

Integrierte Konzepte tiefer Neuronaler Netze zur monokularen Informationsgewinnung im Autonomen Fahrzeug

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Michael Weber

aus Karlsruhe

Tag der mündlichen Prüfung: 10. Mai 2022
Erster Gutachter: Prof. Dr. J. Marius Zöllner
Zweiter Gutachter: Prof. Dr. Rainer Stiefelhagen

Vorwort

Diese Arbeit entstand während meiner Zeit als wissenschaftlicher Mitarbeiter am FZI Forschungszentrum Informatik in der Abteilung Technisch Kognitive Systeme (TKS). Zum Gelingen der Arbeit haben viele Kollegen und Studenten am FZI maßgeblich beigetragen.

Zunächst möchte ich allen Kollegen der Abteilungen TKS und IDS sowie den KIT-Kollegen der Partnerlehrstühle am KIT für die vielfältige Unterstützung danken. Ein besonderer Dank gilt an der Stelle Ralf Kohlhaas für die aufbauenden Gespräche, Karam Daaboul für die unvergesslichen Momente auf Klausurtagungen und gemeinsame nächtliche Fitness-Sessions sowie Stefan Orf, der in schwierigen Situationen bereitwillig als Kummerkasten fungierte und dessen Motivation uns bis auf den höchsten Gipfel der Erde führte. Vielen Dank auch an Tobias Fleck, Daniel Bogdoll und Nico Lambing für das Korrekturlesen meiner Ausarbeitung und wichtige letzte Kommentare. Nicht zu vergessen sind natürlich auch die ehemaligen Mitarbeiter von TKS, die unter anderem durch den Aufbau von Versuchsträgern und Softwarebibliotheken die Voraussetzungen für diese Arbeit geschaffen haben. Insbesondere gilt mein Dank Dennis Nienhüser, ohne dessen Arbeiten zur Detektion von Verkehrselementen und der Sammlung entsprechender Daten Teile dieser Arbeit schlicht nicht möglich gewesen wären.

Ebenfalls danke ich den vielen Studenten, die mich als wissenschaftliche Hilfskraft oder im Rahmen einer Abschlussarbeit unterstützt haben. Insbesondere die tiefgreifenden fachlichen Diskussionen waren vor allem zu Beginn dieser Arbeit von unschätzbarem Wert. Unvergessen sind auch die zahlreichen nächtlichen Coding- und Debug-Sessions. Hervorheben möchte ich an dieser Stelle Peter Wolf, Martin Thoma, Marvin Teichmann, Matthias Huber, Christof Wendenius und Michael Fürst, welche im Rahmen ihrer Abschlussarbeiten einen wertvollen und direkten Beitrag zu dieser Arbeit leisteten. Ein besonderer Dank gilt auch unserer Teamassistenz Sonja Göttl, die mir bei organisatorischen Herausforderungen sowie im Kontakt mit Professoren und der Verwaltung des FZI eine unverzichtbare Hilfe war und stets ein offenes Ohr für meine kleinen und großen Probleme hatte.

Meinem Doktorvater Prof. J. Marius Zöllner danke ich für seine Bereitschaft, meine Promotion zu begleiten, den Freiraum bei der Ausgestaltung des thematischen Forschungsschwerpunktes sowie die konstruktiv kritischen Gespräche. Bei Prof. Rainer Stiefelhagen bedanke ich mich insbesondere dafür, dass er in einer für mich schwierigen Situation ohne zu zögern die Zweitbetreuung dieser Arbeit an der Fakultät für Informatik übernommen hat. Ebenfalls möchte ich

Prof. Ralf Reussner und Prof. Jörn Müller-Quade dafür danken, dass sie bereitwillig die Aufgabe der Prüfer in meiner Prüfung übernahmen. Bei Prof. Raoul Zöllner möchte ich mich für die wertvollen Kommentare und Gespräche zur Erwartungshaltung an Motivation und Struktur wissenschaftlicher Arbeiten bedanken. Herrn Griesbaum danke ich für die hilfsbereite und freundliche technische Betreuung im Vorfeld meiner mündlichen Prüfung und seine beruhigende Ausstrahlung am Tag der Prüfung.

Ein großer Dank gilt meinen Eltern, die mir durch ihre Unterstützung ein Studium und die anschließende Promotion ermöglicht haben. Zuletzt möchte ich meiner Frau Anne danken für ihre unendliche Geduld und ihr Verständnis für all die Entbehrungen, welche die Erstellung dieser Arbeit mit sich brachten. Ihr danke ich zusätzlich dafür, dass sie neben ihrer bedingungslosen Unterstützung auch die wiederkehrende Korrektur dieser Ausarbeitung übernommen und dabei eine regelrechte Obsession für das Auffinden von Rechtschreibfehlern entwickelt hat.

Karlsruhe im Mai 2022

Michael Weber

Kurzfassung

Tiefe Neuronale Netze im Allgemeinen und Convolutional Neural Networks (CNNs) im Speziellen konnten in den letzten Jahren in den Bereichen Maschinelles Sehen und Bildverarbeitung in verschiedensten Domänen große Erfolge erzielen. Herausforderungen bestehen dabei vor allem in der Anpassung der Methoden an domänenspezifische Tasks und die Reduktion des im Normalfall recht hohen Ressourcenbedarfs, insbesondere für mobile Anwendungen wie das Autonome Fahren. Dort werden CNNs zur Bildverarbeitung meist für die Gewinnung relevanter Informationen über die Umgebung aus monokularen Kameras verwendet. Hierzu werden auf einem Kamerabild verschiedene Algorithmen zur Lösung unterschiedlicher Tasks ausgeführt.

Im Rahmen dieser Arbeit wird mit dem MultiNet-Ansatz ein Konzept zur integrierten Bearbeitung verschiedener Tasks in einem gemeinsamen CNN-Modell vorgestellt. Der Ressourcenbedarf kann so im Vergleich zu einer getrennten Ausführung separater Modelle bei gleichbleibender Qualität der Ergebnisse deutlich reduziert werden. Zusätzlich wird ein Verfahren vorgestellt, welches bei ebenfalls gleichbleibender Ergebnisqualität durch eine Kombination der angepassten Methoden Pruning und Knowledge Distillation den Ressourcenverbrauch eines CNN-Modells signifikant reduzieren kann.

Für die Domäne des Autonomen Fahrens werden CNN-Architekturen zur allgemeinen Objektdetektion an die Anforderungen der domänenspezifischen Detektion von Objekten im Fahrzeugumfeld angepasst. Hierbei findet eine getrennte Betrachtung von statischen und dynamischen Verkehrsobjekten statt. Zur Lösung der Herausforderungen bei der Detektion von statischen Verkehrsobjekten wie Ampeln oder Verkehrsschildern wird ein Ansatz zur hierarchischen Detektion gleichartiger Objekte vorgestellt. Für die Detektion von dynamischen Verkehrsobjekten wie anderen Verkehrsteilnehmern wird ein Ansatz zur direkten 3D Detektion von Objekten in einem Kamerabild eingeführt.

Derartige Ansätze offenbaren durch ihre Komplexität allerdings eine Herausforderung beim Training vieler CNN-Modelle. Der Fortschritt und die Konvergenz des Trainingsprozesses werden in wesentlichen Teilen durch die Zusammensetzung der verwendeten Trainingsdaten bedingt. Um den Trainingsprozess mit den jeweils verfügbaren Trainingsdaten bestmöglich durchführen zu können, wird eine dynamische Fehlerfunktion vorgestellt, welche die einzelnen Datenpunkte angepasst an deren aktuelle Schwierigkeit im Trainingsprozess automatisch gewichtet. Eine Evaluation auf öffentlich verfügbaren Datensätzen wird für alle im Rahmen dieser Arbeit vorgestellten Konzepte durchgeführt.

Abstract

Deep neural networks in general and convolutional neural networks (CNNs) in particular have achieved great success in machine vision and image processing in a wide variety of domains in recent years. Challenges are mainly the adaptation of existing methods for domain-specific tasks and the reduction of their typically quite high resource requirements, especially for mobile applications like autonomous driving. In this domain, CNNs for image processing are usually used to extract relevant information of the environment from monocular cameras. This involves executing various algorithms on a particular camera image to solve different tasks.

In the context of this work, with the MultiNet approach, a concept for the integrated processing of different tasks in a common CNN model is presented. The resource requirements can thus be significantly reduced compared to a separate execution of independent models while the quality of the results remains unchanged. In addition, a method is presented which can significantly reduce the resource consumption of a CNN model by combining adapted methods of pruning and knowledge distillation while still maintaining the same quality of results.

For the domain of autonomous driving, CNN architectures for general object detection are adapted to the requirements of domain-specific object detection in the vehicle environment. In this context, a separate consideration of static and dynamic traffic objects is performed. To solve the challenges of detecting static traffic objects such as traffic lights or traffic signs, an approach for hierarchical detection of objects of similar types is presented and evaluated for the example of traffic light detection. For the detection of dynamic traffic objects like other road users, an approach for direct 3D object detection in a camera image is introduced.

However, such approaches due to their complexity reveal a challenge in training many CNN models. The progress and convergence of the training process are largely dependent on the composition of the training data used. In order to perform the training process in the best possible way based on the available training data, a dynamic error function is presented which automatically weights the individual data points according to their current difficulty in the training process. An evaluation on publicly available data sets is performed for all concepts presented.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	2
1.2	Wissenschaftlicher Beitrag	4
1.3	Aufbau der Arbeit	5
2	Grundlagen	7
2.1	Architekturen zur Merkmalsextraktion	7
2.2	Relevante Messgrößen in CNNs	11
2.2.1	Parameter eines CNN	13
2.2.2	Operationen	14
3	Statische Verkehrsobjekte: 2D Objektdetektion	15
3.1	Problembeschreibung	16
3.2	Verwandte Arbeiten	23
3.2.1	2D Objektdetektion mit CNNs	24
3.2.2	Detektion statischer Verkehrsobjekte	30
3.3	Methodik	33
3.3.1	Objektrepräsentation	34
3.3.2	CNN-Architektur	35
3.3.3	Hierarchische Detektion	40
3.3.4	Nachverarbeitung der Netzausgabe	47
3.4	Experimente und Evaluation	48
3.4.1	Datensätze	49
3.4.2	Evaluationsmetriken	55
3.4.3	Ampeldetektion mit CNNs – DeepTLR	59
3.4.4	Hierarchische Ampeldetektion – HDTLR	61
3.4.5	Klassengruppierung zur Hierarchieerzeugung	68
3.5	Diskussion	69
4	Dynamische Verkehrsobjekte: 3D Objektdetektion	71
4.1	Problembeschreibung	72
4.2	Verwandte Arbeiten	74
4.3	Methodik	77
4.3.1	Objektrepräsentation	78
4.3.2	Netzarchitektur	80
4.3.3	Fehlerfunktion	85
4.3.4	Nachverarbeitung der Netzausgabe	87

4.4	Experimente und Evaluation	89
4.4.1	Datensätze	89
4.4.2	Evaluationsmetriken	90
4.4.3	3D Objektdetektion	91
4.5	Diskussion	94
5	Umfeldererkennung mittels MultiTask-Netzen	95
5.1	Problembeschreibung	96
5.2	Verwandte Arbeiten	99
5.3	Konzept	103
5.3.1	Architektur	103
5.3.2	Lernstrategien	106
5.4	Experimente und Evaluation	108
5.4.1	Datensätze	109
5.4.2	Beispielarchitektur zur Evaluation	113
5.4.3	Evaluationsmetriken	119
5.4.4	Evaluation der MultiNet-Architektur	121
5.5	Diskussion	124
6	Gewichtung einzelner Datenpunkte im Trainingsprozess	127
6.1	Problembeschreibung	128
6.2	Verwandte Arbeiten	131
6.3	Methodik	135
6.4	Experimente und Evaluation	141
6.4.1	Datensätze	141
6.4.2	Evaluationsmetriken	142
6.4.3	Experiment 1: Vergleich mit Focal Loss	143
6.4.4	Experiment 2: 3D Objektdetektion	144
6.5	Diskussion	146
7	Optimierung trainierter CNN-Modelle	147
7.1	Problembeschreibung	148
7.2	Verwandte Arbeiten	149
7.2.1	Parameterrepräsentation	150
7.2.2	Pruning	151
7.2.3	Verwendung reduzierter Architekturen	152
7.2.4	Knowledge Distillation	154
7.3	Methodik	155
7.3.1	Netzoptimierung	156
7.3.2	Wissensübertragung	158
7.4	Experimente und Evaluation	159
7.4.1	Evaluationsmetriken	161
7.4.2	Reduktion der Netzarchitektur	161
7.4.3	Wissensübertragung	165
7.5	Diskussion	166

8 Zusammenfassung und Ausblick	169
8.1 Zusammenfassung der Beiträge	169
8.2 Ausblick	171
Verzeichnisse	173
Abkürzungsverzeichnis	175
Abbildungsverzeichnis	179
Tabellenverzeichnis	183
Eigene Veröffentlichungen	185
Studentische Arbeiten	187
Literaturverzeichnis	191

1 Einleitung

Die Techniken der tiefen Neuronalen Netze und insbesondere die [Convolutional Neural Networks \(CNNs\)](#) verleihen seit Jahren dem Forschungsgebiet des Maschinellen Lernens einen enormen Schub. Insbesondere im Bereich des Maschinellen Sehens und der Bildverarbeitung wurden mit Hilfe dieser Techniken in den letzten Jahren große Erfolge erzielt. Zwischenzeitlich werden verschiedenste auf Bildeingaben arbeitende Perzeptionsalgorithmen zur Klassifikation, Segmentierung oder der Detektion von Objekten fast ausschließlich auf Basis von [CNNs](#) realisiert.

In vielen Domänen werden auf einem Eingabebild verschiedenste Perzeptionsalgorithmen ausgeführt, um vielfältige Informationen gewinnen zu können. Dies geschieht meist in Form einer getrennten Ausführung von unterschiedlichen, für eine jeweilige Aufgabe spezialisierten Algorithmen auf demselben Eingabebild. Allerdings sind die durchzuführenden Berechnungen der einzelnen Algorithmen sehr ressourcenintensiv, womit diese für mobile Anwendungen wenig geeignet sind. Insbesondere in den Domänen Robotik oder Autonomes Fahren stehen zur Ausführung lediglich eine begrenzte Menge an Recheneinheiten zur Verfügung. Betrachtet man zusätzlich mobile Anwendungen wie Assistenzfunktionen auf Smartphones, ist die mögliche Anzahl an Rechenoperationen zusätzlich durch eine maximal zur Verfügung stehende Energiemenge begrenzt.

So ist es sinnvoll, die [CNN](#)-Architekturen aktueller Perzeptionsalgorithmen auf eine reduzierte Anzahl benötigter Rechenoperationen zu optimieren. Hierzu bietet sich die Untersuchung des MultiTask-Ansatzes an, also der parallelen Ausführung mehrerer Aufgaben (engl. Tasks) in einem gemeinsamen Neuronalen Netz. Da in verschiedenen Domänen teils spezifische Anforderungen an die einzelnen Algorithmen gestellt werden, müssen hier jeweils Konzepte zur Adaption an einzelne Tasks in der aktuellen Domäne erforscht werden. Ein MultiTask-Ansatz muss an dieser Stelle Raum für die Ausführung derartiger Adaptionen bieten.

In der vorliegenden Arbeit wird die Domäne des Autonomen Fahrens als Grundlage für die Untersuchungen von rechenzeitreduzierten Architekturen wie auch domänenspezifischen Anpassungen von [CNN](#)-Architekturen verwendet. Die im Titel dieser Arbeit – *Integrierte Konzepte tiefer Neuronaler Netze zur monokularen Informationsgewinnung im Autonomen Fahrzeug* – genannten Begriffe werden nachfolgend zur Erlangung eines gemeinsamen Verständnisses genauer erläutert.

1 Einleitung

Begriffsbildung „Integrierte Konzepte“ Zur Lösung verschiedener Tasks werden neben domänenspezifischen Konzepten auch Methoden benötigt, diese Konzepte ressourcenreduziert in gemeinsamen Modellen integriert auszuführen.

Begriffsbildung „Tiefes Neuronales Netz“ Ein *tiefes Neuronales Netz* bezeichnet ein Neuronales Netz mit, je nach verwendeter Definition, mindestens zwei nichtlinearen Aktivierungsfunktionen oder zwei verborgenen Zwischenschichten. Die im Rahmen dieser Arbeit verwendeten CNN-Architekturen erfüllen hierbei stets beide Definitionen. Bei tiefen Neuronalen Netzen wird zwischen Architekturen und Modellen unterschieden, wobei im Kontext dieser Arbeit der Begriff Architektur stets die Definition der Schichtfolge sowie die Parametrierung der einzelnen Schichten bezeichnet. Ein Modell entspricht einer durch einen Trainingsprozess entstandenen, konkreten Ausprägung einer Architektur.

Begriffsbildung „Monokulare Informationsgewinnung“ Die *Informationsgewinnung* bezeichnet die Gewinnung relevanter Informationen aus der durch Sensorik abtastbaren Umgebung eines Autonomen Fahrzeuges. *Monokular*, eine Komposition aus dem griechischen monos für „ein“ und dem lateinischen oculus für „Auge“, steht für die Verwendung einer einzelnen Kamera als Zielsensor für die betrachtete Algorithmik.

Begriffsbildung „Autonomes Fahrzeug“ Ein *Autonomes Fahrzeug* bezeichnet ein Fahrzeug, welches unabhängig von einem konkreten Autonomiegrad unter anderem basierend auf Sensoreingaben Fahrentscheidungen trifft. Der in dieser Arbeit betrachtete Anwendungsfall ist hierbei die Teilnahme am öffentlichen Straßenverkehr.

1.1 Motivation

Zur Realisierung autonomer Fahrfunktionen im öffentlichen Straßenverkehr ist die Erfassung von Informationen über das Fahrzeugumfeld von zentraler Bedeutung. So ist ein Autonomes Fahrzeug mit einer Vielzahl unterschiedlicher Sensoren in verschiedenen Ausprägungen ausgestattet. Eine besondere Rolle nehmen hierbei kamerabasierte Systeme ein. Diese sind in der Lage, visuelle Informationen aus der Umgebung zu extrahieren, welche durch keinen anderen Sensortyp erfasst werden können. So ist beispielsweise der Zustand einer Ampel oder der Inhalt eines Verkehrszeichens von keinem anderen Sensor erfassbar. Generell können anhand eines Kamerasensors Informationen wie die Position regulatorischer Objekte und anderer Verkehrsteilnehmer im Fahrzeugumfeld, die Segmentierung befahrbarer Bereiche als auch allgemeinere Informationen zur aktuellen Umgebung gewonnen werden. Eine beispielhafte Szene ist in Abbildung 1.1 dargestellt.

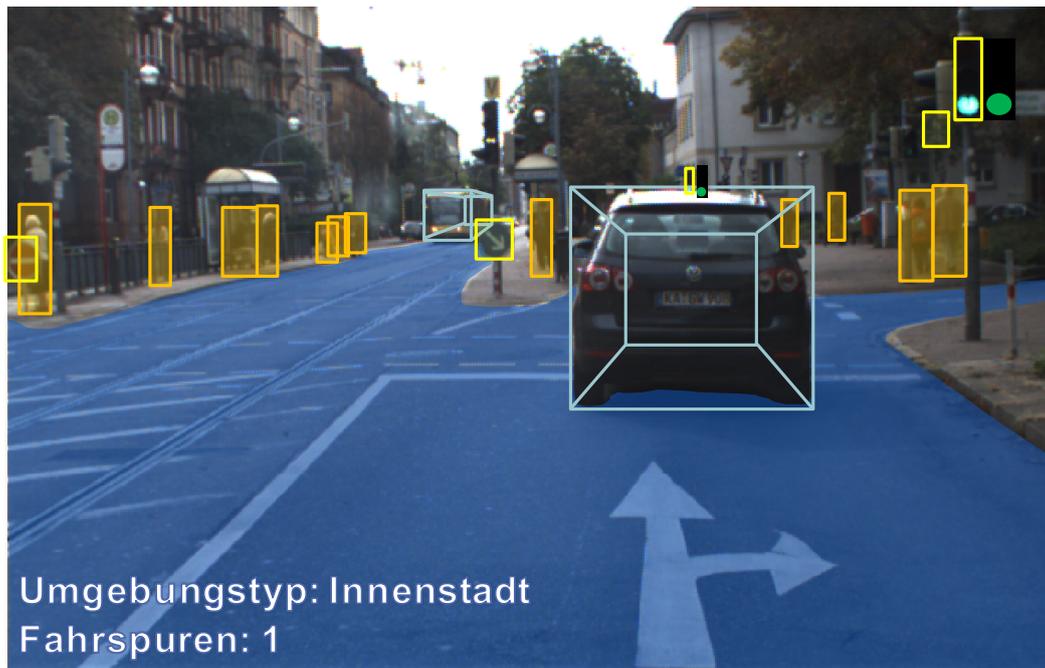


Abbildung 1.1: Beispielhafte Betrachtung einer Verkehrsszene durch eine monokulare Kamera. Aus dem Bild können verschiedene Informationen zu befahrbaren Bereichen (blau), anderen Verkehrsteilnehmern (cyan, orange), Verkehrszeichen und Ampeln (gelb) oder generelle Informationen zur aktuellen Umgebung gewonnen werden. Im Falle der Ampeln kann auch eine Bestimmung des Zustandes sowie möglicher Piktogramme durchgeführt werden.

Basierend auf diesen Informationen können durch aggregierende Algorithmen zeitliche Zusammenhänge hergestellt, das Verhalten anderer Verkehrsteilnehmer analysiert und präzisiert sowie letztendlich eine Fahrentscheidung getroffen werden. Aktuelle Perzeptionsalgorithmen der Bildverarbeitung bieten an dieser Stelle bereits eine breite Palette an Anwendungsmöglichkeiten zur Gewinnung von Informationen aus einem Kamerabild. Es existieren bereits direkt anwendbare [CNN](#)-Architekturen zur Klassifikation einer Szene oder zur pixelgenauen Segmentierung, also der Zuweisung einer Klasse zu jedem einzelnen Pixel in einem Kamerabild.

Im Bereich der Detektion von Objekten können die vorhandenen, generellen Ansätze nur bedingt auf die domänenspezifischen Problemstellungen angewandt werden. So müssen beim Autonomen Fahren im Wesentlichen zwei verschiedene Gruppen von Objekten mit unterschiedlichen Anforderungen erkannt werden. Für regulatorische Objekte wie Verkehrszeichen und Ampeln muss lediglich eine ungefähre Position bestimmt werden, da deren Aussagekraft im Normalfall nicht durch eine exakte Positionierung bestimmt ist. Herausfordernd sind an dieser Stelle allerdings die vorwiegend geringe Größe der Objekte sowie vielfach eine Ähnlichkeit zwischen verschiedenen Klassen.

1 Einleitung

Dynamische Objekte im Fahrzeugumfeld wie andere Verkehrsteilnehmer hingegen müssen möglichst positionsgenau detektiert werden. Um an dieser Stelle den Anforderungen an die Genauigkeit der Detektion anderer Verkehrsteilnehmer zu genügen, ist meist eine Fusion der Ergebnisse mehrerer Sensorarten notwendig. Ein Kamerasensor kann an dieser Stelle wertvolle visuelle Informationen beitragen. Allerdings müssen zu diesem Zweck die Detektionen der Objekte durch die verschiedenen Sensoren im selben Bezugssystem durchgeführt werden. Hierzu müssen auch kamerabasierte Detektionen in einem dreidimensionalen Fahrzeugkoordinatensystem durchgeführt werden.

Letztendlich werden einige Perzeptionsalgorithmen parallel auf einem einzelnen Kamerabild ausgeführt. Aufgrund der begrenzten und mit entsprechenden Kosten verbundenen Ressourcen zur Ausführung derartiger Algorithmen müssen die einzelnen Algorithmen auf ihr Optimierungspotential untersucht werden. Neben den bereits erwähnten MultiTask-Ansätzen sind an dieser Stelle weitere Verfahren zur Komprimierung von taskspezifischen Architekturbestandteilen von Interesse. Auch derartige Verfahren zur Rechenzeitoptimierung werden im Rahmen dieser Arbeit vorgestellt.

1.2 Wissenschaftlicher Beitrag

Ziel dieser Arbeit ist die Erforschung domänenspezifischer Architekturen für Perzeptionsaufgaben im Autonomen Fahrzeug sowie domänenübergreifender Ansätze zur Rechenzeitoptimierung verschiedener Tasks durch gemeinsame Ausführung und Reduktion einzelner Architekturkomponenten.

Zur Gewinnung von Umgebungsinformationen aus Bildern monokularer Kameras wird hierbei der Schwerpunkt auf die Detektion unterschiedlicher Objektklassen gelegt. Eine erste Forschungsfrage ist hierbei, wie sowohl regulatorische Objekte als auch andere Verkehrsteilnehmer entsprechend der objektspezifischen Anforderungen durch CNN-basierte Architekturen detektiert werden können. Hierzu werden im Rahmen dieser Arbeit in Kapitel 3 mit HDTLR ein Ansatz zur hierarchischen Detektion optisch ähnlicher Objekte und in Kapitel 4 mit Direct3D ein Ansatz zur direkten 3D Detektion anderer Verkehrsteilnehmer vorgestellt.

Die zweite Forschungsfrage schließt direkt an die Anforderungen der Direct3D-Architektur an. Der Trainingsprozess einer derart komplexen Architektur mit verschiedenen miteinander in Zusammenhang stehenden Ausgaben und unterschiedlicher Verteilung der Annotationsklassen in den Trainingsdaten bedarf hierbei zusätzlicher Konzepte. So stellt sich die Frage, wie durch eine strategische Fehlerfunktion ein derart komplexer Trainingsprozess zielführend auf wesentliche Trainingsdaten fokussiert werden kann. Im Rahmen dieser Arbeit wird hierzu in Kapitel 6 der Automated Focal Loss als Fehlerfunktion zur adaptiven Gewichtung einzelner Trainingsdaten eingeführt.

Da der Ressourcenverbrauch von CNNs domänenübergreifend ein wichtiges Thema ist, wird an dritter Stelle die Forschungsfrage behandelt, wie Architekturen zur Lösung verschiedener Tasks in einer gemeinsamen Architektur verwendet und die taskspezifischen Anteile zusätzlich komprimiert werden können. Zur Ausführung verschiedener Modelle in einer MultiTask-Architektur mit gemeinsamem Teilnetz wird in Kapitel 5 der MultiNet-Ansatz vorgestellt. Die Komprimierung der taskspezifischen Teilnetze kann anhand des in Kapitel 7 eingeführten, integrierten Konzeptes aus Pruning und Knowledge Distillation durchgeführt werden.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit besteht, wie in Abbildung 1.2 dargestellt, aus fünf Themenblöcken, welche jeweils eigene Kapitel darstellen. Vorangestellt ist diesen inhaltlichen Kapiteln zunächst das aktuelle Kapitel 1 mit einer kurzen Einleitung in die Themenfelder der tiefen Neuronalen Netze und MultiTask-Algorithmen sowie des Anwendungsfelds des Autonomen Fahrens. Die Grundlagen zu den in dieser Arbeit vorgestellten Konzepten sind in Kapitel 2 dargestellt. Hier werden insbesondere verschiedene Aspekte und Bausteine der in dieser Arbeit verwendeten CNNs beschrieben.

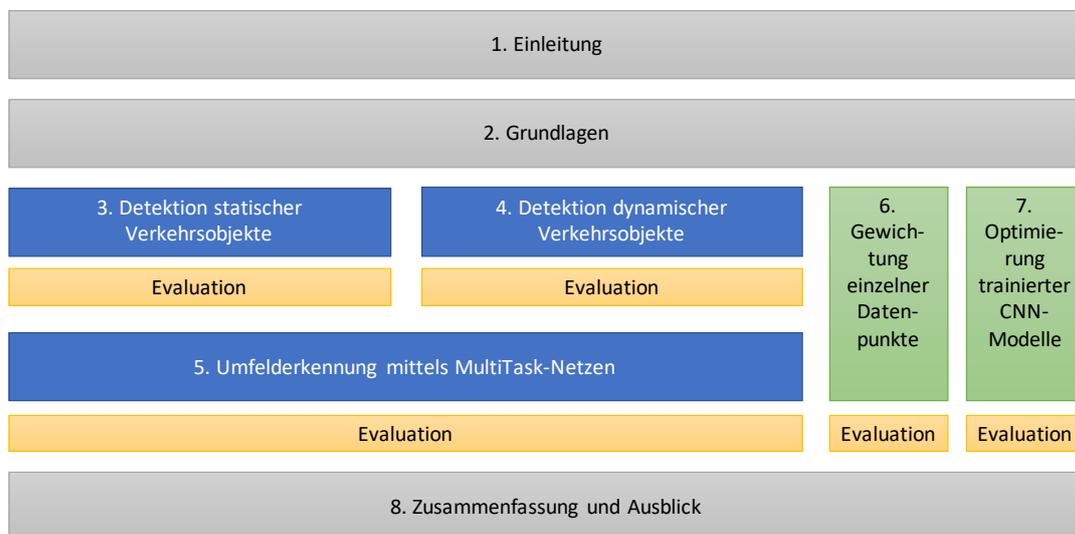


Abbildung 1.2: Struktur und Aufbau der vorliegenden Arbeit.

Der thematische Inhalt dieser Arbeit wird angeführt durch zwei Themenblöcke zur Realisierung von Perzeptionsalgorithmen mittels CNNs. In Kapitel 3 werden zunächst Methoden und Architekturen zur Detektion statischer Verkehrsobjekte, wie beispielsweise Ampeln oder Verkehrsschilder, vorgestellt. Daran anschließend wird in Kapitel 4 mit den dynamischen Verkehrsobjekten die zweite große

1 Einleitung

Gruppe an Verkehrsobjekten, zu denen im Wesentlichen andere Verkehrsteilnehmer im Fahrzeugumfeld gehören, im Kontext von Algorithmik und Methodik betrachtet.

Methoden zur Zusammenführung und Kombination einzelner Perzeptionsalgorithmen in einer gemeinsamen, sogenannten MultiNet-Architektur werden in Kapitel 5 vorgestellt. Neben den grundsätzlichen Konzepten zur Erstellung einer derartigen Architektur wird auch ein konkretes Modell zur kombinierten Szenenklassifikation, Straßensegmentierung und Fahrzeugdetektion vorgestellt.

Die zwei verbleibenden Themenblöcke stehen übergreifend über die bisher angesprochenen Architekturen und können als zusätzliche Methoden für die bisherigen Themenblöcke betrachtet werden. Zunächst werden in Kapitel 6 Methoden vorgestellt, wie mit einer adaptiven Gewichtung der Fehlerfunktion und dem damit einhergehenden Fokus auf eine Teilmenge der Trainingsdaten der Trainingsprozess eines tiefen Neuronalen Netzwerks optimiert und beschleunigt werden kann. In Kapitel 7 wird schließlich eine Strategie vorgestellt, durch welche ein bereits trainiertes CNN-Modell bezüglich des Speicherbedarfs und der Laufzeit bei konstanter oder steigender Güte optimiert werden kann. Mit einer Zusammenfassung der vorgestellten Konzepte schließt Kapitel 8 die Arbeit und gibt einen Ausblick auf mögliche künftige Forschungsthemen.

2 Grundlagen

Nachfolgend werden im Rahmen dieses Kapitels einige Grundlagen zum Verständnis der vorliegenden Arbeit vermittelt. So wird zunächst in Abschnitt 2.1 ein Überblick über verschiedene auf einem [Convolutional Neural Network \(CNN\)](#) aufbauende Encoder-Netzwerke zur Extraktion von Features gegeben. Da die relevanten Messgrößen eines [CNN](#) ein zentraler Bestandteil bei der Optimierung von [CNN](#)-Modellen sind, werden diese in Abschnitt 2.2 näher betrachtet. Hierbei werden insbesondere die Parameter und Operationen eines [CNN](#) betrachtet und Formeln zu deren Berechnung vorgestellt.

2.1 Architekturen zur Merkmalsextraktion

Ein [CNN](#) zur Lösung einer Perzeptionsaufgabe ist meist nach dem Encoder-Decoder Architekturprinzip aufgebaut. In dem Decoder genannten, zweiten Abschnitt des Netzes wird die eigentliche Detektion, Klassifikation oder Segmentierung durchgeführt. Diese basiert auf den Merkmalen (engl. Features), die im Encoder aus den Eingabedaten extrahiert werden. Der Encoder-Abschnitt zu Beginn eines [CNN](#) wird deshalb auch als Feature Extraction bezeichnet. In klassischen Verarbeitungsketten wird hierfür meist der deutsche Begriff Merkmalsextraktion verwendet, als Bezeichnung für den Abschnitt im [CNN](#) ist allerdings der englische Ausdruck deutlich verbreiteter. Im Gegensatz zu klassischen, meist von Hand erstellten, Merkmalsextraktoren ist das Erlernen dieser Features im [CNN](#) ein Hauptbestandteil des Trainingsprozesses.

Allerdings kommt auch dieser Prozess häufig nicht ohne von Hand erstellte Komponenten aus. So wird noch immer die genaue Architektur, also die konkrete Art und Anordnung der Schichten, des Encoder meist manuell festgelegt. Dabei besteht immer die Herausforderung, eine Architektur zu finden, die möglichst gute Ergebnisse für den jeweiligen Task ermöglicht. Da der Trainingsprozess eines Encoders sehr zeit- und ressourcenaufwendig ist, haben sich im Laufe der letzten Jahre verschiedene Methoden zum Entwurf einzelner Architekturen entwickelt. So werden Architekturen zur Extraktion von Features im Normalfall zunächst für den vergleichsweise wenig komplexen Task der Bildklassifikation entwickelt.

Die qualitative Leistungsfähigkeit einzelner Architekturen wird dann in Wettbewerben wie der [ImageNet Large Scale Visual Recognition Challenge \(ILSVRC\)](#) auf standardisierten Datensätzen verglichen. Mit gegebener Annahme, dass sich Features zur Bildklassifikation ebenso für andere Perzeptionsaufgaben wie die

2 Grundlagen

Detektion oder die Segmentierung eignen, werden die Encoder-Architekturen für entsprechende Netze zur Lösung dieser Tasks übernommen. So kommt diesen Klassifikationsnetzen auch für Architekturen zur Lösung anderer Tasks eine wichtige Bedeutung zu. Nachfolgend werden die bekanntesten Ansätze für Netzarchitekturen zur Extraktion von Features in chronologischer Reihenfolge nochmals detaillierter betrachtet. Eine Übersicht über verschiedene Ansätze und deren Charakteristika ist in Tabelle 2.1 zu finden.

AlexNet demonstrierte erstmals die aktuelle Dominanz von **CNNs** für einen Task der Bildverarbeitung und wurde nach seinem Erfinder Alex Krizhevsky benannt [144]. Das Netz ist klassisch sequentiell aufgebaut mit wechselnden Convolution- und Pooling-Schichten im ersten Netzteil sowie mehreren Fully-Connected-Schichten zur Klassifikation am Ende. Im Gegensatz zu den ersten **CNNs** wie LeNet [147] wird die alternierende Abfolge von Pooling und Convolution im Mittelteil bereits durch einen Block mit mehreren aufeinander folgenden Convolution-Schichten aufgebrochen. Die relativ großen Filterkerne in den ersten Schichten führen bei diesem Netz allerdings zu einer großen Zahl an durchzuführenden Rechenoperationen. Da zum Erscheinungszeitpunkt die Recheneinheiten bzgl. ihrer Leistungsfähigkeit noch stark beschränkt waren, enthält die Architektur ein Konzept zur verteilten Ausführung des Modells auf mehreren Recheneinheiten. So werden in den Convolution-Schichten sogenannte *Group Convolutions* eingeführt, bei denen die einzelnen Filterkerne in Gruppen eingeteilt und jeweils nur auf die Eingaben der eigenen Gruppe angewendet werden. So entstehen voneinander unabhängige Stränge, die auf verschiedenen Einheiten unabhängig berechnet werden können und bei denen nur an definierten Stellen in der Architektur Informationen ausgetauscht werden. Eine Optimierung von AlexNet stellt das ZFNet [244] dar, welches durch kleinere Filterkerne und reduzierte Schrittweiten die Ausgabequalität des trainierten Modells weiter steigern konnte.

Network In Network (NIN) [151] ist der Namensgeber eines neuen Designprinzips für **CNNs**. In dieser Netzarchitektur werden erstmals innerhalb einer Convolution-Schicht weitere Schichten als sogenanntes *Mikronetzwerk* eingefügt. Dieses Designprinzip wurde von vielen nachfolgenden Architekturen wie den Inception-Netzen übernommen.

GoogLeNet [222] auch bekannt unter dem Namen Inception-v1 ist eine Netzarchitektur von Google und gewann die **ILSVRC** im Jahr 2014. Der Name Inception stellt dabei eine Anspielung auf die Verwendung des *Network In Network*-Prinzips in Verbindung mit dem *we need to go deeper* Meme aus dem Hollywoodfilm *Inception* dar [222]. Designziel war eine effiziente Architektur mit weniger Parametern und einer geringen Zahl an benötigten Rechenoperationen. Allerdings wurde zum Gewinn der **ILSVRC** ein Ensemble aus sieben verschiedenen trainierten Modellen dieser Architektur verwendet, was diesen Effizienzge-

2.1 Architekturen zur Merkmalsextraktion

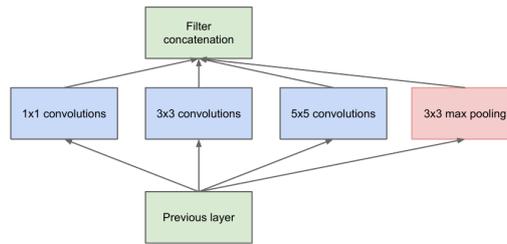


Abbildung 2.1: Inception-Modul aus dem GoogLeNet. Bildquelle: [222]

danken etwas relativiert. Aufbauend auf zwei initialen Convolution-Schichten besteht GoogLeNet im Wesentlichen aus neun sogenannten Inception-Modulen, deren Aufbau in Abbildung 2.1 dargestellt ist. Diese Module sind sequentiell angeordnet, enthalten selbst allerdings mehrere parallele Pfade mit Convolution-Schichten, die sich jeweils in der Größe ihrer Filterkerne unterscheiden. Ziel dieser Inception-Module ist die Nachbildung von dünnbesetzten Filterkernen, da eine effiziente Nutzung von Filtern dieser Art mit aktueller Trainings- und Ausführungshardware nicht möglich ist. Die Konzepte der parallelen Pfade und des modularen Aufbaus sind hierbei neue Designprinzipien. Das Network In Network-Prinzip wird in GoogLeNet verwendet, um die Anzahl der Feature Maps und damit auch die Tiefe der nachfolgenden Filterkerne in den einzelnen Pfaden der Module zu reduzieren.

VGG [212] benannt nach der Visual Geometry Group der University of Oxford war im Jahr 2014 die nach GoogLeNet zweitplatzierte Architektur bei der ILSVRC. Im Gegensatz zu GoogLeNet wird, analog zu AlexNet, eine rein sequentielle Architektur verwendet. Das Netz besteht im Vergleich zu AlexNet aus einer deutlich größeren Anzahl Schichten, ist aufgrund der sequentiellen Architektur also deutlich tiefer. Hierbei ist zu beachten, dass verschiedene Konfigurationen mit unterschiedlicher Tiefe existieren. Der VGG-Encoder enthält hierbei zwischen 8 und 16 Convolution-Schichten. In allen Konfigurationen sind 4 Pooling-Schichten enthalten, womit ein konstanter Reduktionsfaktor im Netz garantiert wird. In der alternierenden Schichtfolge wird nun zwischen den Pooling-Schichten ein Block aus mehreren Convolution-Schichten mit 3×3 Filterkernen verwendet. Dies folgt den Erkenntnissen, dass mehrere Convolution-Schichten mit kleinen Kernen im Vergleich zu einer Schicht mit größeren Kernen einen deutlich geringeren Rechenaufwand bei gleicher Qualität besitzen [212].

ResNet [114] ist benannt nach dessen inhärentem Prinzip, das Lernen auf das Residuum zwischen Eingabe und Zielfunktion zu beschränken. Um dieses Residuum berechnen zu können, werden sogenannte *Residual Connections* oder *Skip Connections* eingeführt, die einem Bypass vorbei an ganzen Blöcken von Schichten entsprechen. Diese Blöcke sind analog zu GoogLeNet fest definiert. Je nach Tiefe des vorgesehenen Netzes werden zwei verschiedene Arten von Blöcken eingesetzt. Für Netze mit einer Tiefe von maximal 50 Schichten werden sogenannte

2 Grundlagen

Residual Blocks eingesetzt. Ab einer Tiefe von 50 Schichten werden *Bottleneck Residual Blocks* verwendet. Die einzelnen Blöcke bestehen hierbei aus einer Sequenz von 2-3 Convolution-Schichten. Das Ziel dieses Designs im Allgemeinen und der Skip Connections im Speziellen ist es, tiefere Netze zu ermöglichen. Dies wird durch das Umgehen des Vanishing Gradient Problems möglich, da über Skip Connections beliebig tiefe Schichten direkt erreicht werden können. Für ResNet existieren vordefinierte Architekturen mit 18, 34, 50, 101 und 152 Schichten.

Darknet [186] wurde im Wesentlichen als Feature Extractor für den Objektdetektor YOLO [187] ab Version 2 entwickelt. Die Architektur des Netzes folgt einem ähnlichen Ansatz wie die VGG-Architektur. Das zunächst verwendete Darknet-19 nutzt einen rein sequentiellen Ansatz und besteht hauptsächlich aus Convolution-Schichten mit 3×3 Filterkernen. Die Reduktion der Auflösung im Netz wird durch mehrere MaxPooling-Schichten erreicht. In der Klassifikationsschicht wird analog zu NiN ein Global Average Pooling verwendet. Die Darknet-Architektur wird in YOLOv3 [188] analog zu ResNet [114] um Residual Connections erweitert. Da das Netz durch diesen Schritt aus 53 Schichten besteht, wird für diese Architektur der Name Darknet-53 verwendet. In YOLOv4 [46] wird Darknet um Konzepte des CSPNet erweitert und somit fortan als CSPDarknet-53 bezeichnet.

Inception-v3 [225] wurde als Name für die erfolgreichste Variation der in derselben Veröffentlichung dargestellten Inception-v2-Architektur gewählt. Es ist damit der eigentliche Nachfolger von GoogLeNet. Im Vergleich zu der ersten Inception-Architektur werden die teilweise in den Inception-Blöcken genutzten Convolution-Schichten mit 5×5 Filterkernen durch mehrere Schichten mit 3×3 Filterkernen ersetzt. Zusätzlich werden neue Inception-Blöcke mit asymmetrischen Filterkernen in den Convolution-Schichten definiert.

DenseNet [120] ist eine Kurzform für *Densely Connected Neural Network*. Die Architektur besteht aus einer Aneinanderreihung sogenannter *DenseBlocks*. Ein Hauptmerkmal und Namensgeber dieser Blöcke ist die Verbindung jeder Schicht in einem Block mit den Ausgaben aller vorherigen Schichten. Mit diesem Design soll die Anzahl der Parameter im Netz reduziert sowie das Vanishing Gradient Problem vermindert werden. Die Dimensionsreduktion im Netz findet in Pooling-Schichten mit einer vorgeschalteten Convolution-Schicht statt, welche sich zwischen den einzelnen DenseBlocks befinden. Diese Zwischenschichten werden als Transition-Schichten bezeichnet.

Inception-ResNet [221] beschreibt einen Weg zur Kombination des Ansatzes von ResNet, das von Facebook stammt, mit dem Inception-Ansatz von Google. So werden für Inception-ResNet drei neue Varianten der Inception-Blöcke vorgestellt, welche analog zu den bisherigen Inception-Architekturen jeweils mehrfach

aneinandergereiht eine sequentielle Architektur realisieren. Das erste Modul des Netzes, auch Stamm genannt, wird abweichend davon aus einer Reihe normaler Convolution-Schichten gebildet.

Xception [62] führt die Depthwise Separable Convolutions [211] in die allgemeinen Feature Extraktoren ein. Es wird gezeigt, dass diese Schichten den Inception-Blöcken in einer extremen Ausprägung sehr ähnlich sind. Aus dieser Erkenntnis heraus wird eine Architektur vorgestellt, welche sich an GoogLeNet orientiert, die Inception-Blöcke allerdings durch Depthwise Separable Convolutions ersetzt.

ResNeXt [239] stammt von Facebook und kombiniert die Architekturen von ResNet und der Inception-Familie ebenso, wie bereits die Inception-ResNet-Architektur von Google. So werden analog zu GoogLeNet und seinen Inception-Blöcken Blöcke mit parallelen Strängen definiert. Diese verfügen allerdings, im Gegensatz zu den Inception-Blöcken aus Abbildung 2.1 mit ihren 4 parallelen Pfaden, über je nach konkreter Ausprägung deutlich mehr parallele Pfade. So werden in der beschriebenen Architektur meist 32 parallele Pfade verwendet. Hierbei wird allerdings je Schicht eine deutlich geringere Anzahl an Filterkernen verwendet. So enthält ein ResNeXt-Block im Schnitt sogar weniger Parameter als ein ResNet-Block. Die finalen Architekturen sind analog zu den ResNet-Architekturen definiert, es wurden lediglich die ResNet- durch ResNeXt-Blöcke ersetzt.

CSPNet [235] steht für **Cross Stage Partial Network** und beschreibt eine an die DenseNet-Architektur angelehnte Architektur. Hierbei werden die DenseBlocks und deren sequentielle Anordnung beibehalten. Allerdings wird zusätzlich ein zu dieser Architektur paralleler, weiterer Pfad eingeführt. So werden nach einer ersten Convolution-Schicht die ausgegebenen Feature Maps aufgespalten. Eine Hälfte wird in das DenseNet-Teilnetz gegeben, während die andere Hälfte direkt mit dessen Ausgabe konkateniert wird.

2.2 Relevante Messgrößen in CNNs

Neben qualitativen Aussagen zu den Ergebnissen eines CNN müssen für dessen Nutzung in einer konkreten Anwendung meist weitere für die Zielumgebung notwendige Messgrößen bekannt sein. Bei der Planung müssen hierfür im Wesentlichen die Größe des fertigen CNN-Modells sowie der zu erwartende Rechenaufwand bekannt sein. Die Größe des Modells wird zur Dimensionierung des Speicherplatzes auf der Zielhardware sowie zur Planung von Übertragungen von Modellupdates über Internetverbindungen benötigt, falls dies durch die Anwendung vorgesehen ist. Angegeben wird die Modellgröße durch die Anzahl

2 Grundlagen

Name	Quelle	Version	Erscheinungsjahr	Parameter	FLOPs
AlexNet	[144]		2012	2,5 Mio	26,2 Mrd
ZFNet	[244]		2013	3,7 Mio	45,2 Mrd
GoogLeNet	[224]		2014	5,6 Mio	55,4 Mrd
VGG	[212]	11	2014	9,2 Mio	275,8 Mrd
		13	2014	9,4 Mio	412,0 Mrd
		16	2014	14,7 Mio	565,2 Mrd
		19	2014	20,0 Mio	718,0 Mrd
ResNet	[112]	18	2015	11,2 Mio	67,2 Mrd
		34	2015	21,3 Mio	135,6 Mrd
		50	2015	23,5 Mio	152,0 Mrd
		101	2015	42,5 Mio	288,8 Mrd
		152	2015	58,1 Mio	426,0 Mrd
Inception-v3	[225]		2015	21,8 Mio	131,2 Mrd
MobileNet_v2	[201]		2018	2,2 Mio	11,8 Mrd
MobileNet_v3	[117]	small	2019	0,9 Mio	2,2 Mrd
		large	2019	3,0 Mio	8,2 Mrd
EfficientNet	[227]	B0	2019	4,0 Mio	0,6 Mrd
		B1	2019	6,5 Mio	0,8 Mrd
		B2	2019	7,7 Mio	0,8 Mrd
		B3	2019	10,7 Mio	1,0 Mrd
		B4	2019	17,5 Mio	1,2 Mrd
		B5	2019	27,9 Mio	1,8 Mrd
		B6	2019	40,7 Mio	2,2 Mrd
B7	2019	63,7 Mio	3,0 Mrd		

Tabelle 2.1: Vergleich verschiedener Netzarchitekturen zur Extraktion von Features. Die Zahlen beziehen sich auf die Encoder der genannten Netze. Da der zur Klassifikation verwendete Standard-Decoder für diese Arbeit nicht relevant ist, wurde dieser nicht beachtet. Für die Berechnung der Rechenoperationen wurde eine Eingabebildgröße von 1.280×720 Pixeln angenommen, da viele aktuelle Trainingsdatensätze im Anwendungsfeld des Autonomen Fahrens Bilddaten in dieser Größe bereitstellen.

der verwendeten Parameter sowie deren Datentyp. Der zu erwartende Rechenaufwand wird vorwiegend anhand der durchgeführten Rechenoperationen angegeben. Diese Messgröße ist ausschlaggebend für die Dimensionierung der Recheneinheiten auf der Zielhardware sowie einer Abschätzung der Laufzeit des Modells auf dieser Hardware.

2.2.1 Parameter eines CNN

Der Speicherbedarf eines CNN in Form der gespeicherten Parameter ergibt sich direkt aus der Definition von dessen Netzarchitektur – also der Anzahl und Art der Schichten sowie deren Parametrierung. Den weitaus größten Teil der gespeicherten Parameter stellen hierbei die Eingangsgewichte der verwendeten Neuronen in den Fully-Connected-Schichten sowie der Filterkerne in den Convolution-Schichten dar. Werden in einer Netzarchitektur Transposed-Convolution-Schichten zur Vergrößerung der Ausgabeauflösung verwendet, sind auch hier einige Parameter in Filterkernen vorhanden.

Für eine Convolution-Schicht lässt sich die Anzahl der zu speichernden Parameter in den Filterkernen anhand der Filtergröße (k_x, k_y) , deren Tiefe (Channels) k_c sowie der Anzahl der verwendeten Filter berechnen:

$$W_c = k_x \times k_y \times k_c \times n_k \quad (2.1)$$

Zusätzlich wird pro Filterkern noch ein sogenannter Bias als Parameter gespeichert:

$$B_c = n_k \quad (2.2)$$

Somit lässt sich die Anzahl der insgesamt in einer Convolution-Schicht vorhandenen Parameter wie folgt berechnen:

$$P_c = W_c + B_c \quad (2.3)$$

Hieraus wird deutlich, dass die Parameter des Bias in Relation zu den Parametern in den Filterkernen vernachlässigbar sind. Die Parameter von Transposed-Convolution-Schichten, auch Deconvolution-Schichten genannt, lassen sich analog zu den Convolution-Schichten bestimmen. Allerdings ist die Besonderheit zu beachten, dass der Bias für diese Schichten optional ist.

Zur Berechnung der Parameter einer Fully-Connected-Schicht müssen zunächst die Gewichte W_{fc} zwischen den Neuronen der aktuellen Schicht N_{fc} und den Neuronen der vorhergehenden Schicht N_{fc-1} betrachtet werden:

$$W_{fc} = N_{fc-1} \times N_{fc} \quad (2.4)$$

Analog zur Convolution-Schicht wird pro Neuron wieder ein Bias als Parameter gespeichert:

$$B_{fc} = N_{fc} \quad (2.5)$$

Die Anzahl der insgesamt in einer Fully-Connected-Schicht vorhandenen Parameter lässt sich damit wie folgt berechnen:

$$P_c = W_c + B_c \quad (2.6)$$

Es ist grundsätzlich möglich, dass Parameter in weiteren Netzelementen wie der Batch Normalization [124] oder in trainierbaren Aktivierungsfunktionen wie Parametric ReLU [112] oder Parametric ELU [230] vorhanden sind. Im Vergleich zu den Parametern in den Convolution- und Fully-Connected-Schichten haben diese auf die Größe des CNN lediglich einen untergeordneten Einfluss.

2.2.2 Operationen

Für die Berechnung des Rechenaufwandes und somit, abhängig von der verwendeten Zielhardware, auch der Laufzeit muss die Anzahl der benötigten Rechenoperationen für jede Schicht des Netzes bekannt sein. Im Gegensatz zu der Anzahl der Parameter ist die Anzahl der Rechenoperationen nicht bei allen Netzarten allein durch die Architektur definiert. Dies betrifft im Wesentlichen **CNNs**, bei denen die Eingabegröße nicht bereits durch die Architektur bestimmt wird. In einem **Fully Convolutional Network (FCN)** [158] beispielsweise wird die Anzahl der Parameter bereits mit der Definition der Netzarchitektur festgelegt. Die Anzahl der benötigten Rechenoperationen kann allerdings erst dann bestimmt werden, wenn auch die Eingabegröße der Bilddaten bekannt ist.

Die Anzahl der Rechenoperationen in einem **CNN** wird meist in Form von **Floating Point Operationen (FLOPs)** angegeben. Im Zusammenhang mit der bei Ausführungshardware zur Quantifizierung der Rechenkapazität verwendeten Maßzahl **Floating Point Operationen pro Sekunde (FLOPS)** kann so eine Abschätzung der Laufzeit eines **CNN** auf einer dedizierten Ausführungseinheit erfolgen. Eine genaue Berechnung ist vorab in der Regel deutlich aufwendiger, da hierbei die konkrete Organisation der Recheneinheiten auf der Ausführungshardware ein entscheidender Faktor ist. Zusätzlich müssen Implementierungsdetails des genutzten Software-Frameworks sowie der verwendeten Bibliotheken wie z. B. **cuDNN** [61] berücksichtigt werden. Für eine generelle Abschätzung des Rechenaufwandes eines **CNN** unabhängig von Hardwarespezifikation und Implementierungsdetails sind **FLOPs** allerdings weit verbreitet [188][238][114][227].

Die mit weitem Abstand meisten **FLOPs** befinden sich bei **CNNs** zur Klassifikation in den Convolution-Schichten. Eine kurze Analyse der VGG-Architektur [212] verdeutlicht dies. So entfallen dort 99,8% der **FLOPs** auf die Convolution-Schichten, während die restlichen Schichten im Encoder des Netzes für 0,12% der **FLOPs** verantwortlich sind. Auf den Decoder des Netzes entfallen lediglich 0,04% der **FLOPs**, obwohl dieser knapp 90% der Parameter beinhaltet. Die theoretische Anzahl der benötigten **FLOPs** kann anhand folgender Formel berechnet werden:

$$\text{FLOPs}_c = (k_w \times k_h \times k_c \times \text{Out}_w \times \text{Out}_h \times \text{Out}_c) \times 2 - 1 \quad (2.7)$$

Letztendlich ist die reale Anzahl allerdings abhängig von den verwendeten Software-Bibliotheken, Treibern und der Hardware, auf der das **CNN** ausgeführt wird.

3 Statische Verkehrsobjekte: 2D Objektdetektion



Abbildung 3.1: 2D Ampeldetektion mit Erkennung des aktuellen Zustandes und der enthaltenen Richtungsangabe. Die Farbe der Bounding Box zeigt den erkannten Zustand der Ampel, das darüber angezeigte Symbol die erkannte Richtungsinformation. Erstveröffentlichung in: [8]

Durch ein Autonomes Fahrzeug müssen bei der Teilnahme am Straßenverkehr vielfältige Objekte mit teils deutlich unterschiedlichen Charakteristika erkannt werden. Für die Erkennung dieser Objekte können je nach Objektklasse somit auch sehr unterschiedliche Anforderungen an Genauigkeit und Geschwindigkeit existieren. Dies macht eine getrennte Behandlung der einzelnen Objektklassen notwendig. Hierzu wird im Rahmen dieser Arbeit zwischen statischen und dynamischen Verkehrsobjekten unterschieden. Die Definition für statische Verkehrsobjekte folgt der Definition von statischen Verkehrselementen von Nienhueser [170] und lautet:



Definition 1. Ein *statisches Verkehrsobjekt* bezeichnet ein ortsfestes Objekt im Verkehrsraum. Es kann also seine Position nicht eigenständig verändern.

Dies beinhaltet im Wesentlichen Objekte wie Lichtsignale oder Verkehrszeichen. Diese Objekte haben eine regulatorische Bedeutung im Straßenverkehr und müssen aus diesem Grund erkannt werden. Eine hochgenaue Positionsbestimmung ist in der Regel nicht erforderlich. So ist eine Detektion der Objekte in 2D Kamerabildern hinreichend zur Erfassung von deren Bedeutung. Die Begriffsbildung des Begriffes Detektion ist an dieser Stelle allerdings nicht eindeutig, da sich hierfür zwei verschiedene Bedeutungen durchgesetzt haben. Während in früheren Arbeiten eine klassenunabhängige Detektion von Objekten von Interesse als Detektion bezeichnet wurde, wird im Rahmen dieser Arbeit folgende Definition verwendet:



Definition 2. Die *Objektdetektion* bezeichnet die Positionsbestimmung eines Objektes im entsprechenden Zielkoordinatensystem sowie dessen Klassifikation.

Eine klassenunabhängige Detektion wird nachfolgend mit dem Begriff *generische Objektdetektion* bezeichnet. Hierbei wird lediglich festgelegt, dass sich an der vom Detektor ausgegebenen Position ein Objekt befindet. Soll die Klasse des Objektes bestimmt werden, muss nachgelagert eine Klassifikation durchgeführt werden.

Im Folgenden wird zunächst in Abschnitt 3.1 das Problem der Detektion statischer Verkehrsobjekte detailliert beschrieben. Insbesondere die untere Schranke für die Detektionslatenz wird hierbei betrachtet. In Abschnitt 3.2 werden verwandte Arbeiten aus dem Bereich der auf CNNs basierenden Objektdetektion im 2D Bildraum sowie der Detektion statischer Verkehrsobjekte betrachtet. Nachfolgend wird in Abschnitt 3.3 die verwendete Methodik zur Detektion statischer Verkehrsobjekte vorgestellt. Die vorgeschlagene CNN-Architektur basiert hierbei auf aus der Literatur bekannten CNN-Architekturen zur Detektion von 2D Objekten, welche entsprechend der Problemstellung angepasst und erweitert werden. Die vorgestellten Konzepte werden in Abschnitt 3.4 anhand der konkreten Anwendung der Ampelerkennung evaluiert.

Teile dieses Kapitels wurden bereits in den Publikationen [13] und [8] vorgestellt. Ebenfalls flossen Ergebnisse der Abschlussarbeiten [39], [20] und [31] in dieses Kapitel ein.

3.1 Problembeschreibung

Allen statischen Verkehrsobjekten ist die Eigenschaft inhärent, dass sie grundsätzlich ortsfest sind. Es ist nicht damit zu rechnen, dass diese über die Zeitspanne einer Vorbeifahrt ihre Position verändern. Es ist dennoch möglich, dass diese Objekte lediglich temporär, also für eine Zeitspanne von mehreren Tagen oder Wochen, vorhanden sind. So kann aus dem alleinigen Wissen über diese Objekte aus vergangenen Fahrten oder Hintergrundinformationen kein finaler Rückschluss über das Vorhandensein eines Objektes gezogen werden. Dass diese

Objekte ortsfest sind bedeutet allerdings nicht, dass auch der Zustand von diesen Objekten statisch sein muss. Gerade Ampeln oder digitale Wechselverkehrszeichen können ihren Zustand dynamisch anpassen. Auch durch diesen Umstand wird eine Detektion der statischen Verkehrsobjekte im Autonomen Fahrzeug notwendig.

Eine Erkennung des aktuellen Zustandes ist dabei lediglich durch visuelle Sensorik wie etwa Kamerasysteme möglich. Neben einer Erkennung ist eine Übertragung des aktuellen Zustandes durch die sogenannte [Car-to-Infrastructure \(C2I\)](#) Kommunikation möglich. Hierzu müsste allerdings jedes Verkehrsobjekt in der Lage sein, den aktuellen Zustand zu übermitteln. Für die spätere Wahl konkreter Algorithmen ist auch die Anforderung an die Art der Verortung ein entscheidendes Kriterium. So muss entschieden werden, ob die Detektion statischer Verkehrsobjekte im Bildraum hinreichend zur Erfüllung der Anforderungen an die Positionsgenauigkeit ist. Andernfalls wäre eine Positionierung der Objekte in [3D](#) Koordinaten notwendig. Konkret werden die gestellten Anforderungen nachfolgend am Beispiel der Ampeldetektion betrachtet.

Ampeln, im Verkehrswesen auch [Lichtsignalanlage \(LSA\)](#) und verkehrsrechtlich [Lichtzeichenanlage \(LZA\)](#) genannt, dienen der Steuerung des Straßenverkehrs mittels Lichtzeichen. Sie werden zur Verbesserung der Verkehrssicherheit und der Qualität des Verkehrsablaufs eingerichtet. [\[83\]](#) Ihre primäre Komponente sind verschiedenfarbige Lampen auf einfarbigem Hintergrund. Auf der Leuchtfläche aufgebraute Piktogramme können die Gültigkeit der Ampel auf einzelne Gruppen von Verkehrsteilnehmern oder Fahrrichtungen einschränken. Sie bestehen aus einer unterschiedlichen Anzahl und Anordnung einzelner Segmente mit Lampen in jeweils verschiedenen Farben.

Die einzelnen Segmente können hierbei sowohl horizontal als auch vertikal angeordnet sein. In einzelnen Ländern können sich die länderspezifischen Regularien hierzu deutlich unterscheiden. Auch die Positionierung von einzelnen Ampeln in Relation zu der jeweils zugehörigen Haltelinie variiert von Land zu Land stark. Im europäischen Straßenverkehr wird eine Ampel für den Fahrzeugverkehr meist in direkter Nähe zu der zugehörigen Haltelinie vor einer Kreuzung platziert. In Nordamerika hingegen stehen Ampeln im Normalfall hinter der Kreuzung und damit in einer größeren Distanz zu der Haltelinie und dem Fahrzeug, aus dem die Ampel erkannt werden muss.

Nachfolgend werden die Regelungen für den deutschen Straßenverkehr herangezogen, da eine Betrachtung aller nationalen Regularien den Umfang dieser Arbeit übersteigen würde. Ebenfalls liegt der Fokus der in diesem Kapitel vorgestellten Konzepte und durchgeführten Experimente auf Ampeln aus dem deutschen Verkehrsraum. Die vorgestellten Konzepte können unter Beachtung der regionalen Besonderheiten auch auf andere Länder übertragen werden. Da Deutschland neben einer Reihe weiterer Länder das *Wiener Übereinkommen über Straßenverkehrszeichen* [\[74\]](#) unterzeichnet hat und dort grundsätzliche Richtlinien auch zur Vereinheitlichung von Lichtsignalanlagen festgeschrieben sind, ist diese Übertragung in vielen Fällen deutlich vereinfacht. In Deutschland sind die zu beachten-

3 Statische Verkehrsobjekte: 2D Objektdetektion

den Regeln für den Bau und Betrieb von Ampeln in der [Richtlinie für Lichtsignalanlagen \(RiLSA\)](#) [83] festgelegt.

Ampeln in Deutschland sind grundsätzlich vor einer Kreuzung platziert, lediglich Ampeln an Fußgängerüberwegen sind am Ende des Überweges oder bei geteilten Überwegen am Ende des jeweiligen Übergangsegmentes angebracht. Eine Ampel besteht im Normalfall aus drei vertikal angeordneten Segmenten (Rot, Gelb, Grün - von oben) bei Fahrzeugampeln und zwei vertikal angeordneten Segmenten bei Fußgängerampeln (Rot, Grün - von oben). Vereinzelt existieren auch für den Fahrzeugverkehr Ampeln mit zwei Segmenten (Rot, Gelb - von oben) oder nur einem Segment (grüner Pfeil). Weitere Konfigurationen sind möglich, wie beispielsweise Anzeigen mit einem Segment zur Information über den Wegfall oder die Sperrung einer Fahrspur. Für die verschiedenen Ampelarten existieren im Wesentlichen folgende Signalfolgen, wobei ein einzelnes Signal auch als Zustand der Ampel bezeichnet wird:

- Rot – Rot-Gelb – Grün – Gelb – Rot (Fahrzeug)
- Dunkel – Gelb – Dunkel (Fahrzeug – außer Betrieb)
- Rot – Grün – Rot (Fußgängerüberweg)
- Dunkel – Gelb – Rot – Dunkel
- Dunkel – Grün [– Gelb] – Dunkel (Abbiegepfeil Grün)

Die Zuordnung einer Ampel zu einer Fahrspur erfolgt auf Seiten der Ampel durch Piktogramme mit Richtungspfeilen, aufgebracht auf die Scheiben der Lampen. Der Ampel ohne Piktogramm fällt hierbei die Bedeutung *für alle anderen Spuren* zu. Bei der Übergabe der Informationen zu detektierten Ampeln ist die Unterscheidung zwischen Ampeln mit und Ampeln ohne Richtungspfeil notwendig, da sich hieraus unterschiedliche Vorrangregeln ableiten. Eine Ampel ohne Richtungspfeil garantiert hierbei keinen Vorrang vor anderen Fahrzeugen. Ist ein Richtungspfeil vorhanden, wird damit auch Vorrang gegenüber anderen Verkehrsteilnehmern impliziert.

Als Gegenstück zu den Richtungspfeilen besitzt eine Fahrspur immer eine Zuordnung zu einer oder mehreren Fahrtrichtungen oder auch möglichen Wegentscheidungen auf der Kreuzung. Sind mehrere Fahrspuren für eine Richtungsfahrbahn vorhanden, so sind diese meist durch aufgemalte Pfeilmarkierungen einzelnen Richtungen zugeordnet. Die Richtungsauswahl kann auch durch Verkehrszeichen für die vorgeschriebene Fahrtrichtung (z. B. Verkehrszeichen 209-30, weißer Pfeil auf blauem Grund) eingeschränkt werden. Die finale Zuordnung zwischen Spur und Ampel wird letztendlich durch die gewünschte Fahrtrichtung des Fahrzeugs bestimmt. Anhand dieser Zielrichtung muss eine passende Fahrspur ausgewählt werden. Ist diese Fahrspur gefunden, ist die Ampel mit dem

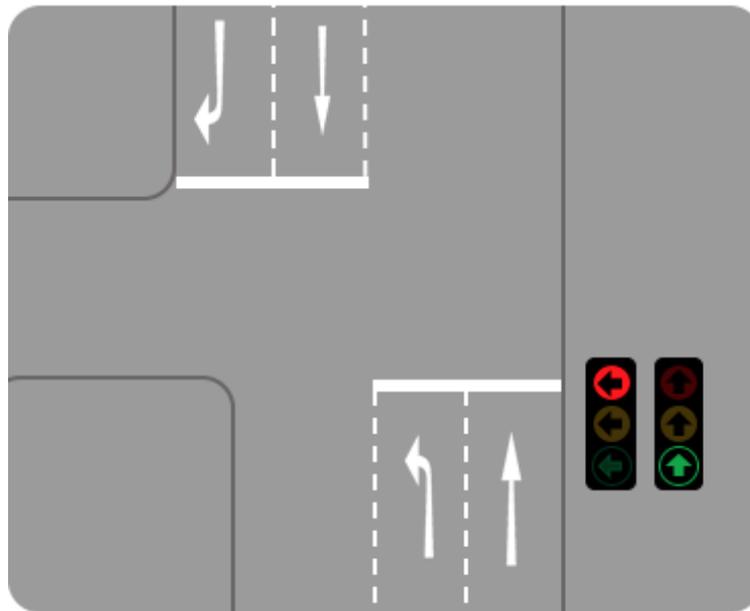


Abbildung 3.2: Ampelkreuzung aus Vogelperspektive. Es existieren getrennte Ampeln für die Geradeaus- und die Linksabbiegerspur. Sowohl die Fahrspuren als auch die Signale sind in diesem Fall durch Richtungspfeile gekennzeichnet, was eine einfache Zuordnung ermöglicht.

Richtungspfeil für die entsprechende Fahrtrichtung zu beachten oder, falls diese nicht vorhanden ist, die allgemeine Ampel. Abbildung 3.2 zeigt hier beispielhaft eine Kreuzungssituation mit getrennten Fahrspuren für die verschiedenen Fahrtrichtungen.

Auf komplexen Kreuzungen ist dieser Zuordnungsprozess bereits für Menschen keine triviale Aufgabe [130]. Allerdings kann angenommen werden, dass eine Zuordnung durch ein Autonomes Fahrzeug meist einfacher durchgeführt werden kann. Durch Navigationsdaten sind hier im Normalfall bereits Informationen zu einzelnen Spuren und deren möglichen Fahrtrichtungen vorhanden. In diesem Fall ist einzig die Zuordnung der Ampeln zu den einzelnen Fahrtrichtungen vorzunehmen. Hierzu müssen durch einen Detektor im Wesentlichen die Piktogramme der Ampeln korrekt erkannt werden. Die genaue Position einer Ampel ist an dieser Stelle nicht von Bedeutung. Lediglich die Zuordnung zu einer Kreuzung und somit einer Ampelanlage muss gewährleistet sein.

Die sich hieraus ergebenden Anforderungen an ein System zur Detektion von Ampeln können ebenso auf die Detektion von Verkehrszeichen übertragen werden. Auch in diesem Fall ist die genaue Position nicht entscheidend für die Bedeutung und Zuordnung der Schilder. So ergibt sich generell für die Detektion statischer Verkehrsobjekte, dass eine exakte Verortung der Objekte in 3D Koordinaten nicht notwendig ist. Sobald eine Detektion in 2D Bildkoordinaten vorliegt und die Größe der Objekte erkannt wurde, kann hieraus hinreichend genau der Gültigkeitsbereich des Verkehrsobjektes bestimmt werden.

Latenz der Detektion

Die Latenz des Objektdetektors gibt die vergangene Zeit zwischen Bereitstellung des Kamerabildes und Vorliegen der detektierten Objekte an. Gerade für die Detektion von Ampeln ist diese Zeitspanne von herausragender Bedeutung, da je nach Ampelzustand ein sofortiger Anhaltvorgang eingeleitet werden muss. Für die Herleitung einer maximalen Latenz ergeben sich mehrere Herangehensweisen. Eine Möglichkeit des Herangehens ergibt sich aus einem Vergleich mit der an die Latenz des menschlichen Fahrers gestellten Anforderung. Nach ÖNORM V 5050 [252] beinhaltet dies die Blickzuwendungszeit, Entscheidungszeit, Reaktionszeit und Umsetzzeit. In Deutschland existieren höchstrichterliche Entscheidungen des Bundesgerichtshofes, welche diese Zeiträume auf 0,7 bis 0,8 Sekunden begrenzen [176]. Allerdings erscheint eine Orientierung an diesen zur Klärung von Schuldfragen festgelegten, rechtlichen Maximalwerten nicht zielführend zur Festlegung der Latenz eines Objektdetektors.

Eine weitere Herangehensweise ist in diesem Fall die Betrachtung regulatorischer Maße und physikalischer Beschränkungen. Hierbei muss bekannt sein, ab welcher Entfernung die Sichtbarkeit einer Ampel gemäß Vorschrift garantiert ist. Grundsätzlich muss diese rechtzeitig gegeben sein, damit ein Fahrzeugführer auf die Signale einer Ampel reagieren kann. Die minimale Sichtbarkeit einer Ampel im deutschen Straßenverkehr ergibt sich aus der RiLSA [83] und beträgt in Abhängigkeit der maximal zulässigen Geschwindigkeit 35 – 80 Meter (siehe Tabelle 3.1).

Max. Geschwindigkeit	Mindestsichtbarkeit
50 km/h ~14 m/s	35 m
70 km/h ~20 m/s	80 m

Tabelle 3.1: Vorgegebene Mindestsichtbarkeit für Lichtsignalanlagen in Deutschland nach RiLSA [83].

Hierbei ist zu beachten, dass Ampelanlagen an Verkehrswegen mit zulässigen Höchstgeschwindigkeiten von mehr als 70 km/h grundsätzlich nicht gestattet sind. Die Mindestsichtbarkeit für die verschiedenen Geschwindigkeitsbeschränkungen entspricht nun genau dem minimal zur Verfügung stehenden Anhalteweg vor einer Ampel. Dieser setzt sich im Falle des Autonomen Fahrzeugs aus dem Bremsweg s_b und dem Reaktionsweg s_r zusammen.

$$s = s_b + s_r \tag{3.1}$$

Der Bremsweg ergibt sich hierbei im Wesentlichen aus der Bremsverzögerung und beträgt bei Betrachtung der Bremsung als negative gleichmäßige Beschleunigung a

$$s_b = \frac{v^2}{2 \cdot a} \tag{3.2}$$

mit gegebener Geschwindigkeit v . Zur Berechnung des benötigten Bremsweges muss nun noch eine Bremsverzögerung festgelegt werden. Da die Bremsverzögerung von verschiedenen Faktoren abhängig ist, werden hier zwei verschiedene Werte für die Berechnungen herangezogen. Für eine defensive Rechnung werden die Werte der sogenannten Komfortbremsung von 3 m/s^2 [48] angenommen. Eine Komfortbremsung entspricht hierbei einer normalen Bremsung im Straßenverkehr und stellt somit die Verzögerung dar, die beim Transport von Passagieren im Normalfall nicht überschritten werden sollte.

Für eine zweite Rechnung wird eine Abwägung zwischen dieser relativ schwachen Bremsung und einer Vollbremsung vorgenommen. Je nach Fahrzeug entspricht eine Vollbremsung einer Verzögerung von $7 - 10 \text{ m/s}^2$ [47]. Als defensiv gewählter Mittelwert für eine Normalbremsung kann hier ein Wert von 5 m/s^2 angenommen werden. Dies entspricht auch der mittleren Vollverzögerung, die ein Kraftfahrzeug nach StVZO § 41 Absatz 4 mindestens erreichen können muss. Die einzelnen Werte für den Bremsweg gemäß Formel (3.2) sind, abhängig von der zulässigen Geschwindigkeit v und der Art der Bremsung, in Tabelle 3.2 zusammengefasst.

	v	Bremsweg	Reaktionsweg	Reaktionszeit
Komfortbremsung (3 m/s^2) [48]	50 km/h	32 m	3 m	0,21 s
	70 km/h	63 m	17 m	0,85 s
Normalbremsung (5 m/s^2)	50 km/h	20 m	15 m	1,07 s
	70 km/h	38 m	42 m	2,10 s

Tabelle 3.2: Verbleibende Reaktionszeit für ein System zur Ampelerkennung gegeben verschiedene Geschwindigkeiten v und Bremsstärken.

Aus Formel (3.1) ergibt sich nun mit dem Anhalteweg, also der in Tabelle 3.1 aufgeführten Mindestsichtbarkeit, und den in Tabelle 3.2 genannten Werten für den jeweiligen Bremsweg der vor Beginn des Bremsvorgangs zur Verfügung stehende Reaktionsweg. Hieraus ergibt sich mit

$$t_r = \frac{s_r}{v} \quad (3.3)$$

die Reaktionszeit t_r , die dem Verarbeitungssystem von der Aufnahme des Bildes bis zum Vorliegen des Ergebnisses als Verarbeitungszeit zur Verfügung steht. Sowohl die einzelnen Werte für den Reaktionsweg, als auch für die Reaktionszeit sind ebenfalls in Tabelle 3.2 aufgeführt. Hieraus ergibt sich eine minimale Reaktionszeit von 0,21 Sekunden bei einer Komfortbremsung mit einer Geschwindigkeit von 50 km/h.

