Smart Mechatronics

Hochschule Bochum
Bochum University
of Applied Sciences

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

Entwicklung einer Bildverarbeitung mit dem Schwerpunkt Personenerkennung für ein autonomes Logistik-Fahrzeug

Autor: Giuliano Montorio
giuliano.montorio@hs-bochum.de
Matrikelnummer: 015202887

Erstgutachter: Prof. Dr.-Ing. Arno Bergmann

Zweitgutachter: M.Sc. Bernd Möllenbeck

Abgabedatum: tt.mm.jjjj

Eidesstattliche Erklärung

Eidesstattliche Erklärung zur Abschlussarbeit:

«Entwicklung einer Bildverarbeitung mit dem Schwerpunkt Personenerkennung für ein autonomes Logistik-Fahrzeug»

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Bochum, 13. Oktober 2020	
Ort, Datum	Giuliano Montorio

Danksagung

Ein großes Dankeschön gilt all jenen Personen, die mich im Rahmen dieser Masterarbeit unterstützt und motiviert haben. Insbesondere möchte ich Herrn Prof. Dr.-Ing. Arno Bergmann, Herrn M.Sc. Bernd Möllenbeck und Herrn Dr.-Ing. Christoph Krimpmann danken, die meine Arbeit durch ihre fachliche und persönliche Unterstützung begleitet haben. Ich danke Guido Stollt und der gesamten Smart Mechatronics GmbH für die tolle Zusammenarbeit. Auch beim Fachbereich Elektrotechnik und Informatik der Hochschule Bochum, insbesondere Herrn Dipl.-Ing. Thorsten Bartsch möchte ich mich bedanken. Meiner Familie spreche ich meinen Dank aus, da sie mir mein Studium durch finanzielle, doch vor allem mentale Unterstützung ermöglicht haben. Bei meinem Bruder bedanke ich mich für die Bereitstellung seiner Räumlichkeiten für eine fortlaufende Entwicklung am autonomen Logistikfahrzeug während der Coronapandemie. Ein spezieller Dank gebührt Hannes Dittmann. Mit ihm habe ich nicht nur einen wegweisenden Mitstreiter gefunden, sondern auch einen tollen Freund.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ALF	**A**utonomes **L**ogistik **F**ahrzeug
CNN	**C**onvolutional **N**eural **N**etworks
COCO	**C**ommon **O**bjects in **Co**ntext
CONSENS	**Con**ceptual Design **S**pecification Technique for the **En**gineering of Complex **S**ystems
cuDNN	**CU**DA **D**eep **N**eural **N**etwork
EA	**E**ndlicher **A**utomat
FN	**F**alse **N**egative
FP	**F**alse **P**ositive
FPS	**F**rames **P**er **S**econd
HOG	**H**istogram of **O**riented **G**radients
IoU	**I**ntersection **o**ver **U**nion
KI	**K**ünstliche **I**ntelligenz
KNN	**K**ünstliches **N**euronales **N**etz
LiDAR	**Li**ght **D**etection **A**nd **R**anging
mAP	**M**ean **A**verage **P**recision
MMI	**M**ensch-**M**aschine-**I**nterface
ReLU	**Re**ctifier **L**inear **U**nit

ROS **R**obot **O**perating **S**ystem

SAE **S**ociety of **A**utomotive **E**ngineers

SSD **S**ingle **S**hot Multibox **D**etector

SVM **S**upport **V**ector **M**achine

TN **T**rue **N**egative

TP **T**rue **P**ositive

SLAM **S**imultaneous **L**ocalization **A**nd **M**apping

Symbolverzeichnis

Symbol	Bedeutung
D_f	Dimension der Eingangsdaten
D_k	Dimension des Kernels
F	veränderte Bilddaten
FN	Anzahl aller False Positive Werte
FP	Anzahl aller False Negative Werte
H	Neuron einer versteckten Schicht
I	Neuron einer Eingabeschicht
K	Anzahl aller Klassen
N	Tiefe der Ausgangsdaten
M	Tiefe der Eingansdaten
O	Neuron einer Ausgabeschicht
R_d	Rechenaufwand bei konventioneller Faltung
R_p	Rechenaufwand bei tiefenorientierter Faltung
TN	Anzahl aller True Negative Werte
TP	Anzahl aller True Positive Werte
b	Bildbreite
d	Distanz

Symbol	Bedeutung
d'	Begrenzung der Distanz durch die Objektivebene
g	Binärfunktion
i, j	Neuron
k	Knotenemenge
m	Modusfunktion
p	Precision
p_x	laterale Positionskoordinate
p'_x	Pixelkoordinate
p_y	longitudinale Positionskooridinate
p'_y	Begrenzung der longitudinalen Positionskoordinate durch die Objektivebene
r	Recall
t	Zeitschritt
x	Bilddaten
w	Gewichte
α	Weitenmultiplikator
β	Sichtwinkel
γ	Ausgabe eines Zustandsautomats
ϵ	Zustand
ζ	Eingabe in Zustandsautomat
ρ	Auflösungsmultiplikator
ϕ	Transitionsfunktion

Symbol	Bedeutung
ψ	Ausgabefunktion

1 Einleitung

Das Thema der künstlichen Intelligenz (KI) dringt zunehmend in den Alltag des Menschen ein. *Smart Home* Geräte wie *Amazons Alexa* oder der *Google Assistant* wurden bereits Ende des Jahres 2018 in Anzahlen jenseits der 20 Millionen verkauft [1]. Derartige Technologien begleiten den Menschen jedoch nicht im Alltag, sondern auch in der Transport- und Logistikbranche. Eine Potenzialanalyse zur künstlichen Intelligenz der Firma *Sopra Steria* zeigt, dass bereits im Jahr 2017 20% aller befragten Unternehmen derartige Systeme einsetzten [2]. 37% planten den zukünftigen Einsatz [2]. Die Implementierung solcher Systeme hat Einfluss auf verschiedene Eigenschaften der Wertschöpfungskette [2]. Einige der zur Logistikbranche gehörenden Transportfahrzeuge sind ebenfalls mit KI ausgestattet [2].

(a) (b)

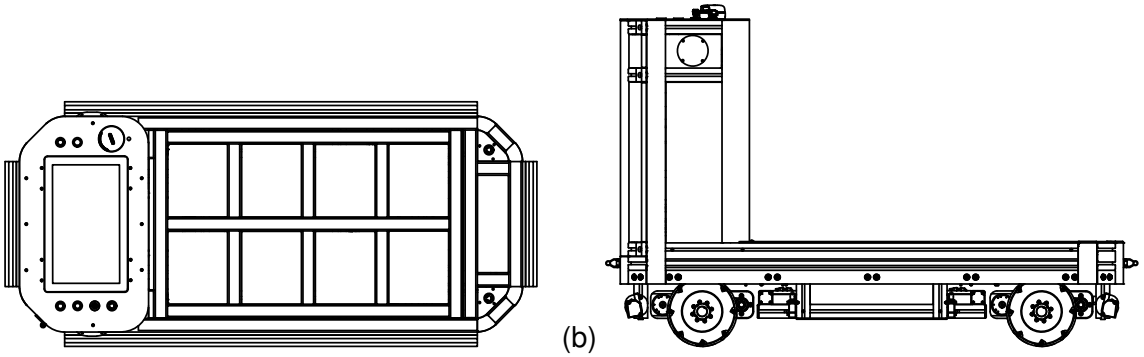


Abbildung 1.1: (a) Darstellung des autonomen Logistik-Fahrzeugs aus der Draufsicht. Die acht benachbarten Rechtecke in der Mitte des Fahrzeugs stellen die Ladefläche des Fahrzeugs dar. (b) Darstellung aus der Seitenansicht. Oben links in der Abbildung ist der Schaltschrank zu sehen. Die Räder des Fahrzeugs befinden sich unten links und unten rechts in der Darstellung.

Das autonome Logistikfahrzeug ALF ist ein solches Transportfahrzeug. Die Idee des ALFs ist es, ein Fahrzeug zu entwickeln, das nach seiner Fertigstellung Logistikaufgaben am Standort der Hochschule Bochum löst. Es dient im Labor für Antriebstechnik der Hochschule Bochum als Versuchs- und Entwicklungsplattform für praktische Anwendungen. Der Entwicklungsprozess

stellt sich aus diversen Bachelor- und Masterarbeiten zusammen, die sowohl Hardware als auch Softwareintegrationen vorsehen.

Bisher wurden zwei Abschlussarbeiten inklusive der praktischen Anwendung am ALF geschrieben. M.Sc. Dennis Hotze und M.Sc. Dominik Eickmann entwickelten in ihrem Masterprojekt das Fahrzeug und konnten Fahraufgaben ferngesteuert und manuell erledigen [3]. Während der darauffolgenden Bachelorarbeit wurde eine Schlupfkompensation entwickelt, die den Drift am Fahrzeug durch Eingabe von Umgebungsinformationen verhindert [4]. Weiterhin wurden Funktionen entwickelt, um grundlegende und autonome Fahraufgaben zu lösen [4]. Das autonome Logistikfahrzeug aus der vorangegangenen Bachelorarbeit dient auch in dieser Masterarbeit als Versuchsplattform.

Der tägliche Kontakt zu Menschen ist nicht nur für das ALF, sondern auch für andere Transportfahrzeuge im öffentlichen Raum häufig unentbehrlich. Kreuzen sich die Wege eines autonomen Fahrzeugs mit der einer oder mehrerer Personen, darf es in keiner Weise zur Kollision kommen. Das ALF zählt mit einem Maximalgewicht von 600 kg zu der Art von Fahrzeugen, die bei einem Zusammenstoß Verletzungen hervorrufen können. Ein solches System muss folglich in der Lage sein, bei einer Annäherung von Personen eine gesonderte Gefahreneinschätzung vorzunehmen.

