitel 4.1 definierten Methoden zur Augmentierung.

Die Abdeckung verschiedener Lichtverhältnisse ist weitestgehend durch den Einbezug von Bilddaten zu allen Zeitpunkten des Tages sichergestellt. Zur Vergrößerung der Abdeckung von Lichtverhältnissen und der Steigerung der Vielfalt an Prozessszenarien, werden zudem Bilddaten von unterschiedlichen Tagen in den Datensatz aufgenommen. Für das Aufzeichnen der Trainingsdaten fand eine Filterung auf Basis der Modelle statt, die

auf dem Datensatz D_1 trainiert worden sind. Gefiltert wurden Bilddaten, die eine Mindestanzahl von zwei Objektinstanzen aufweisen. Dies garantiert eine Basiskomplexität der aufgezeichneten Szenarien. Zudem wurde explizit darauf geachtet, eine Vermeidung von Sequenzen ähnlicher Prozessabläufe zu gewährleisten. Damit wird sichergestellt, dass die Güte der Aussagen über die Generalisierung, auf Basis des Vergleichs zwischen Trainings- und Validierungsmetriken, nicht herabgesetzt wird. Bildsequenzen ähnlicher Prozessabläufe bergen die Gefahr einer doppelten Abdeckung im Trainings- und Validierungsdatensatz und würden daher unwillkürlich zu ähnlichen Metriken führen.

In Kapitel 3.5 werden die populärsten Datensätze für Object Detection vorgestellt, der MS COCO-Datensatz [69] und der Pascal VOC-Datensatz [121]. Diese weisen unterschiedliche starke Abdeckungen, bezogen auf die durchschnittliche Anzahl von Bildern pro Objektklasse auf. Aus Statistiken des Pascal VOC-Datensatz geht eine Abdeckung von durchschnittlich 500 Trainingsbeispielen (Ground-Truth-Bounding-Boxen) pro Objektklasse hervor [99]. Eine Ausnahme bildet die Objektklasse Person, für die 4194 Trainingsbeispiele vorhanden sind. Trotz fehlender Statistiken über die Verteilung von Trainingsbeispielen für den MS COCO-Datensatz ist ersichtlich, dass die Abdeckung im Durchschnitt sehr viel größer ist (80 Objektklassen mit 115.000 Bildern und Annotationen).

Zur Initialisierung der Gewichte der Object Detection-Algorithmen des Benchmarks wird die Methodik des Transfer Learning (siehe Kapitel 3.2) auf Basis von Modellen, die auf dem Pascal VOC-Datensatz oder MS COCO-Datensatz trainiert wurden, angewendet. Auf Grundlage der Initialisierung werden den Modellen Basis-Features zur Verfügung gestellt, die den Lernprozess erheblich beschleunigen. Da der MS COCO-Datensatz sehr viel mehr Trainingsdaten enthält, stellen Modelle, die auf MS COCO-Datensatz trainiert wurden, qualitativ hochwertigere visuelle Features für die Initialisierung der Modelle des Benchmarks bereit. Daher wird zur Anwendung des Transfer Learning, eine Initialisierung auf Basis von Modellen, die auf dem MS COCO-Datensatz trainiert worden sind, bevorzugt. Zudem sind weniger Trainingsdaten notwendig, da die Objektklassen des MS COCO-Datensatz in Relation zu den im Anwendungsfall definierten Objektklassen des Benchmarks stehen. Die Objektklasse *Person* ist im MS COCO-Datensatz als auch in den Datensätzen D_1 und D_2 vertreten. Die Objektklassen *Hubwagen* und *Gabelstapler* weisen eine starke Ähnlichkeit zu den Objektklassen *Auto*, *Bus* und *Truck* auf. Die Ähnlichkeit ist dabei auf die Schnittmenge an gemeinsamen visuellen Features bezogen. Unter Betrachtung der aufgeführten Eigenschaften bisheriger Datensätze und der Anwendung von Transfer Learning, wird die Anzahl an Trainingsdaten daher auf 6.000 Bilder (1500 pro Kamera) festgelegt. Dies entspricht der Hälfte der Anzahl an Bilddaten des Pascal VOC-Datensatzes. Da jedoch nur ein Fünftel der Anzahl von Objektklassen verwendet werden, wird ungeachtet dessen eine ausreichende Abdeckung von Beispieldaten und Annotation pro Objektklasse zu erwarten.

Zur Optimierung von Hyperparametern wird üblicherweise wie folgt vorgegangen: Es wird ein Datensatz zum Training eines Modells verwendet und anschließend dessen Parameter auf Basis einer Evaluierung auf einem Validierungsdatensatz validiert. Dieses Prozedere wird fortgeführt, bis eine optimale Konfiguration gefunden worden ist. Abschließend erfolgt eine Evaluierung auf Basis eines Testdatensatzes zur Angabe der finalen Metriken. Zur Sicherstellung der Güte bezogen auf Aussagen zur Generalisierung der Modelle, wird alternativ zum zuvor beschriebenen Vorgehen die Methodik der k-Fold Cross Validation angewendet [111]. Bei der k-Fold Cross Validation wird anstelle einer Aufteilung in Trainings-, Validierungs- und Testdatensatz, eine Aufteilung k so genannte *Folds* vorgenommen. Anschließend werden k *Splits* aufgebaut, bei denen das Training jeweils auf $k - 1$ Folds und die Validierung der resultierenden Modelle auf dem k -ten Fold erfolgt. Die finale Auswertung der Metriken ist jeweils als Mittel über die Validierungsmetriken aller Splits anzugeben [111]. Auf Basis dieser Metriken sind anschließend Aussagen über die Generalisierung der Modelle zu formulieren und mögliche Overfittings zu identifizieren.

Üblicherweise wird die k-Fold Cross Validation neben den bereits aufgeführten Aspekten zur Bestimmung robuster Modellparameter mit den besten resultierenden Validierungsmetriken herangezogen. Dafür findet nach der Bestimmung optimaler Modellparameter auf Basis der k-Fold Cross Validation eine finale Auswertung auf einem zusätzlichen Testdatensatz statt [80, 118]. Da die k-Fold Cross Validation mit hohen Berechnungskosten verbunden ist, kann aufgrund der begrenzt zur Verfügung stehende Bearbeitungszeit keine Hyperparameteroptimierung auf Basis einer k-Fold Cross Validation stattfinden. Das Vorgehen wird daher wie folgt adaptiert: Die Hyperparameteroptimierung in Kapitel 7.2 erfolgt auf dem Split $S1$. Anschließend wird eine k-Fold Cross Validation mit den bestimmten Hyperparametern durchgeführt, um die eingangs geforderte Güte der Aussagen zu gewährleisten. Abbildung 4.4a stellt das in der Literatur gängige Vorgehen der Hyperparametersuche dar, Abbildung 4.4b das für die Arbeit angepasste Vorgehen. Gezeigt wird die Aufteilung des Datensatzes in die geschilderten Folds und Splits sowie die jeweilige Grundlage zur Bestimmung von Hyperparametern und der finalen Evaluation der Metriken.

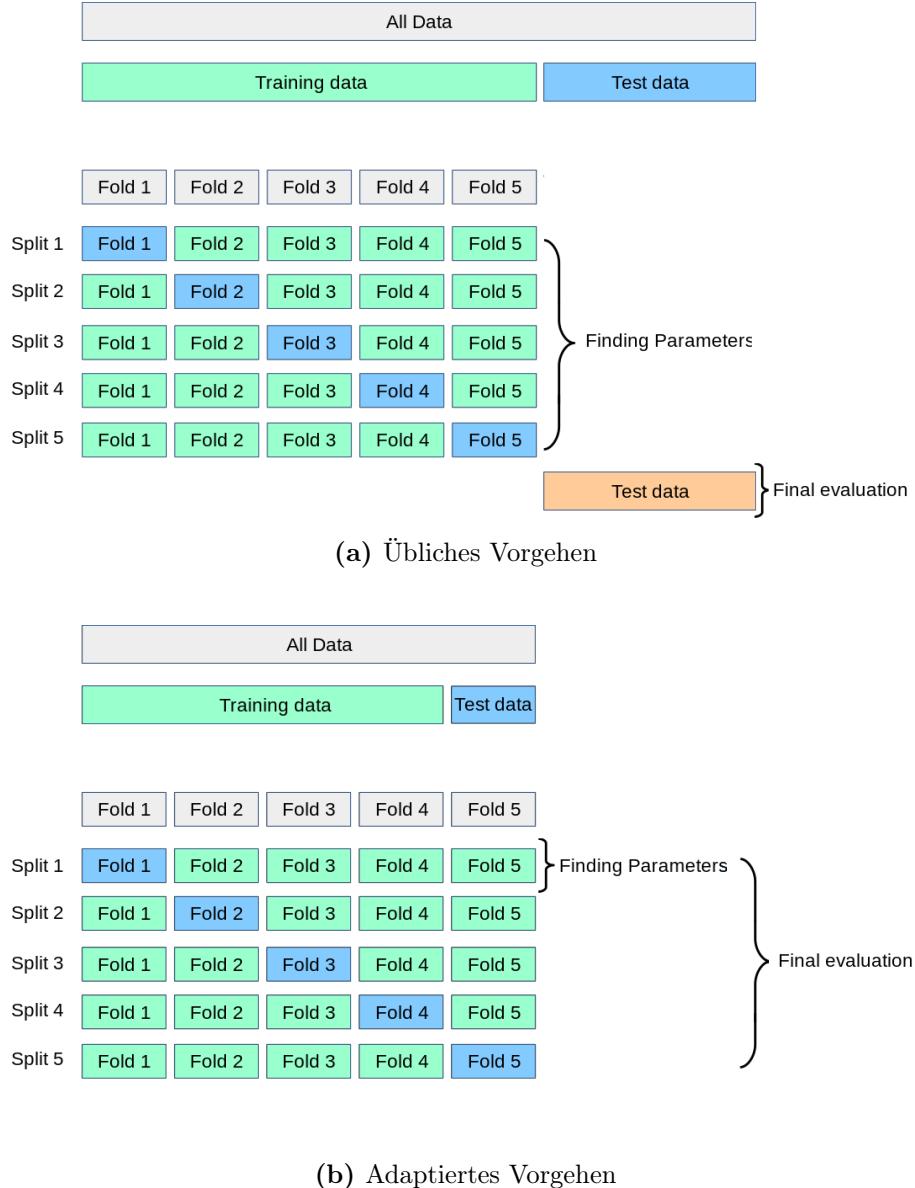


Abbildung 4.4: Illustration der adaptierten k-Fold Cross Validation im Benchmark.

Empirisch belegt kann nach [54] ein Wert von $k = 5$ oder $k = 10$ angenommen werden. Für den Datensatz D2 wird $k = 5$ gewählt, da für $k = 10$ die Modellierung den zeitlichen Rahmen der Arbeit übersteigen würde. Beim Aufbau des Datensatzes D2 wurde explizit darauf geachtet, eine ausgewogene Menge an Szenarien abzudecken. Aufgrund dessen werden für jeden der fünf Splits die Bilddaten nach dem Zufallsprinzip ausgewählt. Dies gewährleistet eine höhere potentielle Generalisierung der Modelle [111]. Tabelle 4.1 listet die Verteilung der Trainingsbeispiele für jeden der fünf Splits auf. Daraus geht eine Gesamtanzahl von 5777 Trainingsbildern hervor. Die Diskrepanz von 223 Bildern beruht auf fehlerhaften Daten, die erst während des Annotationsprozesses aufgefallen sind. Jede Zeile spezifiziert die

Verteilung der Anzahl von Instanzen der definierten Objektklassen auf die gegebenen Trainingsbilder. Die vorletzte Spalte der Tabelle gibt die jeweilige Zeilensumme an. Die Größen sind bezogen auf die *Pixelfläche (PF)* der annotierten Bounding-Boxen, analog zu den AP-Metriken des MS COCO-Datensatzes. Um eine Zuordnung der gleichen PF zwischen denen des MS COCO-Datensatz und des Datensatzes D2 zu erlauben, wurden die annotierten Bounding-Boxen des Datensatzes D2 entsprechend skaliert. Für Größe S gilt $PF < 32^2$, für Größe M gilt $32^2 < PF \leq 96^2$ und für Größe L gilt $PF > 96^2$. In der letzten Zeile ist die jeweilige Gesamtanzahl an Bildern der entsprechenden Splits spezifiziert. Es ist anzumerken, dass pro Bild mehrere Instanzen der gleichen Objektklasse auftreten können. Daher kann die Gesamtanzahl an Instanzen für eine Objektklasse die Gesamtanzahl an Bildern übersteigen. Um eine einheitliche Modellierung der Szenarien auf allen Splits sicherzustellen, wird eine Gleichverteilung der Anzahl an Trainingsbeispielen pro Objektklasse pro Split gefordert. Zur Kontrolle, ob aus der zufälligen Aufteilung eine Ausgewogenheit der Trainingsbeispiele sicher gestellt ist, ist daher in der letzten Spalte die Standardabweichung σ der jeweiligen Zeile angegeben. Trotz einer daraus resultierenden nicht optimalen Gleichverteilung wird anhand dessen zunächst die Ausgewogenheit der Trainingssplits bestätigt. Zur späteren Kontrolle dieser Aussage gilt es, die Standabweichung für die Validierungs-metriken der durchgeführten k-Fold Cross Validation zu bestimmen.

		Instanzen pro Split					Akkumuliert	σ
		S1	S2	S3	S4	S5		
Person	Größe S	771	773	744	715	670	3673	43.14
	Größe M	712	654	657	684	631	3338	31.13
	Größe L	0	0	0	0	0	0	0.00
	Gesamt	1483	1427	1401	1399	1301	7011	65.95
Palette	Größe S	583	556	584	571	530	2824	22.51
	Größe M	742	675	743	716	679	3555	32.90
	Größe L	33	16	25	22	25	121	6.14
	Gesamt	1358	1247	1352	1309	1234	6500	57.69
Hubwagen	Größe S	633	656	613	628	574	3104	30.38
	Größe M	255	191	223	219	219	1107	22.73
	Größe L	0	1	0	2	0	3	0.89
	Gesamt	888	848	836	849	793	4214	34.04
Gabelstapler	Größe S	21	23	21	11	32	108	7.47
	Größe M	125	163	164	109	135	696	24.05
	Größe L	49	51	43	25	42	210	10.25
	Gesamt	195	237	228	145	209	1014	36.21
Gesamtanzahl Bilder		1162	1150	1160	1149	1156	5777	5.81

Tabelle 4.1: Auflistung von Statistiken für den Datensatz D2.

Kapitel 5

Vorauswahl der Benchmark-Algorithmen

In Kapitel 2.1 werden Anforderungen definiert, die an Object Detection-Algorithmen gestellt werden, um als Grundlage für Object Tracking-Algorithmen herangezogen werden zu können. Dabei sind zwei wesentliche Faktoren herausgestellt worden, welche die Performance der Object Tracking-Algorithmen beeinflussen: Die Genauigkeit und die Laufzeit des zugrunde liegenden Object Detection-Algorithmen. Des Weiteren wurde der Konflikt geschildert, welcher bei der Optimierung der beiden Faktoren auftritt.

Auf Grundlage der im Benchmark erlangten Erkenntnisse wird eine Entscheidungsbasis aufgebaut. Diese wird als Anhaltspunkt verwendet, um aufzuzeigen, welcher Object Detection-Algorithmus die höchste Eignung zur Bestimmung von Bounding-Boxen aufweist, die notwendig für ein Object Tracking sind. Da ein Vergleich aller aktuell bestehender Verfahren zeitlich bedingt nicht möglich ist, wird im folgenden Kapitel eine Vorauswahl an Algorithmen getroffen, die im Benchmark verwendet werden.

Zunächst wird der Aufbau von Tabelle B.1 (siehe Anhang) geschildert, aus der die *Grundgesamtheit* der aktuell zur Verfügung stehenden State-of-the-Art Algorithmen hervorgeht. Diese bildet den Ausgangspunkt für die Erarbeitung einer fundierten Entscheidungsbasis. Anschließend folgt die Beschreibung von Filtern, die auf die Grundgesamtheit angewendet werden, um eine finale Auswahl von Object Detection-Algorithmen zu erhalten. Abschließend werden die hinter den ausgewählten Verfahren stehenden algorithmischen Prinzipien vorgestellt.

5.1 Ermittlung der Grundgesamtheit

Die in Kapitel 3.5 beschriebenen Benchmark-Datensätze $D1$ und $D2$ bilden die Basis, um Object Detection-Algorithmen anhand definierter Metriken zu vergleichen. Der populärste Datensatz für die Problemstellung der Object Detection ist der MS COCO-Datensatz.

Die Popularität ist an der Anzahl der Veröffentlichungen gemessen, in denen Metriken für Benchmark-Datensätze aus dem Bereich der Object Detection angegeben sind. Der MS COCO-Datensatz erzielt dabei die höchste Schnittmenge aller Artikel, die sich mit Object Detection-Algorithmen befassen. Eine Übersicht der aktuellen State-of-the-Art Algorithmen, gemessen an den Metriken des MS COCO-Datensatzes, ist als tabellarische Auflistung auf der Website Paperswithcode [97] zu finden. Diese zielt darauf ab, Artikel, Programmiercode und Evaluationsmetriken ausgewählter Problemstellungen des Forschungsfeldes Machine Learning als freie Ressource zur Verfügung zu stellen [97]. Die Evaluationsmetriken sind dabei jeweils an Benchmark-Datensätze gebunden.

Der MS COCO-Datensatz ist aufgeteilt in drei Teildatensätze: Training, Validierung und Test. Letzterer wird mit MS COCO test-dev Datensatz bezeichnet und als Basis für die Bestimmung der Metriken aktueller State-of-the-Art Verfahren für Object Detection herangezogen. Die Rangfolge der State-of-the-Art Verfahren richtet sich nach der in Kapitel 3.5 spezifizierten AP_{coco} -Metrik. Tabelle B.1 wurde basierend auf einer ausführlichen Recherche aufgebaut. Diese listet alle für das Benchmark zur Debatte stehenden Verfahren und wird nachfolgend als Grundgesamtheit bezeichnet. In der Tabelle sind folgende Attribute angegeben: Meta- und Backbone-Architektur des jeweiligen Object Detection-Algorithmus, das Ergebnis der sechs AP-Metriken (siehe Tabelle 3.3) auf dem MS COCO test-dev Datensatz, die FPS sowie die für die Messung der FPS zugrunde liegende GPU-Architektur und die Angabe ob ein MS-Training oder eine MS-Inferenz angewendet wird (siehe Kapitel 4.1). Aus Platzgründen sind MS-Training und MS-Inferenz in Tabelle B.1 mit MS-T und MS-I abgekürzt.

Bei der Recherche zum Aufbau der Tabelle wurden folgende Quellen berücksichtigt: Zunächst wurden alle Verfahren der Top 50 der tabellarische Auflistung auf der Website Paperswithcode für den MS COCO test-dev Datensatz vom Stand vom 01.02.2020 [98] übernommen. Anschließend wurde für jedes dieser Verfahren jede Ausprägung der Kombination aus Meta- und Backbone-Architektur aus den jeweiligen Artikeln in die Tabelle übertragen. Jeder Artikel listet die erzielten AP-Metriken ebenfalls als tabellarischen Vergleich der State-of-the-Art Verfahren. Jedes Verfahren, welches Bestandteil dieser tabellarischen Auflistung ist, wurde ebenfalls in die Tabelle übernommen. Zuletzt wurde speziell nach realzeitfähigen Verfahren recherchiert, da diese in Bezug auf die Laufzeit ein großes Potential für die Performance von Object Tracking-Algorithmen bieten.

Für jedes in der Tabelle aufgeführte Verfahren sind Referenzen angegeben. Dabei handelt es sich im Wesentlichen um die Originalartikel der entsprechenden Object Detection-Algorithmen. Darüber hinaus sind für manche Verfahren zusätzliche Quellen angegeben. Die zusätzlichen Quellen verweisen auf Metriken und Laufzeiten, welche für die entsprechende Architektur gefunden wurden, jedoch nicht im Originalartikel aufgeführt sind.

5.2 Aufbereitung der Grundgesamtheit

Für die Vorauswahl gilt es Filter zu definieren, die auf die Grundgesamtheit (Tabelle B.1) angewendet werden, um eine Auswahl von Object Detection-Algorithmen für das Bechmarking in Kapitel 7 zu selektieren. Die maßgeblichen Kennzahlen, die zur Filterung herangezogen werden, sind die in Tabelle 3.3 aufgeführten AP-Metriken des MS COCO-Datensatzes und die FPS. Dabei dienen die AP-Metriken als Grundlage, um eine Aussage über die Genauigkeit der Verfahren treffen zu können und die FPS als Maß für die Laufzeit der Verfahren. Grundsätzlich schreibt der MS COCO-Benchmark die Berechnung aller sechs AP-Metriken vor, jedoch ist die AP_{coco} -Metrik (Formel 3.8) die einzige, die für alle Object Detection-Algorithmen der Grundgesamtheit vollständig angegeben ist. Daher wird die Genauigkeit allein unter der Betrachtung der AP_{coco} -Metrik beurteilt. Die in Tabelle B.1 aufgeführten FPS sind jeweils bezogen auf die zugrunde liegende GPU-Architektur, welche für die Inferenz des gegebenen Verfahrens genutzt worden ist. Dabei bezieht sich die Inferenz auf die Ausführung der Object Detection-Algorithmen unter der Eingabe von Einzelbildern eines gegebenen Bilddatenstroms. Für die Inferenz sind sieben verschiedene GPUs in den Artikeln der Object Detection-Algorithmus angegeben: NVIDIA Titan X Pascal, NVIDIA Titan Xp Pascal, NVIDIA Titan V Pascal, NVIDIA Tesla P100, NVIDIA Tesla V100, NVIDIA Tesla M40. Um eine faire Filterung der Object Detection-Algorithmen der Grundgesamtheit zu gewährleisten, gilt es, eine Umrechnung für die FPS der verschiedenen GPU-Architekturen in normiertes Maß vorzunehmen.

In [52] wurden Object Detection-Algorithmen bereits auf den Konflikt zwischen der Laufzeit und der Genauigkeit untersucht. Als Alternative zur nativen Laufzeit der Verfahren haben die Autoren die *Floating Point Operations (FLOPs)* (bezogen auf Additionen und Multiplikationen) verwendet. Die FLOPs bilden ein plattformunabhängiges Maß zur Bewertung der Berechnungskomplexität. Es wird jedoch angemerkt, dass ein linearer Zusammenhang zwischen FLOPs und der tatsächlichen Laufzeit der Algorithmen nicht garantiert werden kann [52]. Dies ist auf eine Reihe von Problemfaktoren zurückzuführen: Caching, I/O, Hardwareoptimierung etc. Schlussfolgerungen, die für die zu vergleichenden Verfahren bezüglich der FLOPs getroffen werden, lassen sich damit nicht in Schlussfolgerungen bezüglich der Laufzeit der Algorithmen transferieren. Dies ist jedoch relevant, um die Eignung für die Kombination eines Object Detection-Algorithmus mit einem Object Tracking-Algorithmus zu beurteilen, betrachtet im Kontext des Einsatzes im Projekt bei Fraunhofer IML. Für die Filterung wird daher die nativ gemessene Laufzeit und nicht die FLOPs herangezogen. Die Laufzeit wird dabei über die FPS abgebildet. Zudem sind für die wenigsten Object Detection-Algorithmen die FLOPs in den Artikeln angegeben, was den Einsatz der FPS zur Beurteilung der Laufzeit bestärkt.

Neben der GPU-Architektur ist die Laufzeit von den für die Implementierung genutzten Frameworks abhängig. Dafür ist zum Einen das verwendete Deep Learning-Framework,

beispielsweise Tensorflow [1] oder Pytorch [100], sowie die Version des entsprechenden Frameworks verantwortlich. Dies geht aus Analysen in [122] und [122] hervor, welche die Laufzeiten verschiedener Deep Learning-Frameworks gegenüberstellen. Zum Anderen ist die Version von CUDA [87] und cuDNN [88] ein ausschlaggebender Faktor. CUDA ist eine Hard- und Software-Architektur, die es Programmiersprachen wie C, C++, Fortran, Python oder MATLAB erlaubt, GPU-beschleunigten Programmiercode parallel auszuführen [87]. Zudem bietet die cuDNN Bibliothek stark optimierte Implementierungen für Standardroutinen von Deep Learning-Architekturen (Convolution, Pooling, ...). Die von CUDA und cuDNN zur Verfügung gestellten Optimierungen und GPU-Beschleunigungen unterstehen einer fortwährenden Weiterentwicklung. Damit ergeben sich für verschiedene Versionen verschiedene Optimierungen, welche sich in Laufzeitunterschieden äußern. Informationen sind diesbezüglich in den Release-Notes der jeweiligen Framework-Version zu finden [86]. Da CUDA und cuDNN die Schnittstellen der Deep Learning-Frameworks sind, um GPU-beschleunigten Programmiercode zu nutzen, haben Laufzeitunterschiede zwischen den Versionen von CUDA und cuDNN einen direkten Einfluss auf die Performance der Deep Learning-Frameworks. Zudem werden die Deep Learning-Frameworks selbst fortwährend optimiert, woraus ebenfalls Laufzeitunterschiede zwischen den Versionen der Deep Learning-Frameworks resultieren.

Für das Benchmarking wird die Betrachtung der Laufzeiten in einem normiertem Maß gefordert. Zur Bestimmung von Umrechnungsfaktoren für verschiedene GPUs sind, neben der GPU-Architektur, Informationen bezüglich des für die Implementierung verwendeten Deep Learning-Frameworks sowie der Version von CUDA und cuDNN notwendig. In den Artikeln der Object Detection-Algorithmen sind jedoch nur für die wenigsten Verfahren Informationen diesbezüglich vorhanden. Daher wurden genäherte Umrechnungsfaktoren bestimmt, die es erlauben, die FPS für verschiedene GPU-Architekturen zu vergleichen. Grundlage für die Bestimmung der Umrechnungsfaktoren ist ein Benchmark der ETH Zürich [22]. Dieses stellt die Laufzeiten von einer Vielzahl von Deep Learning-Algorithmen auf verschiedenen GPU-Architekturen tabellarisch gegenüber. Basierend auf den aufgeführten Laufzeiten wurden die Umrechnungsfaktoren wie folgt berechnet: Das Referenzsystem ist, wie in Kapitel 2 geschildert, eine NVIDIA Geforce RTX 2080 Ti. Damit sind alle in Tabelle B.1 angegebene FPS, in FPS der NVIDIA RTX 2080 Ti umzurechnen. In einem ersten Schritt werden alle Laufzeiten (der Inferenz) für die verschiedenen GPU-Architekturen aus der Tabelle in [22] extrahiert. Anschließend wird für jede GPU der Mittelwert über die Laufzeiten gebildet. Die jeweiligen Umrechnungsfaktoren ergeben sich aus der Division der Mittelwerte für die entsprechende GPU und dem Mittelwert für die NVIDIA RTX 2080 Ti. Die daraus resultierenden Umrechnungsfaktoren sind in Tabelle 5.1 aufgeführt. Für die NVIDIA Titan V ist keine Laufzeit angegeben. Aus Ähnlichkeiten in der GPU-Architektur, die aus [3, 4, 42] hervorgehen, wird daher näherungsweise für die NVIDIA Titan V der gleiche Faktor wie für die NVIDIA Tesla V100 verwendet. Um den Faktor für

die NVIDIA Tesla M40 zu berechnen, werden die Laufzeiten der NVIDIA Geforce GTX Titan X herangezogen. Diese sind nahezu baugleich und unterscheiden sich nur in der Art des Einsatzzweckes: Desktop- oder Workstation-GPU [85]. Abschließend ist anzumerken, dass es sich bei den Umrechnungsfaktoren lediglich um eine Näherung handelt. Diese bietet die Möglichkeit eine faire Betrachtung der FPS für unterschiedliche GPU-Architekturen, ist jedoch nicht gleichzusetzen mit einer tatsächlichen Laufzeitmessung für die verschiedenen GPUs.

GPU-Architektur	Faktor relativ zu NVIDIA RTX 2080 Ti
NVIDIA Titan X Pascal	0,6614
NVIDIA Tesla M40	0,5100
NVIDIA Titan Xp Pascal	0,8079
NVIDIA Titan V Pascal	1,0370
NVIDIA Tesla P100	0,6725
NVIDIA Tesla V100	1,0370

Tabelle 5.1: Faktoren zur Umrechnung der FPS zwischen der NVIDIA RTX 2080 Ti .

5.3 Filterung der Grundgesamtheit

In diesem Kapitel wird zunächst ein Konzept ausgearbeitet, um eine Teilmenge der Object Detection-Algorithmen aus der Grundgesamtheit zu selektieren. Daraus resultiert eine Menge von Filtern, die auf die Grundgesamtheit angewendet werden. Anschließend erfolgt die Beschreibung der Anwendung der ermittelten Filter. Abschließend werden die Ergebnisse der Filterung zusammenfassend in Tabelle 5.3 gegenübergestellt.

5.3.1 Erarbeitung eines Konzepts zur Filterung

Für das Benchmark in Kapitel 7 besteht die Anforderung, eine Auswahl von Object Detection-Algorithmen zu treffen, die den bestehenden Speed-Accuracy Trade-Off mit repräsentativen Verfahren abdeckt. Nachfolgend werden Filter definiert, um eine Filterung der in Kapitel 5.1 ermittelten Grundgesamtheit durchzuführen. Dafür werden die FPS zur Einschätzung der Laufzeit und die AP_{coco}-Metrik zur Einschätzung der Detektionsgenauigkeit herangezogen. Hintergrund ist die Bereitstellung einer Entscheidungsbasis, um ein für Object Tracking passendes Verfahren zu bestimmen. Da die exakten Laufzeitanforderungen der Object Detection-Algorithmen bezüglich der Kopplung an einen Object Tracking-Algorithmus nicht bekannt sind, werden die Object Detection-Algorithmen der Grundgesamtheit für mehrere Schwellenwerte der FPS betrachtet. Die Menge der Schwellenwerte wird für nachfolgende Beschreibung mit S bezeichnet, die Menge der Object Detection-Algorithmen in der Grundgesamtheit mit V . Für jeden Schwellenwert $s_n \in S$ wird das Verfahren v_i

selektiert, welches die höchste AP_{coco} -Metrik aufweist. Der Ausgangspunkt für die Auswahl der Schwellenwerte, die zur Filterung herangezogen werden, ist das in Abbildung 5.1 aufgeführte Histogramm. Daraus geht die Verteilung der FPS aller Verfahren $v_i \in V$ hervor. Gezeigt wird die akkumulierte Anzahl von Verfahren über den ganzzahligen Wert der FPS. Der Großteil der Verfahren weist eine FPS zwischen 0 und 30 auf. Verfahren mit einer FPS größer als 70 sind wenig bis gar nicht vertreten. Daher genügt es, die FPS stufenweise in Schritten der Größe 5 zu betrachten. Da für einen spezifizierten Schwellenwert jeweils nur ein Verfahren für das Benchmark selektiert wird, würde ein größere oder kleinere Auswahl der Schrittgröße entweder zu einer zu feingranularen Betrachtung oder einer Filterung potentiell wertvoller Verfahren führen. Die Menge der zu verwendenden Schwellenwerte wird damit definiert als $S = \{0, 5, \dots, 65, 70\}$ und die jeweiligen Filter als $f_n = FPS(v_i) > s_n \forall v_i \in V$ mit $s_n \in S$.

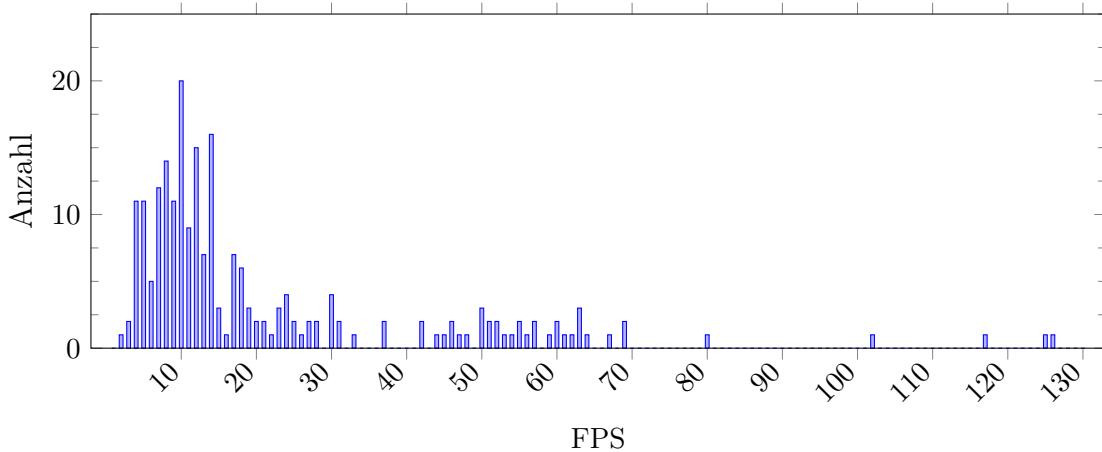


Abbildung 5.1: Histogramm der ganzzahligen Werte der FPS.

Als Alternative zur Filterung anhand von Schwellenwerten können die FPS für feste Wertebereiche betrachtet werden. Anstelle der Filter $f_n = FPS(v_i) > s_n$, werden im Fall der Wertebereiche die Filter $f_n = s_n \leq FPS(v_i) < s_{n+1}$ verwendet. Nachteil an der Betrachtung von Wertebereichen ist die exklusive Filterung der Verfahren. Diese kann sich wie folgt äußern: Ein Verfahren v_i mit 12 FPS und einer AP_{coco} -Metrik von 43 wird als bestes Verfahren für den Wertebereich $w_1 = 10 \leq FPS(v_i) < 15$ bestimmt. Ein Verfahren v_j mit 17 FPS und einer AP_{coco} -Metrik von 46 wird als bestes Verfahren für den Wertebereich $w_2 = 15 \leq FPS(v_i) < 20$ bestimmt. Damit wird für den Wertebereich $10 \leq FPS(v_i) < 15$ ein Verfahren bestimmt, welches eine niedrigere AP_{coco} -Metrik aufweist als das Verfahren, was für den Wertebereich $15 \leq FPS(v_i) < 20$ selektiert wird. Im Fall der Filterung anhand von Schwellenwerten, $s_1 = 10$ und $s_2 = 15$ wird das Verfahren v_j sowohl für den Schwellenwert s_1 als auch für den Schwellenwert s_2 betrachtet und auf Basis der AP_{coco} -Metrik als genauestes Verfahren ausgewählt.

Für die Filterung auf Basis von Schwellenwerten wird damit immer das Verfahren be-

stimmt, welches für die zu betrachtende FPS die höchste Genauigkeit bemessen an der AP_{coco}, aufweist. Eine Ausnahme bildet die Filterung auf Basis des ersten Schwellenwerts $s_0 = 0$. Dieser dient der Ermittlung des Verfahrens mit der insgesamt höchsten AP_{coco}-Metrik, bezogen auf alle Verfahren der Grundgesamtheit. Es sind demnach insbesondere auch Verfahren zu berücksichtigen, für die keine Laufzeit angegeben ist. Das resultierende Verfahren dient während der Evaluierung in Kapitel 8 als Referenz, um die gefilterten Verfahren mit der aktuell maximal erreichbaren AP_{coco}-Metrik zu vergleichen. Dieses Verfahren wird dementsprechend für nachfolgende Erläuterungen als *Referenzarchitektur* definiert.

Nachdem ein Verfahren selektiert worden ist, erfolgt eine Überprüfung der Eignung des Verfahrens auf Basis der zur Verfügung stehenden Open-Source Implementierung. Es muss sowohl Programmiercode für den Trainingsprozess, als auch die Inferenz vorhanden sein. Aufgrund der eingeschränkten Bearbeitungszeit werden keine Verfahren, auf Basis der im entsprechenden Artikel geschilderten Architektur, selbst implementiert. Es gilt lediglich die zur Verfügung stehenden Implementierungen in das Gesamtsystem zu integrieren. Dies umfasst im Wesentlichen das Anpassen des Programmiercodes, der für das Laden der Trainingsbilder und Annotationen zuständig ist. Zur Validierung des Programmiercodes für die Inferenz werden Modelle herangezogen, die größtenteils auf dem MS COCO-Datensatz, vereinzelt aber auch auf dem Pascal VOC-Datensatz, vorgenommen worden sind. Basierend auf den validierten vorgenommenen Modellen, kann im Trainingsprozess die Methodik des Transfer Learning (siehe Kapitel 3.2.1) angewendet werden, um die Gewichte der Modelle zu initialisieren. Anschließend erfolgt ein initiales Training auf dem Datensatz D1 zur Validierung des entsprechenden Programmiercodes. Zur Einordnung der Genauigkeit von Object Detection-Algorithmen im Kontext von Bilddaten und Objektklassen der Informatik, werden die aus dem initialen Training resultierenden Modelle unter Angabe der AP@0.5-Metrik in Tabelle 5.3 gegenübergestellt. Eine genaue Analyse der Ergebnisse findet unter Anwendung einer k-Fold Cross Validation in Kapitel 9 statt.

Das Deep Learning-Framework Pytorch bietet die größte Überschneidung an zur Verfügung stehenden Open-Source Implementierungen für Grundgesamtheit. Daher wurde die, zum Zeitpunkt der Implementierung, aktuelle Pytorch Version 1.3.1 ausgewählt. Damit ist ein einheitlicher und fairer Vergleich der Laufzeiten gewährleistet. Gekoppelt daran wird für CUDA die Version 10.2 und für cuDNN die Version 7.6 festgelegt. Alle nachfolgend beschriebenen Implementierungen unterliegen den Frameworks in der zuvor spezifizierten Version.

In Kapitel 4.1 wird das Prinzip eines MS-Trainings und einer MS-Inferenz geschildert. Durch die mehrfache Ausführung der Object Detection-Algorithmen für eine MS-Inferenz entsteht ein großer Overhead. Dieser wirkt sich negativ auf die Laufzeit der Algorithmen aus. Daher werden für die Filterung nur Verfahren in Betracht gezogen, die keine MS-Inferenz nutzen.

In Kapitel 2 werden Grundanforderungen an das Benchmark gestellt. Unter Betrach-

tung der zuvor geschilderten Aspekte werden die Grundanforderungen zu den in Tabelle 5.2 aufgeführten Anforderungen konkretisiert. Diese gilt es für die nachfolgende Filterung zu Berücksichtigen.

ID	Beschreibung
1	Höchste AP _{coco} -Metrik für ein gegebene Menge von Verfahren die sich aus der Anwendung einer Filters ergeben
2	Angabe einer Laufzeit zur Vorfilterung von Verfahren
3	Verfügbarkeit einer qualitativen Open-Source Implementierung
4	Ausschluss von Verfahren die MS-Inferenz nutzen

Tabelle 5.2: Anforderungen für das Benchmark

5.3.2 Durchführung der Filterung

Die folgenden Abschnitte beschreiben die Anwendung der in ermittelten Menge an Schwellenwerten S . Final gehen daraus die Object Detection-Algorithmen für das in Kapitel 7 durchzuführende Benchmark hervor. Für jeden definierten Schwellenwert $S_n \in S$ wird ein Verfahren unter Beachtung der in Tabelle 5.2 gegebenen Anforderungen ausgewählt. Die nachfolgend verwendeten Bezeichner der Meta- und Backbone-Architekturen stammen aus den Originalartikeln und der Konfiguration der jeweiligen Implementierung. Am Ende der Filterung werden für den weiteren Verlauf kürzere Synonyme der ausgewählten Verfahren angegeben.

FPS ≥ 0 : Bestimmung der Referenzarchitektur

Zur Bestimmung einer Referenzarchitektur wird der Schwellenwert $s_0 = 0$ verwendet. Dies resultiert in einer Filterung, rein bezogen auf den höchsten Wert der AP_{coco}-Metrik. Daraus ergibt sich die Meta-Architektur *EfficientDet-D7 + AA* mit der Backbone-Architektur *EfficientNet-B7*. Für die EfficientDet-Architektur ist keine Originalimplementierung der Autoren des Artikels verfügbar. Daher wird eine alternative Open-Source Implementierung verwendet [79]. Eine Validierung des Inferenz-Programmiercodes auf Basis der zur Verfügung stehenden vtrainierten Modelle konnte erfolgreich durchgeführt werden. Je doch liefert der Trainings-Programmiercode fehlerhafte Modelle. Diese äußern sich während der Inferenz. Die auf dem initialen Datensatz trainierten Modelle detektieren lediglich statische Boxen, die für jedes Eingabebild gleich sind. Abbildung 5.2 veranschaulicht dieses Fehlverhalten. Aufgeführt wird ein Modell, welches auf dem Datensatz *D1* trainiert worden ist.

Da der Inferenz-Programmiercode erfolgreich validiert worden ist, lässt sich der Fehler rein auf das Training zurückführen. Eine andere Open-Source Implementierung [123] bot zum



Abbildung 5.2: Inferenzfehler von *EfficientDet-D7 + AA*. Entnommen aus der Visualisierung der Inferenz.

Zeitpunkt der Auswertung lediglich den Programmiercode für *EfficientDet-D0* mit der Backbone-Architektur *EfficientNet-B0*. Es handelt sich prinzipiell um den gleichen Object Detection-Algorithmus, jedoch wird eine andere Backbone-Architektur verwendet. Damit besteht keine Möglichkeit, ein Modell für *EfficientDet-D7 + AA* mit *EfficientNet-B7* auf benutzerdefinierten Daten zu trainieren, weshalb dieses Verfahren für die Auswahl des Benchmarks ausgeschlossen wird.