Zur Bestimmung der finalen Latenz muss allerdings beachtet werden, dass im Fall des Objektdetektors eine kontinuierliche Verarbeitung der Bilddaten erfolgt. So kann im ungünstigsten Fall ein Verarbeitungszyklus mit dem letzten Bild ohne

das zu detektierende Objekt durchgeführt werden. Der Durchlauf für die eigentliche Detektion kann aufgrund des sequentiellen Charakters aktueller Objektdetektoren in diesem Fall erst nach Ende des vorangegangenen Zyklus durchgeführt werden. Somit ergibt sich für die verfügbare Latenz t_l der Objektdetektion

$$t_l = 0,5 \cdot t_r \quad (3.4)$$

was in diesem ungünstigen Fall einer Komfortbremsung bei 50 km/h einer Latenz von maximal 100 Millisekunden entspricht. Durch eine Normalbremsung bei 50 km/h wäre eine Latenz von 530 Millisekunden hinreichend, um das Fahrzeug rechtzeitig stoppen zu können. Für die komplette Herleitung ist zu beachten, dass insbesondere die Werte der Bremsverzögerung stark von den Witterungs- und Sichtverhältnissen beeinflusst werden. In diesem Fall muss auf die [StVO § 3 Absatz 1](#) verwiesen werden: „Die Geschwindigkeit ist insbesondere den Straßen-, Verkehrs-, Sicht- und Wetterverhältnissen sowie den persönlichen Fähigkeiten und den Eigenschaften von Fahrzeug und Ladung anzupassen.“

Detektionsgeometrie

Bei der Detektion von Ampeln müssen mehrere gegensätzliche geometrische Anforderungen beachtet werden, die durch eine mögliche Positionierung von Ampeln vorgegeben sind. Die maximalen Distanzen, in denen eine Ampel sicher erkannt werden muss, ergeben sich aus den in der [RiLSA \[83\]](#) definierten Mindestsichtbarkeiten (siehe [Tabelle 3.1](#)). In Ländern, in denen Ampeln hinter der Kreuzung positioniert sind, ist die Kreuzungstiefe hier zu addieren.

Neben diesen maximalen Entfernungen ergeben sich im deutschen Verkehrsraum als weitere Anforderung die bei der Erkennung maximal zu erreichenden Winkel. So müssen die zu detektierenden Ampeln zunächst im Sichtbereich der verwendeten Kamera liegen, um erfolgreich detektiert zu werden. Hier ist insbesondere die Fahrzeugposition direkt an der Haltelinie der entsprechenden Ampel von Interesse. In diesem Fall sind die Winkel zu typischen Kamerapositionen im Fahrzeug, wie im oberen mittleren Bereich der Windschutzscheibe, am ungünstigsten.

Neben den seitlich am Ampelmast angebrachten Ampeln sind hier vor allem die meist bei mehrspurigen Fahrbahnen angebrachten Ampelausleger oder Signalbrücken eine Herausforderung. Die an diesen Bauteilen quer über der Fahrbahn befestigten Ampeln müssen nach [RiLSA](#) lediglich die Bedingung eines Mindestabstandes von 4,5 Metern zum Boden einhalten. Eine maximale Höhe ist hierfür in der [RiLSA](#) nicht definiert. Da gemäß [RiLSA](#) die Haltelinie einer Ampel mindestens 2,5 Meter vom Signalgeber entfernt sein muss, wird hier unter ungünstigen Bedingungen ein Öffnungswinkel der Kamera von über 90° benötigt.

Betrachtet man nun zusätzlich erneut die maximale Detektionsentfernung von 80 Metern unter Annahme einer 30 cm breiten Ampel, so können beide Szenarien kaum von einer einzelnen im Fahrzeug integrierten Kamera abgedeckt werden. In diesem Fall ist ein Setup aus mindestens zwei Kameras notwendig. Konkrete Berechnungen sind an dieser Stelle allerdings von den möglichen Kameraparametern abhängig. Die im weiteren Verlauf dieser Arbeit durchgeführten Bewertungen der vorgestellten Algorithmen basieren auf der Breite der Ampel im Kamerabild in Pixeln. So wird auch die maximale Detektionsentfernung einer Ampel in der Form minimale Breite im Eingabebild in Pixeln angegeben. Die ermittelten Werte können so genutzt werden, um je nach konkretem Einsatzzweck die Parameter von einer oder mehreren Kameras zur Erfüllung der Perzeptionsaufgabe entsprechend zu wählen.

3.2 Verwandte Arbeiten

Die Detektion statischer Verkehrsobjekte ist bereits seit den frühen Zeiten der Autonomen Fahrzeuge ein wichtiger Bestandteil an deren Erforschung. Bei ersten autonomen Versuchsfahrzeugen wie dem [Autonomous Land Vehicle \(ALV\)](#) [133] der Carnegie Mellon Universität ab 1985 und dem [Versuchsfahrzeug für autonome Mobilität und Rechnersehen \(VaMoRs\)](#) [71][70] der Universität der Bundeswehr und Mercedes-Benz 1986 aus dem [PROMETHEUS](#)-Programm waren bereits Kamerasensoren zur Wahrnehmung von Perzeptionsaufgaben an Bord. Diese Sensoren dienten allerdings in dieser frühen Phase allein der direkten Steuerung des Fahrzeugs [71]. In ersten Systemkonzepten [72] war die Detektion statischer Verkehrsobjekte in Form einer Verkehrsschilderkennung allerdings schon früh enthalten und wurde auch in nachfolgenden Versuchsträgern realisiert [204].

Die mutmaßlich erste Beschreibung eines konkreten Detektionsansatzes erfolgte 1990 ebenfalls im Rahmen des [PROMETHEUS](#)-Projektes für ein System zur Detektion von Verkehrszeichen [92]. Die hierfür verantwortliche Forschungsgruppe der Daimler-Benz AG veröffentlichte in direkter Folge eine Reihe weiterer Arbeiten zu dieser Problemstellung [191][127][248][79][193][192]. Ein erster Ansatz zur Erkennung von Ampeln [84] aus dem Jahr 1999 stammt ebenfalls aus der Forschung von Daimler-Benz. Auch diese Forschungsgruppe veröffentlichte in den Folgejahren weitere Arbeiten zur Erkennung von Ampeln [85][94][155]. Diese Ansätze basieren im Wesentlichen auf der Verwendung monokularer Kamerasensoren zur Erkennung der Verkehrsobjekte. Im weiteren Verlauf wurden auch vermehrt Stereokameras als Sensorik verwendet. Da der Fokus dieser Arbeit auf monokularen Systemen liegt, werden diese Ansätze bei der weiteren Betrachtung ausgenommen.

Zunächst soll bei der Betrachtung von monokularen Ansätzen die allgemeine Detektion von [2D](#) Objekten unter Verwendung eines [Convolutional Neural Network \(CNN\)](#) genauer beschrieben werden. [CNNs](#) wurden auch durch den Erfolg von AlexNet [144] bei der Challenge zur Bildklassifikation im Rahmen der

ILSVRC 2012 zu einem Standardwerkzeug der Bildverarbeitung. In vielen Bereichen der Bildverarbeitung kann man von einem Paradigmenwechsel weg von klassischen Ansätzen hin zu auf CNNs basierenden, lernenden Ansätzen sprechen. Dieser Paradigmenwechsel fand nach und nach auch in den einzelnen Anwendungsdomänen, wie der Erkennung von Verkehrsschildern, statt [130] und wurde zwischenzeitlich auch für die Ampelerkennung vollzogen. Die algorithmische Evolution von Systemen zur Detektion statischer Verkehrsobjekte wird in Abschnitt 3.2.2 näher betrachtet.

3.2.1 2D Objektdetektion mit CNNs

Nach dem Durchbruch der CNNs bei der Klassifikation von Bildern durch den Erfolg von AlexNet wurden auch bald Ansätze und Architekturen erforscht, mit denen CNNs ebenfalls für die Objektdetektion genutzt werden konnten. Mit dem Ansatz von Szegedy et al. [224], OverFeat [205] und Region CNN [100] wurden bereits im folgenden Jahr drei Ansätze vorgestellt, welche die grundsätzlichen Ansätze für CNN-basierte Architekturen gut repräsentieren. Die generelle Idee der Objektdetektion geht allerdings zurück bis ins Jahr 1998, in dem LeCun et al. [148] CNNs bereits zur Detektion von Schrift in Dokumenten nutzten. Aufgrund fehlender Rechenkapazitäten waren bei diesem Ansatz die Netzarchitekturen allerdings noch sehr klein und die Anwendungen daher begrenzt. Einige der damals vorgestellten Konzepte finden sich auch im Detektor OverFeat wieder.

Die verschiedenen Architekturen lassen sich generell in zwei Gruppen von Ansätzen unterteilen: One-Stage-Detektoren und Two-Stage-Detektoren. Wesentliches Unterscheidungsmerkmal zwischen diesen beiden Ansätzen ist die Anzahl der Verarbeitungsstufen von der Eingabe bis zur Ausgabe. Die One-Stage-Ansätze, zu denen OverFeat und der Ansatz von Szegedy et al. zählen, realisieren die komplette Verarbeitungskette der Objektdetektion vom Eingabebild bis mindestens hin zu klassifizierten Objektkandidaten innerhalb der Architektur eines einzigen CNN. Im Gegensatz hierzu besitzen die Two-Stage-Ansätze wie z. B. Region CNN eine zweistufige Architektur, welche eine gewisse Ähnlichkeit mit klassischen Verarbeitungsketten zur Objektdetektion [177][232] besitzt. So werden meist zunächst Regionen mit potentiellen Objekten generisch detektiert, welche im Anschluss durch einen Klassifikator einer der Objektklassen oder dem Hintergrund zugeordnet werden.

Die Laufzeit der einzelnen One-Stage-Ansätze weist eine höhere Stabilität auf, da nur ein CNN auf ein Eingabebild angewendet wird. Im Gegensatz hierzu wird der Klassifikator eines Two-Stage-Ansatzes auf eine vor der Ausführung nicht bekannte Anzahl an Regionen angewendet. Auch die generelle Laufzeit scheint bei diesen Ansätzen höher zu sein [153]. Allerdings liefern sie ohne weitere Maßnahmen bei One-Stage-Ansätzen die besseren Ergebnisse. Als Hauptursache hierfür wurde eine Unausgeglichenheit der Klassenverteilung in den Trainingsdaten

identifiziert [153]. Detailliert wird dies in Kapitel 6 beschrieben, in welchem diese Thematik ausführlich analysiert wird.

Two-Stage-Ansätze

Die Haupteigenschaft der sogenannten Two-Stage-Objektdetektoren ist die Trennung von generischer Detektion eines Objektes und dessen Klassifikation. Da hierbei die Ergebnisse der Detektion lediglich als Kandidaten für ein mögliches Objekt gesehen werden, welche durch die Klassifikation auch als Hintergrund verworfen werden kann, nennt man diese Ansätze auch *proposal based*. Durch diese Architektur wird ermöglicht, dass nicht der komplette Kern der Verarbeitungskette in Form tiefer Neuronaler Netze realisiert werden muss. Dies war vor allem bei ersten Ansätzen dieser Art üblich, da so CNNs Schritt für Schritt in die Verarbeitungskette integriert werden konnten. Sie wurden beispielsweise direkt zur generischen Detektion [224] oder als Vorverarbeitungsschritt für diese [113] verwendet. Im Klassifikationsschritt wurden CNNs zunächst zur Extraktion von Features für eine nachfolgende Support Vector Machine (SVM) verwendet [100][113], bevor sie in späteren Ansätzen die komplette Klassifikation durchführten [99].

Je nach Ansatz werden an die Genauigkeit der ersten Stufe in der Verarbeitungskette sehr unterschiedliche Anforderungen gestellt. Teilweise wird diese Stufe als Vorschlag möglicher, sehr ungenauer Objektregionen verwendet. Hierfür wird Selective Search [232] in Region CNN und EdgeBoxes [250] verwendet. Soll präziser schon eine geringere Anzahl an möglichen Regionen in der ersten Stufe ausgewählt werden, spricht man an der Stelle bereits von Objektkandidaten. Um diese Kandidaten effektiv zu bestimmen, werden oft ebenfalls CNNs eingesetzt. Neben einer direkten Regression der Objektboxen [78] zählen hierzu die Verfahren DeepMask [178][179], Dense Neural Patterns [251], Region Proposal Network (RPN) [190], Feature Pyramid Network (FPN) [152] und Spatial Pyramid Pooling (SPP) [113].

Im Klassifikationsschritt werden die Bounding Boxen der Regionen meist verfeinert [78] oder nochmals eine unabhängige Regression durchgeführt [190][223]. Die Extraktion der Features erfolgt im Normalfall durch ein CNN, allerdings gibt es auch hier unterschiedliche Vorgehensweisen. So können die Features für jede Region separat [100][99] oder gemeinsam über das komplette Bild berechnet werden [190]. In letzterem Fall werden die entsprechenden Features für die einzelnen Regionen im Nachgang selektiert. Die Klassifikation der einzelnen Regionen kann innerhalb weiterer Schichten eines CNN [99][190] oder durch andere Klassifikatoren wie eine SVM erfolgen [100][113]. Nachfolgend werden einige Ansätze detaillierter dargestellt, welche im Laufe der Entwicklung besondere Aufmerksamkeit auf sich zogen.

Region CNN (R-CNN) bezeichnet eine Familie von Two-Stage-Objektdetektoren. Das ursprüngliche Region CNN [100] verwendet CNNs lediglich zur Extraktion von Features in den zu klassifizierenden Bildausschnitten aus der Detektionsstufe. Zur Auswahl der Regionen wird Selective Search [232] verwendet, was eine große Menge von ca. 2.000 zu untersuchenden Regionen auf einem 227×227 Pixel großen Bild zur Folge hat. Die Laufzeit des Ansatzes liegt hierdurch im Bereich von mehreren Sekunden. Nachdem die Features in den einzelnen Regionen durch ein CNN extrahiert wurden, wird zur Klassifikation eine SVM eingesetzt. Optional werden zusätzlich die Bounding Boxen der Objekte durch ein lineares Regressionsmodell analog zu [82] verfeinert.

Der nachfolgende Ansatz Fast R-CNN [99] kann als eine Verschmelzung des R-CNN Ansatzes mit SPP-net [113] betrachtet werden. So wird nun für alle wesentlichen Verarbeitungsschritte ein CNN verwendet, wodurch die entstehende Architektur in einem Trainingsschritt trainiert werden kann. Hierbei spricht man vom sogenannten Ende-zu-Ende Training. Die Regionswahl erfolgt bei Fast R-CNN durch ein CNN mit RoI-Pooling, einem Spezialfall von SPP. Die Klassifikation der Regionen wird ebenfalls durch ein CNN realisiert, in dem ähnlich zu OverFeat die Klassifikation und die Verfeinerung der Bounding Boxen parallel erfolgen.

Eine der wesentlichen Neuerungen bei Faster R-CNN [190] ist die Berechnung der Features. Waren bislang die Features für jede Region separat berechnet worden, werden nun zunächst die kompletten Feature Maps der Ausgabe berechnet und die für die einzelnen Regionen benötigten Features aus diesem Ergebnis verwendet. Durch diese Vorgehensweise kann an vielen Stellen bei überlappenden Regionen eine Mehrfachberechnung der Features vermieden werden. Die Regionen werden nun durch einen eigenen, RPN genannten Ansatz bestimmt. Bei der Regression der Bounding Boxen wird der sogenannte Anchor Boxes Ansatz eingeführt, durch den die Parameter der Boxen relativ zu vorab bestimmten Durchschnitsboxen angegeben werden.

MultiBox bezeichnet eine Reihe von Ansätzen, deren Fokus die generische Detektion von Objekten durch ein CNN ist. Zu dieser Reihe gehört mit [224] auch der erste CNN-basierte Ansatz zur generischen Detektion von Objekten, auch wenn dieser den Namen MultiBox noch nicht trägt. In diesem Ansatz wird ein AlexNet [144] verwendet, um in den einzelnen Bereichen eines Bildes eine Regression von Bounding Boxen durchzuführen. Diese Bounding Boxen werden in verschiedenen Skalen des Bildes nach und nach verfeinert. Zusätzlich wird für jede Box ein Score bestimmt. Für jede Klasse müssen mit diesem Ansatz allerdings fünf getrennte Netze ausgeführt werden.

Die wesentliche Neuerung von DeepMultiBox [78] ist die Schätzung dieser fünf Werte in einem gemeinsamen Netz. Hierzu wird eine Fehlerfunktion verwendet, welche die einzelnen Fehler gewichtet aufsummiert. Auch dieses Netz ist nicht

auf die Klassifikation der Objektboxen ausgelegt. Allerdings kann für jede Objektklasse ein separates Netz trainiert werden, womit der Score als Indikator für die Klassenwahrscheinlichkeit angesehen werden kann. Für die Regression werden Priors eingeführt, wodurch die Bounding Boxen nun relativ zu diesen Priors berechnet werden. Mit MSC-MultiBox [223] wird als Backbone Netzwerk eine Inception-Architektur [222] verwendet. Zusätzlich entspricht die gesamte Architektur nun dem FCN-Prinzip.

One-Stage-Ansätze

Objektdetektoren aus der Klasse der One-Stage-Ansätze sind im Wesentlichen daran zu erkennen, dass Detektion und Klassifikation der Objekte innerhalb eines CNN durchgeführt werden, welches in einem Trainingsschritt Ende-zu-Ende trainiert werden kann. Aus diesem Grund werden sie oft auch als *proposal free*, also nicht abhängig von vorgeschlagenen Regionen für mögliche Objekte, bezeichnet. In einem möglichen Nachverarbeitungsschritt wird lediglich die Behandlung von mehrfach erkannten Objekten durchgeführt. Hierbei werden je nach Ansatz meist die resultierenden Bounding Boxen durch ein Clustering zusammengeführt [205] oder die Boxen von nicht dominanten Objektkandidaten z. B. durch eine *Non Maximum Suppression* (NMS) verworfen [113][223][153][78]. Teilweise werden auch direkt die detektierten Kandidaten als finale Objekte verwendet [186].

Die Objektdetektion selbst wird direkt in weiteren Schichten durchgeführt, die an den eigentlichen Backbone zur Extraktion von Features angehängt werden. Mit Ausnahme von YOLO [186] entsprechend diese Netze dem FCN-Designprinzip und führen die Regression der Bounding Boxen und deren Klassifikation in parallelen Schichten durch. Die Assoziation von Box und Wahrscheinlichkeiten erfolgt hierbei anhand der Position in der Ausgabe. Die Bounding Boxen werden relativ zur aktuellen Bildposition und meist auch relativ zu vorberechneten Standard-Boxen bestimmt. In verschiedenen Ausprägungen werden diese Standard-Boxen häufig als *Anchor Boxes* [153] oder *Priors* [188][157] bezeichnet.

Ansätze, welche im Laufe der letzten Jahre besondere Aufmerksamkeit erfuhren, werden nachfolgend nochmals mit Fokus auf den eingebrachten Neuerungen beleuchtet. In den letzten Jahren etablierten sich auch spezielle, auf die Laufzeit optimierte Netzarchitekturen. Diese werden im Rahmen der verwandten Arbeiten zur Optimierung von Netzarchitekturen in Abschnitt 7.2 beschrieben.

OverFeat [205] ist einer der ersten mit einem CNN realisierten Objektdetektoren und basiert auf einem Netzwerk, welches zunächst zur Klassifikation und Objektlokalisierung in Bildausschnitten mit lediglich einem sichtbaren Objekt entwickelt wurde. OverFeat verwendet zur Detektion ein zweigeteiltes Verfahren, bei dem die Klassifikation und die genaue Positionsschätzung der detektierten

3 Statische Verkehrsobjekte: 2D Objektdetektion

Objekte in getrennten, parallelen Schichten zeitgleich für alle möglichen Objekte durchgeführt werden. Diese Vorgehensweise wurde von den meisten anderen One-Stage-Detektoren übernommen.

Die Netzarchitektur von OverFeat enthält keine **Fully Connected (FC)**-Schichten, und entspricht somit einem Designprinzip, das zwischenzeitlich unter dem Namen **Fully Convolutional Network (FCN)** [158] bekannt wurde. Dies wird durch eine Neuinterpretation der **FC**-Schichten als Convolution-Schichten mit Filtern der Größe 1×1 erreicht und stammt ursprünglich von LeCun et al. [148]. Da im Netzwerk eine zu starke Reduktion der Auflösung stattfindet, wird zusätzlich eine so genannte Tiling-Schicht zur Erhöhung der Ausgabeauflösung eingeführt. Dieses Netz wird auf mehreren Eingabeauflösungen parallel angewendet und die Objektkandidaten werden final durch ein Clustering zu den finalen Detektionen zusammengefasst.

SSD [157] ist schon allein aufgrund seines Namens **Single Shot MultiBox Detector (SSD)** als eine One-Stage-Fortführung der ursprünglichen Two-Stage-Multi-Box-Ansätze [78][223] erkennbar. **SSD** stammt aus derselben Forschungsgruppe bei Google, erwähnt die vorherigen Arbeiten zu MultiBox als Ideengeber und steht in direkter zeitlicher Folge zu diesen. Eine aus dem Wechsel auf eine One-Stage-Architektur resultierende Verbesserung ist eine Halbierung der trainierbaren Parameter, da statt zwei Netzen der Architektur Inception-v3 [225] in [223] nun lediglich ein einzelnes Inception-v3-Netz verwendet wird. In einer späteren Version von **SSD** wird auch VGG16 anstelle von Inception-v3 verwendet. Wie in Tabelle 2.1 nachzulesen ist, werden hierdurch die Parameter weiter reduziert. Im Gegenzug steigt allerdings die Anzahl der benötigten Rechenoperationen deutlich.

Bei der Schätzung von Bounding Boxen verwendet **SSD** die bereits in MultiBox eingeführten Priors. Diese werden allerdings in einer leicht abgewandelten Form genutzt, die in Faster R-CNN als Anchor Boxes eingeführt wurde. Da eine Detektion der Objekte in **SSD** auf den Feature Maps mehrerer Zwischenstufen erfolgt, können Objekte in verschiedenen Größen effektiv detektiert werden. Hierdurch wird implizit auch jede Anchor Box in verschiedenen Skalen auf einem Eingabebild verwendet. Somit können im Training die einleitend bereits beschriebenen und für One-Stage-Detektoren typischen negativen Auswirkungen durch nicht ausbalancierte Trainingsdaten auftreten. Um dies zu verhindern, wird das Hard Negative Mining eingeführt. So werden aus der überrepräsentierten Klasse der Hintergrundbeispiele lediglich die n Datenpunkte zum Training herangezogen, welche die höchsten Klassenwahrscheinlichkeiten für eine Objektklasse besitzen. Zur Ausführungszeit werden die finalen Objektdetektionen durch eine **NMS** bestimmt.

YOLO steht für **You Only Look Once** und bezeichnet eine ganze Reihe von Ansätzen. Der initiale Ansatz YOLOv1 [186] unterscheidet sich von den meis-

ten anderen One-Stage-Detektoren durch die Tatsache, dass die letzten Schichten des Netzes ähnlich zu [224] aus **Fully Connected (FC)**-Schichten bestehen. Dies konnte sich allerdings im weiteren Verlauf nicht durchsetzen. Der YOLOv2 [187]-Ansatz verwendet anstelle des bisherigen AlexNet mit Darknet eine eigene auf VGG und dem **FCN**-Prinzip aufbauende Architektur. Mit diesem Schritt wurde mit *Dimension Priors* auch eine den Anchor Boxes ähnliche Technik zur Schätzung relativer Bounding Boxes eingeführt. Die Veröffentlichung von YOLOv3 [188] dient nach Aussage der Autoren im Wesentlichen der Dokumentation einiger kleinerer Anpassungen. So wurde das zur Extraktion von Features verwendete Darknet an das Residual-Prinzip [114] angepasst. Statt Softmax werden für die Klassifikation der Objekte Logistic Classifiers verwendet und die Dimension Priors werden modifiziert.

Mit YOLOv4 [46] wird der Ansatz um einige zwischenzeitlich veröffentlichte Techniken erweitert. So wird nun ein CSPDarknet verwendet, welches ein **Cross Stage Partial Network (CSPNet)** [235] zur Extraktion der Features nutzt. In einem auf diesen Backbone folgenden Teilnetz werden die Techniken **Spatial Pyramid Pooling (SPP)** [113] und **Path Aggregation Network (PAN)** [156] sowie ein modifiziertes **Spatial Attention Module (SAM)** [184] verwendet. Als Aktivierungsfunktion wird im gesamten Netz Mish [163] genutzt. Auch werden DropBlock [98] zur Regularisierung und CutMix [243] zur Datenaugmentierung eingesetzt. Weiterhin werden mit **Self-Adversarial Training (SAT)** und **Cross mini-Batch Normalization (CmBN)** neuartige eigene Ansätze präsentiert.

RetinaNet [153] ging aus der Entwicklung des Focal Loss als Fehlerfunktion für neuronale Objektdetektoren hervor. Die grundsätzliche Detektion der Objekte erfolgt nach dem bereits in OverFeat eingeführten Schema der getrennten Bounding Box Regression und Schätzung von Klassenwahrscheinlichkeiten. Die Architektur folgt dem **FCN**-Prinzip und basiert auf einem **Feature Pyramid Network (FPN)** [152], was eine Detektion der Objekte auf fünf verschiedenen Skalen des Eingabebildes ermöglicht. Es werden translationsinvariante Anchor Boxes mit drei verschiedenen Seitenverhältnissen nach [152] verwendet. Die durch die Detektion auf mehreren Skalen und die Verwendung von Anchor Boxes entstehende, große Menge von Objektkandidaten wird anhand eines Schwellwertes für deren Scores reduziert. Durch eine **NMS** werden daraus die finalen Detektionen erstellt. Im Training wird der Focal Loss zum Umgang mit nicht ausbalancierten Trainingsdaten als Fehlerfunktion verwendet.

DSOD [209] steht für **Deeply Supervised Object Detector** und wurde mit dem Ziel entwickelt, einen Objektdetektor ohne das Trainingsprinzip der Verfeinerung bereits vortrainierter Gewichte zu erzeugen. Beim Training anderer Objektdetektoren werden im Normalfall zunächst die Backbones zur Feature Extraction für eine Bildklassifikation mit Millionen von Bildern wie der **ILSVRC** trainiert. Da ein optimales Training hier eine große Menge an Rechenressourcen benötigt, ist dies zwischenzeitlich ein limitierender Faktor bei der Auswahl eines Backbones.

So werden meist bereits existierende Ansätze mit öffentlich verfügbaren Gewichten genutzt. Hierdurch werden allerdings Weiterentwicklungen der Architektur z. B. beim Lernen temporaler Zusammenhänge erschwert. Dieses Problem kann mit **DSOD** durch den Verzicht auf vortrainierte Gewichte vermieden werden. Um dies zu ermöglichen, wird eine Architektur ähnlich zu **SSD** verwendet, welche im Backbone auf die DenseNet-Architektur [120] zurückgreift. Zusätzlich werden sogenannte *STEM-Blöcke* genutzt, die ähnlich zur Inception-Architektur [225] von GoogLeNet einzelne Blöcke aus Convolution-Schichten mit verschiedenen Filtergrößen definiert.

3.2.2 Detektion statischer Verkehrsobjekte

Die Detektion statischer Verkehrsobjekte lässt sich wie bereits erwähnt bis ins Jahr 1990 zurückverfolgen [92]. Ein interessanter Aspekt der damals im Wesentlichen im Rahmen des **PROMETHEUS**-Projekts entstandenen Ansätzen ist, dass bereits nach relativ kurzer Zeit auf lernende Verfahren für einzelne Teilaufgaben zurückgegriffen wurde. So wurden bereits früh Entscheidungsbäume [127], Neuronale Netze [84][85] und **SVMs** [94][146] eingesetzt.

Allerdings waren diese lernenden Verfahren in der Ausführung meist deutlich zu langsam für eine Detektion im Fahrzeug in Echtzeit [155]. Um den zu untersuchenden Bildbereich einzugrenzen, wurde in vielen Ansätzen zunächst mit wenig rechenintensiven Verfahren nach möglichen Regionen gesucht, die in einem weiteren Schritt mit rechenaufwendigeren Verfahren untersucht wurden. Insbesondere zur Erkennung von Ampeln wurden auch vorhandene Karteninformationen in Kombination mit **GPS** verwendet [155][81][86][246][43], um mögliche Bildregionen einzugrenzen.

Zur initialen Auswahl möglicher Regionen wurde häufig eine Segmentierung nach Farbe [127][208][102][175] oder Helligkeit [67] verwendet. Für letzteres wurden meist morphologische Operatoren [67][170][159][101] wie beispielsweise der TopHat-Operator [66] eingesetzt. In den ausgewählten Regionen wurden Features für eine spätere Klassifikation z. B. mit dem **Histogram of Oriented Gradients (HOG)** [65] extrahiert [170][126][43], oder direkt durch weitere Verfahren wie AdaBoost [136], Template Matching [67][66][236], **SVM** [170][138][43], einer Circular Hough Transform [174][175] oder einer regelbasierten Verifikation basierend auf Geometrie- und Farbinformationen [242] bestimmt, ob die entsprechende Region tatsächlich ein Objekt enthält.

Eine ausführliche Übersicht über Verfahren dieser klassischen Verarbeitungskette ist zu finden in [170][69][130]. Mit Beginn der Verwendung von **CNNs** änderte sich diese Verarbeitungskette grundlegend. Nachfolgend werden diese Arbeiten in einem separaten Abschnitt vorgestellt. Der Fokus liegt hierbei auf Arbeiten zur Ampeldetektion und der Abgrenzung zu den im Rahmen dieser Arbeit vorgestellten Ansätzen.

Ansätze basierend auf CNNs

Mit dem Erfolg der CNN-basierten Klassifikatoren und Objektdetektoren wurden diese nach und nach auch für das Lernproblem der Ampeldetektion genutzt. Erste Ansätze verwendeten hierbei das CNN als Klassifikationsnetz für bereits detektierte Regionen [132][131]. Mit dem im Rahmen dieser Arbeit vorgestellten DeepTLR [13] wurde auch der Übergang der kompletten Detektoren hin zu CNN-basierten Systemen angestoßen. Zu diesem Zeitpunkt war die Anzahl der detektierbaren Klassen noch beschränkt. Der ursprüngliche DeepTLR-Ansatz konnte die Zustände *Rot* und *Grün* erkennen. Der Ansatz von Jensen et al. [129] beschränkte sich auf die zustandsunabhängige Detektion.

In den letzten Jahren entstanden weitere Ansätze, welche die Anzahl der Klassen auf weitere Zustände ausweiteten [44][167][182][181][42]. Einen Schritt in Richtung der zusätzlichen Klassifikation der Piktogramme in den Ampeln machten Kim et al. [137]. So wurde für zwei Klassen zwischen *Linksabbieger* und *ohne Piktogramm* unterschieden. Mit dem im Rahmen dieser Arbeit vorgestellten HDTLR-Ansatz wurde eine hierarchische Architektur für die Klassifikation beliebiger Piktogramme eingeführt. Bislang wurden Hierarchien mit maximal den zwei Stufen *Ampel* und *Zustand* verwendet [181], was meist einem klassischen Ansatz zur Two-Stage-Detektion entspricht. Nachfolgend werden einige ausgewählte Ansätze genauer betrachtet.

John et al. (2014) [132][131] verwendet eine eigene CNN-Architektur mit Ähnlichkeit zu LeNet [147] zur Klassifikation von Ampeln. Die möglichen Regionen im Bild werden vorab anhand der aktuellen GPS-Position und einer Datenbank mit Ampelpositionen grob ausgewählt. Kandidaten in den Regionen werden durch klassische Verfahren anhand von Farb- und Geometrieinformationen detektiert. Die verwendete CNN-Architektur besteht aus 5 Convolution-Schichten und dient ausschließlich der Klassifikation der gefundenen Regionen mit den Klassen $K \in \{ \textit{Rot}, \textit{Grün}, \textit{keine Ampel} \}$.

Jensen et al. (2017) [129] wendet den generellen Objektdetektor YOLOv2 [187] in verschiedenen Ausprägungen auf das Problem der Ampeldetektion an. Ziel ist lediglich das Erkennen der Ampel, nicht aber die Klassifikation von deren Zustand. Hierzu wird die Architektur von YOLOv2 durch Modifikationen an das Lernproblem der Ampelerkennung mit charakteristisch kleinen Objekten angepasst.

Behrendt et al. (2017) [44] stellen ein komplettes System zur Detektion und zum zeitlichen Tracking von Ampeln vor. Der im Kontext dieser Arbeit relevante Detektor basiert auf der YOLO-Architektur [186], welche für die Detektion kleiner Objekte adaptiert wird. Hierbei wird im Wesentlichen die Anzahl

der horizontalen und vertikalen Regionen von 7 auf 11 erhöht. Da die YOLO-Architektur Fully-Connected-Schichten verwendet, besteht die Wahl zwischen der Anpassung der Architektur oder der Anpassung der Eingabe. In diesem Detektor wurde letzterer Ansatz verwendet. So werden drei horizontal überlappende Ausschnitte der Größe 448×448 Pixel als Eingabe verwendet. Da die Bilder in dem verwendeten [Bosch Small Traffic Lights Datensatz \(BSTLD\)](#) eine Höhe von 720 Pixeln haben, wird die Annahme getroffen, dass Ampeln meist im oberen Bildbereich zu finden sind. Aus diesem Grund wird der untere Bildbereich des Eingabebildes ignoriert.

Der eigentlich als One-Stage-Detektor verwendete YOLO wird in diesem Ansatz um ein weiteres Netz zur Klassifikation der detektierten Ampeln ergänzt. So dient das YOLO-Modell letztendlich nur als [Region Proposal Network \(RPN\)](#) für einen nachgeschalteten Klassifikator. Für die Klassifikation sind die Klassen $K \in \{ \text{Hintergrund}, \text{Rot}, \text{Grün}, \text{Gelb}, \text{Aus} \}$ festgelegt. Das System ist in dieser Form also nicht in der Lage, Piktogramme in Ampeln zu erkennen.

TL-SSD (2018) [167] basiert auf dem [Single Shot MultiBox Detector \(SSD\)](#)-Ansatz zur Detektion von Objekten. Als Encoder wird hierbei eine Inception-v3-Architektur [225] verwendet. Durch den Einsatz von [SSD](#) werden Ampelkandidaten auf verschiedenen Skalen des Eingabebildes detektiert und in einer klassenunabhängigen [Non Maximum Suppression \(NMS\)](#) auf die finalen Detektionen reduziert. Die Klassifikation der Ampelzustände mit den Klassen $K \in \{ \text{Hintergrund}, \text{Rot}, \text{Grün}, \text{Gelb}, \text{Aus} \}$ wird durch eine zusätzliche, parallele Convolution-Schicht am Netzausgang realisiert. Diese Architektur wird verwendet, da bei direkter Zustandsklassifikation gemeinsam mit der Vordergrund-Hintergrund-Klassifikation ein Rückgang der Detektionsgenauigkeit festgestellt wurde. Die Evaluation des Ansatzes wurde auf dem eigenen DriveU Traffic Light Dataset [87] durchgeführt.

Kim et al. (2019) [139] verwenden einen Two-Stage-Ansatz zur Detektion von Ampeln. So wird zur Detektion der Regionskandidaten ein Binary Semantic Segmentation Network verwendet. Zur Klassifikation werden drei verschiedene Klassifikationsarchitekturen vorgeschlagen. Bei der Klassifikation werden im Wesentlichen die Zustände der Ampeln als Klassen verwendet, wobei für zwei Zustände zwischen allgemein und Linksabbieger unterschieden wird. Daraus ergeben sich die Klassen $K \in \{ \text{Grün-links}, \text{Rot-links}, \text{Rot}, \text{Grün}, \text{Gelb}, \text{Aus} \}$. Diese Klassenauswahl wirkt etwas willkürlich, lässt sich allerdings durch die in relevanter Anzahl annotierten Klassen in den Trainingsdaten des [BSTLD](#) erklären (siehe Tabelle 3.9).

Zur Evaluation des Ansatzes wird ebenfalls auf den Trainingsdatensatz aus dem [BSTLD](#) zurückgegriffen. Um aus diesem Datensatz Trainings- und Testdaten zu generieren, wird ein Split der Daten aus [137] verwendet. Allerdings wird an dieser Stelle auch erwähnt, dass die Daten für den Testdatensatz zufällig aus

dem Trainingsdatensatz gezogen wurden. Kombiniert mit der Tatsache, dass der [BSTLD](#) aus fortlaufenden Videodaten besteht, muss davon ausgegangen werden, dass benachbarte Bilder des Trainingsdatensatzes im Testdatensatz vorhanden sind. Da eine Evaluation auf dem Datensatz lediglich mit einem Baseline-Verfahren basierend auf einem einfachen Faster R-CNN durchgeführt wurde, ist ein Vergleich mit dem State of the Art nur sehr bedingt möglich.

Possatti et al. (2019) [182] verwenden zur Detektion von Ampeln einen YOLOv3 [188] Objektdetektor. Zusätzlich wird ein komplettes System vorgestellt, bei dem durch gespeichertes Vorwissen die Detektionen des CNN-Detektors bereinigt werden. Die Detektion von Ampeln wird in diesem Ansatz als Zweiklassenproblem mit den Klassen $K \in \{ \text{Rot-Gelb}, \text{Grün} \}$ aufgefasst. Eine Anpassung der YOLOv3-Architektur an das Lernproblem der Ampeldetektion und die damit verbundene Detektion kleiner Objekte findet nicht statt. Für die Detektion wird die bekannte Architektur mit einer Eingabegröße von 608×608 Pixeln verwendet. Die Evaluation wird auf verschiedenen Datensätzen durchgeführt. Hierbei bleibt allerdings unklar, wie die Behandlung von weiteren im Datensatz vorhandenen Klassen bei der Berechnung der Ergebnisse durchgeführt wurde. Ebenfalls bleibt unklar, wie mit der unterschiedlichen Größe von Eingabedaten und Netzeingabe verfahren wurde.

3.3 Methodik

Zur Detektion statischer Verkehrsobjekte wird in diesem Kapitel ein auf einem CNN basierender, hierarchischer 2D Objektdetektor vorgeschlagen. Hierzu wird in diesem Abschnitt zunächst die verwendete Repräsentation der Objekte im Eingabebild sowie in der CNN-Architektur näher beleuchtet. Die Architektur der verwendeten Netze wird in einem ersten Schritt für den allgemeinen Fall analog zu der Veröffentlichung [Deep Traffic Light Recognition \(DeepTLR\)](#) [13] vorgestellt. Im weiteren Verlauf wird eine zusätzliche Klassenhierarchie zur Detektion der in dem Lernproblem vorhandenen Klassen analog zu der Veröffentlichung [Hierarchical Deep Traffic Light Recognition \(HDTLR\)](#) [8] vorgestellt.

Analog zur Beschreibung des Lernproblems in Abschnitt 3.1 sind Beispiele zur Vorstellung und Visualisierung der Netzarchitekturen im Wesentlichen auf das Teilproblem der Ampelerkennung beschränkt. Die Methodik ist allerdings auf andere statische Verkehrsobjekte übertragbar, da diese meist vergleichbare Eigenschaften besitzen. Das Teilproblem der Ampelerkennung wurde an dieser Stelle stellvertretend gewählt, da es im Vergleich zu anderen Teilproblemen wie der Verkehrszeichenerkennung die größeren Herausforderungen beinhaltet. So verändert sich der Zustand einer Ampel über die Zeit und als selbstleuchtendes Objekt kommen visuelle Varianzen durch Leuchtmittel und Lichtverhältnisse hinzu.

Generell wird für eine hierarchische Detektion auch eine im Lernproblem definierte Klassenhierarchie benötigt. Zu diesem Zweck wird ein Algorithmus zur automatisierten Anordnung von ähnlichen Klassen vorgestellt. Hierbei wird auf das Lernproblem der Verkehrszeichenerkennung zurückgegriffen, da die Erzeugung einer Klassenhierarchie für Ampelzustände wenig aussagekräftig für den Nachweis der Leistungsfähigkeit eines entsprechenden Algorithmus ist. Abschließend wird die Methodik zur Nachbearbeitung der Netzausgaben und zur Findung der finalen Objektdetektionen näher erläutert.

3.3.1 Objektrepräsentation

Eine Herausforderung für CNNs sind Tasks, bei deren Lösung eine vorher nicht festgelegte Anzahl von Entitäten verarbeitet werden muss. Dies betrifft bereits in abgewandelter Form die Klassifikation von Bildern, bei denen die Anzahl der Klassen bereits innerhalb der Architektur des CNN festgelegt ist. In gleichem Maße lässt sich dies auf die pixelgenaue Segmentierung von Bildbereichen übertragen. Auch hier muss im Wesentlichen vorab die Anzahl der Klassen festgelegt sein. Im Fall von überwachtem Lernen lässt sich diese Herausforderung recht einfach lösen, da die Anzahl der Klassen bereits im Trainingsdatensatz ersichtlich und somit schon bei Festlegung der im Training genutzten Architektur verfügbar ist.

Die Charakteristik des Tasks zur Detektion von Objekten führt an dieser Stelle allerdings zu einer zusätzlichen Herausforderung. So ist die Anzahl der in einem Eingabebild befindlichen Objekte im Vorfeld nicht bekannt, weshalb eine Anpassung der Netzarchitektur nicht möglich ist. Des Weiteren kann sich die Anzahl der Objekte von Eingabe zu Eingabe verändern und das CNN muss in der Lage sein, die Ausgabe variabel anzupassen. Da eine variable Anpassung der Architektur vor allem auch für die Anzahl und Dimension der Netzausgabe nicht vorgesehen ist, ist hier eine Herangehensweise mit fest definierter Architektur notwendig.

Um dies sicherstellen zu können, müssen Ausgaben für eine ausreichend große Zahl an Objekten vorgesehen werden. Hierzu wird das Eingabebild mit Hilfe eines Gitternetzes in Regionen unterteilt, wobei für jede Region die Möglichkeit vorgesehen ist, ein Objekt zu schätzen. Anhand der Gitternetzgröße und somit der Anzahl der vorhandenen Regionen kann dann die maximale Anzahl an detektierbaren Objekten für jede Architektur vorab bestimmt werden. Hierbei wird festgelegt, dass eine Region ein Objekt detektieren soll, wenn sie Teil dieses Objektes ist oder zumindest teilweise innerhalb der Objektgrenzen liegt.

Die Objekte werden hierbei dargestellt als minimal umschließendes Rechteck, eine sogenannte **Bounding Box (BB)**. Diese Bounding Boxen sind definiert durch die Bildkoordinaten der linken oberen und rechten unteren Ecke. Es ist keine notwendige Bedingung, dass eine oder beide Ecken innerhalb der Region in der Ausgabe liegen, deren Resultat die Bounding Box ist. Die Koordinaten werden

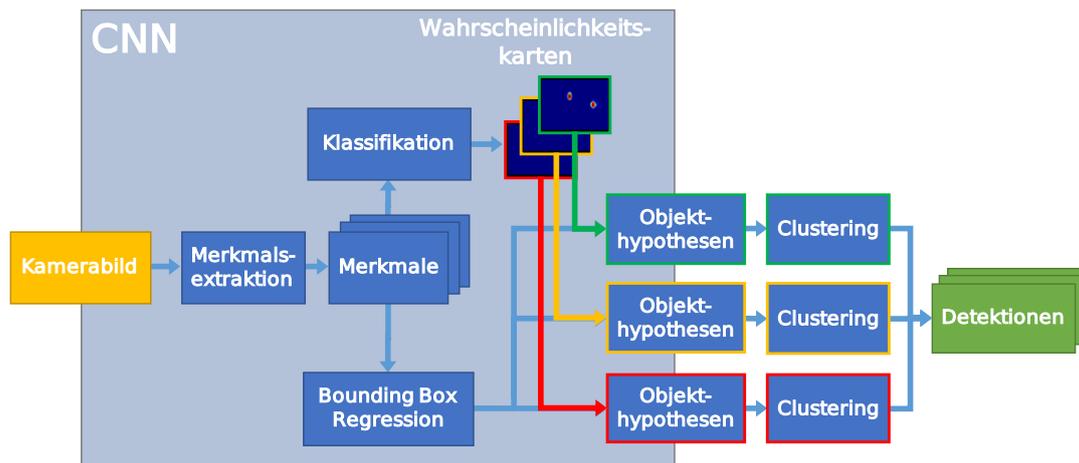


Abbildung 3.3: Genereller Aufbau des in diesem Kapitel vorgestellten Ansatzes zur Detektion von Ampeln. Das Bild zeigt die DeepTLR-Architektur ohne hierarchischen Ansatz.

grundsätzlich relativ zur Position der Region angegeben. Nur so kann sichergestellt werden, dass die Objekte unabhängig von der Position im Bild gelernt werden können.

3.3.2 CNN-Architektur

Um die Architektur für das CNN zur Detektion der statischen Verkehrsobjekte festlegen zu können, muss zunächst eine Grundsatzentscheidung zwischen den in Abschnitt 3.2.1 vorgestellten One-Stage- und Two-Stage-Ansätzen getroffen werden. Betrachtet man die Beschreibung des Lernproblems in Abschnitt 3.1, ist vor allem für die Detektion von Ampeln eine garantierte maximale Laufzeit von großer Wichtigkeit. Nur so kann sichergestellt werden, dass eine Ampel so frühzeitig erkannt wird, dass darauf basierend eine Fahrentscheidung getroffen werden kann. Im Idealfall ist also bereits im Vorfeld bekannt, wie viel Zeit zwischen Beginn der Detektion und dem Vorliegen des Ergebnisses vergehen wird.

Betrachtet man nun die beiden genannten Ansätze, lassen sich Gemeinsamkeiten und Unterschiede feststellen. So wird bei den meisten aktuellen Ausprägungen beider Ansätze eine Nachverarbeitung der erkannten Objektkandidaten durch ein Clustering oder eine *Non Maximum Suppression* (NMS) zum Finden der finalen Detektionen durchgeführt. Hier entsteht also in jedem Fall eine zeitlich nicht komplett vorhersagbare Komponente. Bei der Betrachtung des Weges hin zu diesen Objektkandidaten ergibt sich ein anderes Bild. Für die Two-Stage-Ansätze wird grundsätzlich zunächst ein Schritt zum Finden von Kandidatenregionen mit meist fester Laufzeit durchgeführt. Allerdings findet eine Klassifikation mit einem CNN für jeden dieser Kandidaten statt. Die Laufzeit ist an dieser Stelle also im Wesentlichen von der Anzahl der gefundenen Regionen abhängig und damit nicht vorab bestimmbar.

Schicht	conv1	pool1	conv2	pool2	conv3	conv4
# Filter	96	96	256	256	384	384
Filtergröße	11×11	3×3	5×5	3×3	3×3	3×3
Stride	4	2	1	2	1	1
Padding	0	0	2	0	1	1
Schicht	conv5	pool5	conv6	conv7	conv-prob	conv-bb
# Filter	384	384	4.096	4.096	192	256
Filtergröße	3×3	3×3	6×6	1×1	1×1	1×1
Stride	1	2	1	1	1	1
Padding	1	0	3	0	0	0

Tabelle 3.3: Architektur des [DeepTLR](#)-Ansatzes zur [CNN](#)-basierten Detektion von Ampeln.

Bei Analyse der One-Stage-Detektoren ergibt sich hier ein anderes Bild. So werden die Objektkandidaten bei diesen Ansätzen direkt durch die Ausführung des verwendeten [CNN](#) bestimmt. Von Eingabe des Bildes bis zum Erhalt der Klassenwahrscheinlichkeiten und den entsprechenden Bounding Boxen benötigt der Detektor also bei gleicher Hardwareverfügbarkeit bis auf geringe Schwankungen die gleiche Zeitdauer. So erscheint ein derartiger Ansatz zur Realisierung eines Netzes zur Detektion von statischen Verkehrsobjekten deutlich geeigneter. Die nachfolgend vorgestellte Architektur basiert somit auf dem One-Stage-Ansatz. Als Grundlage für den Aufbau einer Architektur wird hierbei das Design des Detektors [OverFeat](#) [205] verwendet. Die High-Level-Architektur des in diesem Abschnitt vorgestellten grundsätzlichen Ansatzes ist in [Abbildung 3.3](#) dargestellt.

Der erste Teil der [CNN](#)-Architektur besteht aus einem Encoder zur Extraktion der Features. Hier kann grundsätzlich jeder der in [Abschnitt 2.1](#) vorgestellten Ansätze verwendet werden. Die als [DeepTLR](#) veröffentlichte Konfiguration [13] verwendet an der Stelle die Feature Extraction von [AlexNet](#) [144]. Die konkrete Ausprägung dieser Architektur ist in [Tabelle 3.3](#) dargestellt. Für die weitere Verarbeitung der Features werden zunächst sequentiell zwei Convolution-Schichten verwendet. Dies entspricht einer Umwandlung des Klassifikationsteilnetzes bei Architekturen zur Klassifikation von Bildern, wie [AlexNet](#) oder [VGG](#) [212], nach dem [Fully Convolutional Network \(FCN\)](#)-Prinzip.

Durch diese Orientierung am [FCN](#)-Prinzip wird eine Klassifikation von Bildausschnitten ermöglicht und somit eine Grundlage für die Detektion von Objekten geschaffen. Auch ist das resultierende Modell nicht länger auf eine fixe Eingabegröße beschränkt. So kann ein trainiertes Modell auf durch Kameras mit unterschiedlicher Auflösung aufgenommene Bilder angewendet werden. Die eigentliche Detektionsaufgabe ist analog zum [OverFeat](#)-Ansatz durch eine Schicht zur Klassifikation und eine parallele Schicht zur Regression von Bounding Boxen gelöst. Um das [FCN](#)-Prinzip zu erhalten, handelt es sich hierbei jeweils ebenfalls um

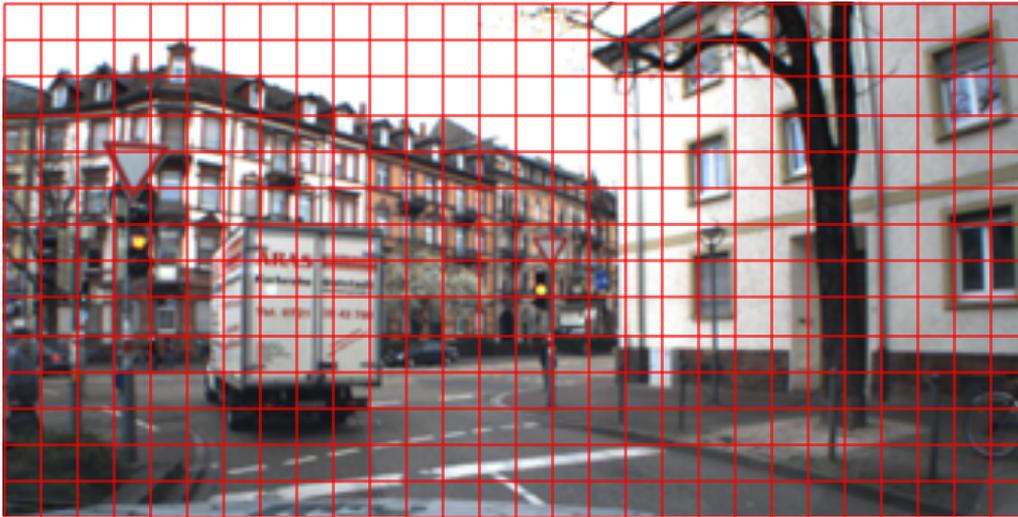


Abbildung 3.4: Einteilung des Eingabebildes in einzelne Regionen. In einem One-Stage-Objektdetektor findet eine Klassifikation sowie eine Schätzung einer Bounding Box für jede Region statt.

eine Convolution-Schicht. Die Betrachtung in Form von zwei getrennten Schichten ist dabei nur eine von zwei möglichen Sichtweisen. Eine Realisierung beider Schichten als eine gemeinsame Convolution-Schicht wäre mit der aus AlexNet bekannten Gruppierung der Faltungskerne gleichwertig möglich. Eine getrennte Betrachtung der Schichten vereinfacht allerdings eine separate Behandlung der einzelnen Ergebnisse in einer gemeinsamen Fehlerfunktion.

Die eigentliche Detektion der Objekte erfolgt im Netzwerk implizit anhand eines Sliding-Window-Verfahrens. Durch die Konzeptionierung der Architektur als FCN und der Adaption einer ursprünglichen Klassifikationsarchitektur kann eine Klassifikation an verschiedenen Punkten auf dem Eingabebild durchgeführt werden. So wird bei der originalen Eingabegröße der Klassifikationsarchitektur auch genau eine Klassifikation durchgeführt. Diese Klassifikation beinhaltet wie üblich eine Wahrscheinlichkeitsverteilung über die bekannten Klassen. Hier werden nun zusätzlich in einer Regression vier Werte für die Koordinaten einer möglichen Bounding Box geschätzt. Diese Bounding Box soll die Position des durch die Klassifikation bestimmten, im Bild befindlichen Objektes umschließen.

Das implizite Sliding-Window-Verfahren kommt zur Anwendung, sobald das Eingabebild vergrößert wird. Durch die Eigenschaften der Convolution-Schichten werden die Faltungen auch auf diesen Bereichen durchgeführt, was in größeren Feature Maps zwischen den Schichten resultiert. Diese zusätzlichen Bereiche werden durch das komplette Netz bis zur Ausgabe propagiert, sofern sie nicht zu klein für die Reduktionseffekte in den einzelnen Schichten sind. Diese entsprechen im Wesentlichen der Schrittweite (engl. Stride) der einzelnen Schichten. Der Stride der einzelnen Schichten kann hierbei in einem sequentiellen Netz zu einem globalen Stride des Netzes durch Multiplikation der einzelnen Strides kombiniert werden. Sobald die Breite des Eingabebildes um die Breite des globalen

3 Statische Verkehrsobjekte: 2D Objektdetektion

Stride vergrößert wird, vergrößert sich ebenfalls die Breite der Ausgabe um ein Feld. Gleiches gilt für die Höhe des Bildes.

Jedes dieser Felder enthält nun eine Wahrscheinlichkeitsverteilung sowie vier Werte für eine Bounding Box. Betrachtet man dies aus Sicht der Eingabe, entspricht jedem Feld der Ausgabe eine $S_g \times S_g$ Pixel große Region des Eingabebildes. S_g bezeichnet hierbei den globalen Stride. Für die meisten Encoder-Architekturen gilt $S_g = 32$. Somit kann eine Objektdetektion nach dem FCN-Prinzip bei Verwendung dieser Architekturen auch als Klassifikation eines Gitternetzes von Bildregionen betrachtet werden. Ein Eingabebild mit visualisierten Regionen ist beispielhaft in Abbildung 3.4 dargestellt.

Dieses Vorgehen entspricht im Wesentlichen der Objekterkennung mit einem klassischen Sliding-Window-Ansatz. Allerdings ist in diesem Fall das Sliding Window lediglich impliziert durch die Convolution-Operationen gegeben. Somit werden Berechnungen für die einzelnen Regionen nicht mehrfach durchgeführt. Aufgrund der Berechnung Schicht für Schicht werden entsprechende Features lediglich einfach berechnet und für die einzelnen Klassifikationen wiederverwendet. Dies hat eine gewisse Ähnlichkeit zur gemeinsamen Berechnung der Features für die Regionen von Interesse bei Faster R-CNN [190].

Da für jede der Regionen im Bild eine Wahrscheinlichkeitsverteilung über alle bekannten Klassen geschätzt wird, können hieraus mit den Wahrscheinlichkeitswerten einer Klasse sogenannte Wahrscheinlichkeitskarten dargestellt werden. Dies entspricht dem Verlauf der Wahrscheinlichkeiten einer Klasse über das Eingabebild hinweg, wie in Abbildung 3.5 (Ebene 2) dargestellt. Für die Detektion der statischen Verkehrsobjekte wird zusätzlich zu den Objektklassen eine explizite Hintergrundklasse verwendet. Da die meisten Bereiche eines Eingabebildes keine statischen Verkehrsobjekte enthalten, können damit als *Hintergrund* klassifizierte Regionen bei der weiteren Erzeugung der Objektkandidaten ignoriert werden. Zur Verwendung dieser Klasse gibt es bei der Annotation der Trainingsdaten zwei Varianten. So muss entweder der Hintergrund annotiert sein oder es müssen alle Objekte der bekannten Klassen vollständig annotiert sein. In diesem Fall können alle nicht annotierten Bereiche als Hintergrund betrachtet werden.

Reduktion der Auflösung

Die Anzahl und Größe der zur Detektion verwendeten Regionen hängt, wie bereits beschrieben, im Wesentlichen vom globalen Stride S_g der CNN-Architektur ab. Dieser beträgt, wie auch für die gewählte Architektur des DeepTLR-Ansatzes in Tabelle 3.3, für die meisten Architekturen 32. Für die DeepTLR-Architektur berechnet sich der Stride wie folgt: $S_g = 4 \times 2 \times 1 \times 2 \times 1 \times 1 \times 1 \times 2 \times 1 \times 1 \times 1 = 32$. Dies hat zur Folge, dass in jedem 32×32 Pixel großen Ausschnitt des Eingabebildes lediglich ein Objekt detektiert werden kann. Die charakteristische Eigenschaft von statischen Verkehrsobjekten ist allerdings eher deren geringe Größe, weshalb eine

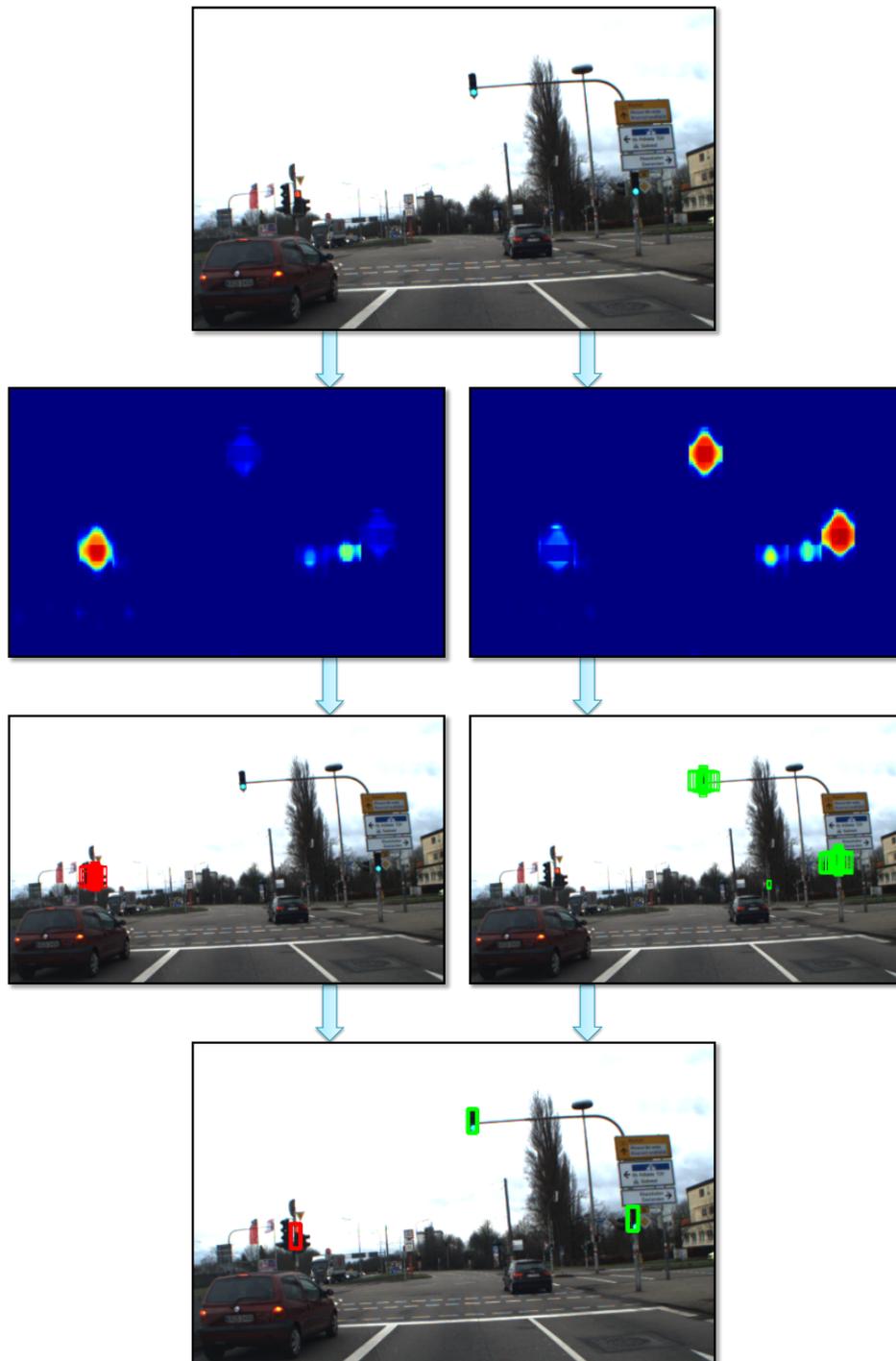


Abbildung 3.5: Erzeugung der Ergebnisse im vorgestellten, nicht hierarchischen [DeepTLR](#)-Ampeldetektor. Aus dem Eingabebild (Ebene 1) werden im [CNN](#)-Detektor die Wahrscheinlichkeitskarten (Ebene 2) für *rote Ampel* (links) und *grüne Ampel* (rechts) ermittelt. Aus diesen Karten und den geschätzten Bounding Boxes werden die Objektkandidaten (Ebene 3) bestimmt. Durch Clustering der Kandidaten ergeben sich die final detektierten Ampeln (Ebene 4). Ähnlich veröffentlicht in: [13]

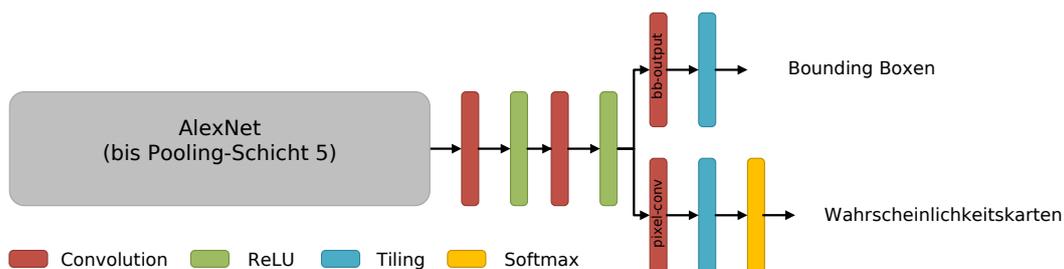


Abbildung 3.6: Detaillierte Ansicht der Architektur von **DeepTLR**. Am Ende des Encoders werden zunächst zwei Convolution-Schichten durchlaufen, bevor in getrennten Schichten Bounding Boxes und Wahrscheinlichkeitskarten für die Klassen bestimmt werden. In beiden Schichten ist eine Tiling-Schicht zur Erhöhung der Ausgabeauflösung vorhanden. Ähnlich veröffentlicht in: [20]

derart grobgranulare Detektion wenig geeignet erscheint. So können insbesondere bei kleinen Objekten mit geringem Abstand Probleme auftreten.

Aus diesem Grund werden in allen im Rahmen dieses Kapitels vorgestellten Architekturen Methoden verwendet, um die Größe der Regionen auf 4×4 Pixel zu reduzieren. Hierfür kommen im Wesentlichen zwei verschiedene Techniken zum Einsatz. So können die als Tiling-Schicht oder auch Skip-Kernel bezeichnete Methode aus OverFeat [205] oder die sogenannten Transposed-Convolution-Schichten [244] ins Netz integriert werden. Für den **DeepTLR**-Ansatz wurden die Tiling-Schichten mit einem Augmentierungsfaktor von 8 verwendet, weshalb in der Ausgabe der Schicht *conv-prob* $8 \times 8 \times 3 = 192$ Filter für die drei Klassen $K \in \{ \text{Rot}, \text{Grün}, \text{Hintergrund} \}$ und in *conv-bb* $8 \times 8 \times 4 = 256$ Filter für die vier Parameter der Bounding Box verwendet wurden. Die Anordnung der Schichten ist in Abbildung 3.6 dargestellt. In Experimenten mit späteren Modellen konnte zwischen beiden Methoden allerdings kein relevanter Unterschied bezüglich der Detektionsqualität festgestellt werden.

3.3.3 Hierarchische Detektion

Bislang wurde mit dem **DeepTLR**-Ansatz lediglich die Detektion einer Ampel sowie die Bestimmung des aktuellen Zustandes beschrieben. Allerdings wird bereits hier deutlich, dass die Anzahl der möglichen verschiedenen Zustände für einen **CNN**-Objektdetektor eine Herausforderung ist. Dies liegt daran, dass Ampeln in verschiedenen Zuständen noch immer eine große Ähnlichkeit haben. Die Architektur eines **CNN**-basierten Objektdetektors sieht für die verschiedenen Klassen im Ergebnis meist eine Wahrscheinlichkeitsverteilung basierend auf der Softmax-Funktion vor. Auf Basis dieser Wahrscheinlichkeiten wird in einem Detektor mittels eines Schwellwertes bestimmt, ob sich an der entsprechenden Stelle ein bekanntes Objekt oder lediglich Hintergrund befindet.

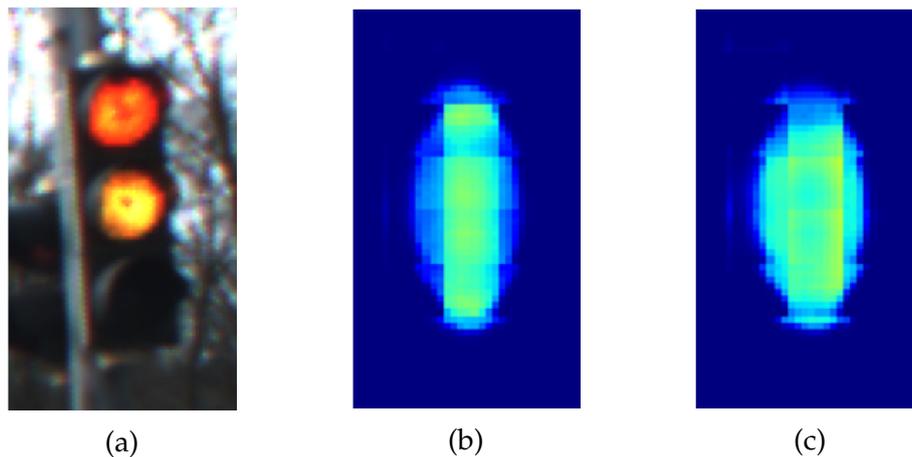


Abbildung 3.7: Detektion einer Ampel mit Zustand *Rot-Gelb* mittels des [Deep-TLR](#)-Ansatzes. (a) Ausschnitt des Eingabebildes. (b) Wahrscheinlichkeitskarte für die Klasse *Rot*. Skala der Wahrscheinlichkeiten beginnt bei 0 (blau) und endet bei 1 (rot) (c) Wahrscheinlichkeitskarte für die Klasse *Rot-Gelb*. Wie zu sehen ist, teilen sich die Wahrscheinlichkeiten zwischen den Klassen auf, weshalb beide Klassen jeweils mittlere Wahrscheinlichkeiten erreichen.

Muss ein Detektor nun eine Gruppe von ähnlichen Objekten erkennen und unterscheiden können, kann das in [Abbildung 3.7](#) dargestellte Phänomen auftreten. Wenn sich unterschiedliche Klassen ähnlich genug sind, kann dies die finale Verteilung der Klassenwahrscheinlichkeiten beeinflussen. Selbst wenn das Objekt nach wie vor korrekt klassifiziert wird, entfällt auch auf die unterlegene Klasse eine gewisse Wahrscheinlichkeit. Da der kompletten Klassifikation eine Wahrscheinlichkeitsverteilung zugrunde liegt, impliziert dies natürlich auch eine geringere Wahrscheinlichkeit für die korrekte Klasse. Die Wahrscheinlichkeit für das Unterschreiten des Schwellwertes einer Detektion steigt somit. Dieses Verhalten wurde parallel auch in der Arbeit zu TL-SSD [[167](#)] beobachtet. Für ein Lernproblem mit zwei Klassen $K = \{ Rot, Grün \}$ stellt dies meist noch kein Problem dar. Betrachtet man allerdings alle möglichen Zustände einer Ampel als Klassen $K = \{ Rot, Grün, Gelb, Rot-Gelb, Aus \}$ und die Tatsache, dass Ampeln jedes Zustandes eine gewisse Ähnlichkeit aufweisen, steigt die Gefahr einer nicht detektierten Ampel.

Da für ein Autonomes Fahrzeug neben dem Ampelzustand auch die weiteren, in Form eines Piktogramms enthaltenen Informationen einer Ampel von Bedeutung sind, steigt die Zahl der zu unterscheidenden Ampelklassen weiter. Zusätzliche Zustände könnten dann eine beliebige Kombination der Zustände $K = \{ Rot, Grün, Gelb, Rot-Gelb \}$ mit den Piktogrammen $P = \{ Fahrrad, Reiter, Fußgänger, Pfeil-links, Pfeil-rechts, Pfeil-gerade, Pfeil-rechts-gerade \}$ und weiterer Piktogramme sein. Somit sind z. B. je nach Lernproblem allein für eine Ampel im Zustand *Rot* 8–10 verschiedene Klassen möglich. Hieraus können sich für eine erfolgreiche Detektion durchaus ungünstige Wahrscheinlichkeitsverteilungen ergeben.

3 Statische Verkehrsobjekte: 2D Objektdetektion

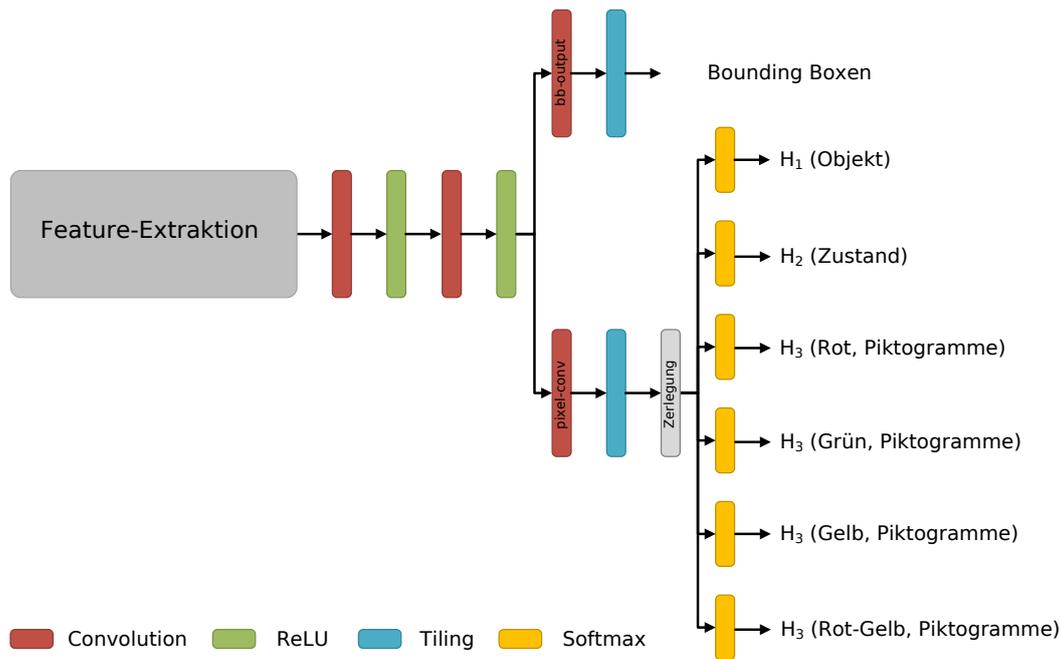


Abbildung 3.8: Detaillierte Ansicht der Architektur von HDTLR. Im Vergleich zur DeepTLR-Architektur wird kein gemeinsamer Softmax über die Wahrscheinlichkeiten der einzelnen Klassen berechnet. Die Wahrscheinlichkeitskarten werden zunächst gemäß der festgelegten Hierarchie zerlegt. Nachfolgend wird je ein Softmax auf die einzelnen Segmente der Hierarchie angewendet. Ähnlich veröffentlicht in: [8]

Zur Lösung dieses Problems wird eine hierarchische Detektion der Objekte vorgeschlagen. Klassenhierarchien wurden für den Task der Klassifikation von Verkehrszeichen und Zusatzzeichen bereits erfolgreich für eine statistische Klassifikation [127] und eine SVM [171][170] eingesetzt. Deshalb erscheint ein Transfer dieser Konzepte auf ein CNN zur Objektdetektion vielversprechend. Für eine hierarchische Detektion muss zunächst eine Klassenhierarchie der bekannten Klassen definiert werden. Für diesen Ansatz ist eine Hierarchie von Objektklassen definiert als verbundener azyklischer Graph in Form eines Wurzelbaumes

$$G(V, E) \tag{3.5}$$

mit einer Menge von Knoten V , welche die Klassen darstellen, und einer Menge von Kanten E , die eine Oberklasse-Unterklasse-Beziehung zwischen den Knoten darstellen. Jeder Knoten hat hierbei genau eine eingehende und eine beliebige Anzahl ausgehender Kanten. Für den Task der Detektion statischer Verkehrsobjekte könnte so ein Kindknoten die Klasse *Ampel* wiederum mit den Kindknoten *Rot*, *Grün*, *Gelb* und *Rot-Gelb* sein. Ein weiterer Kindknoten der Wurzel kann *Verkehrszeichen* mit den weiteren Kindknoten *Ronde*, *Dreieck*, *Raute* usw. sein. An

dieser Stelle sind allerdings verschiedene Optionen denkbar, weshalb die Anordnung von Klassen zur Erzeugung einer Hierarchie in einem nachfolgenden Abschnitt genauer beschrieben wird.

Die Integration dieser Klassenhierarchie in eine CNN-Architektur erfolgt im Rahmen der Erzeugung der Klassenwahrscheinlichkeiten. Der flache DeepTLR-Ansatz sieht an dieser Stelle in einem Schritt eine Netzausgabe für jede Klasse und eine Ermittlung einer Wahrscheinlichkeitsverteilung für jede Region durch die Softmax-Funktion vor. Die Netzausgaben für die einzelnen Klassen werden also in der Softmax-Funktion in Relation zueinander gesetzt. An dieser Stelle soll nun auch die Klassenhierarchie verwendet werden. So wird die Softmax-Funktion, wie in Abbildung 3.8 dargestellt, auf die einzelnen Stufen der Klassenhierarchie aufgeteilt und nicht länger über alle Klassen direkt berechnet. So ergibt sich eine Wahrscheinlichkeitsverteilung jeweils für die Klassen der Kindknoten einer durch einen Knoten repräsentierten Oberklasse.

Um dies erreichen zu können, wird für jeden Knoten in der Klassenhierarchie eine Netzausgabe benötigt, die Tiefe der Klassifikationsschicht muss also der Anzahl der Knoten im Graphen entsprechen. Hierfür müssen natürlich die aus den Annotationen erzeugten Grundwahrheiten ebenfalls an die hierarchische Ausgabe angepasst werden. Aus dem hierarchischen Softmax resultierende Wahrscheinlichkeitskarten für die Hierarchiestufen H_1 und H_2 der Ampelerkennung aus HDTLR sind beispielhaft in Abbildung 3.9 dargestellt. Die Auswirkungen dieser hierarchischen Detektion auf die Nachverarbeitung der Ausgabe werden in Abschnitt 3.3.4 erläutert.