Im Einsatz an der Hochschule Bochum kommt es eher selten zu Begegnungen mit Tieren, die nicht in Begleitung von Personen sind. Dementsprechend wird das System im Rahmen dieses Projekts auf die Erkennung von Personen beschränkt. Ein weiteres, übliches Szenario solcher Systeme im öffentlichen Raum ist die Bedienung durch nicht autorisierte Personen. Für derartige Zwecke muss ein Mensch nicht nur als bewegtes Objekt, sondern auch als solcher erkannt werden.

Das Hauptziel dieser Masterarbeit ist die Entwicklung einer Personenerkennung. Die integrierten *Kinect*-Kameras dienen neben einem *Light Detection and Ranging* (LiDAR) Sensor als einzige optische Sensoren des autonomen Logistikfahrzeugs. Die Sensordaten werden nach wie vor mithilfe des *Robot Operating System* (ROS) verarbeitet [4]. Zwar gibt es technische Lösungen für eine Personenerkennung mithilfe von zweidimensionalen Laserdaten, jedoch nutzen state-of-the-art Systeme Bildverarbeitung zur Erkennung von Objekten [5, 6]. Folglich wird das System zur Personenerkennung mithilfe der beiden *Kinect*-Kameras ausgelegt. Weiterhin wird das System nicht nur Personen detektieren können, sondern diese auch wiedererkennen.

Während der Bearbeitung von Transport- oder Fahraufgaben eines autonomen Fahrzeugs kann es zu diversen Komplikationen kommen. Beispielsweise können besonders im Anwendungsbereich der Hochschule Bochum Objekte den Verlauf einer Route unterbrechen und ein Ziel sogar unerreichbar machen. Häufig können diese Probleme durch menschliche Hilfe beseitigt werden. Wiederum setzt dies eine Interaktion mit umstehenden Personen voraus. Eine Besonderheit dieser Masterarbeit ist die theoretische und praktische Entwicklung parallel zu einem weiteren Projekt. Hannes Dittmann entwickelt in seiner Masterarbeit eine Sprachverarbeitung zur Klassifikation anwendungsorientierter Sprache [7]. Diese stellt eine auditive *Mensch-Maschine-Interface* (MMI) Schnittstelle her. So kann eine Person über ein Aufnahmegerät mit dem System kommunizieren. Die klassifizierte Sprache ist ohne Weiteres nicht in der Lage, das Fahrzeug enstprechend zu steuern. Demnach wird in dieser Masterarbeit ein Konzept zur Steuerung des ALFs mit klassifizierter Sprache entwickelt und am Fahrzeug implementiert.

Die Struktur dieser Masterthesis ist in 4 Kapiteln gegliedert. Beginnend mit Kapitel 2 werden die Grundlagen der eingesetzten Methoden und Systeme vermittelt. Hierbei werden Grundbegriffe des Themenbereichs der künstlichen Intelligenz und insbesondere der neuronalen Netze erklärt. Kapitel 3 zeigt, wie die vermittelten Grundlagen zu einem System zusammengeführt und verwendet werden. Eine Evaluierung der im Feld getesteten Personenerkennung ist in Kapitel 4 beschrieben. Ein Vergleich alternativer Lösungen ist dort ebenfalls präsentiert. Das letzte Kapitel beschreibt zusammenfassend die gesamte Masterarbeit und gibt einen Ausblick für zukünftige Projekte am autonomen Logistikfahrzeug.

Diese und die Masterarbeit von *Hannes Dittmann* bilden im praktischen Kontext ein überarbeitetes Gesamtsystem des bereits bestehenden autonomen Logistikfahrzeugs [7]. Die genannten Ziele sind im angehängten Lastenheft A.1.3 festgehalten. Die erarbeiteten Ergebnisse werden so anhand eines Verifikationsplans am Fahrzeug geprüft.

2 Grundlagen

Für ein besseres Verständnis der in Kapitel 3 angewandten Methoden werden folgend die Grundlagen behandelt. Informationen zu der verwendeten Hard- und Software wurden bereits in der vorangegangenen Bachelorarbeit vermittelt [4]. Die in Kapitel 1 erwähnte Personenerkennung ist eine Abwandlung der Objekterkennung. Im Folgenden werden sowohl herkömmliche als auch state-of-the-art Lösungen zur Objekterkennung präsentiert. Letzteres beinhaltet unter anderem das Gebiet der neuronalen Netzwerke. Somit werden zunächst die neuronalen Netze und ihre Eigenschaften besprochen, bevor die Grundlagen der Objekterkennung erläutert werden.

2.1 Neuronale Netze

Das Neuronennetz des menschlichen Gehirns dient als Vorbild für künstliche, neuronale Netze (KNN) [10]. Sie ermöglichen es, komplexe Strukturen und Muster aus großen Datenmengen zu erkennen und werden daher verwendet, um diverse Anwendungsprobleme zu lösen. [10]. Das in diesem Projekt zugrundeliegende Bildverarbeitungsproblem besitzt jene Eigenschaften und eignet sich für den Einsatz eines neuronalen Netzes als Problemlösung. Anders als bei regelbasierten Systemen verhalten sich KNNs grundlegend verschieden [11]. Sie lernen Verhaltensmuster basierend auf den entsprechenden Trainingsdaten [11]. Diese unterscheiden sich jedoch häufig mit den Daten, die während des Einsatzes im Feld eingegeben werden. Somit kann keine Aussage darüber getroffen werden, welches Verhalten durch das KNN erzeugt wird. Dies führt unter Umständen zu unvorhersehbaren Ergebnissen. Am vorliegenden, autonomen Logistikfahrzeug werden zur Personenerkennung derartige neuronale Netze untersucht.

2.1.1 Eigenschaften von neuronalen Netzen

Die Grundlage für die Eingabe in ein neuronales Netz sind entsprechende, zu analysierende Daten. Bei einem Anwendungsfall, in dem eine Audiospur analysiert werden soll, können beispielsweise Frequenzspektren eingegeben werden. Ein klassisches Bildverarbeitungsproblem

arbeitet mit den Pixeln eines Bildes. Hierbei werden Daten über eine Eingabeschicht in die darin enthaltenen Neuronen gegeben.

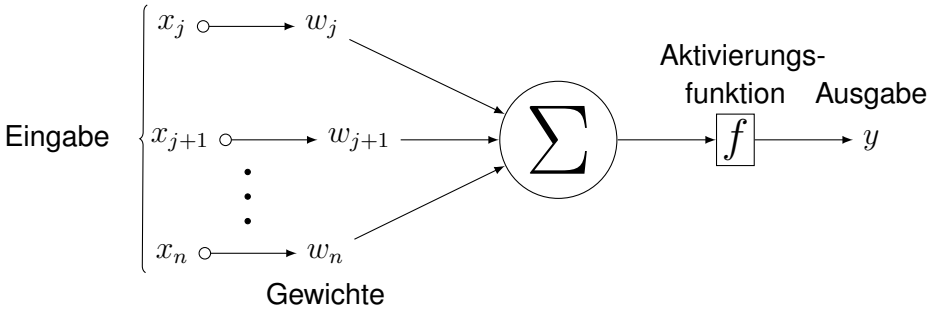


Abbildung 2.1: Darstellung der prinzipiellen Funktion eines Neurons in einem neuronalen Netz. Drei aufeinander folgende Punkte deuten eine Fortsetzung an. [12]

Der grundlegende Aufbau eines neuronalen Netzes besteht aus verschiedenen, miteinander verbundenen Schichten [13]. Ein Neuron i einer Schicht ist jeweils mit dem Neuron j der folgenden Schicht über das Gewicht w_{ij} verbunden [13]. Die typische Struktur eines Neurons ist in Abbildung 2.1 zu sehen.

$$s = \sum_{j=1}^{n} w_{ij} x_j \tag{2.1}$$

Es verarbeitet im Wesentlichen eingehende Bildinformationen x_n und gibt diese durch die Ausgabe y aus. Genauer wird mit den eingehenden Zahlenwerten eine gewichtete Summe s gebildet. Diese wird dann auf eine Aktivierungsfunktion angewendet und ausgegeben [13]. Diese aktiviert bzw. reizt das Neuron ab einem Schwellwert [14]. Es gibt verschiedene Varianten der Aktivierungsfunktion, die je nach Netzarchitektur zur Anwendung kommen können. Gleichung 2.1 zeigt das mathematische Modell der gewichteten Summe s.

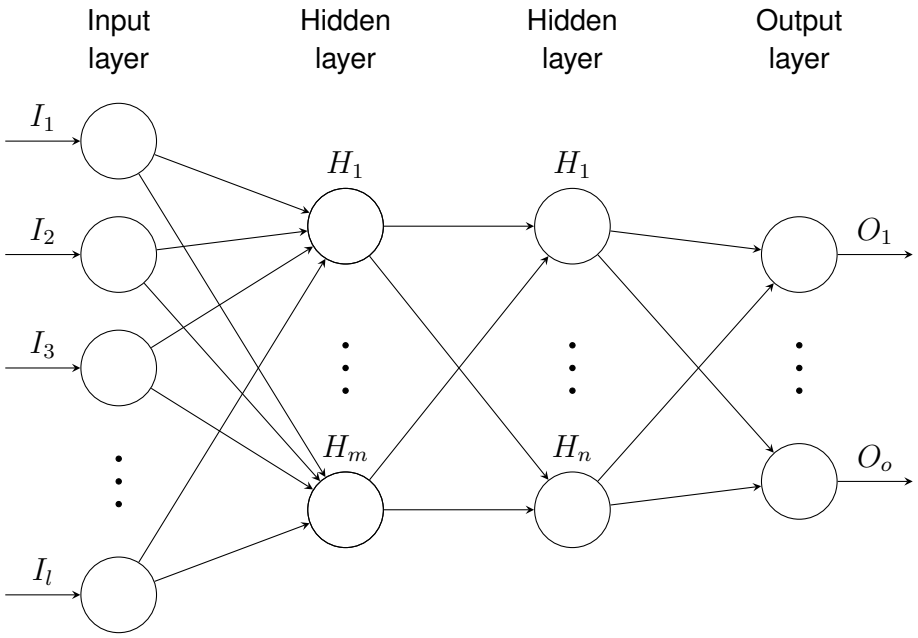


Abbildung 2.2: Prinzipielle Darstellung eines künstlichen, neuronalen Netzwerks. Neuronen werden als Kreise dargestellt. Drei aufeinander folgende Punkte deuten eine Fortsetzung an. Adaptiert aus [12].