Die nächsthöhere AP_{coco}-Metrik weist die Meta-Architektur *Hybrid Task Cascade (HTC)* auf, welche *X-101-64x4d FPN DCN* als Backbone-Architektur verwendet. Dabei handelt es sich um keinen reinen Object Detection-Algorithmus, sondern um eine Architektur, die neben der Object Detection eine Instanzsegmentierung realisiert. Die Open-Source Implementierung bietet die Möglichkeit, den Algorithmus so zu konfigurieren, dass nur der Object Detection-Algorithmus des HTC-Verfahrens verwendet wird. Ein initiales Training wurde erfolgreich durchgeführt. Damit wurde das Verfahren *HTC X-101-64x4d FPN DCN* als Referenzarchitektur für das Benchmark ausgewählt.

FPS ≥ 5

Für den Schwellenwert $s_1 = 5$ weist *EfficientDet-D6 + AA* mit *EfficientNet-B6* die höchste AP_{coco}-Metrik auf. Darauf folgt *EfficientDet-D5 + AA* mit *EfficientNet-B5*. Die Implementierung beider Verfahren beruht auf dem gleichen Programmiercode wie dem von *EfficientDet-D7 + AA* mit der Backbone-Architektur *EfficientNet-B7*. Auf Basis der zuvor getroffenen Begründung (fehlerhafter Trainingsprogrammiercode) wurden beide Verfahren nicht für das Benchmark ausgewählt. Die nächst höhere AP_{coco}-Metrik weist die *ATSS* Meta-Architektur verbunden mit der *dcnv2-X-101-64x4d-FPN-2x* Backbone-Architektur auf. Für die im Artikel von *ATSS* referenzierte Implementierung [156] wurde erfolgreich ein initiales Training durchgeführt und die Validität der Inferenz bestätigt. Daher wurde das

Verfahren ATSS dcnv2-X-101-64x4d-FPN-2x als Repräsentant des Schwellenwertes $s_1 = 5$ für das Benchmark ausgewählt.

FPS ≥ 10

Für den Schwellenwert $s_2 = 10$ weißt *EfficientDet-D4* mit *EfficientNet-B4* die höchste AP_{coco}-Metrik auf. Auch diese nutzt den Programmiercode von *EfficientDet-D7 + AA* mit der Backbone-Architektur *EfficientNet-B7*. Da jedoch, wie zuvor geschildert, das Training fehlerbehaftet ist, wurde zunächst die Architektur mit der zweithöchsten AP_{coco}-Metrik *CenterMask** mit *V-99-FPN* ausgewählt. Bei CenterMask handelt es sich wie auch bei dem HTC-Verfahren um einen gekoppelten Algorithmus aus Object Detection und Instanzsegmentierung. Anders als beim HTC-Verfahren müssen für das Training von CenterMask Objektmasken annotiert werden [152]. Da sich der Benchmark rein mit Object Detection-Algorithmen befasst, liegen nur Trainingsdaten vor, in denen Bounding-Boxen annotiert sind. Eine zusätzliche Annotierung von Masken übersteigt den Umfang dieser Arbeit. Daher wurde auf die nächste Architektur zurückgegriffen: *ATSS* mit *dcnv2-R-101-FPN-2x*. Die Basis der Implementierung [156] ist die gleiche wie von der Meta-Architektur *ATSS* verbunden mit der *dcnv2-X-101-64x4d-FPN-2x* Backbone-Architektur. Diese wurde bereits validiert, weshalb *ATSS* *dcnv2-R-101-FPN-2x* als Verfahren für den Schwellenwert s_2 ausgewählt wird.

FPS ≥ 15

Für den Schwellenwert $s_3 = 15$ weist *EfficientDet-D3* mit *EfficientNet-B3* die höchste AP_{coco}-Metrik auf. Darauf folgt *CenterMask** mit *R-101-FPN*. Beide Verfahren können aufgrund zuvor getroffener Begründungen nicht im Benchmark verwendet werden. Daher wurde das Verfahren mit der nächst höheren AP_{coco}-Metrik, *YOLOv3 @800 + ASFF** mit *DarkNet-53* betrachtet. Hierfür konnten sowohl der Inferenz- als auch der Trainings-Programmiercode der Open-Source Implementierung [114] validiert werden. Daher wurde das Verfahren *YOLOv3 @800 + ASFF** für das Benchmark ausgewählt.

FPS ≥ 20

Für den Schwellenwert $s_4 = 20$ weist *EfficientDet-D3* mit *EfficientNet-B3* die höchste AP_{coco}-Metrik auf, wird jedoch aufgrund der fehlerhaften Implementierung für die Auswahl des Benchmarks ausgeschlossen. Darauf folgt *YOLOv3 @800 + ASFF** mit *DarkNet-53*. Es handelt sich um das gleiche Verfahren wie für den Schwellenwert s_3 . Damit wurden keine weiteren Verfahren für das Benchmark ausgewählt.

FPS ≥ 25

Für den Schwellenwert $s_5 = 25$ weist, wie auch für die Schwellenwerte s_3 und s_4 , das Verfahren *YOLOv3 @800 + ASFF** mit *DarkNet-53* die höchste AP_{coco}-Metrik auf. Daher wurden keine weiteren Verfahren für das Benchmark ausgewählt

FPS ≥ 30 , FPS ≥ 35 , FPS ≥ 40 , FPS ≥ 45

Für den Schwellenwert $s_6 = 30$, sowie für die Schwellenwerte $s_7 = 35$, $s_8 = 40$ und $s_9 = 45$, weist *YOLOv3 @608 + ASFF** mit *DarkNet-53* die höchste AP_{coco}-Metrik auf. Die Basis der Implementierung ist die gleiche wie von *YOLOv3 @800 + ASFF** mit *DarkNet-53*. Das initiale Training sowie die Inferenz wurden erfolgreich durchgeführt. Daher wurde das Verfahren *YOLOv3 @608 + ASFF** für das Benchmark ausgewählt.

FPS ≥ 50

Für den Schwellenwert $s_{10} = 50$, weist *YOLOv3 @416 + ASFF** mit *DarkNet-53* die höchste AP_{coco}-Metrik auf. Die Basis der Implementierung ist die gleiche wie von *YOLOv3 @800 + ASFF** mit *DarkNet-53*. Das initiale Training sowie die Inferenz wurden erfolgreich durchgeführt. Daher wurde das Verfahren *YOLOv3 @416 + ASFF** für das Benchmark ausgewählt.

FPS ≥ 55

Für den Schwellenwert $s_{11} = 55$, weist *YOLOv3 @416 + ASFF* mit *DarkNet-53* die höchste AP_{coco}-Metrik auf. Die Basis der Implementierung ist die gleiche wie von *YOLOv3 @800 + ASFF** mit *DarkNet-53*. Das initiale Training sowie die Inferenz wurden erfolgreich durchgeführt. Daher wurde das Verfahren *YOLOv3 @416 + ASFF* für das Benchmark ausgewählt.

FPS ≥ 60

Für den Schwellenwert $s_{12} = 60$, weist *YOLOv3 @320 + ASFF** mit *DarkNet-53* die höchste AP_{coco}-Metrik auf. Die Basis der Implementierung ist die gleiche wie von *YOLOv3 @800 + ASFF** mit *DarkNet-53*. Das initiale Training sowie die Inferenz wurden erfolgreich durchgeführt. Daher wurde das Verfahren *YOLOv3 @320 + ASFF** für das Benchmark ausgewählt.

FPS ≥ 65 , FPS ≥ 70

Für den Schwellenwert $s_{13} = 65$, weist *RFBNet* mit *VGG-16* die höchste AP_{coco}-Metrik auf. Das Verfahren bietet eine valide Open-Source Implementierung [113]. Ein initiales

Training sowie die Inferenz wurden erfolgreich durchgeführt. Daher wurde das Verfahren RFBNet VGG-16 für das Benchmark ausgewählt

5.3.3 Resultat der Filterung

Die aus der Filterung hervorgegangenen Verfahren sind zusammenfassend in Tabelle 5.3 aufgeführt. Diese listet für den jeweiligen definierten Schwellenwert aus S das Verfahren, welches für die Anwendung im Benchmark ausgewählt worden ist. Zudem ist für jedes Verfahren die AP@0.5-Metrik angegeben, die aus einer Evaluierung auf Basis der Validierungsdaten des Datensatzes $D1$ resultieren. Eine Abkürzung für den Bezeichner der Verfahren ist für den weiteren Verlauf in Spalte *Synonym* aufgeführt.

Schwellenwert	Aus der Filterung hervorgehende Verfahren		AP@0.5	Synonym
	Meta-Architektur	Backbone-Architektur		
$0 \leq FPS$	Hybrid Task Cascade	X-101-64x4d FPN DCN	98.14	HTC X-101-FD
$5 \leq FPS$	ATSS	dcnv2-X-101-64x4d-FPN-2x	95.38	ATSS X-101-FD
$10 \leq FPS$	ATSS	dcnv2-R-101-FPN-2x	94.86	ATSS X-101-FD
$15 \leq FPS$	YOLOv3 @800 + ASFF*	DarkNet-53	88.91	YOLOv3-ASFF* D-53 @800
$20 \leq FPS$				
$25 \leq FPS$				
$30 \leq FPS$				
$35 \leq FPS$	YOLOv3 @608 + ASFF*	DarkNet-53	94.74	YOLOv3-ASFF* D-53 @608
$40 \leq FPS$				
$45 \leq FPS$				
$50 \leq FPS$	YOLOv3 @416 + ASFF*	DarkNet-53	94.34	YOLOv3-ASFF* D-53 @416
$55 \leq FPS$	YOLOv3 @416 + ASFF	DarkNet-53	94.22	YOLOv3-ASFF D-53 @416
$60 \leq FPS$	YOLOv3 @320 + ASFF*	DarkNet-53	94.10	YOLOv3-ASFF* D-53 @320
$65 \leq FPS$	RFB Net300	VGG	62.03	RFBNet VGG-16 @300
$70 \leq FPS$				

Tabelle 5.3: Auflistung der finalen Auswahl der im Benchmark anzuwendenden Verfahren.

Kapitel 6

Algorithmische Konzepte der Benchmark-Verfahren

In Kapitel 5.1 wird der Aufbau der im Anhang aufgeführten Tabelle B.1 und damit der Aufbau der Grundgesamtheit der State-of-the-Art Object Detection-Algorithmen beschrieben. Begründet auf die begrenzt zur Verfügung stehende Bearbeitungszeit können nicht alle in der Tabelle gelisteten Verfahren in der Arbeit behandelt werden. Daher wird in Kapitel 5.3 eine Filterung durchgeführt, deren Ergebnis in Tabelle 5.3 festgehalten ist. Nachfolgend werden die wichtigsten algorithmischen Prinzipien, die aus der Filterung resultierenden Verfahren erläutert. Tiefergehende Information sind den Originalartikeln der Verfahren zu entnehmen. Dies gilt auch für nicht beschriebene Verfahren aus Tabelle B.1. Hierfür sind in der letzten Spalte der Tabelle entsprechende Referenzen aufgeführt.

Zunächst wird die Architektur der Klassifikationsalgorithmen geschildert, die als Backbone-Architekturen in den selektierten Meta-Architekturen integriert sind. Anschließend werden die Konzepte der Meta-Architekturen vorgestellt. Die Reihenfolge der nachfolgend geschilderten Beschreibungen richtet sich nach dem historischen Entwicklungsverlauf der aufgeführten Architekturen.

6.1 Konzepte der Backbone-Architekturen

Wie in Kapitel 3.4 erläutert, stellen die Backbone-Architekturen visuelle Features bereit, welche von den Meta-Architekturen als Grundlage für die Detektion von Objekten herangezogen werden. Alle Backbone-Architekturen wurden dafür entworfen, in erster Linie als Klassifikationsalgorithmus verwendet zu werden. Für den Einsatz der Feature Extraction als Bestandteil von Algorithmen der Object Detection werden zunächst die letzten Layer entfernt. Anschließend erfolgt eine Konkatenation an die jeweilige Meta-Architektur. Nachfolgend werden die Klassifikationsalgorithmen vorgestellt, welche für die ausgewählten Meta-Architekturen zur Feature Extraction eingesetzt werden.

6.1.1 Konzepte – VGG

Die *Visual Geometry Group (VGG)*-Architektur weist eine sehr einfache Struktur auf. Es werden lediglich Convolutional Layer, Max Pooling Layer und Fully-Connected Layer verwendet. In [124] werden verschiedene Konfigurationen aufgeführt, denen jeweils ein Bezeichner von A bis E zugewiesen wird. Insgesamt umfassen die Architekturen zwischen 11 und 19 Layer. Der Hauptentwicklungsschritt zum Zeitpunkt der Veröffentlichung von VGG bestand in der Verwendung von kleineren 3x3 Kernel in den Convolutional Layer. Diese erlauben eine größere Gesamtanzahl an Layer (Netztiefe) im Vergleich zu den Vorgängern von VGG. Zudem werden Convolutional Layer mit einem 1x1 Kernel in Konfiguration C verwendet, um zusätzliche Nichtlinearitäten hinzuzufügen, jedoch keine Änderung an der Größe des Receptive Field vorzunehmen. Dieses Vorgehen ist aus [66] bekannt. Für eine detaillierte Übersicht der in VGG verwendeten CNN-Architektur wird auf den Original-Artikel des VGG [124] verwiesen.

6.1.2 Konzepte – ResNet

Die Tiefe der Architektur von CNNs (Anzahl der Layer) ist ein wichtiger Faktor für deren Performance [47]. Die Architektur des *Residual Network (ResNet)* adressiert ein Kernproblem im Kontext von tiefen Netzstrukturen: Die Schwierigkeit eines erfolgreichen Trainings aufgrund der Charakteristik von *vanishing/exploding gradients*. Während der Anwendung des Backpropagation Algorithmus haben kleine und große Werte der partiellen Ableitungen einzelner Layer folgenden Effekt: Wenn bei tiefen Strukturen viele kleine Werte multipliziert werden, konvergiert der Gradient aller Layer gegen Null (vanishing). Im umgekehrten Fall von großen Werten konvergiert der Gradient gegen unendlich (exploding). Dies resultiert in instabilem Trainingsverlauf. Zur Lösung dieser Problematik werden in der ResNet-Architektur in regelmäßigen Abständen sogenannte Skip- / Shortcut-Verbindungen eingeführt. Abbildung 6.1 illustriert dieses Vorgehen. Die resultierende Funktion am Knotenpunkt der Skip-Verbindung ergibt sich als: $H(x) = F(x) + x$. Damit haben die übersprungenen (weight) Layer die Funktion $F(x) = H(x) - x$ zu lernen. Das Lernen von $F(x)$ wird im Artikel [47] als Residual Mapping, die resultierende Layer-Architektur als Residual Block bezeichnet. Unter Anwendung der Residual-Blöcke wird folgendes Verhalten realisiert: Im Fall von extremen Werten des Gradienten (vanishing oder exploding) kann immer noch der Gradient der Identität, demnach x , zurück propagiert werden, um einem instabilen Trainingsverlauf entgegen zu wirken. Auf Grundlage dessen ist eine Entwicklung von Netzarchitekturen mit 18, 34, 50, 101 und 152 Layern möglich [47]. Für eine detaillierte Übersicht der in den Konfigurationen des ResNet verwendeten CNN-Architekturen, wird auf den Original-Artikel des ResNets [47] verwiesen.

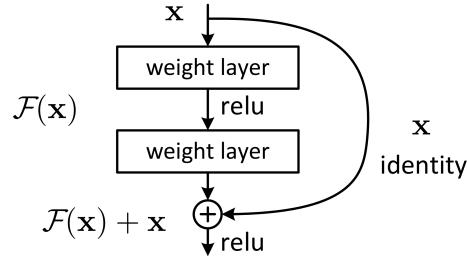


Abbildung 6.1: Residual Block der ResNet-Architektur. Entnommen aus [47].

6.1.3 Konzepte – ResNeXt

ResNeXt ist eine Weiterentwicklung des ResNet. Beschrieben als einfache, hoch modulare Multi-Branch-Architektur, wird das Netzwerk durch die wiederholte Konkatenation von Blöcken der gleichen Topologie aufgebaut. Jeder Block besteht dabei aus einer Aggregation von einer Menge aus Transformationen [150]. Angelehnt ist die Struktur an die Residual-Blöcke im ResNet. Dabei können die Transformationen beliebige Funktionen repräsentieren. Durch den beschriebenen Aufbau wird eine neue Dimension eingeführt, die *Kardinalität*. Diese ist definiert als die Anzahl der aggregierten Transformationen in einem ResNeXt-Block. In [150] wurde gezeigt, dass bei gleichbleibender Komplexität eine Vergrößerung der Kardinalität zu einer erhöhten Genauigkeit bei der Klassifikation von Bildern führt. Zudem ist eine Vergrößerung der Kardinalität effektiver, als tiefere Netzstrukturen aufzubauen. Abbildung 6.2 stellt die Architektur eines ResNet-Block und eines ResNeXt-Block gegenüber. Gezeigt wird die jeweilige Struktur der verwendeten Convolutional Layer. Dies illustriert insbesondere die Aggregation der parallel genutzten Transformationen im ResNeXt-Block (Abb. 6.2b) Für eine detaillierte Übersicht der in ResNeXt verwendeten CNN-Architektur wird auf den Original-Artikel des ResNeXt [150] verwiesen.

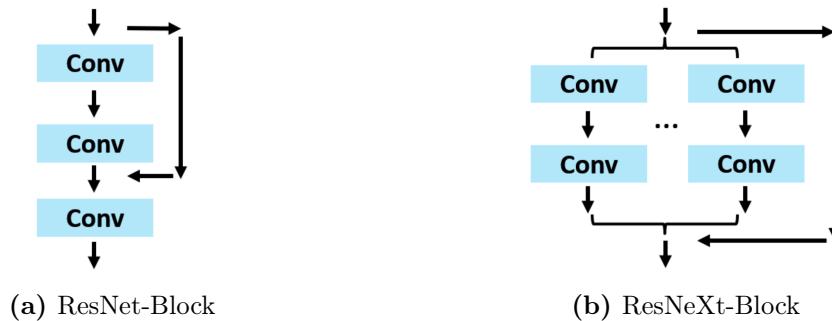


Abbildung 6.2: Vergleich eines ResNet- und ResNeXt-Block. Bearbeitete Version aus [60].

6.1.4 Konzepte – Darknet-53

Das *Darknet-53* wurde eigens für die Meta-Architektur *YOLOv3* entwickelt [110]. Es ist angelehnt an die Architektur von ResNet. Die verwendeten Convolutional Layer verwenden Kernel der Größe 3x3 und 1x1. Insgesamt werden 53 Layer verwendet. Ein großer Vorteil gegenüber des ResNets ist die stark reduzierte Ausführungszeit bei nahezu gleicher Performance [110]. Die Autoren führen dies im Wesentlichen auf die effektive Ausnutzung der GPU sowie die reduzierte Anzahl der Layer (53 gegenüber 101 oder 152). Für eine detaillierte Übersicht der in Darknet-53 verwendeten CNN-Architektur wird auf den Original-Artikel des Darknet-53 [110] verwiesen.

6.1.5 Konzepte – FPN

Objekte verschiedener Größe und Skalierung zu erkennen, ist eine Grundproblematik, der Algorithmen aus dem Bereich der Bildverarbeitung unterliegen [67]. Adressiert wird die Problemstellung durch die Erweiterung bestehender CNN-Architekturen anhand von pyramidenähnlichen Strukturen [68]. Diese erlauben eine Verarbeitung von Feature Maps verschiedener Ebenen in einer CNN-Architektur und ermöglichen damit eine Bereitstellung von Features von Objekten in verschiedenen Skalierungen. Der Object Detection-Algorithmus *Single Shot MultiBox Detector (SSD)* [74] ist eine der ersten Herangehensweisen, Feature-Pyramiden in CNNs zu verwenden. In Kapitel 6.2.2 wird SSD als Basis der RFBNet Meta-Architektur näher beschrieben. Im Artikel des *Feature Pyramid Networks (FPN)* werden Schwachstellen von SSD aufgeführt und die darauf vorgenommenen Anpassungen beschrieben [67]. Vorab ist darauf hinzuweisen, dass das FPN nur in Verbindung mit bestehenden Backbone-Architekturen (wie ResNet oder ResNeXt) verwendet wird und daher als Aufsatz und nicht als eigenständige Architektur zu betrachten ist. Die Grundbausteine bilden ein Bottom-Up-Pfad, ein Top-Down-Pfad und laterale Verbindungen zwischen den Pfaden. Abbildung 6.3 illustriert diesen Aufbau. Der Bottom-Up-Pfad entspricht der *feed-forward*-Berechnung der Backbone-Architektur und stellt damit eine Feature-Hierarchie bestehend aus Feature Maps unterschiedlicher Auflösung bereit. Jede Ebene der Hierarchie ist damit für die Detektion von Objekten in unterschiedlicher Auflösung zuständig, auf Basis der jeweils zur Verfügung stehenden Feature Map. Der Top-Down-Pfad führt ein Upsampling durch und stellt auf diese Weise Features mit einer größeren semantischen Repräsentation zur Verfügung. Die Features des Bottom-Up-Pfads weisen zwar eine geringere semantische Repräsentation auf, jedoch bieten die Aktivierungen der Layer einen besseren Ausgangspunkt zur Lokalisierung von Objekten [67]. Daher wurden laterale Verbindungen zur Konkatenation der Feature Maps beider Pfade eingeführt, um die jeweiligen Informationen simultan nutzen zu können. Im Original-Artikel des FPN [67] sind tiefergehende Details ausgeführt, aus denen hervorgeht, wie das FPN an bestehenden CNN-Architekturen des ResNet oder RexNext konkateniert wird.

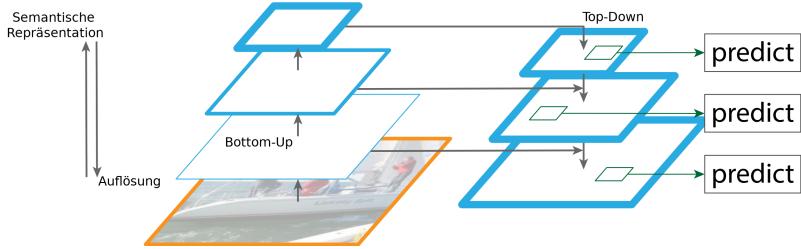


Abbildung 6.3: Struktur des FPN. Bearbeitete Version aus [67].

6.2 Konzepte der Object Detection-Algorithmen

Nachfolgend wird die Architektur sowie die Konfiguration der Hyperparameter für die im Benchmark verwendeten Object Detection-Algorithmen erläutert.

6.2.1 Konzepte – HTC

Für nachfolgende Erläuterungen wird auf die Differenzierung von Object Detection, Semantische Segmentierung und Instanzsegmentierung in Kapitel 3.1.2 verwiesen. Die *Hybrid Task Cascade (HTC)* Meta-Architektur realisiert in erster Linie eine Instanzsegmentierung, für deren Lösung als Zwischenschritt jedoch eine Object Detection notwendig ist. Für die Anwendung im Benchmark der vorliegenden Arbeit werden die Blöcke der Instanzsegmentierung aus der Architektur entfernt und nur die Object Detection verwendet. Aufgebaut ist die HTC auf Basis einer Kombination aus dem Cascade R-CNN [15] und dem Mask R-CNN [46]. Beide beruhen auf der Faster R-CNN-Architektur [112], die als Kernkonzept der Two-Stage Verfahren in Kapitel 3.4 vorgestellt wird.

Das Cascade R-CNN erweitert Faster R-CNN um eine Kaskade bestehend aus mehreren Detektoren, die mit gesteigertem IOU-Grenzwert trainiert werden, um zwischen positiven und negativen Kandidaten-Boxen zu unterscheiden [15]. Das Training ist als sequenzieller Prozess organisiert. Dabei wird der nächste Detektor trainiert, wenn die Detektionsgenauigkeit des Vorgängers genügt, um zur Generierung von Kandidaten-Boxen für den Nachfolger verwendet zu werden. Damit wird eine schrittweise Verbesserung der Detektionsgenauigkeit erreicht. Das Mask R-CNN realisiert eine Instanzsegmentierung. Dafür wird dem Faster R-CNN ein zusätzlicher Zweig hinzugefügt, der parallel zur Detektion von Bounding-Boxen, die in den Bounding-Boxen befindlichen Instanzen als Masken detektiert [46].

Als Weiterentwicklung der Instanzsegmentierung lieferte ein erster Ansatz, das Cascade R-CNN und das Mask R-CNN als parallele Pfade direkt in eine Architektur zu integrieren, nur einen begrenzten Zuwachs an Detektionsgenauigkeit. Der Schlüssel ist nach [20] eine verschachtelte Kombination der Architekturen sowie das Hinzufügen eines Pfades zur Generierung von Features auf Basis einer semantischen Segmentierung . Durch die Verschachtlung profitiert die Instanzsegmentierung von der wechselseitige Beziehung zwischen Object Detection und Instanzsegmentierung. Auf Grundlage der Features, die aus der se-

mantische Segmentierung hervorgehen, stehen mehr Informationen über den räumlichen Kontext zur Verfügung. Diese erlauben es schwer zuzuordnenden Vordergrund von überladenem Hintergrund zu differenzieren. Abbildung 6.4 illustriert den Aufbau der HTC. Zunächst werden wie im Faster R-CNN Kandidaten-Boxen von einem RPN generiert. Anschließend werden mittels ROI-Pooling (pool) die für die Object Detection notwendigen Features auf Basis dieser Kandidaten-Boxen extrahiert. Parallel zum RPN wird die Semantische Segmentierung (S) durchgeführt und die resultierenden Features ebenfalls an den Object Detection-Pfad (B) weitergeleitet. Da wie zuvor angesprochen für diese Arbeit nur Object Detection-Algorithmen betrachtet werden, werden die Pfade der Instanzsegmentierung (M) aus der HTC-Architektur entfernt. Dafür gilt es lediglich Änderungen an der Konfigurationsdatei der verwendeten Implementierung von HTC vorzunehmen. Es erfolgte demnach kein Aufwand, eigene Implementierungen der Architektur durchzuführen. Für eine detaillierte Übersicht, der in HTC verwendeten CNN-Architektur, wird auf den Original-Artikel der HTC [20] verwiesen.

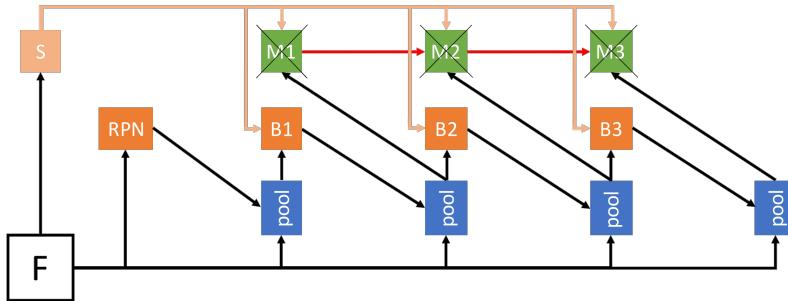


Abbildung 6.4: Architektur des HTC Object Detection-Algorithmus. Bearbeitete Version aus [20].

Für das Benchmark wird für die HTC-Architektur die Backbone-Architektur ResNeXt gekoppelt mit dem Aufsatz eines FPN, sowie dem Einsatz einer Weiterentwicklung des Deformable Convolutional Layers verwendet [20, 150, 164]. Aus [20] geht die in Tabelle 6.1 aufgeführte Grundkonfiguration für das Training von *HTC X-101-FD* hervor. Für das Training im Benchmark wird die Methodik des Transfer-Learnings eingesetzt. Dafür wird die HTC-Architektur mit Gewichten von einem Open-Source Modell aus [93] initialisiert, welches auf dem MS COCO-Datensatz für 20 Epochen trainiert worden ist. Als Optimierer wurde SGD gekoppelt mit einem *schrittbasierten LR-Scheduling* verwendet. Hierfür wird der initiale Wert der LR mit einem definierten Faktor multipliziert, wenn die Epoche einen definierten Schritt erreicht. Dies resultiert in einer absteigenden Treppenfunktion, welche das in Kapitel 3.2 gewünschte Verhalten für den Verlauf des Optimierungsprozesses realisiert. Abbildung 6.5 illustriert den beschriebenen Verlauf der LR am Beispiel eines Trainings mit 20 Epochen. Gezeigt wird der Graph der LR als Funktion der Epochen. Zu Beginn des Scheduling wird die Methodik der Warm-Up Epochen angewendet, bis der initiale Wert der LR bei 500 Trainingsschritte (etwa 0.5 Epochen) erreicht wird. Danach erfolgt eine

Anpassung der LR um den Faktor 0.1 bei den Schritten 16 und 19. Die Eingabebilder werden unter Anwendung eines Multi-Scale-Trainings (siehe Kapitel 4.1) mehrfach während des Verlaufs des Trainingsprozesses skaliert. Die Breite wurde dafür auf 1600 festgesetzt, während die Höhe zufällig zwischen 400 und 1400 anzupassen ist. Dies wird durch die in Tabelle 6.1 aufgeführte Kodierung spezifiziert: $MS\ 1600 \times \{400, \dots, 1400\}$. Für die Inferenz werden die Bilder auf eine Auflösung von 1333×800 skaliert. Zudem wird für nachfolgende Beschreibung einer Auflösung, die während der Inferenz genutzt wird, die Kodierung $@Auflösung$ definiert. Ist für Auflösungen anstelle eines Tupels nur ein einzelner Wert angegeben, ist die quadratische Auflösung gemeint.

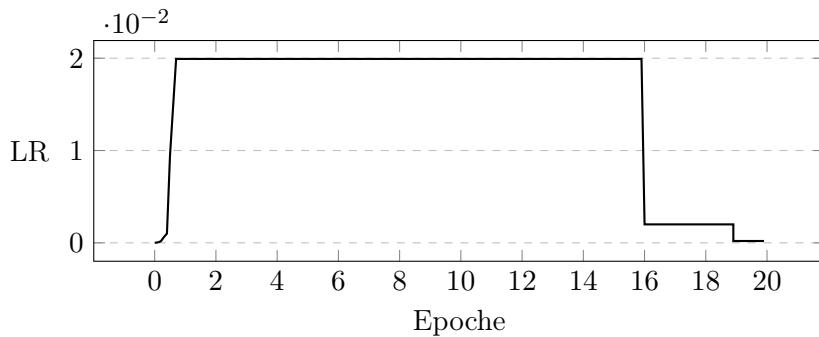


Abbildung 6.5: Schrittbasierter LR-Scheduling

HTC Trainingskonfiguration	
Datensatz	COCO train 115k
Auflösung	Training: $MS\ 1600 \times \{400, \dots, 1400\}$ Inferenz: $@1333 \times 800$
Anzahl Epochen	20
Batch Size	2 pro GPU (8 V100 GPUs)
Optimierer	SGD
Initiale LR	0.2
LR-Scheduling	schrittbasiert
LR-Scheduling Schritte	16., 19. Epoche
LR-Scheduling Faktor	0.1
Warum-Up LR	500 (Trainingsschritte)

Tabelle 6.1: Originalkonfiguration der HTC X-101-FD-Architektur. Entnommen aus [20, 92].

6.2.2 Konzepte – RFBNet

Das *Receptive Field Block Net (RFBNet)* gehört zur Gruppe der One-Stage-Verfahren und nutzt als Basis die Architektur des *Single Shot MultiBox Detector (SSD)*. Dessen Aufbau richtet sich nach dem Grundkonzept der One-Stage-Verfahren, welches in Kapitel 3.4 beschrieben worden ist. Dieses wird um Pfade verschieden skalierter Feature Maps erweitert. Jeder Pfad ist für die Detektion von Objekten verschiedener Größe verantwortlich. SSD nutzt mehrere Anchor-Boxen pro Rasterzelle einer Feature Map. Zur Realisierung der Object Detection gilt es, zunächst jeder Ground-Truth-Bounding-Box eine der vordefinierten Anchor-Boxen zuzuordnen. Anschließend erfolgt eine Bounding-Box-Regression zur Bestimmung der Offsets zwischen Ground-Truth- und Anchor-Box relativ zur Position der in jeweiligen Feature Map. Abbildung 6.6 veranschaulicht dieses Vorgehen. Der blauen und roten Ground-Truth-Bounding-Box werden jeweils Anchor-Boxen verschiedener Feature Maps zugeordnet. Auf Grundlage dessen erfolgt anschließend die Ermittlung der Offsets. Parallel dazu werden die Klassenwahrscheinlichkeiten anhand eines Softmax-Layer berechnet.

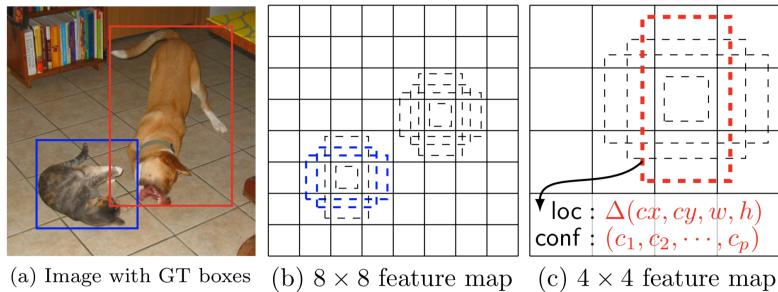


Abbildung 6.6: Zuordnung von Anchor-Boxen zu Ground-Truth Boxen. Entnommen aus [74].

Kernproblem von realzeitfähigen One-Stage-Verfahren ist die reduzierte Detektionsgenauigkeit verglichen mit Two-Stage-Verfahren. Diese ist nach [72] begründet auf die limitierte Qualität der Features der verwendeten leichtgewichtigen Backbone-Architektur (VGG, MobileNet) dieser Object Detection-Algorithmen. Das RFBNet führt in diesem Zusammenhang eine Methode ein, welche die Robustheit und Unterscheidbarkeit von Features leichtgewichtiger Backbone-Architekturen anhand eines manuell angepassten Mechanismus verbessert.

Der Mechanismus ist inspiriert von der Struktur des Receptive Field im menschlichen visuellen Kortex. Erkenntnisse aus dem Bereich der Neurowissenschaft legen nahe, dass die Größe der population Receptive Fields (pRF) (englischer Fachbegriff) eine Funktion der Exzentrizität ist [148]. Demnach sind die Regionen nahe des Mittelpunkts des Receptive Field von großer Bedeutung. Angelehnt an diese Eigenschaft wurde ein RFB Modul entworfen, welches die Erkenntnisse der Neurowissenschaften in eine CNN-Architektur transferiert.

Dafür werden parallele Convolutional Layer mit unterschiedlichem Dilation-Parametern verwendet. Die finale Konkatenation findet über einen Convolutional Layer mit einem 1x1 Kernel konkateniert statt. Für eine detaillierte Übersicht der in RFBNet bzw. in SDD verwendeten CNN-Architektur, wird auf den Original-Artikel des RFBNet [72], sowie von SDD [74], verwiesen.

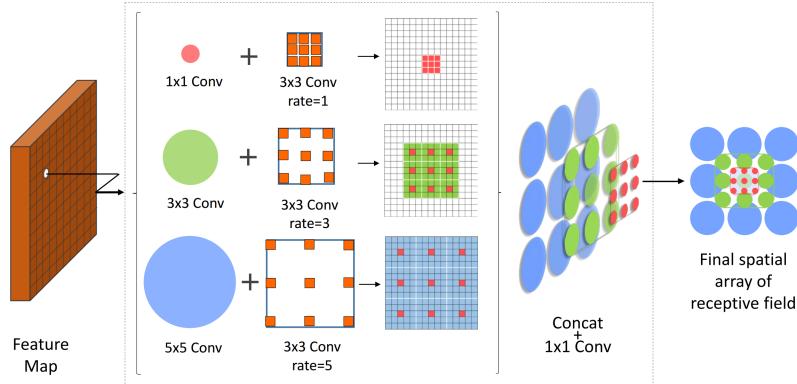


Abbildung 6.7: Aufbau des RFB Moduls. Bearbeitete Version aus [72].

Für das Benchmark wird das RFBNet gekoppelt mit der Backbone-Architektur VGG-16 (Konfiguration *C* [124]) verwendet. Analog zu Kapitel 6.2.1 sind in Tabelle 6.2 die Parameter der Grundkonfiguration für *RFBNet VGG-16 @300* aufgeführt. Für das Training des RFBNet werden zur Initialisierung der Modelle im Benchmark Open-Source Modelle aus [113] genutzt, die unter Verwendung der aufgeführten Parameter auf dem MS COCO-Datensatz trainiert worden sind. Das RFBNet nutzt ebenfalls ein schrittbasieretes LR-Scheduling. Die dazugehörige Beschreibung ist dem vorherigen Abschnitt (Konzepte – HTC) zu entnehmen. Sowohl für das Training als auch für die Inferenz wird eine quadratische Auflösung der Größe 300 verwendet. Diese ist im Bezeichner des RFBNet gekennzeichnet. Es wird kein MS-Training eingesetzt. Für die LR wird zu Beginn des Trainings für 5 Epochen die Methodik der Warm-Up-Epochen eingesetzt.

RFBNet Trainingskonfiguration	
Datensatz	COCO train 115k
Auflösung	Training: 300 Inferenz: @300
Anzahl Epochen	250
Batch Size	32 pro GPU (1 Titan X GPU)
Optimierer	SGD
Initiale LR	0.001
LR-Scheduling	schrittbasiert
LR-Scheduling Schritte	150., 200. Epoche
LR-Scheduling Faktor	0.1
Warum-Up LR	5 Epochen

Tabelle 6.2: Originalkonfiguration der *RFBNet VGG-16 @300*-Architektur. Entnommen aus [72].

6.2.3 Konzepte – ATSS

Die Architektur des *Adaptive Training Sample Selection (ATSS)* Object Detection-Algorithmus beruht auf einem Zusammenschluss von Konzepten der Object Detection-Algorithmen *RetinaNet* [68] und *Fully Convolutional One-Stage Object Detection (FCOS)* [138]. Bei beiden Object Detection-Algorithmen handelt es sich um One-Stage-Verfahren. RetinaNet führt die Bounding-Box-Regression ähnlich wie SDD auf Basis der Offsets von Anchor-Boxen durch, FCOS nutzt hierfür ähnlich zu YOLO eine Bounding-Box-Regression ausgehend von einem zentralen Punkt. In der Literatur wird in diesem Zusammenhang zwischen Anchor-Based- und Achor-Free-Architekturen unterschieden. FCOS und RetinaNet nutzen die gleiche CNN-Architektur, bis auf die in FCOS vorgestellte Erweiterung der Klassifikation: Der *Centerness*-Pfad [138]. Diese wurde zur Filterung von Bounding-Boxen eingeführt, die eine niedrige Klassenwahrscheinlichkeit aufweisen. Bestraft werden Bounding-Boxen, deren Ausgangspunkt der Regression zu weit entfernt von der Mitte entsprechenden Groun-Truth-Bounding-Box ist. Anschließend wird die *Centerness* mit der ermittelten Klassenwahrscheinlichkeit multipliziert und die Bounding-Boxen auf Basis des Resultats gefiltert. Ausgangsbasis für die CNN-Architektur ist ein ResNet als Backbone-Architektur mit einem aufgesetzten FPN. An jede Ebene der Feature-Pyramide wird ein Object Detection-Block konkateniert. Diese sind jeweils aufgesplittet in eine Bounding-Box-Regression und Klassifikation (+ Centerness). Für Details der geschilderten Aspekte zur CNN-Architektur wird auf [138] verwiesen.

Im Original-Artikel des ATSS Object Detection-Algorithmus findet eine Analyse der Unterschiede zwischen RetinaNet und FCOS statt. Daraus geht hervor, dass sich der Unterschied im Wesentlichen auf die Art und Weise beläuft, wie positive und negative Trainingsbeispiele für das Training des Pfades zur Klassifikation definiert werden [157]. Basierend auf dieser Feststellung wurde der Algorithmus 2 entwickelt. Dabei werden die Trainingsbeispiele basierend auf Statistiken bestimmt. Zunächst wird für jede Groun-Truth-Bounding-Box g eine Auswahl von k Anchor-Boxen selektiert, deren Mittelpunkt sich am nächsten zu dem von g befindet. Anschließend wird ein Mittelwert \bar{d} , sowie die Standardabweichung σ^2 , für die IOU-Metrik zwischen g und allen selektierten Anchor-Boxen bestimmt. Auf deren Grundlage wird ein IOU-Grenzwert t ermittelt, der für eine zusätzliche Filterung der k Anchor-Boxen herangezogen wird. Alle nicht gefilterten Anchor-Boxen werden in die Menge an positiven Trainingsbeispielen aufgenommen. Alle übrigen Anchor-Boxen bilden die Menge der negativen Trainingsbeispiele.

Die CNN-Architektur von ATSS folgt der von FCOS mit einer Anpassung: Anstelle eines Anchor-Punktes werden wie in RetinaNet Anchor-Boxen mit achtfacher Skalierung genutzt, um positive und negative Trainingsbeispiele zu definieren. Die Bounding-Box-Regression wird weiterhin ausgehend von den Mittelpunkten der unter Anwendung des ATSS-Algorithmus bestimmten Trainingsbeispiele durchgeführt. Für eine detaillierte Übersicht

Algorithmus 2: Adaptive Training Sample Selection (ATSS). Bearbeitete Version aus [157].