Ableitung einer Klassenhierarchie

Im vorliegenden Fall der Ampelerkennung ist die Erzeugung einer Hierarchie über alle Klassen relativ eindeutig und auch manuell einfach zu bewerkstelligen. Es müssen lediglich die visuell ähnlichen Objektklassen wie *Ampel-Rot*, *Ampel-Grün* usw. der übergeordneten Klasse *Ampel* zugeordnet werden. Einzig die Reihenfolge zwischen Zustand und Piktogramm in der Hierarchie ist zu bestimmen. Bei vielfältigeren Mengen von Klassen stellt sich allerdings die Frage, wie diese für einen Detektor sinnvoll in einer Hierarchie angeordnet werden können. Diese Herausforderung tritt bereits bei dem eng verwandten Thema der Detektion von Verkehrsschildern auf. Eine manuelle Hierarchieerzeugung ist an dieser Stelle möglich, sie wird allerdings durch die Vielzahl an Klassen abhängig vom Ersteller und damit nicht eindeutig sein. Dies ist damit zu begründen, dass die Beurteilung der visuellen Ähnlichkeit von zwei Klassen zu einem gewissen Maß vom Betrachter abhängig ist.

Da die Klassenhierarchie zu dem Zweck erstellt wird, in einem Objektdetektor zwischen visuell ähnlichen Klassen zu unterscheiden, ist letztendlich also die visuelle Ähnlichkeit aus Sicht des Detektors die relevante Grundlage für eine Gruppierung von Elementen zu übergeordneten Klassen. Eine hohe visuelle Ähnlich-

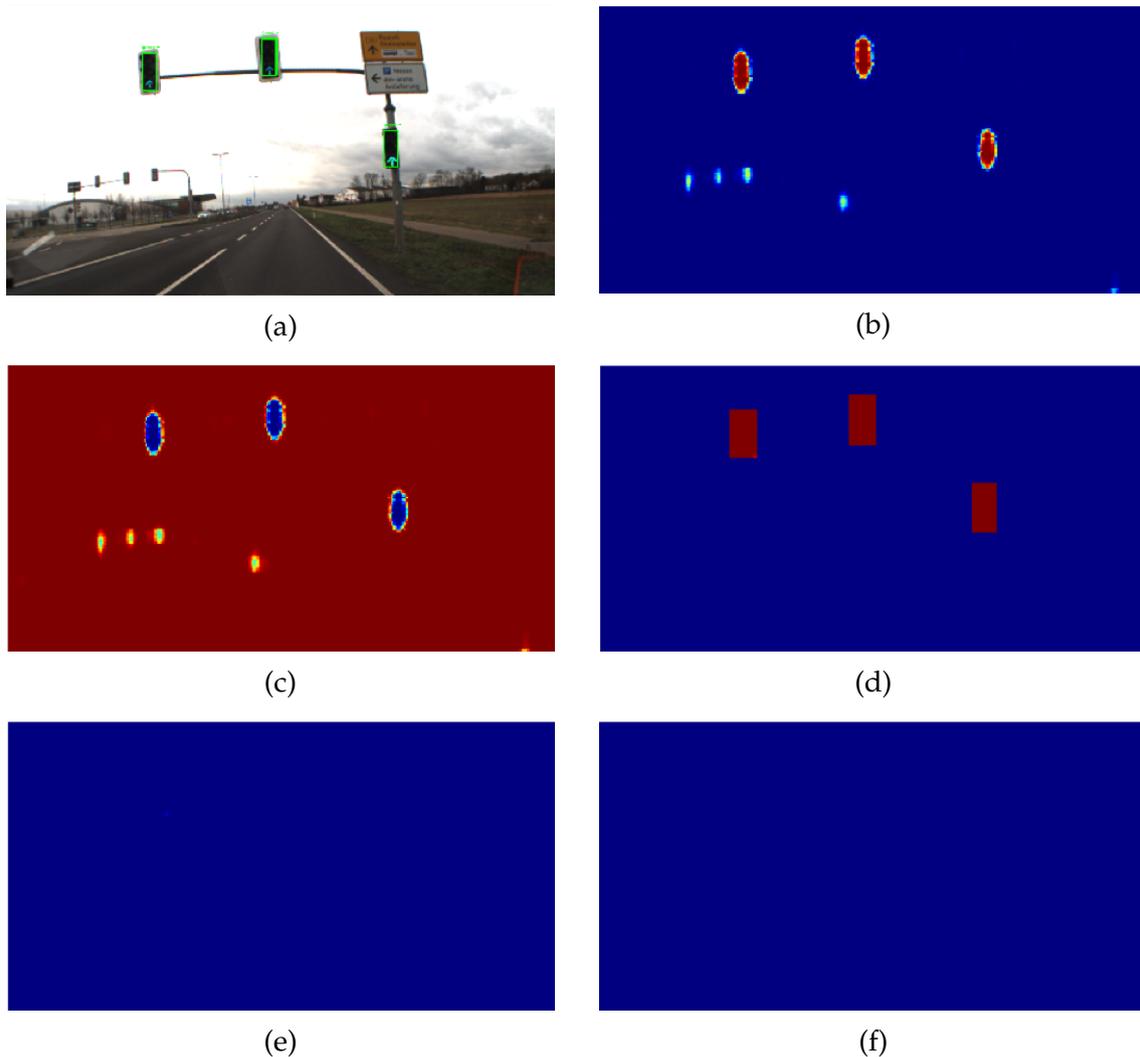


Abbildung 3.9: Hierarchische Detektion von Ampeln in einem Eingabebild nach [HDTLR](#). (a) Detektionsergebnis nach der Ausführung von [HDTLR](#). (b) Wahrscheinlichkeitskarte der Hierarchiestufe 1 für die Klasse *Ampel*. Die drei erkannten Ampeln sind in der Wahrscheinlichkeitskarte klar erkennbar. Zusätzlich entfallen geringe Wahrscheinlichkeiten auf die Rückseite der Ampeln auf der Gegenseite und eine weit entfernte Ampel (dunkelrot entspricht der höchsten Wahrscheinlichkeit). Diese liegen deutlich unterhalb der Detektionsschwelle. (c) Wahrscheinlichkeitskarte der Hierarchiestufe 1 für die Klasse *Hintergrund*. (d)-(f) Wahrscheinlichkeitskarten der Hierarchiestufe 2 für die Klassen (d) *Grün*, (e) *Rot* und (f) *Gelb*. Die Werte werden lediglich an den Stellen angezeigt, an denen sich ein detektiertes Objekt befindet.

keit führt erst zu dem Problem der verteilten Klassenwahrscheinlichkeiten, welches die Grundlage für die Einführung einer Hierarchie ist. Bei Betrachtung der Architektur eines CNN-basierten Objektdetektors wird deutlich, dass die visuelle Erscheinung von Objekten im Wesentlichen im Encoder betrachtet wird. Die Ausprägung der einzelnen Features bestimmt letztendlich auch den Grad der visuellen Ähnlichkeit verschiedener Objekte. Hierbei unterscheiden sich One-Stage-Detektoren und Two-Stage-Detektoren nur unwesentlich. Bei letzteren findet lediglich vorab eine Auswahl möglicher Regionen statt.

Vor diesem Hintergrund kann man die Tatsache nutzen, dass CNN-basierte Objektdetektoren im Wesentlichen auf Bildklassifikatoren beruhen. Meist werden die zur Bildklassifikation vortrainierten Encoder direkt für die Objektdetektoren übernommen (vgl. Abschnitt 2.1). Es findet lediglich eine Feinabstimmung statt. Auf dieser Grundlage wird eine Bestimmung der visuellen Ähnlichkeit anhand eines dem Objektdetektor zugrundeliegenden Klassifikators vorgeschlagen. So kann aus dem Detektionsproblem ein Klassifikationsproblem abgeleitet werden, indem für die jeweiligen Klassen die Bildausschnitte mit den Objekten klassifiziert werden. Anhand der Gesamtheit der falsch klassifizierten Objekte und deren fehlerhaft zugewiesener Klassen kann dann für den Objektdetektor eine visuelle Ähnlichkeit der Objektklassen abgeleitet werden.

Als Basis hierfür wird die Konfusionsmatrix verwendet, welche direkt die Verwechslungen des Klassifikators zwischen einzelnen Objektklassen darstellt. Hierbei wird die Annahme getroffen, dass die Zeilen der Konfusionsmatrix zunächst zufällig mit Objekten belegt sind und die Reihenfolge dieser Objekte beliebig verändert werden kann. Sofern die Reihenfolge der Elemente in den Zeilen durch Vertauschen verändert wird, muss dieselbe Vertauschung auch bei den Spalten der Matrix durchgeführt werden. So wird sichergestellt, dass die korrekten Klassifikationen weiterhin auf der Diagonalen der Matrix dargestellt sind. Zur Gruppierung von visuell ähnlichen Objektklassen ist nun das Ziel, die Einträge für diese Objektklassen in der Konfusionsmatrix in unmittelbare Nachbarschaft zueinander zu bringen. Dies entspricht einer Konfusionsmatrix, bei der alle Verwechslungen zwischen Objektklassen nächstmöglich zur Diagonalen positioniert sind. Eine zu minimierende Zielfunktion hierfür ist

$$\mathcal{L}(C) = \sum_{i=1}^n \sum_{j=1}^n C_{i,j} \cdot |i - j| \quad (3.6)$$

mit einer Konfusionsmatrix C und deren Eintrag $C_{i,j}$, welche Abstände zur Diagonalen linear bestraft. Ein möglicher Algorithmus zum Clustering der Konfusionsmatrix basierend auf dieser Zielfunktion wurde in der Abschlussarbeit [31] entwickelt und ist in Algorithmus 3.1 dargestellt. Die Zielfunktion wird dort als *Score* bezeichnet.

Algorithmus 3.1 Clustering der Konfusionsmatrix. Quelle: [31]

Require: $C \in \mathbb{N}^{n \times n}$, $steps \in \mathbb{N}$, $c \in (0, 1)$

procedure CLUSTERING(C , $steps$, T , c)

$bestScore \leftarrow SCORE(C)$

$bestC \leftarrow C$

for $i = 0; i < steps; i \leftarrow i + 1$ **do**

$p \leftarrow RANDFLOAT(0, 1)$

if $p < 0.5$ **then**

$i \leftarrow RANDINT(1, \dots, n)$

$j \leftarrow RANDINT(1, \dots, n) \setminus \{i\}$

$C' \leftarrow SWAP(C, i, j)$

else

$b \leftarrow RANDINT(1, \dots, n)$

$e \leftarrow RANDINT(b, \dots, n)$

$i \leftarrow RANDINT(1, \dots, n - (e - b))$

$C' \leftarrow MOVEBLOCK(C, b, e, i)$

end if

$s \leftarrow SCORE(C')$

if $s > bestScore$ **then**

$bestScore \leftarrow s$

$bestC \leftarrow C$

end if

end for

return $bestC$

end procedure

3.3.4 Nachverarbeitung der Netzausgabe

Die Ausgabe des Detektionsnetzes besteht für jede $n \times n$ Pixel große Region nun aus einer Wahrscheinlichkeitsverteilung über alle dem Detektor bekannten Klassen sowie den 4 Werten für eine mögliche Bounding Box um das in dieser Region befindliche Objekt. Die Wahrscheinlichkeitswerte für die einzelnen Klassen liegen hierbei in Form von Wahrscheinlichkeitskarten in jeweils einem zweidimensionalen Tensor vor. Basierend auf diesen Ausgaben werden nun zunächst Objektkandidaten und im Anschluss aus den Kandidaten die finalen Objektdetektionen bestimmt. Bei der weiteren Verarbeitung wird zunächst unterschieden, ob eine flache oder hierarchische Detektion durchgeführt wird.

In einer flachen Detektion werden nachfolgend die Objektkandidaten für alle Klassen separat bestimmt. Das gesamte Vorgehen ist in Abbildung 3.5 dargestellt. Wird eine Klassenhierarchie zur Detektion verwendet, werden in diesem Schritt lediglich die Klassen der Hierarchiestufe H_1 betrachtet. Für alle in diesem Schritt betrachteten Klassen wird die Bestimmung der Objektkandidaten anhand einer Schwellwertbetrachtung der einzelnen Einträge $p_{i,j}$ in der Wahrscheinlichkeitskarte P der entsprechenden Klasse durchgeführt:

$$k_{i,j} = \begin{cases} 1 & \text{falls } p_{i,j} > \theta, \\ 0 & \text{sonst.} \end{cases} \quad (3.7)$$

Hieraus ergibt sich mit Schwellwert θ eine binäre Karte mit den Regionen der Objektkandidaten K . Gemeinsam mit den Werten für die vorhergesagte Bounding Box in dieser Region $BB_{i,j}$ ergibt sich ein finaler Objektkandidat. Im Falle einer flachen Detektion sind somit die Objektkandidaten aller Klassen ermittelt und es kann mit der Ermittlung der finalen Objekte begonnen werden. Wird eine hierarchische Detektion durchgeführt, sind nun für alle Klassen der Hierarchiestufe H_1 die Objektkandidaten ermittelt. Die genaue Klassifikation findet in einem späteren Schritt im Anschluss an die Zusammenführung der Objektkandidaten statt.

Im folgenden Schritt müssen einzelne Objektkandidaten, welche dasselbe Objekt repräsentieren, zu einer finalen Objekthypothese vereinigt werden. Hierzu wird für die Objektkandidaten jeder betrachteten Klasse ein Clustering durchgeführt. Details zu dem durchgeführten Clustering sind in der Abschlussarbeit [39] zu finden. Das Ergebnis des Clusterings sind die Objekthypothesen für die verschiedenen Klassen. Zwischen den Objekthypothesen der verschiedenen Klassen wird nachfolgend ein Plausibilitätscheck durchgeführt, um verschiedene Detektionen desselben Objektes auszuschließen. Nachdem in einem vorherigen Schritt bereits doppelte Detektionen der gleichen Klasse ausgeschlossen wurden, können bislang noch Detektionen verschiedener Klassen für dasselbe Objekt vorliegen. So kann eine Ampel beispielsweise von einem flachen Detektor als rote und grüne

3 Statische Verkehrsobjekte: 2D Objektdetektion

Ampel detektiert werden. Um diese Doppeldetektionen zu erkennen, wird folgendes Überdeckungskriterium verwendet:

$$\frac{BB_1 \cap BB_2}{BB_1 \cup BB_2} > \phi \quad (3.8)$$

Übersteigt diese Überlappung den Schwellwert ϕ , wird eine Doppeldetektion angenommen und eine Bewertung der einzelnen Objekthypothese anhand folgender Bewertungsfunktion durchgeführt:

$$score(BB_1) = \frac{|size(BB_1)|}{|size(BB_1) + size(BB_2)|} \cdot conf(BB_1) \quad (3.9)$$

Hierbei wird durch die $size()$ -Funktion die Anzahl der überdeckten Pixel bestimmt und durch die $conf()$ -Funktion die diesen Pixeln entsprechenden Wahrscheinlichkeitswerte in der zur Klasse der Objekthypothese gehörenden Wahrscheinlichkeitskarte summiert. Die Objekthypothese mit dem geringeren Score wird an dieser Stelle verworfen.

Für flache Detektoren stehen an dieser Stelle die finalen Objekthypothesen fest. Wird ein hierarchischer Detektor verwendet, muss im weiteren Verlauf die Klassifikation in den einzelnen Hierarchiestufen bestimmt werden. Um diese Klassifikation auf der Hierarchiestufe H_i durchzuführen, werden die Wahrscheinlichkeitskarten der Klassen auf der entsprechenden Hierarchiestufe an der Stelle der Detektion betrachtet. Hierbei bildet die Klasse die finale Klassifikation, welche in Summe die maximalen Wahrscheinlichkeitswerte auf der Fläche des detektierten Objektes erreicht. Diese Klassifikation wird auf jeder Hierarchiestufe wiederholt. Somit ergeben sich die finalen Objekthypothesen auch für die hierarchische Detektion.

3.4 Experimente und Evaluation

Die experimentelle Evaluation der in diesem Kapitel vorgestellten Ansätze zur Detektion statischer Verkehrsobjekte wird im Wesentlichen anhand des Lernproblems der Ampelerkennung durchgeführt. In einem weiteren Experiment wird die Gruppierung von Klassen zur Generierung einer Hierarchie anhand des Problems der Verkehrszeichenerkennung dargestellt. Für die einzelnen Experimente verwendete Datensätze werden nachfolgend in Abschnitt 3.4.1 eingeführt und detailliert beschrieben. Die zum Vergleich und zur Interpretation der Detektionsergebnisse genutzten Evaluationsmetriken sind in Abschnitt 3.4.2 ausführlich dargestellt.

Die Evaluation der Ansätze selbst ist in drei Bereiche unterteilt. Der erste Bereich in Abschnitt 3.4.3 behandelt die generelle Leistungsfähigkeit der in Abschnitt 3.3.2 vorgestellten CNN-basierten Architektur zur Detektion von statischen Verkehrsobjekten. Der in Abschnitt 3.3.3 eingeführte hierarchische Ansatz

zur Detektion vieler gleichartiger Verkehrsobjekte ist Hauptbestandteil des zweiten Bereiches in Abschnitt 3.4.4. An dieser Stelle werden auch Untersuchungen zu verschiedenen Encodern zur Extraktion von Features durchgeführt. Ebenso findet eine Betrachtung der Laufzeiten von verschiedenen trainierten Modellen statt.

Abschließend für diesen Bereich werden für die beste Konfiguration Untersuchungen über die Leistungsfähigkeit der Detektion in Abhängigkeit von der Größe der Objekte im Bild und damit auch in Abhängigkeit von der Entfernung zum Objekt durchgeführt. Der letzte Bereich behandelt den in Abschnitt 3.3.3 skizzierten Ansatz zur Gruppierung von Objekten für die automatische Generierung einer Klassenhierarchie. Diese Untersuchung ist in Abschnitt 3.4.5 zu finden.

3.4.1 Datensätze

Zur Evaluation der Detektion statischer Verkehrsobjekte werden im Wesentlichen Datensätze aus dem Bereich der Ampeldetektion verwendet. Im Rahmen dieser Arbeit wurden mit dem [DeepTLR](#)-Datensatz und dem [HDTLR](#)-Datensatz zwei eigene Datensätze erstellt. Zusätzlich wurde zur Evaluation der [Bosch Small Traffic Lights Datensatz \(BSTLD\)](#) verwendet. Alle im Rahmen dieser Evaluation genutzten Datensätze zur Ampelerkennung sind in Tabelle 3.4 in einer Übersicht vergleichend dargestellt. Für die Evaluation der Gruppierung von Objektklassen in einem Lernproblem zur Hierarchieerzeugung wurde der [German Traffic Sign Recognition Benchmark \(GTSRB\)](#)-Datensatz zur Klassifikation von Verkehrszeichen verwendet, welcher nachfolgend ebenfalls vorgestellt wird.

	DeepTLR	BSTLD	HDTLR
Auflösung [B×H]	1.280 × 960	1.280 × 720	1.280 × 960
Herkunft der Daten	D	USA	D
Erstellungsdatum	2015	2017	2018
Bilder	10.910	13.927	9.139
Annotationen	14.475	24.249	11.651
Klassen	Z	Z, (P)	Z, P

Tabelle 3.4: Vergleich der verschiedenen Datensätze, welche im Rahmen dieser Evaluation zum Training und zur Beurteilung der Ampelerkennungsalgorithmen verwendet wurden. Die Anzahl der Klassen und Bilder enthält keine augmentierten Daten. Bei der Beschreibung der Klassen steht Z für den Zustand und P für das Piktogramm. [BSTLD](#) enthält lediglich im Trainingsdatensatz Piktogramme.



Abbildung 3.10: Beispielhafte Bildausschnitte annotierter Ampeln aus den Testdaten des DeepTLR-Datensatzes. Erstveröffentlichung in: [13]

DeepTLR-Datensatz

Der **Deep Traffic Light Recognition (DeepTLR)**-Datensatz [13] wurde im Rahmen dieser Arbeit erstellt. Als Basis hierfür dienten Aufnahmen aus den Jahren 2013 und 2014, welche mit dem FZI-Forschungsfahrzeug **CoCar** [141] im Großraum Karlsruhe aufgezeichnet wurden. Die einzelnen Bilder haben hierbei die Größen 1.280×960 und 1.280×640 Pixel. Um alle Bilder auf eine einheitliche Bildgröße anzupassen, wurden für alle Bilder mit einer Höhe von 640 Pixeln jeweils am oberen und unteren Bildrand weiße Balken ergänzt. Hierbei wurde bewusst eine Ergänzung in weißer Farbe gewählt, da sie für das Lernproblem der Ampelerkennung eine untergeordnete Rolle spielt. Bei einer Ergänzung der Bilder mit schwarzen Balken könnten z. B. unbeabsichtigt kreisförmige grüne Objekte am oberen Bildrand zu Strukturen mit einer gewissen visuellen Ähnlichkeit zu einer grünen Ampel ergänzt werden.

Der Datensatz besteht aus 10.910 Bildern, welche in einen Trainingsdatensatz mit 10.433 Bildern und einen Testdatensatz mit 377 Bildern aufgeteilt sind. Beispielhafte Bildausschnitte mit annotierten Ampeln sind in Abbildung 3.10 dargestellt. Für die Ampeln existieren 4 verschiedene Klassen, welche mit $Z \in \{ \text{Rot}, \text{Gelb}, \text{Grün}, \text{Rot-Gelb} \}$ die möglichen Zustände einer Ampel in Deutschland repräsentieren. Die Verteilung der Annotationen auf die einzelnen Klassen ist in Tabelle 3.5 dargestellt. Bei Betrachtung der Verteilung fällt auf, dass für die Klassen *Gelb* und *Rot-Gelb* nur wenige Objektinstanzen vorhanden sind. Diese Ampelzustände sind in Aufnahmen relativ selten zu finden.

Für die Verwendung zur Evaluation wurde der Datensatz durch Augmentierung der Bilder erweitert. Hierzu wurden die Bilder anhand der vertikalen Achse ge-

	Training	Training %	Test	Test %
Rot	5.081	36,91	215	30,20
Gelb	219	1,59	29	4,07
Grün	8.172	59,38	435	61,10
Rot-Gelb	291	2,11	33	4,63
Σ	13.763	100,00	712	100,00

Tabelle 3.5: Verteilung der Ampeln im [DeepTLR](#)-Datensatz auf die verschiedenen Klassen.

spiegelt. Zusätzlich wurden verschiedene perspektivische Verzerrungen durchgeführt. Für die signifikant unterrepräsentierten Klassen wurde eine deutlich größere Zahl an Verzerrungen durchgeführt, um so das Verhältnis zwischen den auftretenden Objektinstanzen der einzelnen Klassen teilweise auszugleichen. Zusätzlich wurde der Trainingsdatensatz in einen Trainings- und einen Validierungsdatensatz aufgeteilt. Hierbei wurde berücksichtigt, dass die verschiedenen augmentierten Instanzen eines Bildes im selben Datensatz verortet wurden. Die finalen im Rahmen dieser Evaluation verwendeten Datensätze sowie die Klassenverteilung nach der Augmentierung sind in Tabelle 3.6 zu finden.

Datensatz	Bilder	Ampeln	Grün	Rot	Gelb	Rot-Gelb
Training	145.180	203.658	108.018	66.612	14.642	14.386
Validierung	1.053	1.445	851	549	21	24
Test	377	713	436	215	29	33

Tabelle 3.6: Verteilung der Ampelklassen auf die Teildatensätze von [DeepTLR](#) nach erfolgter Augmentierung. Diese Aufteilung wurde im Rahmen der nachfolgenden Experimente verwendet.

HDTLR-Datensatz

Der [Hierarchical Deep Traffic Light Recognition \(HDTLR\)](#)-Datensatz [8] wurde, wie bereits der [DeepTLR](#)-Datensatz, im Rahmen dieser Arbeit erstellt. Als Basis hierfür dienten Aufnahmen, welche ebenfalls mit dem [FZI](#)-Forschungsfahrzeug [CoCar](#) [141] im Großraum Karlsruhe aufgezeichnet wurden. Die für den Datensatz gewählten Aufnahmen stellen eine charakteristische Auswahl für den Task der Ampelerkennung aus dem Pool dieser Aufzeichnungen dar. Der Datensatz besteht insgesamt aus 9.139 Bildern der Größe 1.280×960 Pixel. Die Annotationen für diesen Datensatz bestehen aus einer Klasse $K \in \{ \text{Ampel} \}$, einem Zustand $Z \in \{ \text{Rot}, \text{Gelb}, \text{Grün}, \text{Rot-Gelb} \}$ und einem Piktogramm $P \in \{ \text{Kein}, \text{Fußgänger}, \text{Pfeil-links}, \text{Pfeil-gerade}, \text{Pfeil-rechts} \}$.

	# Bilder	# Ampeln
Trainingsdaten	45.458	125.424
Validierungsdaten	1.118	1.144
Testdaten	1.110	916

Tabelle 3.7: Aufteilung des **HDTLR**-Datensatzes in die im Rahmen dieser Evaluation verwendeten Teildatensätze. Die Zahlen beziehen sich bereits auf die augmentierten Daten. Hierbei ist zu beachten, dass die Testdaten auch einige Bilder ohne Ampeln enthalten und die Anzahl der Ampeln deshalb die Anzahl der Bilder unterschreitet.

Zusätzlich wurden einige Regionen in den Trainingsdaten mit der Klasse *unknown* annotiert. Diese Daten werden beim Training der Algorithmen im Normalfall nicht betrachtet. Mit dieser Klasse wurden Bereiche im Bild annotiert, welche den Trainingsprozess behindern könnten. Dies können z. B. Signalanlagen für den ÖPNV oder sehr kleine Ampeln sein. Im Rahmen dieser Arbeit wurden auch die vorhandenen Fußgängerampeln nicht verwendet und daher der Klasse *unknown* zugerechnet. Die Bilder wurden analog zum **DeepTLR**-Datensatz durch Spiegelung an der vertikalen Achse und perspektivische Verzerrung mit sechs unterschiedlichen Stützpunktkonfigurationen augmentiert. Ebenfalls wurden die Bilder des Trainingsdatensatzes in einen Trainings- und einen Validierungsdatensatz aufgeteilt. Die im Rahmen dieser Evaluation verwendeten, finalen Datensätze sind in Tabelle 3.7 charakterisiert. Die Verteilung der Ampeln auf die einzelnen Klassen ist in Abbildung 3.11 dargestellt.

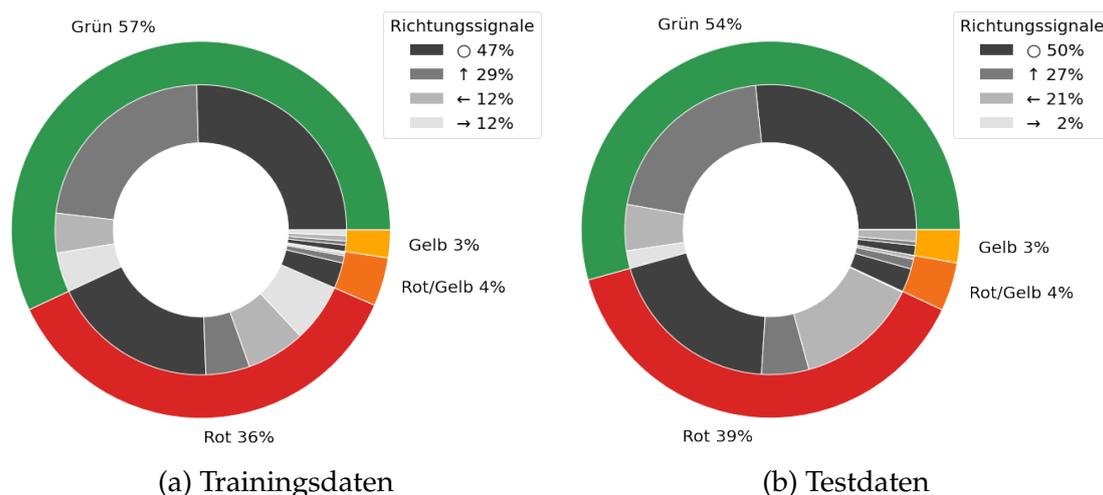


Abbildung 3.11: Verteilung der Ampeln im **HDTLR**-Datensatz auf die verschiedenen Zustände und Piktogramme.

Bosch Small Traffic Lights Datensatz

Der [Bosch Small Traffic Lights Datensatz \(BSTLD\)](#) [44] stammt von Bosch Nordamerika und besteht aus Straßenszenen aus den USA. Der Fokus des Datensatzes liegt, wie der Name bereits andeutet, auf der Erkennung von im Bild nur klein sichtbaren Ampeln. Aufgrund ihrer Größe von oft nur wenigen Pixeln stellen diese für aktuelle Detektoren eine besondere Herausforderung dar. Der Datensatz enthält 13.427 Bilder in der Größe 1.280×720 Pixel. Diese sind aufgeteilt in einen Trainingsdatensatz mit 5.093 Bildern und einen Testdatensatz mit 8.334 Bildern. Da es sich bei diesem Datensatz nicht um die Grundlage für einen öffentlichen Benchmark handelt, sind Annotationen auch für den Testdatensatz verfügbar.

	Minimum	Arithm. Mittel	Median	Maximum
Breite	1,875	9,430	8,500	48,375
Höhe	3,250	26,745	24,500	104,500
Fläche	11,718	313,349	212,109	4734,000

Tabelle 3.8: Größe der Ampeln im [BSTLD](#)-Testdatensatz in Pixeln.

Im Trainingsdatensatz sind 10.756 annotierte Ampeln mit einer Breite von 8,6 Pixeln im Median enthalten. Die annotierten Bilder wurden einem fortlaufenden Video im Abstand von 2 Sekunden entnommen. Details zur Größe der Ampeln in beiden Datensätzen sind in [Tabelle 3.8](#) zu finden. Für die Ampeln existieren 13 verschiedene Klassen, welche einerseits mit $Z \in \{ \textit{Rot}, \textit{Gelb}, \textit{Grün}, \textit{Aus} \}$ die Zustände der Ampeln abdecken und andererseits mit $P \in \{ \textit{Kein}, \textit{Gerade}, \textit{Gerade-links}, \textit{Links}, \textit{Rechts}, \textit{Gerade-rechts} \}$ auch einige Piktogramme für Richtungsanzeigen enthalten. Eine Klasse besteht hierbei immer aus einem Zustand $z \in Z$ und einem Piktogramm $p \in P$. Ein Überblick über die Verteilung der annotierten Ampeln auf die einzelnen Klassen ist in [Tabelle 3.9](#) dargestellt. Hierbei fällt auf, dass sich ein Großteil der Ampeln auf wenige Klassen verteilt. So fallen ca. 77% der Ampeln in die Klassen *Rot* oder *Grün*, jeweils ohne Piktogramm. 7 der 13 Klassen sind mit maximal je 20 Objekten nahezu nicht vertreten. Von den Piktogrammen ist lediglich der Pfeil nach links in einer nennenswerten Anzahl vertreten. Für ein Training der Piktogrammklassen erscheint der [BSTLD](#) daher wenig geeignet. 170 der enthaltenen und annotierten Ampeln sind teilweise verdeckt. Dies entspricht ca. 1,6% der enthaltenen Objekte.

Der Testdatensatz des [BSTLD](#) besteht aus 8.334 fortlaufenden Bildern, welche mit 15 FPS aufgezeichnet wurden. Es sind 13.493 annotierte Ampeln mit einer Breite von 8,5 Pixeln im Median enthalten. Nachdem bereits im Trainingsdatensatz die Piktogrammklassen deutlich unterrepräsentiert waren, sind diese im Testdatensatz überhaupt nicht enthalten. So enthält dieser lediglich vier Klassen, welche der Menge Z aus dem Trainingsdatensatz entsprechen. Von den annotierten Objekten sind 2.088 Ampeln teilweise verdeckt, was einem Anteil von ca. 15,5% aller

3 Statische Verkehrsobjekte: 2D Objektdetektion

	Training	Training %	Test	Test %
Rot	3.057	28,42	5.321	39,44
Rot Gerade	9	0,08	0	-
Rot Gerade-links	1	0,01	0	-
Rot Links	1.092	10,15	0	-
Rot Rechts	5	0,05	0	-
Gelb	444	4,13	154	1,14
Grün	5.207	48,41	7.569	56,10
Grün Gerade	20	0,19	0	-
Grün Gerade-links	1	0,01	0	-
Grün Gerade-rechts	3	0,03	0	-
Grün Links	178	1,65	0	-
Grün Rechts	13	0,12	0	-
Aus	726	6,75	449	3,33
Σ Rot	4.164	38,71	5.321	39,44
Σ Gelb	444	4,13	154	1,14
Σ Grün	5.422	50,40	7.569	56,10
Σ Aus	726	6,75	442	3,33
Σ	10.756	100,00	13.486	100,00

Tabelle 3.9: Verteilung der Ampeln im [BSTLD](#) auf die verschiedenen Klassen.

annotierten Ampeln entspricht. Dies ist kritisch zu sehen, da der Trainingsdatensatz mit 1,6% verdeckten Objekten einen deutlich geringeren Anteil aufweist.

Zur Verwendung in der Evaluation dieser Arbeit wurde der Datensatz augmentiert und ein Mapping der Klassen vorgenommen. Aus den im Verlauf der Beschreibung erwähnten Gründen wurden die Piktogramme im Datensatz nicht verwendet. Hierbei wurden alle Annotationen mit Piktogramm den jeweiligen Annotationen mit gleichem Zustand ohne Piktogramm zugeordnet. Die finale Verteilung der Annotationen über die einzelnen Objektklassen ist in [Tabelle 3.10](#) dargestellt.

Die Bilder des Trainingsdatensatzes wurden im Verhältnis 9:1 in einen Trainings- und einen Validierungsdatensatz aufgeteilt. Die Bilder dieser beiden Datensätze wurden durch zwei Augmentierungstechniken weiter angereichert, der Testdatensatz blieb unverändert. So wurde jedes Bild durch verschiedene perspektivische Verzerrungen verzerrt und zusätzlich an der vertikalen Achse gespiegelt.

Datensatz	Bilder	Ampeln	Grün	Rot	Gelb	Aus
Training	39.727	135.124	68.160	52.256	5.546	9.162
Validierung	4.415	15.097	7.599	5.854	656	988
Test	7.147	13.486	7.569	5.321	154	442

Tabelle 3.10: Verteilung der Ampelklassen auf die Teildatensätze von [BSTLD](#) nach erfolgter Augmentierung. Diese Aufteilung wurde im Rahmen der nachfolgenden Experimente verwendet.

Die daraus resultierende Verteilung der Bilder und Annotationen kann aus Tabelle 3.10 entnommen werden.

German Traffic Sign Recognition Benchmark (GTSRB)

Der [German Traffic Sign Recognition Benchmark \(GTSRB\)](#)-Datensatz [217] ist ein Datensatz zur Klassifikation von deutschen Verkehrszeichen in Bildausschnitten. Er enthält über 50.000 Bilder mit insgesamt 43 Klassen verbreiteter deutscher Verkehrszeichen. Die Bilder stammen aus Aufzeichnungen, die 2010 in der Umgebung von Bochum aufgenommen wurden. Es werden verschiedene Umgebungen von städtisch bis Autobahn abgedeckt, wobei in den Aufnahmen verschiedene Beleuchtungsszenarien abgedeckt sind. Der Datensatz ist aufgeteilt in einen Trainingsdatensatz mit 39.209 annotierten Bildern und einen Testdatensatz mit 12.630 Bildern. Die Annotationen für den Testdatensatz sind nicht veröffentlicht. Die einzelnen Bildausschnitte haben unterschiedliche Größen, wobei sich Breite und Höhe meist zwischen 20 und 60 Pixeln bewegen. Die Ausschnitte sind hierbei nicht notwendigerweise quadratisch. Minimal messen die Bildausschnitte hierbei 16 Pixel und maximal 128 Pixel an der jeweils langen Seite.

3.4.2 Evaluationsmetriken

Die Evaluation der vorgestellten Ansätze erfolgt im Rahmen dieser Arbeit einzelbildbasiert. Hierbei wird in jedem einzelnen Eingabebild des Evaluationsdatensatzes zunächst die Objekterkennung durchgeführt und diese nachfolgend mit den tatsächlich vorhandenen Objekten verglichen. So stehen einer Menge an aktuell erkannten Objekten D die Menge an real vorhandenen Objekten A aus den Annotationsdaten gegenüber. Um die Güte des verwendeten Algorithmus bestimmen zu können, muss zunächst eine Zuordnung zwischen den Objekten dieser Mengen bestimmt werden. Nur so kann erkannt werden, ob detektierte Objekte wirklich im Bild vorhanden sind und im Bild vorhandene Objekte detektiert wurden.

3 Statische Verkehrsobjekte: 2D Objektdetektion

Sowohl die Detektionen als auch die Annotationen liegen allerdings in Form von Bounding Boxen, also minimal umschließenden Rechtecken, vor. Hierbei entsteht die Herausforderung, dass selbst kleine Positionsunterschiede zwischen Detektion und Annotation eine einfache Zuordnung anhand einer Gleichheitsrelation unmöglich machen. Aus diesem Grund müssen die Objekte anhand ihrer Ähnlichkeit zugeordnet werden. Um dies erreichen zu können, muss zunächst ein Maß für die Ähnlichkeit zweier Objekte definiert werden. Bekannte Benchmarks zur Objektdetektion wie [PASCAL VOC](#) [80] verwenden hierzu die [Intersection over Union \(IoU\)](#) als Vergleichsmetrik.

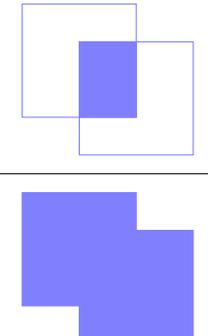
$$\text{IoU} = \frac{\text{Intersection}}{\text{Union}} = \frac{\text{Intersection}}{\text{Union}}$$


Abbildung 3.12: Die [Intersection over Union \(IoU\)](#)-Metrik zur Messung des Überlappungsgrades von zwei Bounding Boxen. Hierbei wird die Schnittmenge der Boxen im Verhältnis zu deren Vereinigung betrachtet.

Hierbei wird, wie in [Abbildung 3.12](#) dargestellt, die Überschneidung der beiden zu vergleichenden Objekte in Relation zu deren belegter Gesamtfläche betrachtet. Interpretiert man die einzelnen Objekte als Menge von Pixeln, entspricht die [IoU](#) dem Jaccard-Koeffizienten [125]. Bezogen auf die Bounding Boxen BB je einer Annotation $a \in A$ und einer Detektion $d \in D$ ist die [IoU](#) definiert als

$$\text{IoU}(BB_d, BB_a) = \frac{BB_d \cap BB_a}{BB_d \cup BB_a} \quad (3.10)$$

Basierend auf diesem Ähnlichkeitsmaß muss nun noch ein Schwellwert festgelegt werden, ab dem zwei Objektinstanzen als das selbe Objekt betrachtet werden. Anhand der definierten Vergleichsmetrik [IoU](#) und dem festgelegten Schwellwert kann nun eine Zuordnung zwischen den Objekten $d \in D$ und $a \in A$ bestimmt werden. Diese Zuordnung ist beidseitig injektiv, in beide Richtungen aber nicht notwendigerweise surjektiv. Hieraus ergeben sich direkt mehrere Metriken für die Einordnung von Annotationen und Detektionen:

True Positive (TP) bezeichnet eine Detektion d , welche einer Annotation a zugeordnet werden konnte. Dies entspricht einer korrekten Detektion.

False Positive (FP) steht für eine Detektion, die keiner Annotation zugeordnet werden konnte. Es wurde ein Objekt detektiert, welches nicht existiert.

False Negative (FN) bezeichnet eine Annotation, welcher keine Detektion zugeordnet werden konnte. Das im Bild vorhandene Objekt wurde durch den Detektor nicht erkannt.

True Negative (TN) steht für eine nicht existente Annotation, welche auch nicht detektiert wurde. Im Rahmen der Objektdetektion trifft dies auf beliebig viele mögliche Objekte im Bild zu, weshalb diese Metrik an dieser Stelle lediglich erwähnt wurde, um der Vollständigkeit zu genügen.

		Annotation (Wahrheit)	
		Positiv	Negativ
Ausgabe	Positiv	True Positive (TP)	False Positive (FP)
	Negativ	False Negative (FN)	True Negative (TN)

Abbildung 3.13: Die verschiedenen bei einer Klassifikation verwendeten Maßzahlen. Hierbei werden die durch einen Klassifikator ausgegebenen Klassen (links) mit den annotierten Klassen (oben) verglichen. Für den Fall einer binären Entscheidung $\in \{ \textit{Positiv}, \textit{Negativ} \}$ entspricht dieses Diagramm einer Konfusionsmatrix.

Eine Übersicht zur Einordnung der genannten Metriken ist in Abbildung 3.13 dargestellt. Zu beachten ist an dieser Stelle, dass bisher nur der Fall einer binären Klassifikation mit den Klassen $K \in \{ \textit{Objekt}, \textit{kein Objekt} \}$ betrachtet wurde. Im Falle der Detektion statischer Verkehrsobjekte soll nun eine weitere Metrik namens **False Classification (FC)** die korrekt detektierten, aber falsch klassifizierten Objekte erfassen. Diese zusätzliche Metrik ist notwendig, da bei einer Betrachtung der Klasse innerhalb der in Abbildung 3.13 zusammengefassten Metriken eine falsche Klassifikation zu einer doppelten Zählung führen würde. So würde die Detektion mit falscher Klassifikation als **FP** gezählt werden, da keine entsprechende Annotation vorhanden ist. Zusätzlich würde die eigentlich der Detektion zugehörige Annotation als **FN** gezählt werden. Zur Evaluation der in diesem Kapitel betrachteten hierarchischen Detektion werden die **FC** in den einzelnen Hierarchiestufen getrennt betrachtet. So ist eine Messung der Güte in den einzelnen Stufen leicht möglich. Die **FC** ergeben sich entsprechend aus den FC_h der einzelnen Hierarchiestufen h wie folgt:

$$FC = \sum_{h=1}^H FC_h \quad (3.11)$$

Mit den bisher vorgestellten Metriken ist ein Vergleich zwischen verschiedenen Detektoren allerdings nur sehr bedingt möglich, da hier gegensätzliche Aussagen der verschiedenen Metriken zu unklaren Ergebnissen führen können. Aus diesem Grund werden weitere, kombinierte Metriken verwendet, um eine bessere Vergleichbarkeit der Detektoren zu erreichen. Hierzu zählt der Anteil der korrekt detektierten Objekte (*TP*) unter allen detektierten Objekten – auch *Precision* genannt.

$$\text{Precision} = \frac{TP}{TP + FC + FP} \quad (3.12)$$

Die Precision ist also eine Maßzahl dafür, wie viele der detektierten Objekte tatsächlich vorhanden sind. Zusätzlich bestimmt wird der Anteil der korrekt detektierten Objekte unter allen Annotationen, der auch als *Recall* bezeichnet wird.

$$\text{Recall} = \frac{TP}{TP + FC + FN} \quad (3.13)$$

Der Recall sagt damit aus, wie viele der vorhandenen Objekte erkannt wurden. Mit Precision und Recall sind nun zwei aussagekräftigere, kombinierte Metriken verfügbar. Allerdings heben beide je eine Eigenschaft von Objektdetektoren hervor und so bleiben erneut zwei verschiedene Maßzahlen zur Bewertung. Für einen direkten Vergleich anhand einer einzigen Metrik wird zusätzlich der sogenannte *F1-Score* verwendet.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3.14)$$

Dieser kombiniert Recall und Precision zu einem einzigen Wert im Wertebereich $[0, 1]$ und ermöglicht so eine vergleichende Aussage zwischen verschiedenen Algorithmen. Die auf den einzelnen Bildern erhobenen Metriken werden im Rahmen der Gesamtevaluation auf dem entsprechenden Testdatensatz entsprechend ihrer Charakteristika mittels Summen- oder Mittelwertbildung kombiniert.

Neben der Detektionsgüte spielt auch die Laufzeit der bewerteten Algorithmen eine wesentliche Rolle. Diese setzt sich zusammen aus der reinen Ausführungszeit des *CNN* und der Laufzeit der Nachverarbeitung. Für die einzelnen Zeiten wird der Mittelwert, die Standardabweichung sowie die maximal beobachtete Laufzeit angegeben. So kann eine obere Abschätzung durchgeführt werden.

3.4.3 Ampeldetektion mit CNNs – DeepTLR

Die Evaluation des in dieser Arbeit in Abschnitt 3.3.2 vorgestellten Ansatzes zur Detektion von statischen Verkehrsobjekten durch einen CNN-basierten Ansatz wurde anhand des Anwendungsfalles der Ampelerkennung durchgeführt. Die konkret verwendete Architektur wird für diesen Anwendungsfall auch als **Deep Traffic Light Recognition (DeepTLR)** bezeichnet. Zur Ermittlung der generellen Leistungsfähigkeit wurden zwei verschiedene Experimente durchgeführt. Diese unterscheiden sich im Wesentlichen durch die im zugrundeliegenden Lernproblem definierten Klassen.

So wurde für das erste Experiment eine Zweiklassendetektion mit den Klassen $K \in \{ \text{Rot}, \text{Grün} \}$ durchgeführt. Im zweiten Experiment wurde die Anzahl der Klassen ausgeweitet, um alle Zustände einer Ampel abdecken zu können. Somit wurde $K \in \{ \text{Rot}, \text{Grün}, \text{Rot-Gelb}, \text{Gelb} \}$ für dieses Experiment verwendet. Beide Experimente wurden auf den Daten des in Abschnitt 3.4.1 beschriebenen **DeepTLR**-Datensatzes durchgeführt. Die Architektur des **DeepTLR**-Ansatzes basiert in beiden Fällen auf dem AlexNet-Encoder.

Rot-Grün-Detektion

Zur Evaluation der Leistungsfähigkeit des **DeepTLR**-Ansatzes wurden in einem ersten Experiment Modelle zur Detektion von roten und grünen Ampeln trainiert. Dies hat im Wesentlichen zwei Gründe: Die Klassen, welche die anderen Zustände einer Ampel repräsentieren, sind im **DeepTLR**-Datensatz aufgrund ihres selteneren Auftretens deutlich unterrepräsentiert. Zusätzlich war von Beginn an zu erwarten, dass der initiale Ansatz mit einer steigenden Anzahl an gleichartigen Klassen eine geringere Detektionsgüte erreicht. So wurde zunächst ein reduziertes Zweiklassenproblem zur Evaluation der generellen Detektionsgüte für Ampeln gewählt.

Zum Training der Modelle wurden zwei verschiedene Versionen des **DeepTLR**-Datensatzes verwendet. In der ursprünglichen Version des Datensatzes wurden alle Objekte mit einer Breite von unter 4 Pixeln herausgefiltert. Für dieses Experiment wurde ein weiterer Datensatz erzeugt, bei dem zusätzlich auch alle Objekte mit einer Breite zwischen 4 und 8 Pixeln entfernt wurden. So sollte der Einfluss dieser Objekte auf den Trainingsprozess untersucht werden. Der Testdatensatz ist davon nicht betroffen und für beide Datensätze identisch.

Zur Evaluation verschiedener Konfigurationen wurden vier verschiedene Modelle trainiert. Zunächst wurde eine Basiskonfiguration erstellt, welche lediglich den Datensatz mit Objekten einer Breite von 8 und mehr Pixeln enthält. Auch wurden hierbei für das Modell keine vortrainierten Gewichte zur Initialisierung der Filterkerne verwendet. In einem zweiten Modell wurden diese vortrainierten Gewichte zur Initialisierung verwendet. Ein drittes Modell wurde auf dem

vollen Datensatz mit den kleinen Objekten trainiert. Da beide Variationen zu Verbesserungen gegenüber der Basiskonfiguration führten, wurde ein finales **DeepTLR**-Modell mit allen Variationen trainiert. Das Training der Modelle fand auf verkleinerten Bildern der Auflösung 640×480 Pixel statt. Die Evaluation wurde für jedes Modell sowohl auf dieser Auflösung als auch auf der Ursprungsauflösung durchgeführt. Die einzelnen Ergebnisse sind in Tabelle 3.11 aufgeführt.

Modell	Auflösung	TP	FP	FN	FC	Precision	Recall	F1
Basiskonfig.	640 x 480	413	42	80	132	70,4%	66,1%	68,2%
Pretrained		427	38	58	140	70,6%	68,3%	69,4%
Kleine Ampeln		546	99	29	50	78,6%	87,4%	82,7%
DeepTLR		567	59	22	36	85,6%	90,7%	88,1%
Basiskonfig.	1.280 x 960	457	15	153	15	93,8%	73,1%	82,2%
Pretrained		488	21	131	6	94,8%	78,1%	85,6%
Kleine Ampeln		569	38	56	0	93,7%	91,0%	92,4%
DeepTLR		571	25	53	1	95,6%	91,4%	93,5%

Tabelle 3.11: Ergebnisse für die verschiedenen Modelle der Rot-Grün-Detektion auf dem **DeepTLR**-Datensatz.

Es ist klar zu erkennen, dass die zusätzliche Verwendung kleiner Objekte einen deutlich positiven Einfluss auf die Ergebnisse des entsprechenden Modells in beiden Auflösungen hatte. Einen geringen positiven Effekt konnte auch die Initialisierung der Gewichte durch ein auf ImageNet [200] trainiertes Klassifikationsnetz beitragen. Dies war ebenfalls für beide Auflösungen zu beobachten. Eine kombinierte Variante konnte die vorherigen Modelle in beiden Auflösungen nochmals übertreffen. Ein weiteres Ergebnis ist die Überlegenheit von den auf der geringen Auflösung trainierten Modellen auch auf der höheren Auflösung. Trotz eines Trainings auf einer lediglich reduzierten Auflösung konnten auf der höheren Auflösung sehr gute Ergebnisse erzielt werden.

Mehrklassen-Detektion

Zur Evaluation der Mehrklassenfähigkeit des vorgestellten **DeepTLR**-Ansatzes in der finalen Konfiguration aus dem ersten Experiment wurde in einem weiteren Experiment ein Modell für alle Ampelklassen trainiert. Da in den ursprünglichen **DeepTLR**-Daten nur wenige Ampeln der Klassen *Gelb* und *Rot-Gelb* vorhanden waren, wurde durch intensive Augmentierung zumindest das Ungleichgewicht zwischen den Klassen vermindert. Die Ergebnisse des trainierten Modells sind in Tabelle 3.12 dargestellt.

Es ist klar erkennbar, dass das trainierte Modell nicht in der Lage war, eines der Objekte der neuen Klassen korrekt zu erkennen. Der Zustand *Gelb* wurde meist

	Hintergrund	Grün	Rot	Gelb	Rot-Gelb
Hintergrund	x	81	8	0	0
Grün	17	418	0	0	0
Rot	24	31	160	0	0
Gelb	3	17	9	0	0
Rot-Gelb	4	2	27	0	0

Tabelle 3.12: Konfusionsmatrix für die Mehrklassendetektion von [DeepTLR](#) mit geringerer Auflösung auf dem [DeepTLR](#)-Datensatz. Die Zeilen enthalten die Ausgabe, in den Spalten sind die annotierten Klassen dargestellt. In der Zeile Hintergrund sind [False Negative](#) Objekte eingetragen.

fälschlicherweise als Zustand *Grün* erkannt und der Zustand *Rot-Gelb* dem Zustand *Rot* zugeordnet. In den jeweiligen Wahrscheinlichkeitsverteilungen ist hierbei meist, wie in [Abbildung 3.7](#) dargestellt, eine gewisse Wahrscheinlichkeit für die korrekte Klasse zu erkennen. Die Klassen *Rot* und *Grün* sind an dieser Stelle allerdings dominant.

3.4.4 Hierarchische Ampeldetektion – HDTLR

Die in [Abschnitt 3.3.3](#) eingeführte Architektur [HDTLR](#) zur hierarchischen Detektion statischer Verkehrsobjekte soll nun am Beispiel der Ampeldetektion evaluiert werden. Hierzu wurde zunächst der hierarchische [HDTLR](#)-Ansatz dem nicht hierarchischen [DeepTLR](#)-Ansatz vergleichend gegenübergestellt. Für diese Evaluation wurde der in [Abschnitt 3.4.1](#) eingeführte [DeepTLR](#)-Datensatz verwendet. Für die Evaluation wurde der Originaldatensatz mit einer Bildgröße von 1.280×960 Pixeln, sowie eine Version mit einer reduzierten Bildgröße von 640×480 Pixeln verwendet. So konnten die Auswirkungen einer verringerten Auflösung auf die beiden Ansätze untersucht werden. Für beide Verfahren wurden mit Ausnahme der Detektionshierarchie identische Netzwerkarchitekturen auf Basis von AlexNet [\[144\]](#) verwendet. Auch die Parameter zum Training der Netze und Clustering der Ausgaben wurden identisch gewählt.

Das zur Evaluation verwendete Lernproblem enthält die 4 Klassen $K \in \{ Rot, Grün, Gelb, Rot-Gelb \}$, wobei für die hierarchische Detektion zusätzlich die Klasse *Ampel* eingeführt wurde. Somit werden zwar die Möglichkeiten der vorgestellten hierarchischen Detektion noch nicht komplett genutzt, allerdings ist aus bisherigen Experimenten bekannt, dass dieses Lernproblem für den klassischen [DeepTLR](#)-Ansatz bereits herausfordernd ist. So kann ein Einfluss der hierarchischen Detektion bereits mit diesem einfachen Beispiel gezeigt werden. Die für dieses Experiment verwendete Hierarchie besitzt zwei Stufen. Stufe 1 kennt nur die Klassen *Ampel* und *Hintergrund*, wobei für die Klasse *Hintergrund* keine Bounding Boxen geschätzt werden. In Stufe 2 besitzt die Klasse *Ampel* als Kindknoten

3 Statische Verkehrsobjekte: 2D Objektdetektion

die genannten Zustandsklassen. Die Ergebnisse dieses Vergleichs sind in Tabelle 3.13 dargestellt.

Modell	Auflösung	TP	FP	FN	FC	Precision	Recall	F1
DeepTLR [13]	640 x 480	578	89	48	86	76,8%	81,2%	78,9%
HDTLR [8]		621	93	65	26	84,0%	87,2%	85,5%
DeepTLR [13]	1.280 x 960	514	17	197	1	96,6%	72,2%	82,6%
HDTLR [8]		682	52	24	6	92,2%	95,8%	94,0%

Tabelle 3.13: Vergleich von DeepTLR mit dem hierarchischen HDTLR mit identischen Encodern auf zwei verschiedenen Auflösungen.

Es ist zu erkennen, dass der F1-Score in beiden Bildauflösungen für den hierarchischen Ansatz dem nicht hierarchischen Ansatz deutlich überlegen ist. Bei dem Experiment in hoher Eingabeauflösung beträgt hier der Unterschied sogar 11 Prozentpunkte. Für die geringere Auflösung kann die Hierarchie auch die Fehlklassifikationen deutlich reduzieren. Hierbei ist für die hohe Auflösung auffällig, dass der hierarchische Ansatz deutlich mehr False Positive (FP) und geringfügig mehr False Classification (FC) verursacht. Dies lässt sich allerdings erklären durch die deutlich geringere Anzahl an nicht erkannten Ampeln. So ist anzunehmen, dass hier das zur Begründung eines hierarchischen Ansatzes dargestellte Problem der verteilten Wahrscheinlichkeiten verstärkt auftritt und so für viele Ampeln der Klassifikationsschwellwert der einzelnen Regionen nicht mehr erreicht wird. Die Ergebnisse zeigen auch klar, dass eine Detektion auf einer höheren Auflösung eine deutliche Steigerung der Detektionsgüte ermöglicht.

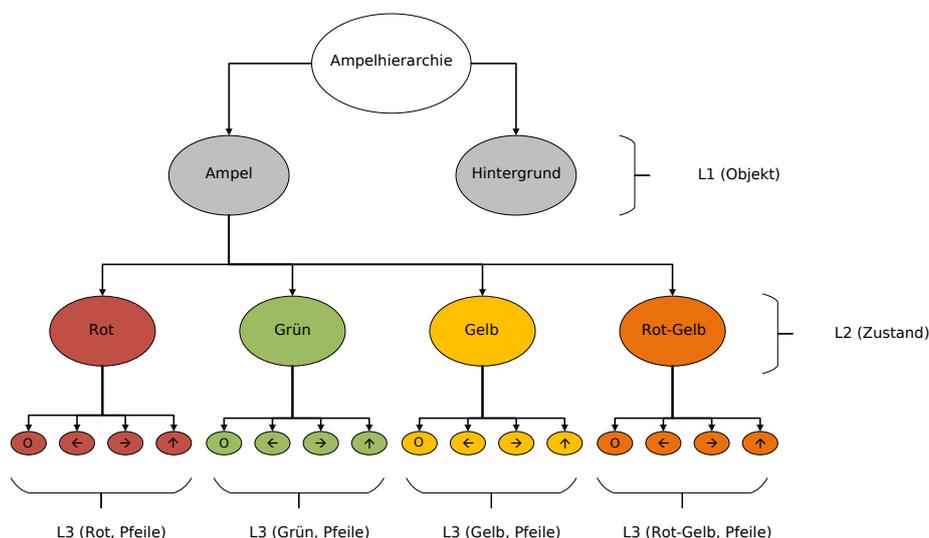


Abbildung 3.14: Dreistufige Hierarchie für das Lernproblem der Ampelerkennung mit Erkennung von Zustand und Piktogrammen. Ähnlich veröffentlicht in: [8]

In einem weiteren Experiment wurde der hierarchische Ansatz mit einer vollen, aus drei Stufen bestehenden, Ampelhierarchie und verschiedenen Encodern zur Extraktion der Features evaluiert. Die verwendete Hierarchie ist in Abbildung 3.14 dargestellt. Um sämtliche Hierarchiestufen der Architektur trainieren zu können, wurde der in Abschnitt 3.4.1 eingeführte **HDTLR**-Datensatz erstellt. Auf diesem Datensatz wurden Modelle mit den Encoder-Architekturen AlexNet [144], VGG16 [212] und GoogLeNet [222] trainiert und evaluiert. Hierbei wurden, wie bereits im vorherigen Experiment, die Untersuchungen für zwei verschiedene Eingabeauflösungen durchgeführt. Die Ergebnisse dieser Experimente sind vergleichend in Tabelle 3.14 dargestellt.

Modell	Auflösung	TP ₁	FP ₁	FN ₁	FC ₂	FC ₃	F1	H ₂	H ₃	Π
AlexNet		730	163	186	5	3	80,7%	99,3%	99,6%	79,8%
VGG	640 x 480	831	152	85	18	9	87,5%	97,8%	98,9%	84,7%
GoogLeNet		714	248	202	16	5	76,0%	97,8%	99,3%	73,8%
AlexNet		768	158	148	2	12	83,4%	99,7%	98,4%	81,9%
VGG	1.280 x 960	876	181	40	5	27	88,8%	99,4%	96,9%	85,6%
GoogLeNet		718	243	198	19	8	76,5%	97,4%	98,9%	73,6%

Tabelle 3.14: Evaluation der hierarchischen Detektion und Vergleich verschiedener Architekturen zur Extraktion von Features auf dem **HDTLR**-Datensatz.

Auch hier ist zu erkennen, dass die Modelle auf der höheren Auflösung bessere Ergebnisse erzielen. Eine Ausnahme ist allerdings das auf GoogLeNet aufsetzende Modell, welches mit Abstand am schlechtesten abschneidet. Hier sind für die unterschiedlichen Eingabeauflösungen keine großen Unterschiede erkennbar. Es ist auch zu erkennen, dass die auf der VGG-Architektur aufbauenden Modelle in beiden Auflösungen überlegen sind. Das VGG-Modell der niedrigen Auflösung übertrifft sogar die Modelle der anderen Encoder-Architekturen der hohen Auflösung deutlich. Bezüglich der hierarchischen Detektion ist festzustellen, dass bei sämtlichen Modellen nur wenige Fehlklassifikationen der erkannten Objekte auftreten. So liegt der Anteil der korrekt klassifizierten Objekte in jeder Hierarchiestufe bei 97% bis knapp 100%.

Analyse der Laufzeiten

Für die in diesem Experiment trainierten Modelle wurde ebenfalls eine umfangreiche Laufzeitanalyse durchgeführt. Grundlage für diese Analyse war eine Nvidia GeForce GTX 980 Ti Grafikkarte sowie die Frameworks CUDA 7.5 und cuDNN 3.0.07 von Nvidia. Hierbei wurde für jedes Modell getrennt die Ausführungszeit des eigentlichen Netzes und die Ausführungszeit der Nachverarbeitung über den kompletten Testdatensatz hinweg gemessen. Es wurde jeweils die durch-

3 Statische Verkehrsobjekte: 2D Objektdetektion

schnittliche Dauer (μ), die Standardabweichung (σ) und die maximale Dauer ermittelt. Die Ergebnisse dieser Untersuchung sind in Tabelle 3.15 dargestellt.

Modell	Auflösung	Netzwerk (ms)			Post (ms)			Σ	
		μ	σ	max	μ	σ	max	μ	max
AlexNet	640 x 480	31,9	2,2	37,8	0,9	0,4	4,1	32,8	41,9
VGG		76,2	1,0	83,0	2,2	0,5	4,9	78,4	87,9
GoogLeNet		71,1	2,3	88,9	2,2	0,3	4,7	73,3	93,6
AlexNet	1.280 x 960	96,2	0,8	105,1	7,4	1,0	15,2	103,6	120,3
VGG		264,7	2,4	269,2	7,7	1,5	22,9	272,4	292,1
GoogLeNet		188,9	1,6	206,7	6,9	0,3	8,1	195,8	214,8

Tabelle 3.15: Laufzeiten der Modelle mit den verschiedenen Encodern auf dem [HDTLR](#)-Datensatz.

Bei Betrachtung der Netzlaufzeiten zwischen den verschiedenen Auflösungen ist festzustellen, dass diese nicht linear mit der Anzahl der verarbeiteten Pixel steigt. Die Zahl der Pixel ist zwischen geringer Auflösung und hoher Auflösung um den Faktor 4 erhöht. Für die verschiedenen Modelle schwankt dieser Faktor deutlich. Für die AlexNet-Modelle beträgt er 3, für die GoogLeNet-Modelle 2,7 und für die VGG-Modelle 3,5. Hier scheint mit VGG das größte Modell die [Graphics Processing Unit \(GPU\)](#) oder genauer gesagt den Speicher der [GPU](#) bereits stark auszulasten.

Betrachtet man die Anforderungen der Anwendungsdomäne an derartige Algorithmen, welche in Abschnitt 3.1 analysiert wurden, so ist dort eine Verarbeitungsgeschwindigkeit von unter 100 ms gefordert, um eine Ausführungsrate von 10 Hz erreichen zu können. Für die bislang nicht optimierten Modelle trifft dies im Wesentlichen auf die Modelle für die geringe Auflösung zu. Sämtliche dieser Modelle erfüllen die Anforderung, unter 100 ms zur Ausführung zu benötigen. Mit dem VGG-Modell ist hierbei auch bereits ein Modell vertreten, welches die Ergebnisse der anderen Encoder auch auf der hohen Auflösung übertrifft. Allerdings wäre eine Anwendung auf der höheren Auflösung wünschenswert, da hierdurch die Qualität der Ausgaben weiter gesteigert werden kann.

Hierarchische Klassifikation

Zur weiteren Bewertung der hierarchischen Klassifikation wurden die korrekt detektierten, aber falsch klassifizierten Objekte näher untersucht. Vor allem die Entfernung dieser falsch klassifizierten Objekte und damit ihre Größe im Eingabebild spielt eine entscheidende Rolle. Hier muss beachtet werden, dass ein Teil dieser Klassifikation die Erkennung von Piktogrammen innerhalb der einzelnen Ampelsegmente enthält. Bei einer Größe der gesamten Ampel im Eingabebild

von nur wenigen Pixeln ist dies nicht zu vernachlässigen. So steigt die Schwierigkeit dieser Klassifikation mit Abnahme der Objektbreite um jeden Pixel deutlich.

Um den Einfluss der Objektgröße im Bild auf eine mögliche Fehlklassifikation zu untersuchen, wurden die Ergebnisse auf dem **HDTLR**-Datensatz ausgewertet. Die Ergebnisse dieser Untersuchung sind in Abbildung 3.15 dargestellt. Es ist klar zu erkennen, dass die Klassifikationsfehler lediglich bei Objekten mit einer maximalen Breite von 10 Pixeln auftreten. Bei Objekten mit einer Größe von mehr als 10 Pixeln im Eingabebild konnte kein Klassifikationsfehler beobachtet werden. Die meisten Klassifikationsfehler treten bis zu einer Objektbreite von 6 Pixeln auf. Somit kann angenommen werden, dass Klassifikationsfehler im Wesentlichen bei weit entfernten Ampeln auftreten. Sobald sich die Entfernung zu den entsprechenden Ampeln verringert, können auch diese zuverlässig klassifiziert werden. Müssen auch Ampeln in größerer Entfernung zuverlässig klassifiziert werden, kann dies durch eine Anpassung des Kamera-Setups am Zielfahrzeug erreicht werden.

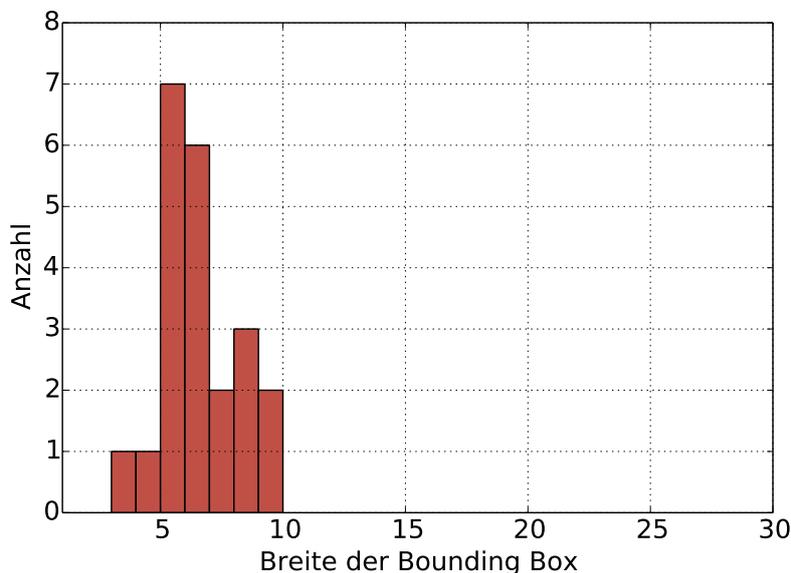


Abbildung 3.15: Analyse der durch die hierarchische Detektion detektierten, aber falsch klassifizierten Ampeln in Abhängigkeit ihrer Größe. Erstveröffentlichung in: [20]

Detektion kleiner Ampeln – BSTLD

Für eine Evaluation der Leistungsfähigkeit besonders bei kleinen Ampeln und einer vergleichenden Evaluation mit anderen Ansätzen wurde der vorgestellte hierarchische Detektor auf dem **BSTLD**-Datensatz evaluiert. Der Datensatz enthält mit den Ampelzuständen im Testdatensatz lediglich zwei Hierarchiestufen, weshalb eine um die Piktogrammebene bereinigte Klassenhierarchie verwendet

3 Statische Verkehrsobjekte: 2D Objektdetektion

wurde. Mit dieser Klassenhierarchie wurde ein hierarchisches VGG-Model trainiert und evaluiert. Die Ergebnisse dieser Evaluation sind in Tabelle 3.16 dargestellt. Auffällig ist hierbei, dass im Vergleich zu den Ergebnissen auf dem HDTLR-Datensatz bei vergleichbarem Recall eine um ca. 4 Prozentpunkte geringere Precision erreicht wird.

Auflösung	TP ₁	FP ₁	FN ₁	FC ₂	Precision	Recall	F1	H ₂
640 x 480	12.280	2.858	1.206	220	81,1%	91,1%	85,2%	98,2%

Tabelle 3.16: Ergebnisse der Evaluation des hierarchischen VGG-Modells auf dem BSTLD-Datensatz.

So scheint ein Training mit vielen sehr kleinen Objekten zu dem Ergebnis zu führen, dass sehr viele Strukturen durch den Detektor als Ampel fehlinterpretiert werden. Dies ist allerdings wenig verwunderlich, da es sich in diesem Fall um Objekte mit einer Breite von 2-3 Pixeln handelt. Mit dieser Anzahl an Pixeln kann die Struktur einer Ampel nicht ansatzweise adäquat abgebildet werden. Meist beschränkt sich die Abbildung auf einen hell-dunkel-hell Übergang bei hellem Hintergrund. Da solche Strukturen in Bildern sehr häufig vorkommen, ist eine fehlerhafte Erkennung von Objekten bei derartigen Trainingseingaben plausibel.

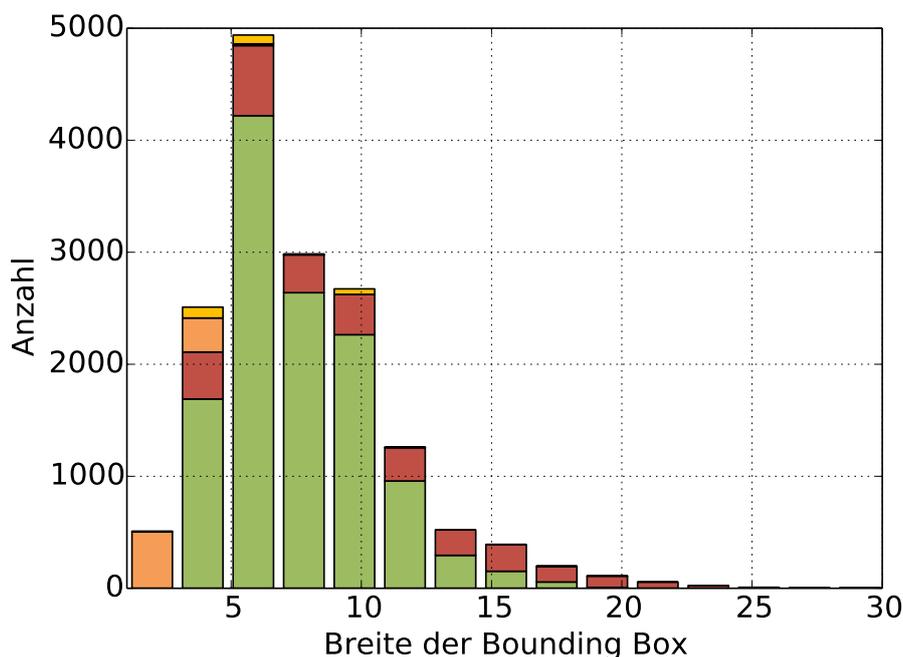


Abbildung 3.16: Güte des VGG-Detektors bei geringer Auflösung auf dem BSTLD-Datensatz abhängig von der Entfernung der Objekte. Grün steht für eine korrekte Detektion, rot für eine Fehldetektion, orange für eine nicht erkannte Ampel und gelb für eine falsche Klassifikation. Erstveröffentlichung in: [8]

Zur Einordnung der Zahlen wurde ebenfalls eine Analyse der Ergebnisse in Abhängigkeit zu der Größe der jeweiligen Bounding Box durchgeführt. Die Ergebnisse sind in Abbildung 3.16 dargestellt. Es ist klar zu erkennen, dass die nicht erkannten Ampeln (FN) im Wesentlichen weniger als 5 Pixel breit sind. Auch die falsch klassifizierten Ampeln sind meist maximal 6 Pixel breit. Alle falsch klassifizierten Ampeln hatten maximal eine Breite von 10 Pixeln, was mit den bisherigen Erkenntnissen auf dem HDTLR-Datensatz übereinstimmt. Bei den Fehldetektionen (FP) ist zu beobachten, dass diese über die komplette Breite der möglichen Ampelbreiten auftreten. Auch treten diese in Bereichen auf, in denen keine realen Ampeln mehr vorhanden sind. Somit erscheint ein Zusammenhang mit kleinen Objekten und zufälligen Strukturen im Bild wahrscheinlich.

Im Rahmen der Evaluation auf dem BSTLD wurden die Ergebnisse auch mit den Ergebnissen des Ansatzes von Behrendt et al. [44] verglichen. Da diese Ergebnisse allerdings lediglich in Form einer grafischen Recall-Precision-Kurve in der entsprechenden Publikation vorlagen, wurden die Werte aus der HDTLR-Evaluation in die entsprechende Grafik eingefügt. Da den finalen Objekten durch den HDTLR-Ansatz keine Wahrscheinlichkeiten zugewiesen werden, ist eine direkte Erzeugung einer entsprechenden Kurve nicht ohne weiteres möglich. Es wurden allerdings die für das VGG-Modell ermittelten Werte für Precision und Recall in der Grafik ergänzt und in Abbildung 3.17 dargestellt. Wie in der Abbildung zu erkennen ist, erreicht das hier vorgestellte hierarchische Modell bei gleicher Precision einen um ca. 20 Prozentpunkte höheren Recall. Die Klassifi-

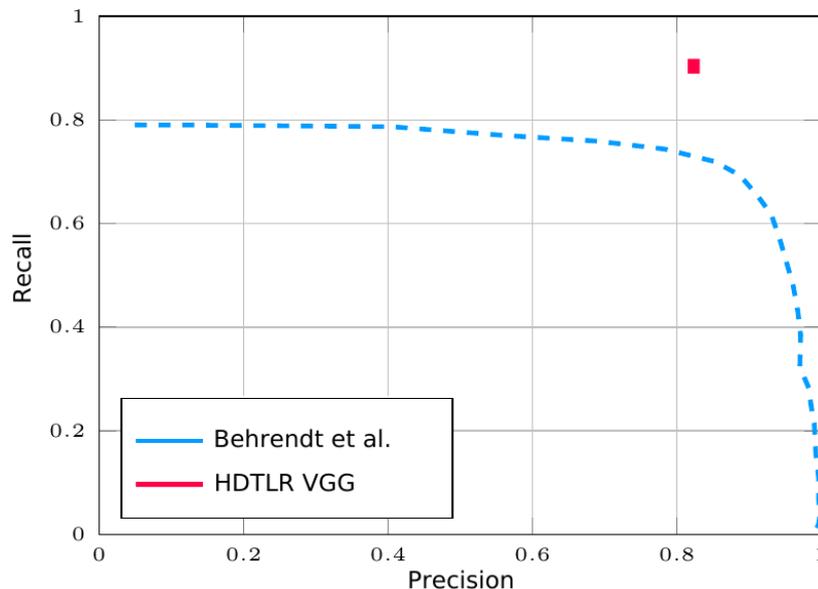


Abbildung 3.17: Vergleich zwischen dem HDTLR-VGG-Modell und dem Ansatz von Behrendt et al. [44]. Da in der Publikation von Behrendt et al. keine Werte angegeben sind, wurde die Kurve aus der Publikation übernommen und um die Ergebnisse von HDTLR ergänzt. Quelle: [44]

3 Statische Verkehrsobjekte: 2D Objektdetektion

kationsgenauigkeit der Ampelzustände ist bei Behrendt et al. mit 95,1 % angegeben, was durch den vorgestellten [HDTLR](#)-Ansatz mit 98,1 % ebenfalls übertroffen werden konnte.

3.4.5 Klassengruppierung zur Hierarchieerzeugung

Die Evaluation der vorgestellten Vorgehensweise zur Gruppierung von Objektklassen soll nachfolgend anhand des Lernproblems der Verkehrszeichenerkennung durchgeführt werden. Durch die Wahl dieses Lernproblems soll eine ausreichend große Menge an unterschiedlichen Klassen als Grundlage für die Gruppierung bereitgestellt werden. Beispielhaft für das Lernproblem der Verkehrszeichenerkennung wurde der in Abschnitt 3.4.1 vorgestellte [German Traffic Sign Recognition Benchmark \(GTSRB\)](#)-Datensatz verwendet. Die verwendete CNN-Architektur zur Klassifikation der Daten ist eine reduzierte Architektur mit Ähnlichkeit zu VGG13. Sie besteht aus 8 Convolution- und 3 MaxPooling-Schichten.