In Abbildung 2.2 ist die Grundstruktur eines neuronalen Netzes veranschaulicht. Neuronen sind hier mit Kreisen angedeutet und bilden einzelne Schichten in den beispielhaft vertikal dargestellten Formationen. Hierbei wird zwischen Eingabe-, Zwischen- und Ausgabeschichten unterschieden. Entsprechende Neuronen sind mit I, H und O bezeichnet. Die Eingabeschicht nimmt Informationen in Form von Daten auf und gibt diese an die erste Zwischenschicht weiter. Die Anzahl der Zwischenschichten, oder auch verdeckten Schichten, ist in der Anwendung neuronaler Netze variabel. Am rechten Bildrand ist die Ausgabeschicht gezeigt, die die entsprechende Ausgabe des Netzes generiert.

2.1.2 Lernprozess

Der Lernprozess von neuronalen Netzen zielt darauf ab, einer Netzstruktur ein gewünschtes Verhalten beizubringen [11]. Genauer sollen die in Abschnitt 2.1.1 beschriebenen Gewichte so modifiziert werden, dass sie eine bestimmte Ausgabe erzeugen.

Es wird zwischen drei wesentlichen Lernverfahren unterschieden, dem unüberwachten, dem bestärkenden und dem überwachten Lernen [14]. Beim unüberwachten Lernen erkennt das

Netz selbst Muster und versucht diese aus der eingegebenen Menge in Klassen zu unterteilen [14]. Anders als beim unüberwachten Lernen lernt das Netz beim bestärkten Lernen mit einer Rückmeldung. Diese enthält Informationen darüber, ob ein errechnetes Ergebnis einer Trainingseinheit richtig oder falsch ist [14]. Das überwachte Lernen setzt eine Trainingsmenge voraus, die neben der Eingabedaten auch das dazugehörige korrekte Ergebnis enthält [14, 13]. Im Falle einer Personenerkennung wäre beispielsweise ein Datensatz aus Bildern von Personen eine geeignete Trainingsmenge. In der sogenannten Vorwärtspropagierung wird durch eine Eingabe eine entsprechende Ausgabe erzeugt und diese mit dem korrekten Ergebnis verglichen [13]. Ausgehend von einem KNN mit lediglich einer Schicht, werden die Gewichte dann mithilfe des aus dem vorangegangenen Vergleich entstandenen Fehlers korrigiert [13].

Die meist genutzte Form des überwachten Lernens ist die Rückwärtspropagierung (engl. Backpropagation) oder Fehlerrückführung genannt [15]. Wie bereits beschrieben, bestehen neuronale Netze häufig aus mehreren verdeckten Schichten. Für diese liegt beim Training jedoch kein Korrekturwert vor [13]. Der Algorithmus der Rückwärtspropagierung ist eine mögliche Lösung dieses Problems. Die mathematische Grundlage für dieses Lernverfahren sind Gradientenabstiegsverfahren [14, 13]. Nach der Vorwärtspropagierung wird die Ausgabe des Netzes mit Sollwerten verglichen [13]. Beim darauffolgenden Rückwärtsschritt wird durch die Fehlerrückführung für jede verdeckte Schicht ein Korrekturwert errechnet [13]. Auch in dieser Masterarbeit werden Netze mithilfe der Rückwärtspropagierung trainiert und in diesem Kapitel 4 genauer erläutert.

2.1.3 Evaluation neuronaler Netze

Es bestehen diverse Metriken für Objekterkennungssysteme, die derartige Systeme messbar machen. Im Rahmen dieser Masterarbeit wird zur Evaluation die Metrik *Precision* und *Recall* eingesetzt. Außerdem werden die neuronalen Netze anhand des Top-x Fehlers sowie der *mean average precision* (mAP) verglichen. Die Grundlagen der entsprechenden Metriken werden im Folgenden vermittelt.

Tabelle 2.1: Tabelle zum besseren Verständnis der Begriffe *True Positive*, *False Positive*, *False Negative* und *True Negative*. In der jeweiligen Zelle wird ein Beispiel im praktischen Kontext der Personenerkennung genannt.

		Realität	
		Wahr	Falsch
Ausgabe	Wahr	*True Positive* (TP) Realität: Person vorhanden Ausgabe: Person vorhanden	*False Positive* (FP) Realität: Keine Person Ausgabe: Person vorhanden
	Falsch	*False Negative* (FN) Realität: Person vorhanden Ausgabe: Keine Person	*True Negative* (TN) Realität: Keine Person Ausgabe: Keine Person

Precision und *Recall* ist ein traditionelles Werkzeug zur Evaluation und Leistungsmessung [16]. In Tabelle 2.1 werden zum besseren Verständnis die Begrifflichkeiten *True Positive* (TP), *False Positive*(FP), *False Negative* (FN) und *True Negative* (TN) näher erläutert. Der *Recall*-Wert $r(t)$ beschreibt die Fähigkeit eines Systems, tatsächlich positive Stichproben zu erkennen. Angewandt auf die Personenerkennung sind Bilder, auf denen Personen zu sehen sind, als tatsächlich positive Stichproben einzustufen. In Gleichung 2.2 wird der *Recall*-Wert durch eine Division von allen wahren positiven Werten $TP(t)$ und die Anzahl aller tatsächlich positiven Werte n_{pos} berechnet. Die Variable t definiert den eingestellten Schwellwert. Im Sachkontext ist der Wert als Konfidenz zu betrachten, bei der eine Person als solche klassifiziert wird. Für die Anwendung dieser Methode ist die Kenntnis über negative Beispiele der Stichprobe nicht notwendig [17].

$$r(t) = \frac{TP(t)}{TP(t) + FN(t)} \tag{2.2}$$

Der *Precision*-Wert $p(t)$ berechnet sich durch das Verhältnis von allen wahren positiven Werten $TP(t)$, zu allen als positiv bewerteten Beispielen $TP(t)$ und $FP(t)$. Gleichung 2.3 zeigt die entsprechende Berechnung. Durch diesen Wert wird verdeutlicht, wie gut ein System in der Lage ist, tatsächlich wahre Werte von tatsächlich falschen Werten zu unterscheiden.

$$p(t) = \frac{TP(t)}{TP(t) + FP(t)} \tag{2.3}$$

Häufig muss in der Praxis ein Kompromiss zwischen *Precision* und *Recall* gefunden werden. Dies lässt sich anhand eines Beispiels in der Personenerkennung veranschaulichen. Das System zur Erkennung wird beispielhaft mit einem hohen *Precision*- und einem niedrigen *Recall*-Wert betrieben. Dies führt dazu, dass irrelevanter Bildinhalt selten als Person klassifiziert wird. Jedoch kommt es häufiger vor, dass keine Person detektiert wird, obwohl eine zu sehen ist. Legt man den Fokus auf einen hohen *Recall*- und einen niedrigen *Precision*-Wert, würden Personen dann häufiger als Person klassifiziert werden. Jedoch wird irrelevanter Bildinhalt ebenfalls häufig als Person klassifiziert. Je nach Anwendungsfall eines Netzes wird dann der entsprechende Betriebspunkt zwischen *Precision* und *Recall* gewählt.

Objekterkennungssysteme geben erkannte Objekte mithilfe eines Begrenzungsrahmens aus. Dieser wird mithilfe von Pixelkoordinaten passend zum untersuchten Bild angegeben. Der *Intersection over Union* (IoU) Wert sagt aus, ob die Lokalisierung eines Objekts in einem Bild entsprechend genau ist. Als Grundlage dient hierfür ein Testdatensatz mit verschiedenen Bildern und grundwahren Begrenzungsrahmen für enthaltene Objekte. Zur Ermittlung des *IoU*-Werts werden von einem Objekterkennungssystem ausgegebene mit grundwahren Begrenzungsrahmen verglichen. Hierbei lässt sich die überlappende und die zusammengesetzte Fläche beider Rahmen in ein Verhältnis setzen. Das Ergebnis ist der *IoU*-Wert.

$$mAP = \frac{1}{K} \sum_{i=1}^{K} \int_{0}^{1} p_i(r_i) dr \tag{2.4}$$

Der *mAP*-Wert oder auch die mittlere durchschnittliche Genauigkeit ist ein Indiz dafür, wie genau die Objekterkennung klassenübergreifend arbeitet. Die Berechnung dieses Werts beinhaltet hierbei die bereits vorgestellten *Precision* und *Recall* Werte. Dieser Messwert ist besonders aussagekräftig, wenn Systeme lediglich eine Objektklasse erkennen können. Gleichung 2.4 zeigt die Berechnung der mittleren durchschnittlichen Genauigkeit. Die Variable K ist die Menge

aller Klassen, die durch das Netz erkannt werden können.

Eine einfache Betrachtung zur Evaluierung der Qualität hinsichtlich der Genauigkeit eines Netzes liefert der Top-x Fehler. Der Platzhalter x kann zunächst durch eine beliebige Zahl ersetzt werden. In der Praxis hat sich jedoch der Top-5 und der Top-1 Fehler als Vergleich durchgesetzt [18]. Hierbei wird zunächst ein Bild durch ein beliebiges KNN analysiert und über die Ausgabeschicht extrahiert. Als Beispiel sollte sich im optimalen Fall die tatsächliche Klasse des Bildes unter den x wahrscheinlichsten Klassen befinden, die über das Netz ausgegeben wurden. Folglich sagt der Top-1 Fehler aus, wie oft ein Netz ein eingegebenes Bild falsch klassifiziert hat.