Eingabe:

G Menge der Ground-Truth Boxen im gegebenen Eingabebild
 L Anzahl der Level in der Feature-Pyramide
 A_i Menge der Anchor-Boxen im i-ten Level der Feature-Pyramide
 A Menge aller Anchor-Boxen
 k Hyperparameter zur Auswahl von maximal k Anchor-Boxen aus A_i

Ausgabe:

P Menge der positiven Trainingsbeispiele
 N Menge der negativen Trainingsbeispiele

```

1 for  $g \in G$  do
2   Erstelle eine Leere Menge zur Speicherung von positiven Kandidaten  $C \leftarrow \emptyset$ 
3   for level  $i \in \{1, \dots, L\}$  do
4      $S_i \leftarrow$  selektiere k Anchor-Boxen von  $A_i$  deren Mittelpunkt am nächsten
        zum Mittelpunkt der Ground-Truth Box  $g$  ist
5      $C = C \cup S_i$ 
6   end
7   Bereche IOU zwischen  $C$  und  $g$  :  $D = \{IOU(c_i; g)\} \forall c_i \in C$ 
8   Bereche den Mittelwert von  $D$  :  $\bar{d} = MEAN(D)$ 
9   Bereche die Standardabweichung von  $D$  :  $\sigma^2 = STDEV(D)$ 
10  Bereche IOU Grenzwert für Ground-Truth  $g$  :  $t = \bar{d} + \sigma^2$ 
11  for candidate  $c_i \in C$  do
12    if  $IOU(c_i, g) \geq t$  and center of  $c_i$  in  $g$  then
13       $P = P \cup c$ 
14    end
15  end
16 end
17  $N = A - P$ 
18 return  $P, N$ 

```

der in ATSS bzw. in FCOS verwendeten CNN-Architektur, wird auf den Original-Artikel von ATSS [157], sowie von FCOS [138], verwiesen.

Für das Benchmark wird für die ATSS-Architektur die Backbone-Architektur ResNet und ResNeXt, jeweils gekoppelt mit dem Aufsatz eines FPN, sowie dem Einsatz einer Weiterentwicklung des Deformable Convolutional Layers verwendet [20, 150, 164]. Analog zu Kapitel 6.2.1 sind in Tabelle 6.2 die Parameter der Grundkonfiguration für *ATSS X-101-FD* und *ATSS X-101-FD* aufgeführt. Für das Training der ATSS-Architektur werden zur In-

initialisierung der Modelle im Benchmark Open-Source Modelle aus [156] verwendet, die unter Verwendung der aufgeführten Parameter trainiert worden sind. Die ATSS-Architektur nutzt ebenfalls ein schrittbares LR-Scheduling. Entgegen der gängigen Methodik, die Länge des Trainings über die Anzahl der Epochen zu definieren, wurde für ATSS die Länge über die Anzahl an Trainingsiterationen definiert. Die Umrechnung in Epochen wird näherungsweise auf Basis der Größe des angegebenen Datensatzes und der insgesamt verwendeten Batch Size (= 16, 2 pro GPU für 8 GPUs) ermittelt. Dazu sind die Trainingsschritte mit der Batch Size zu multiplizieren und anschließend durch die Gesamtgröße des Datensatzes zu teilen. Dies resultiert in eine Gesamtanzahl von ungefähr 25 Epochen. Für beide Architekturen wird ein MS-Training mit der spezifizierten Auflösung angewendet.

Trainingskonfiguration	
Datensatz	COCO train 115k
Auflösung	Training: MS $1333 \times \{640, \dots, 800\}$ Inferenz: @ 1333×800
Anzahl Epochen	180K Trainingsschritte, entsprechen ~ 25 Epochen
Batch Size	2 pro GPU (8 V100 GPUs)
Optimierer	SGD
Initiale LR	0.01
LR-Scheduling	schrittbares
LR-Scheduling Schritte	120K, 160K Trainingsschritt
LR-Scheduling Faktor	0.1
Warum-Up LR	500 Iterationen

Tabelle 6.3: Originalkonfiguration der ATSS-Architekturen. Entnommen aus [157, 156, 161].

6.2.4 Konzepte – YOLO-v3 ASFF

Der Object Detection-Algorithmus *YOLO-v3 ASFF* nutzt als Basis die dritte Version der in Kapitel 3.4 vorgestellten YOLO-Architektur mit einem *Adatively Spatial Feature Fusion (ASFF)* Aufsatz. Zudem wurden zum Aufbau einer Baseline-Architektur erweiterte Trainingsmethoden verwendet [73]. Diese werden in Kapitel 7.2 näher diskutiert. Die Entwicklungsschritte von YOLO zu YOLO-v3 werden nachfolgend vorgestellt. Zunächst fand in YOLO-v2 ein Austausch der Anchor-Free- in eine Anchor-Box-basierte Bounding-Box-Regression, sowie ein Austausch von VGG in Darknet-19 [109] statt. In YOLO-v3 wurde Darknet-19 zu Darknet-53 weiterentwickelt und zusätzlich eine dreifach skalierte Feature-Pyramide eingeführt. Alle Architekturen des YOLO-v3 ASFF Object Detection-Algorithmus nutzen die Backbone-Architektur Darknet-53. Das Konzept der Feature-Pyramide ist aus zuvor beschriebenen Architekturen bekannt. Auf Grundlage dessen wird die Problematik der Detektion von Objekten verschiedener Größe (Pixelfläche (PF)) und Skalierung adressiert. Üblicherweise wird der Heuristik gefolgt, große PF der Instanzen mit unteren und kleine PF der Instanzen mit oberen Ebenen der Feature-Pyramide zu assoziieren. Falls Eingabebilder Instanzen mit großer und kleiner PF enthalten, nimmt die Unterscheidung der Detektion von PF unterschiedlicher Größe die meiste Rechenleistung der Feature-Pyramide in Anspruch [73]. Daraus ergeben sich Inkonsistenzen bei der Berechnung der Gradienten für die Backpropagation. Dies führt letztendlich zu einer Verminderung der Effektivität einer Feature-Pyramide. Das Kernkonzept der ASFF besteht daher aus einem Lernprozess, der auf das Finden einer optimalen Fusion von Features ausgerichtet ist, welche aus unterschiedlichen Ebenen in der verwendeten Feature-Pyramide stammen. Es wird sichergestellt, dass nur nützliche Informationen nach der Konkatenation von Features erhalten bleiben. Die Fusion findet stufenweise entlang der Ebenen der Feature-Pyramide statt. Dafür sind für jede Ebene die Features der anderen Ebenen zunächst in die gleiche Auflösung zu skalieren. Anschließend erfolgt der Lernprozess zur optimalen Fusion, bei dem Features mit widersprüchlichen Informationen gefiltert werden. Der Lernprozess ist mit einem marginalen Overhead bezüglich der Berechnungskosten verbunden und kann in jedes One-Stage-Verfahren integriert werden, die eine Feature-Pyramiden Struktur aufweist. Abbildung 6.8 veranschaulicht den Prozess der Fusion von Features auf unterschiedlichen Ebenen der Feature-Pyramide, die Parameter $X^{i \rightarrow j}$ stehen für die jeweilige Ebene. Die Faktoren α , β und γ gilt es während des Lernprozesses zur Gewichtung der Fusionierung der Features zu bestimmen. Als Erweiterung wurde ein RFB-Modul (siehe Kapitel 6.2.2) und ein Drop-Block-Modul [35] in die bestehende CNN-Architektur von YOLO-v3 ASFF integriert und mit ASFF* gekennzeichnet. Für eine detaillierte Übersicht der in YOLO-v3 ASFF bzw. in YOLOv3 verwendeten CNN-Architektur, wird auf den Original-Artikel von YOLO-v3 ASFF [73], sowie von YOLOv3 [110], verwiesen.

Analog zu den vorherigen Unterkapiteln, sind in Tabelle 6.2 die Parameter der Grundkonfiguration für *YOLOv3-ASFF** *D-53 @800*, *YOLOv3-ASFF** *D-53 @608*, *YOLOv3-*

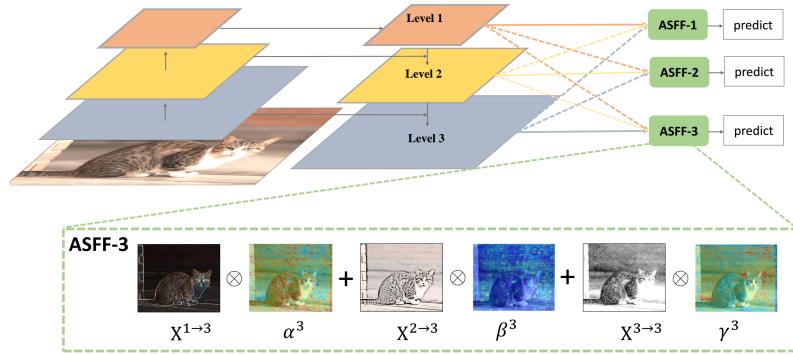


Abbildung 6.8: Illustration der *Adatively Spatial Feature Fusion*. Bearbeitete Version aus [73].

-ASFF* D-53 @416, YOLOv3-ASFF D-53 @416 und YOLOv3-ASFF* D-53 @320 aufgeführt. Die jeweils genutzte Auflösung während der Inferenz ist über den Bezeichner der jeweiligen Architektur gegeben. Die Architekturen YOLOv3-ASFF* D-53 @800 und YOLOv3-ASFF* D-53 @6083 wenden ein MS-Training in der spezifizierten Auflösung an. Für das Training der YOLO-v3 ASFF-Architekturen werden zur Initialisierung der Modelle im Benchmark Open-Source Modelle aus [114] verwendet, die unter Verwendung der aufgeführten Parameter trainiert worden sind. YOLO-v3 ASFF nutzt ein LR-Scheduling, welches dem Verlauf einer Cosinus-Funktion ähnelt. Dabei folgt die LR der in Abbildung 6.9 gegebenen Kurve, in Abhängigkeit der Anzahl der Optimierungsschritte. Das Plato resultiert durch Anwendung einer Startepoche, die definiert, ab welcher Epoche das *Cosinus Scheduling* zum Anpassen der LR angewendet wird. Zudem werden Warm-Up-Epochen zur Initialisierung der Gewichte genutzt (siehe Kapitel 3.2.1). Die Werte der y-Achse sind mit dem am Ende der Achse angegebenen Faktor zu multiplizieren. Abgebildet ist ein Beispielverlauf auf Basis einer Gesamtanzahl von 300 Epochen, einer Startepoche des Schedulings bei 20 und einer initialen LR von 0.001. Jede Epoche umfasst dabei 1000 Optimierungsschritte. Des Weiteren wird zur Verbesserung der Generalisierung auf dem MS COCO-Datensatz ein Mix-Up Training verwendet. Dafür werden die Pixel von zwei Eingabebildern zu einem Bild und die entsprechenden Annotationen beider Bilder zu einer Annotation zusammengefügt. Diese Methodik wurde in [155] eingeführt. Da kein anderer Object Detection-Algorithmus im Benchmark diese Methodik nutzt, gilt es aus Gründen der Fairness, den Effekt eines Mix-Up-Trainings während der Hyperparameteroptimierung der YOLO-v3 ASFF Meta-Architekturen näher zu untersuchen.

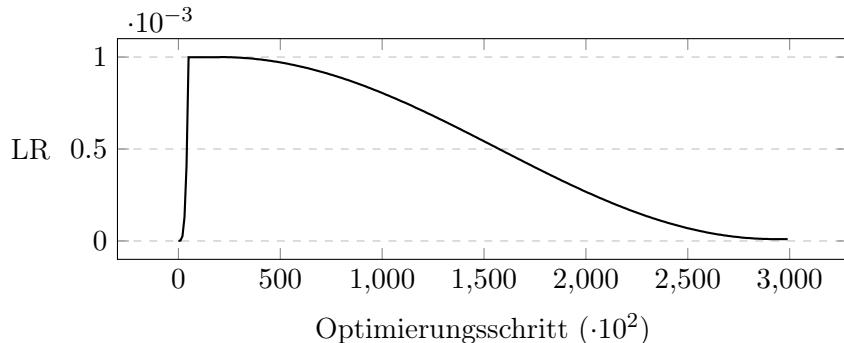


Abbildung 6.9: Cosinus LR-Scheduling

YOLO-v3 ASFF Trainingskonfiguration	
Datensatz	COCO train 115k
Auflösung	Training: 320, 416, {320, ..., 608}, {480, ..., 800} Inferenz: @320, @416, @608, @800
Anzahl Epochen	300 (450 für ASFF*)
Batch Size	16 pro GPU (4 V100 GPUs)
Optimierer	SGD
Initiale LR	0.001
LR-Scheduling	Cosinus
LR-Scheduling Start	20. Epoche
Warum-Up LR	5 Epochen

Tabelle 6.4: Originalkonfigurationen der YOLO-v3 ASFF-Architekturen. Entnommen aus [73].

Kapitel 7

Modellierung der Benchmark-Algorithmen

Im diesem Kapitel wird die Modellierung der in Kapitel 5 ausgewählten Architekturen beschrieben. Aus den geschilderten algorithmischen Konzepten in Kapitel 6 gehen Hyperparameter hervor, die es zu optimieren gilt. Auf Grundlage der optimalen Hyperparameter wird anschließend eine k-Fold Cross Validation auf Basis des in Kapitel 4.2 Datensatzes *D2* durchgeführt. In Kapitel 2 wird der Optimierungskonflikt bei Object Detection-Algorithmen zwischen einer geringen Laufzeit und einer hohen Detektionsgenauigkeit geschildert. Dieser wird als Speed-Accuracy Trade-Off festgehalten. Neben den Hyperparametern der jeweiligen Object Detection-Algorithmen werden daher zusätzliche Einflussfaktoren auf den Speed-Accuracy Trade-Off ausgearbeitet, die es für die Modellierung und die gesamtheitliche Evaluation in Kapitel 9 zu betrachten gilt. Dafür wird auf Ergebnisse einer bereits durchgeführten Analyse Speed-Accuracy Trade-Off zurückgegriffen und anschließend auf deren Basis eine Auswahl an zu untersuchenden Einflussfaktoren für das Benchmarking bestimmt.

7.1 Ermittlung zusätzlicher Einflussfaktoren

Anhand der Object Detection-Algorithmen in der Grundgesamtheit (siehe Tabelle B.1) wird der Speed-Accuracy Trade-Off der aktuellen State-of-the-Art Object Detection-Algorithmen abgedeckt. Dieses Kapitel befasst sich mit der Ermittlung von weiteren Einflussfaktoren, die es bei der Modellierung der in Kapitel 5.3 selektierten Verfahren zu berücksichtigen gilt. Die wesentlichen Faktoren wurden in [52] bereits näher untersucht. Diese werden nachfolgend zusammenfassend dargestellt. Anschließend wird eine Auswahl an Faktoren getroffen, die es im weiteren Verlauf des Benchmarks zusätzlich betrachtet werden.

7.1.1 Einflussfaktoren der aktuellen Literatur

In [52] wird ein detaillierter Vergleich durchgeführt, aus dem die wesentlichen Faktoren hervorgehen, die den Speed-Accuracy Trade-Off von Object Detection-Algorithmen beeinflussen. Untersucht wird der Einfluss der Faktoren auf Basis der Meta-Architekturen SSD [74], Faster R-CNN [112] und R-FCN [23]. Die algorithmischen Konzepte von SSD und Faster R-CNN werden in Kapitel 3.4 sowie Kapitel 6 erläutert. Da in diesem Kapitel lediglich eine Beschreibung der Einflussfaktoren erfolgt, wird für die algorithmischen Konzepte von R-FCN auf den Originalartikel [23] verwiesen. Nachfolgend sind für die untersuchten Einflussfaktoren zusammenfassende Erkenntnisse aufgeführt.

Effekt der Backbone-Architektur

Untersucht wird der Zusammenhang zwischen den von den Backbone-Architekturen (Feature-Extraktor) zur Verfügung gestellten visuellen Features und dem damit verbundenen Trade-Off zwischen der Detektionsgenauigkeit und der Laufzeit. Zudem wird die AP_{coco} -Metrik und der Top-1-Error der Klassifikationsalgorithmen gegenübergestellt. Dies dient der Untersuchung der Korrelation zwischen der Detektionsgenauigkeit des Object Detection-Algorithmus und der Erkennungsrate der Klassifikation. Als Resultat wird eine starke Korrelation der Metriken für Two-Stage-Verfahren (Faster R-CNN und R-FCN), jedoch eine eher schwache Korrelation für One-Stage-Verfahren (SSD) beobachtet. Demnach haben die von der Backbone-Architektur zur Verfügung gestellten visuellen Features nur einen geringen Einfluss auf die Detektionsgenauigkeit von SSD, aber einen hohen Einfluss auf die Two-Stage-Verfahren Faster R-CNN und R-FCN. Als allgemeiner Trend wird trotzdem die Aussage aus Kapitel 3.4 bestätigt: Die Qualität der Features der Backbone-Architektur steht in Korrelation mit der Genauigkeit der Object Detection-Algorithmen.

Effekt der Objektgröße

Alle verwendeten Meta-Architekturen weisen eine sehr viel höhere Genauigkeit bei der Detektion von großen Objekten, gemessen an der AP_l -Metrik, als bei der Detektion von kleinen Objekten auf, gemessen an der AP_s -Metrik. Des Weiteren wird gezeigt, dass trotz der allgemein eher niedrigen Detektionsgenauigkeit des SSD für kleine Objekte dieser eine kompetitive AP_s -Metrik im Vergleich zum Faster R-CNN und R-FCN aufweist. Darüber hinaus besitzt SSD für manche leichtgewichtigen Backbone-Architekturen sogar eine höhere AP_s -Metrik, verglichen mit der gleichen Kombination für Faster R-CNN und R-FCN.

Effekt der Auflösung des Eingabebildes

Die Auflösung hat einen starken Einfluss auf Speed-Accuracy Trade-Off. Eine Reduktion der Auflösung von 600×600 Pixel auf 300×300 Pixel zog eine Reduktion der Laufzeit um

durchschnittlich 27.4% mit sich. Allerdings wurde die AP_{coco}-Metrik dabei im Schnitt um 15.88% gesenkt. Die Verringerung der AP_{coco}-Metrik wird daran begründet, dass Bilder mit einer höheren Auflösung eine signifikant bessere Detektion von kleinen Objekten zulassen. Die Detektion von großen Objekten wird ebenfalls begünstigt. Zudem wird festgestellt, dass eine hohe AP_s-Metrik positiv in Korrelation zu einer AP_l-Metrik steht. Umgekehrt ist dies jedoch nicht der Fall.

Effekt der Anzahl an Kandidaten-Boxen (für Two-Stage Verfahren)

Ein weiterer Faktor, der analysiert wird, ist die Anzahl an Kandidaten-Boxen, die während der Region-Proposal-Phase der Two-Stage-Verfahren generiert werden. Standardmäßig verwenden sowohl Faster R-CNN als auch R-FCN 300 Kandidaten-Boxen. Bei der Reduktion von 300 auf 50 Boxen für Faster R-CNN in Kombination mit dem Inception Resnet, ergab sich eine Reduktion der AP_{coco}-Metrik von 35.4% auf 33.98% und eine Reduktion der Laufzeit um Faktor drei. Ein ähnliches Verhalten wird für die anderen im Benchmark verwendeten Backbone-Architekturen beobachtet. Für R-FCN wird nur ein minimaler Faktor der Reduktion der Laufzeit beobachtet, was die Autoren auf den grundsätzlichen Aufbau der Region-Proposal-Phase der R-FCN Meta-Architektur zurückführen [52].

Floating Point Operations als Maß für die Berechnungskomplexität

Da eine Laufzeitmessung der Algorithmen stark von der zugrunde liegenden GPU-Architektur (sowie den Framework-Versionen) abhängt, werden in [52] anstelle dessen die FLOPs (bezogen auf Additionen und Multiplikationen) gemessen. Diese bieten eine architekturunabhängige Einschätzung der Berechnungskomplexität. Jedoch ist darauf hinzuweisen, dass kein linearer Zusammenhang zwischen den FLOPs und der tatsächlichen Laufzeit garantiert ist. Da für das in der Arbeit durchzuführende Benchmark eine feste GPU-Architektur sowie einheitliche Framework Versionen verwendet werden, ist ein fairer Vergleich bezüglich der Laufzeit der Verfahren gewährleistet. Zudem müssen für die Bereitstellung einer Entscheidungsbasis die Laufzeiten vorliegen, da sonst keine Richtwerte für die Wahl eines für Object Tracking geeigneten Verfahrens verfügbar sind. Auf die Analyse der FLOPs wird daher im Benchmark nicht näher eingegangen.

7.1.2 Einflussfaktoren im Benchmark

Basierend auf den in Kapitel 7.1.1 aufgeführten Aspekten, wird der Haupteinfluss auf den Speed-Accuracy Trade-Off auf folgende Faktoren zurückgeführt: Die verwendete Backbone-Architektur (bei Faster R-CNN und R-FCN), die Auflösung der Eingabebilder und die Anzahl der Kandidaten-Boxen (für Faster R-CNN). Zudem wird herausgestellt, dass die Metriken in Bezug auf die Größe der in den Eingabebildern auftretenden Objektinstanzen

gesondert zu betrachten sind. In diesem Kapitel wird erläutert, wie die in Kapitel 7.1.1 beschriebenen Einflussfaktoren im Benchmark betrachtet werden.

Effekt der Backbone-Architektur

Die resultierenden Verfahren der Filterung in Kapitel 5 haben die in Tabelle 5.2 spezifizierten Anforderungen zu erfüllen. Laut Anforderung Nr. 3 muss eine Open-Source Implementierung verfügbar sein. Alle verfügbaren Kombination aus Backbone- und Meta-Architektur sind in Tabelle B.1 unter Berücksichtigung der in Kapitel 5.1 gegebenen Erläuterungen angegeben. Für die gefilterten Meta-Architekturen existiert keine gemeinsame Schnittmenge an Backbone-Architekturen, die eine Analyse ähnlich zu [52] zulässt. Zudem geht aus den Artikeln der ausgewählten Meta-Architekturen ein jeweiliger Kontext der Anwendung hervor, der eine explizite Auswahl an Backbone-Architekturen vorsieht. Unter Betrachtung der aufgeführten Aspekte findet daher kein Vergleich des Effekts der verwendeten Backbone-Architektur statt.

Effekt der Objektgröße

In [52] wird ein signifikanter Unterschied zwischen den Metriken verschiedener Objektgrößen (bezogen auf die PF) dargelegt. Daher werden die Ergebnisse im Benchmark explizit auf diesen Unterschied hin untersucht. Um dies zu gewährleisten, werden in Kapitel 8.1 entsprechende Metriken ausgewählt, die für die Evaluation der Ergebnisse in Kapitel 8.2 zu betrachten sind.

Effekt der Anzahl an Kandidaten Boxen (für Two-Stage-Verfahren)

Die HTC-Architektur ist die einzige ausgewählte Architektur, die auf der Faster R-CNN-Architektur basiert. In der Grundkonfiguration sind standardmäßig 100 Kandidaten-Boxen vorgesehen. Daher kann die HTC-Architektur nicht gleichermaßen von einer Verringerung der Kandidaten-Boxen profitieren, wie es in [52] für Faster R-CNN der Fall ist. Ein Laufzeittest zeigte diesbezüglich nur eine marginale Verbesserung. Daher finden im Benchmark keine weiteren Untersuchungen bezüglich der Anzahl an Kandidaten-Boxen statt.

Effekt der Auflösung des Eingabebildes

Aus den Erläuterungen in Kapitel 7.1.1 geht ein nicht unerheblicher Einfluss der Auflösung auf die daraus resultierende Laufzeit der Object Detection-Algorithmen hervor. Daher gilt es, diesen Effekt näher im Benchmark zu betrachten. Relevant hierfür ist nur die Auflösung während der Inferenz. Für den YOLO-v3 ASFF und RFBNet Object Detection-Algorithmus ist die Auflösung über die Bezeichnung der Meta-Architektur gegeben: `@Auflösung`. Durch die Selektion der verschiedenen Architekturen des YOLO-v3 ASFF Object Detection-Algorithmus für das Benchmark, sind demnach folgende Auflösungen vertreten: $320 \times$

320, 416×416 , 608×608 und $800 \times$. Für den RFBNet Object Detection-Algorithmus wird nur die Auflösung 300×300 verwendet. Die andere zur Verfügung stehende Konfiguration des RFBNet mit der Auflösung 512×512 wird nicht für das Benchmark ausgewählt, da diese im Zuge der Filterung von Tabelle B.1 als nicht relevant zu betrachten ist. Für die Architekturen HTC und ATSS wird in der Originalkonfiguration nur eine Auflösung von 1333×800 für die Inferenz verwendet. Andere Auflösungen sind nicht in Tabelle B.1 aufgeführt. Um annähernd an die Laufzeitklassen von YOLO-v3 ASFF oder des RFBNet zu gelangen, ist mindestens eine Halbierung der Laufzeit nötig. Daher wird die Auflösung 666×400 ausgewählt, um das Potential einer reduzierten Auflösung und der damit verbundenen reduzierten Laufzeit zu untersuchen. Es wird angenommen, dass eine halbierte Auflösung zu einer geringeren AP-Metriken führt. Dies gilt es während der Auswertung der Ergebnisse näher zu betrachten.

7.2 Hyperparameteroptimierung und Modellierung

Für die Durchführung einer Modellierung auf Basis der Anwendung einer k-Fold Cross Validation gilt es, im Vorfeld für jeden Object Detection-Algorithmus eine Hyperparameteroptimierung vorzunehmen. Dafür wird zunächst ein Konzept ausgearbeitet, aus dem hervorgeht, welche Aspekte für die Durchführung der Hyperparameteroptimierung zu beachten sind. Darauf folgt für jeden Object Detection-Algorithmus eine detaillierte Beschreibung der verwendeten Konfigurationen von Hyperparametern und des sich daraus ergebenden Trainingsverlaufs. Auf Grundlage der jeweiligen Konfiguration mit den höchsten Validierungsmetriken wird abschließend für jede Meta-Architektur eine k-Fold Cross Validation auf Basis des Datensatzes $D2$ durchgeführt.

7.2.1 Erarbeitung des Konzepts zur Hyperparameteroptimierung

Als Grundlage zur Bewertung der Trainingsverläufe, die aus unterschiedlichen Konfigurationen der Hyperparameter hervorgehen, wird allein der Verlauf der AP@0.5-Metrik in Abhängigkeit der Epochen betrachtet. Die AP@0.5-Metrik bietet ein Maß für die Genauigkeit der resultierenden Modelle für jede Epoche. Die gängige Methode der Literatur ist, den Verlauf der Loss-Funktion zu betrachten. Da diese im Kontext der Object Detection unterschiedlich ausfällt und oft ein eher abstrakter Verlauf der Loss-Funktion zu beobachten ist, wird aus Gründen eines einheitlichen Anhaltspunktes die AP@0.5-Metrik zur Bewertung der Trainingsverläufe ausgewählt. Dafür ist jede der in den Abschnitten von Kapitel 7.2.2 angegebene Konfiguration der Object Detection-Algorithmen, auf den Folds des Splits $S1$ zu trainieren und die daraus resultierenden Modelle anschließend auf dem entsprechenden Validierungs-Fold zu evaluieren. Auf Basis der Hyperparameter H^* , aus denen das Modell mit der höchsten AP@0.5-Metrik hervorgeht, wird eine k-Fold Cross Validation angewendet. Hierfür ist ein Training und eine Validierung auf allen Splits $S1$

- S5 unter Anwendung der Hyperparameter H^* durchzuführen. Dies folgt dem in Kapitel 4.2 geschilderten adaptiertem Vorgehen zur Anwendung einer k-Fold Cross Validation. Die Metriken, die aus der k-Fold Cross Validation der jeweiligen Architektur hervorgehen, werden in Kapitel 8 detailliert diskutiert.

Auf Basis der Anpassung der Hyperparameter ergeben sich für jeden Object Detection-Algorithmus verschiedene Parameterkonfiguration. Diese sind in den Tabellen des jeweiligen Unterkapitels aufgeführt, indem die Hyperparameteroptimierung geschildert wird. Zudem ist für jede Parameterkonfiguration ein Verlauf des resultierenden Trainings, ebenfalls im jeweiligen Unterkapitel, zu finden. Gezeigt werden Graphen der AP@0.5-Metrik als Funktion über die Anzahl der Epochen, jeweils für die Trainingsdaten (gestrichelte Linie) und die Validierungsdaten (durchgezogene Linie). Zur Identifikation einer Hyperparameter-Konfiguration im Graphen ist für jede Konfiguration eine ID in der jeweiligen Tabelle spezifiziert. In den Legenden der jeweiligen Abbildungen ist jede der Konfigurations-IDs farblich gekennzeichnet, sowie der jeweilige Maximalwert (Punkt im Graph) der AP@0.5 angegeben. Der Maximalwert wurde unter Berücksichtigung aller Epochen des jeweiligen Trainingsverlaufs bestimmt.

Die aus der k-Fold Cross Validation resultierenden Trainingsverläufe sind für die jeweilige Architektur im Anhang in den Abbildungen D.1 - D.12 aufgeführt. Gezeigt wird, analog zu den Graphen der Hyperparameteroptimierung, der Verlauf der AP@0.5-Metrik für jeden der fünf Splits (schwarz). Zudem ist die gemittelte AP@0.5-Metrik aller Trainingsverläufe (blau), sowie der resultierende Maximalwert der AP@0.5-Metrik (Punkt im Graph) gekennzeichnet.

Nachfolgend werden Aspekte geschildert, die es bei der Anpassung für die Hyperparameter Epoche und Batch Size, den für die Bestimmung der Evaluierungsmetriken notwendigen Parametern T_{score} und T_{nms} sowie bei der Anwendung eines MS-Trainings zu berücksichtigen gilt.

Wahl der Epochen

In Kapitel 6 sind die Hyperparameter für jeden Object Detection-Algorithmus aufgeführt (Tabelle 6.1 bis Tabelle 6.4), die zum Training der Originalmodelle verwendet worden sind. Aus Kapitel 3.2 geht die Relevanz der LR als Parameter für den Optimierer sowie des damit verbundenen Scheduling hervor. Der Wert der LR wird in Abhängigkeit der Anzahl an Epochen von dem konfigurierten Scheduling-Verfahren angepasst, wodurch der Verlauf des Optimierungsprozesses bestimmt wird. Dabei sind die Parameter des Scheduling-Verfahrens in Abhängigkeit der Gesamtanzahl der Epochen festgelegt. Daher wird der Fokus der Hyperparametersuche auf das Finden einer geeigneten Gesamtanzahl von Epochen gelegt.

Durch Anwendung von Transfer Learning wird zunächst angenommen, dass die ursprüngliche Anzahl der Epochen reduziert werden kann. Diese Annahme gilt es, für jeden

Object Detection-Algorithmus zu validieren. Daher erfolgt das Training auf einer reduzierten Anzahl von Epochen in mehreren Stufen. Es ist anzumerken, dass für die Modellierung der Algorithmen nur eine begrenzte Bearbeitungszeit zur Verfügung steht und die gewählten Abstufungen für die Epochen daher abgeschätzt werden. Auf Basis der Gesamtanzahl an Epochen die für die jeweiligen Architekturen ausgewählt wird, sind die Parameter des entsprechenden LR-Schedulings entsprechend dem Faktor zwischen der neu konfigurierten und originalen Gesamtanzahl anzupassen.

Wahl der Batch Size

Ein weiterer wichtiger Parameter ist die Batch Size. Diese definiert die Anzahl der Trainingsbeispiele, die pro Optimierungsschritt verwendet werden. Für das Training der Modelle wurden eine GPU-Workstation und ein GPU-Server herangezogen. Die GPU-Workstation besitzt eine *NVIDIA P5000*, der GPU-Server drei virtuelle Maschinen, mit zwei, zwei und vier *NVIDIA V100*. Aufgrund der Komplexität der CNN-Architektur des HTC Object Detection-Algorithmus wird ein großer GPU Speicher benötigt. Daher ergibt sich für die Batch Size eine obere Grenze der Größe vier. Eine größere Batch Size hat Speicherüberläufe zur Folge. Um einen fairen Optimierungsprozess für alle Architekturen zu erlauben, wird die Batch Size für alle Architekturen auf den Wert vier festgelegt.

Es ist anzumerken, dass die virtuelle Maschine mit vier GPUs erst zu einem späten Zeitpunkt für das Training zur Verfügung stand und die Bestimmung der oberen Grenze auf Basis der Betrachtung der anderen Systeme vorgenommen worden ist. Zudem gilt es, auf Basis der Anwendung von Transfer Learning ein Finetuning der Gewichte vorzunehmen. Eine kleinere Batch Size erlaubt in diesem Kontext das gezielte Optimieren auf die von den Trainingsdaten abgebildeten Szenarien. Des Weiteren führt die Anwendung einer kleinen Batch Size nach [76] zu einer verbesserten Generalisierung sowie zu einem stabileren und zuverlässigeren Trainingsverhalten. Dies bestärkt die Wahl, den Wert der Batch Size auf die Größe vier festzulegen.

Wahl der Evaluierungsparameter

Neben den Modellparametern gilt es, als weitere Hyperparameter für jeden Object Detection-Algorithmus den Grenzwert T_{score} und den Grenzwert T_{nms} festzulegen. Der Grenzwert T_{score} ist in Kapitel 3 als Eingabeparameter für Algorithmus 1 anzugeben und dient der Filterung der Bounding-Boxen mit zu geringer Klassen-Score. Anhand dessen werden die detektierten Bounding-Boxen gefiltert, die sehr Wahrscheinlich zu einer FP-Detektion führen würden. Der T_{nms} dient im NMS-Algorithmus (siehe Algorithmus 3 im Anhang) der Filterung sich überschneidender Bounding-Boxen (Duplikaten) für die gleiche Objektinstanz. Demnach werden durch eine geeignete Wahl von T_{nms} ebenfalls FP-Detektionen verhindert. Die Wahl von T_{score} und T_{nms} beeinflusst damit die Ausprägung der Detektions-

onsgenauigkeit. T_{score} ist für jeden Object Detection-Algorithmus auf Basis einer Heuristik unter Anwendung der initialen Modelle auf Datensatz $D1$ ermittelt worden. Dafür wurde T_{score} schrittweise erhöht. Anschließend wurde das beste Verhältnis zwischen der Filterung von waren FP- und potentiellen TP-Detektionen gewählt. Für Object Tracking ist eine Ansammlung von Duplikaten zu vermeiden. Daher wird der Wert für T_{nms} für jeden Object Detection-Algorithmus auf den Wert 0.1 festgelegt.

Wahl der Auflösung

Die Object Detection-Algorithmen HTC, ATSS und YOLO-v3 ASFF nutzen in ihrer Originalkonfiguration ein MS-Training. Daraus resultieren auflösungsvariante Modelle, die eine erhöhte AP_{coco}-Metrik aufweisen. Da im Datensatz $D2$ alle Bilder das gleiche Format ausweisen, gilt es zu ermitteln, welcher Profit aus der Anwendung eines MS-Trainings gewonnen werden kann. Aus Erläuterungen in Kapitel 7.1.2 geht hervor, dass für ausgewählten Architekturen des HTC und ATSS Object Detection-Algorithmus ein Training mit halbierter Auflösung durchzuführen ist. Den daraus resultierenden Effekt gilt es näher im Rahmen des Speed-Accuracy Trade-Off zu untersuchen. Da nicht bekannt ist, wie die Konfiguration für MS-Training mit halbierter Auflösung auszusehen hat, wird ein MS-Training in diesen Fällen nicht näher betrachtet.

Ein MS-Training ist unter Angabe der Abkürzung MS in der Zeile *Auflösung Training* in den Tabellen der Hyperparameter-Konfigurationen spezifiziert. Alle Bilder des Datensatzes sind auf den zweiten Wert (Höhe) der angegebenen Auflösung zu skalieren. Die Breite darf bei der Skalierung nicht den zuerst angegebenen Wert übersteigen. Für ein MS-Training ist der Wertebereich angegeben, aus dem während des Trainings ein zufälliger Wert in regelmäßigen Abständen gewählt wird. Während der Auswertung der Evaluierungsmaßen wird die unter *Auflösung Inferenz* spezifizierte Auflösung verwendet. Ist bei der Auflösung anstelle eines Tuples nur ein einzelner Wert angegeben, handelt es sich um eine quadratische Auflösung.

7.2.2 Durchführung der Hyperparameteroptimierung

Auf Grundlage des in Kapitel 7.2.1 erarbeiteten Konzepts wird in den nachfolgenden Unterkapiteln die Hyperparameteroptimierung für jede selektierte Architektur geschildert. Die Reihenfolge richtet sich nach absteigender Sortierung der AP_{coco}-Metrik aus den Originalartikeln.

Hyperparameteroptimierung – HTC

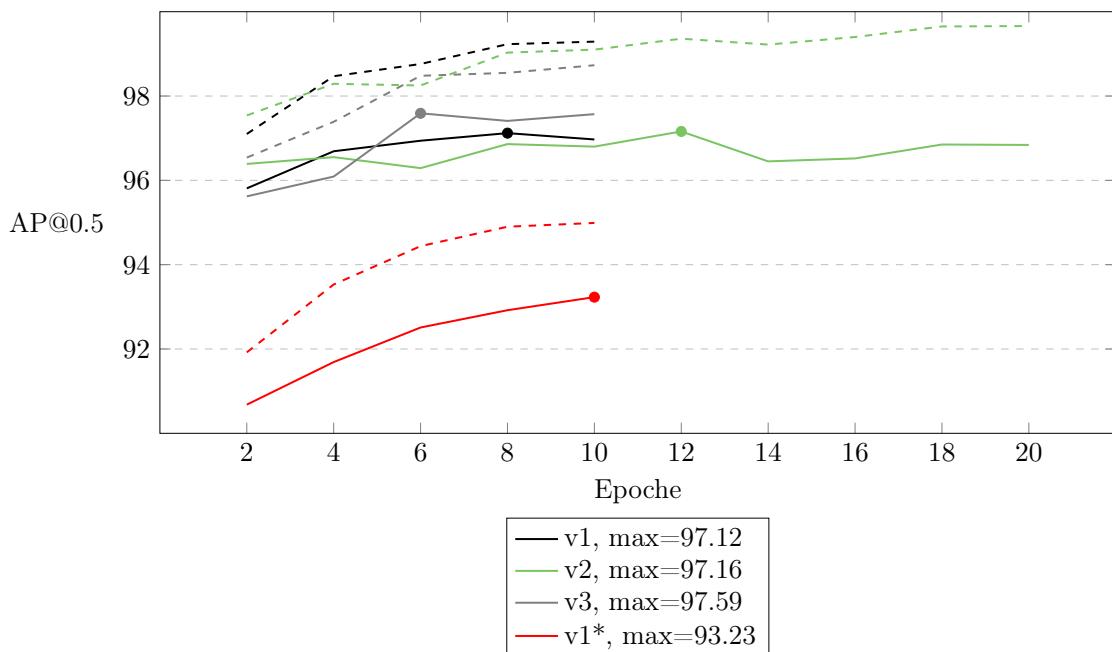
Für den HTC Object Detection-Algorithmus ergeben sich auf Basis der zuvor beschriebenen Vorgehensweise zur Anpassung der Hyperparameter sowie die als Einflussfaktor zu untersuchende verringerte Auflösung die in Tabelle 7.1 aufgeführten Parameterkonfigurationen.

Das Originalmodell nutzt ein schrittbasierteres LR-Scheduling. Im Vergleich zu Originalkonfigurationen wurden nur die Anzahl der Epochen und die damit verbundenen Schritte zum Scheduling der LR geändert. In Tabelle 7.1 werden daher nur die Parameter aufgeführt, die für das durchzuführende Training anzupassen sind. In Bezug auf die Original-Architektur existieren keine weiteren Parameter, die es im Kontext der Arbeit zu optimieren gilt. Auf Basis einer Analyse der FP-Detektionen des Modells, welches auf dem initialen Datensatz trainiert worden ist, wird für die Architekturen des HTC Object Detection-Algorithmus der Wert von T_{score} auf 0.5 festgelegt.

Das Originalmodell, welches zur Initialisierung der Gewichte herangezogen wurde, ist 20 Epochen unter Verwendung des in Tabelle 6.1 angegebenen LR-Scheduling trainiert worden. Als Abschätzung der Anzahl der Epochen wurde zunächst eine Halbierung der Originalepochen auf den Wert 10 vorgenommen. Die Schritte des LR-Schedulings galt es dementsprechend mit dem Faktor 0.5 zu multiplizieren und auf einen ganzzahligen Wert abzurunden. Die resultierende Konfiguration ist unter der Konfigurations-ID v1 aufgeführt. Um einen Anhaltspunkt für die Validität der Abschätzung zu erhalten, wurde eine Konfiguration mit 20 Epochen verwendet (Konfigurations-ID v2). Zur Analyse des Einflusses eines MS-Trainings wurde die Konfiguration mit der ID v3 in die Menge der Konfigurationen aufgenommen. Um Aussagen über den Effekt einer reduzierten Auflösung treffen zu können, wurde die Konfiguration mit der ID v1* eingesetzt. Da aus den in Kapitel 7.2.1 aufgeführten Aspekten eine Anwendung eines MS-Trainings für eine Konfiguration mit halbierter Auflösung ausgeschlossen wird, wurde die Konfiguration von v1* analog zu v1 formuliert.