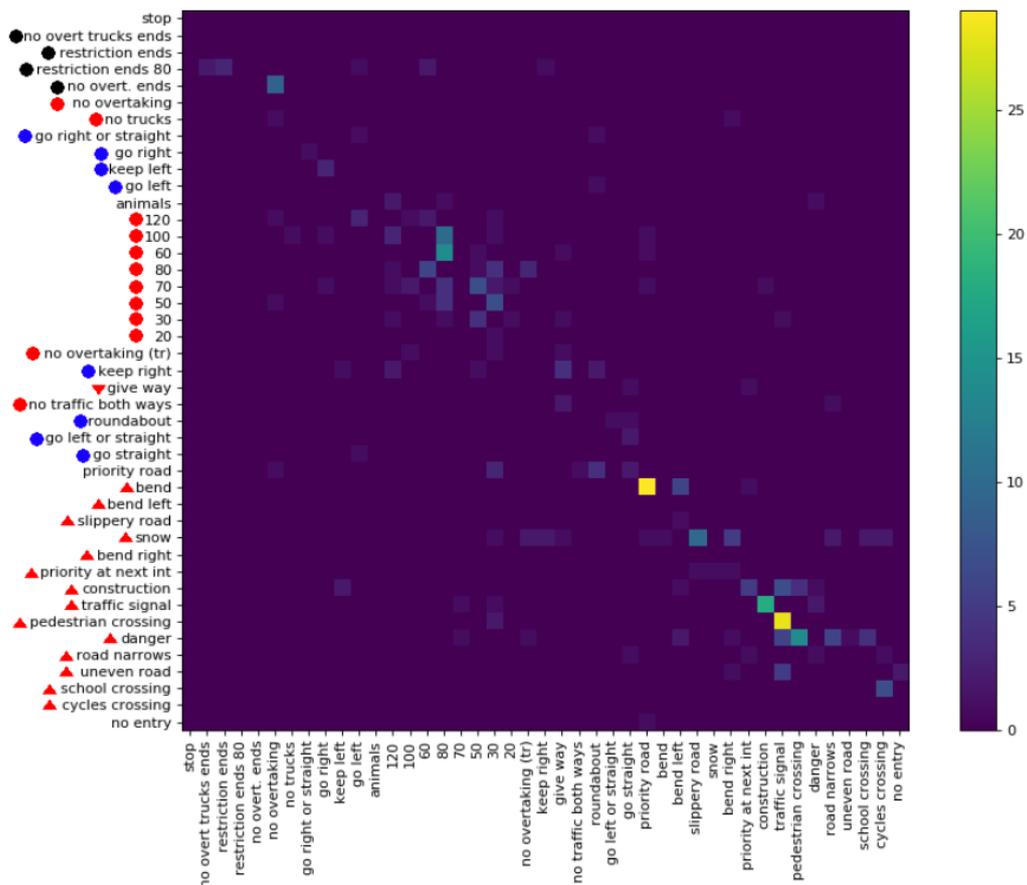


Abbildung 3.18: Konfusionsmatrix nach der Gruppierung der Objektklassen des Lernproblems der Verkehrszeichenerkennung. Erstveröffentlichung in: [31]

Die finalen Schichten sind ein Global Average Pooling, gefolgt von einem Softmax. Details zur Architektur sind in [31] zu finden.

Das Verfahren wurde durchgeführt, wie in Abschnitt 3.3.3 beschrieben und in Algorithmus 3.1 dargestellt. Die hieraus resultierende Konfusionsmatrix ist in Abbildung 3.18 abgebildet.

Die resultierende Ordnung der Klassen entspricht hierbei in weiten Teilen einer für dieses Lernproblem erwartbaren Gruppierung nach geometrischen Gesichtspunkten. Lediglich im mittleren Segment der Matrix scheinen sich die geometrischen Formen etwas zu durchmischen. Ein Blick auf die Klassifikationszahlen in der Matrix zeigt allerdings, dass in diesem Bereich nur wenige Objektinstanzen vorhanden waren und so eine gewisse Ungenauigkeit möglich ist.

Es ist allerdings klar zu erkennen, dass die Besetzung der resultierenden Matrix auf die Diagonale fokussiert ist. Somit erfüllt der vorgeschlagene Algorithmus sein Optimierungsziel. Auch scheint der Algorithmus geeignet, optisch ähnliche Klassen sinnvoll zu ordnen. Um dies folgern zu können, muss allerdings vorausgesetzt werden, dass eine Korrelation zwischen dem Empfinden für optische Ähnlichkeit von Betrachter und Algorithmus gegeben ist.

3.5 Diskussion

In diesem Kapitel wurde zunächst ein CNN-basierter Ansatz zur Detektion von statischen Verkehrsobjekten vorgestellt, welcher um eine Komponente zur hierarchischen Detektion erweitert wurde und so zur Detektion einer Vielzahl ähnlicher Objekte verwendet werden kann. Da ein besonderes Augenmerk bei der Detektion statischer Verkehrsobjekte auf deren meist geringer Abbildungsgröße in den zu untersuchenden Eingabebildern liegt, wurden zusätzlich Strategien zur Detektion kleiner Objekte vorgestellt.

Bereits die nicht hierarchische Architektur zur Detektion der zwei wichtigsten Ampelklassen *Rot* und *Grün* konnte in einer ersten Evaluation sehr gute Ergebnisse erzielen. Die im Rahmen des HDTLR-Ansatzes eingeführte hierarchische Detektion von statischen Verkehrsobjekten konnte in nachfolgenden Versuchen zur hierarchischen Detektion von Ampeln mit sämtlichen Ampelklassen sowie der zusätzlichen Erkennung der Piktogramme zur Richtungsanzeige in der Ampel klar die Überlegenheit gegenüber dem nicht hierarchischen Ansatz zeigen. Auf dem [Bosch Small Traffic Lights Datensatz](#) konnte zudem gezeigt werden, dass der HDTLR-Ansatz dem State of the Art auf diesem Datensatz überlegen ist. Auch ist bisher kein anderes System neben der hier vorgestellten hierarchischen Detektion bekannt, das systematisch verschiedene Piktogramme in Ampeln erkennen kann.

Es konnte auch gezeigt werden, dass der vorgestellte Ansatz mit verschiedenen Netzen zur Extraktion von Features verwendet werden kann, wobei im direkten

3 Statische Verkehrsobjekte: 2D Objektdetektion

Vergleich durchaus qualitative Unterschiede erkennbar sind. Das Teilnetz zur Erfüllung des eigentlichen Tasks der Detektion statischer Verkehrsobjekte kann in dieser Form offenbar als Teil einer Architektur gemäß des in Kapitel 5 vorgestellten MultiNet-Konzeptes eingesetzt werden.

Im Hinblick auf den realen Einsatz in einem Autonomen Fahrzeug liefert der vorgestellte Ansatz für eine Einzelbildklassifikation bereits sehr gute Ergebnisse, für welche allerdings erst im Rahmen einer kompletten Verarbeitungskette eine solide Einschätzung getroffen werden kann. In Bezug auf die geforderte Laufzeit des Verfahrens erfüllen vor allem die qualitativ besten Architekturen die Anforderungen bislang nicht. Allerdings ist an dieser Stelle anzumerken, dass die vorgestellten Architekturen und Modelle bislang nicht mit Optimierungsmethoden für CNNs optimiert wurden. Eine derartige Untersuchung auf Basis des in diesem Kapitel vorgestellten HDTLR ist in Kapitel 7 zu finden. Im Rahmen dieser Untersuchung wird eine Strategie vorgestellt, durch welche die Laufzeit der trainierten HDTLR-Modelle deutlich reduziert werden kann.

4 Dynamische Verkehrsobjekte: 3D Objektdetektion

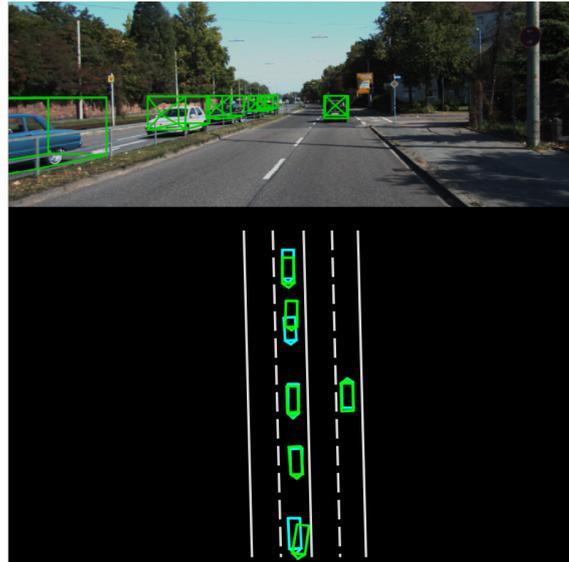


Abbildung 4.1: 3D Detektion dynamischer Verkehrsobjekte in Vogelperspektive. Erkannte Objekte werden in Grün dargestellt, die wahren Objekte in Cyan. Erstveröffentlichung in: [6]

Bisher wurde gezeigt, wie durch einen auf einer CNN-Architektur basierenden Objektdetektor statische Verkehrsobjekte im Bildraum detektiert werden können. Neben diesen Objekten, die im Wesentlichen durch Verkehrszeichen und Ampeln repräsentiert sind, gibt es allerdings eine Vielzahl weiterer Objekte in der Umgebung eines Autonomen Fahrzeugs. Hierunter fallen vor allem andere Verkehrsteilnehmer, welche potentiell relevant für aktuelle und zukünftige Verkehrsentscheidungen sind. Diese lassen sich charakterisieren als nicht ortsfeste, also bezüglich ihrer Position dynamische Objekte. Der Begriff dynamische Verkehrsobjekte wird an dieser Stelle wie folgt definiert:



Definition 3. Ein *dynamisches Verkehrsobjekt* bezeichnet ein Objekt, welches die eigene Position im Verkehrsraum eigenständig ändern kann, also nicht ortsfest ist.

Dies beinhaltet vor allem andere Verkehrsteilnehmer wie Fahrzeuge, Fahrradfahrer oder Fußgänger. An einen Objektdetektor zur Detektion dieser dynamischen Verkehrsobjekte müssen andere Anforderungen gestellt werden, als dies

noch für die Detektion statischer Verkehrsobjekte in Kapitel 3 der Fall war. Wie in Abbildung 4.1 dargestellt, können dynamische Verkehrsobjekte erst in eine Umgebungsrepräsentation integriert werden, wenn diese in 3D detektiert wurden. Detailliert wird die zugrundeliegende Problemstellung in Abschnitt 4.1 erläutert. Nachfolgend wird in Abschnitt 4.2 ein Überblick über verwandte Arbeiten aus der Domäne der Detektion dynamischer Verkehrsobjekte gegeben. In Abschnitt 4.3 wird ein eigener Ansatz zur Detektion dynamischer Objekte vorgestellt. Dieser wird auf verschiedene Architekturen angewandt, welche in Abschnitt 4.4 in einer Evaluation untersucht werden. Eine abschließende Diskussion des vorgestellten Ansatzes wird in Abschnitt 4.5 durchgeführt. Teile dieses Kapitels wurden bereits in der Publikation [6] vorgestellt. Ebenfalls flossen Ergebnisse der Abschlussarbeit [19] in dieses Kapitel ein.

4.1 Problembeschreibung

Im Vergleich zu den in Kapitel 3 vorgestellten statischen Verkehrsobjekten unterscheiden sich die Charakteristika dynamischer Verkehrsobjekte in mehreren Punkten deutlich. So interagieren diese Verkehrsobjekte häufig im selben Straßenbereich mit dem Eigenfahrzeug. Gerade im urbanen Verkehr sind hierbei Abstände von deutlich unter einem Meter nicht selten. Auch muss bei dynamischen Verkehrsobjekten grundsätzlich davon ausgegangen werden, dass sich ihre Position zu jedem Zeitpunkt ändern kann. Dynamische Verkehrsobjekte im Fahrzeugumfeld sind meist in Bewegung, was eine hohe Aktualität der Positionsbestimmung notwendig macht. In einem Idealfall verläuft diese Bewegung parallel zum beobachtenden Eigenfahrzeug, also kreuzungsfrei. Gerade durch Fußgänger im generellen Straßenverkehr oder beliebige andere Verkehrsteilnehmer an Ausfahrten, Einmündungen oder Kreuzungen kann die Bewegungsrichtung auch den Fahrweg des Eigenfahrzeugs kreuzen.

Zusätzlich existiert für die meisten Objektklassen eine hohe Varianz innerhalb der Klasse. So können allein zwischen einzelnen PKW mehr als 100% Größenunterschied liegen, vergleicht man beispielsweise einen Smart mit einem Audi Q7 oder einem Mercedes Vito. Ebenfalls bei Fußgängern oder Radfahrern können gerade zwischen Kindern und erwachsenen Personen erhebliche Größenunterschiede vorhanden sein. So müssen, neben der Position eines dynamischen Verkehrsobjektes, auch dessen individuelle Abmessungen im Rahmen der Detektion ermittelt werden.

Sowohl für die Positionierung der Objekte, als auch die Schätzung der individuellen Abmessungen erscheint daher eine Detektion in Form einer 2D Bounding Box in Bildkoordinaten ungeeignet. Die Position im Kamerabild lässt an dieser Stelle nur wenige Rückschlüsse auf die genaue Position in der dreidimensionalen Umgebung zu. Im Rahmen der Entwicklung von Fahrerassistenzsystemen erfolgte die Detektion bisher oft in Bildkoordinaten, eine Detektion in 3D Kamerakordinaten kann hier allerdings eine deutlich genauere, räumliche Positionierung

liefern. In Form einer detektierten 3D Bounding Box können auch die Objektmessungen direkt und eindeutig durch den Detektor bestimmt werden.

Zu beachten ist hierbei, dass in diesem Fall aus einem zweidimensionalen Sensor anhand von Tiefeninformation eine dreidimensionale Information gewonnen werden soll. Bisher wurde hierfür meist Tiefensensorik wie die Stereokamera [169] verwendet. In den vergangenen Jahren konnte allerdings gezeigt werden, dass eine Schätzung der Tiefe auch aus monokularen Bildern mittels CNN möglich ist [76]. Allerdings ist die Präzision einer Detektion allein basierend auf monokularen Kameras vermutlich nicht ausreichend zur autonomen Erfüllung der Fahraufgabe.

Aus diesem Grund mag eine weitere Verarbeitung der detektierten Objekte im Rahmen einer Fusion mit den Ergebnissen weiterer Sensoren wie LiDAR notwendig sein. Auch hierfür und für die Verwendung in Umgebungskarten scheint eine Detektion in Form einer 3D Bounding Box in Kamera- oder Fahrzeugkoordinaten deutlich besser geeignet. So werden bei der Detektion mittels LiDAR-Sensorik meist ebenfalls 3D Bounding Boxen ermittelt. Auf dieser Ebene wäre eine Fusion der Ergebnisse beider Sensoren einfach realisierbar, da diese in beiden Fällen nicht in einem sensorspezifischen Koordinatensystem wie dem Bildraum der Kamera vorliegen.

Sollen die Ergebnisse eines Detektionsverfahrens in einer gemeinsamen Umgebungsrepräsentation dargestellt werden, kommt häufig ein sogenanntes Umgebungsmodell zum Einsatz. Dies dient einem nachfolgend ausgeführten Algorithmus zur Trajektorienplanung als Planungsgrundlage. Für dieses Umgebungsmodell werden die Informationen aus verschiedenen Quellen meist in eine 2D Karte in Vogelperspektive übertragen. Für den Fall, dass die detektierten Objekte bereits in 3D Koordinaten vorliegen, ist eine Übertragung in ein Umfeldmodell in Vogelperspektive leicht möglich. Hierbei wird auch deutlich, welche der Informationen der 3D Bounding Box eines Objektes im Straßenverkehr wirklich relevant sind.

Die Ausrichtung eines Objektes wird im Straßenverkehr im Wesentlichen durch den Gierwinkel bestimmt. Roll- und Nickwinkel sind im Normalfall zu vernachlässigen. Bei dynamischen Objekten im Straßenverkehr ist davon auszugehen, dass diese bei der Positionsschätzung eine untergeordnete Rolle spielen und sich im zeitlichen Verlauf nur unwesentlich verändern. Ebenfalls sind bei den Abmessungen die Breite und Tiefe eines Objektes besonders relevant. Die Höhe eines Objektes spielt wiederum eine untergeordnete Rolle, da davon auszugehen ist, dass Objekte sich nicht übereinander hinweg bewegen. Es ist also ausreichend, die maximalen Ausdehnungen eines Objektes als Grundfläche mit 2D Bounding Box sowie die Ausrichtung dieser Box in der Straßenebene zu kennen. Dies entspricht dem Gierwinkel des Objektes.

Zusätzlich spielt die verfügbare Latenz zur Detektion auch für dynamische Objekte eine wichtige Rolle. Im Gegensatz zu der Abschätzung für statische Objekte in Abschnitt 3.1 ist hier eine genaue Berechnung allerdings kaum möglich. Sofern

alle Verkehrsteilnehmer die ihnen zugeordneten Straßenbereiche nutzen und die Verkehrsregeln beachten, ist kein plötzlicher Reaktionsbedarf zu erwarten. Situationen, in denen potentiell einem anderen Verkehrsteilnehmer Vorrang einzuräumen ist, müssen frühzeitig erkannt werden.

Anders verhält sich dies, wenn andere Verkehrsteilnehmer von vorgegebenen Wegen abweichen, einzelne Verkehrsregeln missachtet werden oder sich Hindernisse plötzlich auf die Fahrbahn bewegen. Hierbei kann grundsätzlich eine rechtzeitige Detektion und damit eine Verhinderung einer Kollision nicht ausgeschlossen werden. Ebenso wie bei menschlichen Fahrern ist dies allein durch die verbleibende Zeit für eine Reaktion bedingt. Neben der bereits in Abschnitt 3.1 erwähnten, gerichtlich zugestandenen Reaktionszeit von 0,7 bis 0,8 Sekunden [176] ist in derartigen Situationen lediglich der Grundsatz „so schnell wie möglich“ anwendbar. Je schneller eine derartige Situation erkannt wird, desto mehr potentieller Schaden kann durch eine Erkennung und einen Eingriff vermieden werden. Ein autonomes System sollte hier allerdings mindestens den rechtlichen Anforderungen an eine menschliche Reaktion genügen.

Eine weitere bisher nicht betrachtete Grenzsituation ist eine maximale Bremsung eines im Fahrweg vorausfahrenden Fahrzeuges. Diese Situation ist eigentlich bereits durch den Fall der sich korrekt verhaltenden Verkehrsteilnehmer abgedeckt, soll aufgrund ihres speziellen Charakters hier allerdings nochmals kurz analysiert werden. Bei einer auch als Vollbremsung bezeichneten Maximalbremsung muss ein entsprechender Sicherheitsabstand eingehalten werden, um den gesamten Anhalteweg abdecken zu können. Hierin enthalten ist erneut die Reaktionszeit und somit der Reaktionsweg eines Menschen. Hieraus ergeben sich die Möglichkeiten, als maximale Latenz erneut die Reaktionszeit eines menschlichen Fahrers anzunehmen oder aber den Abstand zum vorausfahrenden Fahrzeug an den bekannten maximalen Anhalteweg anzupassen. Da in der vorangegangenen Situation des unvorhergesehenen Eingriffs ebenfalls die menschliche Reaktionszeit als maximaler Wert festgelegt wurde, erscheint dies in der vorliegenden Situation ebenfalls als bessere Alternative.

4.2 Verwandte Arbeiten

Die Detektion dynamischer Verkehrsobjekte veränderte sich durch steigende Anforderungen in den letzten Jahrzehnten deutlich. Bis vor einigen Jahren wurden die Anforderungen durch verschiedene, im Wesentlichen nicht eingreifende Fahrerassistenzsysteme bestimmt. Diese Fahrerassistenzsysteme haben den Zweck, den Fahrer eines Fahrzeuges zu informieren und im Gefahrenfall zu warnen. Hierbei erfolgt kein Eingriff in die Steuerung des Fahrzeuges. In den vergangenen Jahren verschob sich der Fokus allerdings hin zu in die Steuerung des Fahrzeuges eingreifenden Assistenzsystemen und letztlich in Richtung des Autonomen Fahrens. Die hierfür geltenden Anforderungen wurden bereits in Abschnitt 4.1 näher beschrieben.

Zur Erfüllung der Anforderungen von nicht eingreifende Fahrerassistenzsystemen waren meist kamerabasierte Systeme mit klassischen Bildverarbeitungsalgorithmen ausreichend. Hierfür wurden häufig die verbreiteten Verarbeitungsketten zur 2D Detektion von Objekten in Bildkoordinaten verwendet. Für eine Übersicht über einzelne Verfahren sei auf entsprechende Veröffentlichungen mit Survey-Charakter zur Fahrzeugdetektion [214][218] als auch für die Detektion von Fußgängern [97] verwiesen. Im Allgemeinen folgten die hierbei verwendeten Verarbeitungsketten meist einem sehr einheitlichen Muster. So wurden zunächst einzelne Regionen in den Eingabebildern anhand verschiedener Verfahren wie der Vordergrundsegmentierung oder der Erkennung von interessanten Punkten oder Bereichen ausgewählt. In diesen Regionen wurden dann vorhandene Muster mit den gesuchten Objekten abgeglichen (sog. Template Matching) oder Merkmale extrahiert, welche mittels Maschinellem Lernverfahren (z. B. SVM, AdaBoost oder Neuronale Netze) klassifiziert werden konnten.

Diese Methoden ermöglichten lediglich eine rudimentäre Positionsbestimmung anhand der Objektposition im Kamerabild. Mit der steigenden Qualität von Assistenzsystemen, dem wachsenden Funktionsumfang sowie dem Weg hin zu eingreifenden Fahrerassistenzsystemen wuchsen allerdings auch die Anforderungen an die Genauigkeit der Positionsbestimmung. Bezüglich der verwendeten Sensorik verschob sich der Fokus weg von monokularen Kamerasensoren hin zu 3D Sensoren. Dies führte zur Verwendung von Stereokameras [57], Time-of-Flight (ToF)-Kameras mit den sogenannten Photonic Mixing Device (PMD)-Sensoren [203][202] oder LiDAR-Sensoren [150] zur Detektion von dynamischen Verkehrsobjekten. Mit diesen Sensoren wurde eine deutlich genauere Positionsbestimmung der Verkehrsobjekte in der näheren Umgebung ermöglicht. Von diesen 3D Sensoren konnte sich im Wesentlichen der LiDAR-Sensor durchsetzen. Im Vergleich zu Kamerasensorik haben diese allerdings einen deutlich höheren Stückpreis und müssen vorwiegend an exponierten Stellen in einem Fahrzeug verbaut werden.

Seit dem Durchbruch tiefer Neuronaler Netze in der kamerabasierten Bildverarbeitung in Form von CNNs haben monokulare Kamerasensoren für die Detektion dynamischer Verkehrsobjekte als alleinige oder zu LiDAR komplementäre Sensoren [58][183] deutlich an Relevanz gewonnen. Erste CNN-basierte Ansätze hatten hier ebenfalls zum Ziel, eine 2D Detektion in Bildkoordinaten direkt [122] oder unter Verwendung klassischer Konzepte der Bildverarbeitung wie z. B. Klassifikationskaskaden [240] durchzuführen. Zwischenzeitlich werden durch aktuellere Ansätze allerdings auch räumliche Informationen zu den Verkehrsobjekten gewonnen.

Als Grundlage dient hier meist eine durch 2D Detektion selektierte Region in Form einer Bounding Box in Bildkoordinaten. Auf der Grundlage dieser 2D Bounding Box werden dann durch Template Matching [165][54], geometrische Vorannahmen [166] oder das Schätzen zusätzlicher relevanter Punkte [91] 3D Detektionen erzeugt. Der Ansatz von Mottaghi et al. [165] hat das Ziel, die 3D Pose von

erkannten Objekten zu schätzen. Die zu detektierenden Objekte sind dem Ansatz hierbei in Form eines hierarchischen Objektmodells bekannt, wodurch allgemeine Objektklassen wie *Fahrzeug* bei der Detektion genauer spezifiziert werden können, z. B. als *Limousine*. Für die im ersten Detektionsschritt erkannten 2D Bounding Boxen werden in einem Probabilistischen Graphischen Modell (PGM) anhand von Features wie HOG-Templates, CNN-Backbone-Ausgaben und einem Template Matching die Objektklassen in den Hierarchiestufen, sowie die 3D Pose geschätzt. Als Templates hierfür dienen CAD-Modelle der verschiedenen Objektklassen.

In dem von Chen et al. [56] vorgeschlagenen Ansatz werden die 2D Bounding Boxen nicht durch klassische Objektdetektion erzeugt. Für die Eingabebilder wird zunächst eine Bodenebene angenommen, auf welcher 3D Bounding Boxen verschiedener Größe in einer Art Sliding-Window-Ansatz Objektkandidaten erzeugen. Die Parameter der 3D Boxen werden hierbei anhand der Objekte im Trainingsdatensatz erzeugt. Der Ansatz hat somit eine große Ähnlichkeit zu den in der 2D Objektdetektion verwendeten Anchor Boxes [190]. Für jeden der 3D Kandidaten wird im Anschluss eine 2D Bounding Box im Bildraum erzeugt. Anhand der 2D Box wird für jeden Kandidaten eine Bewertung auf Basis von Form, semantischer Segmentierung von Instanz und Klasse, Kontext und der Position in der Szene durchgeführt. Nach Anwendung einer NMS bilden die bestbewerteten Kandidaten das Detektionsergebnis.

Bei dem Deep-Manta-Ansatz [54] werden die 2D Bounding Boxen in einem mehrstufigen Prozess erzeugt. So wird zunächst ein Region Proposal Network [190] zum ungefähren Finden von ersten Objektkandidaten verwendet. Diese werden in zwei Stufen jeweils durch einige CNN-Schichten nochmals verfeinert. In der zweiten Stufe wird neben der Verfeinerung auch eine Klassifikation der Objekte durchgeführt. Zusätzlich werden in der finalen 2D Bounding Box jedes Objektes die Punkte für weitere markante Stellen an den Objekten detektiert. Diese werden mit verschiedenen Templates aus einem vorhandenen Datensatz verglichen, um so die 3D Position der Objekte ermitteln zu können. Hierzu kann zusätzlich die umschließende 3D Bounding Box der Objekte aus den Templates ermittelt werden.

Die Ansätze von Mousavian et al. [166] und Oeljeklaus et al. [173] basieren auf der Annahme, dass die 2D Bounding Box das detektierte Objekt perfekt minimal umschließt. Somit können die Seiten der 2D Box direkt zur Ermittlung einzelner Parameter der 3D Bounding Box verwendet werden. Die genaue Dimension und Ausrichtung der 3D Box sowie die einzelnen Klassenwahrscheinlichkeiten werden bei [166] basierend auf dem Bildausschnitt der 2D Box von einem CNN geschätzt. Bei [173] wird die Klassenwahrscheinlichkeit und der Beobachtungswinkel des Objektes direkt mit der 2D Bounding Box in einem gemeinsamen CNN geschätzt. Zur Bestimmung der weiteren Parameter werden zusätzliche geometrische Annahmen getroffen.

Einen ähnlichen Ansatz verwendet auch das MB-Net [91]. Hierbei wird für jede 2D Bounding Box durch das CNN zur Detektion ein zusätzlicher Parameter ge-

schätzt, der den Übergang zwischen der im Bild sichtbaren Objektseite und der ebenfalls sichtbaren Vorder- oder Rückseite anzeigt. Die 3D Bounding Box wird anschließend unter Verwendung von Templates für verschiedene Objektklassen abgeleitet. Bei den bisher betrachteten CNN-basierten Ansätzen beruht die eigentliche Entscheidung der Detektion grundsätzlich auf der 2D Bounding Box der Objekte. 3D Modelle werden dabei zusätzlich geschätzt oder auf Grundlage der 2D Box ausgewählt.

Andere Wege schlagen die Ansätze BS^{3D} [2] und OFT [194] ein. Bei dem BS^{3D}-Ansatz wird durch ein CNN direkt eine sogenannte *Bounding Shape* für jedes Objekt geschätzt. Im Gegensatz zu einer durch 2 Punkte darstellbaren 2D Bounding Box wird diese Bounding Shape repräsentiert durch 4 Punkte, welche im Bild sichtbar sind. Bei Betrachtung einer 3D Bounding Box um ein Objekt stellen 3 dieser 4 Punkte die sichtbaren Eckpunkte der Bodenfläche dar. Um die eigentliche Bounding Shape zu erhalten, werden diese Punkte entlang der Kanten der 3D Box verbunden. An einem Endpunkt der Verbindungslinie wird diese zusätzlich an der dortigen vertikalen Kante der 3D Bounding Box mit dem entsprechenden nächsten Punkt verbunden. Anhand dieser Bounding Shape und durch weitere Informationen zur Szenengeometrie kann so eine 3D Bounding Box erzeugt werden.

Der *Orthographic Feature Transform* (OFT)-Ansatz [194] verfolgt hingegen die Strategie, eine Zwischenrepräsentation der extrahierten Features im 3D Raum zu erzeugen. Aus dieser Zwischenrepräsentation werden die Features in einen 2D Raum überführt, welcher die Szene in der Vogelperspektive darstellt. Anhand der Features in Vogelperspektive werden durch ein weiteres CNN die Klassenwahrscheinlichkeit sowie Position, Größe und Orientierung für jeden Punkt in der Vogelperspektivenkarte analog zu klassischen 2D Objektdetektoren geschätzt. Die Größe der Objekte wird hierbei anhand der Abweichung von sogenannten Anchor Boxes geschätzt. Eine direkte Detektion der Objekte im 3D Raum wird bislang durch keinen der genannten Ansätze durchgeführt.

4.3 Methodik

Für die Detektion dynamischer Verkehrsobjekte in monokularen Eingabebildern wird nun ein Verfahren eingeführt, welches eine Schätzung der 3D Koordinaten direkt aus einem CNN ermöglicht. Dieses beruht auf den Erfolgsfaktoren von leistungsstarken 2D Objektdetektoren mit kurzen Inferenzzeiten wie YOLO [186], SSD [157] oder OverFeat [205]. So werden insbesondere die Ende-zu-Ende-Verarbeitung in einem *Fully Convolutional Network* (FCN), die regionsbasierte Detektion von Objekten sowie die relative Positionsschätzung als Konzepte übernommen.

Das eingeführte Verfahren kann Encoder-agnostisch verwendet werden – es ist also unabhängig von der Wahl eines konkreten CNNs zur Extraktion von Featu-

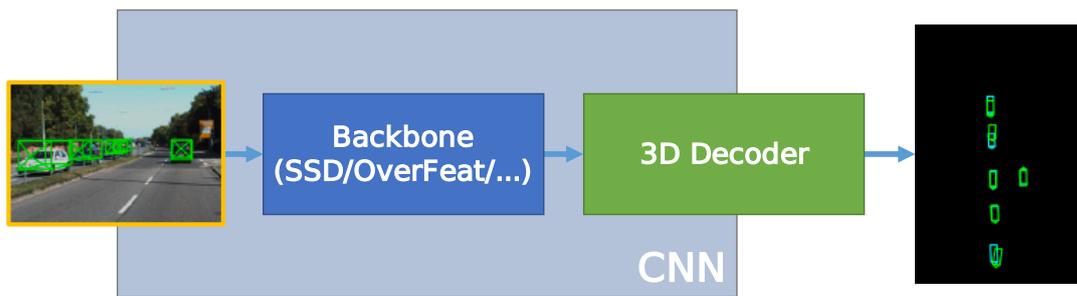


Abbildung 4.2: Aufbau des Netzes zur 3D Objektdetektion, bestehend aus einem Backbone und einem 3D Decoder. Im Eingabebild sind die zum Training verwendeten 3D Bounding Boxes visualisiert. Die erkannten Objekte sind in dieser Darstellung bereits in Vogelperspektive überführt.

res. So kann das Verfahren auch einfach als Teil der in Kapitel 5 beschriebenen MultiNet-Architektur verwendet werden. Auch können als Grundlage statt einfacher Architekturen zur Extraktion von Features bereits bestehende 2D Objektdetektoren verwendet werden. Das komplette Verfahren zur Detektion dynamischer Verkehrsobjekte ist in Abbildung 4.2 dargestellt, wobei hier die Ausgabe des Netzes in Vogelperspektive gezeigt wird.

Im Vergleich zu einem Verfahren zur Detektion von 2D Objekten im Bildraum müssen für ein Verfahren zur 3D Detektion von dynamischen Verkehrsobjekten an einigen Stellen Veränderungen vorgenommen oder neue Methoden eingeführt werden. In Abschnitt 4.3.1 wird zunächst die verwendete 3D Objektrepräsentation beschrieben. Die im vorgestellten Verfahren erstellten Netzarchitekturen werden in Abschnitt 4.3.2 ausführlich erläutert. Auf die zum Training der Modelle verwendete Fehlerfunktion wird in Abschnitt 4.3.3 näher eingegangen. Abschließend wird in Abschnitt 4.3.4 die für die 3D Detektion durchgeführte Nachverarbeitung beschrieben.

4.3.1 Objektrepräsentation

Wie bereits bei der Detektion von 2D Objekten im Bildraum, die in Kapitel 3 beschrieben ist, muss auch für diesen Task wieder eine Kodierung der Objekte innerhalb der CNN-Architektur gefunden werden. Da es sich in diesem Fall um 3D Objekte handelt, deren Position nicht direkt mit dem Bildraum in Verbindung steht, ist die bisherige Herangehensweise nicht direkt übertragbar. Für die 3D Objekte müssen durch das CNN auch deutlich mehr Parameter geschätzt werden. Denn im Vergleich zu den 4 Freiheitsgraden einer 2D Bounding Box in der Form $((x_{\min}, y_{\min}), (x_{\max}, y_{\max}))$ besitzt eine 3D Bounding Box je nach Vorannahmen bis zu 9 Freiheitsgrade. Zusätzlich bleibt zu beachten, dass auch bei dieser Art der Objektdetektion natürlich die Anzahl der zu detektierenden Objekte vorab nicht bekannt ist und von Eingabebild zu Eingabebild variieren kann. So muss von der

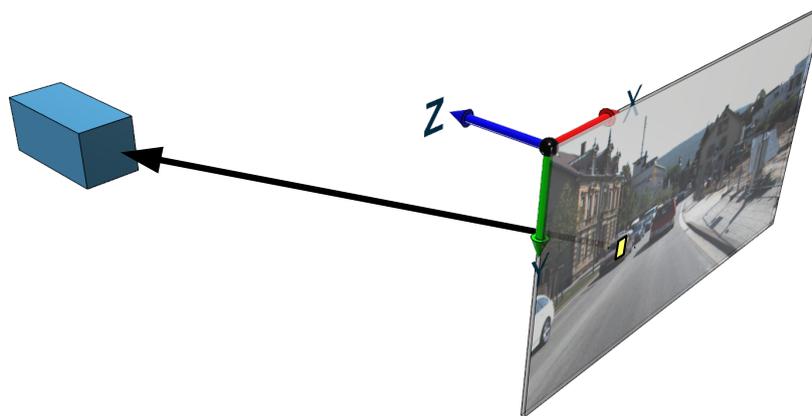


Abbildung 4.3: Schätzung der **3D** Bounding Box anhand der Ursprungsregion des Objektes im Eingabebild (gelb). Zusätzlich wird der Abstand des Objektes zur Kamera geschätzt. Ähnlich veröffentlicht in: [6]

verwendeten Netzarchitektur bereits eine große Menge möglicher Objekte vorgesehen werden.

In Anlehnung an die Detektion von **2D** Objekten wird auch hierfür eine engmaschige Gitterstruktur aus Eingaberegionen verwendet, welche in Abbildung 4.9 dargestellt ist. Hier ergibt sich nun allerdings ein bedeutender Unterschied zur **2D** Detektion. Die Ausgabe des Detektors entspricht nun nicht mehr einer **2D** Bounding Box in Bildkoordinaten. Im Falle der **3D** Detektion werden die Objektdetektionen in einer dreidimensionalen Form in Kamerakoordinaten ausgegeben. Um eine größere aber unbekannte Anzahl derartiger Objektdetektionen ausgeben zu können, wird ein Ankerpunkt für jedes Objekt im Ursprungsbild benötigt. Andernfalls ist es nur schwer möglich, einzelne Netzausgaben darauf zu trainieren, ein bestimmtes Objekt zu detektieren.

Um einen derartigen Anker eindeutig zu definieren, wird eine Projektion von Kamerakoordinaten aus dem Bildraum verwendet, wie in Abbildung 4.3 dargestellt. So ist vergleichbar mit dem Fall des **2D** Detektors eine Bildregion, in der sich ein Objekt befindet, für die Detektion dieses Objektes verantwortlich. Da sich ein Objekt in mehreren Bildregionen befinden kann, ist an dieser Stelle eine Nachverarbeitung der Objektkandidaten notwendig. Die Position eines **3D** Objektes wird nun innerhalb einer Region anhand seines Mittelpunktes in Bildkoordinaten (c_x, c_y) und seiner Distanz d zur Kamera definiert. Um den Trainingsprozess zu vereinfachen, wird der Wertebereich von d normalisiert auf das Intervall $[0, 1]$. Die Normalisierung wird hierbei wie folgt durchgeführt:

$$d = \frac{1}{d_{max}} \min(d, d_{max}) \quad (4.1)$$

4 Dynamische Verkehrsobjekte: 3D Objektdetektion

wobei d_{max} die maximale Distanz eines Objektes ist, die im Trainingsdatensatz beobachtet wurde. Die Bildkoordinaten (c_x, c_y) werden relativ zur Mitte der aktuellen Region angegeben, da nur auf diese Weise die Werte von regionsunabhängigen Filtern gelernt werden können, welche in CNNs und insbesondere FCNs Verwendung finden.

Die Darstellung der Objekte erfolgt durch eine minimal umschließende 3D Bounding Box in der Form eines Quaders mit Breite b , Länge l und Höhe h sowie der Ausrichtung θ . Diese repräsentiert hierbei den Gierwinkel des Objektes. Bezüglich der anderen Rotationsachsen wird eine flache Welt angenommen (sog. Flat World Assumption), daher können Roll- und Nickwinkel vernachlässigt werden. Da eine Schätzung von θ im Wertebereich $[0, 360]$ für den Trainingsprozess ebenfalls suboptimal ist, wird der Winkel kodiert in $(\sin(\theta), \cos(\theta))$ angegeben. Die Maße für Breite, Länge und Höhe werden relativ zu einer Anchor Box für ein Durchschnittsfahrzeug angegeben:

$$(b, h, l) = (b - b_{anchor}, h - h_{anchor}, l - l_{anchor}) \quad (4.2)$$

Die Werte eines Durchschnittsfahrzeuges für die Anchor Box werden anhand der Durchschnittswerte über alle Fahrzeuge des Trainingsdatensatzes berechnet. Zusätzlich zu der Ausdehnung besitzt jedes Objekt eine Klasse c .

4.3.2 Netzarchitektur

Wie bereits im vorigen Abschnitt erwähnt, basieren die Ankerpunkte der potentiellen Objekte analog zur 2D Objektdetektion auf einzelnen Zellen einer Gitterstruktur, in die das Eingabebild unterteilt ist. In der von den 2D Objektdetektoren bekannten Unterscheidung von One-Stage- und Two-Stage-Detektoren (vgl. Abschnitt 3.2.1) entspricht dies der Klasse der One-Stage-Detektoren. Der grundsätzliche Aufbau der verwendeten Architektur basiert dabei analog zur Detektion statischer Verkehrsobjekte (vgl. Abschnitt 3.3) auf dem Designprinzip der FCNs. Hieraus ergibt sich der Vorteil, dass insbesondere die Komponenten zur 3D Detektion flexibel mit bereits vorhandenen Architekturen zur Extraktion von Features oder anderen Detektionsaufgaben kombiniert werden können. Insbesondere im Hinblick auf die Ein- und Ausgabegröße einzelner Teilnetze ermöglicht das FCN-Prinzip einen deutlichen Zugewinn an Flexibilität.

Die generelle Architektur des hier vorgestellten 3D Objektdetektors teilt sich auf in einen Backbone als ersten Netzteil und einen 3D Decoder zur Erzeugung der Detektionsergebnisse, wie in Abbildung 4.2 dargestellt. Der Backbone setzt sich hierbei zusammen aus einem Encoder zur Extraktion von Features und einigen Schichten zur Zwischenverarbeitung dieser Features. Als Encoder können generell beliebige vorhandene oder neue Architekturen (vgl. Abschnitt 2.1) genutzt werden. Ein Überblick über existierende Architekturen zur Extraktion von Features sowie eine kurze Beschreibung der einzelnen Ansätze ist in Abschnitt 2.1 zu

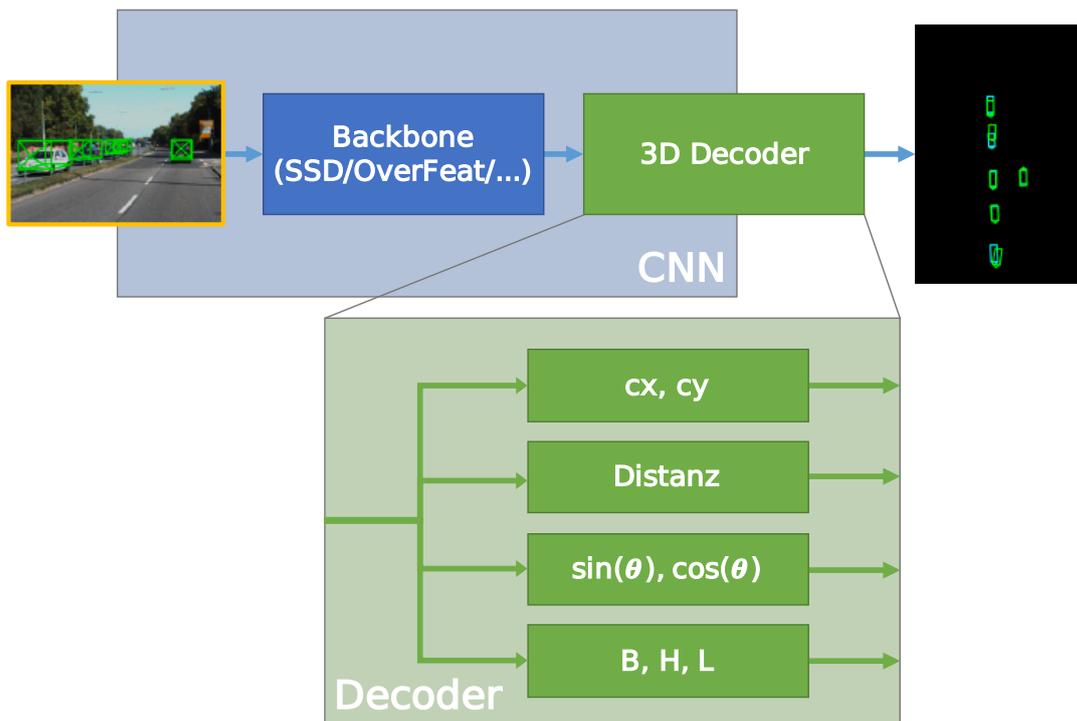


Abbildung 4.4: Aufbau des Decoders zur 3D Objektdetektion. Der 3D Decoder besteht aus vier parallelen Strängen, welche für die einzelnen Parameter der Bounding Box-Schätzung verantwortlich sind. Erstveröffentlichung in: [6]

finden. Alternativ kann für die 3D Objektdetektion als Backbone auch ein Ansatz zur 2D Objektdetektion wie SSD [157] oder RetinaNet [153] verwendet werden.

Die Granularität der Ausgabe ist hierbei von der Wahl des Encoders bzw. des Backbones abhängig. So wird diese im Wesentlichen bestimmt durch verschiedene Designentscheidungen der verwendeten Architekturen. So ist hauptsächlich der globale Stride der Architektur verantwortlich für die Reduktion der Granularität. Dies kann durch verschiedene zusätzliche Schichten wie Tiling [205] oder Deconvolution [244] kompensiert werden. Die Granularität wird meist anhand der Reduktion der Auflösung zwischen Ein- und Ausgabe beschrieben. Hierbei wird angegeben, wie viele Pixel in der Form eines Rechtecks aus dem Eingabebild einer einzigen Ausgabe entsprechen. Die Reduktion der Auflösung reicht meist von 4×4 Pixeln bei z. B. OverFeat [205] bis zu 32×32 Pixeln bei Encodern wie VGG [212].

Das zentrale Element des hier vorgestellten Verfahrens zur 3D Objektdetektion ist der 3D Decoder. Dieser liefert auf Basis der Ausgaben des Backbones die detektierten 3D Objektkandidaten, welche nachfolgend in einem Nachverarbeitungsschritt (siehe Abschnitt 4.3.4) zu finalen 3D Objekten verarbeitet werden. Die 3D Objektkandidaten bestehen dabei entsprechend der in Abschnitt 4.3.1 beschriebenen Objektrepräsentation aus einer Klassenwahrscheinlichkeit für jede trainier-

te Klasse, Bildkoordinaten für den Ankerpunkt, einer Entfernung zur Kamera, der Ausrichtung des Objektes sowie dessen Länge, Breite und Höhe. Diese werden innerhalb des 3D Decoders gemäß des in Kapitel 5 eingeführten MultiNet-Konzeptes in 5 separaten Heads bestimmt. Die Bestimmung der 4 Komponenten für die Objektrepräsentation ist in Abbildung 4.3.2 dargestellt. Hinzu kommt die Schätzung der Klassenwahrscheinlichkeiten für die Objekte in einem separaten Head.

Jeder dieser Heads ist durch eine Convolution-Schicht realisiert, wobei für die Objekteigenschaften mit begrenztem Wertebereich zusätzlich eine Aktivierungsfunktion benötigt wird. Die Filtergröße für alle Convolution-Schichten beträgt hierbei entsprechend dem FCN-Prinzip 1×1 bei einem Stride von 1. Die Anzahl der Filter unterscheidet sich allerdings für die einzelnen Heads. Für die Schätzung der Klassenwahrscheinlichkeit ist die Anzahl der Filter identisch zur Klassenanzahl, welche das resultierende Modell im Trainingsprozess in den Eingaben bekommt. Da die Ausgabe für jeden Objektkandidaten eine Wahrscheinlichkeitsverteilung über alle bekannten Klassen darstellen soll, wird zusätzlich als Aktivierungsfunktion die normalisierte Exponentialfunktion verwendet, welche unter dem Namen Softmax bekannt ist.

Zur Schätzung der Bildkoordinaten werden ein Filter pro Koordinate, in Summe also zwei Filter verwendet. Die Schätzung der Entfernung enthält lediglich einen Filter und nutzt folgende abgewandelte Sigmoidfunktion als Aktivierungsfunktion:

$$A(x) = \frac{d_{max}}{1 + e^{-x}} \quad (4.3)$$

Hierbei wird jede Ausgabe der Schicht auf den Wertebereich $[0, d_{max}]$ mit der maximal im Training beobachteten Distanz d_{max} abgebildet. Für die Schätzung der Orientierung θ werden zwei Filter benötigt, da sowohl $\sin(\theta)$ als auch $\cos(\theta)$ geschätzt werden. Aufgrund des Zielwertebereiches beider trigonometrischer Funktionen wird für die Aktivierungsfunktion in diesem Head der Tangens hyperbolicus genutzt. Der letzte Head verwendet in der Convolution-Schicht 3 Filter, je einen zur Schätzung von Breite, Höhe und Länge des detektierten Objektkandidaten. Da der Wertebereich für diese Objekteigenschaften prinzipiell unbegrenzt ist, wird hierfür keine zusätzliche Aktivierungsfunktion benötigt. Eine Übersicht über die Details der einzelnen Heads ist in Tabelle 4.1 zu finden.

Basierend auf diesem 3D Decoder können nun mit verschiedenen Backbones unterschiedliche Architekturen zur Detektion dynamischer Verkehrsobjekte realisiert werden. Im Rahmen dieser Arbeit werden vier konkrete Netzarchitekturen vorgestellt. Drei Architekturen basieren hierbei direkt auf bekannten 2D Detektoren als Backbone. Zusätzlich wird die Direct3D-Architektur definiert, welche auf klassischen Netzen zur Extraktion von Features als Encoder beruht und diese durch zusätzliche Schichten zu einem eigenen Backbone erweitert. Bei den zusätzlich angefügten Schichten handelt es sich um zwei Convolution-Schichten mit je 4.096 Filterkernen. Die Größe der Filter in der ersten Schicht ist dabei ab-

Ausgabe	Wertebereich	Schicht	zahl	Filtergröße	stride	Aktivierungsfunktion
p_c	$[0, 1]$	conv	#Klassen	1x1	1,1	Softmax
cx, cy	$(-\infty, \infty)$	conv	2	1x1	1,1	-
d	$[0, d_{max}]$	conv	1	1x1	1,1	$d_{max} \cdot \sigma(\cdot)$
$\sin(\theta), \cos(\theta)$	$[-1, 1]$	conv	2	1x1	1,1	$\tanh(\cdot)$
B, H, L	$(-\infty, \infty)$	conv	3	1x1	1,1	-

Tabelle 4.1: Die Architektur des 3D Decoders besteht aus 5 parallelen Convolution-Schichten. Die einzelnen Schichten dienen hierbei zur Schätzung von 1) Klassenwahrscheinlichkeit, 2) Objekt in Bildkoordinaten, 3) Objektdistanz zur Kamera, 4) Orientierung des Objektes und 5) Dimension des Objektes.

hängig von der Wahl des konkreten Encoders. Diese Encoder wurden meist ursprünglich als Klassifikationsnetze mit fester Eingabegröße definiert.

Die Größe der letzten Convolution-Schicht in diesen Architekturen entspricht im Normalfall dem Wert für die Größe der Filter der ersten Convolution-Schicht der Direct3D-Architektur. Grundsätzlich ist dieser Wert innerhalb der Grenzen eines validen Gesamtnetzes variabel. Die Filter der zweiten Convolution-Schicht haben die Größe 1×1 . In beiden Schichten wird die [Rectified Linear Unit \(ReLU\)](#) als Aktivierungsfunktion verwendet. Die komplette Architektur von Direct3D ist in [Abbildung 4.5](#) dargestellt. Diese Architektur entspricht im Wesentlichen dem Ergebnis der üblichen Umwandlung eines [CNN-Klassifikators](#), wie sie beispielsweise bei [OverFeat \[205\]](#) oder [FCN \[158\]](#) durchgeführt wird. Die Direct3D-Architektur kann auch direkt in die in [Kapitel 5](#) vorgestellte MultiNet-Architektur integriert werden, indem das MultiNet als Encoder des Netzes betrachtet wird und die restlichen Schichten von Direct3D aus der Perspektive des MultiNet als ein Head wahrgenommen werden.

Anstelle der beschriebenen Direct3D-Architektur mit einem bekannten Encoder kann das vorgestellte Verfahren wie bereits erwähnt auch direkt mit Architekturen zur [2D Detektion](#) von Objekten als Backbone verwendet werden. Im Rahmen dieser Arbeit werden im Folgenden Architekturen mit Anpassungen für die [2D Detektoren](#) [OverFeat \[205\]](#), [Single Shot MultiBox Detector \(SSD\) \[157\]](#) und [RetinaNet \[153\]](#) vorgestellt. Der OverFeat-Detektor entspricht in weiten Teilen bereits der Architektur des Direct3D-Detektors. Eine grafische Gegenüberstellung der beiden Ansätze ist in [Abbildung 4.5](#) dargestellt. Den wesentlichen Unterschied zwischen beiden Architekturen bildet hierbei eine zusätzliche Tiling-Schicht im Anschluss an die finale Convolution-Schicht. Durch diese Schicht kann die Auflösung der Ausgabe, auf welcher der 3D Decoder letztendlich ausgeführt wird, um ein Vielfaches erhöht werden. Somit wird eine präzisere Erkennung auch von kleineren Objekten möglich, was allerdings eine deutlich erhöhte Parameteranzahl in allen auf den Encoder folgenden Schichten und somit einen erhöhten Speicherverbrauch nach sich zieht.

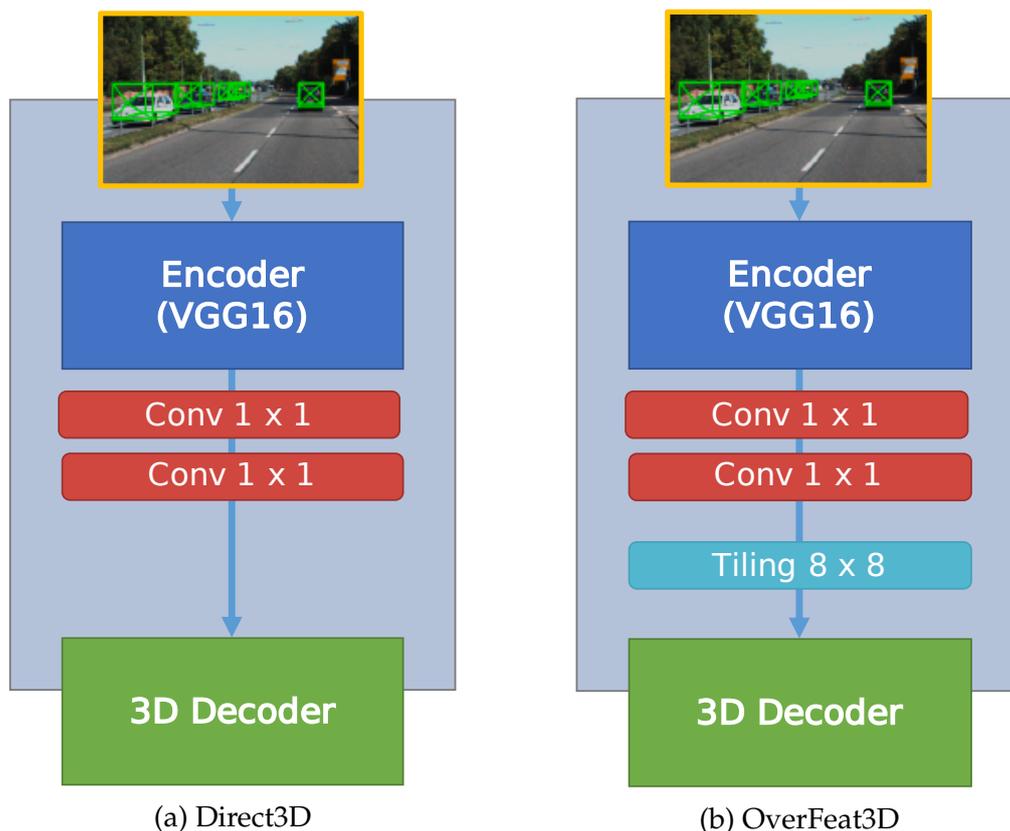


Abbildung 4.5: Vergleich zwischen den Architekturen Direct3D (a) und OverFeat3D (b). Wesentlicher Unterschied zwischen beiden Architekturen ist die zusätzliche Tiling-Schicht bei OverFeat. Hierdurch wird die Ausgabeauflösung um den Tiling-Faktor erhöht, gleichzeitig steigt allerdings auch die Anzahl der Parameter im Netz.

Bei der Verwendung von SSD [157] als Backbone muss beachtet werden, dass die Detektion der Objekte auf verschiedenen Bildskalen erfolgt. Der Backbone besteht in diesem Fall zunächst aus einem VGG16 zur Extraktion der Features, gefolgt von mehreren Blöcken zur Reduktion der Auflösung, wie in Abbildung 4.6 dargestellt. Die einzelnen Blöcke bestehen dabei aus zwei aufeinanderfolgenden Convolution-Schichten, gefolgt von einer Pooling-Schicht. Nach jedem dieser Blöcke, sowie initial nach dem VGG16 Encoder, werden die resultierenden Feature Maps in einen separaten 3D Decoder geleitet. Die gelernten Gewichte der einzelnen 3D Decoder werden hierbei getrennt für jede Stufe trainiert. Durch diese stufenweise Verarbeitung wird die Detektion von zunächst kleinen Objekten mit jedem Block in Richtung größerer Objekte verschoben. Die Anzahl der möglichen Regionen wird mit jeder Stufe um den Faktor 2 in jeder Dimension reduziert.

Wird das RetinaNet [153] als Backbone verwendet, müssen ähnlich zu dem SSD-Backbone die verschiedenen Ausgaben beachtet werden, da dieses Netz in Form eines Feature Pyramid Network (FPN) [152] aufgebaut ist. Wie in Abbildung 4.7

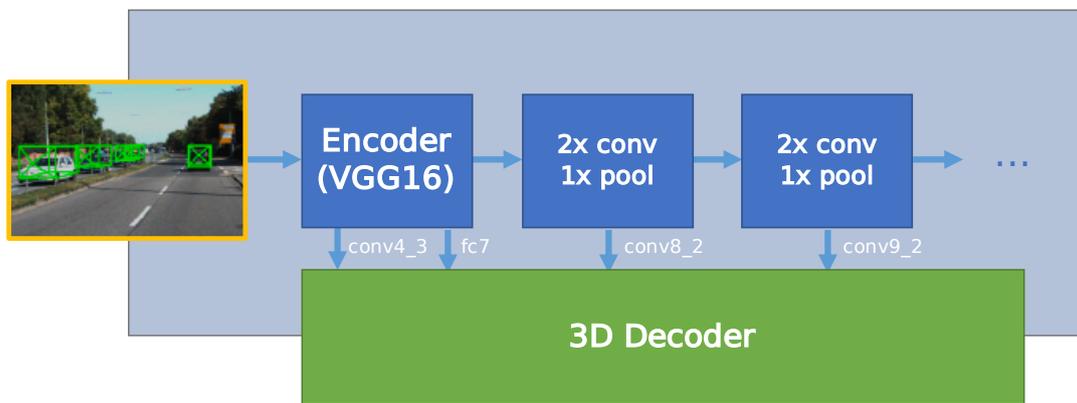


Abbildung 4.6: Aufbau des SSD-Detektors zur 3D Objektdetektion. Wie bei der 2D Objektdetektion mit SSD existieren Ausgänge an verschiedenen Schichten des Netzes (conv4_3, fc7, conv8_2, conv9_2 ...). Diese werden jeweils mit einem 3D Decoder mit separaten Gewichten verbunden.

dargestellt, basieren die zwei Ausgaben P6 und P7 auf den finalen Features des als Encoder genutzten ResNet [186], wobei jeweils eine weitere Convolution-Schicht den Ausgaben vorangeht. Die Ausgaben P3 bis P5 hingegen basieren auf Zwischenausgaben aus dem ResNet, welchen ebenfalls eine Convolution-Schicht folgt. Die Ergebnisse hieraus werden mit dem Ergebnis des jeweils nachfolgenden Ausgangs konkateniert und im Anschluss an eine weitere Convolution-Schicht ausgegeben. Die einzelnen Ausgänge P3–P7 werden jeweils mit einem 3D Decoder verbunden. Im Gegensatz zu der Architektur, welche einen SSD als Backbone verwendet, werden hier für die einzelnen Decoder gemeinsame Gewichte genutzt. Dies erfolgt analog zur Vorgehensweise des originalen 2D Detektors. Da dieser bereits einen verhältnismäßig hohen Speicherbedarf verursacht, kann so die Anzahl der zusätzlich benötigten Parameter etwas reduziert werden.

4.3.3 Fehlerfunktion

Die Netzarchitekturen der verschiedenen 3D Detektoren weisen bereits einige Ähnlichkeiten zu dem in Kapitel 5 vorgestellten MultiNet Architekturprinzip auf. Bei den 3D Detektoren werden allerdings statt unterschiedlicher Tasks die einzelnen Komponenten zur Erfüllung eines Tasks gemeinsam trainiert. Hieraus ergeben sich im Vergleich zu MultiNet-artigen Architekturen einige Vereinfachungen. So sind für alle Ausgänge des Netzes zu jedem Trainingsdatum Annotationen verfügbar, da diese in einem einzigen Annotationsschritt entstehen. Auch sind die Bildbereiche, für die sich eine hohe Aktivierung ergibt, für alle zu schätzenden Werte identisch, da diese auf demselben Grundobjekt basieren. Daher dürften die Anforderungen an einen Encoder zur Extraktion von Features geringer sein, als bei gleichzeitigem Lernen verschiedener Tasks. Im Folgenden wird die

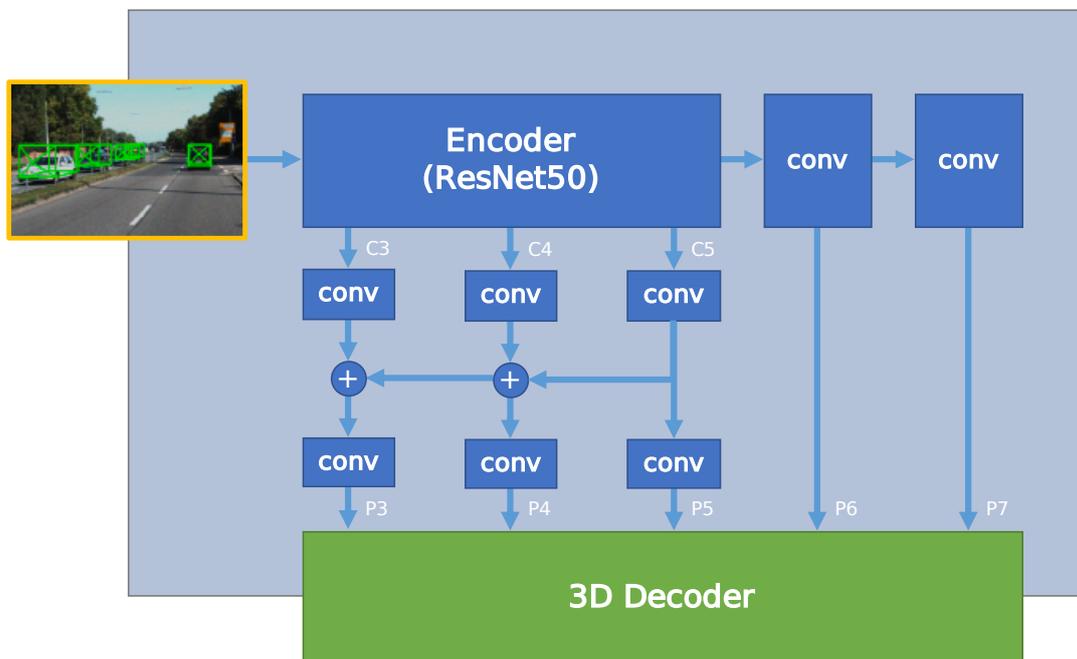


Abbildung 4.7: Aufbau des RetinaNet-Detektors zur 3D Objektdetektion. Der Aufbau entspricht dem ursprünglichen RetinaNet und enthält 5 Ausgänge an verschiedenen Schichten (P3–P7). Diese werden mit je einem 3D Decoder verbunden. Im Trainingsprozess werden die Gewichte der einzelnen Decoder geteilt und somit gemeinsam trainiert.

bereits begonnene sprachliche Trennung fortgeführt und die Komponenten als ebensolche bezeichnet, da es sich streng genommen in diesem Fall nicht um eigenständige Tasks handelt.

Bei der Wahl einer gemeinsamen Fehlerfunktion für die einzelnen Komponenten können allerdings die Erkenntnisse für MultiTask-Netze direkt verwendet werden. So wird für die Netzarchitekturen zur 3D Objektdetektion eine Variante der unsicherheitsbasierten MultiTask-Fehlerfunktion von Kendall et al. [135] verwendet. Diese hat das Ziel, in einem MultiTask-Netz jeden einzelnen Task t_i anhand dessen homoskedastischer Unsicherheit σ_i zu gewichten und ist wie folgt definiert:

$$\mathcal{L}_{MT} = \sum_i \left(\frac{1}{2\sigma_i^2} L_{t_i} + \log(\sigma_i^2) \right) \quad (4.4)$$

Die homoskedastische Unsicherheit stellt hierbei eine Ausprägung der aleatorischen Unsicherheit dar. Diese Klasse der Unsicherheiten bezieht sich auf die inhärente Zufälligkeit der Daten, also deren natürliche Variabilität. Sie kann bzgl. der Eingabedaten in einem Neuronalen Netz einen variablen (heteroskedastische Unsicherheit) sowie einen konstanten (homoskedastische Unsicherheit) Anteil besitzen. Die homoskedastische Unsicherheit ist also über verschiedene Eingabedaten konstant, betrachtet man allerdings ein einzelnes Datum über verschiedene Tasks

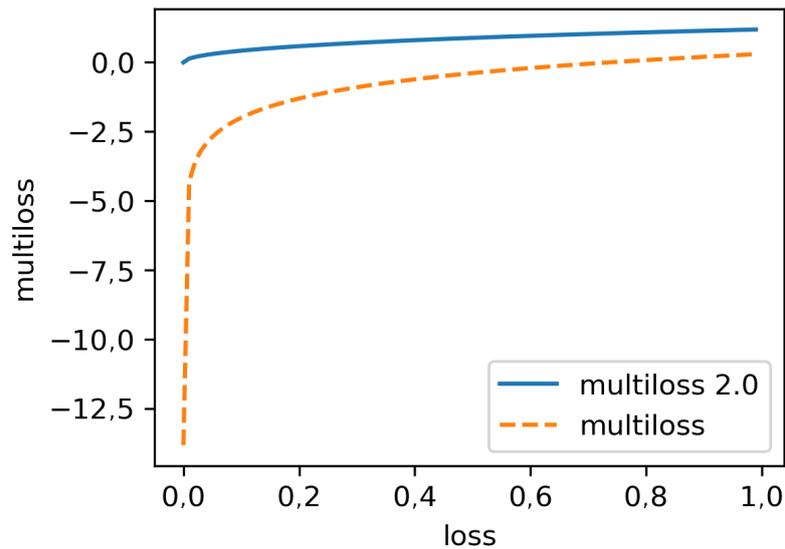


Abbildung 4.8: Unterschiede im Verlauf zwischen der Gewichtung von Kendall et al. [135] (orange) und der modifizierten Gewichtung (blau). Ähnlich veröffentlicht in: [6]

hinweg, muss dies nicht der Fall sein. Ursächlich hierfür kann beispielsweise eine variierende Güte der Annotationen in verschiedenen Datensätzen sein, die für das Training der einzelnen Tasks genutzt werden.

Die homoskedastische Unsicherheit wird nach Kendall et al. [135] anhand einer Gauß-Verteilung modelliert, deren Varianz σ während des Trainingsprozesses erlernt wird. Diese gewichtete Fehlerfunktion hat allerdings in ihrer unveränderten Form einen entscheidenden Nachteil. Nähert sich σ für eine Komponente einem Wert von 0, ergibt sich hieraus:

$$\lim_{\sigma \rightarrow 0} \frac{1}{2\sigma^2} L + \log(\sigma^2) = -\infty \quad (4.5)$$

Dies ist eine für Fehlerfunktionen ungünstige Eigenschaft, kann allerdings durch eine Modifikation leicht korrigiert werden:

$$\mathcal{L}_{MT2} = \sum_i \left(\frac{1}{2\sigma_i^2} L_{t_i} + \log(\sigma_i^2 + 1) \right) \quad (4.6)$$

Die unterschiedlichen Verläufe beider Fehlerfunktionen sind in Abbildung 4.8 nochmals dargestellt. Als Fehlerfunktion L_{t_i} für die einzelnen Komponenten wird der in Kapitel 6 vorgestellte *Automated Focal Loss* verwendet.

4.3.4 Nachverarbeitung der Netzausgabe

Vergleichbar mit der in Kapitel 3 vorgestellten Netzarchitektur zur 2D Objektdetektion sind auch die rohen Ausgaben der Netze zur 3D Detektion dynamischer

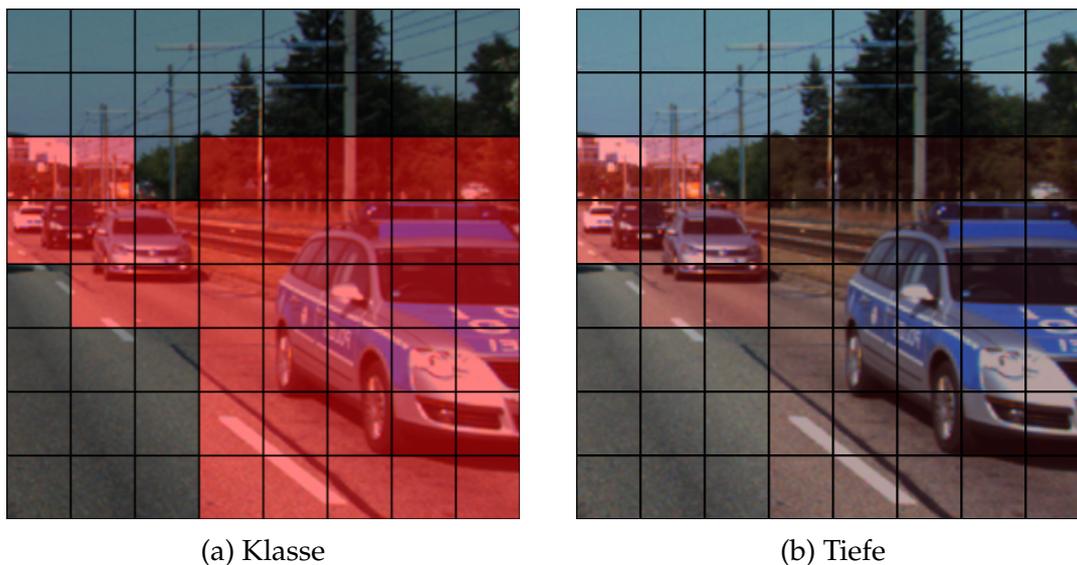


Abbildung 4.9: Ausgaben des Detektornetzes. Eine Zelle des Gitternetzes symbolisiert an dieser Stelle eine Region im Bild. Einzelne Werte sind jeweils für die Regionen eingeblendet, in denen sich erkannte Objekte befinden. Erstveröffentlichung in: [6]

Verkehrsobjekte nicht direkt als finale Detektionen nutzbar. Dies ergibt sich bereits aus der den beiden Architekturen zugrundeliegenden engmaschigen Gitterstruktur der Bildregionen, welche in Abbildung 4.9 dargestellt ist. Da die Ausgabe des Netzes für jede Region einen Objektkandidaten enthält und ein reales Objekt meist mehrere Regionen umfasst, ist eine Mehrfachdetektion einzelner Objekte dem Verfahren immanent.

Um aus den Objektkandidaten der einzelnen Regionen die finalen Detektionen zu erhalten, wird zunächst eine Schwellwertfilterung anhand der vom Netz geschätzten Klassenwahrscheinlichkeit durchgeführt. So können die Objektkandidaten bereits auf wenige Kandidaten eingegrenzt werden, welchen der Detektor eine hohe Konfidenz zuweist. Bei der 2D Objektdetektion von größeren Objekten wie Fahrzeugen werden, basierend auf diesen wenigen Objektkandidaten, anhand der NMS [168] die finalen Detektionen bestimmt. Hierzu wird für zwei Objektkandidaten ab einem bestimmten Überlappungsschwellwert der Kandidat mit der geringeren Konfidenz verworfen. Bei der Anwendung dieses Verfahrens können allerdings nützliche Informationen verloren gehen. So können zugunsten eines Objektkandidaten mit ungenauer Positionsschätzung aber hoher Konfidenz mehrere andere Kandidaten mit geringerer Konfidenz aber genauerer Schätzung der Position verworfen werden.

Aus diesem Grund wird für den hier vorgestellten Ansatz eine probabilistische NMS verwendet, welche auf die in Vogelperspektive projizierten Objektkandidaten angewandt wird. Bei dieser Variante werden die verworfenen Objektkandidaten in einer Liste gespeichert. Nachdem die finalen Kandidaten feststehen, werden aus Gründen der Performance die n besten unterlegenen Kandidaten für

jeden Kandidaten aus der Liste der verworfenen Objektkandidaten ausgewählt. Aus jeder dieser Gruppen wird nun anhand eines gewichteten Mittelwertes eine finale Detektion berechnet. Die Gewichtung der einzelnen Kandidaten ist hierbei durch die Klassenwahrscheinlichkeit gegeben. Hiermit wird für jede Gruppe die wahrscheinlichste Detektion gegeben alle Kandidaten dieser Gruppe ermittelt.

4.4 Experimente und Evaluation

Zur experimentellen Evaluation des in Abschnitt 4.3 vorgestellten Ansatzes zur Detektion dynamischer Verkehrsobjekte werden die eingeführten Netzarchitekturen auf dem nachfolgend in Abschnitt 4.4.1 vorgestellten *KITTI 3D Object Detection Datensatz* trainiert und im direkten Vergleich evaluiert. Die hierzu verwendeten Evaluationsmetriken ergänzen die bereits in Kapitel 3 eingeführten Metriken zur Objektdetektion und werden in Abschnitt 4.4.2 vorgestellt. Eine vergleichende Evaluation der in diesem Kapitel vorgestellten Konzepte und Architekturen ist in Abschnitt 4.4.3 dargestellt.

4.4.1 Datensätze

Als Grundlage der Evaluation von den in diesem Kapitel vorgestellten Ansätzen zur Detektion dynamischer Verkehrsobjekte in 3D Kamerakoordinaten wird der Datensatz aus dem KITTI 3D Object Detection Benchmark [96] verwendet. Der Datensatz sowie die Anpassungen und verwendete Datenaugmentierung werden nachfolgend detailliert erläutert.

KITTI 3D Object Detection Datensatz

Der KITTI 3D Object Detection Datensatz [96] ist bezüglich der Bilder und annotierten Objekte weitgehend identisch zu dem in Abschnitt 5.4.1 beschriebenen KITTI 2D Detection Datensatz. Der hier beschriebene 3D Datensatz enthält ebenfalls Annotationen für die Klassen *PKW*, *LKW*, *Fußgänger*, *Fahrrad* und *Straßenbahn*. Der Datensatz besteht aus 7.481 Trainings- und 7.518 Testbildern mit der Auflösung 1.242×375 Pixeln. Soweit überprüfbar wurden dieselben Objekte annotiert – für diesen Datensatz erfolgte dies allerdings in Form von 3D Bounding Boxen, also minimal umschließenden Quadern. Diese Annotationen wurden erzeugt durch manuelles Annotieren von ebenfalls aufgezeichneten LiDAR Punktwolken, welche in das Kamerabild projiziert wurden. Die Annotation liegen als Textdateien mit folgenden Informationen vor: Position (x, y, z) , Rotation θ (Gierwinkel), Objekttyp, Breite, Höhe, Länge und Kalibrierungsmatrix.

Für die im Rahmen dieser Arbeit durchgeführten Experimente wurde der Datensatz zur Augmentierung der Daten angepasst. Aus jedem Bild wurden 20 zufällige Ausschnitte der Größe 256×256 Pixel ausgeschnitten. Diese Ausschnitte wurden den Trainingsdaten hinzugefügt, falls sich ein Objekt innerhalb des entsprechenden Ausschnittes befindet. Der finale Trainingsdatensatz beinhaltet aufgrund dieser Augmentierung 149.620 Bilder der Größe 256×256 Pixel.

4.4.2 Evaluationsmetriken

Wie bereits in Kapitel 3 beschrieben, erfolgt die Evaluation im Rahmen dieser Arbeit einzelbildbasiert. Es werden also für jedes Bild des Evaluationsdatensatzes die vorhandenen dynamischen Verkehrsobjekte D detektiert und mit den annotierten Objekten A verglichen. Dies ist detailliert in Abschnitt 3.4.2 beschrieben. Im Gegensatz zu den dort verwendeten Metriken Precision, Recall und F1-Score werden für den KITTI 3D Object Detection Benchmark im Wesentlichen die Metriken [Average Precision \(AP\)](#) und [Average Orientation Similarity \(AOS\)](#) verwendet [96].