2.2 Objekterkennung

Bei der visuellen Objekterkennung wird ein Objekt, das auf einem Bild gezeigt ist, mit einer gewissen Wahrscheinlichkeit inklusive der Position in der Abbildung erkannt. Die drei Abstraktionsebenen einer solchen Erkennung unterteilen sich in Bildklassifikation, Objektlokalisierung und semantische Segmentierung [17]. Letzteres kommt in dieser Arbeit nicht zur Anwendung und wird aufgrund dessen im Folgenden nicht behandelt. Die Bildklassifikation beschreibt eine Zuweisung von Objektkategorien zu einem gegebenen Bild. Mithilfe einer Merkmalextraktion werden Merkmalsvektoren extrahiert und mit einem Klassifikator berechnet. In den folgenden Abschnitten wird auf die in dieser Arbeit eingesetzten Methoden zur Objekterkennung eingegangen. Hierbei werden insbesondere alternative Verfahren mit modernen state-of-the-art Lösungen zur Objekterkennung gegenübergestellt.

2.2.1 Objekterkennung durch alternative Verfahren

Neben der neuronalen Netze gibt es weitere Methoden zur Objekterkennung. Ein gängiges Verfahren zur Merkmalsextraktion ist das sogenannte Histogram of oriented gradients (HOG) von *Dalal* in Verbindung mit der *Support Vector Machine* (SVM).

Bei diesem Verfahren wird ein Bild in kleine Bereiche, sogenannte Zellen, aufgeteilt [6]. Für jede Zelle wird ein eindimensionales Histogram extrahiert [6]. Dieses enthält Gradienten, die aus den Informationen der Pixel enstehen, wie zum Beispiel durch die Lichtintensität oder die Farbe. Hieraus lassen sich Kanten und Ecken und somit auch Konturen und Muster aus einem Bild erkennen. In den Abbildungen 2.3 a und b lässt sich die Umwandlung des

HOGs beobachten. Die Realität, so wie sie eine übliche Fotokamera widerspiegeln würde, ist in 2.3a zu erkennen. Die entsprechenden Richtungen der Gradienten werden in 2.3b dargestellt.

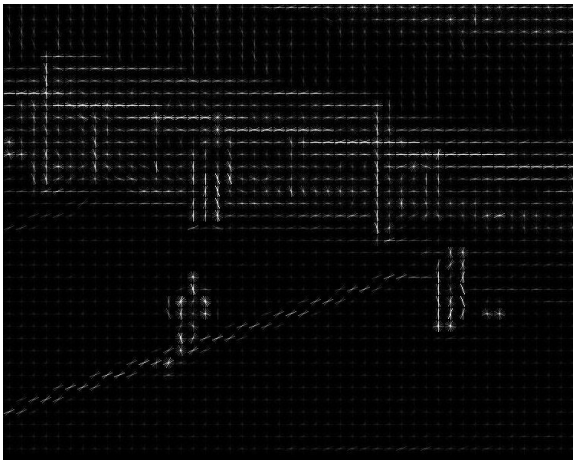
(a)  (b) 

Abbildung 2.3: (a) Foto von drei spielenden Personen [19]. (b) Darstellung der Gradienten, die durch einen *HOG* erzeugt worden sind. Der weiße Farbverlauf deutet die Richtung der analysierten Konturen an. Die Analyse erfolgte mit Bild (a) [19].

Die *SVM* ist ein Funktionsapproximator für eine Objektklassifikation. Häufig wird dieser auf die Ausgangsdaten des *HOGs* angewendet. Hierbei wird ein mathematisches Verfahren angewendet, das Klassen durch Trennungsebenen, sogenannte Hyperebenen, voneinander trennt. Wie auch bei den KNNs gibt es für *SVMs* einen Lernprozess in Form des überwachten Lernens. Ziel der Algorithmen des Trainings einer *SVM* ist es, die Hyperebenen so zu konstruieren, sodass Objekte sicher klassifiziert werden können. In der Realität lassen sich Objekte rein mathematisch häufig nicht linear trennen. Nichtlineare Trennbarkeit bedeutet oft einen höheren Rechenaufwand. Somit verwendet die Methode der *SVM* den sogenannten *Kernel-Trick* [20]. Dieser transformiert Daten in eine höhere Dimension, um eine lineare Trennbarkeit zu erreichen [20]. Diese Methode hat sich vor allem aufgrund ihrer kurzen Rechenzeit durchgesetzt. In Abbildung 2.4a und b ist das Funktionsprinzip prinzipiell dargestellt. Darstellung 2.4a zeigt Objektklassen, die jeweils als Formen dargestellt sind. Eine lineare Trennbarkeit mittels einer Hyperebene ist in diesem Beispiel nicht möglich. Eine dritte Dimension wird in der Darstellung 2.4b verwendet, um die dargestellte, lineare Trennbarkeit zu erreichen.

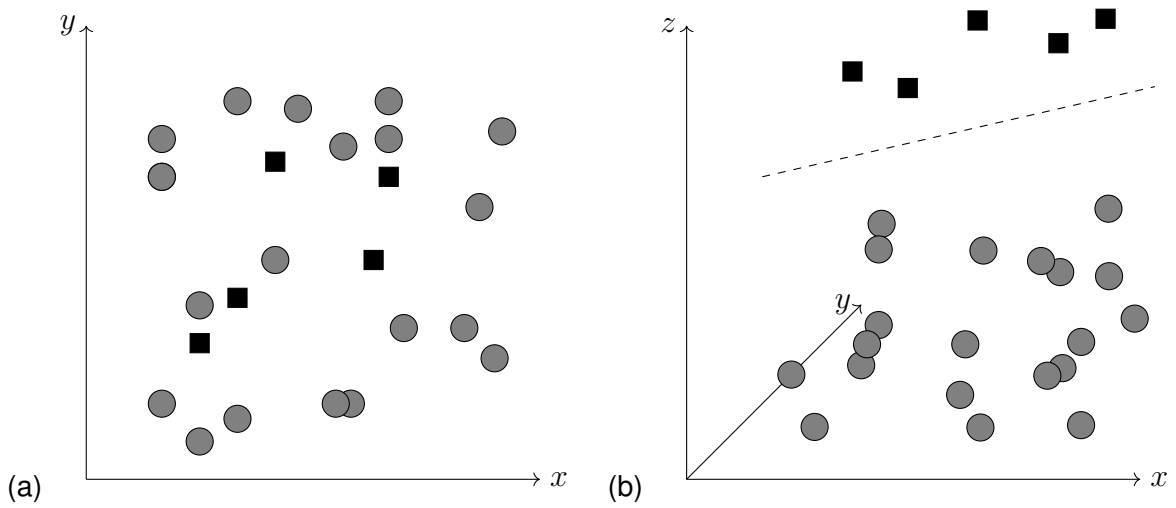


Abbildung 2.4: (a) Abbildung von möglichen, Objektlassen im zweidimensionalen Raum. Die Formen stellen jeweils Klassen dar, wie zum Beispiel die Klassen Hund und Person. (b) Darstellung der in a gezeigten Objektklassen im dreidimensionalen Raum. Die gestrichelte Linie deutet eine lineare Trennung an.

Die *HOG* Methode in Verbindung mit der beschriebenen *SVM* ist ein weitverbreitetes Mittel zur Klassifikation [21]. Die Publikation von *Kibira* und *Hasan* führt Vergleiche hinsichtlich der Trainingszeit und der Genauigkeit von *HOG-SVM*-Kombinationen und Faltungsnetzwerken an [21]. Letzteres weist im direkten Vergleich eine höhere Klassifikationsgenauigkeit bei verhältnismäßig längerer Trainingsdauer auf. Faltungsnetzwerke werden im folgenden Abschnitt behandelt.

2.2.2 Objekterkennung durch neuronale Netze

Die bisher besten Ergebnisse in der Bildverarbeitung wurden durch die Abwandlung der neuronalen Netze, der sogenannten *Convolutional Neural Networks* (CNN) erreicht [22]. Derartige Netzwerke nutzen zur Verarbeitung der Eingangsdaten Faltungsoperationen statt der üblichen Matrizenmultiplikation [22]. Diese werden auch als Konvolution bezeichnet. Der Aufbau eines CNNs setzt sich aus einer Merkmalsextraktion der Eingangsdaten und der darauffolgenden Klassifikation zusammen.