Abbildung 7.1 zeigt den Trainingsverlauf der aus den Konfigurationen resultierenden Modellen. Auch hier steht eine durchgehende Linie für die Validierungsmetrik, die gestrichelte Linie für die Trainingsmetrik. Der maximale Wert der AP@0.5-Metrik für die Validierungsdaten beläuft sich auf 97.59% und wird unter Anwendung der Konfiguration mit der ID v3 erreicht. Damit ist die Abschätzung einer verringerten Gesamtanzahl von Epochen valide. Aus einem Vergleich der Trainings- und Validierungsmetriken von v2 geht zudem ab Epoche 12 ein Overfitting hervor, was die Wahl einer geringeren Gesamtanzahl von Epochen bestärkt. Des Weiteren geht aus einem Vergleich der Trainingsverläufe von v1 und v3, demnach der Anwendung eines MS-Training, ein positiver Effekt hervor. Daher wird die Konfiguration v3 zur Durchführung der k-Fold Cross Validation ausgewählt. Für die Konfiguration mit halbierter Auflösung ist im Vergleich zur Konfiguration v1 eine Differenz von 4.36 Prozentpunkten zu beobachten. Eine genaue Analyse des Speed-Accuracy Trade-Off erfolgt auf Basis der Ergebnisse der k-Fold Cross Validation. Die aus der Anwendung der k-Fold Cross Validation resultierenden Trainingsverläufe für die Konfigurationen v3 und v1* sind in den Abbildungen D.1 und D.2 aufgeführt.

	Konfigurations-ID			
	v1	v2	v3	v1*
Epochen	10	20	10	10
LR-Scheduling Schritte	8, 9	16, 19	8, 9	8, 9
Auflösung Training	1333×800	MS $1600 \times$ $\{400, \dots, 1400\}$	666×400	
Auflösung Inferenz	1333×800		666×400	
T_{score}		0.5		

Tabelle 7.1: Parameterkonfigurationen für HTC X-101-FD @1333x800.**Abbildung 7.1:** Trainingsverläufe der Konfigurationen für HTC X-101-FD @1333x800

Hyperparameteroptimierung – ATSS

Für das Benchmarking wurden zwei Architekturen des ATSS Object Detection-Algorithmus ausgewählt: *ATSS X-101-FD @1333x800* und *ATSS R-101-FD @1333x800*. Während der Modellierungen der initialen Modelle in Kapitel 5.3 wurde für die Architekturen eine erhöhte Dauer für den Trainingsprozess beobachtet. Aufgrund der begrenzt zur Verfügung stehenden Bearbeitungszeit wird daher die Hyperparameteroptimierung zunächst auf Basis der weniger komplexen Architektur *ATSS R-101-FD @1333x800* durchgeführt

und anschließend die daraus resultierende optimale Konfiguration für die *ATSS X-101-FD @1333x800*-Architektur validiert.

Verwendet wird ein schrittbasierter LR-Scheduling. Damit gilt es, eine geeignete Abschätzung der Gesamtanzahl an Epochen vorzunehmen und die davon abhängigen Schritte des LR-Schedulings entsprechend zu skalieren. Dies wurde nach dem in Kapitel 7.2.1 geschildertem Vorgehen durchgeführt. Da die Warm-Up-Epochen eine wichtige Rolle für eine stabile Initialisierung des Trainings spielen, wurde die für die Original-Architektur spezifizierte Anzahl der Warm-Up-Epochen verwendet. Der Trainingsprozess der Original-Architektur ist in Abhängigkeit der Optimierungsschritte und nicht, wie üblich, in Abhängigkeit der Epochen definiert. Daher galt es, die Gesamtschritte des Originalmodels, welches zur Initialisierung der Gewichte herangezogen wird, in Gesamtepochen zu transferieren. Das Vorgehen hierzu wurde bereits in Kapitel 6.2.3 erläutert. Daraus resultiert eine Gesamtanzahl von ungefähr 25 Epochen. Auf Grundlage dessen wurde für das durchzuführende Training die Gesamtanzahl der Epochen zunächst auf den Wert 8 abgeschätzt. Anschließend wurde zur Validität der Abschätzung ein Training mit 16 und ein Training mit 32 Epochen durchgeführt. Eine Gesamtanzahl von 32 Epochen übersteigt die in der Original-Architektur spezifizierte Gesamtanzahl, da diese jedoch im Rahmen der Arbeit umgerechnet worden ist, erlaubt die Anwendung einer Gesamtanzahl von 32 Epochen eine Fehlertoleranz für eine obere Abschätzung. In Bezug auf die Original-Architektur existieren keine weiteren Parameter, die es im Kontext der Arbeit zu optimieren gilt. Insgesamt resultieren aus den geschilderten Aspekten die in Tabelle 7.2 aufgeführten Konfigurationen. Auch hier werden nur die Parameter angegeben, die für das durchzuführende Training angepasst worden sind. Auf Basis einer Analyse der FP-Detektionen des Modells, welches auf dem initialen Datensatz trainiert worden ist, wird für alle Architekturen des ATSS Object Detection-Algorithmus der Wert von T_{score} auf 0.5 festgelegt. Alle Konfigurationen wurden mit Gewichten eines Modells der Originalkonfiguration initialisiert, welches unter [156] zur Verfügung steht.

Abbildung 7.2 zeigt den Trainingsverlauf der aus den Konfigurationen resultierenden Modelle. Die Konfiguration v3 weist die höchste AP@0.5-Metrik für die Validierungsdaten auf und beläuft sich auf 93.72%. Des Weiteren geht aus einem Vergleich der Trainingsverläufe von v3 und v4 hervor, dass die Anwendung eines MS-Trainings keinen positiven Effekt auf die Genauigkeit der Detektion aufweist. Daher wird die Konfiguration v3 zur Durchführung der k-Fold Cross Validation ausgewählt, sowie als Anhaltspunkt für ein Training der Architektur *ATSS X-101-FD @1333x800* herangezogen. Für die Konfiguration von *ATSS R-101-FD @1333x800* mit halbierter Auflösung (v3*) ist im Vergleich zur Konfiguration v3 eine Differenz von 9.96 Prozentpunkten zu beobachten. Eine genaue Analyse des Speed-Accuracy Trade-Off erfolgt auf Basis der Ergebnisse der k-Fold Cross Validation. Die aus der k-Fold Cross Validation resultierenden Trainingsverläufe für die Konfigurationen v3 und v3* sind in den Abbildungen D.5 und D.6 aufgeführt.

In Abbildung C.1 sind die Trainingsverläufe der Architektur *ATSS X-101-FD @1333x800* aufgeführt, die sich auf Basis der Anwendung der Hyperparameter-Konfiguration v1 und v3 sowie v3* ergeben. Diese weisen einen, im Vergleich zur Architektur *ATSS R-101-FD @1333x800*, ähnlichen Verlauf auf. Daher wird die Validität der Anwendung der Hyperparameter-Konfiguration bestätigt. Dennoch ergibt sich für die Hyperparameter-Konfiguration v3 der Architektur *ATSS X-101-FD @1333x800* eine niedrigere AP@0.5-Metrik auf den Validierungsdaten, als für die Hyperparameter-Konfiguration v3 der Architektur *ATSS R-101-FD @1333x800*. Aus einem gesonderten Vergleich der AP@0.5 auf Trainingsdaten ist der umgekehrte Fall zu beobachten. Daher kann ein leichtes Overfitting für die Architektur *ATSS X-101-FD @1333x800* angenommen werden. Um diesen Effekt genauer zu untersuchen, gilt es, die aus der k-Fold Cross Validation resultierenden Metriken für beide Architekturen gegenüberzustellen.

Auf Basis aller Validierungsmetriken der k-Fold Cross Validation wird die Standardabweichung σ bestimmt, um zu kontrollieren, ob es beim Training auf den einzelnen Splits zu einem Overfitting gekommen ist. Für all vier Architekturen des ATSS Object Detection-Algorithmus liegt ein $\sigma < 1$ vor. Zu einem Overfitting auf einen Split ist damit nicht gekommen. Daher lassen sich auf Basis der ermittelten Metriken generalisierte Aussagen treffen.

	Konfigurations-ID				
	v1	v2	v3	v4	v3*
Epochen	8	16	32	32	32
LR-Scheduling Schritte	5, 7	10, 14	20, 28	20, 28	20, 28
Auflösung Training	1333 × 800			MS 1333× {640, ..., 800}	666 × 400
Auflösung Inferenz	1333 × 800				666 × 400
T_{score}	0.5				

Tabelle 7.2: Parameterkonfigurationen für ATSS R-101-FD @1333x800.

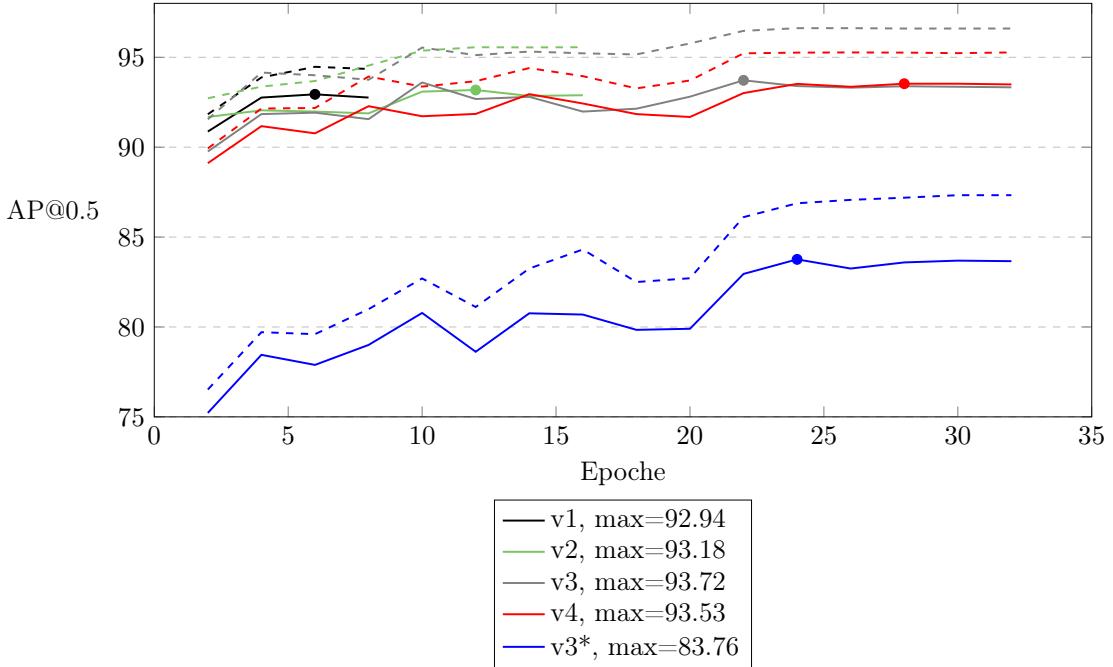


Abbildung 7.2: Trainingsverläufe der Konfigurationen für ATSS R-101-FD @1333x800

Hyperparameteroptimierung – YOLO-v3 ASFF

Für das Benchmarking wurden insgesamt fünf verschiedene Architekturen des YOLO-v3 ASFF Object Detection-Algorithmus ausgewählt: *YOLOv3-ASFF* D-53 @800*, *YOLOv3-ASFF* D-53 @608*, *YOLOv3-ASFF* D-53 @416*, *YOLOv3-ASFF D-53 @416* und *YOLOv3-ASFF* D-53 @320*. Die Architektur *YOLOv3-ASFF* D-53 @608* weist die höchste AP@0.5-Metrik auf den Validierungsdaten des Datensatzes *D1* auf (siehe Kapitel 5.3.3) und wird daher als Referenzsystem herangezogen, um eine Hyperparameteroptimierung für alle Architekturen des YOLO-v3 ASFF Object Detection-Algorithmus durchzuführen. Die resultierenden Hyperparameter sind anschließend für die anderen Architekturen zu validieren. Auf Basis einer Analyse der FP-Detektionen des Modells, welches auf dem initialen Datensatz trainiert worden ist, wird für alle Architekturen des YOLO-v3 ASFF Object Detection-Algorithmus der Wert von T_{score} auf 0.7 festgelegt.

Der YOLO-v3 ASFF Object Detection-Algorithmus nutzt ein cosinus-basiertes LR-Scheduling. Die dafür notwendige Anzahl der Epochen wurde zunächst auf 40 abgeschätzt. Anschließend wurde zur Validität der Abschätzung ein Training mit 72 und ein Training mit 100 Epochen durchgeführt. Damit wird kontrolliert, ob für ein Training unter Anwendung eines Transfer Learning eine derartige Reduzierung der Gesamtanzahl der Epochen angenommen werden darf. Da die Warm-Up-Epochen eine wichtige Rolle für eine stabile Initialisierung des Trainings spielen, wird für alle Konfiguration die originale Anzahl der Warm-Up-Epochen verwendet. Aufgrund der reduzierten Gesamtzahl an Epochen wird der Start des LR-Scheduling auf den Wert 10 halbiert. Eine Anpassung des Startwerts im

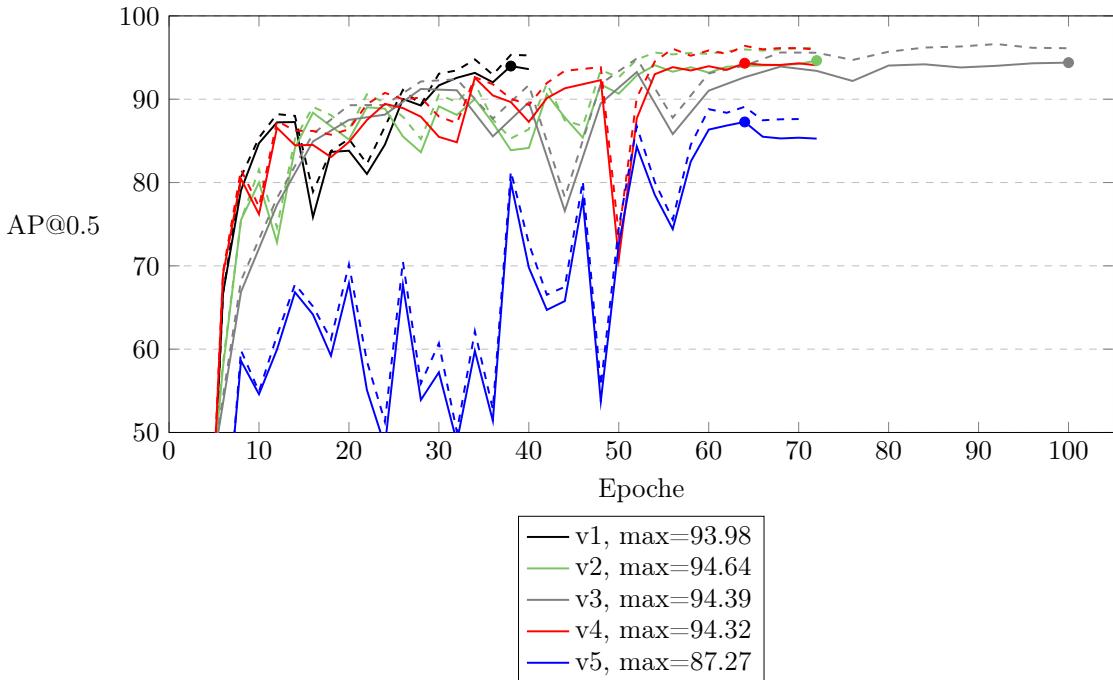
Verhältnis der Originalparameter ist nicht möglich, ohne die Anzahl der Warm-Up-Epochen zu unterschreiten. Daher wird der Wert 10 für den Start des Cosinus LR-Scheduling als untere Schwelle abgeschätzt. Die verwendete Auflösung ist über die Bezeichnung der Architektur gegeben. Weitere Auflösungen sind entsprechend über die Anwendung der anderen oben genannten Architekturen im Benchmark enthalten.

Die aus den zuvor geschilderten Aspekten hervorgehenden Konfigurationen sind unter den IDs v1 - v3 in Tabelle 7.3 ausgeführt. Alle Konfigurationen wurden mit Gewichten eines Modells der Originalkonfiguration initialisiert, welches unter [114] zur Verfügung steht. In Abbildung 7.3 sind die dazugehörigen Trainingsverläufe aufgeführt. Die Konfiguration mit der ID v2 weist die höchste AP@0.5-Metrik auf. Um den Effekt eines MS Trainings, sowie den Effekt eines Mix-Up Trainings aufzuzeigen, wurden auf Grundlage von v2 zwei weitere Trainingsprozesse mit den Konfigurations-IDs v4 und v5 durchgeführt. Insgesamt ergeben sich damit für *YOLOv3-ASFF* D-53 @608* Parameterkonfigurationen v1 - v5. Aus der Analyse der Trainingsverläufe geht keiner positiver Effekt eines Mix-Up Trainings oder eines MS-Trainings hervor. In Bezug auf die Original-Architektur existieren keine weiteren Parameter, die es im Kontext der Arbeit zu optimieren gilt. Unter Betrachtung aller geschilderten Aspekte wird daher die Konfiguration mit der ID v2 ausgewählt, um die in Kapitel 4.2 geforderte k-Fold Cross Validation durchzuführen.

Für die übrigen Architekturen *YOLOv3-ASFF* D-53 @800*, *YOLOv3-ASFF* D-53 @416*, *YOLOv3-ASFF D-53 @416* und *YOLOv3-ASFF* D-53 @320* werden die gleichen Parameterkonfiguration angewandt wie für die Architektur *YOLOv3-ASFF* D-53 @608*, um eine Abschätzung der Gesamtanzahl der Epochen zu bestimmen. Eine gesonderte Analyse eines Mix-Up oder MS-Training wird nicht durchgeführt, da kein positiver Effekt beobachtet wurde. Die Validität der Anwendung der gleichen Parameterkonfiguration gilt es, anhand des resultierenden Trainingsverläufe für jede Architektur zu analysieren. Diese sind im Anhang in den Abbildungen C.2 - C.5 aufgeführt. Für jede der Architekturen weist, wie auch für *YOLOv3-ASFF* D-53 @608*, die Parameterkonfiguration v2 die höchste AP@0.5-Metrik auf. Die Anwendung der gleichen Konfigurations-ID ist damit valide. Daher wird für jede Architektur die Parameterkonfiguration v2 ausgewählt, um die in Kapitel 4.2 geforderte k-Fold Cross Validation für die jeweilige Architektur durchzuführen.

Die Trainingsverläufe der durchgeführten k-Fold Cross Validation sind für alle Architekturen des YOLO-v3 ASFF Object Detection-Algorithmus in den Abbildungen D.7 - D.11 aufgeführt. Auf Basis der ermittelten Standardabweichungen σ ist kein Overfitting zu erkennen. Damit lassen sich auf Basis der ermittelten Metriken generalisierte Aussagen treffen.

	Konfigurations-ID				
	v1	v2	v3	v4	v5
Epochen	40	72	100	72	72
LR-Scheduling Start	10 Epochen				
Auflösung Training	608			MS {480, ..., 800}	
Auflösung Inferenz	608				
Mix-Up Training	nein	nein	nein	nein	ja
T_{score}	0.7				

Tabelle 7.3: Parameterkonfigurationen für YOLOv3-ASFF* D-53 @608.**Abbildung 7.3:** Trainingsverläufe der Konfigurationen für YOLOv3-ASFF* D-53 @608

Hyperparameteroptimierung – RFBNet

Für das Benchmarking wurde die Architektur *RFBNet VGG-16 @300* des RFBNet Object Detection-Algorithmus ausgewählt. Verwendet wird ein schrittbares LR-Scheduling. Damit gilt es, eine geeignete Abschätzung der Gesamtanzahl an Epochen vorzunehmen und die davon abhängigen Schritte des LR-Schedulings entsprechend zu skalieren. Da die Warm-Up-Epochen eine wichtige Rolle für eine stabile Initialisierung des Trainings spie-

len, wurde die für die Original-Architektur spezifizierte Anzahl der Warm-Up-Epochen verwendet. In Bezug auf die Original-Architektur existieren keine weiteren Parameter, die es im Kontext der Arbeit zu optimieren gilt. Das RFBNet nutzt kein MS-Training. Eine gesonderte Betrachtung einer anderen als der über die Architektur spezifizierte Auflösung ist nicht durchzuführen. Dies geht aus den in Kapitel 7.1.2 geschilderten Aspekten hervor. Als Abschätzung der Gesamtanzahl der Epochen wurde, ähnlich zu YOLO-v3 ASFF, zunächst ein Wert von 40 angenommen. Anschließend wurde zur Validität der Abschätzung ein Training mit 72, 100 und 150 Epochen durchgeführt.

Zur Initialisierung der Gewichte wurde ein Modell der Originalkonfiguration herangezogen, welches unter [72] zur Verfügung steht. Dieses wurde basierend auf den Daten des Pascal VOC-Datensatz trainiert. Ein Modell, welches auf dem MS COCO test-dev Datensatz trainiert worden ist, stand nicht frei zur Verfügung. Auf Basis einer Analyse der FP-Detektionen des Modells, welches auf dem initialen Datensatz trainiert worden ist, wird für alle Architekturen des RFBNet Object Detection-Algorithmus der Wert von T_{score} auf 0.5 festgelegt. Insgesamt ergeben sich für die Architektur *RFBNet VGG-16 @300* die in Tabelle 7.4 aufgeführten Konfigurationen. Abbildung 7.4 zeigt den Trainingsverlauf der aus den Konfigurationen resultierenden Modelle. Der maximale Wert der AP@0.5-Metrik für die Validierungsdaten beläuft sich auf 68.42% und wird unter Anwendung der Konfiguration mit der ID v4 erreicht. Daher wird diese ausgewählt, um die in Kapitel 4.2 geforderte k-Fold Cross Validation durchzuführen.

In Abbildung D.12 sind die Trainingsverläufe der k-Fold Cross Validation für die Architektur *RFBNet VGG-16 @300* aufgeführt. Auf Basis der ermittelten Standardabweichung σ ist kein Overfitting zu erkennen. Damit lassen sich auf Basis der ermittelten Metriken generalisierte Aussagen treffen.

RFBNet VGG-16 @300Hyperparametertuning				
Konfigurations-ID	v1	v2	v3	v4
Epochen	72	40	100	150
LR-Scheduling Schritte	24, 32	43,57	60, 80	90, 120
T_{score}			0.5	

Tabelle 7.4: Parameterkonfigurationen für RFBNet VGG-16 @300.

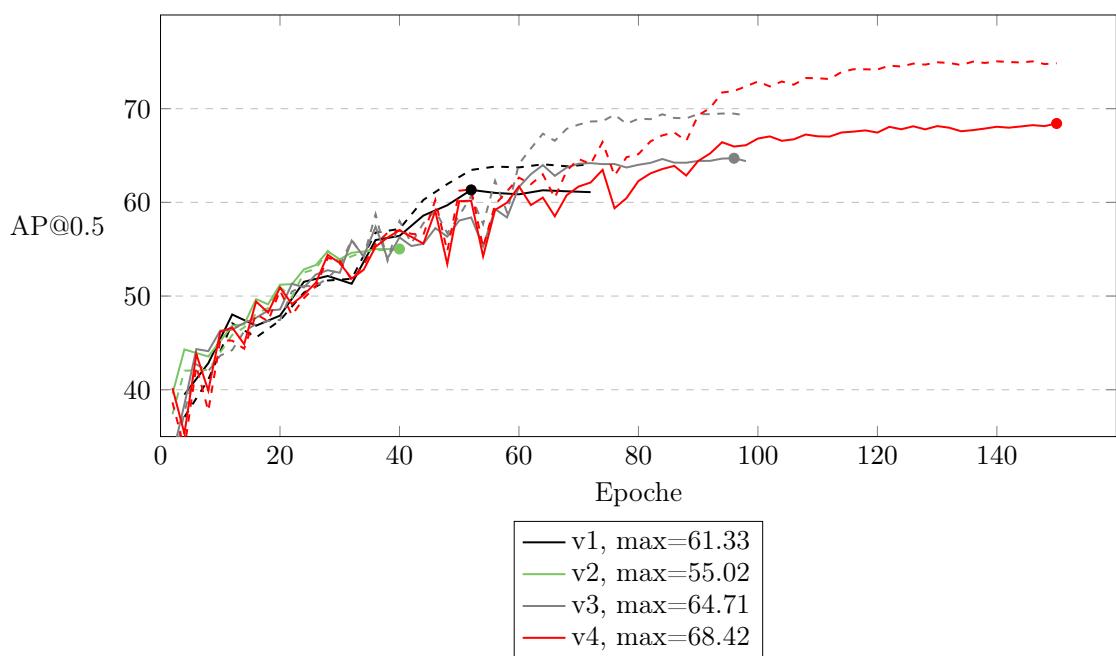


Abbildung 7.4: Trainingsverläufe der Konfigurationen für RFBNet VGG-16 @300

Kapitel 8

Auswertung der Ergebnisse

Dieses Kapitel dient der Auswertung der im Benchmarking erzielten Ergebnisse. Zunächst wird ein Evaluationskonzept erarbeitet, aus dem eine Spezifizierung der im Benchmark zu verwendeten Metriken zur Messung der Genauigkeit sowie die Spezifizierung der Laufzeitmessung hervorgeht. Danach erfolgt eine detaillierte Auswertung der definierten Metriken auf Basis der Ergebnisse der k-Fold Cross Validation. Abschließend wird der Speed-Accuracy Trade-Off auf Grundlage einer finalen Gegenüberstellung der Evaluierungsmetriken und Laufzeitmessungen ermittelt. Dieser untersteht dem Kontext der Anwendung von State-of-the-Art Object Detection-Algorithmen auf Bilddaten der Intralogistik. Damit wird eine Entscheidungsbasis unter Einhaltung der in Tabelle 5.2 definierten Anforderungen zur Verfügung gestellt, um eine für Object Tracking geeignete Architektur auswählen zu können.

8.1 Erarbeitung des Evaluationskonzepts

Nachfolgend wird zunächst ein Evaluationskonzept ausgearbeitet, das definiert, welche Metriken zur Bewertung der Genauigkeit der Architekturen herangezogen werden. Diese sind unter Anwendung der in Kapitel 7.2 durchgeführten k-Fold Cross Validation für jedes Modell auszuwerten. Es wird insbesondere der Kontext beachtet, dass auf Grundlage der ausgewählten Metriken eine Entscheidungsbasis für die Auswahl eines für Object Tracking geeigneten Verfahrens hervorgeht. Anschließend wird das Konzept beschrieben, nach dem die für den Speed-Accuracy Trade-Off notwendige Laufzeitmessung der jeweiligen Architekturen statt gefunden hat.

8.1.1 Spezifizierung der Benchmark-Metriken

Ziel des in der Arbeit durchzuführenden Benchmarkings ist es, den Speed-Accuracy Trade-Off der in Kapitel 5.3 selektierten Verfahren im Kontext von Bilddaten aus der Intralogistik zu analysieren. Auf Grundlage der Ergebnisse wird eine Entscheidungsbasis zur Verfügung

gestellt, aus der die Eignung der Kopplung eines Object Detection-Algorithmus an einen Object Tracking-Algorithmus hervorgeht. In Kapitel 3.5 wird die AP-Metrik definiert. Diese gilt als Maß, um die Detektionsgenauigkeit von aktuellen State-of-the-Art Object Detection-Algorithmen zu vergleichen. Grundlage der Berechnung sind die PR- und RC-Metrik, welche jedoch allgemein nicht in den Artikeln der State-of-the-Art Verfahren angegeben werden. Eine Angabe der PR- und RC-Metrik ist jedoch relevant für die Kopplung eines Object Detection-Algorithmus an einen Object Tracking-Algorithmus. Im Nachfolgenden Abschnitt werden Aspekte geschildert, aus denen diese Relevanz hervorgeht. Abschließend werden alle Metriken, die für die Auswertung der Ergebnisse des Benchmarking verwendet werden, in der Menge $M_{benchmark}$ festgehalten.

Die AP-Metrik ist ein Bewertungskriterium, um die Genauigkeit der Detektion als Mittel über die PR- und RC-Metrik auszudrücken. Für die Konfiguration von Tracking-Algorithmen sind jedoch detaillierte Informationen über die PR- und RC-Metrik notwendig. Dies lässt sich anhand des nachfolgenden Szenarios begründen. Es ist möglich, dass die AP-Metrik für Verfahren mit einem hohen RC-Metrik und niedriger PR-Metrik einen ähnlichen Wert aufweist wie für Verfahren mit einer niedrigen RC-Metrik und einer hohen PR-Metrik. Für die verschiedenen Verfahren müssen die Parameter des Object Tracking-Algorithmus trotz ähnlicher AP_{coco}-Metrik jedoch unterschiedlich konfiguriert werden. Im Fall einer hohen RC-Metrik und niedriger PR-Metrik werden viele Objekte erkannt, allerdings darf aufgrund der Ungenauigkeit nicht jede Detektion in die Berechnung der Tracking-ID einfließen. Im umgekehrten Fall einer niedrigen RC-Metrik und hoher PR-Metrik kann der Object Tracking-Algorithmus jede Detektion verwenden, da mit hoher Wahrscheinlichkeit davon ausgegangen werden kann, dass es sich um eine korrekte Detektion handelt. Demnach muss die Ausgabe der Object Detection-Algorithmen je nach Ausprägung der PR- und RC-Metriken unterschiedlich gewichtet werden. Dies erfolgt durch entsprechende Parameterkonfiguration im Object Tracking-Algorithmus. Für die Evaluation der Ergebnisse der k-Fold Cross Validation werden damit die PR- und RC-Metrik in die $M_{benchmark}$ aufgenommen.

Diese sind auf Grundlage eines IOU-Grenzwerts von $T_{iou} = 0.5$ zu berechnen, dessen Wahl nachfolgend begründet wird. Aus einer Analyse in [52] kann eine (fast) perfekte lineare Korrelationen zwischen den Metriken AP@0.5 und AP@0.75, verglichen mit der AP_{coco}-Metrik, angenommen werden. Zudem wird bei Tracking-Algorithmen eine leicht verschobene Detektion der Bounding-Boxen gegenüber einer fehlenden Detektion präferiert, da fehlende Detektionen die Generierung der Tracking-IDs negativ beeinflussen. Es besteht die Gefahr, einen Track über die Zeit zu verlieren. Zudem beruht die Berechnung aller Metriken maßgeblich auf den vorliegenden Ground-Truth Annotationen. Da diese manuell erstellt werden, kann es, trotz vordefiniert Regeln, zu Abweichungen in der Art und Weise des Annotierens für die gleichen Szenarien kommen. Ein zu hoher IOU-Grenzwert ist demnach stark anfällig für Abweichungen in den Ground-Truth-Bounding-Boxen, wodurch

ein vergrößertes Risiko bei der Einschätzung von TP und FP resultiert, welche die Basis der Berechnung aller Metriken bilden.

Unter Betrachtung aller aufgeführten Aspekte bietet damit ein Grenzwert von $T_{iou}=0.5$, gegenüber einem Grenzwert von $T_{iou}=0.75$, oder $T_{iou}^* = \{0.5, \dots, 0.95\}$ (siehe Kapitel 3.5, die bestmögliche Einschätzung von korrekten Detektionen für den Object Tracking-Algorithmus. Damit gilt es lediglich die AP@0.5-Metrik zu betrachten. Um zusätzlich Aussagen zur Genauigkeit der Detektion von unterschiedlichen Größen der Objekte zu erlauben, werden die Metriken AP_s , AP_m und AP_l verwendet. Für diese wird als Grundlage der Berechnung, abweichend von der ursprünglichen Definition im MS COCO Benchmark, nur der ausgewählte IOU-Grenzwert von $T_{iou}=0.5$ herangezogen. Dies ist notwendig, um gemeinsame Aussagen im Zusammenhang mit den Metriken RC und PR zuzulassen. Daher werden die Metriken AP_s , AP_m und AP_l für das Benchmarking neu spezifiziert zu AP_s^* , AP_m^* und AP_l^* . Insgesamt wird die Menge aller Metriken im Benchmarking definiert als $M_{benchmark} = \{RC, PR, AP_s^*, AP_m^*, AP_l^*\}$. Zudem sind alle AP-Metriken aus $M_{benchmark}$ auf Basis der adaptierten Berechnung für die AP-Metrik zu bestimmen (siehe Formel 3.10).

8.1.2 Spezifizierung der Laufzeitmessung

Für alle Implementierungen der im Benchmark verwendeten Architekturen wird das Deep Learning Framework PyTorch genutzt. Dieses bietet eine Timing-Bibliothek, die die Messung der reinen GPU-Zeit eines CNN-Modells ermöglicht. Als Referenzsystem der Laufzeitmessung wird die in Kapitel 2.2.2 spezifizierte Workstation verwendet. Diese verfügt über vier Kameras und ist mit den in Kapitel 5.3.1 Framework Versionen für cuDNN, CUDA und PyTorch ausgestattet. Im Rahmen des Projekts bei Fraunhofer IML gilt es, eine parallele Inferenz zu realisieren. Hierfür stehen zwei Ansätze zur Verfügung:

Variante 1 (synchron):

Die Worker-Threads werden synchronisiert. In einem von Gesamtsystem vorgegebenen Takt wird von jedem der Worker-Threads ein Kamerabild abgefragt. Die resultierenden Bilder gilt es, vom Gesamtsystem in eine Batch zusammen zu führen. Anschließend wird die so aufgebaute Batch von einem Modell des konfigurierten Object Detection-Algorithmus verarbeitet.

Variante 2 (asynchron):

Es werden vier Modelle eines Object Detection-Algorithmus parallel ausgeführt. Diese erhalten ihre Eingabebilder asynchron über die jeweiligen Worker-Threads, die für die Bereitstellung der einzelnen Kameras zuständig sind.

Der Umfang für die Entwicklung eines Systems zur Laufzeitmessung, welches die in Variante 1 geschilderten Begebenheiten erfüllt, übersteigt den Rahmen der Arbeit. Für Variante 2 ergab sich im Rahmen der Entwicklung folgende Problematik: Eine parallele Inferenz von

mehreren Modellen ist mit der aktuell zur Verfügung stehenden Version des PyTorch Frameworks nur sehr eingeschränkt möglich und mit sehr hohem Aufwand verbunden [105]. Zudem müssen für beide Varianten Seiteneffekte berücksichtigt werden, die sich aus der Gesamtarchitektur der existierenden Software bei Fraunhofer IML ergeben, beispielsweise das Laden der Bilder in den GPU-Speicher oder das Abfragen der Bilder von den Kameras. Im Rahmen der Arbeit wird zu Sicherstellung einer fairen Gegenüberstellung der Laufzeiten der Object Detection-Algorithmen nur die nativ gemessenen GPU-Laufzeit herangezogen. Diese wird in FPS angegeben. Der Messvorgang ist wie folgt aufgebaut:

Insgesamt sind die Resultate über fünf Durchläufe zu mitteln. Dies dient der Ermittlung möglicher Ausreißer, die sich aufgrund von unterschiedlichen Systemauslastungen ergeben können. Jeder Durchlauf umfasst eine Sequenz von 500 Einzelbildern. In einem Zeitabstand von zwei Sekunden wird der Object Detection-Algorithmus jeweils mit einer Batch Size=1 ausgeführt. Gemessen wird die reine Laufzeit des Modells auf der GPU unter Verwendung der von PyTorch zur Verfügung gestellten Timing Bibliothek. Der Zeitabstand ist nötig, um etwaige Seiteneffekte der Implementierung des Gesamtsoftwarearchitektur nicht in die Berechnung der Laufzeit einfließen zu lassen. Am Ende der Durchlaufs wird der Mittelwert über die Laufzeit der Inferenz für die Einzelbilder ermittelt. Davon ausgenommen sind die Laufzeiten der ersten 10 Bilder, da der Start der Inferenz mit einem initialen Overhead verbunden ist. Der Einbezug dieses Overheads würde die Laufzeitmessung verfälschen. Die finale Angabe der FPS beruht auf dem Mittelwert aller Durchläufe.

Auf Grundlage des geschilderten Vorgehens geht eine einheitliche Ermittlung der nativen GPU-Laufzeit hervor. Diese erlaubt eine faire Gegenüberstellung der Geschwindigkeiten für den final aufzuzeigenden Speed-Accuracy Trade-Off der im Benchmark verwendeten Architekturen.

8.2 Gegenüberstellung der Metriken

In Kapitel 8.1.1 wird die Menge aller im Benchmark zu betrachtenden Metriken $M_{benchmark}$ spezifiziert. Nachfolgend werden die Metriken aus $M_{benchmark}$ für alle Architekturen des Benchmarks evaluiert. Diese wurden für jede Klasse der in Kapitel 4.1 definierten Menge K ermittelt. Aus Gründen der Übersichtlichkeit sind die detaillierten Ergebnisse im Anhang in die Tabellen E.1 - E.12 ausgelagert. Darin aufgeführt sind die Metriken aus $M_{benchmark}$ getrennt für jede Objektklasse. Zur Kennzeichnung der entsprechenden Metriken sind diese mit dem Index k versehen. Diesen gilt es durch den Bezeichner der entsprechend gelisteten Objektklasse zu ersetzen.

Des Weiteren sind Metriken der Tabellen jeweils als Mittel über die Validierungsmetriken der Splits für die jeweilige k-Fold Cross Validation angegeben. Dabei wurden zunächst die Einzelergebnisse der jeweiligen Durchläufe der k-Fold Cross Validation akkumuliert und anschließend durch die Gesamtanzahl der Durchläufe geteilt. Im Kontext der k-Fold Cross

Validation gilt es zu ermitteln, ob ein Ungleichgewicht zwischen den Modellierungen der Splits vorliegt. Ist dies der Fall, muss bei der Ermittlung des Mittelwerts der Metriken eine Gewichtung vorgenommen werden. Aus den Standardabweichungen der AP@0.5-Metrik, die im Zuge der k-Fold Cross Validation für jede Architektur bestimmt worden sind, geht ein Gleichgewicht der Modellierung hervor. Daher ist keine Gewichtung vorzunehmen. In Kapitel 4.2 ist eine Verteilung des Datensatzes *D2* angegeben. Auf dessen Grundlage wurde die k-Fold Cross Validation durchgeführt. Aus Tabelle 4.1 geht eine unterschiedliche Verteilung zwischen Objektklassen und unterschiedlichen Größenklassen der Objekte hervor. Die Größe ist dabei bezogen auf die PF (Pixelfläche). Für die Evaluierung sind diese Verteilungen gezielt zu betrachten. Insbesondere die Objektklasse *Gabelstapler* weist im Vergleich zu den anderen Objektklassen eine sehr viel niedrigere Abdeckung von Ground-Truth-Daten auf. Zudem ist anzumerken, dass die Metrik AP_l^* für die Klassen *Person* und *Hubwagen* nicht im Benchmark betrachtet werden, da aus Tabelle 4.1 eine sehr geringe bis nicht vorhandene Anzahl von Instanzen dieser Klassen hervorgeht.

Als Übersicht des Gesamtergebnisses der k-Fold Cross Validation fasst Tabelle 8.1 die Metriken aller Architekturen zusammen. Es sind nur die AP-Metriken aus $M_{benchmark}$ angegeben. Detailergebnisse sind den Tabellen E.1 - E.12 zu entnehmen. In den nachfolgenden Unterkapiteln erfolgt eine detaillierte Betrachtung der Evaluierungsmetriken, separat für die im Benchmark verwendeten Object Detection-Algorithmen. Die Reihenfolge der Beschreibung richtet sich nach absteigender Sortierung der AP_{coco} -Metrik aus den Originalartikeln. Wenn von Metriken spezieller Klassen gesprochen wird, ist die jeweilige Metrik mit dem Index k (Klassenlabel) in der entsprechende Tabelle im Anhang gemeint.

Model	AP_s^*	AP_m^*	AP_l^*	AP@0.5
HTC X-101-FD @1333x800	95.31	98.84	99.05	97.30
HTC X-101-FD @666x400	87.68	98.50	98.89	93.04
ATSS X-101-FD @1333x800	88.24	97.05	97.60	93.58
ATSS X-101-FD @666x400	73.86	94.67	97.04	83.80
ATSS R-101-FD @1333x800	88.67	97.10	97.46	93.28
ATSS R-101-FD @666x400	73.33	94.56	97.50	83.59
YOLOv3-ASFF* D-53 @800	92.24	97.52	98.44	95.43
YOLOv3-ASFF* D-53 @608	90.16	97.28	97.57	94.15
YOLOv3-ASFF* D-53 @416	85.44	96.75	97.28	91.87
YOLOv3-ASFF D-53 @416	85.96	97.07	97.55	92.27
YOLOv3-ASFF* D-53 @320	80.03	96.90	97.85	88.96
RFBNet VGG-16 @300	44.47	86.29	95.62	67.79

Tabelle 8.1: Gegenüberstellung der Ergebnisse der k-Fold Cross Validation.