Diese [Average Precision](#) basiert indirekt ebenfalls auf den Werten von Precision und Recall. Anders als bei dem auch auf diesen Werten basierenden F1-Score werden hierfür allerdings keine festen Grenzwerte für die Detektion eines Objektes festgelegt. Es existiert also keine binäre Entscheidungsgrenze. Die Grundannahme ist an dieser Stelle, dass durch den Objektdetektor jedem Objekt ein Score oder eine Wahrscheinlichkeit zugeordnet wird. Basierend auf diesen Scores und der Korrektheit der Detektion wird die sogenannte [Precision-Recall \(PR\)](#)-Kurve erstellt. Diese stellt die Abhängigkeit von Precision als Ordinate und Recall als Abszisse über alle Scores hinweg dar. Meist wird diese [PR](#)-Kurve in der Form geglättet, dass konvexe Abschnitte der Kurve auf das mit steigender Abszisse folgende, lokale Maximum angehoben werden, um eine monoton fallende Kurve zu erhalten.

Die [Average Precision](#) entspricht nun einer Approximation der Fläche unterhalb der geglätteten [PR](#)-Kurve. Gemäß [PASCAL VOC](#) [80] ist die [AP](#) definiert als arithmetisches Mittel von geglätteten Precision-Werten p an einer Reihe von 11 gleichmäßig auf der Abszisse verteilten Stellen $\{0, 0,1, \dots, 1\}$:

$$\text{Average Precision} = \frac{1}{11} \sum_{r \in \{0,0.1,\dots,1\}} \max_{\tilde{r}:\tilde{r}>r} p(\tilde{r}) \quad (4.7)$$

Der Bewertung anhand der [AP](#) liegt auch immer eine Maßzahl für die minimale Überlappung der verglichenen Bounding Boxen nach Formel (3.10) zugrunde. Hiermit wird festgelegt, welcher Überlappungsgrad zwischen Detektion und Annotation minimal notwendig ist, damit beide als ein Objekt gezählt werden. In dieser Evaluation wird, wie in [96] vorgeschlagen, eine [IoU](#) von 0,5 verwendet. Entsprechend kann hierfür auch die Schreibweise [AP@\[.5\]](#) verwendet werden.

Diese Schreibweise wird im Wesentlichen für die Evaluation auf dem COCO-Datensatz verwendet, da hier die durchschnittliche AP für mehrere IoU-Werte berechnet wird. Dies wird meist als **mean Average Precision (mAP)** bezeichnet. Wird eine Folge von IoU-Werten mit einer festen Schrittweite angegeben, lautet die Schreibweise hierfür $AP@[start:step:stop]$. Die Berechnung wird also beginnend mit dem *start*-Wert mit der Schrittweite *step* wiederholt, bis der *stop*-Wert erreicht ist. Im Rahmen dieser Evaluation wird die AP nicht direkt auf den Bounding Boxen im Bildraum berechnet. Stattdessen werden die Ergebnisse zunächst in eine Vogelperspektive (engl. Bird’s-Eye View) überführt. Die Maßzahl wird deshalb als AP_{BE} bezeichnet.

Die **Average Orientation Similarity (AOS)** wurde für den KITTI 3D Object Detection Benchmark eingeführt [96]. Sie ist bei gegebenem Recall $r = \frac{TP}{TP+FN}$ definiert als:

$$AOS = \frac{1}{11} \sum_{r \in \{0,0.1,\dots,1\}} \max_{\tilde{r}: \tilde{r} > r} s(\tilde{r}) \quad (4.8)$$

Die *Orientation Similarity* $s \in [0, 1]$ für r entspricht einer normalisierten Variante der Kosinus-Ähnlichkeit:

$$s(r) = \frac{1}{|D(r)|} \sum_{i \in D(r)} \frac{1 + \cos(\Delta_{\theta}^{(i)})}{2} \cdot \delta_i \quad (4.9)$$

mit $D(r)$ der Menge aller Detektionen bei Recall r und $\Delta_{\theta}^{(i)}$, der Abweichung der Orientierung von Detektion und Annotation für Objekt i . Der Faktor δ_i entspricht hierbei einem binären Indikator, ob eine Detektion einer Annotation zugewiesen wurde. Weitere Details zur Definition der AOS sind zu finden in [96].

4.4.3 3D Objektdetektion

Für die vergleichende Evaluation der verschiedenen Architekturen zur 3D Objektdetektion wurden die in Abschnitt 4.3.2 vorgestellten Modelle auf den Trainingsdaten des KITTI 3D Object Detection Datensatzes trainiert. Die Ergebnisse der Evaluation sind in Tabelle 4.2 dargestellt.

Architektur	AP_{BE}	AOS	FPS	Konvergenzdauer
Direct3D	25,0	37,3	28,18	6 Std. 30 Min.
OverFeat3D	27,38	34,47	18,92	7 Std.
SSD3D	-	9,1	27,28	9 Std.
RetinaNet3D	OOM	OOM	OOM	9 Std.

Tabelle 4.2: Ergebnisse für die verschiedenen Netzarchitekturen auf dem KITTI 3D Object Detection Datensatz.

Hierbei fallen zunächst die Ergebnisse für die RetinaNet3D-Architektur auf. Aufgrund des limitierten Speichers der verfügbaren GPUs von 12 GB konnte das

Netz nicht erfolgreich evaluiert werden. Da eine Verwendung in einem Autonomen Fahrzeug unter realen Bedingungen mit diesem Ressourcenbedarf ebenfalls nicht möglich ist und andere Netze mit geringerem Bedarf verfügbar sind, wurde diese Architektur von der weiteren Evaluation ausgeschlossen.

Das auf [SSD](#) basierende SSD3D-Modell erreichte in der Evaluation im Vergleich zu den anderen Modellen die schlechtesten Ergebnisse. Dies verwundert auf den ersten Blick, da das Modell zum Ende des Trainingsprozesses den geringsten Fehler auf den Trainingsdaten erreicht. Allerdings scheint bei diesem Modell trotz der Verwendung von Dropout [216] und einer L2-Normalization ein Overfitting auf den Trainingsdaten stattzufinden. Eine mögliche Erklärung hierfür ist, dass für das Modell die vergleichsweise große Anzahl an Parametern außerhalb des Encoders, welche nicht durch bereits vorhandene Gewichte vorinitialisiert werden konnten, ein Problem darstellen. Hierbei ist zu beachten, dass für einen derart komplexen Task die Menge der Trainingsdaten relativ gering ist.

Die Modelle von Direct3D und OverFeat3D lieferten beide bereits recht gute Ergebnisse, die allerdings bei Betrachtung der einzelnen Metriken geringfügig unterschiedlich ausfallen. So erzielte das Direct3D-Modell einen mit knapp drei Punkten besseren [AOS](#). Das auf OverFeat basierende Modell erzielte hingegen eine deutlich höhere AP_{BE} . Dies lässt sich erklären durch eine deutlich höhere Ausgabeauflösung des OverFeat3D-Modells, welche durch die Tiling-Schichten erreicht wird. So erfolgt die Prädiktion in diesem Modell mit einer deutlich höheren Granularität. Dies hat allerdings auch Auswirkungen auf die Laufzeit des Modells. So kommt das Direct3D-Modell auf eine Ausführungsfrequenz von 28 [FPS](#), während das OverFeat3D-Modell mit knapp 19 [FPS](#) lediglich 2/3 davon erreicht.

In einer qualitativen Evaluation der Ergebnisse des Direct3D-Modells wurden zusätzlich insbesondere die Fehler analysiert. In [Abbildung 4.10](#) sind beispielhaft in zwei Abbildungen die häufigsten Fehler des Modells dargestellt. So werden vielfach weit entfernte Objekte nicht gut erkannt, insbesondere dann, wenn sie durch andere Objekte teilverdeckt sind oder sich in deren unmittelbarer Umgebung befinden. Zusätzlich ist in [Abbildung 4.10a](#) zu erkennen, dass nur teilweise im Bild sichtbare, nahe Objekte ebenfalls lediglich ungenau geschätzt werden können.

Zusätzlich wurde für den Direct3D-Ansatz die Verwendung unterschiedlicher Encoder-Architekturen zur Extraktion der Features evaluiert. Hierzu wurden, neben der bisher verwendeten VGG16-Architektur, Modelle basierend auf einer VGG19-, einer Inception-v3- und einer ResNet50-Architektur trainiert. Diese Modelle erzielten minimal schlechtere, aber vergleichbare Ergebnisse. Der Unterschied der einzelnen Ergebnisse lag für den [AOS](#) bei weniger als 0,1 Prozentpunkten. Somit scheinen die bei dieser Evaluation für den Direct3D-Ansatz erreichten Ergebnisse auch auf andere Architekturen übertragbar zu sein.

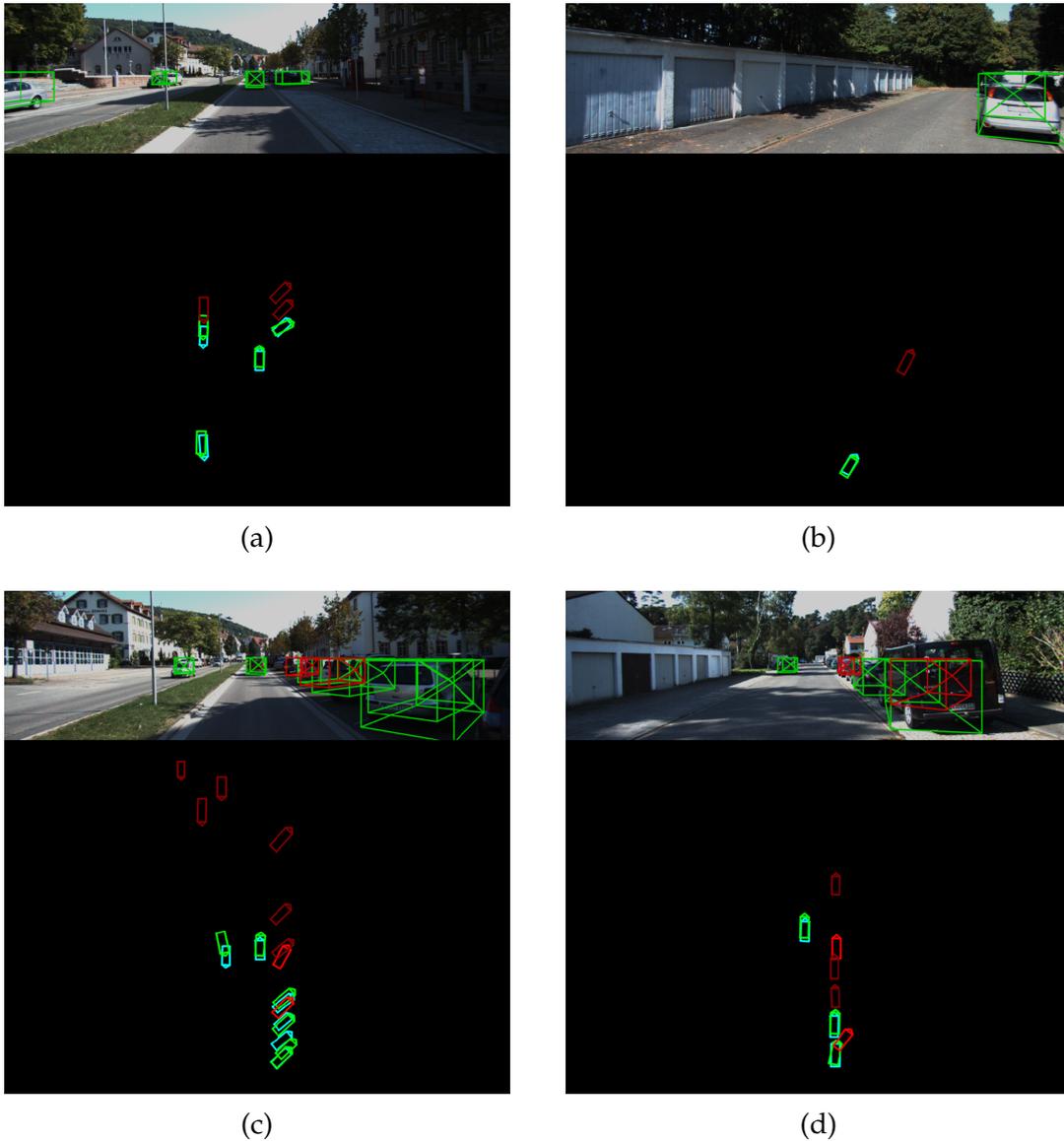


Abbildung 4.10: Häufige Fehler des Direct3D-Systems auf dem KITTI 3D Object Detection Datensatz. Korrekte Detektionen sind in Grün dargestellt, die Annotation in Cyan. Nicht erkannte Objekte sind dunkelrot dargestellt, Fehldetektionen hellrot. Wie in der Vogelperspektive zu sehen ist, enthält der Datensatz auch kleinere Fahrzeuge, die sich oft in der Nähe anderer Autos befinden (a), (c) oder teilweise von ihnen verdeckt werden (b), (d). In beiden Fällen stößt das System an seine Grenzen und erkennt die Objekte meist nicht. Erstveröffentlichung in: [6]

4.5 Diskussion

Im Rahmen dieses Kapitels wurde ein CNN-basierter Ansatz zur direkten Schätzung von dynamischen Verkehrsobjekten in 3D Kamerakoordinaten vorgestellt. Die Parameter der Objekte werden mit dem vorgestellten Verfahren direkt durch das CNN-Modell geschätzt. Im Gegensatz zu dieser direkten Schätzung verwenden vergleichbare verwandte Arbeiten meist Zwischenrepräsentation (OFT) oder Annahmen zur Szenengeometrie (BS³D), um die Position und Größe der Objekte im 3D Raum zu schätzen.

Basierend auf dem vorgestellten Ansatz wurde eine Direct3D-Architektur eingeführt, welche die 3D Detektion basierend auf verschiedenen Encoder-Netzen zur Extraktion der Features durchführen kann. Diese kann direkt in Form eines Heads als Teil des in Kapitel 5 vorgestellten MultiNet-Prinzips verwendet werden. Statt auf einem Encoder aufbauend kann der vorgestellte Ansatz auch eine Architektur zur 2D Detektion als Grundlage verwenden. So wurden ebenfalls Architekturen für die Integration der Detektoren SSD, OverFeat und RetinaNet eingeführt.

Neben der originalen Direct3D-Architektur lieferte von diesen Architekturen vor allem die OverFeat3D-Architektur bereits sehr gute Ergebnisse. Beide Architekturen scheinen grundsätzlich in der Lage zu sein, dynamische Verkehrsobjekte basierend auf monokularen Bilddaten zu schätzen. Auch die Ausführungszeiten erfüllen mit einer Ausführungsfrequenz von mehr als 25 FPS die in Abschnitt 4.1 geforderten Zeiten deutlich, wobei hier analog zu den in Kapitel 3 vorgestellten Modellen noch keine nachgelagerte Optimierung durchgeführt wurde.

Ob die Qualität der erreichten Ergebnisse allerdings bereits ausreichend ist, um darauf basierend eine Fahrentscheidung zu treffen, kann an dieser Stelle nicht abschließend beantwortet werden. Insbesondere räumlich nahe Objekte sowie Fußgänger und Fahrradfahrer sind in dem verfügbaren Datensatz deutlich unterrepräsentiert. Gerade eine präzise Detektion dieser Objekte ist jedoch von hoher Wichtigkeit. Allerdings bieten die durch Direct3D gewonnenen Detektionen eine gute visuelle Grundlage zur Fusion der Detektionen von anderen Sensoren, wie z. B. LiDAR. So kann eine präzise Detektion plausibilisiert und deren Klassenzugehörigkeit durch eine visuelle Klassifikation präzisiert werden. Insbesondere ist es so auch möglich, dass spezielle Objekte, wie beispielsweise Einsatzfahrzeuge, korrekt erkannt werden.

5 Umfelderkennung mittels MultiTask-Netzen

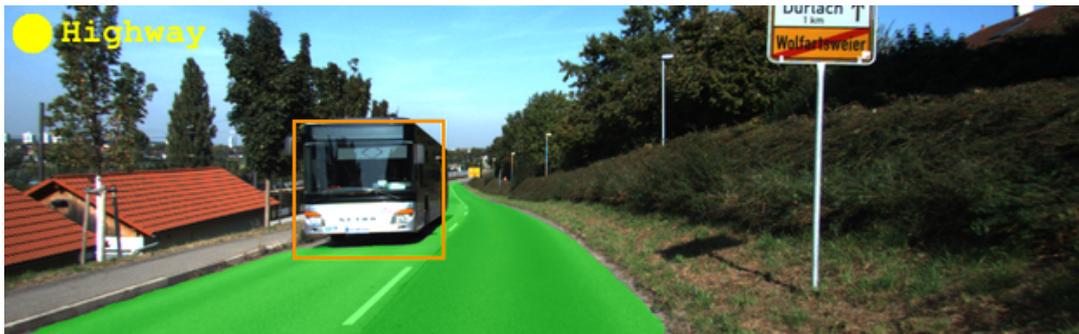


Abbildung 5.1: Ergebnis eines MultiNet-Modells für die drei Tasks Straßensegmentierung (grün), Fahrzeugdetektion (orange) und Szenenklassifikation (gelb) in einem Bild. Ähnlich veröffentlicht in: [5]

Für die Wahrnehmung des Umfelds in einem Autonomen Fahrzeug sind ebenso wie in anderen Domänen vielfältige Erkennungsaufgaben auf den einzelnen durch eine oder mehrere Kameras aufgenommenen Bildern durchzuführen. Für die einzelnen Erkennungsaufgaben wird meist ein eigener spezialisierter Algorithmus zur Bildverarbeitung basierend auf [CNNs](#) eingesetzt. So müssen für die Durchführung jedes dieser Tasks Ressourcen vorgehalten werden. Insbesondere in mobilen Anwendungen wie dem Autonomen Fahren sind die vorhandenen Ressourcen sowie die verfügbare Energie stark beschränkt. Dies stellt gerade für die aktuellen, auf ressourcenintensiven tiefen Neuronalen Netzen aufbauenden Ansätze eine Herausforderung dar.

Aus diesem Grund werden Konzepte benötigt, um einzelne Erkennungsaufgaben effizienter durchführen zu können, ohne die Qualität von deren Ergebnissen negativ zu beeinflussen. Um dies zu erreichen, wird in diesem Kapitel der MultiNet-Ansatz zur gemeinsamen Verarbeitung verschiedener Perzeptionsaufgaben vorgestellt. Hierbei wird die Extraktion von Features für die verschiedenen Tasks in einem gemeinsamen Netz gebündelt, was eine Reduktion der benötigten Ressourcen über alle Tasks hinweg zur Folge hat. Das Ziel an dieser Stelle ist es, dass ein MultiNet-Modell mit der Laufzeit des langsamsten Tasks alle Tasks erledigen kann. Hierbei muss im Vergleich zur Einzelausführung die Qualität der Ergebnisse erhalten bleiben, sowie der Ressourcenbedarf verringert werden.

In Abschnitt 5.1 wird hierzu zunächst die Problemstellung detailliert beleuchtet sowie der Begriff des MultiTask-Lernens in diesem Kontext eingeführt. Ein Überblick über verwandte Arbeiten im Bereich des MultiTask-Lernens wird in Abschnitt 5.2 gegeben. Für die betrachteten Arbeiten wurde der Fokus an dieser Stelle auf Ansätze zur Reduktion des Ressourcenbedarfs gelegt. In Abschnitt 5.3 folgt die Beschreibung des eigentlichen MultiNet-Ansatzes sowie der Auswahl entsprechender Strategien zum Training eines auf diesem Ansatz beruhenden Systems. In einer umfangreichen Evaluation werden abschließend in Abschnitt 5.4 die Auswirkungen des vorgestellten MultiNet-Ansatzes auf Qualität, Ressourcenverbrauch und Laufzeit der verwendeten Algorithmen untersucht. Teile dieses Kapitels wurden bereits in der Veröffentlichung [5] vorgestellt. Ebenfalls flossen Ergebnisse der Abschlussarbeit [30] in dieses Kapitel ein.

5.1 Problembeschreibung

In vielen Domänen kommen mit gesteigerter Leistungsfähigkeit inzwischen Verfahren der Bildverarbeitung zur Gewinnung von Informationen aus Bilddaten zur Anwendung. Häufig werden pro Bild gleich mehrere Verfahren zur Gewinnung unterschiedlicher Informationen angewendet. In der Domäne des Autonomen Fahrens sind bei einer in Fahrtrichtung gerichteten Kamera beispielsweise folgende Verfahren zur Informationsgewinnung denkbar:

- 2D Detektion statischer Verkehrsobjekte
- 3D Detektion dynamischer Verkehrsobjekte
- Straßensegmentierung
- Tiefenschätzung
- Erkennung von Fahrbahnmarkierungen
- Niederschlagenerkennung
- Szenenklassifikation
- Eigenbewegungsschätzung

wobei diese Aufzählung keinen Anspruch auf Vollständigkeit erhebt. Die Verarbeitungsketten jedes Tasks sind im Normalfall darauf ausgelegt, die erforderlichen Berechnungen für genau diesen auf einem Eingabebild auszuführen und ein Ergebnis zurückzuliefern. Ohne weitere Optimierung werden also viele Verarbeitungsketten auf demselben Eingabebild parallel ausgeführt. Da diese Ketten meist gleichartig aufgebaut sind und somit ähnliche Verfahren verwendet werden, finden auch vergleichbare Berechnungen mehrfach statt, wie in Abbildung 5.2 dargestellt.

Unter diesen Voraussetzungen werden Methoden und Mechanismen benötigt, den durch die parallele Ausführung einer größeren Anzahl von Algorithmen

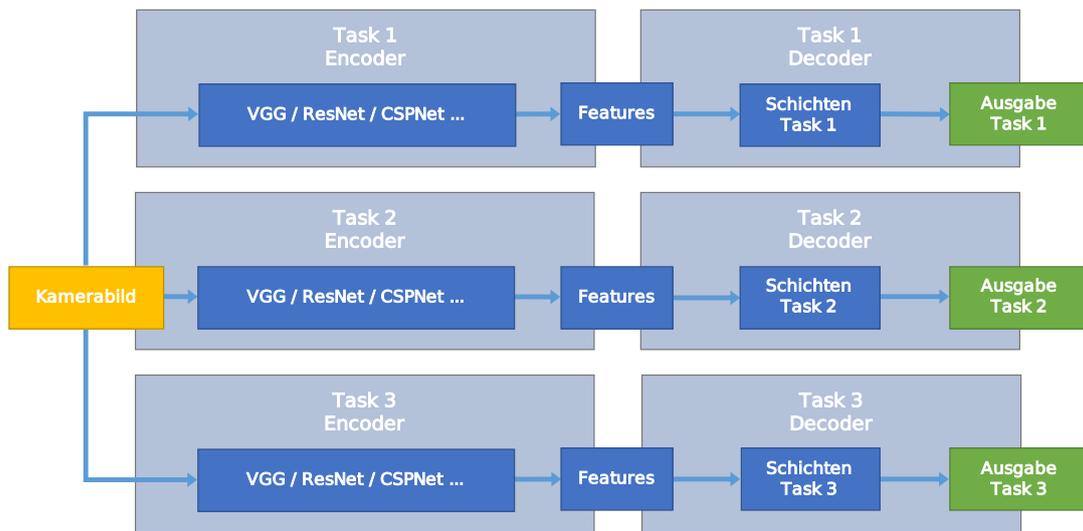


Abbildung 5.2: Getrennte Ausführung mehrerer CNNs zur Erfüllung verschiedener Tasks für die Umgebungswahrnehmung.

entstehenden Ressourcenbedarf zu reduzieren. Für die in aktuellen Systemen durchgängig verwendeten CNNs setzt sich dieser Ressourcenbedarf, wie in Abschnitt 2.2 beschrieben, im Wesentlichen aus der Anzahl vorhandener Parameter sowie der Menge benötigter Rechenoperationen zusammen. Die in den auszuführenden Modellen vorhandenen Parameter wirken sich hierbei in zweifacher Hinsicht aus. So müssen einerseits die Modelle im Speicher der verwendeten Systeme vorgehalten werden, damit sie bei Systemstart geladen werden können. Andererseits müssen die Parameter zur Laufzeit im Arbeitsspeicher der Recheneinheit vorgehalten werden. Diese Auswirkungen müssen hauptsächlich bei der Dimensionierung des Systems betrachtet werden. Sie haben zur Ausführungszeit keinen Einfluss auf die Laufzeit der einzelnen CNNs.

Im Gegensatz hierzu hat die Anzahl der von jedem CNN benötigten Rechenoperationen durchaus einen großen Einfluss, sowohl auf dessen Laufzeit, als auch auf die Laufzeit aller auf der Recheneinheit ausgeführten CNNs. Betrachtet man ein einzelnes CNN, so ist die minimale Ausführungszeit gegeben durch die sequentielle Ausführung aller nicht parallel ausführbaren Operationen. Genau genommen ist zur Bestimmung der realen Ausführungszeit zusätzlich die Taktfrequenz der Recheneinheit zu betrachten. Diese kann bei dieser Betrachtung allerdings ausgeklammert werden, da im Wesentlichen die generelle Auslastung der Einheiten betrachtet wird. Soll ein System für ein konkretes Einsatzszenario mit festgelegten maximalen Laufzeiten konzipiert werden, ist die Taktfrequenz hingegen ein relevanter Faktor.

Generell ist für die Konzeption eines Systems ein Kompromiss zwischen der Anzahl der enthaltenen Recheneinheiten und der Laufzeit der darauf ausgeführten CNNs zu schließen. Sofern für einen Task eine maximale Laufzeit vorgegeben ist, wird damit bei gegebenem Modell bereits die Anzahl der benötigten Recheneinheiten vorgegeben. Somit wird der Fokus wieder auf das trainierte Modell ge-

richtet. Ist die maximale Laufzeit vorgegeben und die Anzahl der Recheneinheiten durch z. B. die entstehenden Kosten nach oben begrenzt, verbleibt lediglich das Modell als variable Größe. So können Methoden zur Reduktion der Rechenoperationen in einem Modell die Ausführung eines Tasks durch Verringerung der Laufzeit unter die Schwelle der Maximallaufzeit erst ermöglichen oder die Kosten für das Hardwaresystem senken.

Zusätzlich kann so auch die parallele Bearbeitung weiterer Tasks auf einer einzelnen Recheneinheit ermöglicht werden. Neben allgemeinen Techniken zur Reduktion von Parametern und somit auch Rechenoperationen in einem Modell, die in Kapitel 7 behandelt werden, kann hierfür auch das sogenannte MultiTask-Lernen [51] zum Einsatz kommen. Der Begriff MultiTask bezeichnet hierbei die gemeinsame Lösung mehrerer Tasks in einer gemeinsamen Komponente. Übertragen auf CNNs entspricht dies der Lösung mehrerer Tasks in einem einzigen CNN-Modell.

Eine der Grundannahmen des MultiTask-Lernens ist, dass erlerntes Wissen für einen Task auch für andere Tasks nützlich sein kann [51][229][52]. Insbesondere bei verschiedenen Tasks aus einer Anwendungsdomäne trifft dies auf ein gewisses Grundwissen über die Domäne zu. Je nach verwendetem Ansatz kann durch ein MultiTask-CNN die Anzahl der Parameter und damit auch die Anzahl der Rechenoperationen verringert werden. Betrachtet man vergleichend die Summe der Ausführungszeiten von den einzelnen Tasks mit der Ausführungszeit eines gemeinsamen MultiTask-Netzes, kann meist auch diese signifikant reduziert werden.

Da in einem MultiTask-Netz isoliert betrachtet zumindest das Netz jedes Tasks ausgeführt wird, kann die Laufzeit für das MultiTask-Netz nie unterhalb der Laufzeit für das langsamste Netz für einen der Tasks sein, welches separat auf gleicher Hardware ausgeführt wird. Im Optimalfall entspricht die Laufzeit des MultiTask-Netzes dieser langsamsten Einzellaufzeit. Sind ausreichend Hardware-Ressourcen verfügbar, ist dies immer der Fall. In diesem Fall wird allerdings auch bei paralleler Ausführung der einzelnen Tasks keine höhere Laufzeit zu beobachten sein.

Relevant für die Verringerung des Ressourcenverbrauchs wird ein MultiTask-Netz erst dann, wenn nicht mehr ausreichend Recheneinheiten zur parallelen Ausführung aller Tasks verfügbar sind. In diesem Fall steigt die summierte Laufzeit über alle Tasks an. Wird ein MultiTask-Netz zur Reduktion des Ressourcenverbrauchs eingesetzt, ist diese Summe eine obere Grenze der Laufzeit, die möglichst stark unterschritten werden muss. Die Möglichkeiten zur Optimierung liegen also zwischen der Laufzeit des langsamsten Netzes für einen Task und der Summe aller Laufzeiten der auf gleicher Hardware parallel durchgeführten Tasks.

5.2 Verwandte Arbeiten

Die Kombination verschiedener Tasks in einem Neuronalen Netz geht zurück auf die Arbeiten von Caruana [51]. Dort wurden 1993 auch die Begriffe MultiTask-Lernen und im Speziellen MultiTask-Netze für derartige Netze geprägt. Ziel dieser Arbeiten war allerdings noch nicht die Reduktion der benötigten Ressourcen, sondern die gegenseitige Verbesserung der einzelnen Tasks in einem gemeinsamen Lernprozess. Hierzu wurden die Neuronen zur Erfüllung der einzelnen Tasks mit einem gemeinsamen **Multi-Layer Perceptron (MLP)** zur Extraktion der Features verbunden.

Es konnte gezeigt werden, dass durch die Verwendung von MultiTask-Netzen die Qualität der Ergebnisse in den einzelnen Tasks verbessert werden konnte. Dies wurde erklärt mit einer besseren Generalisierungsfähigkeit. Auch für tiefe Neuronale Netze wurde die Steigerung der Generalisierungsfähigkeit beobachtet [104]. Ob allerdings die wechselseitige Qualitätssteigerung der einzelnen Tasks direkt auf tiefe Netze übertragen werden kann, erscheint fraglich. So wurden ursprünglich recht kleine Datensätze zum Training eines Tasks verwendet. So konnten zusätzliche Trainingsdaten eines weiteren Tasks auch das Training grundlegender Features noch deutlich verbessern. Die durch heutige **CNNs** gelösten Tasks sind hingegen deutlich komplexer und enthalten auch aufgrund der um ein Vielfaches gesteigerten Menge an Trainingsdaten bereits intensiv trainierte grundlegende Features. Hierbei spielt auch die Tatsache eine Rolle, dass die Gewichte in **CNNs** unabhängig von der aktuellen Bildposition trainiert werden.

Für die in diesen ersten MultiTask-Netzen verwendete Technik der gemeinsam genutzten Schichten wurde später der Begriff des *Hard Parameter Sharing* eingeführt [199]. Dieser fasst alle MultiTask-Ansätze zusammen, welche als Basis der Neuronen für die einzelnen Tasks ein gemeinsames Netz besitzen. Der Name leitet sich dabei aus der Tatsache ab, dass durch die gemeinsame Grundarchitektur alle Tasks dieselben gelernten Parameter verwenden. Der Gegenentwurf hierfür ist das *Soft Parameter Sharing*, bei dem jeder Task auf einem eigenen Netz beruht. Allerdings existieren Abhängigkeiten zwischen den Parametern der einzelnen Netze. So wird eine Regularisierung durchgeführt, damit sich letztendlich ähnliche Parameter in den einzelnen Netzen ausprägen [199]. Beide Paradigmen sowie deren Vergleich sind in Abbildung 5.3 dargestellt.

Neben **CNN**-basierten Ansätzen, die dem klassischen Soft Parameter Sharing folgen [73][241], existieren inzwischen auch erweiterte Ansätze, welche auch direkt Informationen zwischen den Teilnetzen austauschen. Diese werden mitunter auch als *Cross-Talk*-Netze bezeichnet [64]. Zwischen den Netzen der verschiedenen Tasks werden in diesen Ansätzen meist die Resultate der einzelnen Schichten geteilt. Zwei konkrete, auf einer **CNN**-Architektur basierende, Ausprägungen dieses Ansatzes sind die Cross-Stitch Networks [164] und das NDDR-CNN [93].

5 Umfelderkennung mittels MultiTask-Netzen

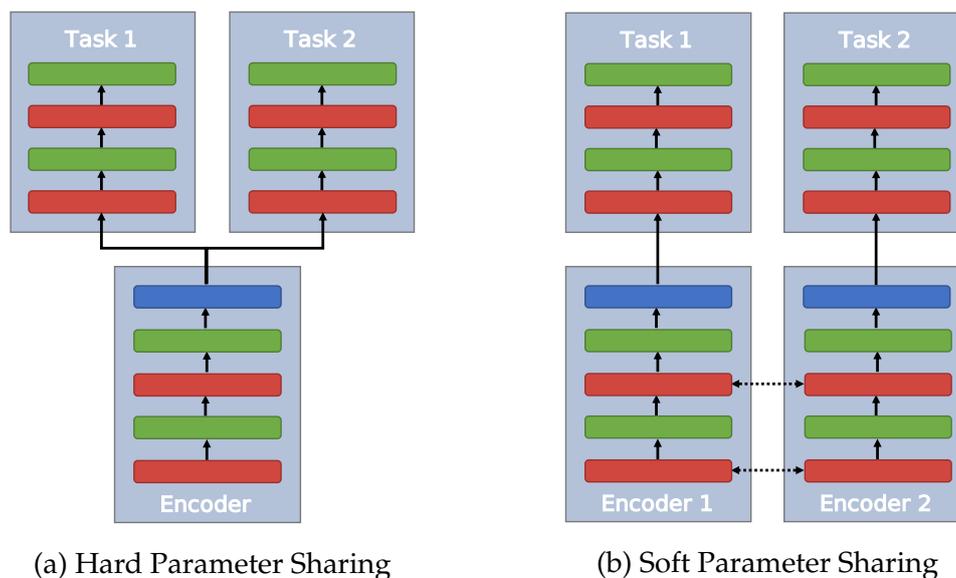


Abbildung 5.3: Beim Hard Parameter Sharing (a) werden für alle Tasks gemeinsame Schichten als Grundlage verwendet – die Parameter für die einzelnen Tasks sind also identisch. Soft Parameter Sharing (b) hingegen verwendet separate Parameter. Zwischen den Parametern der einzelnen Schichten wird allerdings eine Regularisierung durchgeführt.

Bei den Cross-Stitch Networks werden in sogenannten Cross-Stitch Units nach jeder Pooling-Schicht die Ergebnisse der verschiedenen Netze in einer Linearkombination zusammengeführt und an die nächste Schicht der Netze zurückgeführt. Im Gegensatz hierzu werden in den NDDR-CNNs jeweils nach n Convolution-Schichten die Feature Maps der einzelnen Netze konkateniert. Im Anschluss wird auf dem Ergebnis für jedes Netz durch eine netzspezifische Convolution-Schicht mit 1×1 Filterkernen eine Reduktion der Dimension auf die Größe der ursprünglichen Feature Maps vorgenommen.

Zu beachten ist an dieser Stelle, dass mit Hard- und Soft Parameter Sharing in aktuellen Netzarchitekturen meist unterschiedliche Strategien verfolgt werden. So dienen Architekturen nach dem Prinzip des Soft Parameter Sharing im Wesentlichen dem Austausch von Informationen zwischen den verschiedenen Netzen der einzelnen Tasks mit dem Ziel, die Qualität der Ergebnisse für die einzelnen Tasks zu verbessern. Hard Parameter Sharing hingegen wird inzwischen hauptsächlich verwendet, um den Ressourcenverbrauch der Netze zu reduzieren.

Gerade bei komplexen CNNs zur Lösung von Perzeptionsaufgaben, die meist sehr ressourcenintensiv sind, können durch die gemeinsame Nutzung von Teilen des Netzes große Einsparungen erreicht werden. Mit der Verbreitung der CNNs im Gebiet der Bildverarbeitung wurden auch bald erste Ansätze gesucht, MultiTask-Architekturen basierend auf Hard Parameter Sharing zur Reduktion der Netzgröße und somit zur parallelen Ausführung mehrerer Tasks zu erstellen.

Bereits die ersten auf CNNs basierenden Objektdetektoren wie OverFeat [205] verwendeten parallele Schichten zur Klassifikation der Objekte und Regression der umgebenden Bounding Boxen, aufbauend auf einem gemeinsamen Netz. Hierbei kamen jedoch im Wesentlichen identische Schichtfolgen zum Einsatz. Ob diese Schichten in separaten Strängen oder allein durch das aus AlexNet [144] bekannte Channel Grouping realisiert wurden, war lediglich ein Implementierungsdetail. Einen etwas anderen Ansatz verfolgte das Tasks-Constrained Deep Convolutional Network [247]. So wurde ein gemeinsames CNN verwendet, an dessen Ende eine FC-Schicht den Übergang zu den Regressionsmodulen der einzelnen Tasks bildete. Einen ähnlichen Ansatz verfolgte UberNet [142] mit einem gemeinsamen CNN und je einer separaten, finalen Schicht für jeden Task.

In der Domäne des Autonomen Fahrens wurden zunächst MultiTask-Architekturen entwickelt, bei denen das komplette CNN zwischen den Tasks geteilt ist und die verschiedenen Tasks lediglich in den einzelnen Schichten des Ausgabensors repräsentiert waren [196][231]. So entsprach die Ausgabe für jeden Task 1 bis n Schichten der Netzausgabe. Die Kombination verschiedener Tasks in einem gemeinsamen Netz mit getrennten Ausgaben stammt ursprünglich aus [75] und wurde für verschiedene Tasks der pixelweisen Regression verwendet. Hierbei wird jedem Pixel des Eingabebildes ein kontinuierlicher Wert zugewiesen.

Mit dem im Rahmen dieser Arbeit vorgestellten MultiNet wurde 2016 ein Ansatz eingeführt, bei dem zunächst gemeinsame Features in einem Encoder trainiert werden. Diese Features werden pro Task in einem eigenen, *Head* genannten Teilnetz mit weiteren taskspezifischen Schichten als Eingabe genutzt. Der gemeinsame Encoder ist an dieser Stelle nicht auf eine konkrete Architektur festgelegt. Weitere aktuelle Ansätze werden meist ebenfalls nach dem Prinzip der gemeinsamen Encoder und taskspezifischen Decoder realisiert. In der Domäne Autonomes Fahren verwendete Ansätze sind nachfolgend nochmals genauer bzgl. der Unterschiede zu diesem Prinzip beschrieben. Für eine umfassendere Übersicht über verschiedene MultiTask-Architekturen sei an dieser Stelle verwiesen auf folgende Survey-Artikel: [199][64][233].

RBNet [60] löst den Task der Segmentierung der Straße durch eine Interpretation in Form von zwei getrennten Tasks. So wird neben der generellen Segmentierung die Bestimmung an der Grenze des Straßenbereichs als eigenständiger Task behandelt. Hierdurch wird die Segmentierung an Übergängen der verschiedenen Klassen ähnlich zu U-Net [197] nochmals separat betrachtet. Die Architektur des verwendeten Netzes weist einige Besonderheiten auf. So bestehen die aus dem Encoder resultierenden Features aus einer Konkatenation der Ergebnisse mehrerer Schichten. Die Netzarchitektur der verwendeten Heads ist grundsätzlich identisch, allerdings fließen die Ergebnisse der Grenzbetrachtung zusätzlich zur Ausgabe auch durch weitere Schichten in das Ergebnis der Straßensegmentierung ein.

Kendall et al. [135] verwenden eine MultiNet-artige Architektur zur Untersuchung der Gewichtung einzelner Tasks anhand von homoskedastischer Unsicherheit. Diese basiert auf der DeepLabV3-Architektur [55] zur semantischen Segmentierung. Der Encoder entspricht hierbei einem ResNet101 [114] gefolgt von einem **Atrous Spatial Pyramid Pooling (ASPP)**-Modul. Das Netz besitzt task-spezifische Decoder zur semantischen Segmentierung, Instanzsegmentierung und Tiefenschätzung. Der Task zur semantischen Segmentierung ist hierbei als Klassifikationsproblem, die beiden anderen Tasks als Regressionsproblem definiert.

Oeljeklaus et al. [173] verwenden eine Architektur, welche auf dem MultiNet-Ansatz mit einem gemeinsamen Inception-v2-Encoder [225] basiert. Ähnlich zu der mit MultiNet vorgestellten Evaluationsarchitektur [5] werden die Tasks Straßensegmentierung und Objektdetektion bearbeitet. Ein Klassifikationstask wurde in einer weiterführenden Arbeit [172] hinzugefügt. Lediglich dieser unterscheidet sich mit der Klassifikation der aktuellen Straßentopologie von der in MultiNet durchgeführten Szenenklassifikation.

NeurAll [213] erweitert den MultiNet-Ansatz um eine temporale Komponente. So werden in einem Late-Fusion-Netzwerk [134] aus dem aktuellen sowie dem letzten Bild einer Kamera temporale Features extrahiert. Die Late-Fusion-Architektur beschränkt sich hierbei auf den Encoder des Netzes. Die einzelnen im Rahmen einer Evaluation durch das Netzwerk bearbeiteten Tasks sind Szenensegmentierung, Tiefenprädiktion und die Segmentierung sich bewegender Objekte. Hierbei wird der Task zur Tiefenschätzung als Regressionsproblem betrachtet, die beiden Tasks zur Segmentierung als Klassifikationsproblem. Die spezifischen Schichten der einzelnen Tasks sind, abgesehen von der finalen Schicht zur Entscheidungsfindung, identisch.

ShuDA-RFBNet [237] verwendet eine stark an die einzelnen Tasks angepasste Architektur mit einem teilweise gemeinsam genutzten Encoder. Im Netz werden die Tasks **2D** Objektdetektion und Segmentierung der eigenen und benachbarten Fahrspur in Form von zwei verschiedenen Klassen bearbeitet. In einem ersten Netzteil werden gemeinsame Features aus den Ergebnissen von zwei Schichten ähnlich zu RFBNet konkateniert. Diese werden von beiden Heads als Eingabe für weitere Berechnungen genutzt. Für die Detektion der Objekte werden die Ausgaben der letzten Encoder-Schicht allerdings parallel weiterverarbeitet und angelehnt an SSD [157] über verschiedene Schichten hinweg zur Schätzung von Objektkandidaten verwendet.

5.3 Konzept

Der MultiTask-Ansatz scheint allgemein dafür geeignet zu sein, mehrere Tasks in einer gemeinsamen Netzarchitektur parallel auszuführen und hierbei die benötigten Hardware-Ressourcen deutlich zu reduzieren. Auf dieser Basis wird in diesem Abschnitt der MultiNet-Ansatz zur Lösung von CNN-basierten Tasks in einem in Teilen gemeinsamen Netz eingeführt. Der Ansatz beruht hierbei auf dem Encoder-Decoder-Paradigma mit einem gemeinsamen Encoder für alle Tasks und daran anschließenden, taskspezifischen Decodern. Details zur grundsätzlichen Architektur sowie den der Architekturentscheidung zugrundeliegenden Überlegungen und Abwägungen sind in Abschnitt 5.3.1 dargestellt. Der Trainingsprozess einer MultiNet-Architektur und der Zusammenhang zu verfügbaren Daten wird in Abschnitt 5.3.2 genauer beschrieben.

5.3.1 Architektur

Aktuelle Ansätze zur Bildverarbeitung im Allgemeinen und zur kamerabasierten Perzeption für Autonome Fahrzeuge im Speziellen verwenden meist CNNs als Grundlage für ihre Architekturen. Aus diesem Grund müssen auch bei der Betrachtung eines möglichen Einsparpotentials bei den benötigten Hardware-Ressourcen die Besonderheiten von CNNs einbezogen werden. Insbesondere bei Architekturen nach dem FCN-Prinzip kann mit der Entfernung eines einzelnen Parameters in der Form eines Gewichtes noch keine Aussage zu Einsparungen von Rechenoperationen getroffen werden.

Diese Art von Netzen ermöglicht es, Bilder verschiedener Auflösungen mithilfe des Netzes zu verarbeiten. CNNs verwenden hierbei ein sogenanntes *Weight Sharing*, weshalb im Gegensatz zu einem MLP ein Filterkern auf der kompletten Eingabe mit den gleichen Gewichten angewendet wird. Deshalb ist bereits vor der Ausführung die Anzahl der Parameter im Modell bekannt, die Anzahl der Rechenoperationen kann allerdings erst bestimmt werden, wenn die Eingabegröße feststeht. Mit steigender Eingabegröße steigt somit bei konstanter Parameterzahl die Anzahl der Rechenoperationen in den einzelnen Schichten. Insbesondere für die Schichten des Encoders steigt hierbei die Anzahl der Rechenoperationen für größere Eingabebilder stark an. In Relation zu den weiteren Schichten werden an dieser Stelle in vielen Fällen die meisten Rechenoperationen benötigt [160].

Eine Reduktion der verwendeten Parameter kann mit steigender Eingabebildauflösung an dieser Stelle also großes Einsparpotential auf Seiten der Rechenoperationen ermöglichen. Mit anderem Ziel wurden bereits in der ursprünglichen Arbeit von Caruana [51] die Schichten zur Extraktion der Features zwischen einzelnen Tasks in Neuronalen Netzen geteilt. Diese Bildung von generischen Features aus gemeinsamen Parametern ist auch für tiefe Neuronale Netze möglich [104] und wird zwischenzeitlich als Hard Parameter Sharing bezeichnet [199]. Für das

im Rahmen des MultiNet-Ansatzes verfolgte Ziel der Reduktion von benötigten Ressourcen kann das Hard Parameter Sharing somit verwendet werden. Mit dem Soft Parameter Sharing [199] hingegen könnte lediglich das Ziel der gegenseitigen Steigerung der Ausgabequalität zwischen den einzelnen Tasks erreicht werden.

Wird für die Encoder der einzelnen Tasks ein Hard Parameter Sharing verwendet, entspricht dies einem gemeinsam verwendeten Encoder. Somit können bei einer Netzarchitektur mit n Tasks von der ursprünglichen Anzahl der Parameter und Rechenoperationen in den Encodern

$$\frac{n-1}{n} \tag{5.1}$$

mit einem gemeinsamen Encoder eingespart werden. Das Ziel für diesen gemeinsamen Encoder muss sein, abstrakte Features aus dem Eingabebild extrahieren zu können, auf deren Basis für alle Tasks mit Einzelnetzen vergleichbare Ergebnisse erzielt werden können.

Grundsätzlich ist dies bei CNNs gegeben, da im Normalfall in einem Encoder hierarchische Features gelernt werden [244]. Diese sind insbesondere in den tieferen Stufen sehr generische Features wie Kanten oder Farbübergänge, welche allgemein im Bereich der Bildverarbeitung Verwendung finden. Zusätzlich werden die Parameter der einzelnen Modelle meist zu Beginn des Trainingsprozesses mit gelernten Gewichten eines Klassifikationsnetzes, trainiert auf ImageNet [200], initialisiert. Auch diese gemeinsame Grundlage der gelernten Features verschiedener Tasks ist Hinweis auf ein gewisses Abstraktionslevel.

Durch die Verwendung eines gemeinsamen Encoders kann unter Umständen die Kapazität eines Modells reduziert werden. Dies kann sich auch negativ auf die Qualität des trainierten Modells auswirken, wobei mögliche Auswirkungen im Wesentlichen von der Ähnlichkeit der Tasks abhängig sind. [172] Entstemmen die verschiedenen Tasks allerdings einer gemeinsamen Domäne, kann davon ausgegangen werden, dass eine gewissen Ähnlichkeit zwischen den Tasks besteht und auch die Features eine gewisse Ähnlichkeit aufweisen. Somit erscheint dies nicht als Hindernis für die Realisierung eines MultiTask-Ansatzes, bestehend aus Tasks einer gemeinsamen Domäne. Zusätzlich kann durch das Lernen gemeinsamer Features das Risiko für ein Overfitting im Lernprozess verringert werden [199].

Bezüglich des verwendeten Encoders soll der MultiNet-Ansatz möglichst unabhängig von einer konkreten Architektur sein. Aktuell existiert kein global überlegener Encoder, auch werden in schneller Folge neue Architekturen erforscht. Diese neuen Erkenntnisse sollen möglichst nahtlos in den MultiNet-Ansatz übertragen werden können. Auch muss eine konkrete Architektur an den Anwendungsfall und die verfügbare Hardware angepasst werden. Hierzu können auch weitere Reduktionstechniken, wie in Kapitel 7 vorgestellt, angewandt werden. Dass eine Architektur unabhängig von einem konkreten Encoder realisierbar ist, konnte bereits für die Ansätze zur Detektion von Objekten in Kapitel 3 und 4 gezeigt werden.

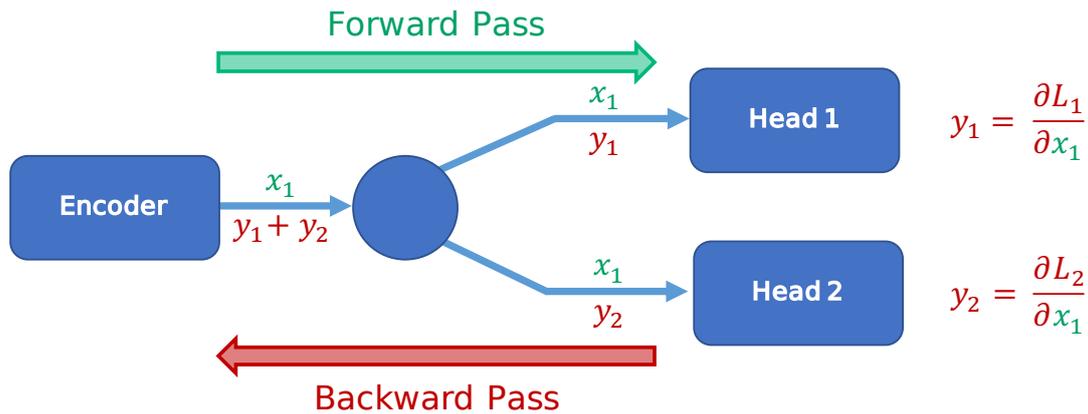


Abbildung 5.4: Copy Gate zur Anbindung der Heads für die einzelnen Tasks an den gemeinsamen Encoder. Der Informationsfluss im Forward Pass vom Eingabebild zu den einzelnen Ausgaben ist in Grün dargestellt. Die rückfließenden Gradienten im Backward Pass sind hingegen in Rot markiert.

Neben dem gemeinsamen Encoder werden meist für jeden Task einige spezifische Schichten in Form eines Decoders benötigt. Grundsätzlich ist es auch möglich, einen gemeinsamen Decoder für alle Tasks zu verwenden, dies führt allerdings zu einigen Einschränkungen. So ist in diesem Fall die Granularität der Ausgaben für alle Tasks fest vorgegeben. Auch sehen viele Architekturen zur Lösung einzelner Tasks spezielle Schichten oder eine bestimmte Kombination von Schichten vor. Aus diesem Grund sieht der MultiNet-Ansatz für jeden Task einen *Head* genannten Decoder vor, dessen Architektur durch den jeweiligen Task frei bestimmbar ist. Hierbei kann neben den finalen Features auch auf die Ergebnisse der Zwischenschichten aus dem Encoder zugegriffen werden. Dies ist zur Realisierung vieler Architekturen wie dem FCN [158] notwendig.

Für ein gemeinsames Training eines kompletten MultiNet-Modells muss an der Schnittstelle zwischen gemeinsamem Encoder und den einzelnen Heads der Übergang der Zwischenergebnisse im Forward Pass und der Gradienten im Backward Pass sichergestellt sein. Nur so kann ein Training Ende-zu-Ende und gemeinsam für alle Tasks ermöglicht werden. Der Übergang selbst wird, wie in Abbildung 5.4 dargestellt, durch ein sogenanntes Copy Gate realisiert. Hierbei werden im Forward Pass die einzelnen Zwischenergebnisse am Übergang kopiert und an alle angeschlossenen Heads weitergeleitet. Im Backward Pass werden die Gradienten am Übergang wieder zusammengeführt. Dies entspricht im Wesentlichen der Invertierung der Operationen, die bei einer Zusammenführung verschiedener Eingänge ausgeführt werden. Der MultiNet-Ansatz ist durch die Verwendung dieser Konzepte nicht auf CNNs begrenzt und grundsätzlich auch mit anderen Methoden des Maschinellen Lernens verwendbar. Sofern das Konzept der Backpropagation anwendbar ist, können auch der Encoder oder einzelne Heads mit anderen Methoden realisiert werden.

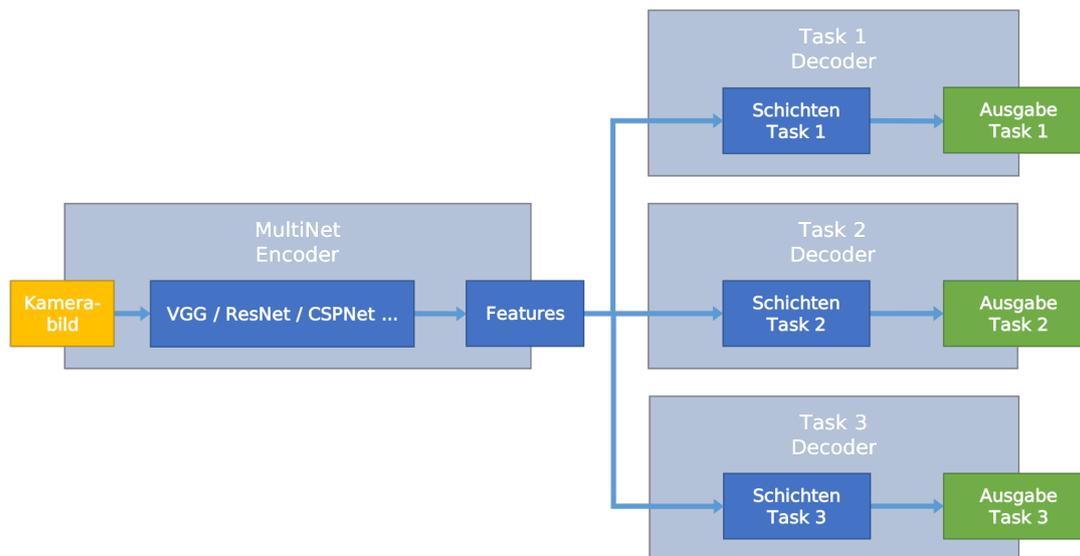


Abbildung 5.5: Eine CNN-Architektur besitzt nach dem MultiNet-Ansatz einen gemeinsamen Encoder zur Extraktion gemeinsamer Features aus dem Eingabebild (gelb). Diese werden pro Task in einem als *Head* bezeichneten taskspezifischen Decoder zur Berechnung der einzelnen Ausgaben (grün) verwendet.

Zusammenfassend realisiert der MultiNet-Ansatz eine MultiTask-Architektur basierend auf dem Encoder-Decoder-Prinzip, wie in Abbildung 5.5 dargestellt. Der gemeinsame Encoder verwendet das Hard Parameter Sharing zum Training gemeinsamer Features für alle Tasks. Für jeden Task existiert ein separater Head mit taskspezifischen Schichten. Die Verbindung der einzelnen Heads mit dem gemeinsamen Encoder erfolgt durch Copy Gates, weshalb ein gemeinsames Ende-zu-Ende-Training mit Backpropagation möglich ist.

5.3.2 Lernstrategien

Ein sehr wichtiger Aspekt beim Training von MultiTask-Netzen ist die Strategie, mit der die Fehlerfunktionen der einzelnen Tasks zu einer gemeinsamen Fehlerfunktion kombiniert werden. Für das Training einer MultiNet-Architektur existiert nicht die eine beste Lernstrategie. Diese ist im Wesentlichen abhängig von den einzelnen Tasks, den für diese Tasks zur Verfügung stehenden Trainingsdaten und deren Qualität. Nachdem bereits in Abschnitt 5.3.1 gezeigt und in Abbildung 5.4 visualisiert wurde, wie eine Ende-zu-Ende-Backpropagation zwischen den einzelnen Heads und dem gemeinsamen Encoder durchgeführt werden kann, muss die Kombination der einzelnen Fehler in einer gemeinsamen Fehlerfunktion betrachtet werden.

Die Fehlerfunktionen der einzelnen Tasks in einer MultiNet-Architektur können dabei grundsätzlich den Fehlerfunktionen von Einzelnetzen zur Ausführung der

Tasks entsprechen. In einer gemeinsamen Fehlerfunktion eines MultiNet werden nun allerdings auch die Fehlerfunktionen in Relation zueinander betrachtet. Durch die gemeinsame Optimierung der Fehler in einem gemeinsamen Trainingsprozess können zusätzliche Effekte auftreten. Eine einfache Möglichkeit zur Kombination der Fehlerfunktionen \mathcal{L}_i für die Tasks $i \in I$ ist gegeben durch:

$$\mathcal{L}_{ges} = \sum_i \mathcal{L}_i \quad (5.2)$$

Unterschiedliche Konvergenzgeschwindigkeiten der einzelnen Tasks können hierbei den Trainingsprozess verlangsamen oder im ungünstigeren Fall die Qualität einzelner Tasks negativ beeinflussen.

Eine Möglichkeit zur Reduktion dieser negativen Effekte ist eine statische Gewichtung der einzelnen Fehlerfunktionen:

$$\mathcal{L}_{ges} = \sum_i w_i \mathcal{L}_i \quad (5.3)$$

Dies wurde bereits von OverFeat [205] und anderen Objektdetektoren erfolgreich eingesetzt. Diese Gewichte müssen für eine neue Kombination verschiedener Tasks allerdings zunächst in einem Prozess mit vielfacher Trainingsdurchführung gefunden werden. Auch bei einem Wechsel der Trainingsdaten kann eine Anpassung dieser statischen Gewichte notwendig sein.

Um eine wiederkehrende und aufwendige Neubestimmung der Gewichtungsfaktoren zu vermeiden, können verschiedene Ansätze zur dynamischen Bestimmung dieser Faktoren verwendet werden. So können die einzelnen Fehlerfunktionen anhand der homoskedastischen Unsicherheit in den Trainingsdaten der zugehörigen Tasks gewichtet werden [135]:

$$\mathcal{L}_{ges} = \sum_i \frac{1}{2\sigma_i^2} \mathcal{L}_i + \log \sigma_i \quad (5.4)$$

σ entspricht hierbei einem Wert für das Rauschen in den Beobachtungen der Trainingsdaten für die einzelnen Tasks. Da dieser Skalar im laufenden Trainingsprozess als Teil der Fehlerfunktion gelernt wird, ist zusätzlich eine Regularisierung vorgesehen.

Neben der Betrachtung der Unsicherheit in den Daten ist es auch möglich, die taskspezifischen Bestandteile der Fehlerfunktion basierend auf Gradienten aus dem Trainingsprozess zu gewichten. Das unter dem Namen GradNorm [59] veröffentlichte Verfahren verwendet folgende Fehlerfunktion:

$$\mathcal{L}_{ges} = \sum_i \nabla_i \mathcal{L}_i \quad (5.5)$$

Die Gewichtung der Fehlerfunktionen für die einzelnen Tasks erfolgt hierbei anhand der Größe der Gradienten ∇ .

Bislang wurde stets angenommen, dass in den Trainingsdaten Annotationen für alle Tasks vorhanden sind. Es muss jedoch auch der Fall behandelt werden, dass einzelne Datensätze nicht für alle Tasks Annotationen besitzen. Die Fehlerfunktion eines einzelnen Tasks T_i , für den im aktuellen Trainingsdatum keine Annotationen vorhanden sind, kann in derartigen Fällen grundsätzlich mit $\mathcal{L}_i = 0$ belegt werden. Allerdings müssen hierbei die Auswirkungen auf den Trainingsprozess im Allgemeinen und den Optimierungsalgorithmus im Speziellen beachtet werden.

Sind in einem kompletten Optimierungsschritt für einen Task keine Annotationen vorhanden und somit auch kein Fehler, kann dies negative Auswirkungen auf das Training dieses Tasks haben. Um dies zu vermeiden, können Anpassungen an dem Optimierungsverfahren vorgenommen werden. So kann ein Optimierungsschritt erst dann durchgeführt werden, wenn für jeden Task ein Fehler > 0 vorhanden ist. Ein solches Optimierungsverfahren ist der asynchrone SGD [142]. Alternativ kann auch für die Auswahl der Datenpunkte in einem Trainingsschritt (mini-batch) die Bedingung formuliert werden, dass zu jedem Task mindestens ein Trainingsdatum mit zugehörigen Annotationen vorhanden ist. Ein optimales Vorgehen ist allerdings auch an dieser Stelle von den Trainingsdaten, der konkreten Architektur sowie den Trainingsvoraussetzungen abhängig.

5.4 Experimente und Evaluation

Die experimentelle Evaluation des in Abschnitt 5.3 vorgestellten MultiNet-Konzeptes wird im Folgenden anhand eines konkreten Anwendungsfalls mit drei Tasks durchgeführt. Hierfür wurde je ein Task aus den Bereichen Segmentierung, Detektion und Bildklassifikation ausgewählt, um die Funktionsfähigkeit des Konzeptes über die verschiedenen Problemklassen in der Bildverarbeitung zu belegen. Aus dem Bereich Bildsegmentierung wurde der Task Straßensegmentierung ausgewählt. Anhand der 2D Detektion von dynamischen Verkehrsobjekten soll die Fähigkeit zur Detektion von Objekten gezeigt werden.

Eine Visualisierung der einzelnen Tasks ist in Abbildung 5.1 zu finden. Die zur Evaluation verwendeten Datensätze werden nachfolgend in Abschnitt 5.4.1 detailliert vorgestellt. In Abschnitt 5.4.2 wird die zur Evaluation verwendete Beispielarchitektur näher beleuchtet, wobei auch die Architekturen zur Lösung der einzelnen Tasks genauer betrachtet werden. Die für diese Evaluation verwendeten Metriken sind in Abschnitt 5.4.3 detailliert erläutert. In Abschnitt 5.4.4 sind die eigentlichen Experimente zum MultiNet-Konzept sowie deren Ergebnisse dargestellt. Die durchgeführten Experimente sind hierbei im Wesentlichen darauf ausgerichtet, die Perzeptionsergebnisse qualitativ und bzgl. des Ressourcenverbrauchs zwischen Einzelmodellen und einem kombinierten MultiNet-Modell zu vergleichen.

5.4.1 Datensätze

Zur Evaluation des MultiNet-Ansatzes werden Datensätze für die Tasks [2D](#) Objektdetektion, semantische Segmentierung und Szenenklassifikation benötigt. Für die beiden erstgenannten Tasks wurden entsprechende Datensätze aus der KITTI Vision Benchmark Suite [\[95\]](#) gewählt. Diese ist eine Sammlung von Benchmarks für verschiedene Bildverarbeitungsaufgaben aus dem Bereich der Umfeldwahrnehmung für Autonomes Fahren. Jeder Benchmark besteht hierbei aus je einem Datensatz mit Trainings- und Testdaten, definierten Evaluationsmetriken und einer Bestenliste zum öffentlichen Vergleich verschiedener Verfahren.

Die KITTI-Datensätze wurden gewählt, da mit der Objektdetektion und der Segmentierung bereits für zwei der drei Tasks Annotationen verfügbar sind und die Daten einfach um Annotationen für die Szenenklassifikation erweitert werden konnten. Da die genannten Tasks von einem gemeinsamen Modell erlernt werden sollen, ist ein möglichst einheitlicher Datensatz mit Annotationen für alle Tasks von Vorteil. So konnten Einflüsse auf die Ergebnisse durch unterschiedliche Bildcharakteristika bereits im Vorfeld ausgeschlossen werden. Auch konnte durch die einheitliche Größe der Eingabebilder der Trainingsprozess vereinfacht und beschleunigt werden.

KITTI Road Detection Datensatz

Dieser Datensatz ist Teil des KITTI Road/Lane Detection Benchmark [\[89\]](#), der neben Daten zum Training und Test von Modellen auch Metriken zum Vergleich verschiedener Verfahren und eine öffentliche Bestenliste beinhaltet. Insgesamt sind 579 Bilder enthalten, welche in einen Trainingsdatensatz mit 289 Bildern und einen Testdatensatz mit 290 Bildern unterteilt wurden. Die Daten entstammen 5 verschiedenen Videosequenzen, welche in der Region Karlsruhe aufgenommen worden sind. Sie liegen meist in der Auflösung 1.242×375 Pixeln vor, mit einer Abweichung von bis zu 16 Pixeln nach unten bei einzelnen Bildern. Sowohl die Bilddaten als auch die Annotationsdaten liegen in Form von einzelnen [PNG](#) Dateien vor.

Zu den Bilddaten des Trainingsdatensatzes sind manuell erstellte Annotationen mit den Klassen $K \in \{ \textit{Straße}, \textit{Hintergrund}, \textit{Ignore} \}$ pro Pixel enthalten. Die Annotationen für die Testdaten sind nicht öffentlich verfügbar, da sie die Bewertungsgrundlage für den KITTI Road Detection Benchmark bilden. Die Klasse *Straße* beinhaltet im Wesentlichen den Bereich der Fahrbahn, der von Fahrzeugen legal genutzt werden darf. Bei getrennten Richtungsfahrbahnen ist dieser Bereich auf die Fahrbahn des Aufnahmefahrzeugs beschränkt. Bei fließenden Übergängen v. a. im Innenstadtbereich sind die Fußgängerbereiche nicht als Teil der Straße annotiert, wie in [Abbildung 5.6b](#) zu sehen ist. Die Daten wurden zu Zeiten mit relativ geringer Verkehrsdichte aufgenommen, weshalb meist ein großer Teil der Straße sichtbar ist.

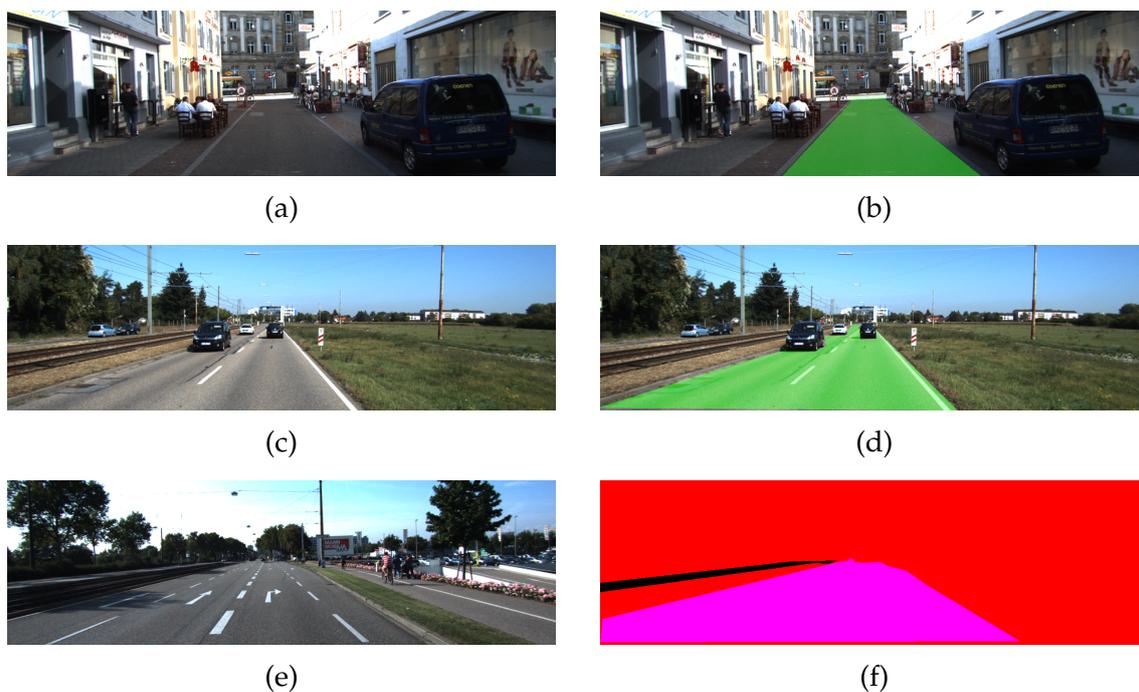


Abbildung 5.6: Beispieldaten aus dem KITTI Road Detection Datensatz. Die einzelnen Zeilen enthalten Bilder der Klassen **UU** (a), (b), **UM** (c), (d) und **UMM** (e), (f). Die linke Spalte enthält die Originaldaten, in der rechten Spalte sind die zugehörigen Annotationen dargestellt. Die Annotationen in (b) und (d) sind hierbei zur besseren Einordnung als Overlay dargestellt, während die Annotation (f) in Form der Originalannotation dargestellt ist. Magenta steht dabei für die Klasse *Straße*, rot für die Klasse *Hintergrund* und die schwarzen Bereiche werden der Klasse *Ignore* zugeordnet. Bildquelle: KITTI Vision Benchmark Suite

Der Datensatz ist dabei unterteilt in folgende Unterkategorien:

Urban with Lane Markings (UM) wird in der Originalbeschreibung mit „marked two-way road“ beschrieben. Meist entspricht dies einer gemeinsamen Fahrbahn mit je einer Fahrspur pro Fahrtrichtung und einer visuellen Abtrennung der Fahrspuren durch Fahrbahnmarkierungen, wie beispielhaft in Abbildung 5.6c zu sehen ist. Allerdings sind auch Daten enthalten, bei denen getrennte Fahrbahnen mit mehreren Fahrspuren pro Fahrtrichtung vorhanden sind. An dieser Stelle scheint die Kategorie etwas unscharf, da Bilder derselben Straße auch in der **UMM**-Kategorie enthalten sind. Im Datensatz sind 95 Trainings- und 96 Testbilder der Kategorie **UM** enthalten.

Urban with Multiple Lane Markings (UMM) wird charakterisiert mit den Worten „marked multi-lane road“. Die Kategorie enthält also Straßen mit Fahrbahn-

markierungen und mehreren Fahrspuren je Fahrtrichtung. Hierbei werden auch bereits Abbiegespuren als eigene Fahrspur gewertet, wodurch es gerade am Beginn oder Ende dieser Spuren zu visuellen Ähnlichkeiten mit der **UM**-Kategorie kommen kann. Es sind 96 Trainings- und 94 Testbilder dieser Kategorie im Datensatz enthalten. Ein Beispiel ist in Abbildung 5.6e dargestellt.

Urban Unmarked (UU) enthält Straßen ohne Fahrbahnmarkierungen. Dies beinhaltet Straßen in Wohngebieten, verkehrsberuhigten Innenstadtbereichen wie in Abbildung 5.6a oder kleine Kreisstraßen. Von dieser Kategorie sind 98 Trainings- und 100 Testbilder im Datensatz enthalten.

KITTI 2D Object Detection Datensatz

Der KITTI 2D Object Detection Datensatz [95] ist Teil des 2D Object Detection Benchmark innerhalb der KITTI Vision Benchmark Suite [96]. Er besteht aus 7.481 annotierten Bildern im Trainings- und 7.518 Bildern im Testdatensatz. Die Annotationen für den Testdatensatz sind hierbei, wie bei allen KITTI-Benchmarks üblich, nicht öffentlich verfügbar und werden zur Berechnung der öffentlichen Bestenliste zurückgehalten. Der Datensatz verfügt über Annotationen der Objekte von 8 verschiedenen Klassen sowie einer *DontCare* Klasse für Regionen, die schwer zu annotieren oder uneindeutig sind. Die Annotationen für den Trainingsdatensatz sind verfügbar in Form einer separaten Textdatei, welche die Koordinaten der Bounding Boxen für die einzelnen Objekte enthält. Insgesamt sind 80.256 Bounding Boxen für Trainings- und Testdatensatz vorhanden. Die Verteilung der Objekte auf die einzelnen Klassen ist für den Trainingsdatensatz in Tabelle 5.1 aufgeschlüsselt.

Klasse	# Annotationen
Car	28.742
Pedestrian	4.487
Van	2.914
Cyclist	1.627
Truck	1.094
Tram	511
Person sitting	222
Misc	973
DontCare	11.295
Σ	51.865

Tabelle 5.1: Verteilung der einzelnen Klassen im Trainingsdatensatz des KITTI 2D Object Detection Benchmark.

Innerhalb des Datensatzes existieren für alle annotierten Objekte die drei verschiedenen Schwierigkeitslevel *easy*, *moderate* und *hard*. Die Einordnung der einzelnen Objekte erfolgt anhand der Kriterien Höhe der Bounding Box, Verdeckungslevel und Überhang über das Bildende. Die Details können der Tabelle 5.2 entnommen werden. Die Kriterien Höhe der Box und Überhang werden dabei automatisch ausgewertet, die Verdeckung wird manuell festgelegt. Für die Evaluation des MultiNet-Ansatzes wurden die Klassen *Car* und *Van* zu der Klasse *Car* zusammengefasst. Das beinhaltet auch die Fahrzeugklasse *Bus*, da diese im KITTI Datensatz in die Klasse *Van* eingeordnet wird.

Schwierigkeit	Min. Höhe BB	Max. Verdeckung	Max. Überhang
easy	40 Pixel	Sichtbar	15%
moderate	25 Pixel	Teilverdeckt	30%
hard	25 Pixel	Schwer erkennbar	50%

Tabelle 5.2: Charakterisierung der einzelnen im KITTI 2D Object Detection Datensatz definierten Schwierigkeitslevel.

KITTI basierter Datensatz zur Szenenklassifikation

Für die Szenenklassifikation wurde im Rahmen einer Abschlussarbeit [30] ein eigener Datensatz basierend auf Bilddaten der KITTI Vision Benchmark Suite [96] und öffentlich verfügbaren Straßeninformationen von [OpenStreetMap \(OSM\)](#) erstellt. Die KITTI Bilddaten wurden dabei anhand der mitgelieferten [GPS](#)-Daten mit den Straßeninformationen von [OSM](#) verbunden [162]. Hierbei ist im Wesentlichen der Straßentyp von Interesse, da hieraus geschlossen werden kann, ob das Fahrzeug sich aktuell auf einer eher ruhigeren Straße oder in einem Bereich einer größeren Straße befindet.

Anhand der verschiedenen Labels¹, welche in [OSM](#) zur Klassifikation von Straßen verfügbar sind, können die Bilder dann in Szenen unterteilt werden. In den verwendeten KITTI-Daten kamen hierbei 10 verschiedene Straßenklassen vor, da in [OSM](#) der Straßentyp sehr feingranular unterschieden wird. Da diese Klassen visuell teilweise sehr ähnlich sind oder sich in manchen Klassen wie z. B. in *residential* und *living_street* kaum unterscheiden und einzelne Klassen nur relativ selten vorkommen, wurden die verschiedenen Klassen zu zwei Oberklassen zusammengefasst. Die Klasse *Highway* ist der amerikanischen Straßenklasse Highway nachempfunden und beinhaltet Straßen von der Autobahn bis zur gut ausgebauten und mit Spurmarkierungen versehenen Landstraße.

In der Klasse *Minor Road* sind dagegen kleinere Straßen vorwiegend Innerorts zusammengefasst. Die genaue Zuordnung der einzelnen [OSM](#)-Klassen zu den

¹<https://wiki.openstreetmap.org/wiki/DE:Key:highway>



Abbildung 5.7: Beispieldaten aus dem auf KITTI basierenden Datensatz zur Szenenklassifikation. Die erste Spalte enthält Bilder der Klasse *Highway* (a), (c), die zweite Spalte der Klasse *Minor Road* (b), (d). Bildquelle: KITTI Vision Benchmark Suite

hier verwendeten Klassen ist in Tabelle 5.3 zu finden. Insgesamt enthält der Datensatz 5.364 Bilder, davon 3.736 Bilder in einem Trainingsdatensatz und 1.628 Bilder in einem Testdatensatz. Die Trennung zwischen Trainings- und Testdatensatz erfolgte dabei auf Basis ganzer Videosequenzen. So konnte garantiert werden, dass Bilder aus einem Video nicht in beiden Datensätzen vorhanden sind. Der Testdatensatz wurde lediglich für die finale Evaluation verwendet, weshalb zur Validierung der Modelle während des Trainingsprozesses ein Validationsdatensatz mit 437 Bildern vom Trainingsdatensatz abgetrennt wurde. Die genaue Verteilung der einzelnen Klassen in beiden Datensätzen ist in Tabelle 5.3 dargestellt.