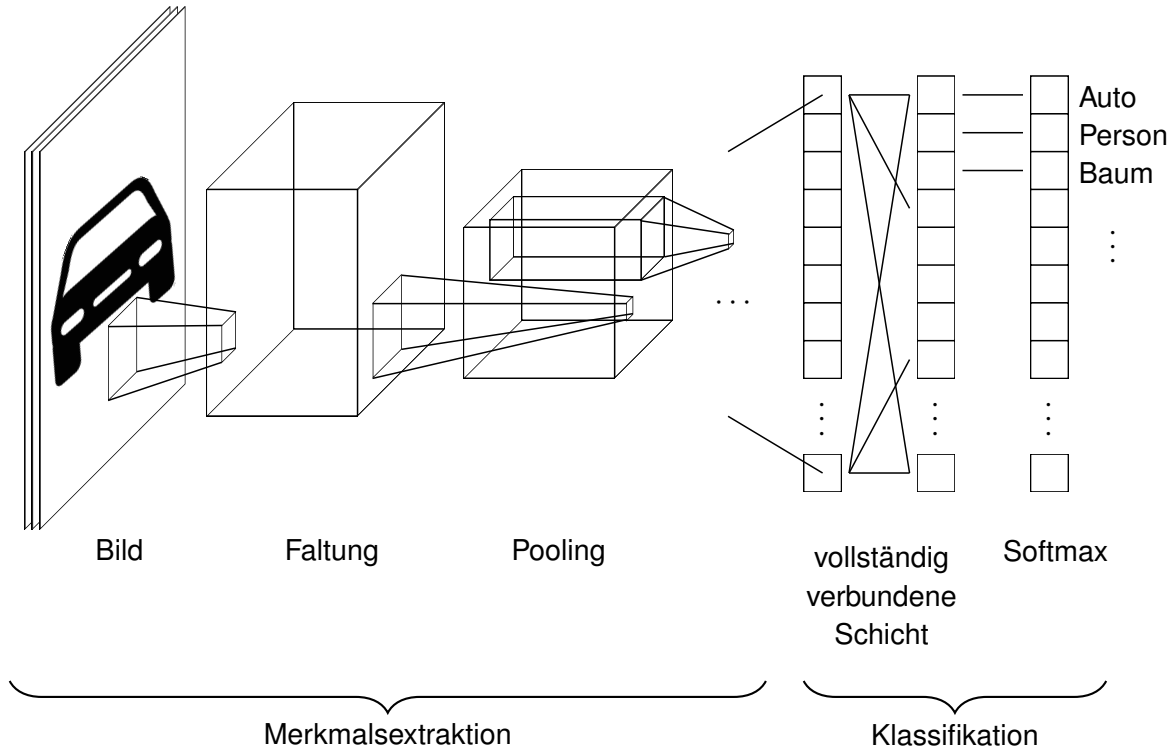


Abbildung 2.5: Prinzipdarstellung eines Faltungsnetzwerks. Fortsetzungen werden mit drei aufeinander folgenden Punkten gekennzeichnet. Durch rechteckige Formen werden Schichten dargestellt. Angedeutet durch Pyramidenstümpfe werden Prozesse der Faltung und des Poolings dargestellt. Darstellung adaptiert aus [23, 24].

Eine Einheit der Merkmalsextraktion besteht im grundlegenden Fall aus drei Unterschichten. Dabei können sich diese innerhalb der Merkmalsextraktion hintereinander wiederholen. Dies hat jedoch Einfluss auf die Eigenschaften eines Netzes. Die erste Unterschicht führt Faltungsprozesse mit den Eingangsdaten durch [22]. Im zweiten Schritt wird eine nichtlineare Aktivierungsfunktion wie zum Beispiel die *Rectified Linear Unit* (ReLU) Funktion auf die Ausgangsdaten der Konvolutionsschicht angewendet. In der dritten Unterschicht wird das sogenannte *Pooling* durchgeführt. In einigen Fällen wird die Zusammensetzung der drei Stufen als Konvolutionsschicht bezeichnet, obwohl lediglich die erste Unterschicht die Konvolution vollzieht [22]. Im Laufe dieses Abschnitts wird auf die Motivation der Nutzung und der Funktionsweise eines CNNs eingegangen.

Es gibt drei Grundsätze für die Nutzung von Faltung in einem neuronalen Netz. Hierzu gehören die eingeschränkte Konnektivität, die Parameterverteilung und die äquivariante Darstellung [22].

Im folgenden Abschnitt werden diese Punkte näher erläutert.

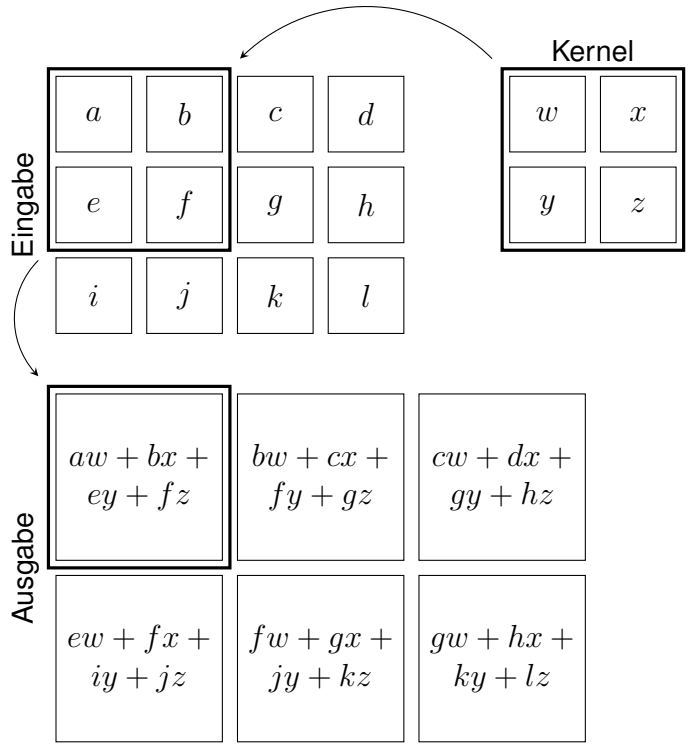
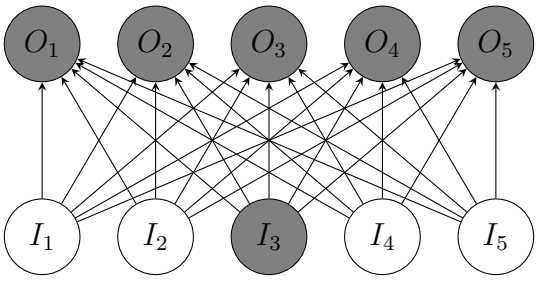


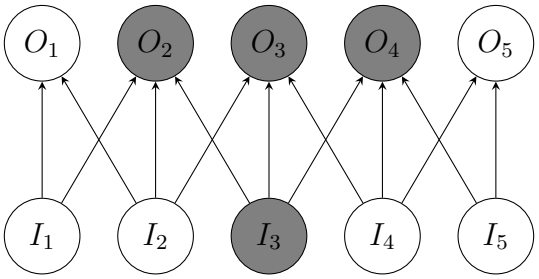
Abbildung 2.6: Prinzipielle Darstellung des Faltungsprozesses. Mit einem Quadrat umrandete Buchstaben stellen Daten wie zum Beispiel einen Pixel dar. Der Kernel fasst in dieser Abbildung beispielsweise vier Daten zu einer Information zusammen und gibt diese aus. Adaptiert aus [22].

Während der Konvolution werden eingehende Daten in Filter, sogenannte Kernel, eingegeben. Abbildung 2.6 zeigt den prinzipiellen Vorgang der Faltung. Hierbei wird die Matrix des Kernels mit der Eingabematrix elementweise multipliziert und erzeugt die gezeigte Ausgabe. Die Konfiguration des Kernels ist beispielhaft mit w, x, y und z dargestellt. Eingehende Datenpunkte sind hier mit den Buchstaben a bis l gekennzeichnet. Die Filter extrahieren bestimmte Merkmale je nach Konfiguration, wie im unteren Teil der Abbildung als Ausgabe dargestellt. So können verschiedene Schichten diverse Merkmale extrahieren. Die Ausgänge der Schichten üblicher, neuronaler Netze sind mit jedem Eingang der folgenden Schicht verknüpft. Ein typischer Aufbau wurde bereits in Abschnitt 2.1.1 in Abbildung 2.2 gezeigt. Wird bei derartigen Netzen eine Schicht mit n Ausgaben und eine mit m Eingaben verknüpft, werden $m \cdot n$ Parameter benötigt [22]. Durch den Faltungsprozess wird diese Konnektivität eingeschränkt. Beispielsweise wird

der Datenpunkt b der Eingabe aus der Darstellung 2.6 lediglich in zwei von sechs Datenpunkten der Ausgabe berücksichtigt. Die Ausmaße der eingeschränkten Konnektivität lassen sich in der folgenden Abbildung 2.7 verdeutlichen. Die Eingabepunkte I_n geben je nach Netzart ihre Informationen an alle oder benachbarten Ausgabepunkte O_n weiter. Die Einschränkung der Konnektivität hängt von der Dimension des Kernels ab. Folglich nehmen CNNs deutlich weniger Speicher im Vergleich zu herkömmlichen KNNs ein[22]. Gleichzeitig wird durch die Faltung eine höhere statistische Effizienz erreicht [22].



(a)



(b)

Abbildung 2.7: (a) Abbildung zweier Schichten eines herkömmlichen, künstlichen, neuronalen Netzes. Die Flussrichtung der Bildinformationen ist mit Pfeilen angedeutet. Horizontal ausgerichtete Kreise stellen eine Schicht dar. Graue Kreise zeigen Neuronen, die voneinander abhängig sind. (b) Grundlegende Darstellung zweier Schichten eines Faltungsnetzes. Es gelten dieselben Darstellungsprinzipien wie in Abbildung (a) [22].

Als Parameterverteilung bezeichnet man die Nutzung eines Parameters pro Funktion eines Netzes [22]. Bei herkömmlichen, ausschließlich vorwärtspropagierenden Netzwerken wird jede Eingabe eines Neurons, wie in Abschnitt 2.1.1 mit einem Gewicht verrechnet [22]. Wie bereits beschrieben, wird bei CNNs ein Kernel pro Schicht für die Merkmalsextraktion verwendet. Dies

führt zu deutlich weniger Speicheraufwand, da das Netz lediglich die Kernel abspeichert statt jedes Gewichte pro Neuron.

Die äquivariante Darstellung bezieht sich auf die Auswirkung der Ausgabe eines Netzes bei einer Änderung der Eingabedaten. Die Merkmalsextraktion eines Netzes erzeugt eine zweidimensionale Karte, die sogenannte *Feature Map*. In dieser werden Merkmale eines Bildes eingetragen. Wird ein Objekt in dem eingegebenen Bild des CNNs bewegt, verändert sich die Merkmalskarte im selben Maße wie das Objekt im Bild [22].