Evaluation – HTC

Aus einer Gegenüberstellung der AP@0.5-Metrik für die einzelnen Objektklassen (siehe Tabelle E.1) geht eine ausgewogene Detektionsgenauigkeit der Architektur *HTC X-101-FD @1333x800* hervor. Ähnliches wird für alle anderen Metriken aus $M_{benchmark}$ beobachtet. Trotz einer verringerten Abdeckung von Ground-Truth Daten für die Objektklasse *Gabelstapler* weist die AP@0.5-Metrik dieser Objektklasse den größten Wert auf. Ein Vergleich der AP_s^* , AP_m^* und AP_l^* -Metrik zeigt ähnlich hohe Werte für AP_m^* und AP_l^* -Metrik, betrachtet für alle Objektklassen. Die AP_s^* -Metrik weicht im Schnitt um etwa 3.6 Prozentpunkte von der AP_m^* und AP_l^* -Metrik ab. Damit weist die *HTC X-101-FD @1333x800*-Architektur eine allgemein geringere Detektionsgenauigkeit für kleine PF auf.

In Kapitel 3.5 wird die Berechnung der AP-Metriken geschildert. Diese beruht auf der Ermittlung der Fläche unter der PR-RC-Kurve. Die RC-Metriken der Klassen *Person*, *Pallette* und *Gabelstapler* sind im Schnitt um etwa 1 – 2 Prozentpunkte höher als die PR-Metriken, sodass die entsprechenden AP-Metriken der Klassen auf Grund des Berechnungsdefinition über die PR-Metriken beschränkt sind. Für die Klasse *Hubwagen* ist keine nennenswerte Abweichung zwischen der PR- und RC-Metrik gegeben. In Kapitel 8.1.1 wird die Relevanz der PR- und RC-Metriken im Kontext der Parameterkonfiguration eines Object Tracking-Algorithmus diskutiert. Die PR-Metriken der Objektklassen weisen für *HTC X-101-FD @1333x800* Werte nah an 100% auf. Betrachtet für den Einsatz als Kopplung an einen Object Tracking-Algorithmus kann damit für dessen Parameterkonfiguration mit hoher Wahrscheinlichkeit davon ausgegangen werden, dass es sich um TP-Detektionen handelt.

Eine Betrachtung der Metriken für die Architektur mit halbierter Auflösung (*HTC X-101-FD @666x400*) liefert folgende Erkenntnisse: Die Werte der AP_m^* und AP_l^* sind in etwa gleich ausgeprägt. Für die AP_s^* -Metrik sind jedoch verhältnismäßig starke Abweichungen von bis zu 12 Prozentpunkten für die Klassen *Person*, *Palette* und *Hubwagen* zu beobachten. Insgesamt ist zu erkennen, dass sich die Verringerung der AP@0.5-Metrik um 4.26 Prozentpunkte maßgeblich auf die Verringerung der RC-Metrik zurückführen lässt, begründet analog zu der zuvor getroffenen Schilderung (Berechnungsdefinition der AP-Metrik).

Evaluation – ATSS

Für die Architektur *ATSS X-101-FD @1333x800* ergibt sich für die Objektklassen *Person*, *Palette* und *Gabelstapler* eine ähnlich hohe AP@0.5-Metrik von durchschnittlich 94.7% (siehe Tabelle E.3). Für die Objektklasse *Hubwagen* ist, im Vergleich zu den anderen Objektklassen, eine Verringerung von durchschnittlich etwa 5 Prozentpunkten zu beobachten. Insgesamt ist die AP@0.5-Metrik um etwa 4 Prozentpunkte geringer gegenüber der *HTC X-101-FD @1333x800*-Architektur. Dies beruht maßgeblich auf einer geringen RC-Metrik. Aus den Verteilungen in Tabelle 4.1 geht eine hohe Anzahl von kleinen PF (Größe S) für

die Klasse *Hubwagen* hervor. Aus den aufgeführten Aspekten in Kapitel 7.1.1 geht ein allgemeiner Trend für eine geringe Detektionsgenauigkeit bei kleinen PF hervor. Dieser fällt aufgrund der hohen Anzahl von kleinen PF für die Klasse *Hubwagen* besonders ins Gewicht. Insgesamt wird in Bezug auf die Detektionsgenauigkeit für die verschiedenen PF, analog zu *HTC X-101-FD @1333x800*, eine Diskrepanz zwischen der AP_s^* -Metrik verglichen mit den Metriken AP_m^* und AP_l^* beobachtet. Dieser Effekt ist verstärkt für die Objektklasse *Gabelstapler* zu erkennen. Aufgrund von größeren Anzahlen mittelgroßer und großer PF (Größe M und L) hat die geringe Ausprägung für die AP_s^* -Metrik jedoch keinen großen Effekt auf die Ausprägung der AP@0.5-Metrik. Im Gegensatz zu *HTC X-101-FD @1333x800* lassen sich für alle Objektklassen niedrige Werte der AP-Metriken auf eine niedrige RC-Metrik zurückführen.

Aus Tabelle E.4 geht hervor, dass die zuvor geschilderten Aspekte über die jeweiligen Ausprägungen der AP-Metriken bei der Anwendung einer halbierten Auflösung verstärkt werden. Insbesondere für AP_s^* -Metrik der Klasse *Hubwagen* ist eine Verringerung von 30 Prozentpunkten zu beobachten. Dabei sind alle PR-Metriken in etwa gleich ausgeprägt. Damit ist erneut eine verringerte Detektionsgenauigkeit allein auf niedrige Ausprägungen der RC-Metrik zurück zu führen. Ausgenommen hiervon ist die Detektionsgenauigkeit für die Klasse *Gabelstapler*. Hier weist die AP@0.5-Metrik sogar einen höheren Wert auf. Damit hat eine halbierte Auflösung keinen negativen Effekt auf die Detektionsgenauigkeit von Objekten dieser Klasse.

Für die Architektur *ATSS R-101-FD @1333x800* und *ATSS R-101-FD @666x400* weisen alle Metriken nahezu identische Werte auf. Daher werden für diese Architekturen keine weiteren Betrachtungen aufgeführt. Des Weiteren wird dadurch gezeigt, dass für den ATSS Object Detection-Algorithmus keine nennenswerte Verbesserung der Detektionsgenauigkeit unter Verwendung der komplexeren Backbone-Architektur ResNeXt erzielt wird. Eine Reduzierung der Auflösung um 50% resultiert in einer Verringerung der AP@0.5-Metrik um etwa 10 Prozentpunkte für beide Architekturen des ATSS Object Detection-Algorithmus.

Evaluation – YOLO-v3 ASFF

Für die Architektur *YOLOv3-ASFF* D-53 @800* ist über alle Objektklassen hinweg eine ausgewogene AP@0.5-Metrik zu beobachten. Auch hier wird der Trend bestätigt, dass die Detektionsgenauigkeit für Instanzen mit einer kleinen PF geringer ist gegenüber der Detektionsgenauigkeit für mittelgroße und große PF. Der Unterschied der AP_s^* -Metrik im Vergleich zu den Metriken AP_m^* und AP_l^* beläuft sich auf 4 - 8 Prozentpunkte. Ähnlich zu den Architekturen des ATSS Object Detection-Algorithmus ist die RC-Metrik der ausschlaggebende Faktor für eine Verringerung der Werte für die AP-Metriken.

Für die Architekturen *YOLOv3-ASFF* D-53 @608*, *YOLOv3-ASFF* D-53 @416*, *YOLOv3-ASFF D-53 @416* und *YOLOv3-ASFF* D-53 @320*, sind große Veränderungen ge-

genüber den Metriken von *YOLOv3-ASFF* D-53 @800* lediglich für die AP_s^* -Metriken zu beobachten. Damit ergibt sich eine reduzierte Detektionsgenauigkeit für Objektinstanzen mit einer kleinen PF. Diese ist auf die jeweils verwendeten Auflösungen der Architekturen zurückzuführen. Mit zunehmend geringerer Auflösung werden zunehmend geringere AP_s^* -Metriken beobachtet. Die AP_m^* und AP_l^* sind für alle Architeturen des YOLO-v3 ASFF Object Detection-Algorithmus ähnlich ausgeprägt. Die geringere Detektionsgenauigkeit für kleine PF hat einen negativen Effekt auf die AP@0.5Metrik. Insgesamt ergibt sich aus der Anwendung von verschiedenen Auflösungen eine Diskrepanz von 6 Prozentpunkten bezogen auf die AP@0.5-Metrik von *YOLOv3-ASFF* D-53 @800* mit 95.43% und 88.96% für die *YOLOv3-ASFF* D-53 @320*-Architektur.

Aus Gründen der Übersichtlichkeit wird für den nachfolgenden Abschnitt die Bezeichnung *A* für die Architektur *YOLOv3-ASFF* D-53 @416* und *B* für die Architektur *YOLOv3-ASFF D-53 @416* verwendet. Ein Vergleich der Metriken aus Tabelle 8.1 zeigt einen höheren Wert der AP@0.5-Metrik für *B* als für *A*. Den Ergebnissen des MS COCO-Datensatz zufolge müsste die AP@0.5-Metrik jedoch umgekehrt ausgeprägt sein. Bei einer Betrachtung der Hyperparameteroptimierung auf dem Split S1 ist für *A* wiederum eine höhere AP@0.5 zu beobachten als für *B*. Damit ist zu vermuten, dass die Detektionsgenauigkeit von YOLO-v3 ASFF für die verwendete Auflösung auf einen Wert von etwa 92% für AP@0.5-Metrik konvergiert. Eine Verwendung des RFB- und Drop-Block Moduls in *B* (gekennzeichnet mit *) bringt somit keinen Profit. Um allgemeine Schlüsse zu ziehen, ist der Effekt für alle verwendeten Auflösungen des YOLO-v3 ASFF Object Detection-Algorithmus zu analysieren. Dies ist jedoch nicht Gegenstand der Arbeit.

Evaluation – RFBNet

Die AP@0.5-Metriken sind für die *RFBNet VGG-16 @300*-Architektur sehr unterschiedlich ausgeprägt. Es sind Diskrepanzen von 20 - 30 Prozentpunkten zu beobachten. Ähnlich zu den anderen Object Detection-Algorithmen weist die *RFBNet VGG-16 @300*-Architektur die geringste Detektionsgenauigkeit für die Objektklasse *Hubwagen* auf, mit einer AP@0.5-Metrik von 51.10%. Dies ist begründet durch die große Anzahl von Objektinstanzen mit kleiner PF. Für die Klassen *Person*, *Palette* und *Gabelstapler* ist eine ähnlich hohe PR-Metrik, im Vergleich zu den anderen Object Detection-Algorithmen, gegeben. Auch für die *RFBNet VGG-16 @300*-Architektur bilden niedrige Werte der RC-Metrik den ausschlaggebenden Faktor für eine Verringerung der Werte für die AP-Metriken.

8.3 Gesamtheitliche Betrachtung

Analog zum Ergebnis auf den Bilddaten des MS COCO-Datensatzes weist die *HTC X-101-FD @1333x800*-Architektur die höchsten Werte der AP@0.5-Metrik auf. Darauf folgen die

Architekturen *YOLOv3-ASFF* D-53 @800* und *YOLOv3-ASFF* D-53 @608*. Gesamt-heitlich betrachtet ergibt sich für diese eine höhere AP@0.5-Metrik, gegenüber der Archi-tekturen *ATSS X-101-FD @1333x800* und *ATSS R-101-FD @1333x800*, was dem umgekehrten Fall der Ergebnisse auf dem MS COCO-Datensatz entspricht. Nachfolgend wird näher auf diesen Effekt eingegangen. Im MS COCO-Datensatz gilt es, 80 Objektklassen zu detektieren, im vorliegenden Bilddatensatz, der Szenarien aus dem Kontext der Intra-logistik abbildet, lediglich 4. Zudem ist der MS COCO-Datensatz auf Basis einer großen Menge von unterschiedlichen Domänen aufgebaut worden. Neben der großen Variabilität der Objektinstanzen, ist zudem eine großes Spektrum an Hintergründen gegeben. In den Bilddaten der Intralogistik ist dieses Spektrum stark reduziert. Dies führt zu einer Ver-ringerung der Detektionsperformance, die erbracht werden muss, um hohe AP-Metriken auf dem Datensatz *D2* zu erzielen. Die Object Detection-Algorithmen ATSS und YOLO-v3 ASFF nutzen beide, als Grundlage für Bestimmung der finalen Bounding-Boxen, das Konzept der Anchor-Boxen. Das Kernkonzept des ATSS Object Detection-Algorithmus ist auf eine Optimierung der Auswahl von positiven und negativen Trainingsbeispielen für das Training der Bounding-Box-Klassifikation ausgelegt. Die Grundlage dafür bilden Statisti-ken, welche auf Basis der Ground-Truth-Bounding-Boxen und selektierten Anchor-Boxen berechnet werden. Das Kernkonzept des YOLO-v3 ASFF Object Detection-Algorithmus besteht aus einem Lernprozess, der auf das Finden einer optimalen Fusion von Features ausgerichtet ist, welche aus unterschiedlichen Ebenen in der verwendeten Feature-Pyramide stammen. In Bezug auf die geschilderte Evaluierung für die ATSS-Architekturen wird das Kernproblem einer verringerten Detektionsgenauigkeit auf die Problematik der Detektion von Objektinstanzen zurückgeführt, die eine kleine PF ausweisen. Die optimale Fusion der Features in der Architektur des YOLO-v3 ASFF Object Detection-Algorithmus liefert, auf Basis der Bilddaten der Intralogistik, einen besseren Ansatz, um PF unterschiedlicher Größe zu detektieren. Insbesondere für Objektinstanzen mit kleiner PF ist eine erhöhte Detektionsgenauigkeit der zuvor aufgeführten Architekturen des YOLO-v3 ASFF Object Detection-Algorithmus gegenüber der des ATSS Object Detection-Algorithmus zu beob-achten. Aus einem Vergleich der Architekturen des YOLO-v3 ASFF und HTC Object Detection-Algorithmus geht eine Änderung der AP@0.5-Metrik von 2 - 9 Prozentpunkte hervor. Damit werden auf Basis der Cascaden-Struktur von HTC zwar eine höhere Metrik erzielt, der daraus entstehende Profit wird jedoch durch die stark gestiegerte Komplexität der Architektur gemildert. Insgesamt weist die Architektur des RFBNet die mit Abstand geringste AP@0.5-Metrik auf. Dies ist im Wesentlichen auf die Verwendung von VGG-19 als Backbone-Architektur zurückzuführen. Die geringe Anzahl der Layer resultiert in einer geringen Komplexität und limitiert damit die Detektionsgenauigkeit, die unter Anwendung des VGG-19 erbracht werden kann. Zusammenfassend lässt sich für alle Architekturen des Benchmarks festhalten, dass die AP-Metriken maßgeblich von der Ausprägung der RC-

-Metrik abhängig sind. Dieser Effekt wird für Objektinstanzen mit kleiner PF verstärkt. Eine Ausnahme bildet lediglich die *HTC X-101-FD @1333x800*-Architektur.

Die Abbildungen 8.1 und 8.2 stellen die FPS und die AP@0.5-Metrik aller im Benchmark verwendeten Architekturen gegenüber. Auf der Grundlage der so gegebenen Gegenüberstellung geht für jede Architektur der finale Speed-Accuracy Trade-Off hervor. Mit einer Reduzierung der Auflösung um 50% wird eine Reduzierung der Laufzeit um 50% für die *HTC X-101-FD @1333x800*-Architektur erreicht. Für die Architekturen ATSS X-101-FD @1333x800 und ATSS R-101-FD @666x400 ist eine Verringerung um etwa 67% zu beobachten. Dennoch bieten die Architekturen der ATSS und HTC Object Detection-Algorithmus gegenüber der des YOLO-v3 ASFF Object Detection-Algorithmus ein unausgewogenes Verhältnis aus Laufzeit und Detektionsgenauigkeit. Insgesamt werden die Architekturen des YOLO-v3 ASFF Object Detection-Algorithmen als Architekturen mit dem geringsten Speed-Accuracy Trade-Off festgehalten.

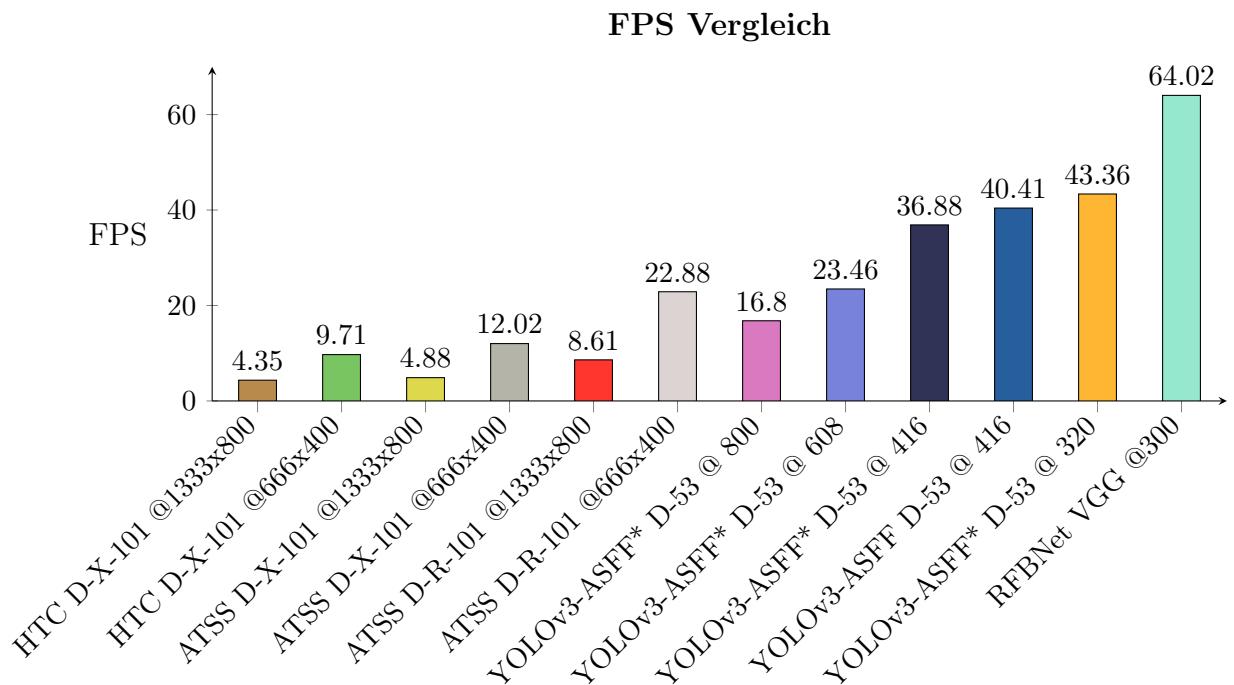


Abbildung 8.1: FPS Vergleich aller Architekturen.

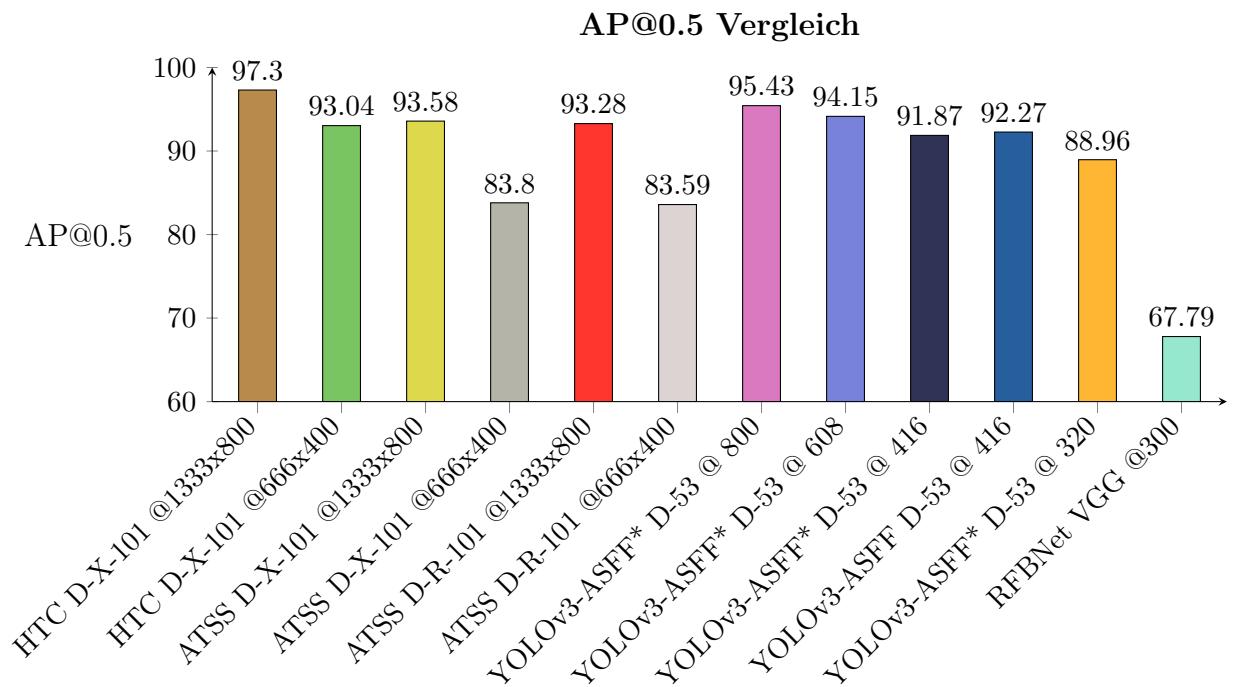


Abbildung 8.2: AP@0.5-Metrik Vergleich aller Architekturen

Kapitel 9

Schlussbetrachtung

In diesem Kapitel erfolgt zunächst eine Zusammenfassung aller in der Arbeit ermittelten Ergebnisse. Abschließend wird in einem Ausblick auf die weiter notwendigen Schritte eingegangen, um eine Integration der vorgestellten Algorithmen in einen Produktivbetrieb zu realisieren.

9.1 Zusammenfassung

Eine automatisierte Ermittlung von Bewegungsprofilen trägt zur Steigerung der Transparenz von Lagerprozessen im Kontext der Intralogistik bei. Zur Lösung der Problemstellung wird in der Arbeit das Potential von bilddatengestützten Object Tracking-Algorithmen aufgezeigt. Als Grundlage dafür werden Deep Learning-basierte Object Detection-Algorithmen herangezogen. Gegenstand der Arbeit ist die Bereitstellung einer fundierten Entscheidungsbasis, die eine Bewertung des Optimierungskonflikts zulässt, dem Object Detection-Algorithmen unterliegen. Dieser wird unter dem Synonym Speed-Accuracy Trade-Off verwendet. Die zur Bewertung notwendigen Metriken werden im Rahmen eines Benchmarks erarbeitet, das unter dem Kontext der Anwendung auf Bilddaten der Intralogistik steht. Aus der Grundgesamtheit aller (zum Zeitpunkt der Arbeit) zur Verfügung stehenden Deep Learning-basierten Object Detection-Algorithmen wurden dafür Verfahren selektiert, die den Speed-Accuracy Trade-Off bestmöglich abdecken. Für die Modellierung der ausgewählten Verfahren wird für den Aufbau eines Datensatzes, explizit auf eine große Abdeckung aller in den Lagerprozessen bestehenden Szenarien geachtet. Die Anwendung einer k-Fold Cross Validation garantiert die Güte der Aussagen, die auf Grundlage der erzielten Metriken getroffen werden. Insgesamt wird anhand der geschilderten Aspekte eine Abbildung des aktuellen Stands der Forschung, in Bezug auf die ermittelten Ergebnisse sicher gestellt, betrachtet im Kontext einer automatisierten Detektion von Objekten in Lagerprozessen der Intralogistik.

Anhand einer hohen Detektionsgenauigkeit und gleichzeitig niedriger Laufzeit wird für den Object Detection-Algorithmus YOLO-v3 ASFF der geringste Speed-Accuracy Trade-Off beobachtet. Zudem wird die Detektion von Objektinstanzen, die eine kleine Pixelfläche aufweisen, als Limitierung der Detektionsgenauigkeit herausgestellt. Insgesamt ist erfolgreich eine Entscheidungsbasis ermittelt worden, die als Anhaltspunkt für die Wahl eines für Object Tracking geeignetes Verfahrens zur Object Detection hervorgeht.

9.2 Ausblick

Auf Grundlage des aufgezeigten Speed-Accuracy Trade-Off geht die Wahl von YOLO-v3 ASFF zur Integration in die bestehende Lösung zum Object Tracking im Rahmenprojekt bei Fraunhofer IML hervor. Die aufgeführten Metriken geben zudem einen Anhaltspunkt, wie die Parameter des Object Tracking-Algorithmus zu konfigurieren sind. Für eine finale Lösung gilt es zunächst, eine der in Kapitel 8.1.2 geschilderten Varianten für eine parallele Inferenz zu implementieren. Im Weiteren gilt es zu untersuchen, mit welchen Mitteln eine Vergrößerung der durchschnittlichen PF für die zu detektierenden Objektinstanzen realisiert werden kann. Auf Basis der aufgeführten Analyse in Kapitel 8.2 geht eine allgemein hohe Detektionsgenauigkeit für größere Objektinstanzen hervor. Damit bietet die Reduzierung der Menge an Objektinstanzen mit kleiner PF die Chance, die allgemeine Detektionsgenauigkeit zu steigern. Erreicht werden kann dies durch eine Abdeckung der Lagerhalle mit einer vergrößerten Anzahl von Kameras. Die Verwendung einer zentralen Workstation sorgt in diesem Kontext für eine Limitierung, da nur eine begrenzte Anzahl von Kameras angeschlossen werden können. Daher gilt es, das Potential eines dezentralen Ansatzes zu untersuchen. Dieser bietet die Chance, eine größere Anzahl von Kameras in der Lagerhalle zu installieren, da die Algorithmen direkt auf dem Gerät ausgeführt werden. Ergibt sich durch den Einsatz von mehreren Kameras eine Änderung in der Verteilung der PF, können insbesondere die weniger komplexen, dafür mit geringerer Laufzeit verbundenen Algorithmen profitieren. In diesem Kontext ist zudem auf bestehende Optimierungsansätze hinzuweisen, die gezielt darauf ausgerichtet sind, eine Ausführung von Deep Learning-Algorithmen auf ressourcenbeschränkten Systemen zu realisieren. Ansätze dazu werden unter anderem in [146, 135, 106] vorgestellt.

Die Ergebnisse der Arbeit basieren auf dem Stand einer Recherche von Februar 2020. Während der Arbeit gab es unter anderem Weiterentwicklungen für den YOLO Object Detection-Algorithmus [9, 145]. Zudem steht seit April 2020 die offizielle Version des EfficientDet Object Detection-Algorithmus zur Verfügung [40, 133]. Allgemein untersteht das Forschungsfeld Deep Learning einer schnellen Weiterwicklung. Daher bietet eine fortwährende Betrachtung von neuen Errungenschaften das Potential einer stetigen Optimierung der bestehenden Lösung. Die Standardschnittstellen der Deep Learning-Frameworks bieten die Möglichkeit einer einfachen Integration in die Gesamtsoftwarearchitektur, sodass

der Mehrwert neuer Entwicklungen im Bereich der Object Detection-Algorithmen schnell abgeschätzt werden kann.

Anhang A

Algorithmen

Algorithmus 3: NMS Algorithmus. Bearbeitete Version aus [10].

Eingabe:

$B = \{b_1, \dots, b_N\}$, Liste der initialen Bounding-Boxen

$S = \{s_1, \dots, s_N\}$, Liste der zu B gehörigen Klassen-Scores (-Wahrscheinlichkeiten)

T_{nms} , gegebener IOU-Grenzwert

Ausgabe:

D , Liste der gefilterten Bounding-Boxen

S^* , Liste der zu D gehörigen Klassenwahrscheinlichkeiten

```
1  $D \leftarrow \{\}$ 
2 while  $B \neq \emptyset$  do
3    $m \leftarrow \text{argmax } S$ 
4    $D \leftarrow D \cup \{b_m\}$ 
5    $B \leftarrow B - \{b_m\}$ 
6   for  $i \in \{1, \dots, |B|\}$  do
7     if  $\text{IOU}(b_m, b_i) \geq T_{nms}$  then
8        $B \leftarrow B - b_i$ 
9        $S \leftarrow S - s_i$ 
10      end
11    end
12 end
13 return  $D, S^*$ 
```

Anhang B

Tabelle der Grundgesamtheit an Object Detection-Algorithmen

ANHANG B. TABELLE DER GRUNDGESAMTHEIT AN OBJECT DETECTION-ALGORITHMEN

Tabelle B.1: Grundgesamtheit der Object Detection.

Meta-Architektur	Backbone-Architektur	AP	AP@0.5	AP@0.75	AP _S	AP _M	AP _L	FPS	MS-T	MT-I	GPU	Referenz
ATSS	R-101-FPN-2x	43.6	-	-	-	-	-	17.54	ja	nein	V100	[157]
ATSS	R-50-FPN-1x	39.3	-	-	-	-	-	22.73	ja	nein	V100	[157]
ATSS	X-101-32x4d-FPN-2x	45.1	-	-	-	-	-	9.09	ja	nein	V100	[157]
ATSS	X-101-64x4d-FPN-2x	45.6	-	-	-	-	-	8.93	ja	nein	V100	[157]
ATSS	dconv2-R-101-FPN-2x	46.3	-	-	-	-	-	13.7	ja	nein	V100	[157]
ATSS	dconv2-R-50-FPN-1x	43	-	-	-	-	-	18.52	ja	nein	V100	[157]
ATSS	dconv2-X-101-32x8d-FPN-2x	47.7	-	-	-	-	-	6.99	ja	nein	V100	[157]
ATSS	dconv2-X-101-32x8d-FPN-2x	50.6	-	-	-	-	-	-	ja	ja	-	[157]
ATSS	dconv2-X-101-64x4d-FPN-2x	47.7	-	-	-	-	-	6.94	ja	nein	V100	[157]
ATSS	dconv2-X-101-64x4d-FPN-2x	50.7	-	-	-	-	-	-	ja	ja	-	[157]
Cascade Mask R-CNN	HRNetV2p-W18	41.9	-	-	-	-	-	-	nein	nein	-	[16, 131, 92]
Cascade Mask R-CNN	HRNetV2p-W32	44.5	-	-	-	-	-	-	nein	nein	-	[16, 131, 92]
Cascade Mask R-CNN	HRNetV2p-W48	46	-	-	-	-	-	-	nein	nein	-	[16, 131, 92]
Cascade Mask R-CNN	R-101 FPN	43.3	-	-	-	-	-	6.8	nein	nein	V100	[16, 92]
Cascade Mask R-CNN	R-50 FPN	41.2	-	-	-	-	-	7.4	nein	nein	V100	[16, 92]
Cascade Mask R-CNN	R-50 FPN	42.3	-	-	-	-	-	7.4	nein	nein	V100	[16, 92]
Cascade Mask R-CNN	ResNeXt152	48.3	67	52.8	-	-	-	-	nein	nein	-	[16, 75]
Cascade Mask R-CNN	ResNeXt152 Dual	50	68.8	54.6	-	-	-	-	nein	nein	-	[16, 75]
Cascade Mask R-CNN	ResNeXt152 Dual	52.8	70.6	58	-	-	-	-	nein	ja	-	[16, 75]
Cascade Mask R-CNN	X-101 FPN - DCN	47.1	-	-	-	-	-	-	nein	nein	-	[16, 17, 92]
Cascade Mask R-CNN	X-101 FPN DCN GC(c3-c5-116)	47.9	-	-	-	-	-	-	nein	nein	-	[16, 17, 92]
Cascade Mask R-CNN	X-101 FPN DCN GC(c3-c5-4)	47.9	-	-	-	-	-	-	nein	nein	-	[16, 17, 92]
Cascade Mask R-CNN	X-101 FPN GC(c3-c5 r16)	45.9	-	-	-	-	-	-	nein	nein	-	[16, 17, 92]
Cascade Mask R-CNN	X-101 FPN GC(c3-c5 r4)	46.5	-	-	-	-	-	-	nein	nein	-	[16, 17, 92]
Cascade Mask R-CNN	X-101-32x4d FPN	44.4	-	-	-	-	-	6.6	nein	nein	V100	[16, 92]
Cascade Mask R-CNN	X-101-32x4d FPN	44.7	-	-	-	-	-	6.6	nein	nein	V100	[16, 92]
Cascade Mask R-CNN	X-101-64x4d FPN	45.4	-	-	-	-	-	5.3	nein	nein	V100	[16, 92]
Cascade Mask R-CNN	X-101-64x4d FPN	45.7	-	-	-	-	-	5.3	nein	nein	V100	[16, 92]
Cascade R-CNN	HRNetV2p-W18	41.2	-	-	-	-	-	-	nein	nein	-	[16, 131, 92]
Cascade R-CNN	HRNetV2p-W32	43.7	62	47.4	25.5	46	55.3	-	nein	nein	-	[16, 131, 92]
Cascade R-CNN	HRNetV2p-W48	44.6	-	-	-	-	-	-	nein	nein	-	[16, 131, 92]
Cascade R-CNN	R-101 FPN	42	-	-	-	-	-	10.3	nein	nein	V100	[16, 92]
Cascade R-CNN	R-101 FPN	43.1	61.7	46.7	24.1	45.9	55	-	nein	nein	-	[16, 131, 92]
Cascade R-CNN	R-101 FPN+	42.8	62.1	46.3	23.7	45.5	55.2	8.93	nein	nein	Titan Xp	[16]
Cascade R-CNN	R-50 FPN	40.4	-	-	-	-	-	11.9	nein	nein	V100	[16, 92]

Meta-Architektur	Backbone-Architektur	AP	AP@0.5	AP@0.75	AP _S	AP _M	AP _L	FPS	MS-T	MT-I	GPU	Referenz
Cascade R-CNN	R-50 FPN	41.1	-	-	-	-	-	-	nein	nein	-	[16, 92]
Cascade R-CNN	R-50 FPN+	40.6	59.9	44	22.6	42.7	52.1	10.87	nein	nein	Titan Xp	[16]
Cascade R-CNN	ResNet101	42.8	62.1	46.3	-	-	-	-	nein	nein	-	[16, 75]
Cascade R-CNN	ResNet101 Dual	44.3	62.5	48.1	-	-	-	-	nein	nein	-	[16, 75]
Cascade R-CNN	ResNet101 Triple	44.9	63.9	48.9	-	-	-	-	nein	nein	-	[16, 75]
Cascade R-CNN	X-101-32x4d FPN	43.6	-	-	-	-	-	8.9	nein	nein	V100	[16, 92]
Cascade R-CNN	X-101-32x4d FPN	44	-	-	-	-	-	-	nein	nein	-	[16, 92]
Cascade R-CNN	X-101-64x4d FPN	44.5	-	-	-	-	-	-	6.7	nein	V100	[16, 92]
Cascade R-CNN	X-101-64x4d FPN	44.7	-	-	-	-	-	-	nein	nein	-	[16, 92]
Cascade R-FCN	R-101	33.3	52.6	35.2	12.1	36.2	49.3	18.25	nein	nein	Titan X	[16, 29]
Cascade R-FCN	R-50	30.9	49.9	32.6	10.5	33.1	46.9	20.2	nein	nein	Titan X	[16, 29]
CenterMask	R-101-FPN	43.1	-	-	25.2	46.1	54.4	13.89	nein	nein	V100	[64]
CenterMask	V-99-FPN	45.8	-	-	27.8	48.3	57.6	11.9	nein	nein	V100	[64]
CenterMask	X-101-FPN	44.6	-	-	27.1	47.2	55.2	8.13	nein	nein	V100	[64]
CenterMask*	R-101-FPN	44	-	-	25.8	46.8	54.9	15.15	nein	nein	V100	[64]
CenterMask*	V-99-FPN	46.5	-	-	28.7	48.9	57.2	12.99	nein	nein	V100	[64]
CenterMask-Lite	M-v2-FPN	30.2	-	-	14.2	31.9	40.9	62.5	nein	nein	Titan Xp	[64]
CenterMask-Lite	R-50-FPN	36.7	-	-	18.7	39.4	48.2	43.1	nein	nein	V100	[64]
CenterMask-Lite	V-19-FPN	35.9	-	-	19.6	38	45.9	54.35	nein	nein	Titan Xp	[64]
CenterMask-Lite	V-39-FPN	40.7	-	-	22.4	43.2	53.5	44.64	nein	nein	Titan Xp	[64]
CenterNet	HRNetV2-W48	43.5	62.1	46.5	22.2	46.5	57.8	-	nein	nein	-	[26, 131]
CenterNet511	Houglas-s-104	44.9	62.4	48.1	25.6	47.4	57.4	2.94	nein	nein	P100	[26]
CenterNet511	Houglas-s-104	47	64.5	50.7	28.9	49.9	58.9	2.94	nein	ja	P100	[26]
CenterNet511	Houglas-s-52	41.6	59.4	44.2	22.5	43.1	54.1	3.7	nein	nein	P100	[26]
CenterNet511	Houglas-s-52	43.5	61.3	46.7	25.3	45.3	55	3.7	nein	ja	P100	[26]
CornerNet511	Houglas-s-104	40.6	56.4	43.2	19.1	42.8	54.3	-	nein	nein	-	[61]
CornerNet511 ms	Houglas-s-104	42.2	57.8	45.2	20.7	44.8	56.6	-	nein	ja	-	[61]
CornerNet512	Houglas-s-104	40.5	57.8	45.3	20.8	44.8	56.7	6.67	nein	ja	Titan X	[61]
CoupleNet	R-101	33.1	53.5	35.4	11.6	36.3	50.1	12.42	nein	nein	Titan X	[165]
CoupleNet	R-101	34.4	54.8	37.2	13.4	38.1	52	12.42	ja	nein	Titan X	[165]
D-RFCN + SNIP	R-101	43.4	65.6	48.4	27.2	46.5	54.9	1.68	nein	nein	Titan X	[125]
DSSD321	R-101	28	46.1	29.2	7.4	28.1	47.6	17.88	nein	nein	Titan X	[32]
DSSD513	R-101	33.2	53.3	35.2	13	35.4	51.1	-	nein	nein	-	[74, 151]
DSSD513	R-101	33.2	53.3	35.2	13	35.4	51.1	9.7	nein	nein	Titan X	[32]
DeNet-101	R-101	31.9	50.5	34.2	9.7	34.9	50.6	51.52	nein	nein	Titan X	[143]
DeNet-101 (skip)	R-101	32.3	51.4	34.6	10.5	35.1	50.9	50	nein	nein	Titan X	[143]
DeNet-101 (wide)	R-101	33.8	53.4	36.1	12.3	36.1	50.8	25.76	nein	nein	Titan X	[143]
DeNet-101 + Fitness-NMS	R-101	32.3	47.9	34.5	8.7	33.7	54.5	56.06	nein	nein	Titan X	[144, 143]
DeNet-101 + Fitness-NMS	R-101	32.9	52.5	35	22	39.9	34.4	4.55	nein	nein	Titan X	[144, 143]
DeNet-101 + Fitness-NMS	R-101	36.3	56.2	39	22.1	42.1	42.3	6.06	nein	nein	Titan X	[144, 143]
DeNet-101 + Fitness-NMS	R-101	36.9	53.9	39.5	13.8	39.3	56.5	36.36	nein	nein	Titan X	[144, 143]
DeNet-101 + Fitness-NMS	R-101	38.8	58.3	41.8	21.3	43.4	49.4	9.09	nein	nein	Titan X	[144, 143]