5.4.2 Beispielarchitektur zur Evaluation

Zur Evaluation der in Abschnitt 5.3 vorgestellten MultiNet-Konzepte wurde eine konkrete Netzarchitektur mit einem gemeinsam genutzten Encoder und drei Heads zur Lösung verschiedener Tasks definiert. Die drei Tasks sind dabei im Einzelnen

- Segmentierung von Straßenbereichen
- Detektion von Fahrzeugen
- Klassifikation der Verkehrsszene

und können jeweils als eigenständiger Decoder betrachtet werden. Die Architekturen der Teilnetze zur Lösung der einzelnen Tasks werden in den nachfolgenden Unterabschnitten detailliert vorgestellt.

Klasse	OSM Klasse	Anzahl Datenpunkte		
		Datensatz	Training	Test
Highway	motorway	166	-	166
	trunk	219	50	169
	primary	667	562	105
	secondary	405	402	3
	Σ	1.457	1.014	443
Minor Road	tertiary	685	646	39
	unclassified	88	88	-
	residential	3.004	1.859	1.145
	living street	84	84	-
	service	33	32	1
	track	13	13	-
	Σ	3.907	2.722	1.185

Tabelle 5.3: Klassendefinition und Verteilung der Daten des Datensatzes zur Szenenklassifikation.

Der Encoder dieser MultiNet-Architektur erhält als Eingabe ein nicht vorverarbeitetes RGB-Bild der festen Größe 1.248×384 Pixel. Die Fixierung der Eingabebildgröße ist notwendig, da in einzelnen Heads Fully-Convolutional-Schichten enthalten sind und der Head zur Klassifikation aufgabenbedingt auf eine Ausgabegröße von 1×1 festgelegt ist. Die resultierende Ausgabegröße für die extrahierten Features wurde auf 32×12 bei einer Anzahl von 512 Feature Maps festgelegt. Dies entspricht einer Reduktion der vertikalen und horizontalen Auflösung um den Faktor 32. Da dieser Faktor seit AlexNet als Standard etabliert wurde, kann als Encoder grundsätzlich jede aktuelle Architektur zur Feature Extraction (vgl. Abschnitt 2.1) verwendet werden. Hierzu wird die Ausgabe der letzten zur Auflösungsreduktion verwendeten Pooling-Schicht als Ausgabe des Encoders verwendet. Für die nachfolgende Evaluation wurden die Architekturen VGG16 [212], ResNet50 und ResNet101 [114] verwendet.

Die Gewichte des Encoders wurden initialisiert mit bereits vorhandenen Gewichten, welche aus auf dem ImageNet-Datensatz vortrainierten Modellen für den jeweils verwendeten Encoder extrahiert wurden. Während des Trainings wurden diese Gewichte allerdings nicht fixiert, sondern durch ein sogenanntes Fine-Tuning an die Charakteristika der Eingabedaten, sowie der zu lösenden Tasks angepasst. Um diese vortrainierten Gewichte bestmöglich verwenden zu können, muss im ersten Schritt des Encoders je Kanal ein vorgegebener Mittelwert von den einzelnen Pixelwerten subtrahiert werden. Dies ist notwendig, da während des ursprünglichen Trainings der übernommenen Gewichte dieser Berechnungsschritt ebenfalls durchgeführt wurde. Die exakte Bedeutung der gelernten Features kann nur beibehalten werden, wenn eine identische Vorverarbeitung der Eingabedaten durchgeführt wird. Die komplette Architektur des verwendeten

MultiNet ist in Abbildung 5.8 dargestellt. Die Details der einzelnen Heads werden nachfolgend erläutert.

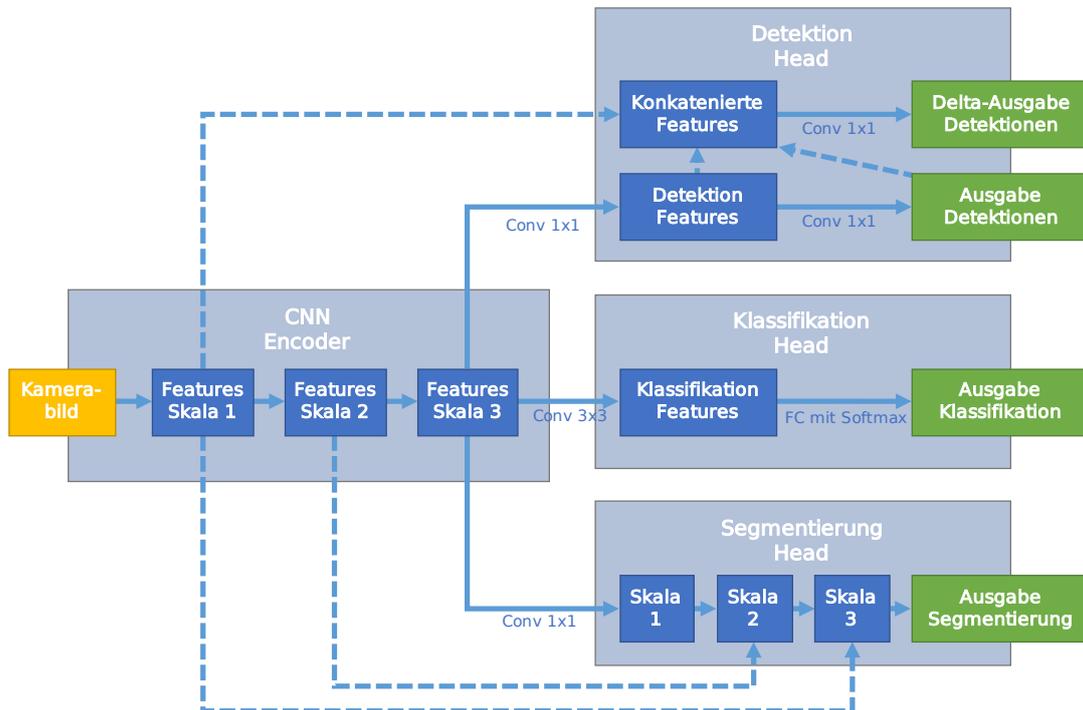


Abbildung 5.8: Im Rahmen dieser Evaluation verwendete Architektur von MultiNet. Der gemeinsame Encoder (links) liefert die finalen Features (blaue Linie, solide), wobei auch Features aus Zwischenschichten über Skip Connections nach außen gegeben werden können (blaue Linie, gestrichelt). Die verschiedenen Tasks werden in den spezifischen Heads der jeweiligen Tasks (rechts) bearbeitet.

Segmentierung von Straßenbereichen

Im Autonomen Fahren ist es von hoher Wichtigkeit, den Straßenbereich und damit auch den befahrbaren Bereich genau zu kennen. Daher ist das Ziel dieses Tasks, den Straßenbereich im Eingabebild genau zu erkennen, also jedem Pixel eine der vorgegebenen Klassen zuzuordnen. Damit ist dieser Task letztendlich ein Lernproblem der pixelweisen Klassifikation. Im vorliegenden Task sind die Klassen *road* und *non-road* definiert. Diese müssen vom trainierten Modell unterschieden werden können.

Die Architektur des für diesen Task verwendeten Teilnetzes ist in Abbildung 5.9 dargestellt und folgt dem [Fully Convolutional Network \(FCN\)](#)-Prinzip [158]. Sie entspricht im Wesentlichen der originalen FCN-Architektur zur Lösung semantischer Segmentierungsaufgaben. Hierbei werden Skip Connections und Transposed Convolutions (vgl. Abschnitt 3.3.2) verwendet. Durch die Skip Connections können die Features aus Zwischenschichten des Encoders mit dem durch die

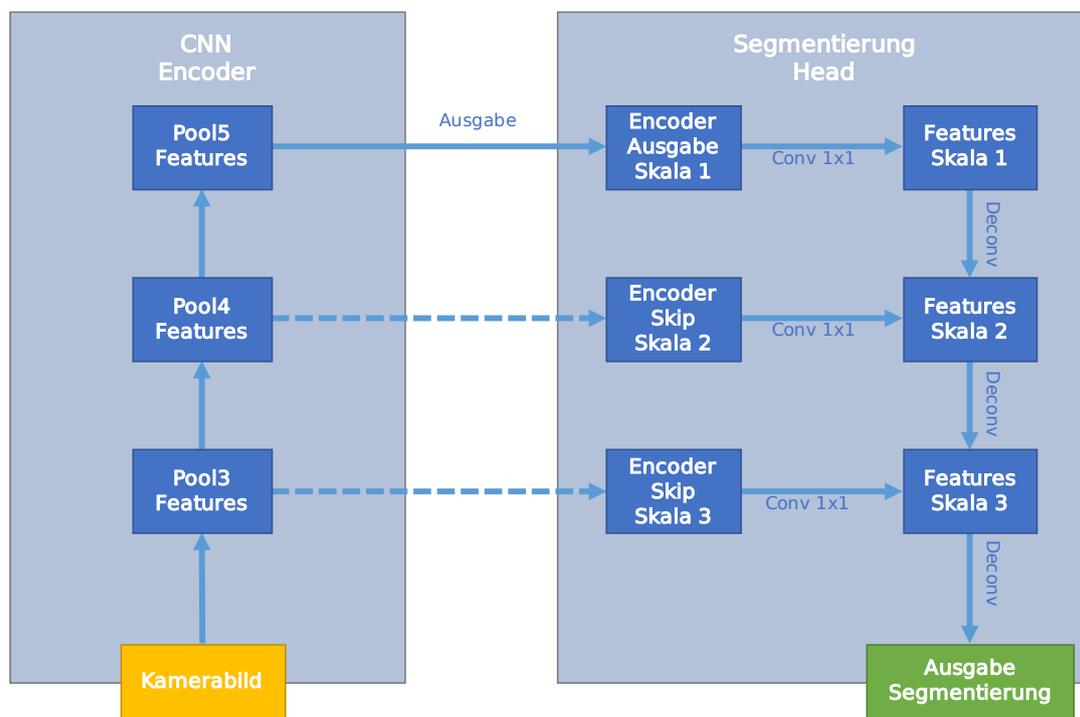


Abbildung 5.9: Architektur des Head zur Segmentierung von Straßenbereichen. Neben der eigentlichen Ausgabe werden die Features von zwei Zwischenauflösungen in verschiedenen Ebenen in den Head eingespeist. Diese werden jeweils durch eine Convolution-Schicht vorverarbeitet, bevor sie dann als Eingabe für die Transposed Convolution-Schicht (Deconv) dienen.

Transposed Convolutions hochskalierten Zwischenergebnissen des Heads zur Segmentierung kombiniert werden. So können auch die Features tieferer Schichten in die Bestimmung des Endergebnisses eingehen und so ein feingranulareres Ergebnis ermöglichen. Details zu den einzelnen Schichten sind in Tabelle 5.4 aufgeführt. Als Fehlerfunktion wird für dieses Teilnetz ein Softmax Cross Entropy Loss verwendet.

Detektion von Fahrzeugen

Aufgabe der Detektionskomponente ist das Finden eines minimal umschließenden Rechtecks, also einer Bounding Box, um jedes sichtbare Fahrzeug, sowie die Zuordnung des detektierten Objektes zu einer Klasse. Die hierfür verwendete Architektur fällt in die Klasse der so genannten One-Stage-Objektdetektoren (vgl. Abschnitt 3.2.1), bestimmt also direkt aus der Eingabe mögliche Objektklassen im Bild, sowie durch Regression assoziierte Bounding Boxen. Die Regionsgröße, auf denen eine einzelne Klassifikation mit Schätzung einer Bounding Box beruht, beträgt hierbei architekturbedingt 32×32 Pixel. Dies wird auch als globaler Stride der Architektur bezeichnet, da bei Betrachtung des Eingabebildes der Abstand

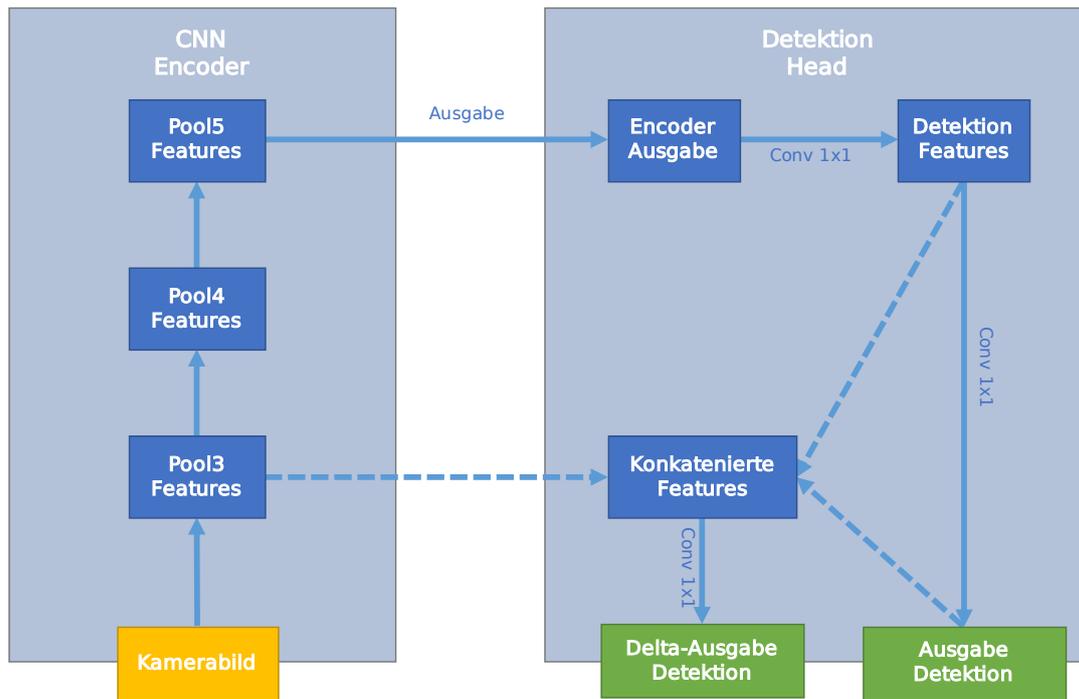


Abbildung 5.10: Architektur des Head zur Detektion von Fahrzeugen. Zunächst werden die Detektionen ermittelt, in einem anschließenden Schritt werden diese mit Hilfe weiterer Features aus einer Zwischenschicht des Encoders präzisiert. Ergebnis hiervon ist ein Delta zur Korrektur der ursprünglichen Detektionen.

zwischen zwei Ausgaben in der horizontalen und vertikalen Dimension 32 Pixel beträgt.

Für jede Region besteht die Ausgabe des Modells aus 6 Werten. Die ersten beiden Werte sind hierbei die Klassifikationsergebnisse für die Klassen *Car* und *Background*. Die restlichen 4 Werte bilden die Regressionswerte für die mit der Region assoziierte Bounding Box. Hierbei kodieren zwei Werte das Zentrum der jeweiligen Box in Relation zur Mitte der Region. Die verbleibenden beiden Werte enthalten die präzisierten Werte für Breite und Höhe der Bounding Box.

Die Architektur für diesen Task folgt ebenfalls dem FCN-Prinzip. Bei der verwendeten Eingabebildgröße von 1.248×384 Pixeln und dem üblichen Reduktionsfaktor entspricht dies einer Eingabegröße der Features von 39×12 Pixeln. Im Decoder werden allerdings, analog zu dem für die Segmentierung vorgestellten Teilnetz, Features aus einer Zwischenschicht des Encoders als Eingabe verwendet. Das Teilnetz selbst besitzt einen zweistufigen Aufbau zur Detektion der Objekte. In einem ersten Schritt werden grobgranulare Bounding Boxen geschätzt, welche in einem zweiten Schritt unter Zuhilfenahme der Zwischenfeatures nochmals präzisiert werden.

So werden die finalen Features des Encoders zunächst durch zwei sequentielle Convolution-Schichten in initiale Detektionen überführt. Die Ausgaben beider



Abbildung 5.11: Beispielausgabe der Detektionskomponente aus dem MultiNet-Modell. Die oberen Bilder (a), (b) zeigen die Ausgabe vor der Nachverarbeitung, in den unteren Bildern (c), (d) ist das finale Detektionsergebnis nach der [Non Maximum Suppression](#) zu sehen. Erstveröffentlichung in: [5]

Schichten sowie die Features aus der Zwischenschicht bilden zusätzlich gemeinsam die Eingabe in eine weitere Convolution-Schicht. In dieser Schicht werden Anpassungen für die ursprünglichen Objekthypothesen geschätzt, um eine genauere Detektion zu ermöglichen. Dieses Verfahren wurde erstmals in MaskRCNN [111] vorgestellt. Im Anschluss wird eine [Non Maximum Suppression \(NMS\)](#) zur Eliminierung doppelter Detektionen durchgeführt. Weitere Informationen zu den einzelnen Schichten sind in Tabelle 5.4 zu finden. Beispielhafte Ergebnisse des Netzes sind in Abbildung 5.11 dargestellt. Als Fehlerfunktion wird eine Kombination aus Mean Cross Entropy für die Klassifikation und L1 Loss für die Regression der Bounding Boxes verwendet. Der Fehler für die Regression wird nur an Stellen berechnet, an denen die Klasse *Car* klassifiziert wurde.

Klassifikation der Verkehrsszene

Ziel dieses Tasks ist es, das Eingabebild als gesamtes Element einer der vorgegebenen Klassen zuzuordnen. So kann die gesamte im Eingabebild sichtbare Verkehrsszene klassifiziert werden. Für dieses Beispiel wurde ein Lernproblem mit den zwei Klassen *Highway* und *Minor Road* erstellt. So soll entschieden werden, ob sich das Fahrzeug auf einer größeren Straße oder eher in einem Wohngebiet befindet. Zum Training wurde für diesen Task der in Abschnitt 5.4.1 vorgestellte Datensatz zur Szenenklassifikation verwendet. Die Eingabegröße der verwendeten Netzarchitektur beträgt 39×12 Pixel, welche durch eine FC-Schicht am Netzende auf eine Ausgabe von zwei Neuronen reduziert wird.

Die Architektur des Netzes ist in Abbildung 5.12 dargestellt. Sie besteht aus einer Convolution-Schicht zur Reduktion der Features, gefolgt von einer finalen

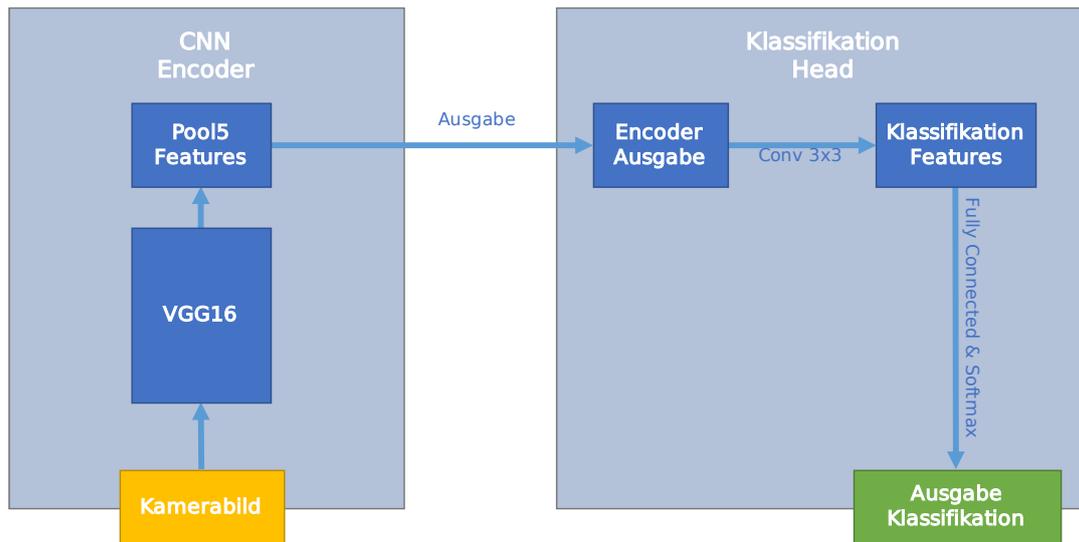


Abbildung 5.12: Architektur des Head zur Klassifikation der Verkehrsszene. Basierend auf der Ausgabe des Encoders wird eine Klassifikation des gesamten Bildes mit anschließendem Softmax durchgeführt.

FC-Schicht zur Klassifikation der Verkehrsszene. Dies entspricht einer recht einfachen Architektur eines Neuronalen Netzes zur Erfüllung einer Klassifikationsaufgabe. Details zu den einzelnen Schichten sind in Tabelle 5.4 zu finden.

5.4.3 Evaluationsmetriken

Zur Bewertung der Ergebnisse werden in dieser Evaluation basierend auf den Charakteristika der einzelnen Tasks verschiedene Metriken verwendet. Sofern verfügbar, wird auf die definierten Metriken der einzelnen KITTI Benchmarks zurückgegriffen. Die im Einzelnen verwendeten Metriken werden nachfolgend für die verschiedenen Tasks genauer erläutert.

Straßensegmentierung Die Qualität der Ergebnisse des MultiNet-Modells auf dem Task der Straßensegmentierung wird im Rahmen dieser Evaluation anhand der Metriken [Average Precision \(AP\)](#) und MaxF1-Score bewertet. Der MaxF1-Score beruht dabei auf dem F1-Score aus Formel (3.14). Für diesen akzeptiert der KITTI Benchmark die Angabe von Klassenwahrscheinlichkeiten ohne harte Entscheidung für eine Klasse. Der beste F1-Score wird an der Stelle aus den für alle Thresholds $p \in [0, 1]$ möglichen F1-Scores ermittelt und stellt den MaxF1-Score dar. Der MaxF1-Score ist dabei die Bewertungsgrundlage für die verschiedenen Algorithmen und ausschlaggebend für die Position in der Bestenliste. Die verwendete [Average Precision](#) entspricht der Formel (4.7).

5 Umfeldererkennung mittels MultiTask-Netzen

Schicht	Modul	Filter	Größe	Eingabedimension	Parameter	FLOPs
conv1_1	FE	64	3×3	1.248 × 384 × 3	1,7 Tsd	1,6 Mrd
conv1_2	FE	64	3×3	1.248 × 384 × 64	36 Tsd	35,3 Mrd
MaxPool	FE	-	2×2	1.248 × 384 × 64	-	
conv2_1	FE	128	3×3	624 × 192 × 64	73 Tsd	17,7 Mrd
conv2_2	FE	128	3×3	624 × 192 × 128	147 Tsd	35,3 Mrd
MaxPool	FE	-	2×2	624 × 192 × 128	-	
conv3_1	FE	256	3×3	312 × 96 × 128	295 Tsd	17,7 Mrd
conv3_2	FE	256	3×3	312 × 96 × 256	590 Tsd	35,3 Mrd
conv3_3	FE	256	3×3	312 × 96 × 256	590 Tsd	35,3 Mrd
MaxPool	FE	-	2×2	312 × 96 × 256	-	
conv4_1	FE	512	3×3	156 × 48 × 256	1,1 Mio	17,7 Mrd
conv4_2	FE	512	3×3	156 × 48 × 512	2,3 Mio	35,3 Mrd
conv4_3	FE	512	3×3	156 × 48 × 512	2,3 Mio	35,3 Mrd
MaxPool	FE	-	2×2	156 × 48 × 512	-	
conv5_1	FE	512	3×3	78 × 24 × 512	2,3 Mio	8,8 Mrd
conv5_2	FE	512	3×3	78 × 24 × 512	2,3 Mio	8,8 Mrd
conv5_3	FE	512	3×3	78 × 24 × 512	2,3 Mio	8,8 Mrd
MaxPool	FE	-	2×2	78 × 24 × 512	-	
conv_s1	Seg	2	1×1	39 × 12 × 512	1 Tsd	1,2 Mio
deconv1	Seg	2	4×4	39 × 12 × 2	< 0,1 Tsd	30 Tsd
conv4_u	Seg	2	1×1	78 × 24 × 2	< 0,1 Tsd	16 Tsd
deconv2	Seg	2	4×4	78 × 24 × 2	< 0,1 Tsd	0,1 Mio
conv3_u	Seg	2	1×1	156 × 48 × 2	< 0,1 Tsd	62 Tsd
deconv3	Seg	2	16×16	156 × 48 × 2	1 Tsd	7,6 Mio
conv_k1	Klas	30	3×3	39 × 12 × 512	138 Tsd	0,1 Mrd
FC	Klas			37 × 10 × 30	22 Tsd	44 Tsd
conv_d1	Det	500	1×1	39 × 12 × 512	256 Tsd	0,2 Mrd
conv_d2	Det	6	1×1	39 × 12 × 500	3 Tsd	2,8 Mio
conv_re	Det	6	1×1	39 × 12 × 1.526	9 Tsd	6,4 Mio
Σ					14,7 Mio	293 Mrd

Tabelle 5.4: Detaillierte Ansicht der VGG-Architektur von MultiNet. FE steht hierbei für die Schichten der Feature Extraction, welche dem Encoder von VGG entsprechen. Seg bezeichnet die Schichten im Head der Straßensegmentierung, Klas die Schichten für die Szenenklassifikation und Det die Schichten für die Objektdetektion. Die Floating Point Operationen (FLOPs) enthalten alle Multiplikations- und Additionsoperationen, die in der entsprechenden Schicht ausgeführt werden.

Objektdetektion Für den KITTI 2D Object Detection Benchmark ist die zentrale Metrik ebenfalls die [Average Precision](#) aus Formel (4.7). Zur Zuordnung von detektierten Objekten zu annotierten Objekten muss analog zu Kapitel 3 wieder die minimale Überlappung festgelegt werden, ab der ein Objekt als korrekt detektiert gilt. Der KITTI Benchmark verlangt für diesen Task, dass die [Intersection over Union](#) für Fahrzeuge gemäß Formel (3.10) mindestens 0,7 beträgt. Es muss also eine Überlappung von 70 % vorliegen. Die einzelnen Objekte des Datensatzes sind, wie bereits erwähnt, in die Schwierigkeitslevel *easy*, *moderate* und *hard* unterteilt. Die [AP](#) wird jeweils separat auf die Objekte der einzelnen Stufen angewandt, wodurch das Ergebnis für jeden Algorithmus eigentlich aus drei Ergebnissen besteht. Der KITTI Benchmark sieht hier als relevante Metrik für die Bestenliste die [AP](#) des Schwierigkeitslevels *moderate* vor.

Szenenklassifikation Als Metriken für die Szenenklassifikation werden hauptsächlich die bereits in Kapitel 3 vorgestellten Precision und Recall verwendet. Da das in diesem Kapitel bewertete Klassifikationsproblem allerdings binär ist, weicht die Formulierung beider Metriken leicht von deren Definition in Kapitel 3 ab. Die Precision ist für diese Evaluation definiert als

$$\text{Precision} = \frac{TP}{TP + FP} \quad (5.6)$$

und der Recall analog als

$$\text{Recall} = \frac{TP}{TP + FN} \quad (5.7)$$

Dies entspricht der üblichen Definition der Metriken, in denen Fehlklassifikationen meist nicht betrachtet werden. Eine genauere Beschreibung von [True Positive \(TP\)](#), [True Negative \(TN\)](#), [False Positive \(FP\)](#) und [False Negative \(FN\)](#) ist in Abschnitt 3.4.2 zu finden.

Zusätzlich wird die Accuracy verwendet, die wie folgt definiert ist:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \quad (5.8)$$

Hierbei werden die korrekten Klassifikationen in ein Verhältnis zu allen durchgeführten Klassifikationen gesetzt.

5.4.4 Evaluation der MultiNet-Architektur

Zur Evaluation des MultiNet-Ansatzes wird die in Abschnitt 5.4.2 vorgestellte Beispiellarchitektur mit den Tasks Segmentierung der Straße, 2D Detektion von Fahrzeugen und Klassifikation der Verkehrsszene verwendet. Hierzu wird neben einem gemeinsamen MultiNet-Modell auch ein SingleTask-Modell für jeden einzelnen Task trainiert. Die Architektur dieser Modelle entspricht weitgehend der im jeweiligen Experiment verwendeten MultiNet-Architektur. Die Anzahl der im

5 Umfelderkennung mittels MultiTask-Netzen

	Segmentierung		Objektdetektion			Szenenklassifikation		
	MaxF1	AP	Average Precision (AP)			Accuracy	Precision	Recall
			moderate	easy	hard			
VGG16	95,99 %	92,31 %	84,68 %	92,06 %	72,08 %	95,75 %	100 %	91,50 %
ResNet50	96,35 %	92,13 %	86,92 %	96,84 %	72,75 %	98,36 %	100 %	96,73 %
ResNet101	95,99 %	91,99 %	89,30 %	96,31 %	75,42 %	98,61 %	99,30 %	97,38 %

Tabelle 5.5: Ergebnisse für die einzelnen Tasks auf den jeweiligen Testdatensätzen mit verschiedenen Encodern bei gemeinsamer Ausführung der Tasks in einem gemeinsamen MultiNet.

	Segmentierung		Objektdetektion			Szenenklassifikation		
	MaxF1	AP	Average Precision (AP)			Accuracy	Precision	Recall
			moderate	easy	hard			
VGG16	95,80 %	92,19 %	84,76 %	92,18 %	68,23 %	97,34 %	98,52 %	87,58 %
ResNet50	95,89 %	92,10 %	86,63 %	95,55 %	74,61 %	98,86 %	100 %	94,11 %
ResNet101	96,29 %	92,34 %	89,79 %	96,13 %	77,65 %	99,84 %	98,70 %	100 %

Tabelle 5.6: Ergebnisse für die einzelnen Tasks auf den jeweiligen Testdatensätzen mit verschiedenen Encodern bei getrennter Ausführung der Tasks in einzelnen SingleTask-Netzen.

Modell gelösten Tasks stellt an der Stelle den einzigen Unterschied dar. Dies ermöglicht eine direkte Gegenüberstellung der qualitativen Ergebnisse zwischen den SingleTask-Modellen und dem MultiNet-Modell. So können die SingleTask-Modelle hinsichtlich der Güte der Ergebnisse als auch des Ressourcenverbrauchs mit dem MultiNet-Modell verglichen werden.

Zum Training der Modelle werden die in Abschnitt 5.4.1 vorgestellten Datensätze verwendet. Zur Steigerung der Diversität der verwendeten Daten wird eine Augmentierung der Eingabedaten durchgeführt. Aus der Gruppe der Methoden zur Texturaugmentierung werden Variationen von Helligkeit und Kontrast angewandt. Zusätzlich werden die horizontale Spiegelung, zufällige Ausschnitte und ein zufälliges Skalieren angewendet. Eine Beschreibung der einzelnen Techniken zur Datenaugmentierung ist in [195] zu finden.

Die weiteren Details des durchgeführten Trainingsprozesses sowie die Belegung der Hyperparameter werden nachfolgend beschrieben: Zur Anpassung der Gewichte wird der Adam Optimizer [140] mit einer initialen Lernrate von 10^{-5} verwendet. Eine sogenannte *weight decay* Regularisierung mit einem Gewicht von $5 \cdot 10^{-4}$ wird als Teil der Fehlerfunktion verwendet. Innerhalb des Netzes wird Dropout mit einer Wahrscheinlichkeit von 0,5 für alle 3×3 Convolution-Schichten der Klassifikation und 1×1 Convolution-Schichten der Detektion verwendet.

Im Rahmen dieser Evaluation werden Experimente mit den State of the Art Encodern VGG16, ResNet50 und ResNet101 durchgeführt. Die qualitativen Ergebnisse sind für die SingleTask-Modelle in Tabelle 5.6 und für die MultiNet-Modelle in Tabelle 5.5 dargestellt. Es wird deutlich, dass sich die Ergebnisse über alle Encoder hinweg zwischen SingleTask und MultiNet lediglich geringfügig unterschei-

den. Für jeden Encoder sind die einzelnen Modelle jeweils in einem Teil der Tasks dem anderen Modell überlegen.

Die größte Abweichung erreicht das MultiNet-Modell auf Basis von VGG16 mit knapp 4 Prozentpunkten gegenüber dem SingleTask-Modell bei der Objektdetektion im Schwierigkeitslevel *hard* sowie beim Recall der Szenenklassifikation. Bei Verwendung der auf ResNet basierten Encoder ergibt sich an der Stelle allerdings ein leicht gegensätzliches Bild, weshalb nicht von einem generellen Effekt ausgegangen werden kann. Über alle betrachteten Tasks hinweg kann grundsätzlich von einer gleichbleibenden Qualität der Ergebnisse ausgegangen werden, wobei einzelne Metriken bei bestimmten Architekturentscheidungen eine leichte Abweichung sowohl nach oben, als auch nach unten zeigen. In Abbildung 5.13 sind beispielhaft einige Ergebnisse des MultiNet-Modells dargestellt

	Segmentierung	Objektdetektion	Klassifikation	MultiNet
VGG16	42,14 ms	37,31 ms	37,83 ms	42,48 ms
ResNet50	39,56 ms	40,09 ms	44,27 ms	60,22 ms
ResNet101	69,91 ms	65,89 ms	71,62 ms	79,70 ms

Tabelle 5.7: Laufzeiten der verschiedenen Netze zunächst als SingleTask-Modell und abschließend gemeinsam in einem MultiNet.

Auch die Laufzeit der einzelnen Modelle wurde einer eingehenden Analyse unterzogen. Die Ergebnisse sind in Tabelle 5.7 zu finden. Für alle Modelle ist zu beobachten, dass die Laufzeit der gemeinsamen Ausführung deutlich unterhalb der Summe der Einzelaufzeiten liegt. Für das auf VGG16 basierende Modell ist diese Laufzeit nur knapp oberhalb der längsten Einzeltasklaufzeit. Allerdings fällt auf, dass der Abstand mit sinkender Anzahl der Rechenoperationen im Encoder deutlich steigt. So scheint der Anteil der in den Heads ausgeführten Operationen überproportional zur Gesamtlaufzeit beizutragen. Dennoch liegt die Laufzeit bei ResNet50, dem Modell mit der größten Differenz zwischen der Ausführung von SingleTask und MultiNet lediglich um 23 % über der Laufzeit des langsamsten Einzeltasks.

Allerdings ist die Laufzeit der einzelnen Modelle immer abhängig von der konkret verwendeten Hardware zur Evaluation und eine Reduktion der Rechenoperationen resultiert nicht immer auch in einer Reduktion der Laufzeit. Aus diesem Grund wurden für das auf VGG16 basierende MultiNet in Tabelle 5.4 ebenfalls die benötigten Rechenoperationen in **FLOPs** berechnet. Bei einer Größe des Eingabebildes von 1.248×384 Pixeln besitzt das VGG16-MultiNet 14,7 Mio. Parameter und benötigt für eine Ausführung 293 Mrd. **FLOPs**. Von den 14,7 Mio. Parametern entfallen mit 14,3 Mio. 97 % auf den gemeinsamen Encoder. Die Objektdetektion enthält 0,27 Mio., die Klassifikation 0,16 Mio. und die Segmentierung 2.000 Parameter. Somit entfällt der weitaus größte Teil des Speicherverbrauchs auf den Encoder des Netzes. Dieser konnte durch Verwendung der MultiNet-Architektur um annähernd zwei Drittel reduziert werden.

5 Umfeldererkennung mittels MultiTask-Netzen

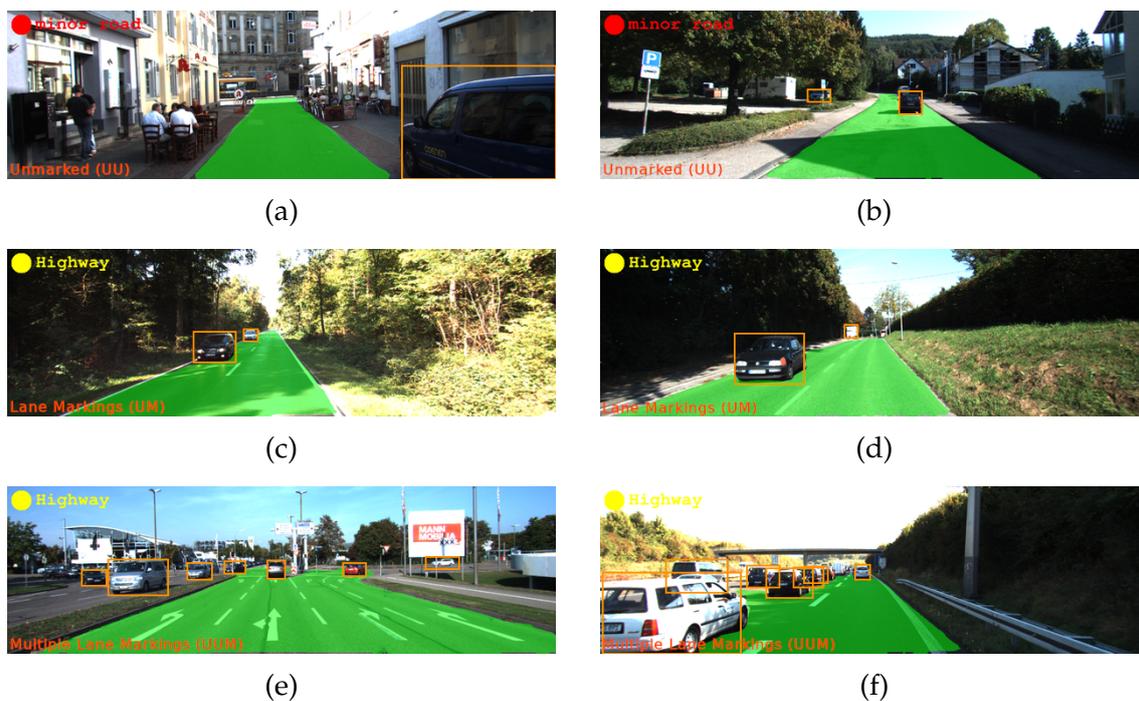


Abbildung 5.13: Beispiele für die Ergebnisse des MultiNet-Modells bei der gleichzeitigen Ausführung aller Tasks. Ähnlich veröffentlicht in: [5]

Viel wichtiger ist an dieser Stelle allerdings eine Betrachtung der Rechenoperationen. Von den 293,2 Mrd. **FLOPs** entfallen mit 292,9 Mrd. über 99 % auf den Encoder. Die Segmentierung verwendet 9 Mio. **FLOPs**, die Klassifikation 100 Mio. **FLOPs** und für die Objektdetektion werden 209 Mio. **FLOPs** benötigt. So kann auch für die Anzahl der benötigten Rechenoperationen eine Reduktion um zwei Drittel der **FLOPs** durch eine Verwendung des MultiNet-Ansatzes beobachtet werden.

Die Tendenz dieser Reduktion lässt sich auch auf die anderen Encoder übertragen. Aus Tabelle 2.1 kann zumindest die grobe Tendenz übernommen werden, dass ein ResNet101 ca. die Hälfte und ein ResNet50 ca. ein Viertel der Rechenoperationen eines VGG16 besitzt. Die Anzahl der Parameter für die einzelnen Heads bleibt bei Verwendung anderer Encoder identisch. So beträgt auch bei einer angenommenen Reduktion der Rechenoperationen um 75 % der Anteil des Encoders an allen Operationen noch immer über 99 %.

5.5 Diskussion

In diesem Kapitel wurde mit dem MultiNet-Ansatz eine Grundlage für MultiTask-Architekturen vorgestellt. MultiNet ist in der Lage, **CNNs** für verschiedene Perzeptionsaufgaben auf demselben Eingabebild in einem Modell mit gemeinsa-

mem Encoder zeitgleich auszuführen. So ist es möglich, Ergebnisse für sämtliche Tasks bei nur einer Ausführung eines Encoder-Netzes zu erhalten. Die Grundannahme war an dieser Stelle, dass durch den Wegfall der zusätzlichen Encoder der Ressourcenbedarf für Speicher und vor allem Recheneinheiten deutlich reduziert werden kann.

Um dies zu bestätigen, wurde eine Evaluation mit einem MultiNet zur Lösung je eines Klassifikations-, Segmentierungs- und Detektionsproblems durchgeführt. Diese Probleme wurden bewusst gewählt, um die wesentlichen im Bereich der Bildverarbeitung vorkommenden Problemarten abzudecken. Hierfür wurde zunächst die benötigte Anzahl an Rechenoperationen für die SingleTask-Architekturen sowie die MultiNet-Architektur bestimmt. Aus dieser Untersuchung ging hervor, dass der Encoder in den betrachteten Architekturen für mehr als 99 % der Rechenoperationen verantwortlich ist. Gegeben das im Rahmen dieser Evaluation verwendete MultiNet zur gleichzeitigen Ausführung von drei Tasks können also durch den architekturbedingten Wegfall von zwei Encodern ca. 65 % der Rechenoperationen eingespart werden.

An dieser Stelle konnte auch beobachtet werden, dass dies einen deutlichen Einfluss auf die Laufzeit der Modelle hat. Durch den Wegfall eines Großteils der Rechenoperationen reduziert sich die Laufzeit für das größte Modell nahezu auf die Laufzeit des langsamsten Tasks. Diese Laufzeit des langsamsten Tasks stellt für eine gemeinsame Ausführung das Optimum dar, da alle Bestandteile dieses Tasks nach wie vor ausgeführt werden. Für die kleineren Modelle konnten in maximal 123 % der Laufzeit des langsamsten Einzeltasks alle Tasks im MultiNet ausgeführt werden. Auffällig scheint an dieser Stelle allerdings auch die Tatsache, dass die Unterschiede in den Laufzeiten zwischen den einzelnen Encodern nicht unbedingt der Abschätzung anhand der Rechenoperationen entsprechen.

Der quantitative Vergleich der Ergebnisse zwischen den SingleTask-Netzen und dem MultiNet ergab grundsätzlich vergleichbare Resultate. Der Unterschied der Ergebnisse für die einzelnen Metriken lag in den meisten Fällen bei einer Abweichung von unter einem Prozentpunkt. Lediglich bei 5 Vergleichen war das MultiNet mehr als einen Prozentpunkt überlegen, in 4 Vergleichen waren die Einzeltasks um mehr als einen Prozentpunkt überlegen. Inwiefern hierbei MultiTask-Effekte zur besseren Generalisierung einzelner Tasks eine Rolle spielen, stellt an dieser Stelle eine interessante Fragestellung für zukünftige Forschungsarbeiten dar. Insbesondere ist auch die Frage von Interesse, ob diese Effekte durch zusätzliche Daten zum Training vermindert werden können.

6 Gewichtung einzelner Datenpunkte im Trainingsprozess



Abbildung 6.1: Ziel der in diesem Kapitel vorgestellten Ansätze ist die Gewichtung der einzelnen Datenpunkte im Training.

Der Trainingsprozess eines jeden tiefen Neuronalen Netzes ist eine kritische Phase während der Entwicklung eines Modells zur Bildverarbeitung und ist mitentscheidend für dessen spätere Qualität. Die Auswahl einer Fehlerfunktion und die Gewichtung der einzelnen Datenpunkte im Trainingsdatensatz spielen an dieser Stelle eine entscheidende Rolle. Die Gewichtung von Datenpunkten dient hierbei im Wesentlichen zwei Zielen. Zum einen sollen herausfordernde Trainingsdatenpunkte höher gewichtet werden, da der Lernprozess im Training für diese zeitaufwendiger ist. Zum anderen sollen häufig auftretende oder ähnliche Trainingsbeispiele zur Hervorhebung seltener Trainingsdaten geringer gewichtet werden [50].

Dies lässt sich gut am Beispiel der Detektion von **2D** Objekten im Bildraum nachvollziehen. Die wenigen Bereiche in den Trainingsbildern, in denen Objekte vorhanden sind, müssen im Trainingsprozess deutlich stärker gewichtet werden als Bereiche, in denen sich kein Objekt befindet. Die Details dieser Problemstellung werden in Abschnitt 6.1 nochmals erläutert. Je nach verwendetem Netzwerkdesign sind in den Architekturen bereits implizite Mechanismen inhärent vorhanden. Andernfalls müssen im Trainingsprozess explizit Maßnahmen getroffen werden, um einen ausbalancierten Trainingsprozess zu ermöglichen. Vorhandene Mechanismen und bisher verwendete Maßnahmen werden in Abschnitt 6.2 dargestellt.

Dies beinhaltet auch den *Focal Loss* [153], bei dem mittels zusätzlichem Wissen aus den Trainingsdatensätzen ein Fokus auf bestimmte Datenpunkte gelegt wird. Allerdings muss hier aus diesem zusätzlichen Wissen für jeden Trainingsdatensatz zunächst ein neuer Parameter bestimmt werden. Somit wird der Optimierung der Hyperparameter im Trainingsprozess ein weiterer Parameter hinzugefügt, mit deutlichen Auswirkungen auf die zu erwartende Trainingsdauer. Auch ist dieser zusätzliche Parameter über die Dauer des fortschreitenden Trainingsprozesses statisch und kann so nicht auf Trainingsfortschritte reagieren.

In Abschnitt 6.3 wird zur Lösung dieser Nachteile deshalb der *Automated Focal Loss* eingeführt. Hierbei erfolgt eine direkte Gewichtung der Trainingsdatenpunkte anhand des jeweiligen Trainingsfortschrittes. Die Gewichtung ist daher dynamisch am Trainingsfortschritt orientiert. In Abschnitt 6.4 wird in mehreren Experimenten der Automated Focal Loss im Vergleich zum bisherigen Standardverfahren, dem Focal Loss, untersucht. Hierbei wird unter anderem der sehr komplexe Task der Detektion dynamischer Verkehrsobjekte aus Kapitel 4 zum Vergleich der Fehlerfunktionen herangezogen. Eine Diskussion der Ergebnisse findet in Abschnitt 6.5 statt. Teile dieses Kapitels wurden bereits in der Publikation [7] vorgestellt. Ebenfalls flossen Ergebnisse der Abschlussarbeit [19] in dieses Kapitel ein.

6.1 Problembeschreibung

Bei der Nutzung von überwachten Maschinellen Lernverfahren spielen die zum Training verwendeten, annotierten Daten eine kritische Rolle. Neben den gewählten Verfahren und Modellarchitekturen sind diese Trainingsdaten hauptverantwortlich für das resultierende Modell. Anhand dieser Daten muss die eigentliche, aber unbekannte Zielfunktion approximiert werden. Dabei gilt generell, dass ein trainiertes Modell besser auf unbekannte Daten generalisierbar ist, je mehr Daten zum Training verwendet wurden [104].

Der Einfluss einzelner Datenpunkte auf die Qualität des trainierten Modells kann hierbei allerdings von unterschiedlicher Stärke sein. Wird dies im Trainingsprozess nicht entsprechend abgebildet, können sich verschiedene Implikationen für den Verlauf des Trainingsprozesses und die Charakteristik der resultierenden Modelle ergeben. So ist es möglich, dass sich die Dauer eines Trainings deutlich erhöht oder die Qualität des Modells in manchen Bereichen drastisch verschlechtert. Maßgeblich für den Einfluss ist die Verortung und Verteilung der Datenpunkte im Eingaberaum, also dem Raum aller möglichen Eingabedaten. Dies soll nachfolgend am Beispiel der Klassifikationsprobleme näher beschrieben werden. Die Klassifikation hat an dieser Stelle die vereinfachende Eigenschaft, dass der Ergebnisraum diskret ist. Daher existieren harte Entscheidungsgrenzen für die Zielklasse im Eingaberaum. Auch sind so die aus der Verteilung der Datenpunkte im Ergebnisraum entstehenden Herausforderungen einfacher darstellbar.

Bei einem Klassifikationsproblem ist einem Trainingsdatenpunkt im Eingaberaum eine Klasse aus dem Ergebnisraum zugeordnet. Alle einer Klasse zugeordneten Datenpunkte repräsentieren diese Klasse im Trainingsdatensatz und somit im Eingaberaum. Zwischen den Gruppen der Datenpunkte jeder Klasse liegen die Entscheidungsgrenzen für die jeweils angrenzenden Klassen. Diese sind für das jeweilige Klassifikationsproblem fest definiert, können anhand der verfügbaren Trainingsdatenpunkte allerdings lediglich approximiert werden, wie in Abbildung 6.2 dargestellt.

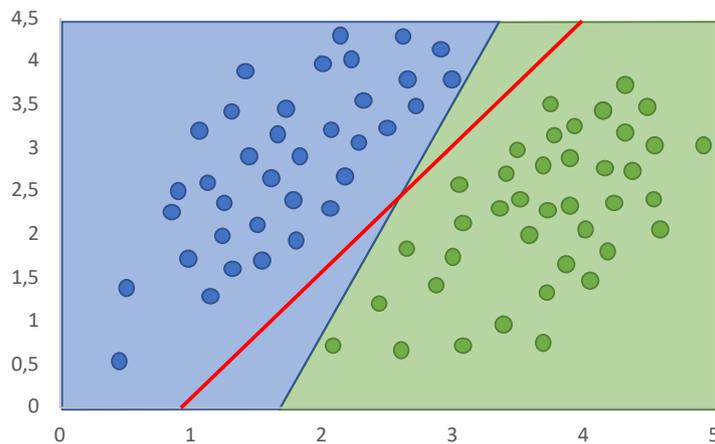


Abbildung 6.2: Visualisierung der Entscheidungsgrenze bei einem einfachen Klassifikationsproblem mit zwei Features. Blaue und grüne Punkte stellen die vorhandenen Datenpunkte dar. Die blaue Trennlinie entspricht der tatsächlichen Trennlinie zwischen den Klassen. Die rote Trennlinie repräsentiert das Ergebnis eines Klassifikators, der nach dem Prinzip des maximalen Abstandes zwischen den Klassen entscheidet.

Den einzelnen Trainingsdatenpunkten wird in verschiedenen Arbeiten eine sogenannte *Schwierigkeit* zugewiesen [45]. Dahinter steht die grundsätzliche Aussage, dass manche Datenpunkte für das Training eines Klassifikators relevanter sind als andere. Hierbei ist grundsätzlich zwischen zwei Bedeutungen des Begriffes Schwierigkeit zu unterscheiden. Einerseits wird mit der Schwierigkeit eine generelle Einordnung der Lernbarkeit eines konkreten Datenpunktes getroffen. Ein Beispiel hierfür kann die räumliche Nähe des Datenpunktes zur nächsten Entscheidungsgrenze sein.

Andererseits kann die Schwierigkeit auch abhängig vom aktuellen Trainingsfortschritt oder der verwendeten Methode sein. So kann beispielsweise ein einfacher Klassifikator auf das Problem angewandt und falsch klassifizierte Datenpunkte können als schwierig eingeordnet werden. [45] Bei Algorithmen aus der Klasse der Boosting-Methoden beispielsweise kann hier der iterative Trainingsprozess genutzt werden. So wird bei DataBoost-IM [106] während des Trainingsprozesses der aktuelle Trainingsstand des Klassifikators verwendet, um die Schwierigkeit der Datenpunkte in jedem Trainingsschritt zu bestimmen. In diesen Fällen ist die

Einordnung der Daten statt von der realen Entscheidungsgrenze von der aktuell gelernten Entscheidungsgrenze abhängig.

Bei einem als schwierig eingeordneten Datenpunkt besteht allerdings auch immer die Möglichkeit einer fehlerhaften Annotation, weshalb es an dieser Stelle auch sinnvoll sein kann, die Bedeutung dieses Datenpunktes gar nicht oder nur gering zu gewichten. Insbesondere gegen Ende des Trainingsprozesses, wenn bereits ein Großteil der Datenpunkte korrekt gelernt wurde, können derartige Fehler große Auswirkungen haben. So kann hierdurch Overfitting entstehen und damit die Generalisierungsfähigkeit des trainierten Modells reduziert werden.

Neben der Betrachtung der Schwierigkeit einzelner Datenpunkte ist die Verteilung der Datenpunkte innerhalb des Eingaberaumes ein wichtiger Faktor. Meist sinkt bei annotierten Daten der Einfluss einzelner Datenpunkte mit der Anzahl der vorhandenen Datenpunkte für die jeweilige Klasse. Ist eine Klasse in den Trainingsdaten überrepräsentiert, spricht man auch vom *Class Imbalance Problem* [128]. Der Terminologie folgend ist eigentlich jeder Datensatz mit ungleicher Verteilung der Datenpunkte über die einzelnen Klassen betroffen. Allerdings wird meist erst ein deutliches Missverhältnis von z. B. 1 : 100 als wirkliche Class Imbalance bezeichnet [110].

Das Class Imbalance Problem kann durch das Optimierungsverfahren begünstigt werden. Dieses behandelt alle Datenpunkte gleich, sofern nicht separat durch die Fehlerfunktion eine Gewichtung erfolgt. Während der ursprüngliche Gradientenabstieg für einen Trainingsschritt noch alle Datenpunkte verwendet, kommen bei stochastischen Gradientenabstiegen nur noch einzelne Datenpunkte oder im Falle des Gradientenabstiegs basierend auf Mini-Batches eine kleine Gruppe von Datenpunkten für die Berechnung der Gradienten in einem Trainingsschritt zum Einsatz. In diesem Fall bekommt nun auch die Reihenfolge der im Training verwendeten Datenpunkte eine Relevanz. So steigt bei einer Ungleichverteilung der Daten über die Klassen die Wahrscheinlichkeit dafür, dass in einem Update des Optimierungsverfahrens einzelne Klassen überhaupt nicht repräsentiert sind.

Aus einer Class Imbalance ergeben sich zwei verschiedene Probleme. Der Backpropagation-Algorithmus konvergiert aufgrund der Class Imbalance wesentlich langsamer [40]. Somit wird der gesamte Trainingsprozess ebenfalls deutlich verlangsamt und somit verlängert. Auch kann sich eine Ungleichverteilung der Daten auf die Qualität des trainierten Modells auswirken [128][110]. So kann durch die Fehlerfunktion das Verhalten erlernt werden, dass nur noch die stark repräsentierten Klassen in den Hypothesen beachtet werden.

Dies lässt sich am Beispiel der Segmentierung von Verkehrsampeln visualisieren, welche Bestandteil der in Kapitel 3 vorgestellten Ampelerkennung ist. Ein Eingabebild der Größe 1.280×960 Pixel = 1.228.800 Pixel enthält bei einer pixelgenauen Segmentierung 1.228.800 Datenpunkte. Eine Ampel der Größe 10×30 Pixel bedeckt mit 300 Pixeln davon lediglich einen Bruchteil. Enthält ein Eingabebild eine Ampel dieser Größe, entspricht dies einem Verhältnis der Klassen *Ampel* zu

Hintergrund von 1 : 4.096. Bei einer einfachen Mittelwertbildung der Fehlerfunktionen für die einzelnen Pixel würde mit einer Klassifikation des gesamten Bildes als *Hintergrund* der Klassifikationsfehler somit unter 0,03 % liegen. Ein Klassifikator, welcher grundsätzlich keine Ampeln erkennt, würde von der Fehlerfunktion also nahezu keinen Fehler zugewiesen bekommen.

Bei dem in dieser Arbeit behandelten Task der Objektdetektion tritt das Problem der Ungleichverteilung je nach verwendeter Architektur unterschiedlich stark auf. Architekturbedingt sind die sogenannten One-Stage-Detektoren deutlich stärker betroffen als Two-Stage-Detektoren. Nach Lin et al. [153] ist diese Ungleichverteilung die Hauptursache für die im Verhältnis schlechte Qualität der One-Stage-Detektoren in vergangenen Jahren. So müssen diese Detektoren eine breite Repräsentation von Hintergrund lernen, während bei Two-Stage-Detektoren bereits eindeutige Datenpunkte der Klasse *Hintergrund* in einem ersten Schritt vor dem eigentlichen Klassifikator ausgefiltert werden.

6.2 Verwandte Arbeiten

Die Frage nach der Betrachtung einzelner Datenpunkte im Trainingsprozess eines Maschinellen Lernverfahrens geht über die reine Betrachtung der Gewichtung vorhandener Datenpunkte hinaus. Die Problemstellung und -lösung beginnt hierbei bereits bei der Auswahl der Trainingsdatenpunkte. Auswahl und Gewichtung sind an dieser Stelle im Wesentlichen die Bestandteile der Herangehensweise zur Problemlösung. Grundsätzlich existieren hierbei zwei unterschiedliche Herangehensweisen:

Durch Hinzufügen und Entfernen einzelner Datenpunkte kann eine Veränderung der Charakteristika des Trainingsdatensatzes durchgeführt werden. Hierbei können nach bestimmten Kriterien ähnliche Datenpunkte entfernt oder zusätzliche Daten generiert werden. Auch ist eine Auswahl verschiedener Teilmengen des Trainingsdatensatzes in unterschiedlichen Phasen des Trainingsprozesses möglich. Alternativ können durch die verwendete Fehlerfunktion die vorhandenen Datenpunkte nach verschiedenen Charakteristika unterschiedlich gewichtet werden.

Auswahl der Daten im Datensatz

Bei der bewussten Wahl der Daten im aktuellen Trainingsdatensatz werden im Wesentlichen zwei verschiedene Strategien verwendet, um die beschriebenen Probleme abzumildern oder bestenfalls aus dem Datensatz zu entfernen. So kann aus einem vorhandenen Trainingsdatensatz eine Teilmenge der Datenpunkte für das komplette Training oder den aktuellen Trainingsschritt ausgewählt werden. Alternativ dazu kann der Trainingsdatensatz bewusst um weitere Datenpunkte erweitert oder um vorhandene Datenpunkte reduziert werden.

Die Strategie des Hinzufügens weiterer Daten wird beispielsweise im Themenfeld des aktiven Lernens [206][143] verfolgt. Hierbei werden aus einer großen Menge nicht annotierter, zusätzlich zu den annotierten Trainingsdaten vorhandener Daten, relevante Daten während des Trainings ausgewählt, welche nachfolgend händisch annotiert werden. Das aktive Lernen kann somit als eine Nachfrage an einen Experten betrachtet werden. Die Auswahl der hinzuzufügenden Daten kann anhand verschiedener Kriterien erfolgen, ist zunächst allerdings unabhängig von einer konkreten Klassenzugehörigkeit.

Im Gegensatz hierzu werden bei der Klasse der sogenannten bootstrapping-Verfahren [198][219][220] lediglich Negativbeispiele, also Bilddaten ohne das zu erkennende Objekt, hinzugefügt. Hierbei wird im Trainingsprozess der zu trainierende Algorithmus auf Bildern ausgeführt, die keine der gesuchten Objekte enthalten. Die Regionen von Fehldetektionen werden in die Negativklasse der Trainingsdaten übernommen und beim weiteren Training der Klassifikationsalgorithmen in der Detektionskette verwendet.

Für Detektionsverfahren, welche auf die Sliding-Window-Methode zur Erzeugung der Kandidatenregionen für eine Klassifikation zurückgreifen, kann eine ähnliche Methode mit leicht anderen Vorbedingungen durchgeführt werden [82]. So existiert durch die Anwendung der Sliding-Window-Methode bereits eine große Menge an Negativbeispielen für die Nutzung in einem Klassifikationsverfahren. An dieser Stelle werden dann nur die *Hard Negatives*, also die Negativbeispiele mit einer hohen Klassifikationswahrscheinlichkeit für eine vorhandene Klasse, zum Training verwendet.

Mit dem *Hard Negative Mining* [157] wurde dieser Ansatz auch für CNN-basierte Objektdetektoren adaptiert. Insbesondere die Architektur von One-Stage-Detektoren mit der impliziten Verwendung eines Sliding Windows besitzt an dieser Stelle eine große Ähnlichkeit zu den klassischen Sliding-Window-Verfahren. Die Negativklasse wird an dieser Stelle meist als *Hintergrund* bezeichnet und ist im Normalfall auch die dominante Klasse. Mit diesem Hard Negative Mining wird ebenfalls nur ein geringer Anteil der Beispiele dieser Klasse zum Training des Klassifikators verwendet. So wird meist ein maximales Verhältnis zwischen Trainingsbeispielen der Objektklassen und des Hintergrundes angegeben.

Mit dem *Online Hard Example Mining (OHEM)* [210] wurde auch ein entsprechend angepasstes Verfahren für Two-Stage-Objektdetektoren eingeführt. Hierbei werden während der Ausführung eines Trainingsschrittes die in der ersten Stufe gefundenen Regionskandidaten zunächst anhand ihres Klassifikationsfehlers geordnet. Für den Trainingsschritt im Klassifikator werden lediglich die Regionen mit hohem Fehler verwendet. Dies kann auch als binäre Gewichtung der einzelnen Trainingsdaten und somit als ein Vorgänger des im nächsten Abschnitt beschriebenen Focal Loss interpretiert werden.

Datengewichtung durch die Fehlerfunktion

Die Gewichtung einzelner Datenpunkte in Fehlerfunktionen wird bei Algorithmen des Maschinellen Lernens bereits seit vielen Jahren genutzt. Bereits Mitte des letzten Jahrhunderts nutzte der *Huber Loss* [121] einen Mechanismus, um den Fehler bei Daten mit großer Abweichung der Hypothese von der Annotation weniger stark in der Fehlerfunktion zu gewichten. Dies geschah mit der Absicht, den Einfluss von sogenannten Ausreißern in den Trainingsdaten zu begrenzen. Auch *AdaBoost* [88] berechnete den Fehler einer Hypothese bereits anhand von datenspezifischen Gewichten, welche in jeder Iteration angepasst wurden.

Eine Gewichtung von Trainingsdaten in Neuronalen Netzen wird in *Curriculum Learning* [45] durchgeführt. Dieser Ansatz kann als eine Sammlung mehrerer Sätze von Trainingsgewichten gesehen werden, die nacheinander im Training Anwendung finden. Ziel hierbei ist es, zunächst möglichst einfache Trainingsdatenpunkte zu favorisieren und erst im Laufe des Trainings den Fokus auch auf schwierige Datenpunkte zu richten. Ein ähnlicher Ansatz ist das *Self-Paced Learning* [145], bei dem allerdings eine binäre Betrachtung der Datenpunkte durchgeführt wird. Diese werden also erst nach und nach dem Trainingsprozess hinzugefügt. Durch die Binarisierung der Gewichte kann dieser Ansatz auch der Kategorie *Auswahl der Daten* zugeordnet werden.

Im Kontext von **CNNs** wurde mit der Verfügbarkeit größerer Mengen an Trainingsdaten auch bald der Bedarf deutlich, eine Ungleichverteilung der Datenpunkte zwischen den einzelnen Klassen ausgleichen zu können. So wurde das sogenannte α -balancing in vielen Ansätzen eingesetzt, dessen genauer Ursprung allerdings unbekannt ist. Hierbei findet eine Gewichtung der Datenpunkte einer Klasse anhand der inversen Auftrittshäufigkeit im gesamten Trainingsdatensatz statt. Aus diesem Grund wird teilweise auch der Name *inverse class frequency weighting* verwendet. Die Gewichtung der einzelnen Trainingsbeispiele ist in diesem Fall statisch bereits vor Beginn des Trainings festgelegt.

Beim Training eines **CNN** wird ein Update-Schritt des Optimierungsalgorithmus meist basierend auf einer kleinen Teilmenge der Trainingsdaten, einem sogenannten Mini-Batch, durchgeführt. Deshalb wird häufig auch eine Gewichtung anhand der inversen Klassenhäufigkeit in dem aktuellen Mini-Batch durchgeführt. So können Effekte wie ein sprunghaftes Verändern des Fehlers durch ungleich verteilte Datenpunkte in einzelnen Mini-Batches vermindert werden. Hierfür muss allerdings die Klassenhäufigkeit für jeden Mini-Batch neu berechnet werden. Da sich im Idealfall die Aufteilung der Trainingsdaten in die einzelnen Mini-Batches mit jeder Trainingsepoche verändert, muss diese Berechnung online während des Trainings erfolgen.

Das Generalized Max-Pooling [50] soll mit einem klassenagnostischen Ansatz für **CNNs** zur Bildsegmentierung die Inter-Class- und Intra-Class-Imbalance, also die Unausgeglichenheit der Datenpunkte zwischen den Klassen und innerhalb einer Klasse, kompensieren. Hierzu wird die Annahme getroffen, dass weniger

häufig vertretene Daten gleichbedeutend mit einem hohen Fehler für den entsprechenden Pixel sind. Daher wird auf Pixelbasis der Fehler von Pixeln mit bereits hohem Fehler nochmals verstärkt. Dies erscheint vor allem für Segmentierungsprobleme sinnvoll, da hier bereits in jedem Bild eine große Menge an Klassifikationsentscheidungen getroffen wird. So wird die Wahrscheinlichkeit reduziert, dass manche Klassen in einem Mini-Batch mit mehreren Bildern überhaupt nicht vorkommen und so ein sprunghaftes Verändern des Fehlers auftritt. Dieses sprunghafte Verhalten könnte durch das vorgeschlagene Generalized Max-Pooling noch verstärkt werden.

Eine andere Methode zur Gewichtung des Fehlers einzelner Pixel bei der semantischen Segmentierung wird in U-Net [197] vorgestellt. So wird festgelegt, dass bei einer semantischen Segmentierung der betrachteten medizinischen Daten eine korrekte Segmentierung vor allen Dingen am Übergang zwischen den einzelnen Klassen eine hohe Wichtigkeit besitzt. Aus diesem Grund wird ebenfalls klassenagnostisch der Fehler in diesen Bereichen bei der Berechnung des Fehlers über das gesamte Bild deutlich stärker gewichtet, als in den übrigen Bereichen des Bildes.

Im Gegensatz zu diesen auf Segmentierungsansätze ausgelegten Ansätzen sind das Ziel des Focal Loss [153] One-Stage-Ansätze zur Objektdetektion. Der Focal Loss stellt dabei eine Anpassung des herkömmlichen Cross Entropy Loss dar, bei dem ein zusätzlicher Vorfaktor anhand der vorhergesagten Wahrscheinlichkeit der korrekten Klasse für das erkannte Objekt bestimmt wird. So sollen korrekt erkannte Trainingsdaten, welche bereits mit einer hohen Wahrscheinlichkeit bestimmt wurden, bei der Berechnung des Fehlers nur noch wenig berücksichtigt werden. Die eigentliche Fehlerfunktion ist gegeben durch:

$$\mathcal{L}_{FL} = -(1 - p_t)^\gamma \cdot \log(p_t) \quad (6.1)$$

Der namensgebende Fokus hat damit zum Ziel, den Fehler auf Trainingsdaten zu konzentrieren, bei denen die korrekte Klasse p_t eine sehr niedrige Wahrscheinlichkeit besitzt. Mit dem Parameter γ wird die Stärke dieses Fokus bestimmt. Hierbei ist anzunehmen, dass zumindest für jeden Anwendungsfall das zu verwendende γ durch Optimierung als Hyperparameter neu bestimmt werden muss. Der Verlauf des Fehlers in Relation zur Wahrscheinlichkeit der korrekten Klasse ist für verschiedene γ in Abbildung 6.3 dargestellt. Eine Herausforderung des Focal Loss ist das Verhalten bei fortgeschrittenem Trainingsprozess. So sind in dieser Phase die meisten Trainingsdaten bereits mit einer hohen Wahrscheinlichkeit korrekt klassifiziert, was zu deutlich kleineren Gradienten führt und dadurch ein langsames Fortschreiten des Trainingsprozesses verursacht.

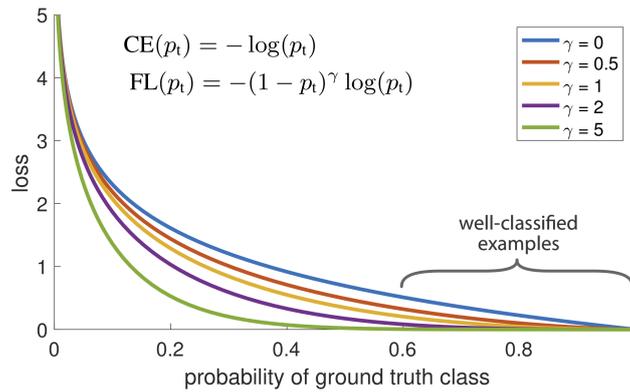


Abbildung 6.3: Mögliche Verläufe des Fehlers für verschiedene Belegungen des Parameters γ beim Focal Loss. Bildquelle: [153]

6.3 Methodik

Zur Gewichtung einzelner Datenpunkte im Trainingsprozess wird an dieser Stelle der **Automated Focal Loss (AFL)** als gewichtende Fehlerfunktion vorgestellt. Für eine formale Definition müssen hierfür noch einige weitere Definitionen eingeführt werden. So wird nachfolgend eine binäre Klassifikation mit y als korrekter Klasse und der Ausgabewahrscheinlichkeit $p \in [0, 1]$ für die Klasse $y = 1$ betrachtet. Der **Automated Focal Loss** \mathcal{L}_{AFL} basiert auf dem Focal Loss [153] \mathcal{L}_{FL} . Dieser fokussiert den Trainingsfehler auf eine Untergruppe der Datenpunkte im Training, welche durch die verwendete Fehlerfunktion schlecht bewertet wurden. Der Focal Loss ist basierend auf dem Cross Entropy Loss zur binären Klassifikation \mathcal{L}_{CE} definiert:

$$\mathcal{L}_{CE} = \begin{cases} -\log(p) & \text{falls } y = 1, \\ -\log(1-p) & \text{sonst.} \end{cases} \quad (6.2)$$

Zur Vereinfachung nachfolgender Definitionen soll in der formalen Notation von der klassenbedingten Fallunterscheidung abstrahiert werden. Hierzu wird die korrekte Klasse eingeführt, welche der annotierten Klasse entspricht. Mit der folgenden formalen Definition der korrekten Klasse $p_{korrekt}$ in einem binären Klassifikationsproblem analog zu [153]:

$$p_{korrekt} = \begin{cases} p & \text{falls } y = 1, \\ 1-p & \text{sonst.} \end{cases} \quad (6.3)$$

ergibt sich für \mathcal{L}_{CE} die Formulierung:

$$\mathcal{L}_{CE} = -\log(p_{korrekt}) \quad (6.4)$$

Die Definition von $p_{korrekt}$ lässt sich in diesem Fall leicht von der binären Klassifikation auf eine Klassifikation mit beliebig vielen Klassen ausweiten. \mathcal{L}_{CE} ist an

6 Gewichtung einzelner Datenpunkte im Trainingsprozess

dieser Stelle bereits generisch formuliert. Der Focal Loss \mathcal{L}_{FL} entspricht hierbei einer gewichteten Fehlerfunktion

$$\mathcal{L}_{FL} = \omega \cdot \mathcal{L}_{CE} \quad (6.5)$$

mit dem Gewichtungsfaktor ω

$$\omega = (1 - p_{korrekt})^\gamma \quad (6.6)$$

Grundlage für die Gewichtung sind somit die Ausgabewahrscheinlichkeiten aller Klassen exklusive der korrekten Klasse. Der Faktor γ repräsentiert den Fokus, der auf Datenpunkte mit geringer $p_{korrekt}$ gerichtet wird. Für diese Datenpunkte besteht eine hohe Wahrscheinlichkeit einer Fehlklassifikation. In [Abbildung 6.3](#) sind mögliche Verläufe für \mathcal{L}_{FL} in Abhängigkeit von γ und $p_{korrekt}$ dargestellt. Hieraus ist ersichtlich, dass mit wachsendem γ der Fokus stärker auf eine geringere Zahl an Datenpunkten mit größerem Fehler gerichtet wird. Wird $\gamma = 0$ gewählt, entspricht dies einer ungewichteten Fehlerfunktion.

Allein durch diese Eigenschaft kann bereits das Problem der Ungleichverteilung einzelner Klassen in den Trainingsdaten reduziert werden. Bei einer Prädiktion von ausschließlich häufig auftretenden Klassen würde eine Fokussierung auf den Fehler der Trainingsdatenpunkte von selten vorkommenden Klassen stattfinden. Somit erfolgt an dieser Stelle implizit bereits eine Gewichtung von seltenen Klassen in den Trainingsdaten, falls diese im Trainingsprozess nicht bereits beachtet wurden.

Zu beachten ist, dass bis zu diesem Zeitpunkt der Parameter γ zu Beginn des Trainings eines Modells festgelegt wird. Er kann als Hyperparameter betrachtet werden, der im Rahmen des kompletten Trainingsprozesses optimiert wird. Zur Visualisierung der Auswirkungen von ω und damit auch von γ auf die rückpropagierten Gradienten im Neuronalen Netz ist ein Blick auf die Ableitung des Focal Loss notwendig. Bei Betrachtung der partiellen Ableitung der Fehlerfunktion nach den Gewichten W des Neuronalen Netzes wird deutlich, dass ω als Konstante hier unverändert erhalten bleibt.

$$\frac{\partial \mathcal{L}_{FL}}{\partial W} = \omega \cdot \frac{\partial \mathcal{L}}{\partial W} \quad (6.7)$$

Somit wirkt sich ω und damit auch die Wahl von γ direkt auf den Fortschritt in verschiedenen Phasen des Trainings aus. Einen normalen Trainingsverlauf angenommen, bei dem zu Beginn $p_{korrekt}$ für die meisten Datenpunkte relativ gering ist und im Verlauf des Trainings im Optimalfall gegen 1 konvergiert, ergeben sich für verschiedene Werte von γ folgende Charakteristika:

Für einen relativ hohen Wert, beispielsweise wie in [Abbildung 6.4](#) dargestellt $\gamma = 5$, ergeben sich zu Beginn des Trainings noch relativ hohe Gewichte. Im Durchschnitt werden diese im Verlauf des Trainingsprozesses allerdings schnell sehr gering, da deutlich weniger schlecht klassifizierte Trainingsdatenpunkte verbleiben. Hiermit werden auch die Gradienten in Summe signifikant reduziert,

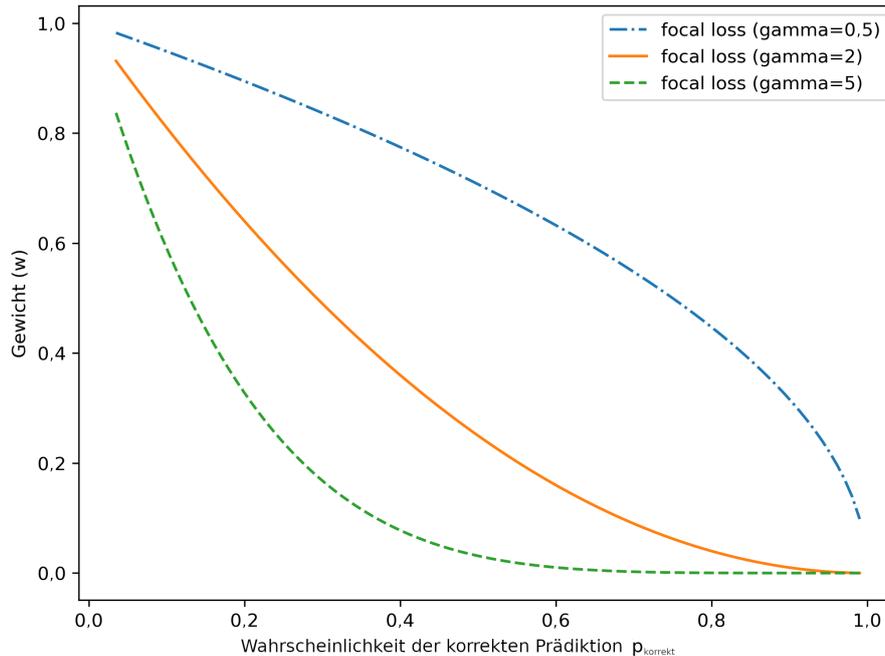


Abbildung 6.4: Verschiedene Verläufe des Gewichtungsfaktors w in Abhängigkeit der Wahl verschiedener fester Werte für γ . Ähnlich veröffentlicht in: [19]

was wiederum zu einem deutlich langsameren Trainingsfortschritt führt. Wird im Gegensatz ein sehr kleiner Wert für γ verwendet, wird die Wirkung des Gewichtungsfaktors w deutlich reduziert. In Abbildung 6.4 ist hier der Verlauf für $\gamma = 0,5$ dargestellt.