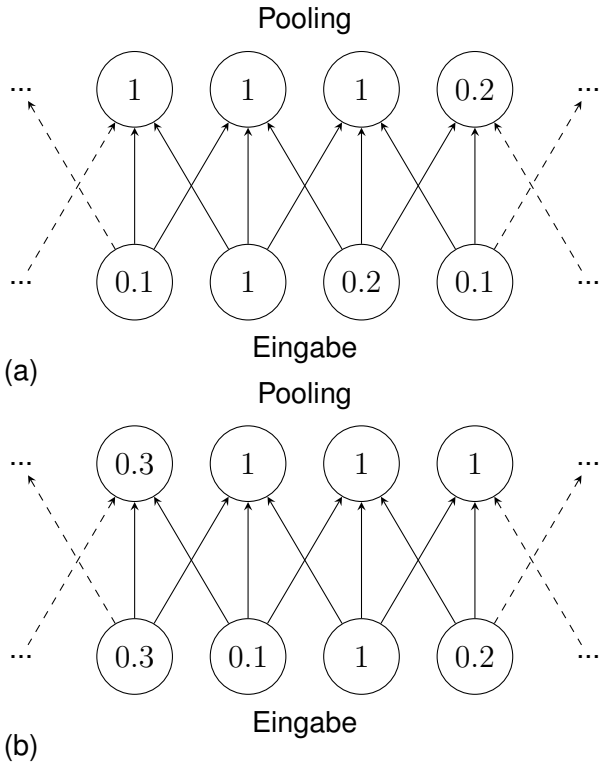


Abbildung 2.8: (a) Beispielhafter Poolingprozess. Die Zahlenwerte deuten Bildinformationen als Fallbeispiel an. (b) Es gelten dieselben Darstellungsprinzipien wie in Abbildung (a). Die Eingangsdaten wurden hierbei um ein Neuron nach rechts versetzt. [22]

Beim *Pooling* werden die stärksten Merkmale der eingehenden Daten weitergegeben [22]. Kleine Änderungen der Eingabewerte haben durch *Pooling* keinen oder einen geringen Einfluss auf die Ausgabe. Ein Beispiel hierfür liefert die folgende Darstellung 2.8. Hierbei werden die stärksten Merkmale der Eingangsdaten durch das *Pooling* weitergeleitet. Obwohl in der unteren

Darstellung alle Eingangsdaten um eine Stelle nach rechts verschoben wurden, hält das Verfahren zwei Merkmale konstant.

Für die Klassifikation der Bilddaten in neuronalen Netzen setzen sich die letzten Schichten meist aus einer oder mehrerer vollständig verbundenen Schichten und einer *Softmax*-Schicht zusammen [22]. Die vollständig verbundene Schicht gibt einen Vektor mit K Elementen aus, wobei K für die Anzahl der ausgegebenen Klassen steht. Eine *Softmax*-Funktion repräsentiert grundlegend eine Wahrscheinlichkeitsverteilung des Eingabevektors K.

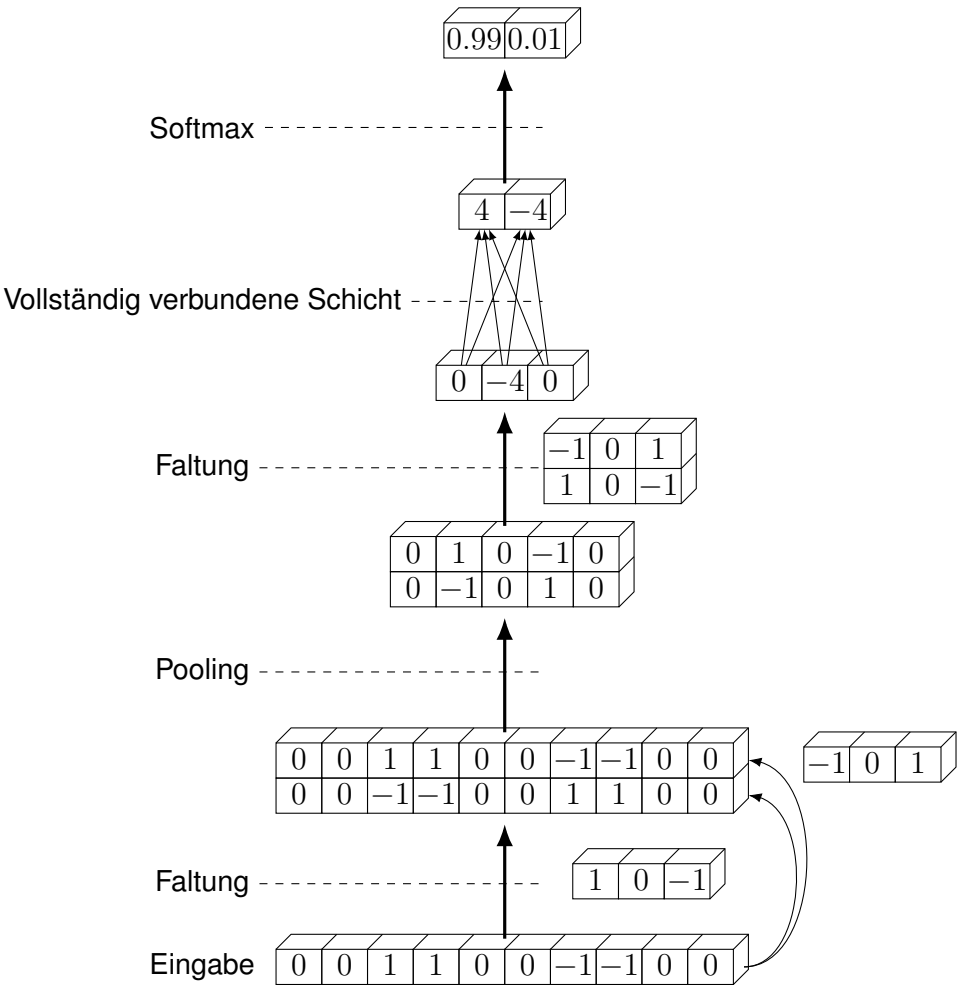


Abbildung 2.9: Fallbeispiel eines Faltungsnetzwerks. Dreidimensional dargestellte Boxen stellen Datenpunkte dar und bilden in der horizontalen Anordnung jeweils eine Schicht. Die Prozesse der Faltung und des Poolings sind beispielhaft dargestellt. Die Flussrichtung der Bildinformationen ist durch Pfeile gekennzeichnet. [25]

Durch die *Softmax*-Schicht wird der Vektor in einem Zahlenbereich von null bis eins transformiert. Die Summe aller Elemente des Vektors ergeben eins. Jedes Element wird als Konfidenz der jeweiligen Klasse interpretiert. An dieser Stelle sind alle Daten vollständig bearbeitet und werden als Vektor aus dem CNN ausgegeben. In Abbildung 2.9 wird die Funktionsweise eines Faltungsnetzwerks anhand eines Beispiels dargestellt.

2.3 Vergleich möglicher Konvolutionsnetze

Die Vielfalt der Architekturen von Konvolutionsnetzwerken ist sehr umfassend. Jedes Netz unterscheidet sich in der jeweiligen Architektur der Schichten. Durch die Änderung verschiedener Parameter, beispielsweise bei der Faltung oder bei dem Pooling, können CNNs im Einsatz jeweils anders reagieren. Bei der Auswahl des Modells sind die Gesichtspunkte Bearbeitungszeit, Genauigkeit und je nach Anwendungsfall der Speicherplatz entscheidend.

Auf dem Rechner des ALFs laufen häufig Softwarekomponenten, die für die Realisierung der Fahraufgaben aufgerufen werden. Eine Rechenauslastung von maximal 60% auf dem Computer des ALFs wird hierbei durch den Aufruf der Komponenten erzeugt. Außerdem sind die *Kinect*-Kameras des ALFs in der Lage, bis zu 30 Bilder in der Sekunde aufzunehmen. Folglich wird eine Netzarchitektur verwendet, die Bilder möglichst schnell, mit einer hohen Genauigkeit verarbeitet und eine geringe Anzahl an Rechenoperationen benötigt. Für eine zukünftige Auslagerung auf ein eingebettetes System ist der Speicherplatz des Netzes sowie seine Rechenzeit von Bedeutung.

Für die Auswahl der Netzarchitektur werden state-of-the-art Lösungen hinzugezogen. In *Canzianis* wissenschaftlichen Beitrag [18] werden bekannte CNNs nach der Genauigkeit über die Anzahl der Rechenoperationen pro eingegebenem Bild in einem Diagramm aufgetragen. Als dritte Eigenschaft ist dort ebenfalls die Größe der Architekturen bemessen. Die höchsten Genauigkeiten erzielten hierbei die *ResNet*, *Inception* und *VGG* Architekturen [18]. *Canziani* veröffentlichte sein Paper im Jahr 2016. Ein Jahr später entwickelte *Howard* [5] die *MobileNet* Architektur. Sie zeichnet sich durch die ihre Geschwindigkeit bei teilweise höherer Genauigkeit im Vergleich zu bekannten Architekturen aus [5]. Weiterhin werden in diesem Abschnitt die gängigen Lösungen *R-CNN* und *SSD* als optionale Klassifikatoren präsentiert.