ANHANG B. TABELLE DER GRUNDGESAMTHEIT AN OBJECT DETECTION-ALGORITHMEN

Meta-Architektur	Backbone-Architektur	AP	AP@0.5	AP@0.75	AP _S	AP _M	AP _L	FPS	MS-T	MT-I	GPU	Referenz
DeNet-101 + Fitness-NMS	R-101	39.5	58	42.6	18.9	43.5	54.1	16.67	nein	nein	Titan X	[144, 143]
DeNet-101 + Fitness-NMS	R-101	41.8	60.9	44.9	21.5	45	57.5	7.58	nein	nein	Titan X	[144, 143]
DeNet-34	R-34	29.4	46.2	31.2	7.8	30.8	47.4	125.76	nein	nein	Titan X	[143]
DeNet-34 (skip)	R-34	29.5	47.9	31.1	8.8	30.9	47	124.24	nein	nein	Titan X	[143]
DeNet-34 (wide)	R-34	30	48.9	31.8	10.1	30.9	45.7	66.67	nein	nein	Titan X	[143]
EfficientDet-D0	EfficientNet-B0	32.4	-	-	-	-	-	62.5	nein	nein	Titan V	[133]
EfficientDet-D1	EfficientNet-B1	38.3	-	-	-	-	-	50	nein	nein	Titan V	[133]
EfficientDet-D2	EfficientNet-B2	41.1	-	-	-	-	-	41.67	nein	nein	Titan V	[133]
EfficientDet-D3	EfficientNet-B3	44.3	-	-	-	-	-	23.81	nein	nein	Titan V	[133]
EfficientDet-D4	EfficientNet-B4	46.6	-	-	-	-	-	13.51	nein	nein	Titan V	[133]
EfficientDet-D5 + AA	EfficientNet-B5	49.8	-	-	-	-	-	7.09	nein	nein	Titan V	[133]
EfficientDet-D6 + AA	EfficientNet-B6	50.6	-	-	-	-	-	5.26	nein	nein	Titan V	[133]
EfficientDet-D7 + AA	EfficientNet-B7	51	-	-	-	-	-	3.82	nein	nein	Titan V	[133]
ExtremeNet	Hourglass-104	40.2	55.5	43.2	20.4	43.2	53.1	-	nein	nein	-	[162]
ExtremeNet	Hourglass-104	43.7	60.5	47	24.1	46.9	57.6	-	nein	ja	-	[162]
FCOS	HRNet-W18-51-2x	37.7	-	-	-	-	-	13.89	nein	nein	V100	[138, 51, 131]
FCOS	HRNet-W18-51-2x	39.4	-	-	-	-	-	13.89	nein	nein	V100	[138, 51, 131]
FCOS	HRNet-W18-61-2x	37.8	-	-	-	-	-	9.43	nein	nein	V100	[138, 51, 131]
FCOS	HRNet-W32-51-2x	41.9	-	-	-	-	-	11.49	nein	nein	V100	[138, 51, 131]
FCOS	HRNet-W32-51-2x	42.5	-	-	-	-	-	11.49	nein	nein	V100	[138, 51, 131]
FCOS	HRNet-W32-61-2x	42.1	-	-	-	-	-	8	nein	nein	V100	[138, 51, 131]
FCOS	HRNet-W32-61-2x	42.9	-	-	-	-	-	8	nein	nein	V100	[138, 51, 131]
FCOS	HRNet-W40-61-3x	42.6	-	-	-	-	-	7.04	nein	nein	V100	[138, 51, 131]
FCOS	HRNetV2p-W18	37.8	56.1	40.4	21.6	39.8	47.4	-	nein	nein	-	[138, 51, 131]
FCOS	HRNetV2p-W32	40.5	59.3	43.3	23.4	42.6	51	-	nein	nein	-	[138, 51, 131]
FCOS	R-101 FPN	39.2	58.8	41.6	21.8	41.7	50	-	nein	nein	-	[138, 51, 131]
FCOS	R-101 FPN	41.5	60.7	45	24.4	44.8	51.6	16.95	nein	nein	V100	[138, 161]
FCOS	R-101-FPN-2x	41.5	-	-	-	-	-	16.95	nein	nein	V100	[138, 161]
FCOS	R-50 FPN	37.3	56.4	39.7	20.4	39.6	47.5	-	nein	nein	-	[138, 51, 131]
FCOS	R-50-FPN-1x	37.4	-	-	-	-	-	22.22	nein	nein	V100	[138, 161]
FCOS	ResNeXt-32x8d-101 FPN	42.7	62.2	46.1	26	45.6	52.6	9.09	nein	nein	V100	[138, 161]
FCOS	ResNeXt-64x4d-101 FPN	43.2	62.8	46.6	26.5	46.2	53.3	8.85	nein	nein	V100	[138, 161]
FCOS	ResNet-101-51-2x	41.4	-	-	-	-	-	13.51	nein	nein	V100	[138, 51, 131]
FCOS	ResNet-101-61-2x	41.5	-	-	-	-	-	8.26	nein	nein	V100	[138, 51, 131]
FCOS	ResNet-50-51-2x	37.1	-	-	-	-	-	14.08	nein	nein	V100	[138, 51, 131]
FCOS	ResNet-50-61-2x	37.1	-	-	-	-	-	10.2	nein	nein	V100	[138, 51, 131]
FCOS	imprv-R101-FPN-2x	43	-	-	-	-	-	17.54	nein	nein	V100	[138, 161]
FCOS	imprv-R50-FPN-1x	38.7	-	-	-	-	-	22.73	nein	nein	V100	[138, 161]
FCOS	imprv-X101-32x8d-FPN-2x	44	-	-	-	-	-	9.09	nein	nein	V100	[138, 161]

Meta-Architektur	Backbone-Architektur	AP	AP@0.5	AP@0.75	AP _S	AP _M	AP _L	FPS	MS-T	MT-I	GPU	Referenz	
FCOS	imprv-X-101-64x4d-FPN-2x	44.7	-	-	-	-	-	8.93	nein	nein	V100	[138, 161]	
FCOS	imprv-dcinv2-R-101-FPN-2x	45.6	-	-	-	-	-	13.7	nein	nein	V100	[138, 161]	
FCOS	imprv-dcinv2-R-50-FPN-1x	42.3	-	-	-	-	-	18.52	nein	nein	V100	[138, 161]	
FCOS	imprv-dcinv2-X-101-32x8d-FPN-2x	46.4	-	-	-	-	-	6.99	nein	nein	V100	[138, 161]	
FCOS	imprv-dcinv2-X-101-64x4d-FPN-2x	46.6	-	-	-	-	-	6.94	nein	nein	V100	[138, 161]	
FCOS w/ improvements	ResNeXt-64x4d-101 FPN	44.7	64.1	48.4	27.6	47.5	55.6	-	nein	nein	-	[138, 161]	
FSAF	R-101	40.9	61.5	44	24	44.2	51.3	-	nein	nein	-	[163]	
FSAF	R-101	42.8	63.1	46.5	27.8	45.5	53.2	-	nein	ja	-	[163]	
FSAF	X-101	42.9	63.8	46.3	26.6	46.2	52.7	-	nein	nein	-	[163]	
FSAF	X-101	44.6	65.2	48.6	29.7	47.1	54.6	-	nein	ja	-	[163]	
Faster R-CNN	Aligned-Inception-ResNet	30.8	49.6	-	9.6	32.5	49	-	nein	nein	-	[112, 24]	
Faster R-CNN	HRNetV2p-W18	36.1	-	-	-	-	-	-	nein	nein	-	[112, 131, 92]	
Faster R-CNN	HRNetV2p-W18	38.3	-	-	-	-	-	-	nein	nein	-	[112, 131, 92]	
Faster R-CNN	HRNetV2p-W32	39.5	-	-	-	-	-	-	nein	nein	-	[112, 131, 92]	
Faster R-CNN	HRNetV2p-W32	40.6	-	-	-	-	-	-	nein	nein	-	[112, 131, 92]	
Faster R-CNN	HRNetV2p-W32	41.1	62.3	44.9	24	43.1	51.4	-	nein	nein	-	[112, 131, 92]	
Faster R-CNN	HRNetV2p-W40	42.1	63.2	46.1	24.6	44.5	52.6	-	nein	nein	-	[112, 131, 92]	
Faster R-CNN	HRNetV2p-W48	40.9	-	-	-	-	-	-	nein	nein	-	[112, 131, 92]	
Faster R-CNN	HRNetV2p-W48	41.5	-	-	-	-	-	-	nein	nein	-	[112, 131, 92]	
Faster R-CNN	HRNetV2p-W48	42.4	63.6	46.4	24.9	44.6	53	-	nein	nein	-	[112, 131, 92]	
Faster R-CNN	R-101	29.4	48	-	9	30.5	47.1	-	nein	nein	-	[112, 24]	
Faster R-CNN	R-101 FPN	38.5	-	-	-	-	-	-	11.9	nein	-	[112]	
Faster R-CNN	R-101 FPN	38.8	-	-	-	-	-	-	11.5	nein	-	[112]	
Faster R-CNN	R-101 FPN	39.4	-	-	-	-	-	-	nein	nein	-	[112]	
Faster R-CNN	R-101 FPN	39.4	61.5	42.8	-	-	-	12.27	nein	nein	Titan X	[112, 75]	
Faster R-CNN	R-101 FPN	40.3	61.8	43.9	22.6	43.1	51	-	nein	nein	-	[112]	
Faster R-CNN	R-101 FPN	42	-	-	-	-	-	19.61	nein	nein	V100	[112]	
Faster R-CNN	R-101 FPN Dual	41	62.4	44.5	-	-	-	8.33	nein	nein	Titan X	[112, 75]	
Faster R-CNN	R-101 FPN Triple	41.7	64	45.5	-	-	-	-	nein	nein	-	[112, 75]	
Faster R-CNN	R-101-C4	41.1	-	-	-	-	-	7.19	nein	nein	V100	[112]	
Faster R-CNN	R-101-DC5	40.6	-	-	-	-	-	11.63	nein	nein	V100	[112, 29]	
Faster R-CNN	R-152 FPN	40.6	62.1	44.3	22.6	43.4	52	-	nein	nein	-	[112, 131]	
Faster R-CNN	R-50 FPN	36.4	-	-	-	-	-	13.6	nein	nein	V100	[112, 92]	
Faster R-CNN	R-50 FPN	36.6	-	-	-	-	-	13.5	nein	nein	V100	[112, 92]	
Faster R-CNN	R-50 FPN	37.7	-	-	-	-	-	-	nein	nein	-	[112, 92]	
Faster R-CNN	R-50 FPN	37.9	-	-	-	-	-	-	26.32	nein	nein	V100	[112, 29]
Faster R-CNN	R-50-C4	33.9	-	-	-	-	-	-	9.3	nein	nein	V100	[112, 92]
Faster R-CNN	R-50-C4	34.9	-	-	-	-	-	9.5	nein	nein	V100	[112, 92]	
Faster R-CNN	R-50-C4	35.7	-	-	-	-	-	9.8	nein	nein	V100	[112, 29]	

ANHANG B. TABELLE DER GRUNDGESAMTHEIT AN OBJECT DETECTION-ALGORITHMEN

Meta-Architektur	Backbone-Architektur	AP	AP@0.5	AP@0.75	AP _S	AP _M	AP _L	FPS	MS-T	MT-I	GPU	Referenz	
Faster R-CNN	R-50-C4	35.9	-	-	-	-	-	9.4	nein	V100	[112, 92]		
Faster R-CNN	R-50-C4	36.5	-	-	-	-	-	9.3	nein	nein	V100	[112, 92]	
Faster R-CNN	R-50-C4	38.4	-	-	-	-	-	9.62	nein	nein	V100	[112, 29]	
Faster R-CNN	R-50-DC5	37.3	-	-	-	-	-	14.71	nein	nein	V100	[112, 29]	
Faster R-CNN	R-50-DC5	39	-	-	-	-	-	14.29	nein	nein	V100	[112, 29]	
Faster R-CNN	X-101-32x4d FPN	40.1	-	-	-	-	-	10.3	nein	nein	V100	[96, 92]	
Faster R-CNN	X-101-32x4d FPN	40.4	-	-	-	-	-	-	nein	nein	-	[96, 92]	
Faster R-CNN	X-101-64 × 4d FPN	41.1	62.8	44.8	23.5	44.1	52.3	-	nein	nein	-	[112, 131]	
Faster R-CNN	X-101-64x4d FPN	40.7	-	-	-	-	-	-	nein	nein	-	[96, 92]	
Faster R-CNN	X-101-64x4d FPN	41.3	-	-	-	-	-	-	7.3	nein	nein	V100	[112, 92]
Faster R-CNN	X101 FP N	43	-	-	-	-	-	-	10.2	nein	nein	V100	[112, 29]
Faster R-CNN	R-101 FPN	37.8	-	-	-	-	-	-	19.57	nein	nein	P100	[112]
Faster R-CNN	R-50 FPN	40.2	-	-	-	-	-	-	26.32	nein	nein	V100	[112, 29]
Faster R-CNN + DCNv1	Aligned-Inception-ResNet	34.1	51.1	-	12.2	36.5	52.4	-	nein	nein	-	[112, 24]	
Faster R-CNN + DCNv2	R-101	44	65.9	48.1	23.2	47.7	59.6	-	nein	nein	-	[112, 164]	
Faster R-CNN + DCNv2	R-101	44.8	66.3	48.8	24.4	48.1	59.6	-	nein	nein	-	[112, 164]	
Faster R-CNN + DCNv2	R-101	44.8	66.3	48.8	24.4	48.1	59.6	-	nein	nein	-	[112, 164]	
Faster R-CNN + DCNv2	R-101	46	67.9	50.8	27.8	49.1	59.6	-	nein	nein	-	[112, 164]	
Faster R-CNN + S-NMS	R-101	25.5	46.6	-	8.8	27.9	38.5	-	nein	nein	-	[112]	
Faster R-CNN + S-NMS L G	R-101	25.5	46.7	-	8.8	27.9	38.3	-	nein	nein	-	[112]	
Faster R-CNN G-RM[44]	Inception-ResNet-v2	34.7	55.5	36.7	13.5	38.1	52	-	nein	nein	-	[112]	
Faster R-CNN w TDM[45]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	52.1	-	nein	nein	-	[112]	
Faster R-CNN + SNIP (RPN)	R-101	43.1	65.3	48.1	26.1	45.9	55.2	-	nein	nein	-	[112, 125]	
Faster-RCNN + SNIP (RPN+RCN)	R-101	44.4	66.2	49.9	27.3	47.4	56.9	-	nein	nein	-	[112, 125]	
FoveaBox	R-101	38.5	-	-	-	-	-	11.5	nein	nein	V100	[58]	
FoveaBox	R-101	38.5	-	-	-	-	-	11.5	nein	nein	V100	[58]	
FoveaBox	R-101	39.4	-	-	-	-	-	11.5	nein	nein	V100	[58]	
FoveaBox	R-101	41.9	-	-	-	-	-	11.5	nein	nein	V100	[58]	
FoveaBox	R-50	36.5	-	-	-	-	-	13.5	nein	nein	V100	[58]	
FoveaBox	R-50	36.9	-	-	-	-	-	13.5	nein	nein	V100	[58]	
FoveaBox	R-50	37.9	-	-	-	-	-	13.5	nein	nein	V100	[58]	
FoveaBox	R-50	40.1	-	-	-	-	-	13.5	nein	nein	V100	[58]	
Hybrid Task Cascade	HRNet V2p-W18	43.1	-	-	-	-	-	-	nein	nein	-	[20, 92]	
Hybrid Task Cascade	HRNet V2p-W32	45.3	-	-	-	-	-	-	nein	nein	-	[20, 92]	
Hybrid Task Cascade	HRNet V2p-W32	45.6	64.1	49.4	26.7	47.7	58	-	nein	nein	-	[20, 92]	
Hybrid Task Cascade	HRNet V2p-W48	46.8	-	-	-	-	-	-	nein	nein	-	[20, 92]	
Hybrid Task Cascade	HRNet V2p-W48	47.3	65.9	51.2	28	49.7	59.8	-	nein	nein	-	[20, 92]	
Hybrid Task Cascade	R-101 FPN	44.9	-	-	-	-	-	4	nein	nein	V100	[20, 92]	

Meta-Architektur	Backbone-Architektur	AP	AP@0.5	AP@0.75	AP _S	AP _M	AP _L	FPS	MS-T	MT-I	GPU	Referenz
Hybrid Task Cascade	R-101 FPN	45.1	64.3	49	25.2	48	58.2	4	nein	nein	V100	[20, 92]
Hybrid Task Cascade	R-101 FPN	45.3	39.7	43.1	21	42.2	53.5	3.64	nein	nein	Titan X	[20]
Hybrid Task Cascade	R-50 FPN	42.1	-	-	-	-	-	4.1	nein	nein	V100	[20, 92]
Hybrid Task Cascade	R-50 FPN	43.2	-	-	-	-	-	4.1	nein	nein	V100	[20, 92]
Hybrid Task Cascade	R-50 FPN	43.6	38.4	41.5	20.4	40.7	51.2	3.79	nein	nein	Titan X	[20]
Hybrid Task Cascade	X-101 FPN	47.1	41.2	44.7	22.8	43.9	54.6	3.18	nein	nein	Titan X	[20]
Hybrid Task Cascade	X-101-32x4d FPN	46.1	-	-	-	-	-	3.8	nein	nein	V100	[20, 92]
Hybrid Task Cascade	X-101-64x4d FPN	47.2	66.6	51.3	27.5	50.1	60.6	-	nein	nein	-	[20, 92]
Hybrid Task Cascade	X-101-64x4d FPN	46.9	-	-	-	-	-	3.5	nein	nein	V100	[20, 92]
Hybrid Task Cascade	X-101-64x4d FPN DCN	50.7	-	-	-	-	-	-	ja	nein	-	[20, 92]
LRF-Net	R-101	34.3	54.1	36.6	13.2	38.2	50.7	79.74	nein	nein	Titan X	[149]
LRF-Net	R-101	37.3	58.5	39.7	19.7	42.8	50	47.35	nein	nein	Titan X	[149]
LRF-Net	VGG-16	32	51.5	33.8	12.6	34.9	47	116.55	nein	nein	Titan X	[149]
LRF-Net	VGG-16	36.2	56.6	38.7	19	39.9	48.8	58.28	nein	nein	Titan X	[149]
Libra Fast R-CNN	R-50-FPN	38.5	-	-	-	-	-	16.3	nein	nein	V100	[96, 92]
Libra Faster R-CNN	R-101-FPN	40.3	-	-	-	-	-	10.4	nein	nein	V100	[96, 92]
Libra Faster R-CNN	R-50 FPN	38.5	-	-	-	-	-	12	nein	nein	V100	[96, 92]
Libra Faster R-CNN	R-50-FPN	38.5	-	-	-	-	-	12	nein	nein	V100	[96, 92]
Libra Faster R-CNN	X-101-64x4d-FPN	42.7	-	-	-	-	-	6.8	nein	nein	V100	[96, 92]
Libra RetinaNet	R-50-FPN	37.7	-	-	-	-	-	11.8	nein	nein	V100	[96, 92]
M2Det	R-101	34.3	53.5	36.5	14.8	38.8	47.9	32.88	nein	nein	Titan X	[159]
M2Det	R-101	38.8	59.4	41.7	20.5	43.9	53.4	23.94	nein	nein	Titan X	[159]
M2Det	R-101	39.7	60	43.3	25.3	42.5	48.3	-	nein	ja	-	[159]
M2Det	R-101	43.9	64.4	48	29.6	49.6	54	-	nein	ja	-	[159]
M2Det	VGG-16	33.5	52.4	35.6	14.4	37.6	47.6	50.61	nein	nein	Titan X	[159]
M2Det	VGG-16	37.6	56.6	40.5	18.4	43.4	51.2	27.27	nein	nein	Titan X	[159]
M2Det	VGG-16	38.9	59.1	42.4	24.4	41.5	47.6	-	nein	ja	-	[159]
M2Det	VGG-16	41	59.7	45	22.1	46.5	53.8	17.88	nein	nein	Titan X	[159]
M2Det	VGG-16	42.9	62.5	47.2	28	47.4	52.8	-	nein	ja	-	[159]
M2Det	VGG-16	44.2	64.6	49.3	29.2	47.9	55.1	-	nein	ja	-	[159]
Mask R-CNN	R-101 FPN	38.2	60.3	41.7	20.1	41.1	50.2	-	nein	nein	-	[46]
Mask R-CNN	R-101 FPN	42.9	-	-	-	-	-	17.86	nein	nein	V100	[46, 29]
Mask R-CNN	R-101 FPN GC(c3-c5 r16)	41.1	-	-	-	-	-	9	nein	nein	V100	[46, 17, 92]
Mask R-CNN	R-101 FPN GC(c3-c5 r4)	41.7	-	-	-	-	-	8.9	nein	nein	V100	[46, 17, 92]
Mask R-CNN	R-101-C4	42.6	-	-	-	-	-	6.9	nein	nein	V100	[46, 29]
Mask R-CNN	R-101-DC5	41.9	-	-	-	-	-	10.87	nein	nein	V100	[46, 29]
Mask R-CNN	R-50 FPN	38.6	-	-	-	-	-	23.26	nein	nein	V100	[46, 29]
Mask R-CNN	R-50 FPN	41	-	-	-	-	-	23.26	nein	nein	V100	[46, 29]
Mask R-CNN	R-50 FPN GC(c3-c5 r16)	37.2	-	-	-	-	-	10.2	nein	nein	V100	[46, 92]
Mask R-CNN	R-50 FPN GC(c3-c5 r4)	39.4	-	-	-	-	-	9.9	nein	nein	V100	[46, 17, 92]
Mask R-CNN	R-50 FPN GC(c3-c5 r4)	39.9	-	-	-	-	-	9.4	nein	nein	V100	[46, 17, 92]

ANHANG B. TABELLE DER GRUNDGESAMTHEIT AN OBJECT DETECTION-ALGORITHMEN

Meta-Architektur	Backbone-Architektur	AP	AP@0.5	AP@0.75	AP _S	AP _M	AP _L	FPS	MS-T	MT-I	GPU	Referenz
Mask R-CNN	R-50-C4	36.8	-	-	-	-	-	9.09	nein	nein	V100	[46, 29]
Mask R-CNN	R-50-C4	39.8	-	-	-	-	-	9.01	nein	nein	V100	[46, 29]
Mask R-CNN	R-50-DC5	38.3	-	-	-	-	-	13.16	nein	nein	V100	[46, 29]
Mask R-CNN	R-50-DC5	40	-	-	-	-	-	13.16	nein	nein	V100	[46, 29]
Mask R-CNN	ResNet101 Dual	41.8	62.9	45.6	37	59.5	39.3	-	nein	nein	-	[46, 75]
Mask R-CNN	ResNet101 Triple	42.4	64	46.7	38.1	59.9	40.8	-	nein	nein	-	[46, 75]
Mask R-CNN	X-101 FPN	39.8	62.3	43.4	22.1	43.2	51.2	-	nein	nein	-	[46]
Mask R-CNN	X-101 FPN GC(c3-c5 r16)	42.4	-	-	-	-	-	7.7	nein	nein	V100	[46, 17, 92]
Mask R-CNN	X-101 FPN GC(c3-c5 r4)	42.9	-	-	-	-	-	7.6	nein	nein	V100	[46, 17, 92]
Mask R-CNN	X-152-32x8d	45.2	-	-	-	-	-	3.08	nein	nein	P100	[36]
Mask R-CNN	X101 FPN	44.3	-	-	-	-	-	7.75	nein	nein	V100	[46, 29]
NAS-FPN (RetinaNet)	AmoebaNet	43.4	-	-	-	-	-	4.97	nein	nein	P100	[36]
NAS-FPN (RetinaNet)	AmoebaNet (7 @ 384)	48	-	-	-	-	-	3.59	nein	nein	P100	[36]
NAS-FPN (RetinaNet)	AmoebaNet (7 @ 384) + DropBlock	48.3	-	-	-	-	-	3.59	nein	nein	P100	[36]
NAS-FPN (RetinaNet)	R-50	40.1	-	-	-	-	-	13.7	nein	nein	P100	[36]
NAS-FPN (RetinaNet)	R-50	41.1	-	-	-	-	-	11.95	nein	nein	P100	[36]
NAS-FPN (RetinaNet)	R-50 (7 @ 256)	39.9	-	-	-	-	-	17.83	nein	nein	P100	[36, 142]
NAS-FPN (RetinaNet)	R-50 (7 @ 256)	44.2	-	-	-	-	-	10.86	nein	nein	P100	[36]
NAS-FPN (RetinaNet)	R-50 (7 @ 256)	44.8	-	-	-	-	-	7.58	nein	nein	P100	[36]
NAS-FPN (RetinaNet)	R-50 (7 @ 384)	45.4	-	-	-	-	-	5.2	nein	nein	P100	[36]
NAS-FPN (RetinaNet)	R-50 (7 @ 384) + DropBlock	46.6	-	-	-	-	-	5.2	nein	nein	P100	[36]
R-FCN	Aligned-Inception-ResNet	32.9	54.5	-	12.5	36.3	48.3	-	nein	nein	-	[23, 24]
R-FCN	R-101	29.9	51.9	-	10.8	32.8	45	-	ja	nein	-	[23]
R-FCN	R-101	30.8	52.6	-	11.8	33.9	44.8	-	nein	nein	-	[23, 24]
R-FCN	R-101	31.1	52.5	-	14.4	34.9	43	-	nein	nein	-	[23, 12]
R-FCN + DCNv1	Aligned-Inception-ResNet	36.1	56.7	-	14.8	39.8	52.2	-	nein	nein	-	[23, 24]
R-FCN + DCNv1	Aligned-Inception-ResNet	37.5	58	-	19.4	40.1	52.5	-	nein	nein	-	[23, 24]
R-FCN + DCNv1	DPN-98	41.2	63.5	45.9	25.7	43.9	52.8	-	nein	nein	-	[23, 125]
R-FCN + DCNv1	R-101	37.4	59.6	-	17.8	40.6	51.4	-	nein	nein	-	[23, 12]
R-FCN + DCNv1 + MST	R-101	39.8	62.4	-	22.6	42.3	52.2	-	nein	nein	-	[23, 12]
R-FCN + DCNv1 + MST	R-101	40.9	62.8	-	23.3	43.6	53.3	-	nein	nein	-	[23, 12]
R-FCN + DCNv1 + SNIP	DPN-98 (with flip)	45.7	67.3	51.1	29.3	48.8	57.1	-	nein	nein	-	[23, 125]
R-FCN + DCNv1 + SNIP	R-101 (R-101 proposals)	43.4	65.5	48.4	27.2	46.5	54.9	-	nein	nein	-	[23, 125]
R-FCN + DCNv1 + SNIP	DPN-98	44.2	65.6	49.7	27.4	47.8	55.8	-	nein	nein	-	[23, 125]
(RCN)	R-FCN + DCNv1 + SNIP	44.7	66.6	50.2	28.5	47.8	55.9	-	nein	nein	-	[23, 125]
(RCN+RPN)	R-FCN + DCNv1 S-NMS	38.4	60.1	-	18.5	41.6	52.5	-	nein	nein	-	[23, 12]
G	R-FCN + MultiScale	34.5	55	-	16.8	37.3	48.3	-	nein	nein	-	[23, 24]

Meta-Architektur	Backbone-Architektur	AP	AP@0.5	AP@0.75	AP _S	AP _M	AP _L	FPS	MS-T	MT-I	GPU	Referenz
R-FCN + MultiScale + Iterative BBox	Aligned-Inception-ResNet	35.5	55.6	-	17.8	38.4	49.3	-	nein	nein	-	[23, 24]
R-FCN + S-NMS G	R-101	32.4	53.4		15.2	36.1	44.3	-	nein	nein	-	[23, 12]
R-FCN + S-NMS L	R-101	32.2	53.4		15.1	36	44.1	-	nein	nein	-	[23, 12]
R-FCN+ MultiScale + DCNv1	Aligned-Inception-ResNet	37.1	57.3	-	18.8	39.7	52.3	-	nein	nein	-	[23, 24]
RFB Net300	VGG	30.3	49.3	31.8	11.8	31.9	45.9	101.01	nein	nein	Titan X	[72]
RFB Net512	VGG	33.8	54.2	35.9	16.2	37.1	47.4	50.51	nein	nein	Titan X	[72]
RFB Net512-E	VGG	34.4	55.7	36.4	17.6	37	47.6	45.91	nein	nein	Titan X	[72]
RUN2WAY300	VGG	27.4	46.1	28.4	8.9	27.9	43.8	63.33	nein	nein	Titan X	[63]
RUN2WAY512	VGG	31.7	52.1	33.6	13.2	33.9	46.5	30.45	nein	nein	Titan X	[63]
RUN3WAY300	VGG	28	47.5	28.9	9.9	28.6	43.9	60.61	nein	nein	Titan X	[63]
RUN3WAY512	VGG	32.4	53.5	34.2	14.7	34	46.7	29.55	nein	nein	Titan X	[63]
RefineDet	R-101	34.4	-	-	-	-	16.84	nein	nein	nein	Titan X	[158]
RefineDet320	R-101	32	51.4	34.2	10.5	34.7	50.4	-	nein	nein	-	[158]
RefineDet320	VGG-16	29.4	49.2	31.3	10	32	44.4	61.06	nein	nein	Titan X	[158]
RefineDet320+	R-101	38.6	59.9	41.7	21.1	41.7	52.3	-	nein	nein	-	[158]
RefineDet320+	VGG-16	35.2	56.1	37.7	19.5	37.2	47	-	nein	nein	-	[158]
RefineDet512	R-101	36.4	57.5	39.5	16.6	39.9	51.4	-	nein	nein	-	[158]
RefineDet512	VGG-16	33	54.5	35.5	16.3	36.3	44.3	36.52	nein	nein	Titan X	[158]
RefineDet512+	R-101	41.8	62.9	45.7	25.6	45.1	54.1	-	nein	nein	-	[158]
RefineDet512+	VGG-16	37.6	58.7	40.8	22.7	40.3	48.3	-	nein	nein	-	[158]
RetinaNet	R-101	34.4	53.1	36.8	14.7	38.5	49.1	-	nein	nein	-	[158]
RetinaNet	R-101 FPN	39.1	59.1	42.3	21.8	42.7	50.2	9.9	nein	nein	M40	[68]
RetinaNet	X-101	40.8	61.1	44.1	24.1	44.2	51.2	-	nein	nein	-	[68, 151]
RetinaNet	X-101 FPN	40.8	61.1	44.1	24.1	44.2	51.2	9.9	nein	nein	M40	[68]
SNIP	DPN-98	45.7	67.3	51.1	29.3	48.8	57.1	-	nein	nein	-	[125]
SNIP	R-50 (fixed BN)	43.6	65.2	48.8	26.4	46.5	55.8	-	nein	nein	-	[125]
SNIPER	MobileNet-V2	34.1	54.4	37.7	18.2	36.9	46.2	-	nein	nein	-	[126]
SNIPER	R-101	46.1	67	51.6	29.6	48.9	58.1	5	nein	nein	V100	[126]
SNIPER	R-101 + OpenImages	46.8	67.4	52.5	30.5	49.4	59.6	5	nein	nein	V100	[126]
SNIPER	R-101 + OpenImages + Seg Binary	47.1	67.8	52.8	30.2	49.9	60.2	5	nein	nein	V100	[126]
SNIPER	R-101 + OpenImages + Seg Softmax	47.6	68.5	53.4	30.9	50.6	60.7	5	nein	nein	V100	[126]
SNIPER	R-50 (fixed BN)	43.5	65	48.6	26.1	46.3	56	-	nein	nein	-	[126]
SSD300*	VGG	25.1	43.1	25.8	6.6	25.9	41.4	-	nein	nein	-	[74, 32]
SSD321	Residual-101	28	45.4	29.3	6.2	28.3	49.3	24.85	nein	nein	Titan X	[74, 32]
SSD512*	VGG	28.8	48.5	30.3	10.9	31.8	43.5	-	nein	nein	-	[74, 32]
SSD513	Residual-101	31.2	50.4	33.3	10.2	34.5	49.8	12.12	nein	nein	Titan X	[74, 32]
ScarF Faster R-CNN	X-101	38.5	59.9	41.5	19.1	42.9	54.1	-	nein	nein	-	[151]
ScarF Faster R-CNN	X-101	42.8	64.3	47.1	26	45.7	52.9	-	nein	nein	-	[151]

ANHANG B. TABELLE DER GRUNDGESAMTHEIT AN OBJECT DETECTION-ALGORITHMEN

Meta-Architektur	Backbone-Architektur	AP	AP@0.5	AP@0.75	AP _S	AP _M	AP _L	FPS	MS-T	MT-I	GPU	Referenz	
Scarf RetinaNet	R-101	35.1	53.8	37.7	15.8	49	-	nein	nein	-	[151]		
Scarf RetinaNet	X-101	41.6	62	44.6	24.5	52.3	-	nein	nein	-	[151]		
Scarf SSD513	R-101	34.5	54.1	36.3	15.1	36.1	51.6	-	nein	nein	-	[151]	
TridentNet	R-101	42.7	63.6	46.5	23.9	46.6	56.6	-	nein	nein	-	[65]	
TridentNet*	R-101 DCNv1	46.8	67.6	51.5	28	51.2	60.5	-	nein	nein	-	[65]	
TridentNet* + Image Pyramid	R-101 DCNv1	48.4	69.7	53.5	31.8	51.3	60.3	-	nein	nein	-	[65]	
YOLOACT-400	R-101-FPN	28.4	-	-	10.7	28.9	43.1	56.82	nein	nein	Titan Xp	[13]	
YOLOACT-550	R-101-FPN	31	-	-	14.4	31.8	43.7	41.67	nein	nein	Titan Xp	[13]	
YOLOACT-550	R-50-FPN	30.3	-	-	14	31.2	43	54.35	nein	nein	Titan Xp	[13]	
YOLOACT-700	R-101-FPN	33.7	-	-	16.8	35.6	45.7	29.76	nein	nein	Titan Xp	[13]	
YOLOv2	DarkNet-19	21.6	44	19.2	5	22.4	35.5	-	nein	nein	-	[109]	
YOLOv3 @320 + ASFF	DarkNet-53	36.7	57.2	39.5	15.8	39.9	51.3	63	nein	nein	V100	[110, 73]	
YOLOv3 @320 + ASFF*	DarkNet-53	38.1	57.4	42.1	16.1	41.6	53.6	60	nein	nein	V100	[110, 73]	
YOLOv3 @416 + ASFF	DarkNet-53	39	60.2	42.5	19.6	42.3	51.4	56	nein	nein	V100	[110, 73]	
YOLOv3 @416 + ASFF*	DarkNet-53	40.6	60.6	45.1	20.3	44.2	54.1	54	nein	nein	V100	[110, 73]	
YOLOv3 @608 (baseline)	DarkNet-53	38.8	58.3	43	24.6	42.9	51.6	50	nein	nein	V100	[110, 73]	
YOLOv3 @608 + ASFF	DarkNet-53	40.7	62.9	44.1	24.5	43.6	49.3	46.6	ja	nein	V100	[110, 73]	
YOLOv3 @608 + ASFF*	DarkNet-53	42.4	63	47.4	25.5	45.7	52.3	45.5	ja	nein	V100	[110, 73]	
YOLOv3 @800 + ASFF*	DarkNet-53	43.9	64.1	49.2	27	46.6	53.4	29.4	ja	nein	V100	[110, 73]	
YOLOv3-320	DarkNet-53	28.2	51.5	-	-	-	-	68.87	nein	nein	Titan X	[110]	
YOLOv3-320	DarkNet-53	28.2	51.5	-	-	-	-	69	nein	nein	V100	[110, 73]	
YOLOv3-416	DarkNet-53	31	55.3	-	-	-	-	52.25	nein	nein	Titan X	[110]	
YOLOv3-416	DarkNet-53	31	55.3	-	-	-	-	60	nein	nein	V100	[110, 73]	
YOLOv3-608	DarkNet-53	33	57.9	34.4	18.3	35.4	41.9	29.71	ja	nein	Titan X	[110]	
YOLOv3-608	DarkNet-53	33	57.9	34.4	18.3	35.4	41.9	52	ja	nein	V100	[110, 73]	

Anhang C

Trainingsverläufe der Benchmark-Verfahren

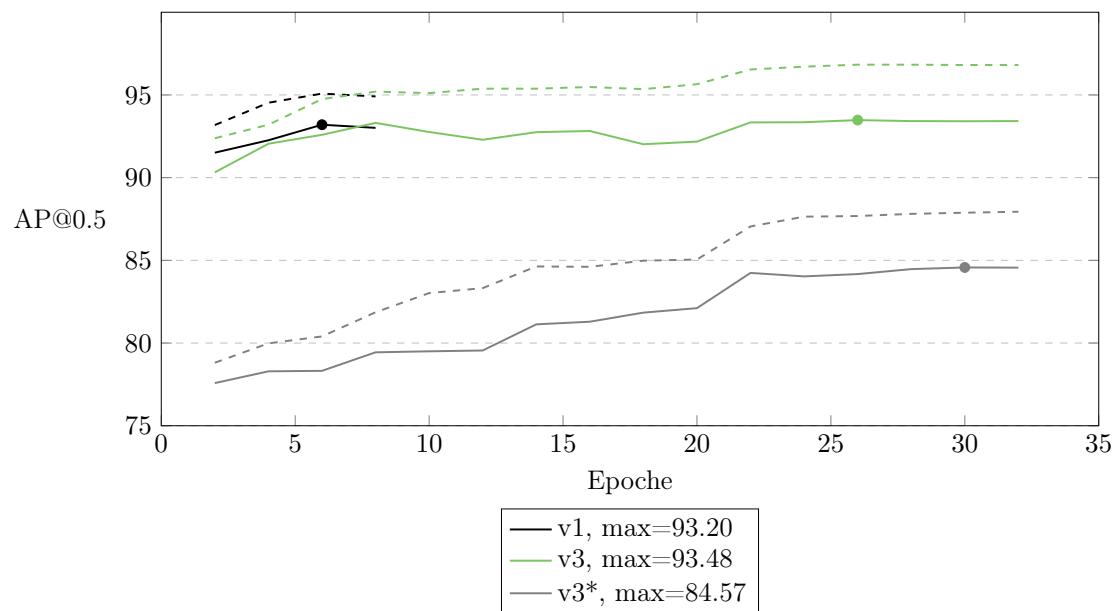


Abbildung C.1: Trainingsverläufe der Konfigurationen für ATSS X-101-FD @1333x800

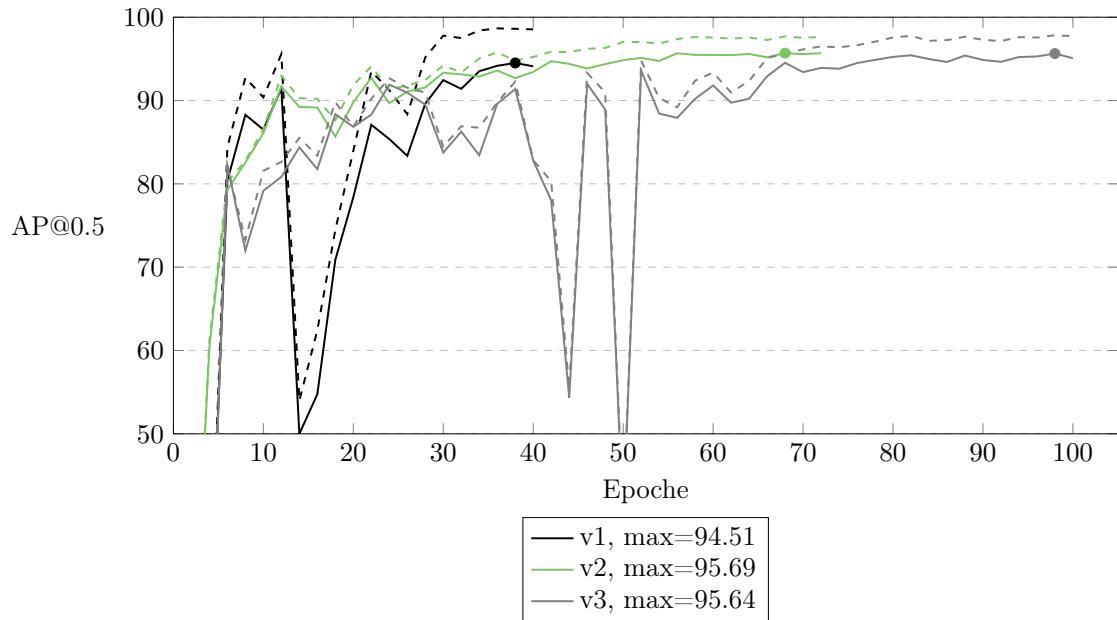


Abbildung C.2: Trainingsverläufe der Konfigurationen für YOLOv3-ASFF* D-53 @800

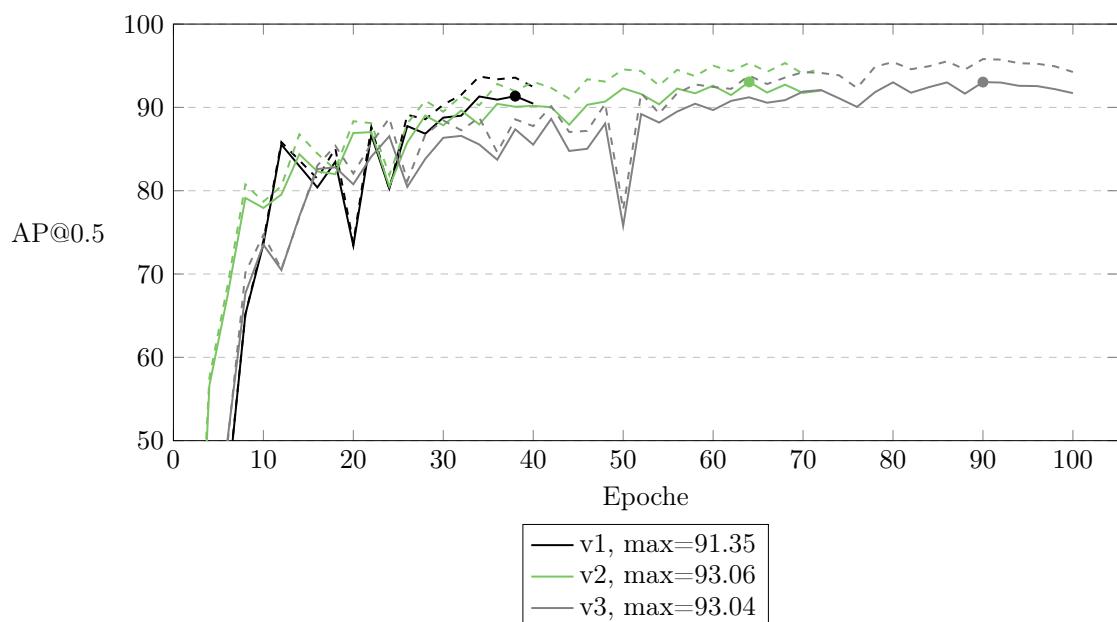


Abbildung C.3: Trainingsverläufe der Konfigurationen für YOLOv3-ASFF* D-53 @416

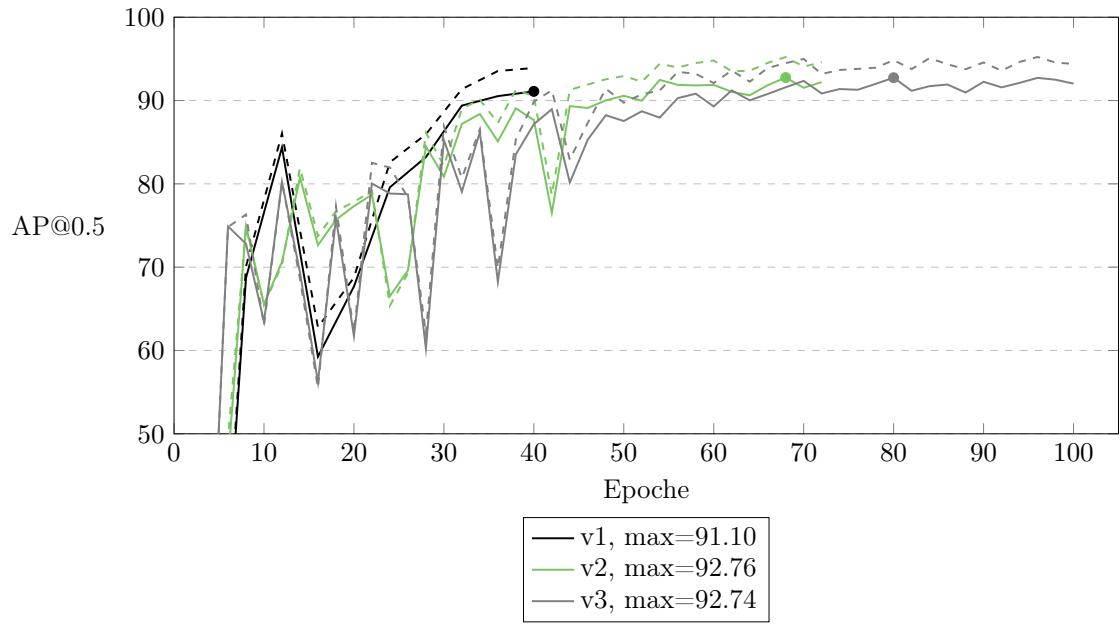


Abbildung C.4: Trainingsverläufe der Konfigurationen für YOLOv3-ASFF D-53 @416

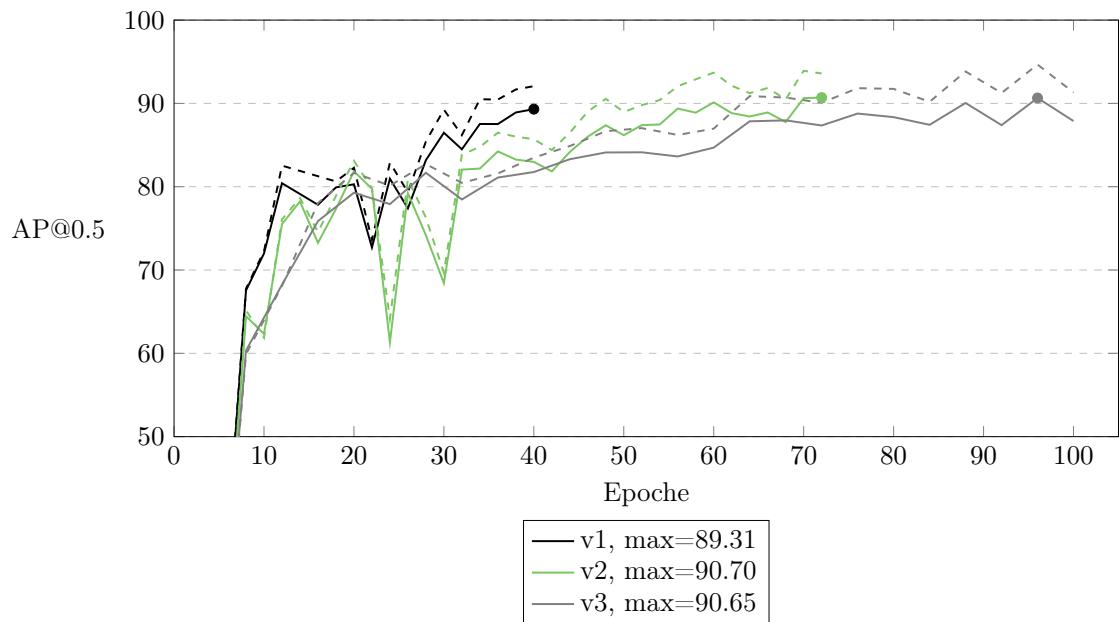


Abbildung C.5: Trainingsverläufe der Konfigurationen für YOLOv3-ASFF* D-53 @320

Anhang D

Verlauf der k-Fold Cross Validation der Benchmark-Verfahren

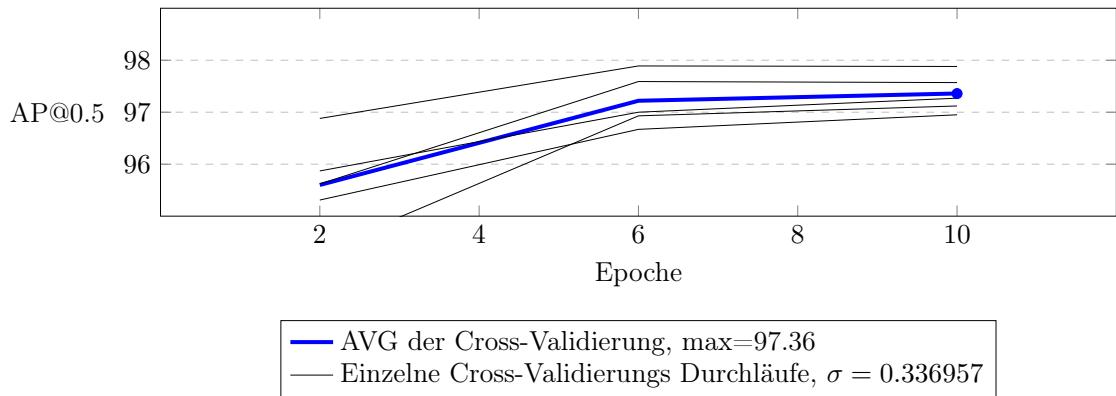


Abbildung D.1: Cross-Validierung der Modellkonfiguration für HTC X-101-FD @1333x800