Eine Kombination aus verschiedenen γ erscheint an dieser Stelle sinnvoll. Zu Beginn des Trainings ist grundsätzlich ein eher großer Wert für γ wünschenswert, um hier den Fokus auf unsicher oder falsch klassifizierte Datenpunkte zu legen. In Summe über alle Trainingsdatenpunkte sind die Gradienten in dieser Phase meist in ausreichender Größe vorhanden, um den Trainingsfortschritt nicht stark zu verlangsamen. Im weiteren Fortschritt des Trainings werden diese allerdings bedingt durch den Erfolg des Trainingsprozesses sinken. Um hier zu vermeiden, dass diese kleineren Gradienten im Verlaufe der Backpropagation verschwinden und so der weitere Lernprozess verlangsamt oder gestoppt wird, sollte γ im weiteren Trainingsverlauf in Richtung des Wertes 0 streben.

Um an dieser Stelle eine Adaption zu ermöglichen, wird eine automatische Anpassung von γ an den Trainingsfortschritt vorgeschlagen. Ziel dieser dynamischen Anpassung soll ein starker Fokus auf schlecht klassifizierte Datenpunkte zu Beginn des Trainings sein, der sich im Laufe des Trainingsfortschrittes wieder gleichmäßig auf alle Trainingsdatenpunkte verteilt, um das Problem verschwindender Gradienten [116] zu vermeiden.

Der Trainingsfortschritt wird hierbei anhand des Erwartungswertes für die kor-

6 Gewichtung einzelner Datenpunkte im Trainingsprozess

rekte Klassifikation

$$\hat{p}_{korrekt} = E(p_{korrekt}) \quad (6.8)$$

modelliert. Da $\hat{p}_{korrekt}$ einen direkten Einfluss auf die erwartete Gewichtung hat, kann so eine Abhängigkeit zwischen ω und dem Trainingsfortschritt erreicht werden. Dies wird unter Anwendung von Formel (6.6) ersichtlich:

$$E(\log_{\gamma}(w)) = E(1 - p_{korrekt}) = 1 - \hat{p}_{korrekt} \quad (6.9)$$

Zu Beginn des Trainings eines Klassifikators ist $\hat{p}_{korrekt}$ gegeben durch die Anzahl der möglichen Klassen:

$$\hat{p}_{korrekt} \approx \frac{1}{\#\text{Klassen}} \quad (6.10)$$

Im Verlauf des Trainings kann die erwartete Wahrscheinlichkeit der korrekten Vorhersage $\hat{p}_{korrekt}$ durch Mittelwertbildung von $p_{korrekt}$ über den jeweiligen Batch mit Größe N approximiert werden:

$$\hat{p}_{korrekt} \approx \frac{1}{N} \sum_{i=1}^N p_{korrekt}^i \quad (6.11)$$

Sollte pro Batch nur eine kleine Anzahl Trainingsdatenpunkte vorhanden sein, können auch die $\hat{p}_{korrekt}$ der vorherigen Batches durch eine exponentielle Glättung einbezogen werden:

$$\hat{p}_{korrekt}^n = \alpha \cdot \hat{p}_{korrekt}^{n-1} + (1 - \alpha) \cdot p_{korrekt}^n \quad (6.12)$$

Für sehr kleine Batches hat sich hier für α ein Wert von 0,95 empirisch bewährt.

Die Bestimmung von γ in Abhängigkeit des Erwartungswertes für die korrekte Klassifikation kann nun anhand der Informationstheorie [207] basierend auf der Information I erfolgen:

$$\gamma = I(korrekt) = -\log(\hat{p}_{korrekt}) \quad (6.13)$$

So wird der Fokus im Trainingsprozess anhand des Informationsgehalts in $\hat{p}_{korrekt}$ gesteuert. Dieser ist in Abbildung 6.5 dargestellt. Zu Beginn des Trainingsprozesses wird hier ein sehr hohes γ verwendet, während im Verlauf des Trainings mit steigendem Fortschritt eine Konvergenz gegen 0 zu erwarten ist.

In Abbildung 6.6 ist ein Vergleich mit dem festen γ des Focal Loss dargestellt. Im Vergleich hierzu sind die einzelnen Kurven für den Automated Focal Loss Momentaufnahmen zu einem speziellen Zeitpunkt im Training. So wird bei geringem Trainingsfortschritt ($\hat{p}_{korrekt} = 0,01$) der in Orange dargestellte Verlauf maßgeblich. Dieser geht mit steigendem Trainingsfortschritt in den grünen und gegen Ende des Trainings in den roten Verlauf über.

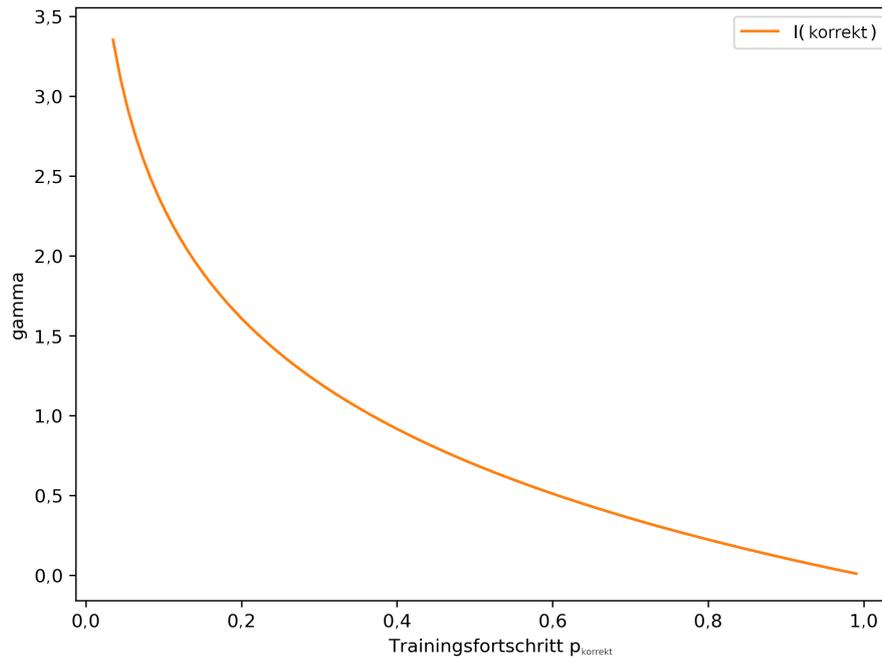


Abbildung 6.5: Die Bestimmung von γ anhand der Information der korrekten Prädiktion $I(korrekt) = -\log(\hat{p}_{korrekt})$. Ähnlich veröffentlicht in: [7]

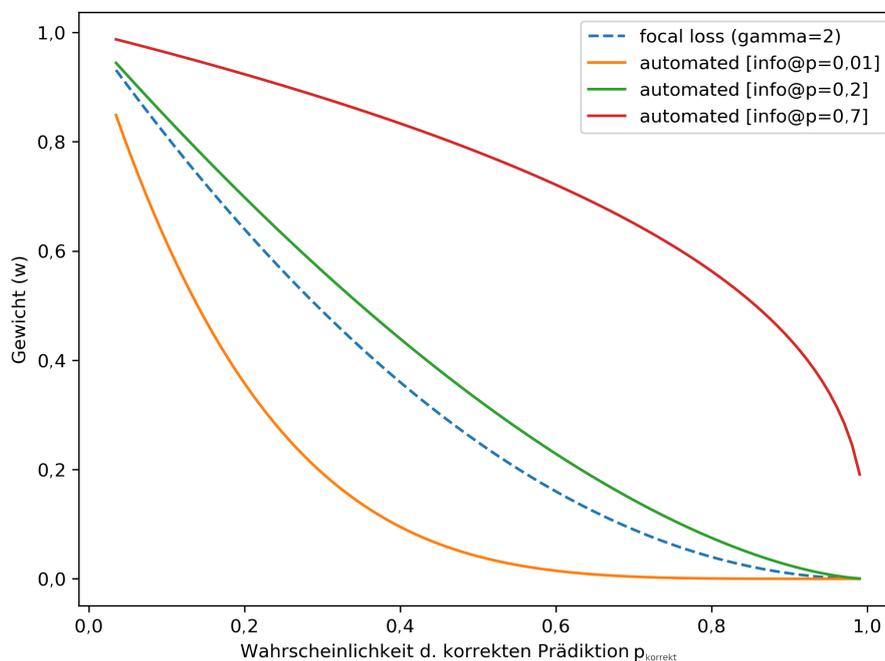


Abbildung 6.6: Verschiedene Verläufe des Gewichtungsfaktors w in Abhängigkeit der Wahl verschiedener fester Werte für γ . Ähnlich veröffentlicht in: [7]

Automated Focal Loss für Regressionsprobleme

Bis zu diesem Punkt wurde der [Automated Focal Loss](#) allein für Klassifikationsprobleme betrachtet. Fokussierende Fehlerfunktionen können allerdings durchaus auch auf Regressionsprobleme wie die Schätzung von Bounding Boxen zur Detektion von Objekten angewandt werden. Um dies zu erreichen, wird eine neue Methode zur Anwendung der fokussierenden Fehlerfunktionen auf die Ausgaben von Fehlerfunktionen zur Regression vorgestellt. Hierzu werden die Regressionswerte in Wahrscheinlichkeitswerte überführt, welche in fokussierenden Fehlerfunktionen verwendet werden können.

Zur Anwendung des Focal Loss auf Regressionsprobleme ist die zentrale Idee, die Wahrscheinlichkeit für den Fall zu berechnen, dass die Ground Truth von der Hypothese besser repräsentiert wird, als von der Annotation. Dies legt die Annahme zugrunde, dass die Annotationen fehlerbehaftet sein können. Da diese meist manuell oder durch Referenzsensorik erzeugt werden, ist eine Verschiebung der Annotation um einen oder wenige Pixel im Kamerabild ein plausibles Szenario. Δx bezeichnet an dieser Stelle die Abweichung zwischen Hypothese und Ground Truth und Δx_t die Abweichung zwischen Annotation und Ground Truth. $p_{korrekt}$ ergibt sich damit in Abhängigkeit von der Wahrscheinlichkeit, dass Δx_t nicht zwischen $\pm|\Delta x|$ liegt:

$$p_{korrekt} = 1 - p(-|\Delta x| < \Delta x_t < |\Delta x|) \quad (6.14)$$

Hierzu wird die Annahme getroffen, dass die Verteilung der Annotationen um die Ground Truth durch eine Gauß-Verteilung mit einer Varianz von σ^2 dargestellt werden kann. Somit kann die Wahrscheinlichkeit $p_{korrekt}$ anhand der kumulativen Verteilungsfunktion Φ der Gauß-Verteilung formuliert werden:

$$p_{korrekt} = 1 - \left(\Phi \left(\frac{|\Delta x|}{\sigma} \right) - \Phi \left(-\frac{|\Delta x|}{\sigma} \right) \right) \quad (6.15)$$

Zur Berechnung von Φ muss allerdings zunächst die Varianz σ^2 bestimmt werden. σ^2 ist den Trainingsdaten inhärent, kann allerdings nicht direkt berechnet werden. Die Unsicherheit, mit denen die Trainingsdaten eines Tasks behaftet sind, kann nach Kendall et al. [135] im Trainingsprozess erlernt werden. Hierzu wird die Fehlerfunktion um den Term $\log(\sigma^2 + 1)$ erweitert, wodurch die Variable σ^2 wie ein Gewicht des Neuronalen Netzes behandelt und im Verlauf des Trainings gelernt wird. Für fokussierende Fehlerfunktionen zur Regression ergibt sich unter Verwendung von $p_{korrekt}$ aus Formel (6.15):

$$\mathcal{L}_{FL} = \omega \cdot \mathcal{L} + \log(\sigma^2 + 1) \quad (6.16)$$

mit

$$\omega = (1 - p_{korrekt})^\gamma \quad (6.17)$$

γ kann hierbei, wie im Fall des klassischen Focal Loss, konstant gewählt werden oder automatisch fokussierend, wie in Gleichung (6.13) vorgeschlagen. Wie beim

Automated Focal Loss für Klassifikationsprobleme kann nun zu Beginn des Trainingsprozesses der Fokus auf Trainingsdatenpunkte mit geringer $p_{korrekt}$ gelegt werden. Im weiteren Verlauf des Trainings wird der Fokus wieder breiter über alle Datenpunkte gestreut. Dies kann auch dazu beitragen, dass der Einfluss einzelner Ausreißer auf die Performance des Modells begrenzt wird.

6.4 Experimente und Evaluation

Die experimentelle Evaluation der in diesem Kapitel vorgestellten Ansätze zur Gewichtung einzelner Datenpunkte im Trainingsprozess wird im Folgenden anhand des Lernproblems der allgemeinen Objektdetektion sowie der 3D Detektion von Objekten vorgestellt. Der zur Evaluation der allgemeinen Objektdetektion verwendete Datensatz **Microsoft Common Objects in Context (MS COCO)** ist in Abschnitt 6.4.1 charakterisiert. Zur Evaluation der 3D Detektion von Objekten wurde der bereits in Abschnitt 4.4.1 vorgestellte KITTI 3D Object Detection-Datensatz [96] verwendet. Für die einzelnen Experimente wurden unterschiedliche Evaluationsmetriken genutzt, welche bereits in den vorherigen Kapiteln eingeführt wurden und in Abschnitt 6.4.2 nochmals zusammengefasst dargestellt sind. Im ersten durchgeführten Experiment wurde in Abschnitt 6.4.3 der in diesem Kapitel vorgestellte Automated Focal Loss mit dem Focal Loss verglichen, welcher aktuell den Stand der Technik im Bereich der Gewichtung einzelner Datenpunkte für Objektdetektionsaufgaben darstellt. In einem weiteren Experiment wurde in Abschnitt 6.4.4 die Leistungsfähigkeit des Automated Focal Loss auf dem Lernproblem der 3D Objektdetektion untersucht.

6.4.1 Datensätze

Zur Evaluation der vorgestellten Konzepte zur Gewichtung der Daten im Trainingsprozess werden verschiedene Datensätze verwendet. Ein zur Detektion von 3D Objekten verwendeter Datensatz ist der *KITTI 3D Object Detection*-Datensatz aus der KITTI Vision Benchmark Suite, der bereits in Abschnitt 4.4.1 vorgestellt wurde. Dieser enthält im Wesentlichen Bilder von Straßenszenen, welche aus einem Fahrzeug aufgenommen wurden. Die Klassen in diesem Datensatz beschränken sich auf dynamische Verkehrsobjekte, wobei die Klassen für Fahrzeuge dominieren. Zusätzlich wird der **Microsoft Common Objects in Context (MS COCO)** [154] im Rahmen der Evaluation verwendet. Dieser ist ein weit verbreiteter Datensatz im Bereich der Bildverarbeitung und wird im nachfolgenden Abschnitt detailliert vorgestellt.

Microsoft Common Objects in Context

Der **Common Objects in Context (COCO)** wurde ursprünglich von Microsoft veröffentlicht, weshalb er noch immer meist als **MS COCO** bezeichnet wird. **COCO** selbst ist ein umfangreicher Bildverarbeitungsdatensatz zum Training Maschinellem Lernverfahren. Für verschiedene Tasks wie die **2D** Objektdetektion, die Objektsegmentierung oder die Posenschätzung sind Annotationen im Datensatz vorhanden. Zusätzlich sind auch Informationen zu den Beziehungen zwischen den vorhandenen Objekten enthalten. **COCO** ist hierbei für die verschiedenen Tasks ein anerkannter Standarddatensatz zum Vergleich verschiedener Verfahren. Dies gilt insbesondere auch für den Task der **2D** Objektdetektion. Der Datensatz besteht insgesamt aus 300.000 Bildern in unterschiedlichen Auflösungen, von denen 220.000 Bilder mit Annotationen versehen sind. Die Bilder entstammen hierbei der Fotocommunity flickr¹.

Für den in diesem Kapitel vorgestellten **Automated Focal Loss** sind insbesondere die Annotationen für den Task der **2D** Objektdetektion von Interesse. Hier enthält **COCO** Annotationen in Form von **2D** Bounding Boxen für 80 verschiedene Objektklassen. Neben klassischen Objekten einer Verkehrsszene wie *Fahrzeug*, *Person* oder *Fahrrad* sind hier auch Objekte und Tiere aus Landschaftsszenen oder verschiedene Objekte aus dem Innenbereich wie Einrichtungsgegenstände oder Nahrungsmittel enthalten. Im Rahmen der hier vorgestellten Evaluation wurde der train/val 2017 Teildatensatz verwendet. Dieser besteht aus 123.287 Bildern und enthält 886.284 Objektinstanzen.

6.4.2 Evaluationsmetriken

Im Rahmen dieser Evaluation werden die beiden Tasks der **2D** Objektdetektion und der **3D** Detektion dynamischer Verkehrsobjekte betrachtet. Zur Messung der Detektionsqualität kommen für beide Tasks Metriken zum Einsatz, welche bereits in Abschnitt 4.4.2 detailliert eingeführt wurden. Konkret sind dies für den Task **2D** Objektdetektion die **Average Precision (AP)** aus Formel (4.7) in den Ausprägungen $AP@[.50:.05:.95]$ und $AP@[.50]$, bezeichnet als AP_{50} . Diese entsprechen den definierten Evaluationsmetriken für den **COCO**-Datensatz. Für den Task der **3D** Detektion dynamischer Verkehrsobjekte sind dies ebenfalls die **Average Precision (AP)** aus Formel (4.7), welche auf die aus der Vogelperspektive betrachteten Objekte angewendet wird (AP_{BE}) und die **Average Orientation Similarity (AOS)** aus Formel (4.8).

Da durch den **Automated Focal Loss** auch der Trainingsprozess selbst beeinflusst wird, müssen hierfür ebenfalls Metriken zur Messung des Erfolgs definiert werden. Für eine Fehlerfunktion sind hier im Wesentlichen die bereits beschriebene erreichbare Güte sowie die Zeitspanne relevant, die ein Trainingsprozess zwischen Beginn des Trainings und dessen Konvergenz benötigt. Diese Zeit-

¹<http://www.flickr.com>

spanne wird im Folgenden als Konvergenzdauer des Trainingsprozesses bezeichnet. Für die einzelnen Trainingsinstanzen wird hierbei manuell eine globale Anzahl an Iterationen festgelegt, die deutlich oberhalb der Konvergenzdauer aller betrachteten Fehlerfunktionen liegt. Die Konvergenzdauer selbst entspricht der Zeitspanne vom Start des Trainingsprozesses bis zu dem Zeitpunkt, zu dem der durch für die jeweilige Fehlerfunktion geringste Fehler erreicht wird.

6.4.3 Experiment 1: Vergleich mit Focal Loss

Für eine Evaluation des [Automated Focal Loss](#) ist ein wichtiger Vergleich die automatische Anpassung der Fokussierung über die Dauer des Trainingsprozesses. Im Vergleich hierzu steht die Gewichtung des Fokus anhand eines statischen Parameters, welcher im Vorfeld für einen Trainingsdatensatz bestimmt werden muss. Da für den in Abschnitt [6.4.1](#) vorgestellten [COCO](#)-Datensatz dieser Parameter mit $\gamma = 2$ bereits bekannt ist [[153](#)], kann hier eine vergleichende Evaluation einfach durchgeführt werden. So kann anhand eines extern bestimmten, optimalen γ ein direkter Vergleich gezogen werden. Der betrachtete Task ist an dieser Stelle die [2D](#) Objektdetektion, mit welcher die Notwendigkeit des Focal Loss ursprünglich gezeigt wurde.

Um einen objektiven Vergleich zu erhalten, wurden die originalen Parameter aus den Experimenten des Focal Loss [[153](#)] übernommen. Ebenfalls wurde mit [RetinaNet](#) [[153](#)] die ursprüngliche Architektur basierend auf [ResNet50](#) [[114](#)] mit einer Skalierung der Eingabebilder auf 400 Pixel verwendet, mit welcher auch der Focal Loss evaluiert wurde. Die Modelle wurden mit der originalen Implementierung des Focal Loss trainiert, die in den eigenen Experimenten eine [AP](#) von 30,41 erreichte. Somit erreichen die selbst trainierten Modelle eine etwas geringere [AP](#), als die bei Veröffentlichung angegebene [AP](#) von 30,5. Dies erscheint allerdings plausibel und ist bereits allein durch die zufällige Initialisierung der Gewichte erklärbar. Das erneute Training von Modellen war an dieser Stelle notwendig, um einen Vergleich der Konvergenzdauer durchführen zu können.

Für die mit dem [Automated Focal Loss](#) trainierten Modelle wurde kein α -balancing verwendet, da die Gewichtung bereits durch die automatische Anpassung des Fokus gegeben ist. Im Gegensatz dazu verwenden die mit dem Focal Loss trainierten Modelle analog zu [[153](#)] α -balancing. Die Resultate in [Tabelle 6.1](#) zeigen, dass die mit dem [Automated Focal Loss](#) trainierten Modelle mit 30,38 dennoch eine vergleichbare [AP](#) erreichen. Der AP_{50} übertrifft mit 51,18 für das mit [Automated Focal Loss](#) trainierte Modell sowohl das publizierte als auch das selbst mit dem Focal Loss trainierte Modell. Somit kann festgehalten werden, dass der [Automated Focal Loss](#) ohne weitere Mechanismen wie α -balancing mit dem Focal Loss vergleichbare Ergebnisse erreicht.

Betrachtet man nun die Konvergenzdauer der durchgeführten Trainingsprozesse, beträgt diese 44 Stunden für den Focal Loss und 30 Stunden für den [Automated Focal Loss](#) auf identischer Hardware. Somit wird für das Training eines Modells

Ansatz	AP	AP ₅₀	Konvergenzdauer Training
Focal Loss – Publiziert [153]	30,5	47,8	n. v.
Focal Loss – Reproduziert	30,41	46,58	44 Std
Automated Focal Loss	30,38	51,18	30 Std

Tabelle 6.1: Vergleich der mit Focal Loss [153] und Automated Focal Loss trainierten Modelle. Zur Messung der Konvergenzdauer mussten die mit dem Focal Loss trainierten Modelle mit dem originalen Framework reproduziert werden. Es ist erkennbar, dass das mit dem Automated Focal Loss trainierte Modell vergleichbare Ergebnisse erzielt – mit lediglich 2/3 der benötigten Trainingszeit.

mit vergleichbaren Ergebnissen lediglich 2/3 der Trainingszeit eines mit dem Focal Loss trainierten Modells benötigt.

Ein interessanter Aspekt des durchgeführten Experiments ist der Verlauf von γ im fortschreitenden Trainingsprozess. Nachdem γ zu Beginn des Trainings meist Werte in der Größenordnung von 6 annahm, verharrte der Wert über weite Strecken bei einer Größe von $\gamma = 2,2$. Dies ist bemerkenswert, da dieser Wert sehr nah an dem mit $\gamma = 2$ angegebenen, optimalen Wert für den Focal Loss liegt. Sobald die AP des Modells im Trainingsprozess einen Wert um 30,0 erreicht hatte und damit in ein Plateau eintrat, begann γ gegen 0 zu konvergieren. Dies ist vor allem aus der Perspektive bemerkenswert, dass der Automated Focal Loss in einzelnen Trainingsprozessen sehr nah an einen angegebenen Optimalwert heranreicht, der anhand verschiedener Experimente manuell optimiert wurde.

6.4.4 Experiment 2: 3D Objektdetektion

Neben einem direkten Vergleich mit dem Focal Loss anhand des Tasks zur Detektion von 2D Objekten im Bildraum auf dem COCO-Datensatz, wurde der Automated Focal Loss auch auf dem komplexen Task der 3D Detektion dynamischer Verkehrsobjekte evaluiert. Hierzu wurde die in Kapitel 4 eingeführte Direct3D-Architektur mit verschiedenen Fehlerfunktionen auf dem in Abschnitt 4.4.1 vorgestellten KITTI 3D Detection Benchmark evaluiert. Als Encoder wurde hierbei die VGG16-Architektur [212] verwendet. Als Eingabebild wurde ein zufälliger Ausschnitt der Größe 256×256 Pixel eines Trainingsbildes verwendet, welches als Bedingung mindestens ein Objekt enthielt. Von jedem im Trainingsdatensatz vorhandenen Bild wurden 20 zufällige Ausschnitte ausgewählt.

Neben dem Automated Focal Loss wurden zum Training weiterer Modelle verschiedene anderen Fehlerfunktionen genutzt, um die Qualität der Detektionsergebnisse vergleichen zu können. Der Focal Loss wurde an dieser Stelle nicht für

Fehlerfunktion	AP_{BE}	AOS	FPS
Normal	-	35,5	28,40
α -balance	-	35,9	28,41
Unsicherheitsb. MultiTask-Loss [135]	20,1	36,1	28,34
Automated Focal Loss (Klass.)	24,5	37,0	28,43
Automated Focal Loss (Klass. + Reg.)	25,0	37,3	28,18

Tabelle 6.2: Ergebnisse der Untersuchung verschiedener Fehlerfunktionen auf dem KITTI 3D Object Detection Datensatz. Neben verschiedenen Ansätzen aus der Literatur wurden der **Automated Focal Loss** zur Klassifikation und der **Automated Focal Loss** für Klassifikation und Regression evaluiert. Als Metrik verwendet wurden hierbei die Average Precision aus Vogelperspektive (AP_{BE}) und die **Average Orientation Similarity (AOS)**. Zusätzlich wurde die Geschwindigkeit der Modelle in **Frames pro Sekunde (FPS)** gemessen. Alle Fehlerfunktionen wurden mit derselben Architektur trainiert.

einen Vergleich herangezogen, da für den verwendeten Datensatz bisher kein optimales γ bekannt und somit ein neutraler Vergleich nicht möglich ist. Stattdessen wurde als einfachste Fehlerfunktion ein nicht gewichteter Fehler verwendet, der im Folgenden als *Normal* bezeichnet wird.

Als einfachste gewichtete Option wurde ein Modell mit durch α -balancing gewichteten Datenpunkten trainiert. Zum Vergleich wurde ebenfalls die in Kapitel 4 eingeführte Variante des unsicherheitsgewichteten MultiTask-Loss [135] verglichen. Der **Automated Focal Loss** wurde in zwei verschiedenen Varianten zum Training verwendet. Einerseits wurde sowohl die Regression als auch die Klassifikation basierend auf dem **Automated Focal Loss** verwendet. Vergleichend hierzu wurde ein Modell mit nicht fokussierender Regression trainiert. So kann auch gezeigt werden, ob die automatisch fokussierte Regression zur Verbesserung des Ergebnisses beiträgt.

Die Konfiguration des Trainingsprozesses war für alle trainierten Modelle die Folgende: Als Optimierungsalgorithmus wurde der Adam-Optimizer [140] mit Standardparametern eingesetzt, welcher bei einer Batchgröße von 16 Bildern für 160.000 Schritte trainiert wurde. Die Lernrate wurde beginnend bei 10^{-4} mit exponentiellem Zerfall bis auf 10^{-6} zum Ende des Trainings reduziert. Für die Fehlerfunktionen Normal und α -balanced wurde der Klassifikationsfehler \mathcal{L}_{CE} gegenüber dem Regressionsfehler mit einem Faktor $\beta = 10$ gewichtet. Bei der α -balanced Fehlerfunktion wurde die Gewichtung auf Basis der Klassenverteilung im jeweiligen Batch berechnet.

Die Ergebnisse der Evaluation für die einzelnen Fehlerfunktionen sind in Tabelle 6.2 aufgeführt. Es ist klar zu erkennen, dass die auf dem **Automated Focal Loss** basierenden Fehlerfunktionen im Vergleich zu den anderen Funktionen bessere Ergebnisse liefern. Der Unterschied zu dem nächstbesten unsicherheitsbasierten

MultiTask-Loss beträgt fast einen Prozentpunkt beim AOS. Ebenfalls geht aus den Ergebnissen hervor, dass eine zusätzliche Verwendung des Automated Focal Loss auch für die Regression mit einer weiteren Verbesserung des AOS von 0,3 Prozentpunkten einhergeht. Eine Betrachtung der AP aus Vogelperspektive (AP_{BE}) bestätigt diese Ergebnisse.

6.5 Diskussion

Der in diesem Kapitel vorgestellte Automated Focal Loss hat zum Ziel, im Trainingsprozess eines Neuronalen Netzes eine automatische Fokussierung auf unterschiedliche Datenpunkte in verschiedenen Phasen des Trainings zu ermöglichen. Nachdem bereits in den Arbeiten zum Focal Loss [153] der Nutzen einer Adaption mittels eines statischen Parameters gezeigt werden konnte, stellt der im Rahmen dieser Arbeit vorgestellte Automated Focal Loss eine Strategie vor, wie auf diesen zusätzlichen Parameter verzichtet werden kann. So konnte anhand der Versuche in Abschnitt 6.4.3 gezeigt werden, dass eine automatische Anpassung des Parameters nach dem vorgeschlagenen Verfahren bei einer geringeren Konvergenzdauer des Trainings vergleichbare Resultate erzielt. Zusätzliche Verfahren zur Anpassung der Trainingsdaten, wie das Hard Negative Mining, oder gewichtete Anpassungen der Fehler einzelner Klassen, wie durch α -balancing, werden für das Training nicht mehr benötigt. Der Automated Focal Loss ermöglicht es dem CNN, sich automatisch auf die wichtigsten Beispiele für den aktuellen Trainingsfortschritt zu konzentrieren.

Die Versuche in Abschnitt 6.4.4 zeigen darüber hinaus, dass der Automated Focal Loss auch für komplexere Lernprobleme erfolgreich eingesetzt werden kann. Hierbei konnte gezeigt werden, dass er anderen Verfahren, wie dem α -balancing oder dem unsicherheitsbasierten MultiTask-Loss überlegen ist. Auch für den vorgestellten Automated Focal Loss zur Regression konnte gezeigt werden, dass dieser die Qualität der Ergebnisse bei einer lediglich minimalen Reduktion der Ausführungsgeschwindigkeit weiter verbessert.

7 Optimierung trainierter CNN-Modelle

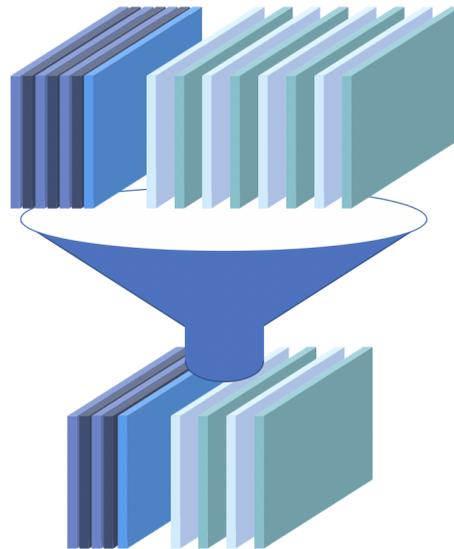


Abbildung 7.1: Optimierung eines CNN-Modells durch Transfer des Wissens auf ein reduziertes Modell mittels Knowledge Distillation.

Da ein Einsatz von CNN-Modellen speziell in der Domäne des Autonomen Fahrens auf dedizierter Hardware erfolgt, sind die Indikatoren Ressourcenverbrauch und Laufzeit eines Modells von großer Wichtigkeit. Durch die Optimierung eines trainierten CNN-Modells können diese Indikatoren für ein optimiertes Modell reduziert werden. Hierbei darf je nach Festsetzung die Qualität der Netzausgaben eine definierte untere Schranke nicht unterschreiten oder nicht unterhalb der Qualität des Originalmodells liegen. Mit der Integration verschiedener Teilnetze in eine gemeinsame MultiNet-Architektur ist in Kapitel 5 bereits ein Konzept vorgestellt worden, wie der Ressourcenverbrauch mehrerer Architekturen zur Erfüllung einzelner Perzeptionsaufgaben durch Integration in ein gemeinsames Modell signifikant reduziert werden kann.

Auch für diese Architekturen wird allerdings bisher im Wesentlichen auf bereits bestehende Bausteine, insbesondere in Form der Encoder zur Extraktion der Features, zurückgegriffen. Diese Standardarchitekturen werden häufig nicht optimiert auf Laufzeit und Ressourcenverbrauch entwickelt. Werden Architekturen auf diese Kriterien optimiert, ist eine Adaption auf die Lernprobleme anderer

Domänen meist nicht trivial, vor allem bei deutlichen Abweichungen der Charakteristika dieser Lernprobleme. Insbesondere die im Rahmen dieser Arbeit untersuchte Detektion statischer Verkehrsobjekte ist hierbei zu nennen, da bei diesem Lernproblem mehrheitlich sehr kleine Objekte zu detektieren sind.

Neben dem Ansatz der Erzeugung problemagnostischer reduzierter Netzarchitekturen existieren weitere Ansätze, bereits trainierte Netzmodelle zu reduzieren oder die zugrundeliegende Architektur unter Beachtung des Lernproblems zu optimieren. In diesem Kapitel soll das Potential derartiger Algorithmen zur Reduktion von Rechenoperationen und Laufzeit untersucht werden. Im Rahmen dieser Untersuchungen wird in Abschnitt 7.3 ein kombinierter Ansatz aus Erstellen einer optimierten Architektur durch gezieltes Entfernen einzelner Filterkerne (Pruning) und der gezielten Übertragung des Wissens eines trainierten Modells auf diese optimierte Architektur mittels Knowledge Distillation vorgestellt. Dieser Ansatz wird anhand der in Kapitel 3 definierten Architektur zur Detektion statischer Verkehrsobjekte bezüglich der Leistung und des Optimierungspotentials in Abschnitt 7.4 evaluiert. Teile dieses Kapitels wurden bereits in der Publikation [12] vorgestellt. Ebenfalls flossen Ergebnisse der Abschlussarbeit [37] in dieses Kapitel ein.

7.1 Problembeschreibung

Die Erforschung neuer Architekturen für CNNs ist getrieben durch das Ziel des bestmöglichen Abschneidens in aktuellen Benchmarks zur Bildklassifikation. So wurden bis 2017 jährlich bei der [ImageNet Large Scale Visual Recognition Challenge \(ILSVRC\)](#) [200] die besten Netzarchitekturen ermittelt. Seit 2018 ist die [ILSVRC](#) auf der zu Google gehörenden Online-Community [Kaggle](#)¹ zu finden, welche sich auf die Durchführung von Data-Science-Wettbewerben spezialisiert hat. Diejenigen CNN-Architekturen, welche in der [ILSVRC](#)-Kategorie Klassifikation ein gutes Ergebnis erzielen können, haben meist sehr gute Chancen, in die Gruppe der weit verbreiteten Feature Extraktoren für CNNs aufzusteigen. Bekannte Netze wie AlexNet [144] (2012 - 1. Platz), GoogLeNet [224] (2014 - 1. Platz), VGG [212] (2014 - 2. Platz) oder ResNet [114] (2015 - 1. Platz) sind hierfür gute Beispiele. Obwohl sich der Fokus der [ILSVRC](#) in den letzten Jahren weg von der Klassifikation hin zur Lokalisation und Detektion von Objekten verschoben hat, blieb dennoch das Abschneiden bei der Klassifikation als das primäre Qualitätskriterium für CNN-Architekturen bestehen.

Leider ist das Optimierungskriterium für ein gutes Abschneiden bei derartigen Wettbewerben meist allein die erreichte Fehlerrate. Der Fokus liegt somit auf der erreichten Qualität und nicht auf den für dieses Ergebnis benötigten Ressourcen.

¹www.kaggle.com

Dies führte zu einer stetigen Entwicklung von immer tieferen und ressourcenintensiveren Modellen. Das bestimmende Designkriterium ist damit bei der Entwicklung genereller CNN-Architekturen zur Extraktion der Features nicht geeignet, um ressourcen- und laufzeitoptimale Architekturen zu erzeugen. Vorteil dieser Architekturen hingegen ist, dass eine Übertragung auf Lernprobleme jenseits der Bildklassifikation in unterschiedliche Anwendungsdomänen zwischenzeitlich breit untersucht ist.

Eine Optimierung dieser Ansätze in Anwendungen mit Lernproblemen für verschiedene Tasks und zeitgleicher Ausführung der resultierenden Modelle ist mit dem MultiNet-Prinzip aus Kapitel 5 möglich. So kann der Ressourcenverbrauch verschiedener Modelle zur Lösung unterschiedlicher Tasks mit gleichem Encoder deutlich gesenkt werden. Dies geschieht unter Beibehaltung der bisherigen Architektur, bei welcher die Leistungsfähigkeit für die einzelnen Lernprobleme bekannt ist. Allerdings kann so lediglich der Ressourcenverbrauch für das Teilnetz zur Extraktion der Features reduziert werden. Die Decoder der einzelnen Tasks werden bei dieser Methode nicht beachtet.

Zwischenzeitlich existieren auch vielfältige Bestrebungen, optimierte Encoder-Netze direkt durch Modifikation der Architekturen sowie der im Einzelnen verwendeten Schichten zu erstellen. Hieraus hervorgegangene Architekturen werden in Abschnitt 7.2.3 detailliert vorgestellt. Um diese Architekturen für Lernprobleme wie die Objektdetektion zu verwenden, werden meist auch die Detektoren selbst nochmals an die veränderten Architekturen angepasst. Die domänenübergreifende Anwendbarkeit dieser Detektoren ist bislang nicht in gleichem Umfang evaluiert, wie die Detektion basierend auf den in Abschnitt 3.2.1 beschriebenen, standardisierten Objektdetektoren. Auch erreicht der Trainingsprozess im Normalfall bessere Ergebnisse, wenn zunächst große Architekturen trainiert, und diese nachfolgend reduziert werden [41][189].

7.2 Verwandte Arbeiten

Bei der Betrachtung von Methoden zur Optimierung von CNNs lassen sich zwei verschiedene Paradigmen unterscheiden. Zum einen kann die Repräsentation der Parameter in den einzelnen Filterkernen angepasst werden. So kann bei entsprechend verfügbarer Hardware auch die Laufzeit der Netzausführung verringert werden. Zum anderen kann die Anzahl der Parameter im Netz reduziert werden. Da die Anzahl der Parameter einen großen Einfluss auf die Anzahl der durchzuführenden Rechenoperationen im Netz hat, kann so auch diese Anzahl reduziert werden. Bei Verwendung der gleichen Hardware reduziert sich somit meist ebenfalls die Laufzeit der Modelle.

Eine Ausnahme stellt hier lediglich der Fall einer überdimensionierten Hardware dar. Bei einem Überangebot an Recheneinheiten kann eine Reduktion der Laufzeit nicht garantiert werden. In Anwendungsfällen wie dem betrachteten Au-

tonomen Fahren allerdings werden Recheneinheiten im Normalfall anhand des Ressourcenbedarfs der einzelnen Modelle dimensioniert. So wird der Fall eines Überangebotes an Recheneinheiten an dieser Stelle nicht betrachtet.

Die einzelnen Ansätze der genannten Paradigmen werden nachfolgend detaillierter beleuchtet. So wird in Abschnitt 7.2.1 das Paradigma der angepassten Parameterrepräsentation betrachtet. Zur Reduktion der Anzahl der Parameter werden mehrere Ansätze vorgestellt. So wird in Abschnitt 7.2.2 zunächst die Technik des Prunings von Parametern genauer betrachtet. Ein Überblick über den generellen Entwurf reduzierter Architekturen ist in Abschnitt 7.2.3 zu finden. Abschließend wird mit [Knowledge Distillation \(KD\)](#) eine Technik zum Übertragen von gelerntem Wissen auf reduzierte Architekturen in Abschnitt 7.2.4 beschrieben.

7.2.1 Parameterrepräsentation

Ein einfacher Ansatz zur Reduktion des Ressourcenverbrauchs von [CNN-Modellen](#) ist die Anpassung der Repräsentation bei den enthaltenen Parametern. Ohne weitere Optimierung werden diese Parameter in Form von float32-Werten gespeichert. Somit werden zur Speicherung jedes Werts 32 Bit benötigt. Für die Durchführung einer Rechenoperation auf einem oder mehreren dieser Werte bedarf es ebenfalls einer Ausführungseinheit der entsprechenden Breite. Eine Anpassung der Repräsentation bietet an dieser Stelle also Potential zur Reduktion des Speicherverbrauchs und einer Verwendung kleinerer Ausführungseinheiten.

Eine Anpassung in Form einer Quantisierung verfolgt das Ziel, die Speichergröße und somit auch den Berechnungsaufwand erlernter Modelle durch Diskretisierung der Parameter zu reduzieren. Bei tiefen Neuronalen Netzen im Allgemeinen und [CNNs](#) im Speziellen werden die Parameter der einzelnen Schichten während des Trainingsprozesses in Form von 32 Bit Gleitkommazahlen im Speicher gehalten. Werden diese Parameter nun in 8 Bit Integerwerte quantisiert, benötigen sie weniger Speicherplatz und können je nach Verfügbarkeit auf anderen Recheneinheiten ausgeführt werden. Grundsätzlich sind auch Quantisierungen in andere Datentypen möglich. Diese werden allerdings seltener verwendet. Bei einer Quantisierung in 8 Bit Integer kann der kontinuierliche Wertebereich der Parameter in $2^8 = 256$ Intervalle eingeteilt werden.

Verschiedene Ansätze unterscheiden sich hierbei meist durch die Art der Festlegung der Intervallgrenzen. Eine einfache Möglichkeit zur Festlegung dieser Grenzen bietet die lineare Quantisierung [234], welche eine Teilung des ursprünglichen Wertebereichs in n Intervalle gleicher Größe durchführt. Zur Bestimmung der Intervalle kann auch ein Clustering der verwendeten Parameter mittels k-Means-Algorithmus durchgeführt werden [103][108]. Ebenso ist eine Quantisierung in Zweierpotenzen möglich [249]. Einen weiterführenden Ansatz stellt das XNOR-Net [185] dar. In dieser Netzarchitektur wird eine binäre Repräsentation der Parameter genutzt.

Neben einer Quantisierung können auch Gleitkommazahlen mit reduzierter Genauigkeit zur Repräsentation der Parameter verwendet werden. So kann einerseits der Speicherverbrauch für die Parameter reduziert werden. Andererseits kann je nach verwendeten Recheneinheiten auch die Zahl der zeitgleich ausgeführten Berechnungen erhöht werden. So ist es mittlerweile häufig möglich, mit einer Recheneinheit statt einer Operation auf 32 Bit Gleitkommazahlen zwei Operationen auf 16 Bit Gleitkommazahlen in einem Zeitschritt auszuführen. Hierzu wurden verschiedene Untersuchungen mit unterschiedlichen Fest- und Gleitkommaformaten durchgeführt [63][107].

7.2.2 Pruning

Das Pruning in Neuronalen Netzen geht zurück auf den *Optimal Brain Damage (OBD)*-Ansatz [149]. Bereits 1990 wurde dort erkannt, dass in einem Neuronalen Netz viele unwichtige Kanten existieren, welche aus dem trainierten Netz mit vernachlässigbaren Einbußen bei der Genauigkeit der Ausgaben entfernt werden können. Dies geht vermutlich darauf zurück, dass die Architekturen für Neuronale Netze im Normalfall für das zu lernende Lernproblem überparametriert sind [68][77]. Für die ursprünglichen Arbeiten zu Pruning wurde ein klassisches **Künstliches Neuronales Netz (KNN)** verwendet, weshalb die Entfernung von einzelnen Kanten den Weg zur Reduktion der Parameter des Netzes bildete.

Bei der Verwendung von **CNNs** bieten sich mehrere Strategien, das Pruning von Parametern einzusetzen. Besteht ein Teil der Architektur aus Fully-Connected-Schichten, können grundsätzlich die Strategien der klassischen **KNNs** verwendet werden [215]. Betrachtet man die architekturbedingten Eigenheiten von **CNNs**, kann ein Pruning auf drei verschiedene Bestandteile angewandt werden. So können einzelne Parameter, ganze Filter oder komplette Schichten aus einem Netz entfernt werden. Allerdings ist die Entfernung einer Schicht meist eher der Definition einer Architektur zuzurechnen und wird für ein bereits trainiertes **CNN** selten verwendet.

Das Pruning einzelner Parameter wird häufig in klassischen **KNNs** durchgeführt, ist jedoch nicht einfach auf **CNNs** übertragbar. So bestehen **KNNs** aus einzelnen Kanten mit je einem Parameter, die leicht aus dem Gesamtnetz zu entfernen sind. Die Parameter in **CNNs** befinden sich im Wesentlichen in den Filterkernen der Convolution-Schichten, wobei ein Filterkern im Normalfall aus einer Vielzahl an Parametern besteht. Eine Entfernung einzelner Parameter ist an dieser Stelle bei Verwendung herkömmlicher Hardware nicht direkt möglich, allerdings kann dies durch ein Ersetzen der Parameter mit 0 simuliert werden. Eine Reduktion von Speicherverbrauch oder benötigten Rechenoperationen ist auf diese Weise jedoch nicht möglich [77]. Das Pruning einzelner Parameter [109] hat nur einen Einfluss auf den Berechnungsaufwand, falls spezialisierte und somit abgestimmte Hardware verwendet wird.

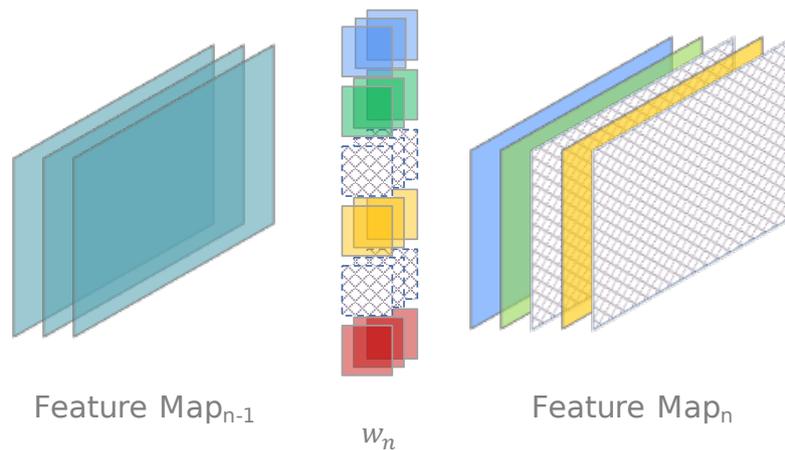


Abbildung 7.2: Durchführung von Channel Pruning auf den Filterkernen einer Schicht (Mitte). Aus den Ausgaben der vorherigen Schicht (links) werden die Ausgaben der aktuellen Schicht (rechts) berechnet. Werden durch Pruning zwei Filterkerne entfernt (grau), reduziert sich auch die Größe der Ausgabe für die aktuelle Schicht.

Bei Verwendung von herkömmlicher Hardware müssen Parameter strukturiert entfernt werden, um einen Effekt auf den Berechnungsaufwand zu erzeugen. Eine einfache Möglichkeit zur strukturierten Entfernung mehrerer Parameter ist dabei das Pruning ganzer Filterkerne. Wird in einer Convolution-Schicht ein Filterkern entfernt, entfällt eine Faltungsoperation auf der Eingabe der Schicht, wie in Abbildung 7.2 dargestellt. Zusätzlich ist die Ausgabe der Schicht um eine Feature Map reduziert. Die Tiefe der Filterkerne in der nachfolgenden Convolution-Schicht wird somit auch um eins reduziert. Dies wiederum reduziert die Anzahl der benötigten Rechenoperationen in der nachfolgenden Schicht.

Da bereits trainierte Modelle zum Pruning verwendet werden, befinden sich diese vor dem Pruning in einem lokalen Optimum. Nach Durchführung eines Prunings ist dies größtenteils nicht mehr der Fall, weshalb ein Fine-Tuning des Modells durchgeführt werden sollte. [180] So kann in einzelnen Schichten der Wegfall von Eingaben durch die Verbindung mit ähnlichen Eingaben kompensiert werden.

7.2.3 Verwendung reduzierter Architekturen

Im Gegensatz zu dem in Abschnitt 7.2.2 vorgestellten Ansatz des Prunings von Gewichten bei trainierten Modellen kann die Präferenz von kleinen Modellen auch bereits in die grundsätzliche Architekturentscheidung einfließen. So kann die Wahl und Ausgestaltung der einzelnen Schichten in einem CNN bereits die Anzahl der Parameter und die letztendliche Laufzeit maßgeblich beeinflussen. Beispielsweise ist das ursprüngliche GoogLeNet dem zeitgleich erschienenen und von der Qualität der Ergebnisse vergleichbaren VGG bezüglich der geringeren

Anzahl von Parametern und Rechenoperationen deutlich überlegen (vgl. Tabelle 2.1).

Für die Definition neuer reduzierter Architekturen werden überwiegend bereits vorhandene Schichten und Verfahren wie die Depthwise Separable Convolutions [211] oder Group Convolution [144] genutzt sowie neue Verfahren wie Fire Modules [123] vorgestellt. Mit diesen werden dann meist größere Architekturbausteine innerhalb der einzelnen Ansätze definiert und in den Architekturen vielfach verwendet [123][118][245]. Teilweise wird zur Erzeugung der verwendeten finalen Architekturen auch auf automatisierte Techniken zur Architektursuche zurückgegriffen [117][227].

Die reduzierten Architekturen zur Detektion von Objekten basieren größtenteils auf entsprechenden reduzierten Architekturen zur Extraktion von Features. So wird bei EfficientDet [228] das EfficientNet [227] zur Extraktion der Features genutzt. SqueezeDet [238] verwendet zum selben Zweck SqueezeNet [123], SSDLite [201] nutzt das MobileNetv2 [201] und in ThunderNet [184] wird eine Variante von ShuffleNetv2 [161] verwendet. Da meist ein großer Anteil der Parameter und Rechenoperationen in den Schichten zur Extraktion der Features enthalten ist (vgl. Tabelle 5.4), kann so bereits die Größe und Laufzeit deutlich reduziert werden.

Zusätzlich werden von den einzelnen Ansätzen weitere auf die Detektion von Objekten zugeschnittene Techniken vorgestellt und verwendet. In SqueezeDet werden ConvDet-Schichten zur Detektion eingeführt, die eine Abwandlung des RPN [190] darstellen. EfficientDet führt mit Bidirectional Feature Pyramid Network (BiFPN) ein optimiertes FPN zur Fusion mehrerer Feature Maps ein. Bei der Erstellung von SSDLite [201] werden auch für die Detektionsarchitektur die bereits für die Extraktion von Features verwendeten Depthwise Separable Convolutions [211] verwendet. ThunderNet [184] führt mit Spatial Attention Module (SAM) und Context Enhancement Module (CEM) zwei neue Architekturbausteine zur Verschmelzung von Feature Maps aus verschiedenen Skalen und zum Verwenden von Informationen aus RPNs ein.

Nachfolgend werden einige reduzierte Architekturen zur Extraktion von Features inklusive der durch sie eingeführten Architekturbausteine und Techniken nochmals zusammengefasst:

SqueezeNet [123] ist eine der ersten reduzierten Architekturen. Bei einem Vergleich der Klassifikationsgüte mit AlexNet [144] erreicht SqueezeNet eine vergleichbare Performance bei einer Modellgröße von lediglich 0,5 MB. Wesentlicher Bestandteil der Architektur ist das sogenannte *Fire Module*, das auch *squeeze and expand*-Schicht genannt wird. Dieses wird hauptsächlich dazu eingesetzt, die Anzahl der Eingabekanäle für 3×3 Filter zu reduzieren. Zusätzlich werden in den Schichten einige 3×3 Filter durch 1×1 Filter ersetzt.

MobileNet bezeichnet eine Gruppe von leichtgewichtigen CNN-Architekturen für die Extraktion von Features. In MobileNetV1 [118] werden als zentraler Bestandteil die Depthwise Separable Convolutions [211] zur Verwendung in reduzierten Architekturen eingeführt. Diese werden auch zur Reduktion der Auflösung im Netz anstelle von Pooling-Schichten verwendet. MobileNetV2 [201] führt zusätzlich einen *Mobile Inverted Residuals Bottleneck Block* als neue Architekturkomponente sowie ReLU6 als neue Aktivierungsfunktion ein. Für MobileNetV3 [117] werden im Wesentlichen dieselben Bausteine verwendet, die bereits in den Vorgängerarchitekturen eingeführt wurden. Allerdings wird zur Findung einer Architektur verstärkt auf eine *Network Architecture Search* (NAS) zurückgegriffen.

ShuffleNet bezeichnet bislang zwei Architekturen optimierter CNNs, die sich vorwiegend mit der Gruppierung von Filtern beschäftigen. Das Gruppieren von Filterkernen in Convolution-Schichten geht auf AlexNet [144] zurück und wurde ursprünglich zum verteilten Training auf limitierten Hardware-Ressourcen eingeführt. In ShuffleNetV1 [245] wird diese Group Convolution mit Depthwise Separable Convolutions [211] zur *Pointwise Group Convolution* kombiniert. Mit *Channel Shuffle* wird zusätzlich eine Technik zur Verteilung der Informationen zwischen den Gruppen eingeführt, die weitgehend dem Tiling aus OverFeat [205] entspricht. ShuffleNetV2 [161] verändert im Wesentlichen die Anordnung dieser Blöcke, wodurch die Gruppenbehandlung außerhalb ganzer Netzblöcke durchgeführt wird.

EfficientNet [227] betrachtet bei der Reduktion der Architektur das Zusammenwirken von Tiefe, Breite und Eingabeauflösung eines CNN. So kann durch eine sorgfältige Abstimmung dieser Parameter die Performance der resultierenden Modelle gesteigert werden. Die Ableitung verschiedener Netzarchitekturen durch gemeinsame Anpassungen der genannten Parameter wird als *Compound Scaling* bezeichnet und durch eine automatisierte Architektursuche [226] realisiert. Auf diese Art werden acht verschiedene Architekturen definiert (B0-B7). Hauptbestandteile dieser Architekturen sind invertierte Bottleneck-Blöcke [201] und eine squeeze-and-excitation Optimierung [119].

7.2.4 Knowledge Distillation

Der Begriff der *Knowledge Distillation* (KD) geht zurück auf Hinton et al. [115], bezeichnet allerdings eine bereits 10 Jahre zuvor unter dem Namen Model Compression [49] vorgestellte Methode zur Reduktion der Größe eines Neuronalen Netzes. Zentrale Idee dieses Ansatzes ist das Training eines kleinen Netzes, das die gelernte Funktion eines größeren Netzes approximiert. Das Netz, welches die zu approximierende Zielfunktion bereitstellt, wird auch als *Teacher* bezeichnet, das lernende Netz entsprechend als *Student*. Das Training des Students kann bei

KD auf dem originalen Trainingsdatensatz oder einem anderen Transferdatensatz durchgeführt werden.

Grundsätzliches Ziel von KD ist es, die zur Bearbeitung eines bestimmten Tasks benötigten Hardware-Ressourcen zu reduzieren. So wird das Wissen eines großen Modells oder eines ganzen Ensembles von Modellen [53] in Form der Ausgaben auf ein kleines Modell während dessen Trainingsprozess übertragen. Dieses kleine Modell kann bereits an die maximale Größe der Zielhardware angepasst sein. Die Qualität der Netzausgaben soll hierbei natürlich erhalten bleiben, was allerdings durch eine starke Reduktion der Kapazität der Netzarchitektur nicht notwendigerweise gegeben ist.

Bis zu einer gewissen Reduktion der Kapazität kann die Generalisierungsfähigkeit des Modells beibehalten oder sogar gesteigert werden [115]. Wird jedoch eine Architektur mit zu geringer Kapazität verwendet, nimmt diese zwangsläufig ab. Falls die Zielhardware bereits festgelegt ist, kann so allerdings die maximal erreichbare Qualität der Ergebnisse auf einer an diese Hardware angepasste Architektur ermittelt werden. Eine detaillierte Übersicht über aktuelle Arbeiten zu KD ist in [105] zu finden.

7.3 Methodik

Zur Optimierung eines CNNs wird in dieser Arbeit ein kombinierter Ansatz aus Channel Pruning und Knowledge Distillation vorgeschlagen, wie in Abbildung 7.3 dargestellt. In einem ersten Schritt wird die Anzahl der Parameter eines bereits trainiertes Modells durch Entfernen einzelner Filterkerne reduziert. Nach jedem Pruning-Schritt findet ein Fine-Tuning des Modells statt, da sich das Modell durch das Entfernen einzelner Filterkerne nicht mehr in dem im Trainingsprozess erreichten lokalen Optimum befindet. Die genaue Vorgehensweise dieses Channel Prunings ist in Abschnitt 7.3.1 beschrieben.

Die aus diesem Verfahren resultierende Architektur wird nun im Rahmen der Knowledge Distillation als Student Network verwendet. Das als Grundlage für das Pruning genutzte Modell wird entsprechend als Teacher Network verwendet. Das ursprünglich im Training des originalen Modells gelernte Wissen kann so direkt an die optimierte Architektur übertragen werden. Die Details der vorgeschlagenen Methode sind in Abschnitt 7.3.2 beschrieben.

Durch die Anwendung der vorgeschlagenen Methode wird lediglich die Architektur des ursprünglichen Modells verändert. Die Repräsentation der Parameter ist hiervon nicht betroffen. So benötigt das resultierende Modell im Gegensatz zu anderen Optimierungstechniken zur Ausführung keine spezialisierte Hardware. Bei Bedarf können zusätzlich weitere Methoden zur Optimierung des Modells, insbesondere aus dem in Abschnitt 7.2.1 beschriebenen Gebiet der Anpassung der Gewichtsrepräsentation, angewandt werden.

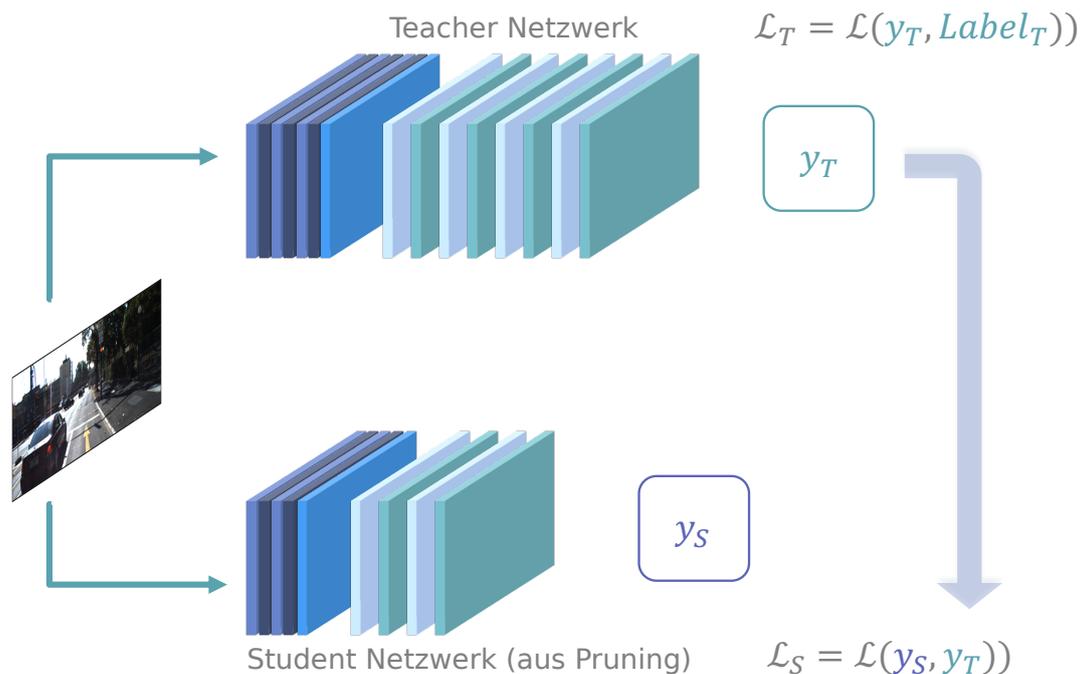


Abbildung 7.3: Vorgeschlagenes Verfahren zur Optimierung eines trainierten CNN-Modells. Eine durch Channel Pruning reduzierte Architektur wird mittels Knowledge Distillation mit den Ausgaben des Originalmodells trainiert.

7.3.1 Netzoptimierung

Im ersten Schritt des vorgestellten Verfahrens werden aus dem zu optimierenden Modell Schicht für Schicht mittels Channel Pruning einzelne Filterkerne entfernt. Somit sind die resultierenden Modelle auf herkömmlicher Hardware ausführbar und nicht beschränkt auf spezielle Recheneinheiten für spärlich besetzte Filter. Das genutzte Verfahren ist motiviert durch die für ThiNet [160] vorgestellte Vorgehensweise eines Channel Pruning.

Mit einem gegebenen Modell erfolgt die Optimierung Schicht für Schicht von der Eingabeschicht bis zur Ausgabeschicht gemäß folgender Schritte:

1. Bestimmung der Wichtigkeit aller Filterkerne
2. Pruning der unwichtigsten Filterkerne
3. Fine-Tuning des neuen Modells

Im Gegensatz zu dem in ThiNet vorgeschlagenen Verfahren werden hierbei die Schritte (1) und (2) auf alle Filterkerne der betrachteten Schicht zeitgleich angewandt. So kann die Dauer des Verfahrens deutlich reduziert werden.

Generell besteht die Bestimmung der Wichtigkeit einzelner Filterkerne in der ersten Stufe aus einer Abwägung zwischen der Qualität des Modells und einer realistischen Durchführungsdauer für das Verfahren. Gesetzt den Fall unbegrenz-

Die Anzahl der zu entfernenden Filterkerne kann anhand der Pruning-Rate entweder global für das ganze Modell oder dynamisch für jede Schicht individuell festgelegt werden. So kann für die resultierende Architektur die Größe der einzelnen Schichten bestimmt werden. Eine dynamische Bestimmung anhand der Fehlerrate oder dem MSE der einzelnen Filterkerne in der betreffenden Schicht ist ebenfalls möglich.

Nachdem das Pruning für eine Schicht abgeschlossen wurde, muss ein Fine-Tuning des Modells durchgeführt werden. Dies ist notwendig, da das Modell sich durch das Training in einem lokalen Optimum befindet, welches allerdings durch das Pruning verlassen wird. Durch das Fine-Tuning hat das Modell die Möglichkeit, das Fehlen der entfernten Schichten durch andere Eingaben zu kompensieren. Da hierbei allerdings nur kleine Änderungen im Modell durchgeführt werden sollen, wird an dieser Stelle lediglich eine kleine Lernrate verwendet.

7.3.2 Wissensübertragung

Die Methode der Knowledge Distillation kann nun verwendet werden, um das gelernte Wissen des ursprünglichen Modells nochmals auf das durch Channel Pruning reduzierte Modell zu übertragen. Für ein hierarchisches Detektionsmodell muss hierbei zwischen der hierarchischen Klassifikation der einzelnen Bildregionen in Wahrscheinlichkeitskarten und der Regression der finalen Bounding Boxen für die Position der Objektkandidaten unterschieden werden. Dies spiegelt sich auch direkt in der verwendeten Fehlerfunktion wieder:

$$\mathcal{L} = \sum_h \mathcal{L}_h + \alpha \cdot \mathcal{L}_{KD} + \beta \cdot \mathcal{L}_{reg} \quad (7.1)$$

Diese setzt sich aus drei verschiedenen Teiltermen zusammen, welche durch die Faktoren α und β gewichtet werden. Der erste Term entspricht an dieser Stelle der ursprünglichen Fehlerfunktion für die Klassifikation des Modells $\sum_h \mathcal{L}_h$ über alle Hierarchiestufen hinweg. Der Fehler wird hierbei anhand der Netzausgabe des Student Network und den ursprünglichen Annotationen berechnet. In einem zweiten Term wird das Verfahren zur Knowledge Distillation [115] auf den Klassifikationsteil des Netzes angewendet. Basis der Fehlerfunktion sind an dieser Stelle die Ausgabe von Teacher und Student Network. Hierbei sind allerdings die Besonderheiten der hierarchischen Detektion zu beachten. So besteht die Fehlerfunktion der Knowledge Distillation für die Klassifikation \mathcal{L}_{KD} aus je einem Fehleranteil für jede Hierarchiestufe:

$$\mathcal{L}_{KD} = \mathcal{L}_{k1} + \mathcal{L}_{k2} + \mathcal{L}_{k3} \quad (7.2)$$

Für die erste Stufe der Hierarchie H_1 und deren Fehleranteil \mathcal{L}_{k1} wird jeweils die komplette Ausgabe von Student und Teacher verwendet. Für alle späteren Stufen

wird analog zu der Fehlerfunktion in Abschnitt 3.3.3 lediglich die Ausgabe beider Modelle an den Stellen betrachtet, an denen ein Objekt vorhanden ist. Für die Klassifikation wird folgende Fehlerfunktion angewandt:

$$\mathcal{L}_{k_h} = - \sum_o \sum_c w_{h,c} \cdot t_{o,c} \cdot \log(s_{o,c}) \quad (7.3)$$

Hierbei bezeichnet $t_{o,c}$ die Ausgabe des Teachers an Pixel o für Klasse c . Analog dazu ist $s_{o,c}$ für den Student definiert. Der Gewichtungparameter $w_{h,c}$ bestimmt die Gewichtung der einzelnen Klassen in der jeweiligen Hierarchiestufe.

Eine Nutzung von Knowledge Distillation für Regressionsprobleme ist im ursprünglichen Ansatz nicht vorgesehen. Nachfolgend wird ein eigener Ansatz anhand folgender Fehlerfunktion vorgestellt:

$$\mathcal{L}_{reg} = \sum_o \sum_r \min\{\text{L1}(t_{o,r}, s_{o,r}), \text{L1}(l_{o,r}, s_{o,r})\} \quad (7.4)$$

Dieser bezeichnet den Fehler an einer Stelle o im Bild für einen Parameter r einer Bounding Box. Für die Ausgabe des Students wird hierbei die L1-Distanz zur Ausgabe des Teachers und der Annotation bestimmt. Das Minimum hieraus entspricht dem finalen Fehler. Auch diese Fehlerfunktion wird lediglich an Stellen ausgewertet, an denen im Eingabebild gemäß der Annotation ein Objekt vorhanden ist.

7.4 Experimente und Evaluation

Die Evaluation des vorgestellten Ansatzes zur Netzoptimierung erfolgt anhand des in Kapitel 3 beschriebenen HDTLR-Systems zur hierarchischen Ampelerkennung. Dieses CNN entspricht einer typischen Architektur für bildbasierte Systeme zur Umgebungswahrnehmung. Die grundsätzliche Architektur kann in dieser Form für verschiedenste 2D Detektionsaufgaben verwendet werden, wobei der für die Klassifikation genutzte Teil des Netzes mit den resultierenden Wahrscheinlichkeitskarten in dieser Form auch die Grundlage für Systeme zur Bildsegmentierung bildet. Ebenso kann die Netzarchitektur zur Klassifikation ganzer Bilder verwendet werden, bei entsprechender Abstimmung von Eingabebildgröße mit den Netzparametern. Dies ergibt sich aus dem Umstand, dass diese 2D Detektionsnetze auf Erweiterungen von Klassifikationsnetzen zurückgehen. So können die hier gewonnenen Erkenntnisse auch auf vergleichbare Systeme zur Bildklassifikation, Bildsegmentierung oder 2D Objektdetektion übertragen werden.

Die für diese Evaluation konkret verwendete CNN-Architektur ist in Tabelle 7.1 genauer beschrieben. Die Basis bildet der bereits in Abschnitt 2.1 näher erläuterte VGG-Encoder [212], der um zwei weitere Convolution-Schichten erweitert

Schicht	Filter	Größe	Eingabedimension	Parameter	FLOPs
conv1_1	64	3×3	960 × 1.280 × 3	1,7 Tsd	4,2 Mrd
conv1_2	64	3×3	960 × 1.280 × 64	36 Tsd	90,6 Mrd
MaxPool	-	2×2	960 × 1.280 × 64	-	
conv2_1	128	3×3	480 × 640 × 64	73 Tsd	45,3 Mrd
conv2_2	128	3×3	480 × 640 × 128	147 Tsd	90,6 Mrd
MaxPool	-	2×2	480 × 640 × 128	-	
conv3_1	256	3×3	240 × 320 × 128	295 Tsd	45,3 Mrd
conv3_2	256	3×3	240 × 320 × 256	590 Tsd	90,6 Mrd
conv3_3	256	3×3	240 × 320 × 256	590 Tsd	90,6 Mrd
MaxPool	-	2×2	240 × 320 × 256	-	
conv4_1	512	3×3	120 × 160 × 256	1,1 Mio	45,3 Mrd
conv4_2	512	3×3	120 × 160 × 512	2,3 Mio	90,5 Mrd
conv4_3	512	3×3	120 × 160 × 512	2,3 Mio	90,5 Mrd
MaxPool	-	2×2	120 × 160 × 512	-	
conv5_1	512	3×3	60 × 80 × 512	2,3 Mio	22,6 Mrd
conv5_2	512	3×3	60 × 80 × 512	2,3 Mio	22,6 Mrd
conv5_3	512	3×3	60 × 80 × 512	2,3 Mio	22,6 Mrd
MaxPool	-	2×2	60 × 80 × 512	-	
conv6-tl	4.096	7×7	30 × 40 × 512	102 Mio	247 Mrd
conv7-tl	4.096	1×1	30 × 40 × 4.096	16 Mio	40,3 Mrd
conv pr	832	1×1	30 × 40 × 4.096	1,0 Mio	2,5 Mrd
conv bb	256	1×1	30 × 40 × 4.096	3,4 Mio	8,2 Mrd
Σ				138 Mio	1.049 Mrd

Tabelle 7.1: Detaillierte Ansicht der VGG-Architektur von [HDTLR](#). Die [Floating Point Operationen \(FLOPs\)](#) enthalten alle Multiplikations- und Additionsoperationen, die in der entsprechenden Schicht ausgeführt werden.

wurde. Die Klassifikation der Objekte und die Regression der Bounding Boxen finden in daran anschließenden, parallelen Convolution-Schichten statt. Als Aktivierungsfunktion wird nach jeder Convolution-Schicht die [Rectified Linear Unit \(ReLU\)](#) verwendet.

Als Evaluationsdatensatz dient der in Abschnitt [3.4.1](#) vorgestellte [HDTLR](#)-Datensatz. Die Metriken, welche zur Evaluation der optimierten Modelle verwendet wurden, sind in Abschnitt [7.4.1](#) genauer beschrieben. Die eigentliche Evaluation der vorgestellten Methodik erfolgt in zwei Etappen. In einem ersten Experiment wurden die in Abschnitt [7.3.1](#) vorgestellten Strategien zum Pruning von Filterkernen vergleichend evaluiert. Die Ergebnisse dieses Experimentes werden in Abschnitt [7.4.2](#) vorgestellt.

Die hieraus resultierende, beste reduzierte Architektur wurde in einem weiteren Experiment als Student Network für den in Abschnitt [7.3.2](#) vorgestellten Ansatz zur Knowledge Distillation verwendet. Als Teacher Network diente hierfür erneut das [HDTLR](#)-Modell. Die Details zu diesem Experiment sind in Abschnitt [7.4.3](#) beschrieben. Zusätzlich wurde abschließend eine vergleichende Evaluation aller erstellten Modelle durchgeführt.

7.4.1 Evaluationsmetriken

Der qualitative Vergleich zwischen den verschiedenen Modellen erfolgt in dieser Evaluation anhand der bereits in Abschnitt [3.4.2](#) im Kontext des ursprünglichen Ampelkennungssystems erläuterten Metriken. Die Laufzeit der Modelle wird ebenfalls analog berechnet und bezieht sich auf die Zeitspanne von Eingabe des Bildes bis zum Vorliegen der finalen Ergebnisse. Hierbei ist zu beachten, dass bei der Auswertung nur Einzelbilder in das [CNN](#) eingegeben werden, die Batch Size also 1 beträgt. Eine größere Batch Size würde das Ergebnis für den Anwendungsfall verfälschen, da hier unter Echtzeitbedingungen nicht mehrere aufeinanderfolgende Bilder zur gleichen Zeit vorliegen. Somit würde die Laufzeit nicht mehr der Latenz zwischen Eingabe des Bildes und Ausgabe des Netzes entsprechen und die Latenz der finalen Anwendung unterschätzt werden. Als Grundlage der in diesem Kapitel genannten Laufzeitangaben dient eine Nvidia GTX 1080 Ti Grafikkarte.

Als zusätzliche Metrik zur Messung der Größe von Architekturen wird die Anzahl der im Netz vorhandenen Parameter verwendet. Hieraus kann der Speicherplatzbedarf des finalen Modells auf der Ausführungseinheit ermittelt werden. Eine genauere Beschreibung hierzu ist in Abschnitt [2.2](#) zu finden.

7.4.2 Reduktion der Netzarchitektur

Ein erster Schritt beim Pruning eines trainierten Modells ist die Festlegung des Vergleichsdatensatzes. Insbesondere die benötigte Anzahl an Bildern ist hierbei

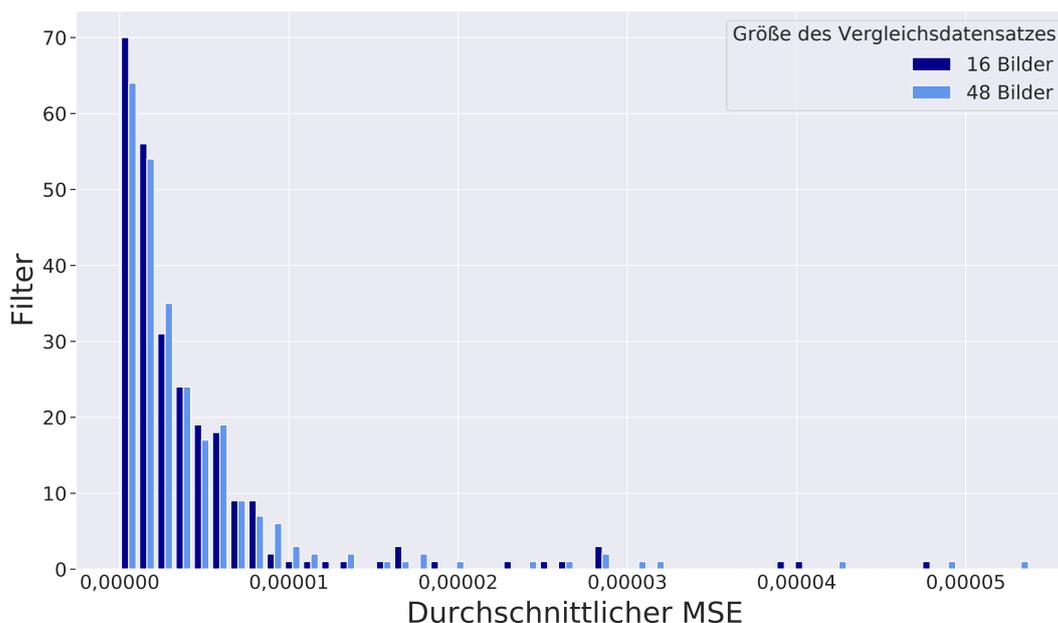


Abbildung 7.5: Veränderung des MSE bei unterschiedlicher Größe des Vergleichsdatensatzes. Ähnlich veröffentlicht in: [37]

eine entscheidende Frage. Da die benötigten Rechenoperationen des Pruning-Verfahrens mit der Anzahl der Bilder linear ansteigen, ist aus Sicht der Durchführungsdauer des Verfahrens ein kleiner Vergleichsdatensatz wünschenswert. Ein größerer Vergleichsdatensatz hingegen würde die Varianz der für die Pruning-Entscheidung betrachteten Szenen erhöhen. Zur Festlegung einer notwendigen Bilderanzahl wurden mehrere Versuche mit Vergleichsdatensätzen unterschiedlicher Größe durchgeführt.