VGG

Die VGG Architektur wurde im Jahr 2015 von *Simonyan* und *Zisserman* [26] vorgestellt. Oft wird die Bezeichnung *VGG-xx* verwendet, wobei *xx* für die Anzahl der Konvolutionsschichten in Addition mit den vollständig verbundenen Schichten steht. Bei dieser Architektur werden ausschließlich sehr kleine 3×3 Filter für die Faltung verwendet [26]. Im Gegenzug setzten die Entwickler auf eine ausgeprägte Tiefe der Netzarchitektur. In der Publikation sind Netze bis zu einer Tiefe von 19 Schichten untersucht worden [26]. Es hat sich herausgestellt, dass die Veränderung der Tiefe eines Netzes über die Genauigkeit der Klassifikation bestimmt [26]. So ist das Netz in der Lage sehr hohe Genauigkeiten zu erzielen. Wiederum hängt die Anzahl der Schichten direkt mit dem Speicher- und dem Rechenaufwand eines Netzes zusammen. Ein typisches *VGG-16* Netz nimmt circa 530 MB Speicherplatz wegen seiner 138 Millionen Parameter ein [27].

ResNet

ResNet steht für *Residual Network* und wurde erstmals im Jahr 2015 durch das Paper von *He* veröffentlicht [28]. Diese Architektur fällt besonders durch Verbindungen auf, die es Bildinformationen ermöglicht Schichten zu überspringen [28]. So sind die Eingangswerte der Zwischenschichten nicht nur von der Ausgabe der vorherigen Schicht abhängig. In Abbildung 2.10 ist eine Prinzipdarstellung der Architektur gezeigt. Hierbei wird eine allgemeine Eingabe x in beliebig viele Zwischenschichten eingegeben, die eine Ausgabe $F(x)$ erzeugt. Außerdem wird x durch eine Verbindung mit $F(x)$ addiert.

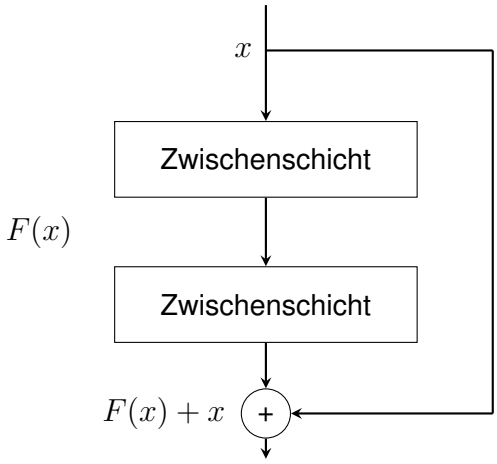


Abbildung 2.10: Markantes Merkmal eines *ResNets*. Die Flussrichtung der Bildinformationen ist als Pfeil dargestellt. Adaptiert aus [28]

ResNet Netze erreichen je nach Größe eine höhere Genauigkeit als die bereits erwähnten *VGG* Architekturen [27]. Die Größe der Netze beträgt hierbei 152 Schichten [28]. Trotz der entsprechenden Tiefe enthält ein derartiges CNN circa die Hälfte der Parameter im Vergleich zu *VGG* Netzen [27]. Dementsprechend liegt der Speicherbedarf bei circa 230 MB [27].

Inception/GoogleNet

Gemessen an der Top-1 Genauigkeit schneiden *Inception/GoogLeNet* Netze besser ab als *VGG* und *ResNet* Architekturen [18]. *Szegedy* erklärt die erste Version der *Inception* CNNs in seiner Publikation [29] aus dem Jahr 2015. Die Besonderheit hierbei ist der Einsatz multipler Kernel auf die jeweiligen Schichten [29]. Durch diesen Aufbau benötigt ein solches Netz keine vollständig verbundene Schicht zur Klassikation, da die Genauigkeit nur geringfügig beeinträchtigt wird [29]. Gleichzeitig erfolgt eine Einsparung eines Großteils der Parameter [29]. Bei *VGG* Architekturen befinden sich beispielsweise circa 90$\%$ aller Parameter in den vollständig verbundenen Schichten am Ende des Netzes. Die klassische *Inception* Architektur enthält insgesamt 27 Schichten, 22 davon enthalten Parameter [29]. Übliche *Inception* Netze sind circa 90 MB groß und verfügen über 23 Millionen Parameter [27].

MobileNet

Zu den bekanntesten state-of-the-art Lösungen gehört die *MobileNet* Architektur aus dem Jahr 2017. *Howard* beschreibt in seinem Paper die hohe Effizienz des Netzes hinsichtlich des Zusammenspiels zwischen Geschwindigkeit und Genauigkeit [5]. Anders als bei den bisher genannten Methoden besitzen einige Kernel der Konvolutionsschichten eine dritte Dimension [5]. Diese sind somit in der Lage tiefenorientierte Faltungsprozesse durchzuführen. Weiterhin wird der Rechenaufwand durch sogenannte Weiten- und Auflösungsmultiplikatoren reduziert [5]. In diesem Abschnitt wird näher auf die Funktionsweise eines *MobileNet*-CNNs eingegangen.

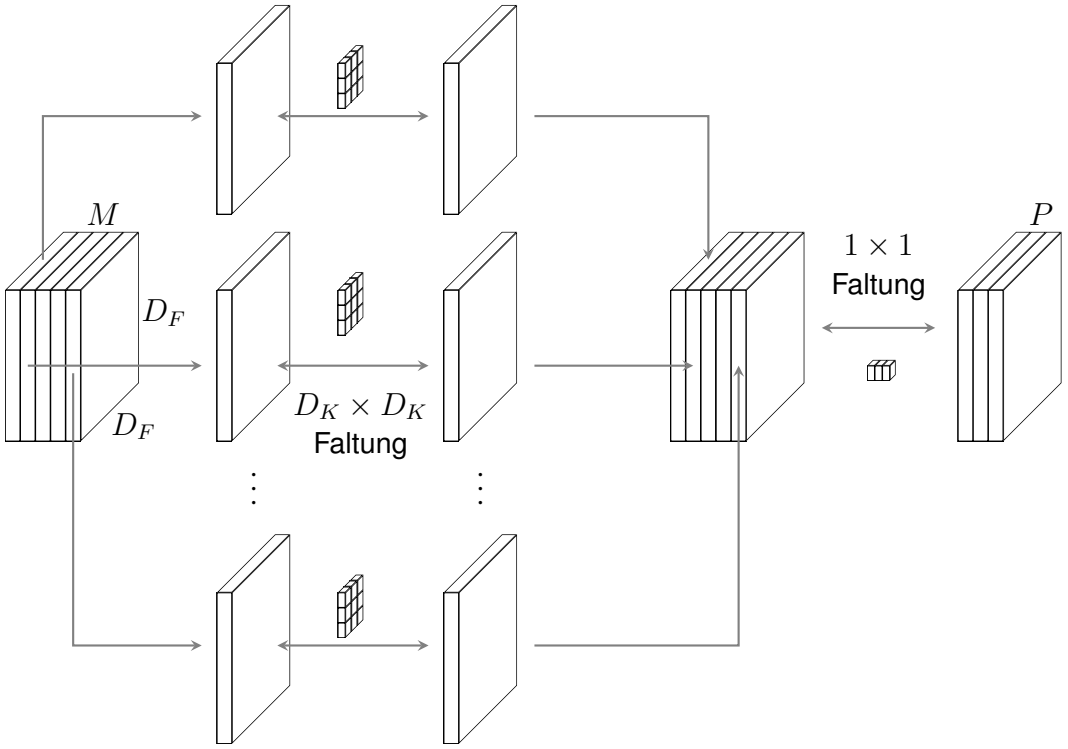


Abbildung 2.11: Prinzipdarstellung einer tiefenorintierten, trennbaren Konvolution. Große, rechteckige und dreidimensional dargestellte Boxen zeigen die Tiefenebenen eines Bidles. Die kleinen, feinmaschigen, dreidimensionalen Strukturen deuten Kernel an. [30]

Ein *MobileNet* Modell basiert grundlegend auf tiefenorientierter, trennbarer Konvolution [5]. Diese setzt sich aus Tiefenkonvolutionen und einer 1×1 Faltung, auch punktuelle Konvolution genannt, zusammen und ist schematisch in Abbildung 2.11 dargestellt [5].

Der Rechenaufwand R_p einer konventionellen Konvolutionsschicht kann mit Gleichung 2.5 beschrieben werden [5]. Hierbei wird die Dimension der Eingangsdaten durch D_F und die Tiefe durch M ausgedrückt. Die Größe des Kernels wird in der Gleichung durch die Variable D_K, sowie die Tiefe der Ausgangsdaten mit N dargestellt. Im Gegensatz zu üblichen Faltungsprozessen werden die Eingangskanäle während einer tiefenorientierten Konvolution lediglich gefiltert und nicht zusammengeführt [5]. Somit ist die Tiefe der Ausgangsdaten für die Berechnung des Rechenaufwands R_d bei einer tiefenorientierten Faltung zu vernachlässigen und ist in Gleichung 2.6 gezeigt.

$$R_p = D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \tag{2.5}$$

$$R_d = D_K \cdot D_K \cdot M \cdot D_F \cdot D_F \tag{2.6}$$

Durch die Zusammensetzung einer tiefenorientierten und einer punktuellen Faltung kann die Rechenbelastung der tiefenorientierten, trennbaren Konvolution durch die in Gleichung 2.7 gezeigte Addition errechnet werden. Die Kombination der beiden Methoden reduziert den Rechenaufwand um den in Gleichung 2.8 dargestellten Faktor $\frac{1}{N} + \frac{1}{D_K^2}$ [5]. In Zahlen ausgedrückt erreicht die *MobileNet* Architektur eine Reduktion der Rechenoperationen um den Faktor 8 bis 9 gegenüber konventionellen Faltungsmethoden [5].

$$R_{dp} = D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \tag{2.7}$$

$$R_{rdp} = \frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2} \tag{2.8}$$

Weiterhin nutzt *MobileNet* Weiten- und Auflösungsmultiplikatoren für die Optimierung des Netzes [5]. Der Weitenmultiplikator α reduziert sowohl die Tiefe M der Eingangsdaten durch αM als auch die Tiefe N der Ausgangsdaten durch αN [5]. Eine Minimierung der Dimension D_F der Eingangsdaten erfolgt durch den Auflösungsmultiplikator ρ durch ρD_F [5]. Beide Multiplikatoren reduzieren den Rechenaufwand nochmals um α^2 und ρ^2 [5]. Dieser kann somit mit folgender Gleichung dargestellt werden.

$$R_{dpm} = D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F \text{ mit } \alpha \in (0, 1]; \rho \in (0, 1] \quad (2.9)$$

Howard vergleicht in seinem Paper die *MobileNet* Architektur mit anderen Netztypen, wie zum Beispiel *Inception* oder *VGG 16*. Bei dem Vergleich ist zu erkennen, dass die verwendeten *MobileNet* CNNs deutlich weniger Parameter nutzen als übliche Netze. Trotzdem wurden lediglich minimale negative Auswirkung bezüglich der Genauigkeit festgestellt. Gleichzeit ist das Netz um ein Vielfaches schneller als herkömmliche Architekturen mit hoher Genauigkeit. Zudem bewirkt die Einsparung der enthaltenen Parameter einen geringeren Speicheraufwand.