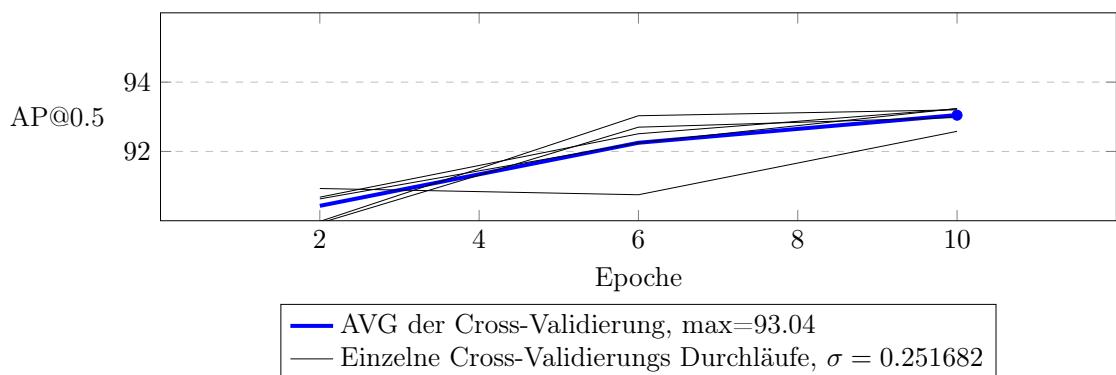


Abbildung D.2: Cross-Validierung der Modellkonfiguration für HTC X-101-FD @666x400

ANHANG D. VERLAUF DER K-FOLD CROSS VALIDATION DER BENCHMARK-VERFAHREN

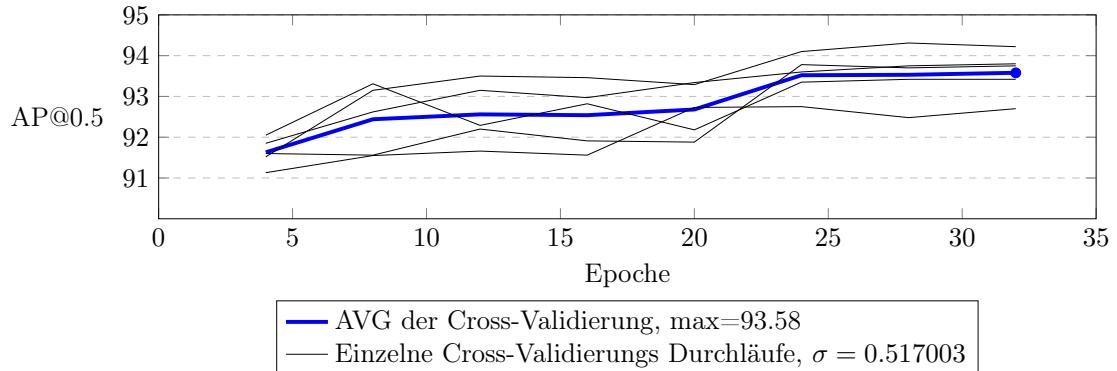


Abbildung D.3: Cross-Validierung der Modellkonfiguration für ATSS X-101-FD @1333x800

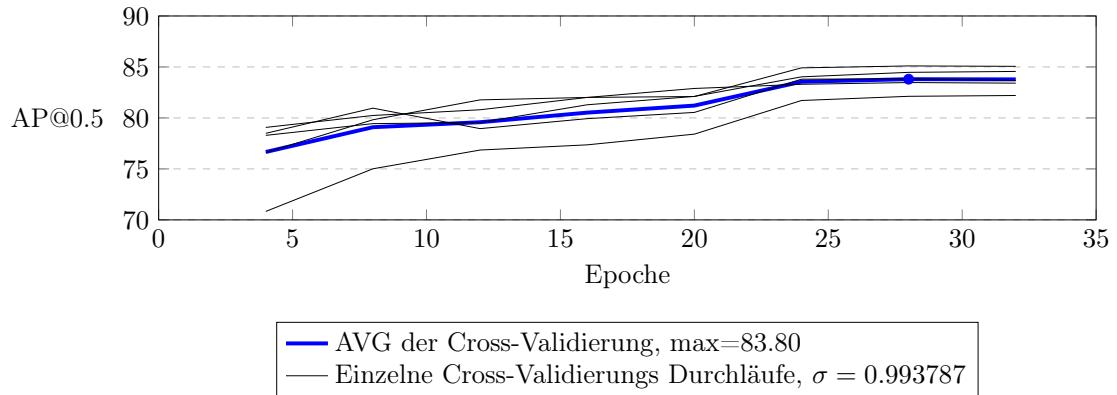


Abbildung D.4: Cross-Validierung der Modellkonfiguration für ATSS X-101-FD @666x400

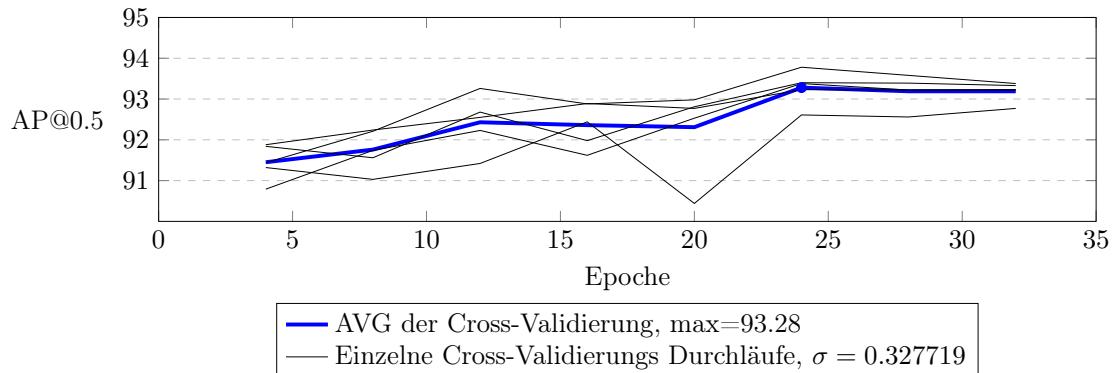


Abbildung D.5: Cross-Validierung der Modellkonfiguration für ATSS R-101-FD @1333x800

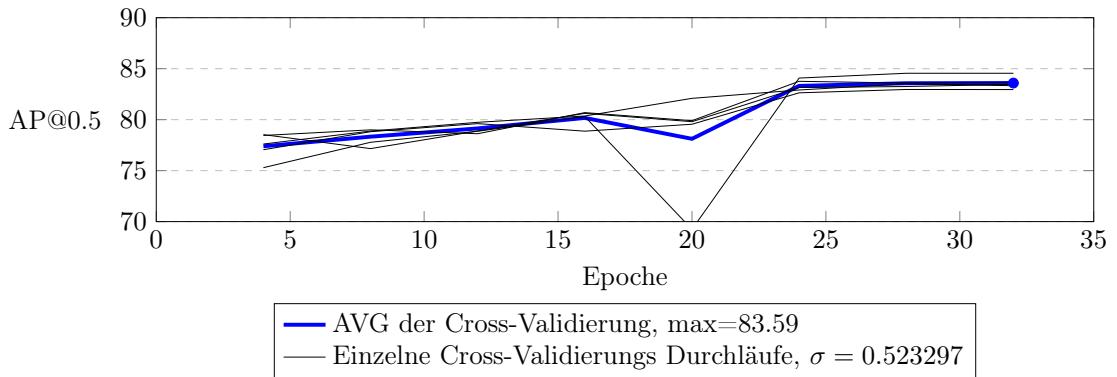


Abbildung D.6: Cross-Validierung der Modellkonfiguration für ATSS R-101-FD @666x400

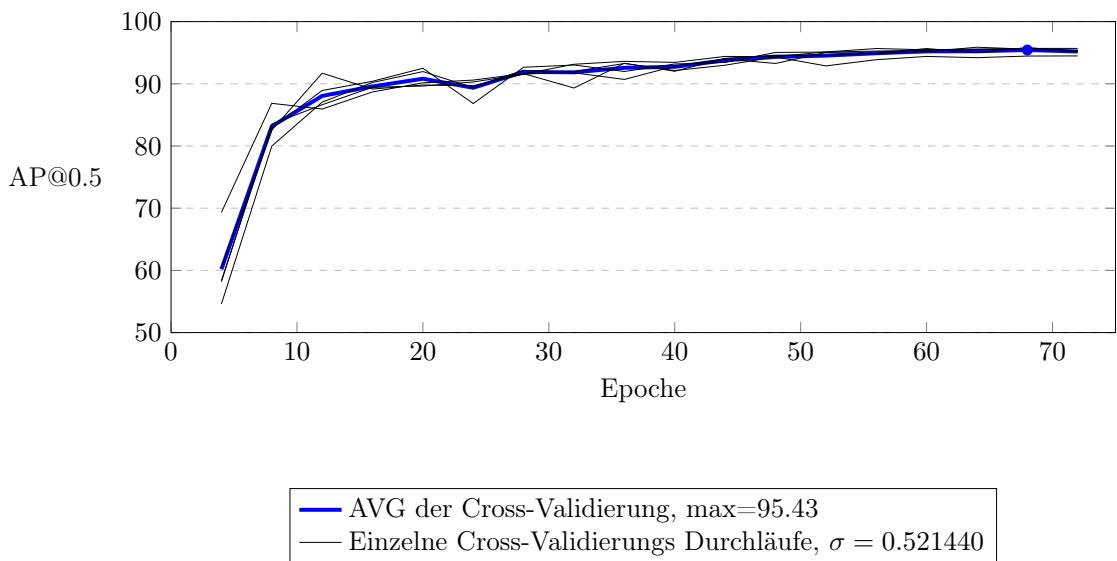


Abbildung D.7: Cross-Validierung der Modellkonfiguration für YOLOv3-ASFF* D-53 @800

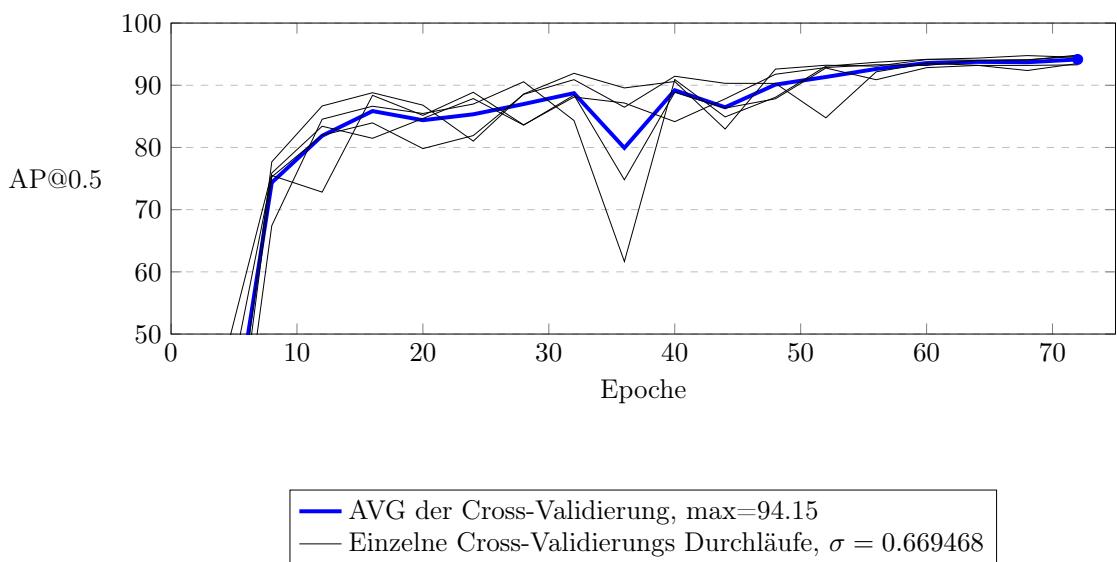


Abbildung D.8: Cross-Validierung der Modellkonfiguration für YOLOv3-ASFF* D-53 @608

ANHANG D. VERLAUF DER K-FOLD CROSS VALIDATION DER BENCHMARK-VERFAHREN

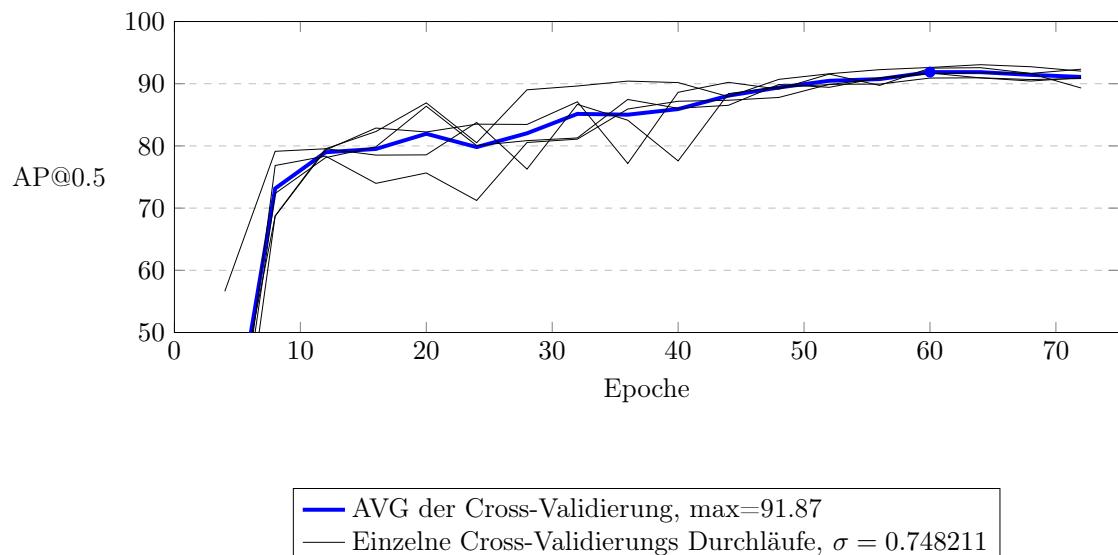


Abbildung D.9: Cross-Validierung der Modellkonfiguration für YOLOv3-ASFF* D-53 @416

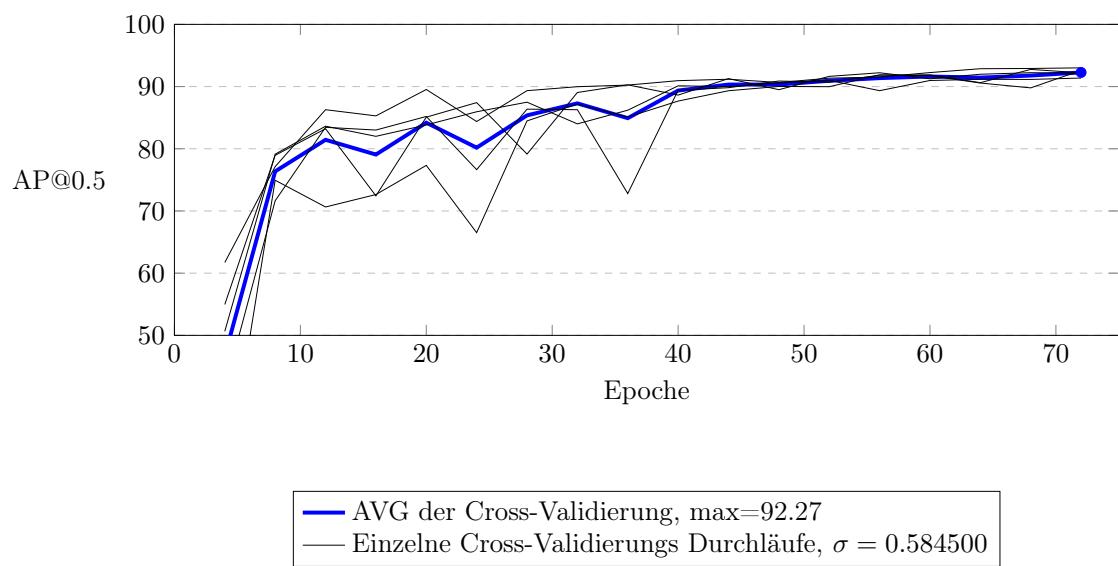


Abbildung D.10: Cross-Validierung der Modellkonfiguration für YOLOv3-ASFF D-53 @416

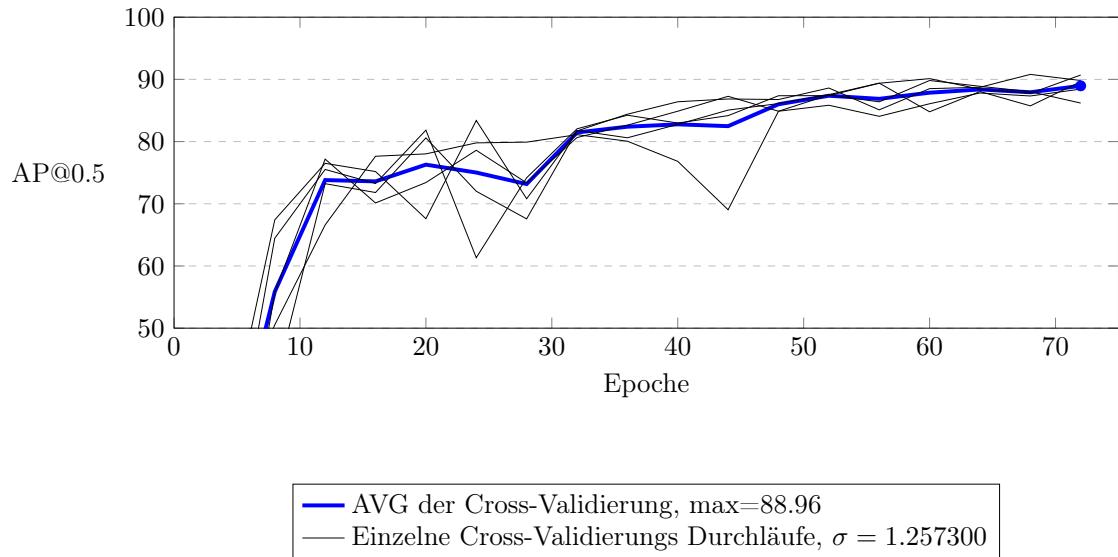


Abbildung D.11: Cross-Validierung der Modellkonfiguration für YOLOv3-ASFF* D-53 @320

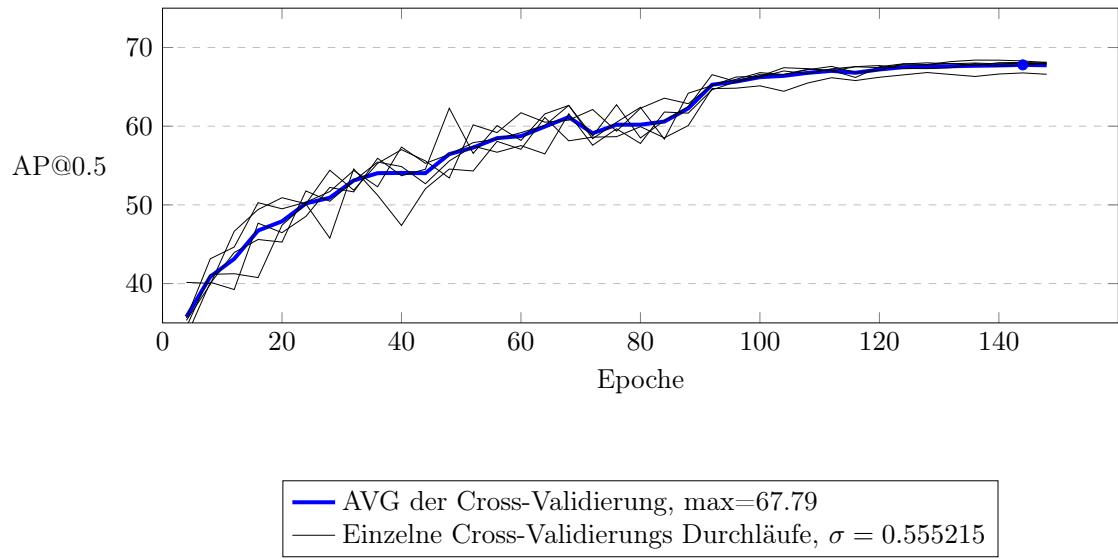


Abbildung D.12: Cross-Validierung der Modellkonfiguration für RFBNet VGG-16 @300

ANHANG D. VERLAUF DER K-FOLD CROSS VALIDATION DER
BENCHMARK-VERFAHREN

Anhang E

Detaillierte Metriken der Benchmark-Verfahren

Objektklasse	RC_k	PR_k	$\text{AP}_{k,s}^*$	$\text{AP}_{k,m}^*$	$\text{AP}_{k,l}^*$	$\text{AP}_k @ 0.5$
Person	97.66	95.66	95.33	99.50	-	97.39
Palette	97.61	95.83	95.53	98.70	99.94	97.36
Hubwagen	96.34	96.67	95.46	98.15	-	96.07
Gabelstapler	98.51	97.29	94.90	99.02	98.15	98.37

Tabelle E.1: Evaluationsmetriken für HTC X-101-FD @1333x800

Objektklasse	RC_k	PR_k	$\text{AP}_{k,s}^*$	$\text{AP}_{k,m}^*$	$\text{AP}_{k,l}^*$	$\text{AP}_k @ 0.5$
Person	93.74	94.83	87.87	99.29	-	93.43
Palette	93.66	95.05	86.25	98.58	99.05	93.31
Hubwagen	87.76	94.90	83.90	97.40	-	87.41
Gabelstapler	98.33	96.94	92.70	98.73	98.74	98.03

Tabelle E.2: Evaluationsmetriken für HTC X-101-FD @666x400

Objektklasse	RC_k	PR_k	AP_{k,s}[*]	AP_{k,m}[*]	AP_{k,l}[*]	AP_{k@0.5}
Person	94.91	97.16	90.44	98.66	-	94.42
Palette	95.57	98.08	92.14	97.24	98.40	95.09
Hubwagen	90.08	97.73	87.48	95.66	-	89.60
Gabelstapler	95.49	98.37	82.89	96.63	96.80	95.20
AP@0.5						93.58

Tabelle E.3: Evaluationsmetriken für ATSS X-101-FD @1333x800

Objektklasse	RC_k	PR_k	AP_{k,s}[*]	AP_{k,m}[*]	AP_{k,l}[*]	AP_{k@0.5}
Person	84.75	97.04	71.69	97.77	-	84.28
Palette	90.11	97.73	80.00	96.80	98.40	89.68
Hubwagen	66.96	96.66	58.59	87.88	-	66.38
Gabelstapler	95.07	98.27	85.16	96.22	95.68	94.85
AP@0.5						83.80

Tabelle E.4: Evaluationsmetriken für ATSS X-101-FD @666x400

Objektklasse	RC_k	PR_k	AP_{k,s}[*]	AP_{k,m}[*]	AP_{k,l}[*]	AP_{k@0.5}
Person	94.31	97.08	89.23	98.84	-	93.89
Palette	95.40	97.94	92.16	97.15	98.40	95.08
Hubwagen	88.87	97.11	85.83	95.54	-	88.40
Gabelstapler	95.90	98.63	87.48	96.86	96.52	95.75
AP@0.5						93.28

Tabelle E.5: Evaluationsmetriken für ATSS R-101-FD @1333x800

Objektklasse	RC_k	PR_k	AP_{k,s}[*]	AP_{k,m}[*]	AP_{k,l}[*]	AP_{k@0.5}
Person	84.30	96.57	70.55	97.67	-	83.72
Palette	90.21	97.71	80.56	96.64	99.20	89.81
Hubwagen	66.65	97.13	58.29	87.85	-	66.15
Gabelstapler	94.81	98.52	83.92	96.07	95.81	94.69
AP@0.5						83.59

Tabelle E.6: Evaluationsmetriken für ATSS R-101-FD @666x400

Objektklasse	RC_k	PR_k	AP_{k,s}[*]	AP_{k,m}[*]	AP_{k,l}[*]	AP_k@0.5
Person	95.89	97.43	92.15	98.81	-	95.41
Palette	96.18	96.20	93.30	97.26	100.00	95.60
Hubwagen	94.95	97.07	93.75	96.95	-	94.50
Gabelstapler	96.29	98.67	89.77	97.08	96.88	96.20
AP@0.5						95.43

Tabelle E.7: Evaluationsmetriken für YOLOv3-ASFF* D-53 @800

Objektklasse	RC_k	PR_k	AP_{k,s}[*]	AP_{k,m}[*]	AP_{k,l}[*]	AP_k@0.5
Person	94.26	97.47	88.93	98.83	-	93.71
Palette	95.97	96.87	93.16	97.04	99.07	95.37
Hubwagen	92.71	97.41	90.68	97.14	-	92.33
Gabelstapler	95.51	98.15	87.85	96.12	96.07	95.19
AP@0.5						94.15

Tabelle E.8: Evaluationsmetriken für YOLOv3-ASFF* D-53 @608

Objektklasse	RC_k	PR_k	AP_{k,s}[*]	AP_{k,m}[*]	AP_{k,l}[*]	AP_k@0.5
Person	91.65	96.28	84.02	98.36	-	90.92
Palette	93.94	96.10	88.43	96.54	98.94	93.04
Hubwagen	90.34	93.78	86.94	96.41	-	89.35
Gabelstapler	94.54	98.13	82.35	95.69	95.62	94.18
AP@0.5						91.87

Tabelle E.9: Evaluationsmetriken für YOLOv3-ASFF* D-53 @416

Objektklasse	RC_k	PR_k	AP_{k,s}[*]	AP_{k,m}[*]	AP_{k,l}[*]	AP_k@0.5
Person	91.42	96.88	83.43	98.60	-	90.74
Palette	94.92	95.31	90.47	97.16	99.91	94.30
Hubwagen	90.66	94.80	87.66	97.16	-	90.12
Gabelstapler	94.35	98.36	82.28	95.37	95.18	93.93
AP@0.5						92.27

Tabelle E.10: Evaluationsmetriken für YOLOv3-ASFF D-53 @416

Objektklasse	RC_k	PR_k	AP_{k,s}[*]	AP_{k,m}[*]	AP_{k,l}[*]	AP_k@0.5
Person	87.27	95.75	75.20	98.28	-	86.48
Palette	92.26	96.57	84.19	96.70	99.11	91.37
Hubwagen	84.99	91.55	78.71	96.93	-	83.63
Gabelstapler	94.76	98.35	82.03	95.70	96.59	94.35
AP@0.5						88.96

Tabelle E.11: Evaluationsmetriken für YOLOv3-ASFF* D-53 @320

Objektklasse	RC_k	PR_k	AP_{k,s}[*]	AP_{k,m}[*]	AP_{k,l}[*]	AP_k@0.5
Person	67.87	90.74	44.48	86.82	-	65.74
Palette	69.70	92.06	34.69	88.98	99.20	66.93
Hubwagen	53.58	86.63	40.70	78.62	-	51.10
Gabelstapler	88.25	97.15	58.02	90.74	92.03	87.40
AP@0.5						67.79

Tabelle E.12: Evaluationsmetriken für RFBNet VGG-16 @300

Abbildungsverzeichnis

2.1	Visualisierung der Ausgabe eines Object Detection-Algorithmus	6
2.2	Hardware Setup in der Lagerhalle	9
2.3	Dezentrale vs zentrale Inferenz auf unterschiedlichen Plattformen	10
3.1	Abgrenzung künstliche Intelligenz und Deep Learning	14
3.2	Gegenüberstellung Deep Learning-basierter Algorithmen der Bildverarbeitung	16
3.3	Berechnungsmodell eines einzelnen künstlichen Neurons	17
3.4	Architektur eines ANN	18
3.5	Underfitting und Overfitting im Vergleich	20
3.6	Funktionsweise eines CNNs	21
3.7	Prinzip eines Convolutional Layers	22
3.8	Funktionsweise eines Pooling Layers	23
3.9	Funktionsweise eines ROI Pooling Layer	23
3.10	Gegenüberstellung eines normalen Kernel und eines Kernel mit Dilation .	25
3.11	Gegenüberstellung eines normalen Kernel und eines Kernel mit Offsets.	26
3.12	CNN zur Bildklassifikation	27
3.13	Gegenüberstellung der Architekturen von One- und Two-Stage Verfahren .	29
3.14	Ergebnis vor und nach Anwendung des NMS-Algorithmus	30
3.15	PR-RC-Kurve p auf Grundlage der Daten in Tabelle 3.1	34
3.16	Illustration der Treppefunktion p_{interp}	35
3.17	Graph zur Schilderung der Berechnung der adaptierten AP-Metrik	36
4.1	Übersicht der Phasen von CRISP-DM	38
4.2	Gegenüberstellung der zu detektierenden Objektklassen	40
4.3	Erstellung von Annotationen mit CVAT	41
5.1	Histogramm der ganzzahligen Werte der FPS.	56
5.2	Inferenzfehler von <i>EfficientDet-D7 + AA</i>	59
6.1	Residual Block der ResNet-Architektur	65
6.2	Vergleich eines ResNet- und ResNeXt-Block	65

6.3	Struktur des FPN.	67
6.4	Architektur des HTC Object Detection-Algorithmus	68
6.5	Schrittbasierter LR-Scheduling	69
6.6	Zuordnung von Anchor-Boxen zu Ground-Truth Boxen	70
6.7	Aufbau des RFB Moduls	71
6.8	Illustration der <i>Adatively Spatial Feature Fusion</i>	77
6.9	Cosinus LR-Scheduling	78
7.1	Trainingsverläufe der Konfigurationen für HTC X-101-FD @1333x800 . . .	88
7.2	Trainingsverläufe der Konfigurationen für ATSS R-101-FD @1333x800 . . .	91
7.3	Trainingsverläufe der Konfigurationen für YOLOv3-ASFF* D-53 @608 . . .	93
7.4	Trainingsverläufe der Konfigurationen für RFBNet VGG-16 @300	95
8.1	FPS Vergleich aller Architekturen	107
8.2	AP@0.5-Metrik Vergleich aller Architekturen	107
C.1	Trainingsverläufe der Konfigurationen für ATSS X-101-FD @1333x800 . . .	125
C.2	Trainingsverläufe der Konfigurationen für YOLOv3-ASFF* D-53 @800 . . .	126
C.3	Trainingsverläufe der Konfigurationen für YOLOv3-ASFF* D-53 @416 . . .	126
C.4	Trainingsverläufe der Konfigurationen für YOLOv3-ASFF D-53 @416 . . .	127
C.5	Trainingsverläufe der Konfigurationen für YOLOv3-ASFF* D-53 @320 . . .	127
D.1	Cross-Validierung der Modellkonfiguration für HTC X-101-FD @1333x800 .	129
D.2	Cross-Validierung der Modellkonfiguration für HTC X-101-FD @666x400 .	129
D.3	Cross-Validierung der Modellkonfiguration für ATSS X-101-FD @1333x800 .	130
D.4	Cross-Validierung der Modellkonfiguration für ATSS X-101-FD @666x400 .	130
D.5	Cross-Validierung der Modellkonfiguration für ATSS R-101-FD @1333x800 .	130
D.6	Cross-Validierung der Modellkonfiguration für ATSS R-101-FD @666x400 .	131
D.7	Cross-Validierung der Modellkonfiguration für YOLOv3-ASFF* D-53 @800 .	131
D.8	Cross-Validierung der Modellkonfiguration für YOLOv3-ASFF* D-53 @608 .	131
D.9	Cross-Validierung der Modellkonfiguration für YOLOv3-ASFF* D-53 @416 .	132
D.10	Cross-Validierung der Modellkonfiguration für YOLOv3-ASFF D-53 @416 .	132
D.11	Cross-Validierung der Modellkonfiguration für YOLOv3-ASFF* D-53 @320 .	133
D.12	Cross-Validierung der Modellkonfiguration für RFBNet VGG-16 @300 . . .	133

Tabellenverzeichnis

2.1	Grundanforderungen an das Benchmarking.	11
3.1	Gemessene PR- und RC-Metrik für Experiment E	34
3.2	Werte für $p_{interp}(r)$ mit $r \in R$ für Experiment E	34
3.3	Metriken des MS COCO-Datensatzes	35
4.1	Auflistung von Statistiken für den Datensatz $D2$.	49
5.1	Faktoren zur Umrechnung der FPS zwischen der NVIDIA RTX 2080 Ti	55
5.2	Anforderungen für das Benchmark	58
5.3	Auflistung der finalen Auswahl der im Benchmark anzuwendenden Verfahren	62
6.1	Originalkonfiguration der <i>HTC X-101-FD</i> -Architektur	69
6.2	Originalkonfiguration der <i>RFBNet VGG-16 @300</i> -Architektur	72
6.3	Originalkonfiguration der ATSS-Architekturen	75
6.4	Originalkonfigurationen der YOLO-v3 ASFF-Architekturen	78
7.1	Parameterkonfigurationen für HTC X-101-FD @1333x800	88
7.2	Parameterkonfigurationen für ATSS R-101-FD @1333x800	90
7.3	Parameterkonfigurationen für YOLOv3-ASFF* D-53 @608	93
7.4	Parameterkonfigurationen für RFBNet VGG-16 @300	94
8.1	Gegenüberstellung der Ergebnisse der k-Fold Cross Validation.	101
B.1	Grundgesamtheit der Object Detection.	116
E.1	Evaluationsmetriken für HTC X-101-FD @1333x800	135
E.2	Evaluationsmetriken für HTC X-101-FD @666x400	135
E.3	Evaluationsmetriken für ATSS X-101-FD @1333x800	136
E.4	Evaluationsmetriken für ATSS X-101-FD @666x400	136
E.5	Evaluationsmetriken für ATSS R-101-FD @1333x800	136
E.6	Evaluationsmetriken für ATSS R-101-FD @666x400	136
E.7	Evaluationsmetriken für YOLOv3-ASFF* D-53 @800	137

TABELLENVERZEICHNIS

E.8	Evaluationsmetriken für YOLOv3-ASFF* D-53 @608	137
E.9	Evaluationsmetriken für YOLOv3-ASFF* D-53 @416	137
E.10	Evaluationsmetriken für YOLOv3-ASFF D-53 @416	137
E.11	Evaluationsmetriken für YOLOv3-ASFF* D-53 @320	138
E.12	Evaluationsmetriken für RFBNet VGG-16 @300	138