Ein Vergleich der Größen 16 und 48 Bilder ist in Abbildung 7.5 dargestellt. Hierbei wurde die Verteilung des MSE über die einzelnen Filter betrachtet. Die Veränderungen sind hierbei minimal, obwohl ein um den Faktor drei größerer Vergleichsdatensatz verwendet wurde. Bei kleineren Vergleichsdatensätzen wurden deutlich größere Abweichungen festgestellt. Aus diesem Grund wird für die folgenden Versuche ein Vergleichsdatensatz mit 16 Bildern verwendet.

In einer nächsten Versuchsreihe wurden mögliche Ansätze zur Festlegung der Pruning-Rate für jede Schicht untersucht. Hierbei wurde eine Verwendung fester Pruning-Raten mit verschiedenen Methoden zur dynamischen Festlegung von dedizierten Pruning-Raten für jede Schicht verglichen. Zur dynamischen Festlegung wurden ein Ansatz basierend auf dem Fehler des Netzes und ein Ansatz zur Verwendung des MSE verwendet. Basierend auf der Strategie der festen Pruning-Raten für alle Schichten wurde das Pruning mit den Raten 0,4 und 0,5 durchgeführt.

Für die dynamische Festlegung dedizierter Pruning-Raten wurde zunächst ein Verfahren basierend auf dem Fehler des Modells verwendet. Dafür wurde initi-

al für jede Schicht eine Pruning-Rate von 0,5 verwendet. Hiermit wurde Schicht für Schicht ein Pruning ausgeführt und der Fehler des Modells auf den Validierungsdaten gemessen. Lag der Fehler oberhalb des Fehlers des Baseline-Modells, wurde die Pruning-Rate reduziert. Teilweise lag der Fehler auch unterhalb des originalen Fehlers, dann wurde die Rate entsprechend erhöht.

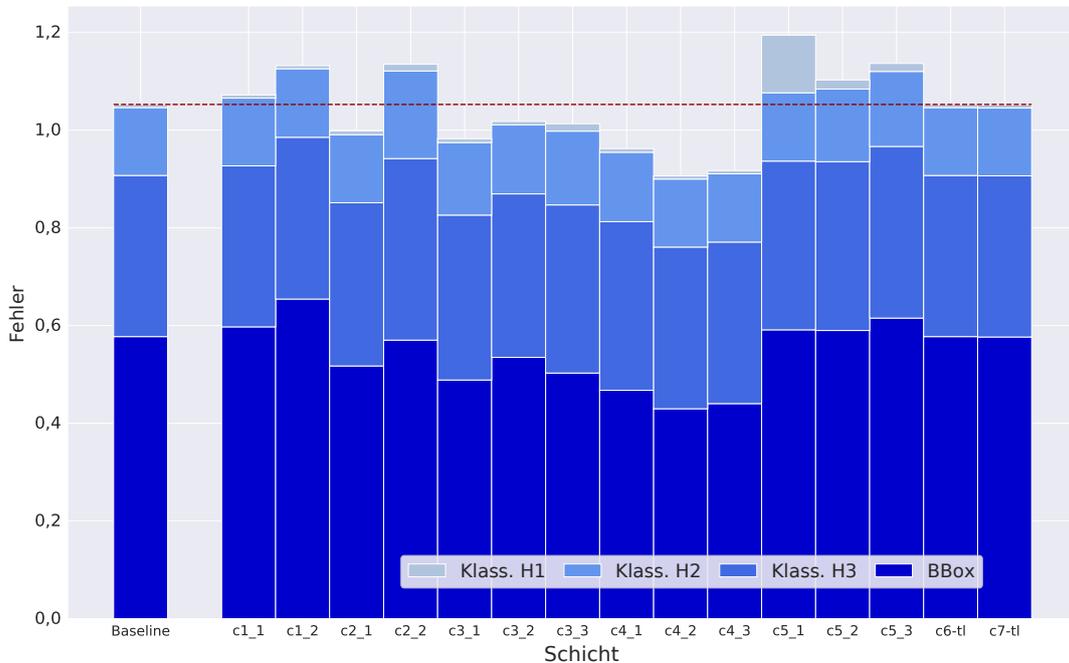


Abbildung 7.6: Änderung des Fehlers bei einer Pruning-Rate von 0,5. Die rote Linie stellt den Fehler des Baseline-Modells dar. Der Fehler setzt sich hierbei zusammen aus dem Fehler jeder der drei Hierarchiestufen und dem Bounding-Box-Fehler. Ähnlich veröffentlicht in: [37]

Die initialen Fehler bei Anwendung des Pruning in den einzelnen Schichten sind in Abbildung 7.6 dargestellt. Finale Pruning-Raten für die einzelnen Schichten sind in Tabelle 7.2 aufgeführt. Es fällt auf, dass für einige Schichten Pruning-Raten von über 0,5 erreicht werden. Aus Erfahrung hat eine Reduzierung der Filterkerne in einer Schicht um mehr als 50% meist eine deutlich negative Auswirkung auf die Qualität des Modells. Aus diesem Grund wurden für das fehlerbasierte Pruning zwei verschiedene Konfigurationen verwendet. Neben den bereits in Tabelle 7.2 aufgeführten Pruning-Raten wurde eine weitere Konfiguration verwendet, bei der die Pruning-Rate bei 0,5 gedeckelt wurde.

Basierend auf dem MSE der einzelnen Filter kann ebenfalls eine Pruning-Entscheidung durchgeführt werden. In einem ersten einfachen Ansatz wurde keine feste Pruning-Rate für jede Schicht definiert. Es wurden lediglich alle Filterkerne entfernt, für die der Mean Squared Error (MSE) bei 0 lag. Mit dieser Methode konnten allerdings nur verhältnismäßig wenige Filterkerne entfernt werden.

Schicht	Rate	Schicht	Rate	Schicht	Rate
conv1_1	0,4	conv3_2	0,5	conv5_1	0,2
conv1_2	0,3	conv3_3	0,5	conv5_2	0,3
conv2_1	0,5	conv4_1	0,6	conv5_3	0,3
conv2_2	0,3	conv4_2	0,7	conv6-tl	0,5
conv3_1	0,5	conv4_3	0,7	conv7-tl	0,5

Tabelle 7.2: Aus der Fehlerbetrachtung resultierende Pruning-Raten für die einzelnen Schichten des Modells.

Aus diesem Grund wird zusätzlich eine spezifische Pruning-Rate pro Schicht basierend auf den summierten *MSE*-Werten der entsprechenden Schicht festgelegt. Diese spezifischen Pruning-Raten sind in Tabelle 7.3 aufgeführt.

Schicht	Rate	Schicht	Rate	Schicht	Rate
conv1_1	0,3	conv3_2	0,5	conv5_1	0,4
conv1_2	0,3	conv3_3	0,5	conv5_2	0,4
conv2_1	0,3	conv4_1	0,5	conv5_3	0,4
conv2_2	0,3	conv4_2	0,5	conv6-tl	0,55
conv3_1	0,35	conv4_3	0,5	conv7-tl	0,5

Tabelle 7.3: Aus der *MSE*-Betrachtung resultierende Pruning-Raten für die einzelnen Schichten des Modells.

Die resultierenden Ergebnisse für die einzelnen durch verschiedene Pruning-Methoden verkleinerten Modelle sind in Tabelle 7.4 aufgeführt. Das statische Pruning mit einer festen Pruning-Rate von 0,4 erreicht bei einer deutlichen Reduktion der Laufzeit einen lediglich 2,5 Prozentpunkte niedrigeren F1-Score. Hier ist allerdings erkennbar, dass bei einer Erhöhung der Pruning-Rate auf 0,5 das Ergebnis deutlich schlechter wird. Die Laufzeit beträgt an dieser Stelle dann noch ein Drittel des ursprünglichen Ansatzes.

Bei der fehlerbasierten Auswahl der Pruning-Rate fällt auf, dass die Ergebnisse lediglich im Bereich der statischen Pruning-Rate von 0,5 anzusiedeln sind. Wie zu erwarten war, wird durch eine Verwendung von Pruning-Raten über 0,5 für einzelne Schichten die Qualität des Ergebnisses nochmals schlechter. Die Beschränkung der Pruning-Rate auf 0,5 reduziert diesen Effekt. Die Laufzeiten der Modelle sind erwartungsgemäß zwischen den Laufzeiten der festen Pruning-Raten von 0,4 und 0,5 anzusiedeln. Auch die mit 50,8 ms leicht höhere Laufzeit des Modells mit der auf 0,5 begrenzten Pruning-Rate lässt sich durch Entfernung einer geringeren Zahl an Filtern erklären.

Die besten Ergebnisse erreicht das Modell, bei dem ausschließlich die Filterkerne mit $MSE = 0$ entfernt wurden. Hier ist erwartungsgemäß lediglich ein geringer Rückgang des F1-Score zu verzeichnen. Da allerdings für dieses Modell eine

Name	TP	FP	FN	FC ₂	FC ₃	F1-Score	Parameter	Laufzeit
Baseline	800	74	42	41	32	85,93	138 Mio	135 ms
Pruning Rate 0,4	783	94	50	41	41	83,56	51 Mio	54,7 ms
Pruning Rate 0,5	754	100	67	39	55	80,94	35 Mio	41,7 ms
Fehlerbasiert	743	116	68	42	62	79,13	46,7 Mio	47,7 ms
Fehlerbasiert 0,5	763	93	57	43	52	81,78	47,7 Mio	50,8 ms
MSE-Threshold 0	798	74	44	41	32	85,81	93,3 Mio	107,9 ms
MSE-Pruning	776	82	60	37	42	83,80	38,2 Mio	47,8 ms

Tabelle 7.4: Vergleich der Ergebnisse für die verschiedenen Verfahren zur Festlegung der Pruning-Raten mit der Baseline. Die angegebene Laufzeit bezieht sich hierbei auf die Verarbeitung eines Einzelbildes.

geringere Menge an Filtern entfernt wurde, ist die Reduktion der Laufzeit auf 108 ms auch relativ gering. Das beste Verhältnis zwischen Reduktion der Laufzeit und Rückgang des F1-Score erreichte in dieser Untersuchung die MSE-basierte Auswahl der spezifischen Pruning-Raten für jede Schicht. So konnte bei einem Rückgang des F1-Score um zwei Prozentpunkte eine Reduktion der Laufzeit um knapp zwei Drittel erreicht werden. Dieses Modell kann nun mit mehr als 20 FPS ausgeführt werden.

7.4.3 Wissensübertragung

Für die Evaluation des in Abschnitt 7.3.2 vorgestellten Ansatzes zur Knowledge Distillation eines hierarchischen Objektdetektors wurde das beste Modell aus dem vorherigen Experiment mit signifikanter Verbesserung der Laufzeit als Student verwendet. Als Teacher diente das ursprüngliche HDTLR-Modell. Die Ergebnisse dieser Evaluation sind in Tabelle 7.5 dargestellt. Ein visualisierter Vergleich dieser Evaluation mit den bereits trainierten Modellen ist in Abbildung 7.7 zu finden.

Name	TP	FP	FN	FC ₂	FC ₃	F1-Score	Laufzeit
Baseline	800	74	42	41	32	85,93	135 ms
KD MSE-based	824	88	32	38	21	87,38	47,8 ms

Tabelle 7.5: Ergebnisse des Trainings mittels Knowledge Distillation. Verglichen wurde das ursprüngliche HDTLR-Modell als Baseline mit dem als Student Network verwendeten MSE-based-Pruning-Modell.

Es ist klar zu erkennen, dass durch die vorgeschlagene Kombination aus Channel Pruning und Knowledge Distillation die bereits im vorherigen Experiment beobachtete Laufzeitreduktion beibehalten werden konnte. Da die Architektur des

7 Optimierung trainierter CNN-Modelle

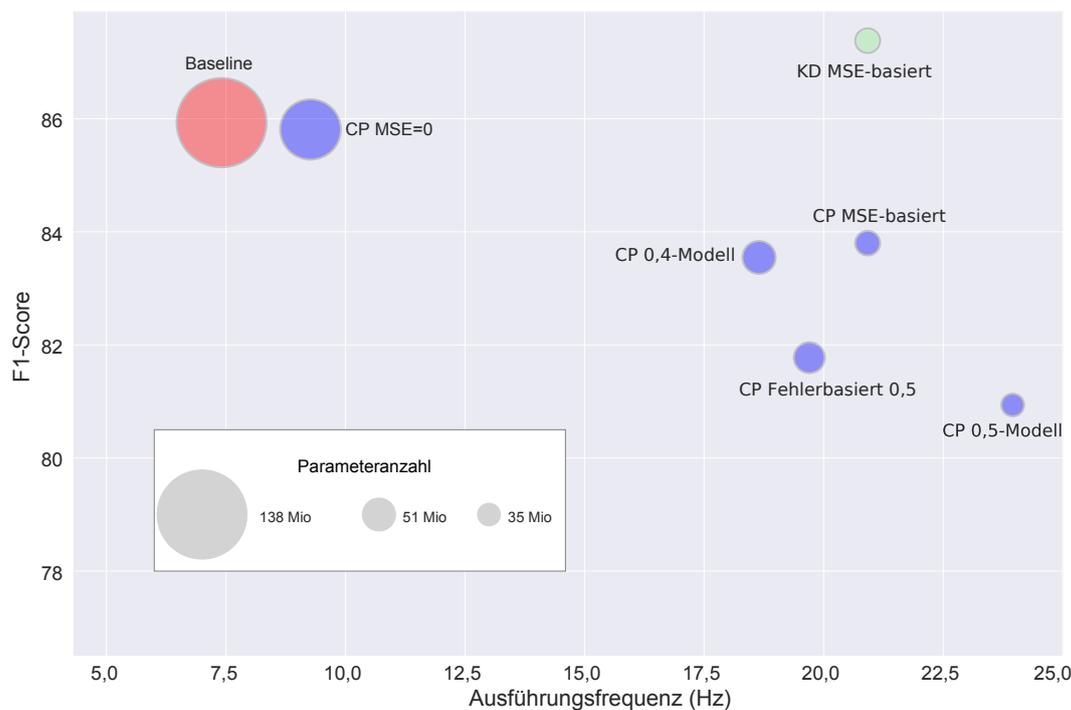


Abbildung 7.7: Ergebnisse des vorgeschlagenen Optimierungsverfahrens (grün) im Vergleich mit den im [Channel Pruning \(CP\)](#) reduzierten Modellen (blau) und dem ursprünglichen [HDTLR-Modell](#) als Baseline (rot). Ähnlich veröffentlicht in: [7]

Netzes für dieses Experiment nicht weiter angepasst wurde, war eine konstante Laufzeit im Vergleich zum letzten Experiment zu erwarten.

Etwas überraschend erscheint zunächst die Tatsache, dass das Student Network bezüglich der Detektionsgüte das Teacher Network übertraf. So erreichte der Student einen um 1,5 Prozentpunkte höheren F1-Score im Vergleich zu dem Teacher. Bei einer leicht erhöhten Anzahl an Fehldetektionen (FP) war die Anzahl der nicht detektierten (FN) oder falsch klassifizierten Ampeln (FC) jeweils deutlich geringer. Die Steigerung der Detektionsgüte ist an dieser Stelle ein positiver Nebeneffekt, bedarf allerdings einer Erklärung. Eine Erklärung ist die bei Furlanello et al. [90] beschriebene Steigerung der Generalisierungsfähigkeit durch das Training eines Student Networks.

7.5 Diskussion

In diesem Kapitel wurde ein Verfahren zur Reduktion der Parameter und damit einer Verringerung der Laufzeit eines [CNN](#)-basierten Modells zur Lösung einer Perzeptionsaufgabe vorgestellt. Dieses auf einer Kombination der Techniken Channel Pruning und Knowledge Distillation basierende Verfahren konnte

in einer Evaluation die Laufzeit des in Kapitel 3 vorgestellten Ansatzes zur hierarchischen Detektion statischer Verkehrsobjekte um den Faktor 2,8 reduzieren. Diese Reduktion der Laufzeit wurde erreicht bei gleichzeitiger Steigerung der Detektionsgüte, was auf eine bessere Generalisierungsfähigkeit des resultierenden Modells hindeutet.

Die verwendeten Methoden können sowohl auf Klassifikations- als auch auf Regressionsprobleme angewendet werden. Eine Anwendung auf Modelle der in Kapitel 5 vorgestellten MultiNet-Architektur ist ebenfalls möglich, wobei hier zwischen den einzelnen Teilen der Architektur differenziert werden muss. Auf die Heads der einzelnen Tasks ist eine Anwendung des vorgestellten Verfahrens ohne Probleme möglich. Der Detektionsteil des in diesem Kapitel verwendeten HDTLR-Modells kann ebenso als Head in einem MultiNet-Modell verwendet werden. Für den gemeinsam verwendeten Encoder bleibt zu klären, ob und in welchem Maße die Kapazität des CNNs weiter reduziert werden kann. Eine Reduktion dieser Kapazität wird mutmaßlich bereits durch die Verwendung eines gemeinsamen Encoders durchgeführt. Bislang wurde allerdings im Rahmen der Forschung zu MultiTask-Netzen nicht untersucht, wie groß diese Reduktion bei der Verwendung verwandter Tasks einer gemeinsamen Domäne tatsächlich ist.

Die im Rahmen dieses Kapitels untersuchten Methoden zur Optimierung eines vorhandenen Modells beschränken sich auf die Optimierung der Netzarchitektur sowie den Wissenstransfer auf diese optimierte Architektur. Die Anwendung weiterer Techniken zur Optimierung trainierter Modelle wie die in den verwandten Arbeiten vorgestellte Quantisierung der Parameterwerte oder einer Reduktion der Genauigkeit von deren Repräsentation wird durch das vorgestellte Verfahren zur Optimierung nicht beschränkt.

8 Zusammenfassung und Ausblick

Tiefe Neuronale Netze leisten in Form von CNNs wichtige Beiträge für die Bildverarbeitung in verschiedenen Domänen. Die Architekturen bereits existierender Netze müssen allerdings teilweise an die Anforderungen der entsprechenden Domäne angepasst werden. Die einzelnen Problemstellungen einer Domäne werden hierbei im Normalfall durch die Verwendung verschiedener CNN-Modelle gelöst. Zur Ausführungszeit findet meist eine isolierte Anwendung dieser Modelle statt. Im Rahmen der vorliegenden Arbeit lag der Fokus auf einer Konzeption spezieller Architekturen sowie der Reduktion des Ressourcenbedarfs durch eine gemeinsame Ausführung verschiedener Netze und einer zusätzlichen Komprimierung netzspezifischer Schichten.

Der betrachtete Anwendungsfall für die durchgeführten Arbeiten war die Informationsgewinnung aus Bildern monokularer Kameras mittels spezifischer Perzeptionsalgorithmen in der Domäne des Autonomen Fahrens. Hierzu wurden vorhandene Architekturen erweitert und um neue Methoden zur Lösung der domänenspezifischen Problemstellungen ergänzt. In diesem Rahmen wurden auch neue Konzepte zum Training der resultierenden Modelle erforscht. Zur Reduktion des Ressourcenbedarfs in der mobilen Anwendungsumgebung eines Autonomen Fahrzeugs wurde ein Konzept zur gemeinsamen Ausführung tiefer Netzarchitekturen vorgestellt, welches auch zur ressourcenschonenden Ausführung multipler Netze auf gemeinsamen Eingabedaten in anderen Domänen eingesetzt werden kann.

8.1 Zusammenfassung der Beiträge

Im Rahmen dieser Arbeit wurden domänenspezifisch insbesondere Architekturen zur Detektion von verschiedenen Objektarten erforscht. Zusätzlich wurden domänenübergreifende Konzepte zur Reduktion des Ressourcenbedarfs durch Ausführung der Modelle mehrerer Tasks in einem gemeinsamen MultiTask-Netzwerk sowie zur Kompression einzelner Schichten vorgestellt. Des Weiteren wurde eine Fehlerfunktion für ein adaptives Training der vorgestellten Architekturen auf nicht ausbalancierten Datensätzen sowie in MultiTask-Szenarien eingeführt.

Zur Gewinnung von Umgebungsinformationen in der Domäne des Autonomen Fahrens wurde der algorithmische Schwerpunkt dieser Arbeit auf die Detektion

von Objekten gelegt. Wie im Rahmen der Evaluation des MultiNet-Ansatzes gezeigt werden konnte, existieren zur Lösung von Tasks basierend auf Segmentierungs- oder Klassifikationsproblemen bereits sehr leistungsfähige Architekturen. Insbesondere für die Detektion von Objekten bestand allerdings durchaus der Bedarf an domänenspezifischen Konzepten. Hierzu wurde das Problem der Objektdetektion zunächst unterteilt in die Detektion statischer Verkehrsobjekte, wie z. B. Ampeln oder Verkehrszeichen, und die Detektion dynamischer Verkehrsobjekte, wie z. B. andere Verkehrsteilnehmer.

Herausforderungen bei der Detektion statischer Verkehrsobjekte waren die größtenteils geringe Größe sowie eine große Ähnlichkeit einer Vielzahl verschiedener Objekte. Zur Lösung dieser Herausforderungen wurde der **HDTLR**-Ansatz zur hierarchischen Objektdetektion vorgestellt. Dieser basiert auf einer hierarchischen Softmax-Klassifikation und ist so in der Lage, eine Vielzahl ähnlicher Klassen zu unterscheiden. Dies konnte im Rahmen einer Evaluation anhand der Detektion von Ampeln verschiedener Zustände mit unterschiedlichen Richtungspiktogrammen gezeigt werden. Der vorgestellte hierarchische **HDTLR**-Ansatz war hierbei in der Lage, eine Vielzahl verschiedener Piktogramme korrekt zu unterscheiden. Dies konnte bislang von keinem anderen Ansatz demonstriert werden.

Für die Detektion dynamischer Verkehrsobjekte wurde mit Direct3D eine Architektur zur direkten **3D** Detektion vorgestellt. So ist eine direkte Verortung des detektierten Objektes sowie eine Übertragung in eine Umgebungskarte in Vogelperspektive möglich. Die direkte **3D** Detektion liefert eine gute Grundlage zur Fusion der Ergebnisse mit den Detektionen anderer Sensortypen. So kann die höhere Genauigkeit räumlicher Sensorik mit der Fähigkeit zur Klassifikation eines visuellen Sensors zur Steigerung der Detektionsqualität kombiniert werden.

Zum erfolgreichen Training von komplexen Architekturen wie dem vorgestellten Detektor für dynamische Objekte wurde der **Automated Focal Loss** eingeführt. Diese Fehlerfunktion dient der Fokussierung des Trainingsprozesses auf zum jeweiligen Zeitpunkt herausfordernde Trainingsbeispiele. Insbesondere bei nicht ausbalancierten Datensätzen sowie Modellen mit unterschiedlichen Ausgaben ist dies eine hilfreiche Eigenschaft. Erst diese Fehlerfunktion ermöglichte ein erfolgreiches Training des Direct3D-Detektors. In einer vergleichenden Evaluation mit dem bisherigen State of the Art konnte gezeigt werden, dass der Automated Focal Loss für das Lernproblem der **2D** Objektdetektion bei gleichen Ergebnissen eine deutlich kürzere Konvergenzdauer des Trainingsprozesses ermöglicht. Auf ein Tuning der Hyperparameter kann hierbei im Vergleich zu anderen Ansätzen verzichtet werden.

Da die parallele Ausführung mehrerer Modelle auf einem gemeinsamen Eingabebild domänenübergreifend eine große Herausforderung insbesondere für mobile Anwendungen darstellt, wurde der MultiNet-Ansatz zur Ausführung verschiedener Tasks in einer gemeinsamen MultiTask-Architektur vorgestellt. Der MultiNet-Ansatz besteht hierbei aus einem geteilten Encoder zur Extraktion gemeinsamer Features und taskspezifischen Decodern mit dedizierten Schichten

zur Erzeugung der Ausgaben für jeden Task. In einer Evaluation konnte gezeigt werden, dass bei der gemeinsamen Ausführung von drei Tasks aus der Domäne des Autonomen Fahrens durch einen geteilten Encoder deutliche Ressourceneinsparungen möglich sind.

Eine weitere Komprimierung der einzelnen Decoder oder von separaten Modellen zur Lösung einzelner Tasks kann durch die vorgestellte Optimierung mit einem kombinierten Einsatz von Pruning und Knowledge Distillation erreicht werden. So kann in einem zweistufigen Prozess zunächst die Architektur reduziert und in einem zweiten Schritt das ursprüngliche Wissen transferiert werden. In einer Evaluation auf Grundlage des [HDTLR](#)-Modells zur Detektion statischer Objekte konnte eine Reduktion der Laufzeit um zwei Drittel bei gleichzeitiger minimaler Steigerung der Generalisierungsfähigkeit des Modells erreicht werden.

8.2 Ausblick

Basierend auf den im Rahmen dieser Arbeit vorgestellten Konzepten ergeben sich sowohl domänenübergreifend als auch spezifisch für die Domäne des Autonomen Fahrens Anschlusspunkte für zukünftige Forschungsarbeiten.

Das vorgestellte Verfahren zur direkten dreidimensionalen Detektion von Objekten stellt bereits eine gute Grundlage für eine Fusion mit anderen, räumlichen Sensoren dar. Bei einer Betrachtung unabhängig von der Domäne des Autonomen Fahrens sind derartige Sensoren allerdings selten gemeinsam mit Bilddaten verfügbar. Vergleicht man die Entwicklung klassischer [2D](#) Objektdetektoren mit der aktuellen Entwicklung der [3D](#) Detektoren, existiert an dieser Stelle noch Potential für weitere Adaptionen. Die höhere Komplexität des Problems der [3D](#) Objektdetektion scheint an dieser Stelle allerdings die Entwicklung zu verlangsamen. Im Rahmen dieser Arbeit wurde deutlich, dass teilweise zunächst neue Konzepte wie der Automated Focal Loss erforscht werden müssen, um erfolgreich eine nächste Entwicklungsstufe erreichen zu können.

Für [CNN](#)-basierte Perzeptionsalgorithmen wurden in dieser Arbeit zwei grundsätzliche Ansätze zur Optimierung im Sinne der Reduktion des Ressourcenverbrauchs vorgestellt. Mit dem MultiNet-Ansatz kann durch einen gemeinsamen Encoder der Ressourcenverbrauch mehrerer Tasks in einem gemeinsamen Modell reduziert werden. Der Verbrauch für die Heads der einzelnen Tasks kann durch die vorgestellte Optimierung bestehend aus Pruning und Knowledge Distillation deutlich reduziert werden. Offen bleibt an dieser Stelle bislang die Frage, ob auch der gemeinsame Encoder durch die vorgestellte Optimierungstechnik weiter reduziert werden kann. Insbesondere für Tasks der gleichen Domäne erscheint hier eine Reduktion durchaus möglich. Generell scheint die Frage von Interesse zu sein, welche Abhängigkeit zwischen dem Optimierungspotential und der konkreten Auswahl der Tasks sowie deren Verbindung in gemeinsamen Domänen besteht.

Betrachtet man den MultiNet-Ansatz im Hinblick auf Anforderungen der Domäne des Autonomen Fahrens oder anderer Domänen, bei denen die Bildverarbeitung basierend auf Videodaten durchgeführt wird, stellt sich auch die Frage nach der Betrachtung zeitlicher Zusammenhänge. Es existieren bereits verschiedene Arbeiten zur Extraktion temporaler Informationen aus Bildfolgen mittels CNN-Architekturen. Allerdings ist die Frage der Integration dieser Methoden zur Extraktion temporaler Informationen in ein MultiTask-Netz bislang offen. Bereits bestehende temporale Encoder für die Objektdetektion [11] bieten an dieser Stelle eine mögliche Grundlage für einen gemeinsamen temporalen Encoder.

Aus der Anwendung im Autonomen Fahren heraus eröffnet sich auch die Frage nach einer Möglichkeit zur Integration verschiedener Sensoren in eine MultiNet-Architektur. An dieser Stelle wären eine multimodale Extraktion von Features oder eine Anbindung der einzelnen Heads an verschiedene Encoder zur sensor-spezifischen Extraktion von Features interessante Forschungsansätze.

Verzeichnisse

Abkürzungsverzeichnis

2D	zweidimensional
3D	dreidimensional
AFL	Automated Focal Loss
ALV	Autonomous Land Vehicle
AOS	Average Orientation Similarity
AP	Average Precision
ASPP	Atrous Spatial Pyramid Pooling
BB	Bounding Box
BiFPN	Bidirectional Feature Pyramid Network
BSTLD	Bosch Small Traffic Lights Datensatz
C2I	Car-to-Infrastructure
CAD	Computer-Aided Design
CEM	Context Enhancement Module
CmBN	Cross mini-Batch Normalization
CNN	Convolutional Neural Network
CoCar	Cognitive Car
COCO	Common Objects in Context
CP	Channel Pruning
CSPNet	Cross Stage Partial Network
DeepTLR	Deep Traffic Light Recognition
DSOD	Deeply Supervised Object Detector
FC	Fully Connected
FC	False Classification
FCN	Fully Convolutional Network
FE	Feature Extraction
FLOPS	Floating Point Operationen pro Sekunde
FLOPs	Floating Point Operationen
FN	False Negative
FP	False Positive
FPN	Feature Pyramid Network
FPS	Frames pro Sekunde
FZI	FZI Forschungszentrum Informatik

Abkürzungsverzeichnis

GB	Gigabyte
GPS	Global Positioning System
GPU	Graphics Processing Unit
GTSRB	German Traffic Sign Recognition Benchmark
HDTLR	Hierarchical Deep Traffic Light Recognition
HOG	Histogram of Oriented Gradients
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoU	Intersection over Union
KD	Knowledge Distillation
KNN	Künstliches Neuronales Netz
LiDAR	Light Detection and Ranging
LSA	Lichtsignalanlage
LZA	Lichtzeichenanlage
mAP	mean Average Precision
MLP	Multi-Layer Perceptron
MS COCO	Microsoft Common Objects in Context
MSE	Mean Squared Error
NAS	Network Architecture Search
NiN	Network in Network
NMS	Non Maximum Suppression
OFT	Orthographic Feature Transform
OHEM	Online Hard Example Mining
OOM	Out of memory
OSM	OpenStreetMap
PAN	Path Aggregation Network
PASCAL VOC	Pattern Analysis Statistical Modelling and Computational Learning Visual Object Classes
PGM	Probabilistische Graphische Modelle
PKW	Personenkraftwagen
PMD	Photonic Mixing Device
PNG	Portable Network Graphics
PR	Precision-Recall
PROMETHEUS	Programme for a European Traffic with Highest Efficiency and Unprecedented Safety
ReLU	Rectified Linear Unit

RiLSA	Richtlinie für Lichtsignalanlagen
RPN	Region Proposal Network
SAM	Spatial Attention Module
SAT	Self-Adversarial Training
SPP	Spatial Pyramid Pooling
SPP-net	Spatial Pyramid Pooling Network
SSD	Single Shot MultiBox Detector
StVO	Straßenverkehrs-Ordnung
StVZO	Straßenverkehrs-Zulassungs-Ordnung
SVM	Support Vector Machine
TN	True Negative
ToF	Time-of-Flight
TP	True Positive
UM	Urban with Lane Markings
UMM	Urban with Multiple Lane Markings
UU	Urban Unmarked
VaMoRs	Versuchsfahrzeug für autonome Mobilität und Rechnersehen
YOLO	You Only Look Once
ÖPNV	Öffentlicher Personennahverkehr

Abbildungsverzeichnis

1.1	Beispielhafte Betrachtung einer Verkehrsszene durch eine monokulare Kamera.	3
1.2	Struktur und Aufbau der vorliegenden Arbeit.	5
2.1	Inception-Modul aus dem GoogLeNet. Bildquelle: [222]	9
3.1	Ampelerkennung mit Zustand und Piktogramm.	15
3.2	Ampelkreuzung aus Vogelperspektive.	19
3.3	Aufbau der DeepTLR -Architektur.	35
3.4	Regionsraster eines CNN-Detektors.	37
3.5	Erzeugung der Ergebnisse im DeepTLR -Ampeldetektor.	39
3.6	Detaillierte Architektur von DeepTLR	40
3.7	Detektion ähnlicher Objekte.	41
3.8	Detaillierte Architektur des HDTLR -Ansatzes.	42
3.9	Beispielhafte Ausgaben von HDTLR	44
3.10	Beispiele des DeepTLR -Datensatzes.	50
3.11	Verteilung der Ampeln im HDTLR -Datensatz.	52
3.12	Die Intersection over Union (IoU) -Metrik.	56
3.13	Die bei einer Klassifikation verwendeten Maßzahlen.	57
3.14	Hierarchie für das Lernproblem der Ampelerkennung.	62
3.15	Analyse der Breite von Fehlklassifikationen.	65
3.16	Güte auf dem BSTLD -Datensatz.	66
3.17	Vergleich zwischen HDTLR und BSTLD -Baseline.	67
3.18	Gruppierung ähnlicher Objekte.	68
4.1	3D Detektion dynamischer Verkehrsobjekte.	71
4.2	Aufbau des Netzes zur 3D Objektdetektion.	78

4.3	Schätzung der 3D Bounding Box.	79
4.4	Aufbau des Decoders zur 3D Objektdetektion.	81
4.5	Vergleich zwischen Direct3D und OverFeat 3D.	84
4.6	Aufbau des SSD-Detektors zur 3D Objektdetektion.	85
4.7	Aufbau des RetinaNet-Detektors zur 3D Objektdetektion.	86
4.8	Verlauf des MultiLoss.	87
4.9	Ausgaben des Detektornetzes.	88
4.10	Häufige Fehler des Direct3D-Systems.	93
5.1	Ergebnis eines MultiNet-Modells.	95
5.2	Getrennte Ausführung mehrerer CNNs zur Umgebungswahrnehmung.	97
5.3	Vergleich zwischen Hard und Soft Parameter Sharing.	100
5.4	Copy Gate zur Anbindung der MultiNet-Heads.	105
5.5	CNN-Architektur nach dem MultiNet-Prinzip.	106
5.6	Beispieldaten aus dem KITTI Road Detection Datensatz.	110
5.7	Beispieldaten aus dem Datensatz zur Szenenklassifikation.	113
5.8	Architektur des zur Evaluation verwendeten MultiNet.	115
5.9	Architektur des Head zur Segmentierung.	116
5.10	Architektur des Head zur Detektion von Fahrzeugen.	117
5.11	Beispielausgabe der Detektion aus dem MultiNet-Modell.	118
5.12	Architektur des Head zur Klassifikation.	119
5.13	Beispiele für die Ergebnisse des MultiNet-Modells.	124
6.1	Gewichtung einzelner Datenpunkte.	127
6.2	Entscheidungsgrenze eines einfachen Klassifikationsproblems.	129
6.3	Verläufe des Fehlers beim Focal Loss.	135
6.4	Verschiedene Verläufe des Gewichtungsfaktors w	137
6.5	Die Bestimmung von γ anhand der Information.	139
6.6	Gewichtungsfaktor w in Abhängigkeit fester Werte für γ	139
7.1	Optimierung eines CNN-Modells durch Knowledge Distillation.	147

7.2	Durchführung von Channel Pruning.	152
7.3	Vorgeschlagenes Verfahren zur Optimierung.	156
7.4	Pruning der einzelnen Filterkerne.	157
7.5	Veränderung des MSE bzgl. Datensatzgröße.	162
7.6	Änderung des Fehlers bei einer Pruning-Rate von 0,5.	163
7.7	Vergleich der Ergebnisse.	166

Tabellenverzeichnis

2.1	Vergleich verschiedener Netzarchitekturen zur Extraktion von Features.	12
3.1	Vorgegebene Mindestsichtbarkeit für Lichtsignalanlagen in Deutschland nach RiLSA [83].	20
3.2	Verbleibende Reaktionszeit für ein System zur Ampelerkennung gegeben verschiedene Geschwindigkeiten v und Bremsstärken. . .	21
3.3	Architektur des DeepTLR -Ansatzes zur CNN -basierten Detektion von Ampeln.	36
3.4	Vergleich verschiedener Datensätze zur Ampelerkennung.	49
3.5	Klassenverteilung der Ampeln im DeepTLR -Datensatz.	51
3.6	Verteilung der Ampelklassen auf die Teildatensätze von DeepTLR . . .	51
3.7	Aufteilung des HDTLR -Datensatzes.	52
3.8	Größe der Ampeln im BSTLD -Testdatensatz in Pixeln.	53
3.9	Verteilung der Ampeln im BSTLD auf die verschiedenen Klassen. . .	54
3.10	Verteilung der Ampelklassen im BSTLD	55
3.11	Ergebnisse von DeepTLR	60
3.12	Konfusionsmatrix für die Mehrklassendetektion von DeepTLR . . .	61
3.13	Vergleich von DeepTLR mit dem hierarchischen HDTLR	62
3.14	Evaluation von HDTLR mit verschiedenen Feature Extraktoren. . .	63
3.15	Laufzeiten verschiedener Encodern auf dem HDTLR -Datensatz. . .	64
3.16	Ergebnisse von HDTLR auf dem BSTLD -Datensatz.	66
4.1	Architektur des 3D Decoders.	83
4.2	Ergebnisse auf dem KITTI 3D Object Detection Datensatz.	91
5.1	Verteilung der einzelnen Klassen im Trainingsdatensatz des KITTI 2D Object Detection Benchmark.	111

TABELLENVERZEICHNIS

5.2	Charakterisierung der einzelnen im KITTI 2D Object Detection Datensatz definierten Schwierigkeitslevel.	112
5.3	Klassendefinition und Verteilung der Daten des Datensatzes zur Szenenklassifikation.	114
5.4	Detaillierte Ansicht der VGG-Architektur von MultiNet.	120
5.5	Ergebnisse bei gemeinsamer Ausführung.	122
5.6	Ergebnisse bei getrennter Ausführung.	122
5.7	Laufzeiten der verschiedenen Modelle.	123
6.1	Vergleich von Focal Loss und Automated Focal Loss.	144
6.2	Ergebnisse der Untersuchung verschiedener Fehlerfunktionen. . .	145
7.1	Detaillierte Ansicht der VGG-Architektur von HDTLR	160
7.2	Aus der Fehlerbetrachtung resultierende Pruning-Raten für die einzelnen Schichten des Modells.	164
7.3	Aus der MSE -Betrachtung resultierende Pruning-Raten für die einzelnen Schichten des Modells.	164
7.4	Vergleich der Ergebnisse für die verschiedenen Verfahren zur Festlegung der Pruning-Raten.	165
7.5	Ergebnisse des Trainings mittels Knowledge Distillation.	165

Eigene Veröffentlichungen

Dieses Verzeichnis listet alle Publikationen, bei denen der Autor dieser Dissertation entweder der Erstautor ist, oder als Co-Autor maßgeblich zu der Veröffentlichung beigetragen hat (in Form von Problemstellung, -lösung, Diskussion oder experimenteller Evaluation).

- [1] Sebastian Bittel, Timo Rehfeld, Michael Weber, and J. Marius Zöllner. Estimating High Definition Map Parameters with Convolutional Neural Networks. In *International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, 2017.
- [2] Nils Gählert, Jun-Jun Wan, Michael Weber, J. Marius Zöllner, Uwe Franke, and Joachim Denzler. Beyond Bounding Boxes: Using Bounding Shapes for Real-Time 3D Vehicle Detection from Monocular RGB Images. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2019.
- [3] Svetlana Pavlitskaya, Christian Hubschneider, Michael Weber, Ruby Moritz, Fabian Huger, Peter Schlicht, and J. Marius Zöllner. Using Mixture of Expert Models to Gain Insights into Semantic Segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR) - Workshops*. IEEE, 2020.
- [4] Florian Piewak, Timo Rehfeld, Michael Weber, and J. Marius Zöllner. Fully Convolutional Neural Networks for Dynamic Object Detection in Grid Maps. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2017.
- [5] Marvin Teichmann, Michael Weber, J. Marius Zöllner, Roberto Cipolla, and Raquel Urtasun. MultiNet: Real-time Joint Semantic Reasoning for Autonomous Driving. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2018.
- [6] Michael Weber, Michael Fürst, and J. Marius Zöllner. Direct 3D Detection of Vehicles in Monocular Images with a CNN based 3D Decoder. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2019.
- [7] Michael Weber, Michael Fürst, and J. Marius Zöllner. Automated Focal Loss for Image based Object Detection. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2020.
- [8] Michael Weber, Matthias Huber, and J. Marius Zöllner. HDTLR: A CNN based Hierarchical Detector for Traffic Lights. In *International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018.

- [9] Michael Weber, Christoph Rist, and J. Marius Zöllner. Learning Temporal Features with CNNs for Monocular Visual Ego Motion Estimation. In *International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2017.
- [10] Michael Weber, Lucas Volkert, Christian Hubschneider, and J. Marius Zöllner. CNN based Multi-View Object Detection and Association. In *International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2019.
- [11] Michael Weber, Tassilo Wald, and J. Marius Zöllner. Temporal Feature Networks for CNN based Object Detection. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2021.
- [12] Michael Weber, Christof Wendenius, and J. Marius Zöllner. Runtime Optimization of a CNN Model for Environment Perception. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2020.
- [13] Michael Weber, Peter Wolf, and J. Marius Zöllner. DeepTLR: A single Deep Convolutional Network for Detection and Classification of Traffic Lights. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2016.
- [14] Peter Wolf, Christian Hubschneider, Michael Weber, André Bauer, Jonathan Härtl, Fabian Dürr, and J. Marius Zöllner. Learning How to Drive in a Real World Simulation with Deep Q-Networks. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2017.

Studentische Arbeiten

Dieses Verzeichnis listet studentische Arbeiten, die durch den Autor dieser Dissertation im Rahmen seiner Forschung betreut wurden. Dies beinhaltet die maßgebliche Vorgabe der Problemstellung, Diskussion der Arbeit, sowie Randvorgaben zur Lösung, Visualisierung und experimentellen Evaluation.

- [15] Selcuk Aklanoglu. Domain Adaptation with GANs for Improving Road Scene Reconstruction. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2018.
- [16] André Bauer. Advanced Lateral Control of an Autonomous Vehicle with Deep Neural Networks. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2017.
- [17] Sebastian Bayer. Augmenting End-to-End Vehicle Control Networks with Instance Segmentation. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2018.
- [18] Sebastian Bittel. Estimating Parameters of High Definition Maps with Convolutional Neural Networks. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2017.
- [19] Michael Fürst. Detektion dynamischer Objekte mittels Convolutional Neural Networks. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2018.
- [20] Matthias Huber. Hierarchische Detektion von Ampeln mittels Convolutional Neural Networks. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2017.
- [21] Alexander Kruck. Predicting Red Light Durations of Adaptive Traffic Lights Based on Floating Car Data. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2017.
- [22] Alexandru Lesi. Learning from depth images for monocular object detection with convolutional neural networks. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2017.
- [23] Sebastian E. C. Martin. Development of a traffic light detection pipeline with neural networks for integration in autonomous vehicles. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2021.

- [24] Christian Ostapczuk. Kombiniertes Lernen eines vereinheitlichten CNN zur visuellen Umfelderkennung im autonomen Fahrzeug. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2020.
- [25] Florian Piewak. Fully Convolutional Neural Networks for Dynamic Object Detection in Grid Maps. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2016.
- [26] Bernardo Pires. Object Detection using Temporal Information and Convolutional Neural Networks. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2016.
- [27] Nikolai Polley. Adversarial Attacks on Temporal Fusion Neural Networks. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2021.
- [28] Rupert Polley. Improving Monocular 3D Object Detection by Implicitly Learned Depth Information. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2021.
- [29] Christoph Rist. Lernen temporaler Features zur visuellen Ego-Bewegungsschätzung mit einem Convolutional Neural Network. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2017.
- [30] Marvin Teichmann. Visual scene understanding in autonomous driving using one unified deep learning model. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2016.
- [31] Martin Thoma. Analysis and Optimization of Convolutional Neural Network Architectures. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2017.
- [32] Phillip Thomas. Visuelle Segmentierung von Umgebungsbildern zur Optimierung einer Lokalisierung basierend auf Punktwolken. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2017.
- [33] Lucas Volkert. Multiperspektivische Objektdetektion und -assoziation mittels Convolutional Neural Networks. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2019.
- [34] Tassilo Wald. Kombination von temporaler und räumlicher Informationsgewinnung in einem CNN zur Steigerung der Güte eines Objektdetektors. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2019.
- [35] Jun-Jun Wan. Real-Time 3D Vehicle Detection from Monocular RGB Images. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2019.
- [36] Felix Weller. Visualisierung des internen Status tiefer neuronaler Netze. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2016.
- [37] Christof Wendenius. Laufzeitoptimierung tiefer Neuronaler Netze im Anwendungsfeld autonomes Fahren. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2019.

- [38] Sebastian Werling. Steigerung der Robustheit von Ampelerkennungssystemen durch Verwendung vorhandener Umgebungsinformationen. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2015.
- [39] Peter Wolf. Detektion von Ampeln mittels Convolutional Neural Networks. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2016.

Literaturverzeichnis

- [40] Rangachari Anand, Kishan G. Mehrotra, Chilukuri K. Mohan, and Sanjay Ranka. An Improved Algorithm for Neural Network Classification of Imbalanced Training Sets. *Transactions on Neural Networks (TNN)*, 4(6):962–969, 1993.
- [41] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured Pruning of Deep Convolutional Neural Networks. *Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32, 2017.
- [42] Martin Bach, Stephan Reuter, and Klaus Dietmayer. Multi-Camera Traffic Light Recognition Using a Classifying Labeled Multi-Bernoulli Filter. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2017.
- [43] Dan Barnes, Will Maddern, and Ingmar Posner. Exploiting 3D Semantic Scene Priors for Online Traffic Light Interpretation. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2015.
- [44] Karsten Behrendt, Libor Novak, and Rami Botros. A Deep Learning Approach to Traffic Lights: Detection, Tracking, and Classification. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.
- [45] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum Learning. In *International Conference on Machine Learning (ICML)*, 2009.
- [46] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. *CoRR*, abs/2004.10934, 2020.
- [47] Bert Breuer and Karlheinz H. Bill. *Bremsenhandbuch*. Springer Vieweg, 2017.
- [48] Heiner Bubb, Klaus Bengler, Heiner E. Grünen, and Marc Vollrath. *Automobilergonomie*. Springer Vieweg, 2015.
- [49] Cristian Buciluă, Rich Caruana, and Alexandru Niculescu-Mizil. Model Compression. In *Conference on Knowledge Discovery and Data Mining (SIGKDD)*. ACM, 2006.
- [50] Samuel Rota Buló, Gerhard Neuhold, and Peter Kotschieder. Loss Max-Pooling for Semantic Image Segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

- [51] Rich Caruana. Multitask Learning: A Knowledge-Based Source of Inductive Bias. In *International Conference on Machine Learning (ICML)*, 1993.
- [52] Rich Caruana. Multitask Learning. In *Learning to Learn*, pages 95–133. Springer, 1998.
- [53] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble Selection from Libraries of Models. In *International Conference on Machine Learning (ICML)*, 2004.
- [54] Florian Chabot, Mohamed Chaouch, Jaonary Rabarisoa, Céline Teulière, and Thierry Chateau. Deep MANTA: A Coarse-to-fine Many-Task Network for joint 2D and 3D vehicle analysis from monocular image. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.
- [55] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. *CoRR*, abs/1706.05587, 2017.
- [56] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3D Object Detection for Autonomous Driving. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.
- [57] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G. Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3D Object Proposals for Accurate Object Class Detection. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.
- [58] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-View 3D Object Detection Network for Autonomous Driving. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.
- [59] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks. In *International Conference on Machine Learning (ICML)*, 2018.
- [60] Zhe Chen and Zijing Chen. RBNet: A Deep Neural Network for Unified Road and Road Boundary Detection. In *International Conference on Neural Information Processing (ICONIP)*. Springer, 2017.
- [61] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient Primitives for Deep Learning. *CoRR*, abs/1410.0759, 2014.
- [62] François Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.
- [63] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Low Precision Arithmetic for Deep Learning. In *International Conference on Learning Representations (ICLR)*, 2014.

- [64] Michael Crawshaw. Multi-Task Learning with Deep Neural Networks: A Survey. *CoRR*, abs/2009.09796, 2020.
- [65] Navneet Dalal and Bill Triggs. Histograms of Oriented Gradients for Human Detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2005.
- [66] Raoul De Charette and Fawzi Nashashibi. Real Time Visual Traffic Lights Recognition Based on Spot Light Detection and Adaptive Traffic Lights Templates. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2009.
- [67] Raoul De Charette and Fawzi Nashashibi. Traffic Light Recognition using Image Processing Compared to Learning Processes. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2009.
- [68] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. Predicting Parameters in Deep Learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.
- [69] Moises Diaz, Pietro Cerri, Giuseppe Pirlo, Miguel A. Ferrer, and Donato Impedovo. A Survey on Traffic Light Detection. In *International Conference on Image Analysis and Processing - Workshops*. Springer, 2015.
- [70] Ernst Dickmanns, Reinhold Behringer, Dirk Dickmanns, Thomas Hildebrandt, Markus Maurer, Frank Thomanek, and Joachim Schiehlen. The Seeing Passenger Car 'VaMoRs-P'. In *Intelligent Vehicles Symposium (IV)*. IEEE, 1994.
- [71] Ernst Dickmanns and Volker Graefe. Applications of Dynamic Monocular Machine Vision. *Machine Vision and Applications (MVA)*, 1(4):241–261, 1988.
- [72] Ernst Dickmanns and Alfred Zapp. Autonomous High Speed Road Vehicle Guidance by Computer Vision. *IFAC Proceedings Volumes*, 20(5):221–226, 1987.
- [73] Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. Low Resource Dependency Parsing: Cross-lingual Parameter Sharing in a Neural Network Parser. In *Annual Meeting of the Association for Computational Linguistics and International Joint Conference on Natural Language Processing*. The Association for Computer Linguistics, 2015.
- [74] United Nations Economic and Social Council. *Vienna Convention on Road Signs and Signals*. United Nations Economic Commission for Europe, 2006.
- [75] David Eigen and Rob Fergus. Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture. In *International Conference on Computer Vision (ICCV)*. IEEE, 2015.
- [76] David Eigen, Christian Puhersch, and Rob Fergus. Depth Map Prediction from a Single Image using a Multi-Scale Deep Network. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

- [77] Lukas Enderich, Fabian Timm, and Wolfram Burgard. Holistic Filter Pruning for Efficient Deep Neural Networks. In *Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2021.
- [78] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable Object Detection using Deep Neural Networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2014.
- [79] Stéphane Estable, J. Schick, F. Stein, Reihard Janssen, R. Ott, Werner Ritter, and Y.-J. Zheng. A Real-Time Traffic Sign Recognition System. In *Intelligent Vehicles Symposium (IV)*. IEEE, 1994.
- [80] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The PASCAL Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.
- [81] Nathaniel Fairfield and Chris Urmson. Traffic Light Mapping and Detection. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2011.
- [82] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object Detection with Discriminatively Trained Part Based Models. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(9):1627–1645, 2010.
- [83] Forschungsgesellschaft für Straßen- und Verkehrswesen Arbeitsgruppe Verkehrsmanagement, Forschungsgesellschaft für Straßen- und Verkehrswesen Arbeitsausschuss Verkehrsbeeinflussung Innerorts. *Richtlinien für Lichtsignalanlagen RiLSA; Lichtzeichenanlagen für den Straßenverkehr*. FGSV-Verlag, 2010.
- [84] Uwe Franke, Dariu Gavrila, Steffen Görzig, Frank Lindner, Frank Paetzold, and Christian Wöhler. Autonomous Driving approaches Downtown. *IEEE Intelligent Systems & Their Applications*, 13(6):40–48, 1998.
- [85] Uwe Franke, Dariu Gavrila, Steffen Görzig, Frank Lindner, Frank Paetzold, and Christian Wöhler. Bildverstehen im innerstädtischen Verkehr. In *Autonome Mobile Systeme 1998*. Springer, 1999.
- [86] Uwe Franke, David Pfeiffer, Clemens Rabe, Carsten Knoeppel, Markus Enzweiler, Fridtjof Stein, and Ralf G. Herrtwich. Making Bertha See. In *International Conference on Computer Vision (ICCV) - Workshops*. IEEE, 2013.
- [87] Andreas Fregin, Julian Müller, Ulrich Kreßel, and Klaus Dietmayer. The DriveU Traffic Light Dataset: Introduction and Comparison with Existing Datasets. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2018.
- [88] Yoav Freund and Robert E Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

- [89] Jannik Fritsch, Tobias Kuehnl, and Andreas Geiger. A New Performance Measure and Evaluation Benchmark for Road Detection Algorithms. In *International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2013.
- [90] Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born-Again Neural Networks. In *International Conference on Machine Learning (ICML)*. PMLR, 2018.
- [91] Nils Gähler, Marina Mayer, Lukas Schneider, Uwe Franke, and Joachim Denzler. MB-Net: MergeBoxes for Real-Time 3D Vehicles Detection. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2018.
- [92] Reinhard Gämlich and Werner Ritter. A Knowledge Based System for Traffic Sign Recognition. In *Mustererkennung 1990*, pages 82–89. Springer, 1990.
- [93] Yuan Gao, Jiayi Ma, Mingbo Zhao, Wei Liu, and Alan L. Yuille. NDDR-CNN: Layerwise Feature Fusing in Multi-Task CNNs by Neural Discriminative Dimensionality Reduction. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019.
- [94] Dariu Gavrila, Uwe Franke, Christian Wohler, and Steffen Gorzig. Real Time Vision for Intelligent Vehicles. *IEEE Instrumentation & Measurement Magazine*, 4(2):22–27, 2001.
- [95] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)*, 2013.
- [96] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012.
- [97] David Geronimo, Antonio M. Lopez, Angel D. Sappa, and Thorsten Graf. Survey of Pedestrian Detection for Advanced Driver Assistance Systems. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(7):1239–1258, 2009.
- [98] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. DropBlock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.
- [99] Ross Girshick. Fast R-CNN. In *International Conference on Computer Vision (ICCV)*. IEEE, 2015.
- [100] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2014.
- [101] Andres E. Gomez, Francisco A. R. Alencar, Paulo V. Prado, Fernando S. Osorio, and Denis F. Wolf. Traffic Lights Detection and State Estimation Using Hidden Markov Models. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2014.

- [102] Jianwei Gong, Yanhua Jiang, Guangming Xiong, Chaohua Guan, Gang Tao, and Huiyan Chen. The Recognition and Tracking of Traffic Lights Based on Color Segmentation and CAMSHIFT for Intelligent Vehicles. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2010.
- [103] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing Deep Convolutional Networks using Vector Quantization. *CoRR*, abs/1412.6115, 2014.
- [104] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [105] Jianping Gou, Baosheng Yu, Stephen John Maybank, and Dacheng Tao. Knowledge Distillation: A Survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.
- [106] Hongyu Guo and Herna L. Viktor. Learning from Imbalanced Data Sets with Boosting and Data Generation: The DataBoost-IM Approach. *ACM SIGKDD Explorations Newsletter*, 6(1):30–39, 2004.
- [107] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep Learning with Limited Numerical Precision. In *International Conference on Machine Learning (ICML)*, 2015.
- [108] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations (ICLR)*, 2016.
- [109] Song Han, Jeff Pool, John Tran, and William Dally. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.
- [110] Haibo He and Eduardo A. Garcia. Learning from imbalanced data. *Transactions on Knowledge and Data Engineering (TKDE)*, 21(9):1263–1284, 2009.
- [111] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *International Conference on Computer Vision (ICCV)*. IEEE, 2017.
- [112] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *International Conference on Computer Vision (ICCV)*. IEEE, 2015.
- [113] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 37(9):1904–1916, 2015.
- [114] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.

- [115] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.
- [116] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diplomarbeit, Technische Universität München, 1991.
- [117] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, et al. Searching for MobileNetV3. In *International Conference on Computer Vision (ICCV)*. IEEE, 2019.
- [118] Andrew Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR*, abs/1704.04861, 2017.
- [119] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-Excitation Networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018.
- [120] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.
- [121] Peter J. Huber. Robust Estimation of a Location Parameter. *The annals of mathematical statistics*, 35(1):73–101, 1964.
- [122] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, et al. An Empirical Evaluation of Deep Learning on Highway Driving. *CoRR*, abs/1504.01716, 2015.
- [123] Forrest Iandola, Song Han, Matthew Moskewicz, Khalid Ashraf, William Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size. *CoRR*, abs/1602.07360, 2016.
- [124] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning (ICML)*, 2015.
- [125] Paul Jaccard. Lois de distribution florale dans la zone alpine. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 38:69–130, 1902.
- [126] Chulhoon Jang, Chansoo Kim, Dongchul Kim, Minchae Lee, and Myoung-ho Sunwoo. Multiple Exposure Images based Traffic Light Recognition. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2014.
- [127] Reinhard Janssen, Werner Ritter, F. Stein, and S. Ott. Hybrid Approach for Traffic Sign Recognition. In *Intelligent Vehicles Symposium (IV)*. IEEE, 1993.
- [128] Nathalie Japkowicz and Shaju Stephen. The Class Imbalance Problem: A Systematic Study. *Intelligent Data Analysis*, 6(5):429–449, 2002.

- [129] Morten B. Jensen, Kamal Nasrollahi, and Thomas B. Moeslund. Evaluating State-of-the-art Object Detector on Challenging Traffic Light Data. In *Conference on Computer Vision and Pattern Recognition (CVPR) - Workshops*. IEEE, 2017.
- [130] Morten B. Jensen, Mark P. Philipsen, Andreas Møgelmoose, Thomas B. Moeslund, and Mohan M. Trivedi. Vision for Looking at Traffic Lights: Issues, Survey, and Perspectives. *Transactions on Intelligent Transportation Systems (T-ITS)*, 17(7):1800–1815, 2016.
- [131] Vijay John, Keisuke Yoneda, Zheng Liu, and Seiichi Mita. Saliency Map Generation by the Convolutional Neural Network for Real-Time Traffic Light Detection Using Template Matching. *Transactions on Computational Imaging*, 1(3):159–173, 2015.
- [132] Vijay John, Keisuke Yoneda, B. Qi, Zheng Liu, and Seiichi Mita. Traffic Light Recognition in Varying Illumination using Deep Learning and Saliency Map. In *International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2014.
- [133] Takeo Kanade, Chuck Thorpe, and William Whittaker. Autonomous Land Vehicle Project at CMU. In *Conference on Computer Science (CSC)*. ACM, 1986.
- [134] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale Video Classification with Convolutional Neural Networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2014.
- [135] Alex G. Kendall, Yarin Gal, and Roberto Cipolla. Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018.
- [136] Hyun-Koo Kim, Ju H. Park, and Ho-Youl Jung. Effective Traffic Lights Recognition Method for Real Time Driving Assistance System in the Daytime. *International Journal of Electrical and Computer Engineering*, 5(11):1429–1432, 2011.
- [137] Hyun-Koo Kim, Ju H. Park, and Ho-Youl Jung. An Efficient Color Space for Deep-Learning Based Traffic Light Recognition. *Journal of Advanced Transportation*, 2018, 2018.
- [138] Hyun-Koo Kim, Young-Nam Shin, Sa-gong Kuk, Ju H. Park, and Ho-Youl Jung. Night-Time Traffic Light Detection Based On SVM with Geometric Moment Features. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 7(4):472–475, 2013.
- [139] Hyun-Koo Kim, Kook-Yeol Yoo, Ju H Park, and Ho-Youl Jung. Traffic Light Recognition Based on Binary Semantic Segmentation Network. *Sensors*, 19(7), 2019.

- [140] Diederik P. Kingma and Jimmy Ba. ADAM: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, 2015.
- [141] Ralf Kohlhaas, Thomas Schamm, Dominik Lenk, and J. Marius Zöllner. Towards driving autonomously: Autonomous cruise control in urban environments. In *Intelligent Vehicles Symposium (IV) - Workshops*. IEEE, 2013.
- [142] Iasonas Kokkinos. UberNet: Training a ‘Universal’ Convolutional Neural Network for Low-, Mid-, and High-Level Vision using Diverse Datasets and Limited Memory. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.
- [143] Anita Krishnakumar. Active Learning Literature Survey. Technical report, University of California, Santa Cruz, 2007.
- [144] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.
- [145] M. Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-Paced Learning for Latent Variable Models. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.
- [146] Sergio Lafuente-Arroyo, Pedro Gil-Jimenez, R. Maldonado-Bascon, Francisco López-Ferreras, and Saturnino Maldonado-Bascon. Traffic sign shape classification evaluation I: SVM using Distance to Borders. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2005.
- [147] Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural computation*, 1(4):541–551, 1989.
- [148] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [149] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal Brain Damage. In *Advances in Neural Information Processing Systems (NIPS)*, 1990.
- [150] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle Detection from 3D Lidar Using Fully Convolutional Network. In *Robotics: Science and Systems*, 2016.
- [151] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. In *International Conference on Learning Representations (ICLR)*, 2014.
- [152] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

- [153] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal Loss for Dense Object Detection. In *International Conference on Computer Vision (ICCV)*. IEEE, 2017.
- [154] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *European Conference on Computer Vision (ECCV)*. Springer, 2014.
- [155] Frank Lindner, Ulrich Kressel, and Stephan Kaelberer. Robust Recognition of Traffic Signals. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2004.
- [156] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path Aggregation Network for Instance Segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018.
- [157] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.
- [158] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015.
- [159] Kuo-Hao Lu, Chao-Ming Wang, and Shu-Yuan Chen. Traffic Light Recognition. *Journal of the Chinese institute of engineers*, 31(6):1069–1075, 2008.
- [160] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. In *International Conference on Computer Vision (ICCV)*. IEEE, 2017.
- [161] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *European Conference on Computer Vision (ECCV)*. Springer, 2018.
- [162] Wei-Chiu Ma, Shenlong Wang, Marcus A. Brubaker, Sanja Fidler, and Raquel Urtasun. Find Your Way by Observing the Sun and Other Semantic Cues. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.
- [163] Diganta Misra. Mish: A Self Regularized Non-Monotonic Activation Function. In *British Machine Vision Conference*. BMVA Press, 2020.
- [164] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch Networks for Multi-task Learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.
- [165] Roozbeh Mottaghi, Yu Xiang, and Silvio Savarese. A Coarse-to-Fine Model for 3D Pose Estimation and Sub-category Recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015.

- [166] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Košecká. 3D Bounding Box Estimation Using Deep Learning and Geometry. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.
- [167] Julian Müller and Klaus Dietmayer. Detecting Traffic Lights by Single Shot Detection. In *International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018.
- [168] Alexander Neubeck and Luc Van Gool. Efficient Non-Maximum Suppression. In *International Conference on Pattern Recognition (ICPR)*, 2006.
- [169] Daniel Neumann, Tobias Langner, Fritz Ulbrich, Dorothee Spitta, and Daniel Goehring. Online Vehicle Detection using Haar-like, LBP and HOG Feature based Image Classifiers with Stereo Vision Preselection. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2017.
- [170] Dennis Nienhüser. *Kontextsensitive Erkennung und Interpretation fahrrelevanter statischer Verkehrselemente*. Dissertation, Karlsruhe Institute of Technology, 2014.
- [171] Dennis Nienhüser, Thomas Gump, J. Marius Zöllner, and Koba Natroshvili. Fast and reliable recognition of supplementary traffic signs. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2010.
- [172] Malte Oeljeklaus. *An Integrated Approach for Traffic Scene Understanding from Monocular Cameras*. Dissertation, TU Dortmund, 2021.
- [173] Malte Oeljeklaus, Frank Hoffmann, and Torsten Bertram. A Fast Multi-Task CNN for Spatial Understanding of Traffic Scenes. In *International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018.
- [174] Masako Omachi and Shinichiro Omachi. Traffic Light Detection with Color and Edge Information. In *International Conference on Computer Science and Information Technology (ICCSIT)*. IEEE, 2009.
- [175] Masako Omachi and Shinichiro Omachi. Detection of Traffic Light Using Structural Information. In *International Conference on Signal Processing (ICSP)*. IEEE, 2010.
- [176] Wolters Kluwer Online. Bundesgerichtshof Urt. v. 26.11.1969, Az.: 4 StR 458/69 – Redaktioneller Leitsatz. <https://research.wolterskluwer-online.de/document/e5a00fb8-d840-4803-97a0-bbb8a8d7e16f>, 1969. Accessed: 2021-09-23.
- [177] Constantine Papageorgiou and Tomaso Poggio. A Trainable System for Object Detection. *International Journal of Computer Vision*, 38(1):15–33, 2000.
- [178] Pedro O. Pinheiro, Ronan Collobert, and Piotr Dollár. Learning to Segment Object Candidates. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

- [179] Pedro O. Pinheiro, Tsung-Yi Lin, Ronan Collobert, and Piotr Dollár. Learning to Refine Object Segments. In *European Conference on Computer Vision (ECCV)*. Springer, 2016.
- [180] Adam Polyak and Lior Wolf. Channel-Level Acceleration of Deep Face Representations. *IEEE Access*, 3:2163–2175, 2015.
- [181] Alex Pon, Oles Adrienko, Ali Harakeh, and Steven L. Waslander. A Hierarchical Deep Architecture and Mini-Batch Selection Method For Joint Traffic Sign and Light Detection. In *Conference on Computer and Robot Vision (CRV)*. IEEE, 2018.
- [182] Lucas C. Possatti, Rânik Guidolini, Vinicius B. Cardoso, Rodrigo F. Berriel, Thiago M. Paixão, Claudine Badue, et al. Traffic Light Recognition Using Deep Learning and Prior Maps for Autonomous Cars. In *International Joint Conference on Neural Networks (IJCNN)*, 2019.
- [183] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum PointNets for 3D Object Detection from RGB-D Data. *CoRR*, abs/1711.08488, 2017.
- [184] Zheng Qin, Zeming Li, Zhaoning Zhang, Yiping Bao, Gang Yu, Yuxing Peng, and Jian Sun. ThunderNet: Towards Real-time Generic Object Detection on Mobile Devices. In *International Conference on Computer Vision (ICCV)*. IEEE, 2019.
- [185] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *European Conference on Computer Vision (ECCV)*. Springer, 2016.
- [186] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.
- [187] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.
- [188] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. *CoRR*, abs/1804.02767, 2018.
- [189] Russell Reed. Pruning Algorithms – A Survey. *Transactions on Neural Networks (TNN)*, 4(5):740–747, 1993.
- [190] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.
- [191] Werner Ritter. Traffic Sign Recognition in Color Image Sequences. In *Intelligent Vehicles Symposium (IV)*. IEEE, 1992.

- [192] Werner Ritter. Automatische Verkehrszeichenerkennung. *Koblenzer Schriften zur Informatik*, 5, 1997.
- [193] Werner Ritter, F. Stein, and Reinhard Janssen. Traffic Sign Recognition Using Colour Information. *Mathematical and computer modelling*, 22(4-7):149–161, 1995.
- [194] Thomas Roddick, Alex Kendall, and Roberto Cipolla. Orthographic Feature Transform for Monocular 3D Object Detection. In *British Machine Vision Conference*. BMVA Press, 2019.
- [195] Eduardo Romera, Luis M. Bergasa, Jose M. Alvarez, and Mohan Trivedi. Train Here, Deploy There: Robust Segmentation in Unseen Domains. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2018.
- [196] Eduardo Romera, Luis M. Bergasa, and Roberto Arroyo. Can we unify monocular detectors for autonomous driving by using the pixel-wise semantic segmentation of CNNs? In *Intelligent Vehicles Symposium (IV) - Workshops*. IEEE, 2016.
- [197] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*. Springer, 2015.
- [198] Henry A. Rowley, Shumeet Baluja, and Takeo Kanade. Human Face Detection in Visual Scenes. In *Advances in Neural Information Processing Systems (NIPS)*, 1996.
- [199] Sebastian Ruder. An Overview of Multi-Task Learning in Deep Neural Networks. *CoRR*, abs/1706.05098, 2017.
- [200] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
- [201] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018.
- [202] Thomas Schamm. *Modellbasierter Ansatz zur probabilistischen Interpretation von Fahrsituationen*. Dissertation, Karlsruhe Institute of Technology, 2014.
- [203] Thomas Schamm, J. Marius Zöllner, Stefan Vacek, Joachim Schröder, and Rüdiger Dillmann. Obstacle Detection with a Photonic Mixing Device-Camera in Autonomous Vehicles. *International Journal of Intelligent Systems Technologies and Applications (IJISTA)*, 5(3-4):315–324, 2008.
- [204] Joachim Schiehlen and Ernst Dickmanns. A Camera Platform for Intelligent Vehicles. In *Intelligent Vehicles Symposium (IV)*. IEEE, 1994.

- [205] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *CoRR*, abs/1312.6229, 2013.
- [206] Burr Settles. Active Learning Literature Survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.
- [207] Claude Elwood Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3):379–423, 1948.
- [208] Yehu Shen, Umit Ozguner, Keith Redmill, and Jilin Liu. A Robust Video based Traffic Light Detection Algorithm for Intelligent Vehicles. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2009.
- [209] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. DSOD: Learning Deeply Supervised Object Detectors from Scratch. In *International Conference on Computer Vision (ICCV)*. IEEE, 2017.
- [210] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training Region-Based Object Detectors with Online Hard Example Mining. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.
- [211] Laurent Sifre and Stéphane Mallat. Rigid-Motion Scattering for Texture Classification. *CoRR*, abs/1403.1687, 2014.
- [212] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR)*, 2015.
- [213] Ganesh Sistu, Isabelle Leang, Sumanth Chennupati, Senthil Yogamani, Ciarán Hughes, Stefan Milz, and Samir Rawashdeh. NeurAll: Towards a Unified Visual Perception Model for Automated Driving. In *International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2019.
- [214] Sayanan Sivaraman and Mohan M. Trivedi. A Review of Recent Developments in Vision-Based Vehicle Detection. In *Intelligent Vehicles Symposium (IV)*. IEEE, 2013.
- [215] Suraj Srinivas and R. Venkatesh Babu. Data-free Parameter Pruning for Deep Neural Networks. In *British Machine Vision Conference*. BMVA Press, 2015.
- [216] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [217] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition. *Neural Networks*, 32:323–332, 2012.

- [218] Zehang Sun, George Bebis, and Ronald Miller. On-Road Vehicle Detection: A Review. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 28(5):694–711, 2006.
- [219] Kah-Kay Sung. *Learning and Example Selection for Object and Pattern Detection*. Dissertation, Massachusetts Institute of Technology, 1996.
- [220] Kah-Kay Sung and Tomaso Poggio. Example-Based Learning for View-Based Human Face Detection. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 20(1):39–51, 1998.
- [221] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *AAAI Conference on Artificial Intelligence*, 2017.
- [222] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, et al. Going Deeper with Convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015.
- [223] Christian Szegedy, Scott Reed, Dumitru Erhan, Dragomir Anguelov, and Sergey Ioffe. Scalable High-Quality Object Detection. *CoRR*, abs/1412.1441, 2014.
- [224] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep Neural Networks for Object Detection. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.
- [225] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.
- [226] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019.
- [227] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *International Conference on Machine Learning (ICML)*, 2019.
- [228] Mingxing Tan, Ruoming Pang, and Quoc V. Le. EfficientDet: Scalable and Efficient Object Detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020.
- [229] Sebastian Thrun. Is Learning The n-th Thing Any Easier Than Learning The First? In *Advances in Neural Information Processing Systems (NIPS)*, 1996.
- [230] Ludovic Trottier, Philippe Giguere, and Brahim Chaib-Draa. Parametric Exponential Linear Unit for Deep Convolutional Neural Networks. In *International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2017.

- [231] Jonas Uhrig, Marius Cordts, Uwe Franke, and Thomas Brox. Pixel-level Encoding and Depth Layering for Instance-level Semantic Labeling. In *German Conference on Pattern Recognition (GCPR)*. Springer, 2016.
- [232] Jasper R. R. Uijlings, Koen E. A. Van De Sande, Theo Gevers, and Arnold W. M. Smeulders. Selective Search for Object Recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.
- [233] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. Multi-Task Learning for Dense Prediction Tasks: A Survey. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021.
- [234] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. Improving the speed of neural networks on CPUs. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.
- [235] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. CSPNet: A New Backbone that can Enhance Learning Capability of CNN. In *Conference on Computer Vision and Pattern Recognition (CVPR) - Workshops*. IEEE, 2020.
- [236] Chunxiang Wang, Tao Jin, Ming Yang, and Bing Wang. Robust and Real-Time Traffic Lights Recognition in Complex Urban Environments. *International Journal of Computational Intelligence Systems*, 4(6):1383–1390, 2011.
- [237] Zhenyang Wang, Zhiwei Cheng, Hongcheng Huang, and Jiabin Zhao. ShuDA-RFBNet for Real-time Multi-task Traffic Scene Perception. In *Chinese Automation Congress (CAC)*. IEEE, 2019.
- [238] Bichen Wu, Forrest Iandola, Peter H Jin, and Kurt Keutzer. SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving. In *Conference on Computer Vision and Pattern Recognition (CVPR) - Workshops*. IEEE, 2017.
- [239] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.
- [240] Fan Yang, Wongun Choi, and Yuanqing Lin. Exploit All the Layers: Fast and Accurate CNN Object Detector with Scale Dependent Pooling and Cascaded Rejection Classifiers. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.
- [241] Yongxin Yang and Timothy Hospedales. Trace Norm Regularised Deep Multi-Task Learning. In *International Conference on Learning Representations (ICLR) - Workshops*, 2017.
- [242] Chunhe Yu, Chuan Huang, and Yao Lang. Traffic Light Detection During Day and Night Conditions by a Camera. In *International Conference on Signal Processing*. IEEE, 2010.

- [243] Sangdoon Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. In *International Conference on Computer Vision (ICCV)*. IEEE, 2019.
- [244] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In *European Conference on Computer Vision (ECCV)*. Springer, 2014.
- [245] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018.
- [246] Yue Zhang, Jianru Xue, Geng Zhang, Yingwei Zhang, and Nanning Zheng. A Multi-Feature Fusion Based Traffic Light Recognition Algorithm for Intelligent Vehicles. In *Chinese Control Conference*. IEEE, 2014.
- [247] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial Landmark Detection by Deep Multi-task Learning. In *European Conference on Computer Vision (ECCV)*. Springer, 2014.
- [248] Yong-Jian Zheng, Werner Ritter, and Reinhard Janssen. An Adaptive System for Traffic Sign Recognition. In *Intelligent Vehicles Symposium (IV)*. IEEE, 1994.
- [249] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. In *International Conference on Learning Representations (ICLR)*, 2017.
- [250] C. Lawrence Zitnick and Piotr Dollár. Edge Boxes: Locating Object Proposals from Edges. In *European Conference on Computer Vision (ECCV)*. Springer, 2014.
- [251] Will Y. Zou, Xiaoyu Wang, Miao Sun, and Yuanqing Lin. Generic Object Detection with Dense Neural Patterns and Regionlets. In *British Machine Vision Conference*. BMVA Press, 2014.
- [252] Straßenverkehrsunfall und Fahrzeugschaden – ÖNORM V 5050:2011-04-15, April 2011.