Liste der Algorithmen

1	Bestimmung der True Positive und True Negative	32
2	Adaptive Training Sample Selection (ATSS)	74
3	NMS Algorithmus	113

Literaturverzeichnis

- [1] ABADI, Martin ; AGARWAL, Ashish ; BARHAM, Paul ; BREVDO, Eugene ; CHEN, Zhifeng ; CITRO, Craig ; CORRADO, Greg S. ; DAVIS, Andy ; DEAN, Jeffrey ; DEVIN, Matthieu ; GHEMAWAT, Sanjay ; GOODFELLOW, Ian ; HARP, Andrew ; IRVING, Geoffrey ; ISARD, Michael ; JIA, Yangqing ; JOZEFOWICZ, Rafal ; KAISER, Lukasz ; KUDLUR, Manjunath ; LEVENBERG, Josh ; MANE, Dandelion ; MONGA, Rajat ; MOORE, Sherry ; MURRAY, Derek ; OLAH, Chris ; SCHUSTER, Mike ; SHLENS, Jonathon ; STEINER, Benoit ; SUTSKEVER, Ilya ; TALWAR, Kunal ; TUCKER, Paul ; VANHOUCKE, Vincent ; VASUDEVAN, Vijay ; VIEGAS, Fernanda ; VINYALS, Oriol ; WARDEN, Pete ; WATTENBERG, Martin ; WICKE, Martin ; YU, Yuan ; ZHENG, Xiaoqiang: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* <https://www.tensorflow.org/>. Version: 2015. – Software available from tensorflow.org
- [2] ADLINK: *Frame Grabbers - Video Capture Cards PCIe-GIE72/74.* https://www.adlinktech.com/Products/Machine_Vision/FrameGrabbers_VideoCaptureCards/PCIe-GIE72_74?lang=en, o. J.. – Abgerufen am: 17.07.2020
- [3] ANANDTECH: *NVIDIA Volta Unveiled: GV100 GPU and Tesla V100 Accelerator Announced.* <https://www.anandtech.com/show/11367/nvidia-volta-unveiled-gv100-gpu-and-tesla-v100-accelerator-announced>, 2017. – Abgerufen am: 17.07.2020
- [4] ANANDTECH: *The NVIDIA Titan V Deep Learning Deep Dive: It's All About The Tensor Cores.* <https://www.anandtech.com/show/12673/titan-v-deep-learning-deep-dive>, 2018. – Abgerufen am: 17.07.2020
- [5] ATHANASIADIS, Ioannis ; MOUSOULIOTIS, Panagiotis ; PETROU, Loukas: A Framework of Transfer Learning in Object Detection for Embedded Systems. In: *ArXiv* abs/1811.04863 (2018)
- [6] BASLER: *aca1440-73gc - Basler ace.* <https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/aca1440-73gc/>, o. J.. – Abgerufen am: 17.07.2020
- [7] BASLER: *Basler Lens C125-0818-5M-P f8mm - Lens.* <https://www.baslerweb.com/en/products/vision-components/lenses/basler-lens-c125-0818-5m-p-f8mm/>, o. J.. – Abgerufen am: 17.07.2020

- [8] BHANDE, Anup: *What is underfitting and overfitting in machine learning and how to deal with it.* <https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76>, 2018. – Abgerufen am: 17.07.2020
- [9] BOCHKOVSKIY, Alexey ; WANG, Chien-Yao ; LIAO, Hong-Yuan M.: *YOLOv4: Optimal Speed and Accuracy of Object Detection.* 2020
- [10] BODLA, Navaneeth ; SINGH, Bharat ; CHELLAPPA, Rama ; DAVIS, Larry S.: Improving Object Detection With One Line of Code. In: *CoRR* abs/1704.04503 (2017). <http://arxiv.org/abs/1704.04503>
- [11] BODLA, Navaneeth ; SINGH, Bharat ; CHELLAPPA, Rama ; DAVIS, Larry S.: Soft-NMS — Improving Object Detection with One Line of Code. In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), S. 5562–5570
- [12] BODLA, Navaneeth ; SINGH, Bharat ; CHELLAPPA, Rama ; DAVIS, Larry S.: Soft-NMS — Improving Object Detection with One Line of Code. In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), S. 5562–5570
- [13] BOLYA, Daniel ; ZHOU, Chong ; XIAO, Fanyi ; LEE, Yong J.: YOLACT: Real-Time Instance Segmentation. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), S. 9156–9165
- [14] BROWNLEE, Jason: *How to Avoid Overfitting in Deep Learning Neural Networks.* <https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/>, 2019. – Abgerufen am: 17.07.2020
- [15] CAI, Zhaowei ; VASCONCELOS, Nuno: Cascade R-CNN: High Quality Object Detection and Instance Segmentation. In: *CoRR* abs/1906.09756 (2019). <http://arxiv.org/abs/1906.09756>
- [16] CAI, Zhaowei ; VASCONCELOS, Nuno: Cascade R-CNN: High Quality Object Detection and Instance Segmentation. In: *IEEE transactions on pattern analysis and machine intelligence* (2019)
- [17] CAO, Yue ; XU, Jiarui ; LIN, Stephen ; WEI, Fangyun ; HU, Han: GCNet: Non-Local Networks Meet Squeeze-Excitation Networks and Beyond. In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)* (2019), S. 1971–1980
- [18] CHAPMAN, Pete ; CLINTON, Julian ; KERBER, Randy ; KHABAZA, Thomas ; REINARTZ, Thomas ; SHEARER, Colin ; WIRTH, Rudiger: CRISP-DM 1.0 Step-by-step data mining guide / The CRISP-DM consortium. 2000. – Forschungsbericht

- [19] CHATTERJEE, Chandra C.: *Basics of the Classic CNN*. <https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add>, 2019. – Abgerufen am: 17.07.2020
- [20] CHEN, Kai ; PANG, Jiangmiao ; WANG, Jiaqi ; XIONG, Yu ; LI, Xiaoxiao ; SUN, Shuyang ; FENG, Wansen ; LIU, Ziwei ; SHI, Jianping ; OUYANG, Wanli ; LOY, Chen C. ; LIN, Dahua: Hybrid Task Cascade for Instance Segmentation. In: *CoRR* abs/1901.07518 (2019). <http://arxiv.org/abs/1901.07518>
- [21] CHOLLET, François u. a.: *Keras*. <https://keras.io>, 2015. – Abgerufen am: 17.07.2020
- [22] COMPUTER VISION LAB, ETH ZURICH, SWITZERLAND: *AI Benchmark for Windows, Linux and macOS: Let the AI Games Begin...* http://ai-benchmark.com/ranking_deeplearning_detailed.html, o. J.. – Abgerufen am: 17.07.2020
- [23] DAI, Jifeng ; LI, Yi ; HE, Kaiming ; SUN, Jian: R-FCN: Object Detection via Region-Based Fully Convolutional Networks. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. Red Hook, NY, USA : Curran Associates Inc., 2016 (NIPS 16). – ISBN 9781510838819, S. 379–387
- [24] DAI, Jifeng ; QI, Haozhi ; XIONG, Yuwen ; LI, Yi ; ZHANG, Guodong ; HU, Han ; WEI, Yichen: Deformable Convolutional Networks. In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), S. 764–773
- [25] DEBEASI, Paul: *Training versus Inference*. <https://blogs.gartner.com/paul-debeasi/2019/02/14/training-versus-inference/>, 2019. – Abgerufen am: 17.07.2020
- [26] DUAN, Kaiwen ; BAI, Song ; XIE, Lingxi ; QI, Honggang ; HUANG, Qingming ; TIAN, Qi: CenterNet: Keypoint Triplets for Object Detection. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), S. 6568–6577
- [27] EVERINGHAM, Mark ; GOOL, Luc V. ; WILLIAMS, Christopher K. I. ; WINN, John M. ; ZISSERMAN, Andrew: The Pascal Visual Object Classes (VOC) Challenge. In: *International Journal of Computer Vision - IJCV* 88(2) (2010), S. 303–338
- [28] EVERINGHAM, Mark ; WINN, John: *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit*. http://host.robots.ox.ac.uk/pascal/VOC/voc2012/html/doc/devkit_doc.html, 2012. – Abgerufen am: 17.07.2020
- [29] FACEBOOK RESEARCH: *Detectron2 Model Zoo and Baselines*. https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md, o. J.. – Abgerufen am: 17.07.2020
- [30] FRAUNHOFER IML: *Data Driven Logistics*. https://www.iml.fraunhofer.de/de/abteilungen/b1/software_engineering.html, o. J.. – Abgerufen am: 17.07.2020

- [31] FRAUNHOFER-INSTITUT FÜR MATERIALFLUSS UND LOGISTIK IML: *Institutsprofil*. <https://www.iml.fraunhofer.de/de/unser-institut/Institutsprofil.html>, o. J.. – Abgerufen am: 17.07.2020
- [32] FU, Cheng-Yang ; LIU, Wei ; RANGA, Ananth ; TYAGI, Ambrish ; BERG, Alexander C.: DSSD : Deconvolutional Single Shot Detector. In: *CoRR* abs/1701.06659 (2017). <http://arxiv.org/abs/1701.06659>
- [33] GEISSLER, Otto ; OSTLER, Ulrike: *Was ist Inferenz?* <https://www.datacenter-insider.de/was-ist-inferenz-a-670450/>, 2017. – Abgerufen am: 17.07.2020
- [34] GERON, Aurelien: *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA : O'Reilly Media, 2017. – ISBN 978-1491962299
- [35] GHIASI, Golnaz ; LIN, Tsung-Yi ; LE, Quoc V.: DropBlock: A regularization method for convolutional networks. In: *CoRR* abs/1810.12890 (2018). <http://arxiv.org/abs/1810.12890>
- [36] GHIASI, Golnaz ; LIN, Tsung-Yi ; PANG, Ruoming ; LE, Quoc V.: NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection. In: *CoRR* abs/1904.07392 (2019). <http://arxiv.org/abs/1904.07392>
- [37] GIRSHICK, Ross B.: Fast R-CNN. In: *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), S. 1440–1448
- [38] GIRSHICK, Ross B. ; DONAHUE, Jeff ; DARRELL, Trevor ; MALIK, Jagannath: Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition* (2014), S. 580–587
- [39] GOODFELLOW, Ian ; BENGIO, Yoshua ; COURVILLE, Aaron: *Deep Learning*. MIT Press, 2016. – <http://www.deeplearningbook.org>
- [40] GOOGLE BRAIN AUTOML: *EfficientDet*. <https://github.com/google/automl/tree/master/efficientdet>, o. J.. – Abgerufen am: 17.07.2020
- [41] GOYAL, Kechit: *Deep Learning vs Neural Networks: Difference Between Deep Learning and Neural Networks*. <https://www.upgrad.com/blog/deep-learning-vs-neural-networks-difference-between-deep-learning-and-neural-networks/>, 2019. – Abgerufen am: 30.06.2020
- [42] GPUZOO.COM: *Theoretical performance comparison*. http://www.gpuzoo.com/Compare/NVIDIA_Tesla_V100_SMX2_vs__NVIDIA_Titan_V/, o. J.. – Abgerufen am: 17.07.2020

- [43] GREL, Tomasz: *Region of interest pooling explained*. <https://deepsense.ai/region-of-interest-pooling-explained/>, 2017. – Abgerufen am: 17.07.2020
- [44] HALE, Jeff: *Deep Learning Framework Power Scores 2018*. <https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a>, 2018. – Abgerufen am: 17.07.2020
- [45] HALE, Jeff: *Which Deep Learning Framework is Growing Fastest?* <https://www.kdnuggets.com/2019/05/which-deep-learning-framework-growing-fastest.html>, 2020. – Abgerufen am: 18.06.2020
- [46] HE, Kaiming ; GKIOXARI, Georgia ; DOLLÁR, Piotr ; GIRSHICK, Ross B.: Mask R-CNN. In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), S. 2980–2988
- [47] HE, Kaiming ; ZHANG, Xiangyu ; REN, Shaoqing ; SUN, Jian: Deep Residual Learning for Image Recognition. In: *CoRR* abs/1512.03385 (2015). <http://arxiv.org/abs/1512.03385>
- [48] HENDRYCKS, Dan ; DIETTERICH, Thomas: Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. In: *CoRR* abs/1807.01697 (2018). <http://arxiv.org/abs/1807.01697>
- [49] HIEN, Dang Ha T.: *A guide to receptive field arithmetic for Convolutional Neural Networks*. <https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807>, 2017. – Abgerufen am: 17.07.2020
- [50] HOWARD, Andrew G. ; ZHU, Menglong ; CHEN, Bo ; KALENICHENKO, Dmitry ; WANG, Weijun ; WEYAND, Tobias ; ANDRETTI, Marco ; ADAM, Hartwig: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. In: *CoRR* abs/1704.04861 (2017). <http://arxiv.org/abs/1704.04861>
- [51] HRNET: *High-resolution Networks for FCOS*. <https://github.com/HRNet/HRNet-FCOS>, o. J.. – Abgerufen am: 17.07.2020
- [52] HUANG, Jonathan ; RATHOD, Vivek ; SUN, Chen ; ZHU, Menglong ; KORATTIKARA, Anoop ; FATHI, Alireza ; FISCHER, Ian ; WOJNA, Zbigniew ; SONG, Yang ; GUADARRAMA, Sergio ; MURPHY, Kevin: Speed/accuracy trade-offs for modern convolutional object detectors. In: *CoRR* abs/1611.10012 (2016). <http://arxiv.org/abs/1611.10012>
- [53] HUI, Jonathan: *mAP (mean Average Precision) for Object Detection*. https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173, 2018. – Abgerufen am: 17.07.2020

- [54] JAMES, Gareth ; WITTEN, Daniela ; HASTIE, Trevor ; TIBSHIRANI, Robert: *An Introduction to Statistical Learning: with Applications in R.* Springer, 2013 <https://faculty.marshall.usc.edu/gareth-james/ISL/>
- [55] JIA, Yangqing ; DEVELOPER, Lead ; SHELHAMER, Evan: *Caffe.* <https://caffe.berkeleyvision.org/>, o. J.. – Abgerufen am: 17.07.2020
- [56] JIAO, Licheng ; ZHANG, Fan ; LIU, Fang ; YANG, Shuyuan ; LI, Lingling ; FENG, Zhixi ; QU, Rong: A Survey of Deep Learning-based Object Detection. In: *CoRR* abs/1907.09408 (2019). <http://arxiv.org/abs/1907.09408>
- [57] KIEFER, J. ; WOLFOWITZ, J.: Stochastic Estimation of the Maximum of a Regression Function. In: *Ann. Math. Statist.* 23 (1952), 09, Nr. 3, 462–466. <http://dx.doi.org/10.1214/aoms/1177729392>. – DOI 10.1214/aoms/1177729392
- [58] KONG, Tao ; SUN, Fuchun ; LIU, Huaping ; JIANG, Yuning ; SHI, Jianbo: FoveaBox: Beyond Anchor-based Object Detector. In: *CoRR* abs/1904.03797 (2019). <http://arxiv.org/abs/1904.03797>
- [59] KULIN, Merima ; KAZAZ, Tarik ; MOERMAN, Ingrid ; POORTER, Eli D.: A survey on Machine Learning-based Performance Improvement of Wireless Networks: PHY, MAC and Network layer. In: *ArXiv* abs/2001.04561 (2020)
- [60] LAO, Qicheng ; FEVENS, Thomas: Cell Phenotype Classification Using Deep Residual Network and Its Variants. In: *International Journal of Pattern Recognition and Artificial Intelligence* 33 (2019), Nr. 11, 1940017. <http://dx.doi.org/10.1142/S0218001419400172>. – DOI 10.1142/S0218001419400172
- [61] LAW, Hei ; DENG, Jia: CornerNet: Detecting Objects as Paired Keypoints. In: *CoRR* abs/1808.01244 (2018). <http://arxiv.org/abs/1808.01244>
- [62] LECUN, Yann ; BENGIO, Y. ; HINTON, Geoffrey: Deep Learning. In: *Nature* 521 (2015), 05, S. 436–44. <http://dx.doi.org/10.1038/nature14539>. – DOI 10.1038/nature14539
- [63] LEE, Kyoungmin ; CHOI, Jaeseok ; JEONG, Jisoo ; KWAK, Nojun: Residual Features and Unified Prediction Network for Single Stage Detection. In: *ArXiv* abs/1707.05031 (2017)
- [64] LEE, Youngwan ; PARK, Jongyoul: CenterMask : Real-Time Anchor-Free Instance Segmentation. In: *ArXiv* abs/1911.06667 (2019)
- [65] LI, Yanghao ; CHEN, Yuntao ; WANG, Naiyan ; ZHANG, Zhaoxiang: Scale-Aware Trident Networks for Object Detection. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), S. 6053–6062

- [66] LIN, Min ; CHEN, Qiang ; YAN, Shuicheng: Network In Network. In: *CoRR* abs/1312.4400 (2014)
- [67] LIN, Tsung-Yi ; DOLLÁR, Piotr ; GIRSHICK, Ross B. ; HE, Kaiming ; HARIHARAN, Bharath ; BELONGIE, Serge J.: Feature Pyramid Networks for Object Detection. In: *CoRR* abs/1612.03144 (2016). <http://arxiv.org/abs/1612.03144>
- [68] LIN, Tsung-Yi ; GOYAL, Priya ; GIRSHICK, Ross B. ; HE, Kaiming ; DOLLAR, Piotr: Focal Loss for Dense Object Detection. In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), S. 2999–3007
- [69] LIN, Tsung-Yi ; MAIRE, Michael ; BELONGIE, Serge J. ; BOURDEV, Lubomir D. ; GIRSHICK, Ross B. ; HAYS, James ; PERONA, Pietro ; RAMANAN, Deva ; DOLLÁR, Piotr ; ZITNICK, C. L.: Microsoft COCO: Common Objects in Context. In: *CoRR* abs/1405.0312 (2014). <http://arxiv.org/abs/1405.0312>
- [70] LIN, Tsung-Yi ; MAIRE, Michael ; BELONGIE, Serge J. ; HAYS, James ; PERONA, Pietro ; RAMANAN, Deva ; DOLLÁR, Piotr ; ZITNICK, C. L.: *COCO - Common Object in Context*. <http://cocodataset.org/#detection-eval>, o. J.. – Abgerufen am: 17.07.2020
- [71] LISA LAB: *theano*. <http://deeplearning.net/software/theano/>, o. J.. – Abgerufen am: 17.07.2020
- [72] LIU, Songtao ; HUANG, Di ; WANG, Yunhong: Receptive Field Block Net for Accurate and Fast Object Detection. In: *The European Conference on Computer Vision (ECCV)*, 2018
- [73] LIU, Songtao ; HUANG, Di ; WANG, Yunhong: Learning Spatial Fusion for Single-Shot Object Detection. In: *ArXiv* abs/1911.09516 (2019)
- [74] LIU, Wei ; ANGUELOV, Dragomir ; ERHAN, Dumitru ; SZEGEDY, Christian ; REED, Scott E. ; FU, Cheng-Yang ; BERG, Alexander C.: SSD: Single Shot MultiBox Detector. In: LEIBE, Bastian (Hrsg.) ; MATAS, Jiri (Hrsg.) ; SEBE, Nicu (Hrsg.) ; WELLING, Max (Hrsg.): *ECCV (1)* Bd. 9905, Springer, 2016 (Lecture Notes in Computer Science). – ISBN 978-3-319-46447-3, 21-37
- [75] LIU, Yu dong ; WANG, Yongtao ; WANG, Siwei ; LIANG, Tingting ; ZHAO, Qijie ; TANG, Zhi ; LING, Haibin: CBNet: A Novel Composite Backbone Network Architecture for Object Detection. In: *ArXiv* abs/1909.03625 (2019)
- [76] MASTERS, Dominic ; LUSCHI, Carlo: Revisiting Small Batch Training for Deep Neural Networks. In: *CoRR* abs/1804.07612 (2018). <http://arxiv.org/abs/1804.07612>

- [77] MC.AI: *Detection and segmentation.* <https://mc.ai/detection-and-segmentation/>, 2018. – Abgerufen am: 27.06.2020
- [78] MiFCOM: *Workstation Xeon Silver 4210 - Quadro P2200.* <https://www.mifcom.ch/workstation-xeon-silver-4210-quadro-p2200-id9426>, o. J.. – Abgerufen am: 17.07.2020
- [79] MINH, Toan D.: *Implementation EfficientDet: Scalable and Efficient Object Detection in PyTorch.* <https://github.com/toandaominh1997/EfficientDet.Pytorch.git>, o. J.. – Abgerufen am: 17.07.2020
- [80] MITCHELL, T ; BUCHANAN, B ; DEJONG, G ; DIETTERICH, T ; ROSENBLOOM, P ; WAIBEL, A: Machine Learning. In: *Annual Review of Computer Science* 4 (1990), Nr. 1, 417-433. <http://dx.doi.org/10.1146/annurev.cs.04.060190.002221>. – DOI 10.1146/annurev.cs.04.060190.002221
- [81] MITCHELL, Tom M.: *Machine Learning.* 1. USA : McGraw-Hill, Inc., 1997. – ISBN 0070428077
- [82] MOHRI, Mehryar ; ROSTAMIZADEH, Afshin ; TALWALKAR, Ameet: Foundations of Machine Learning. In: *Adaptive computation and machine learning*, 2012
- [83] MORIK, Katharina: *Vorlesung: Wissensentdeckung in Datenbanken SoSe 2008.* Vorlesung, 2008
- [84] MOTIONMINERS GMBH: *Motion-Mining - Automatisierte und anonymisierte Analyse und Optimierung manueller Prozesse.* <https://www.motionminers.com/>, o. J.. – Abgerufen am: 17.07.2020
- [85] NOTEBOOKCHECK.NET: *NVIDIA GeForce GTX TITAN X vs Tesla M40.* <https://technical.city/en/video/GeForce-GTX-TITAN-X-vs-Tesla-M40>, o. J.. – Abgerufen am: 17.07.2020
- [86] NVIDIA: *NVIDIA CUDA TOOLKIT - Release Notes for Windows, Linux, and Mac OS.* https://docs.nvidia.com/cuda/pdf/CUDA_Toolkit_Release_Notes.pdf, 2020. – Abgerufen am: 17.07.2020
- [87] NVIDIA CORPORATION: *About CUDA.* <https://developer.nvidia.com/cuda-zone>, o. J.. – Abgerufen am: 17.07.2020
- [88] NVIDIA CORPORATION: *NVIDIA cuDNN.* <https://developer.nvidia.com/cudnn>, o. J.. – Abgerufen am: 17.07.2020
- [89] NVIDIA CORPORATION: *Whitepaper - NVIDIA's Next Generation CUDA Compute Architecture: Fermi.* https://www.nvidia.com/content/PDF/fermi_white_papers/

- NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf, o. J.. – Abgerufen am: 17.07.2020
- [90] OLIVAS, Emilio S. ; GUERRERO, Jose David M. ; SOBER, Marcelino M. ; BENEDITO, Jose Rafael M. ; LOPEZ, Antonio Jose S.: *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques - 2 Volumes*. Hershey, PA : Information Science Reference - Imprint of: IGI Publishing, 2009. – ISBN 1605667668
- [91] OPENCV: *Computer Vision Annotation Tool (CVAT)*. <https://github.com/opencv/cvat>, o. J.. – Abgerufen am: 17.07.2020
- [92] OPENMMLAB: *Benchmark and Model Zoo*. https://github.com/open-mmlab/mmdetection/blob/master/docs/MODEL_ZOO.md, o. J.. – Abgerufen am: 17.07.2020
- [93] OPENMMLAB: *mmdetection - Benchmark and Model Zoo*. https://github.com/open-mmlab/mmdetection/blob/master/docs/MODEL_ZOO.md, o. J.. – Abgerufen am: 17.07.2020
- [94] OUAKNINE, Arthur: *Review of Deep Learning Algorithms for Object Detection*. <https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852>, 2018. – Abgerufen am: 28.06.2020
- [95] PAN, Sinno J. ; YANG, Qiang: A Survey on Transfer Learning. In: *IEEE Trans. on Knowl. and Data Eng.* 22 (2010), Oktober, Nr. 10, 1345–1359. <http://dx.doi.org/10.1109/TKDE.2009.191>. – DOI 10.1109/TKDE.2009.191. – ISSN 1041-4347
- [96] PANG, Jiangmiao ; CHEN, Kai ; SHI, Jianping ; FENG, Huajun ; OUYANG, Wanli ; LIN, Dahua: Libra R-CNN: Towards Balanced Learning for Object Detection. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), S. 821–830
- [97] PAPERS WITH CODE: *Papers with code*. <https://paperswithcode.com/>, o. J.. – Abgerufen am: 17.07.2020
- [98] PAPERS WITH CODE: *Papers With Code - Object Detection on COCO test-dev*. <https://paperswithcode.com/sota/object-detection-on-coco>, o. J.. – Abgerufen am: 17.07.2020
- [99] PASCAL2 - NETWORK OF EXCELLENCE ON PATTERN ANALYSIS, STATISTICAL MODELLING AND COMPUTATIONAL LEARNING: *PASCAL Visual Object Classes Challenge*. <http://host.robots.ox.ac.uk/pascal/VOC/>, o. J.. – Abgerufen am: 17.07.2020

- [100] PASZKE, Adam ; GROSS, Sam ; MASSA, Francisco ; LERER, Adam ; BRADBURY, James ; CHANAN, Gregory ; KILLEEN, Trevor ; LIN, Zeming ; GIMELSHEIN, Natalia ; ANTIGA, Luca ; DESMAISON, Alban ; KOPF, Andreas ; YANG, Edward ; DEVITO, Zachary ; RAISON, Martin ; TEJANI, Alykhan ; CHILAMKURT, Sasank ; STEINER, Benoit ; FANG, Lu ; BAI, Junjie ; CHINTALA, Soumith: PyTorch: An Imperative Style, High-Performance Deep Learning Library. Version: 2019. <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>. In: WALLACH, H. (Hrsg.) ; LAROCHELLE, H. (Hrsg.) ; BEYGELZIMER, A. (Hrsg.) ; ALCH-BUC, F. textquotesingle (Hrsg.) ; FOX, E. (Hrsg.) ; GARNETT, R. (Hrsg.): *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, 8024–8035
- [101] PATTERSON, Josh ; GIBSON, Adam: *Deep Learning: A Practitioner's Approach*. Beijing : O'Reilly, 2017 <https://www.safaribooksonline.com/library/view/deep-learning/9781491924570/>. – ISBN 978–1–4919–1425–0
- [102] POTTER, Rayan: *The Use Of Bounding Boxes in Image Annotation for Object Detection*. <https://medium.com/analytics/the-use-of-bounding-boxes-in-image-annotation-for-object-detection-6371711eabba>, 2019. – Abgerufen am: 17.07.2020
- [103] PRIJONO, Benny: *Student Notes: Convolutional Neural Networks (CNN) Introduction*. <https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/>, o. J.. – Abgerufen am: 17.07.2020
- [104] PROEVE, Paul-Louis: *An Introduction to different Types of Convolutions in Deep Learning*. <https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d>, 2017. – Abgerufen am: 17.07.2020
- [105] PYTORCH: *Most effective multiple model inference*. <https://discuss.pytorch.org/t/most-effective-multiple-model-inference/40666>, 2019. – Abgerufen am: 30.06.2020
- [106] PYTORCH: *PYTORCH MOBILE*. <https://pytorch.org/mobile/home/>, o. J.. – Abgerufen am: 17.07.2020
- [107] RAZA, Ali: *Type of convolutions: Deformable and Transformable Convolution*. <https://towardsdatascience.com/type-of-convolutions-deformable-and-transformable-convolution-1f660571eb91>, 2019. – Abgerufen am: 17.07.2020
- [108] REDMON, Joseph ; DIVVALA, Santosh K. ; GIRSHICK, Ross B. ; FARHADI, Ali: You Only Look Once: Unified, Real-Time Object Detection. In: *CoRR* abs/1506.02640 (2015). <http://arxiv.org/abs/1506.02640>

- [109] REDMON, Joseph ; FARHADI, Ali: YOLO9000: Better, Faster, Stronger. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), S. 6517–6525
- [110] REDMON, Joseph ; FARHADI, Ali: YOLOv3: An Incremental Improvement. In: *CoRR* abs/1804.02767 (2018). <http://arxiv.org/abs/1804.02767>
- [111] REITERMANOVA, Zuzana: Data splitting. In: *WDS Proceedings of Contributed Papers* Bd. 10, 2010, S. 31–36
- [112] REN, Shaoqing ; HE, Kaiming ; GIRSHICK, Ross B. ; SUN, Jian: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: *CoRR* abs/1506.01497 (2015). <http://arxiv.org/abs/1506.01497>
- [113] RUINMESSI: *Receptive Field Block Net for Accurate and Fast Object Detection, ECCV 2018*. <https://github.com/ruinmessi/RFBNet>, o. J.. – Abgerufen am: 17.07.2020
- [114] RUINMESSI: *yolov3 with mobilenet v2 and ASFF*. <https://github.com/ruinmessi/ASFF>, o. J.. – Abgerufen am: 17.07.2020
- [115] RUMELHART, David E. ; HINTON, Geoffrey E. ; WILLIAMS, Ronald J.: Learning Internal Representations by Error Propagation. In: RUMELHART, David E. (Hrsg.) ; MCCLELLAND, James L. (Hrsg.): *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*. Cambridge, MA : MIT Press, 1986, S. 318–362
- [116] RUSSAKOVSKY, Olga ; DENG, Jia ; SU, Hao ; KRAUSE, Jonathan ; SATHEESH, Sanjeev ; MA, Sean ; HUANG, Zhiheng ; KARPATHY, Andrej ; KHOSLA, Aditya ; BERNSTEIN, Michael S. ; BERG, Alexander C. ; LI, Fei-Fei: ImageNet Large Scale Visual Recognition Challenge. In: *CoRR* abs/1409.0575 (2014). <http://arxiv.org/abs/1409.0575>
- [117] RUSSELL, Stuart ; NORVIG, Peter: *Artificial Intelligence: A Modern Approach*. 3rd. USA : Prentice Hall Press, 2009. – ISBN 0136042597
- [118] SCIKIT-LEARN: 3.1. *Cross-validation: evaluating estimator performance*. https://scikit-learn.org/stable/modules/cross_validation.html, o. J.. – Abgerufen am: 27.06.2020
- [119] SERMANET, Pierre ; EIGEN, David ; ZHANG, Xiang ; MATHIEU, Michaël ; FERGUS, Rob ; LECUN, Yann: OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In: *CoRR* abs/1312.6229 (2014)
- [120] SHEARER, Colin: The CRISP-DM Model: The New Blueprint for Data Mining. In: *JOURNAL OF DATA WAREHOUSING* 5 (2000), Nr. 4

- [121] SHETTY, Suyash: Application of Convolutional Neural Network for Image Classification on Pascal VOC Challenge 2012 dataset. In: *CoRR* abs/1607.03785 (2016). <http://arxiv.org/abs/1607.03785>
- [122] SHI, Shaohuai ; WANG, Qiang ; XU, Pengfei ; CHU, Xiaowen: Benchmarking State-of-the-Art Deep Learning Software Tools. In: *2016 7th International Conference on Cloud Computing and Big Data (CCBD)* (2016), S. 99–104
- [123] SIGNATRIX GMBH: (*Pretrained weights provided*) *EfficientDet: Scalable and Efficient Object Detection implementation by Signatrix GmbH*. <https://github.com/signatrix/efficientdet>, o. J.. – Abgerufen am: 17.07.2020
- [124] SIMONYAN, Karen ; ZISSERMAN, Andrew: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *CoRR* abs/1409.1556 (2014). <http://arxiv.org/abs/1409.1556>
- [125] SINGH, Bharat ; DAVIS, Larry S.: An Analysis of Scale Invariance in Object Detection - SNIP. In: *CoRR* abs/1711.08189 (2017). <http://arxiv.org/abs/1711.08189>
- [126] SINGH, Bharat ; NAJIBI, Mahyar ; DAVIS, Larry S.: SNIPER: Efficient Multi-Scale Training. In: *NeurIPS*, 2018
- [127] SINGH, Seema: *Understanding the Bias-Variance Tradeoff*. <https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229>, 2018. – Abgerufen am: 17.07.2020
- [128] SMART VISION EUROPE: *Phases of the CRISP-DM reference model*. <http://crisp-dm.eu/reference-model>, 2015. – Abgerufen am: 17.07.2020
- [129] STANDFORT VISION AND LEARNING LAB: *Imagenet - Large Scale Visual Recognition Challenge (ILSVRC)*. <http://www.image-net.org/challenges/LSVRC/2017>, 2020
- [130] STANFORD VISION AND LEARNING LAB: *CS231n Convolutional Neural Networks for Visual Recognition*. <https://cs231n.github.io/convolutional-networks/>, o. J.. – Abgerufen am: 17.07.2020
- [131] SUN, Ke ; ZHAO, Yang ; JIANG, Borui ; CHENG, Tianheng ; XIAO, Bin ; LIU, Dong ; MU, Yadong ; WANG, Xinggang ; LIU, Wenyu ; WANG, Jingdong: High-Resolution Representations for Labeling Pixels and Regions. In: *ArXiv* abs/1904.04514 (2019)
- [132] SZEGEDY, C. ; WEI LIU ; YANGQING JIA ; SERMANET, P. ; REED, S. ; ANGUELOV, D. ; ERHAN, D. ; VANHOUCKE, V. ; RABINOVICH, A.: Going deeper with convolutions. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. – ISSN 1063–6919, 1-9

- [133] TAN, Mingxing ; PANG, Ruoming ; LE, Quoc V.: EfficientDet: Scalable and Efficient Object Detection. In: *ArXiv* abs/1911.09070 (2019)
- [134] TECHMAN ROBOT: *Basler Industrial Camera*. <https://www.tm-robot.com.cn/en/product/basler-industrial-camera/>, o. J.. – Abgerufen am: 17.07.2020
- [135] TENSORFLOW: *Deploy machine learning models on mobile and IoT devices*. <https://www.tensorflow.org/lite>, o. J.. – Abgerufen am: 17.07.2020
- [136] THE APACHE SOFTWARE FOUNDATION - APACHE MXNET: *mxnet - A FLEXIBLE AND EFFICIENT LIBRARY FOR DEEP LEARNING*. <https://mxnet.apache.org/>, o. J.. – Abgerufen am: 17.07.2020
- [137] THE LINUX FOUNDATION: *ONNX - Open Neural Network Exchange*. <https://onnx.ai/>, o. J.. – Abgerufen am: 17.07.2020
- [138] TIAN, Zhi ; SHEN, Chunhua ; CHEN, Hao ; HE, Tong: FCOS: Fully Convolutional One-Stage Object Detection. In: *CoRR* abs/1904.01355 (2019). <http://arxiv.org/abs/1904.01355>
- [139] TSANG, Sik-Ho: *Review: DilatedNet - Dilated Convolution (Semantic Segmentation)*. <https://towardsdatascience.com/review-dilated-convolution-segmentation-9d5a5bd768f5>, 2018. – Abgerufen am: 17.07.2020
- [140] TSANG, Sik-Ho: *Review: DCN / DCNv1 — Deformable Convolutional Networks, 2nd Runner Up in 2017 COCO Detection (Object Detection)*. <https://towardsdatascience.com/review-dcn-deformable-convolutional-networks-2nd-runner-up-in-2017-coco-detection-object-14e488efce44>, 2019. – Abgerufen am: 17.07.2020
- [141] TURING, Alan M.: Computing machinery and intelligence. In: *Mind* 49 (1950), Nr. 236, 433-460. <https://www.csee.umbc.edu/courses/471/papers/turing.pdf>
- [142] TUSIMPLE: *NAS-FPN*. <https://github.com/TuSimple/simpledet/tree/master/models/NASFPN>, o. J.. – Abgerufen am: 17.07.2020
- [143] TYCHSEN-SMITH, Lachlan ; PETERSSON, Lars: DeNet: Scalable Real-time Object Detection with Directed Sparse Sampling. In: *CoRR* abs/1703.10295 (2017). <http://arxiv.org/abs/1703.10295>
- [144] TYCHSEN-SMITH, Lachlan ; PETERSSON, Lars: Improving Object Localization with Fitness NMS and Bounded IoU Loss. In: *CoRR* abs/1711.00164 (2017). <http://arxiv.org/abs/1711.00164>

- [145] ULTRALYTICS LLC: *YOLOv5*. <https://github.com/ultralytics/yolov5>, o. J.. – Abgerufen am: 17.07.2020
- [146] URBANN, Oliver ; CAMPHAUSEN, Simon ; MOOS, Arne ; SCHWARZ, Ingmar ; KERNER, Sören ; OTTEN, Maximilian: *A C Code Generator for Fast Inference and Simple Deployment of Convolutional Neural Networks on Resource Constrained Systems*. 2020
- [147] VOGEL-HEUSER, Birgit (Hrsg.) ; BAUERNHANSL, Thomas (Hrsg.) ; HOMPEL, Michael ten (Hrsg.): *Handbuch Industrie 4.0*. (to-be-published) 3. Springer
- [148] WANDELL, Brian ; WINAWER, Jonathan: Computational neuroimaging and population receptive fields. In: *Trends in Cognitive Sciences* 19 (2015), 04. <http://dx.doi.org/10.1016/j.tics.2015.03.009>. – DOI 10.1016/j.tics.2015.03.009
- [149] WANG, Tiancai ; ANWER, Rao M. ; CHOLAKKAL, Hisham ; KHAN, Fahad S. ; PANG, Yanwei ; SHAO, Ling: Learning Rich Features at High-Speed for Single-Shot Object Detection. In: *The IEEE International Conference on Computer Vision (ICCV)*, 2019
- [150] XIE, Saining ; GIRSHICK, Ross B. ; DOLLÁR, Piotr ; TU, Zhuowen ; HE, Kai-ming: Aggregated Residual Transformations for Deep Neural Networks. In: *CoRR* abs/1611.05431 (2016). <http://arxiv.org/abs/1611.05431>
- [151] YOO, Jin H. ; PARK, Seong H. ; CHOI, Jun W.: ScarfNet: Multi-scale Features with Deeply Fused and Redistributed Semantics for Enhanced Object Detection. In: *ArXiv* abs/1908.00328 (2019)
- [152] YOUNGWANLEE: *CenterMask2 on top of detectron2, in CVPR 2020*. <https://github.com/youngwanLEE/centermask2>, o. J.. – Abgerufen am: 17.07.2020
- [153] YU, Fisher ; KOLTUN, Vladlen: Multi-Scale Context Aggregation by Dilated Convolutions. In: *CoRR* abs/1511.07122 (2016)
- [154] ZHANG, Du ; TSAI, Jeffrey J. P.: *Advances in Machine Learning Applications in Software Engineering*. USA : IGI Global, 2007. – ISBN 1591409411
- [155] ZHANG, Hongyi ; CISSÉ, Moustapha ; DAUPHIN, Yann N. ; LOPEZ-PAZ, David: mixup: Beyond Empirical Risk Minimization. In: *CoRR* abs/1710.09412 (2017). <http://arxiv.org/abs/1710.09412>
- [156] ZHANG, Shifeng: *Bridging the Gap Between Anchor-based and Anchor-free Detection via Adaptive Training Sample Selection, CVPR, Oral, 2020*. <https://github.com/sfzhang15/ATSS>, o. J.. – Abgerufen am: 17.07.2020

- [157] ZHANG, Shifeng ; CHI, Cheng ; YAO, Yongqiang ; LEI, Zhen ; LI, Stan Z.: Bridging the Gap Between Anchor-based and Anchor-free Detection via Adaptive Training Sample Selection. In: *ArXiv* abs/1912.02424 (2019)
- [158] ZHANG, Shifeng ; WEN, Longyin ; BIAN, Xiao ; LEI, Zhen ; LI, Stan Z.: Single-Shot Refinement Neural Network for Object Detection. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), S. 4203–4212
- [159] ZHAO, Qijie ; SHENG, Tao ; WANG, Yongtao ; TANG, Zhi ; CHEN, Ying ; CAI, Ling ; LING, Haibin: M2Det: A Single-Shot Object Detector based on Multi-Level Feature Pyramid Network. In: *CoRR* abs/1811.04533 (2018). <http://arxiv.org/abs/1811.04533>
- [160] ZHAO, Zhong-Qiu ; ZHENG, Peng ; XU, Shou-tao ; WU, Xindong: Object Detection with Deep Learning: A Review. In: *CoRR* abs/1807.05511 (2018). <http://arxiv.org/abs/1807.05511>
- [161] ZHI, Tian: *FCOS: Fully Convolutional One-Stage Object Detection*. <https://github.com/tianzhi0549/FCOS>, o. J.. – Abgerufen am: 17.07.2020
- [162] ZHOU, Xingyi ; ZHUO, Jiacheng ; KRÄHENBÜHL, Philipp: Bottom-Up Object Detection by Grouping Extreme and Center Points. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), S. 850–859
- [163] ZHU, Chenchen ; HE, Yihui ; SAVVIDES, Marios: Feature Selective Anchor-Free Module for Single-Shot Object Detection. In: *CoRR* abs/1903.00621 (2019). <http://arxiv.org/abs/1903.00621>
- [164] ZHU, Xizhou ; HU, Han ; LIN, Stephen ; DAI, Jifeng: Deformable ConvNets V2: More Deformable, Better Results. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), S. 9300–9308
- [165] ZHU, Yousong ; ZHAO, Chaoyang ; WANG, Jinqiao ; ZHAO, Xu ; WU, Yi ; LU, Hanqing: CoupleNet: Coupling Global Structure with Local Parts for Object Detection. In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), S. 4146–4154
- [166] ZOU, Zhengxia ; SHI, Zhenwei ; GUO, Yuhong ; YE, Jieping: Object Detection in 20 Years: A Survey. In: *CoRR* abs/1905.05055 (2019). <http://arxiv.org/abs/1905.05055>

Eidesstattliche Versicherung (Affidavit)

Otten, Maximilian

156796

Name, Vorname
(Last name, first name)

Matrikelnr.
(Enrollment number)

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem folgenden Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present Bachelor's/Master's* thesis with the following title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution.

Titel der Bachelor-/Masterarbeit*:
(Title of the Bachelor's/ Master's* thesis):

Benchmarking von Deep-Learning-Algorithmen zur Detektion von Objekten im Kontext der Intralogistik

*Nichtzutreffendes bitte streichen
(Please choose the appropriate)

Dortmund, 20.07.2020

Ort, Datum
(Place, date)

Unterschrift
(Signature)



Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:

Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to €50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, section 63, subsection 5 of the North Rhine-Westphalia Higher Education Act (*Hochschulgesetz*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:**

Dortmund, 20.07.2020

Ort, Datum
(Place, date)

Unterschrift
(Signature)



**Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.