Wie auch das bereits vorgestellte *Inception* Netz wurde auch die *MobileNet* Architektur weiterentwickelt. *MobileNet V2* kombiniert die tiefenorientierte, trennbare Konvolution mit der Technik des *ResNet* Netzes [31]. Bildinformationen werden etappenweise von verschiedenen Zwischenschichten analysiert und das Ergebnis mit den bestehenden Daten addiert [31]. Abbildung 2.10 zeigt die Technik beispielhaft. *MobileNet V2* arbeitet 2,5 Mal schneller als die Grundversion dieser Architektur bei minimaler Einsparung in der Genauigkeit [31].

R-CNN

R-CNN steht für *Region-based Convolutional Neural Network* und wurde 2014 im Paper von *Girshick* erstmals vorgestellt [32]. Diese Netzarchitektur kann sowohl als vollständiges Objekterkennungssystem als auch als Klassifikator eingesetzt werden [32, 33]. Mittlerweile gibt es durch diverse Modifikationen dieser Methode die Erweiterungen *fast R-CNN* und *faster R-CNN* [34, 35]. Alle zielen darauf ab, eine Echtzeitanalyse durchzuführen.

In der grundlegendsten Form besteht ein derartiges Netz aus drei Modulen [32]. Das erste Modul definiert bezüglich der Bildinformationen kategorieunabhängige Bereichsvorschläge, in denen sich Objekte befinden könnten [32]. Danach folgt ein Faltungsnetzwerk zur Merkmalsextraktion je Bereichsvorschlag [32]. Im dritten Modul erfolgt eine Klassifikation der Merkmale mithilfe eines *SVMs* [32]. Häufig wird für die drei Entwicklungsstufen des *R-CNN*, die *VGG* Architektur

zur Merkmalsextraktion verwendet [34]. In *Rens* Publikation über das *Faster R-CNN* wird eine mittlere Durchschnittsgenauigkeit von 0,788 für die Analyse auf den *PASCAL VOC 2017* Testdatensatz angegeben. Jedoch beläuft sich die Bearbeitungszeit pro Bild auf im Schnitt 200 ms bei einem Einsatz auf einer Grafikkarte [35].

SSD

Der *Single Shot Multibox Detector* (SSD) ist zunächst ein reiner Klassifikator. Dieser lässt sich beispielsweise mit einem *VGG-16* Netz für die Merkmalsextraktion zu einem Objekterkennungssystem kombinieren [36]. Anders als bei *R-CNN* wird beim *SSD* kein herkömmlicher Detektor als letztes Modul verwendet. Vielmehr handelt es sich hierbei um ein vollständiges Faltungsnetzwerk.

Im November 2016 präsentierte *Liu* in seinem Paper den *SSD*, ein modifiziertes *VGG-16* Netzwerk [36]. Hierbei wurden die vollständig verbundenen Schichten des *VGG-16* Netzes entfernt und durch Konvolutionsschichten ersetzt [36]. Der Name *SSD* unterteilt sich in *Single Shot*, *Multibox* und *Detector*. Anhand dessen lässt sich die Funktionsweise dieser Methode erklären. *Single Shot* drückt aus, dass die Architektur mit nur einer Bildanalyse auskommt und die Objekterkennung durchführen kann. Die *Multibox* Methode wurde von *Szegedy* entwickelt [37]. Diese gibt klassenbasierte Vorschläge zur Objekterkennung in Form von Begrenzungsrahmen aus [37]. Der Aufbau ähnelt in der Grundidee der bereits erwähnten *Inception* Architektur. Die *SSD300* Netzarchitektur erreicht im PASCAL VOC Test aus *Lius* Veröffentlichung einen mAP-Wert von 74.3 bei 59 FPS [36].

In der Publikation von *Sandler* wird eine erweiterte Form des *SSD*-Klassifikators eingeführt [31]. Bei der sogenannten *SSDLite*-Struktur werden die bereits erwähnten Konvolutionsschichten ausgetauscht. Tiefenorientierte, trennbare Konvolution wird stattdessen für eine Steigerung der Geschwindigkeit bei minimaler Regression der Genauigkeit eingesetzt [31].

2.4 Zustandsautomat

In der vorangegangenen Bachelorarbeit werden diverse Modi beschrieben, die den Aufruf von unterschiedlichen ROS-Knoten voraussetzen [4]. Aufgrund der Analogie zwischen den

beschriebenen Modi und der Zustände eines Zustandsautomaten wird die Nutzung eines solchen Automaten begründet. Im Folgenden wird auf die Eigenschaften eines endlichen Automats eingegangen.

Im Allgemeinen geht es bei einem Zustandsautomaten um die Beschreibung von Zuständen (engl. States) eines Objekts [38]. Dabei stellt das Objekt meist das Gesamtsystem dar, etwa einen Getränkeautomat oder wie in dieser Masterarbeit ein autonomes Fahrzeug [38]. States sind durch Bedingungen verknüpft und lösen während sogenannter Ereignisse eine Transition aus, die den Wechsel des Zustands nach sich zieht [38]. Weiterhin bilden die Zustände in ihrer Gesamtheit den Lebenszyklus des Objekts [38]. Ein Getränkeautomat befindet sich bekanntermaßen beim Eintreffen eines Kunden in einer Art Bereitschaft. Übertragen auf die Theorie eines Zustandsautomaten wäre dies ein Bereitschaftszustand. Die Auswahl des Getränks und die Eingabe des entsprechenden Geldbetrags können beispielhaft als Ereignisse interpretiert werden. Somit wird eine Transition durchgeführt und der Zustand der Getränkeausgabe ausgelöst. Wurde das Getränk ausgegeben und entnommen, geschieht der Wechsel in den Bereitschaftszustand und der beschriebene Zyklus ist komplettiert.

Seit dem Bestehen der endlichen Automaten haben sich in der Praxis zwei Typen durchgesetzt [39]. *Mealy*- und *Moore*-Automaten unterscheiden sich grundlegend in ihrem Verhalten und können durch folgende Gleichungen beschrieben werden.

$$\epsilon_{t+1} = \phi(\zeta_t, \epsilon_t) \text{ mit t} \in \mathbb{N} \tag{2.10}$$

$$\gamma_t = \psi(\zeta_t, \epsilon_t) \text{ mit t} \in \mathbb{N} \tag{2.11}$$

Durch die Gleichungen 2.10 und 2.11 wird das Verhalten eines Mealy Zustandsautomaten beschrieben. Die Transitionsfunktion ϕ und die Ausgabefunktion ψ des Mooreautomats stehen jeweils in Abhängigkeit von ζ_t, der aktuellen Eingabe, und ϵ_t, dem aktuellen Zustand. Mithilfe der Transitionsfunktion lässt sich der Zustand ϵ_{t+1} bestimmen, der im folgenden Zeitschritt $t + 1$ angestrebt wird. Der Ausgang des *Mealy*-Automaten wird durch γ_t ausgedrückt. Dieser hängt genau wie die Transitionsfunktion von der Eingabe und dem Zustand zum Zeitpunkt t ab. Das Verhalten eines *Moore*-Automaten wird mathematisch durch die Gleichungen 2.12 und 2.13

ausgedrückt. Im Vergleich zum Verhalten eines *Mealy*-Automaten ist zu erkennen, dass die Ausgangsfunktion ψ lediglich vom Ausgang zum Zeitpunkt t abhängig ist. [39]

$$\epsilon_{t+1} = \phi(\zeta_t, \epsilon_t) \text{ mit t} \in \mathbb{N} \tag{2.12}$$

$$\gamma_t = \psi(\epsilon_t) \text{ mit t} \in \mathbb{N} \tag{2.13}$$

Eine Unterkategorie der finiten Automaten ist der hierarchische Zustandsautomat. Die Besonderheit hierbei ist, dass mehr als nur ein Zustand aktiv sein kann. Genauer sind alle Zustände aktiv, die bis zur Aktivierung des aktuellen Zustands aufgerufen worden sind. Diese sind bei dem hier beschriebenen hierarchisch aufgebauten Automaten ebenfalls aktiv. So besteht die Möglichkeit eines aufeinander aufbauenden Endzustands. [40]

2.5 Bestimmung von Positionskoordinaten

Während der Durchführung autonomer Fahr- bzw. Logistikaufgaben können diverse Probleme auftreten, die eine erfolgreiche Bearbeitung verhindern können. Beispielsweise können Türen geschlossen sein oder Gegenstände die geplante Route blockieren. Da das ALF nicht über die technischen Möglichkeiten verfügt, derartige Probleme selbstständig zu lösen, müssen umstehende Personen um Unterstützung gebeten werden. Für diese Zwecke ist die Kenntnis über die Position von erfassten Personen notwendig.

Anstehende Fahraufgaben werden, bedingt durch das Vorgängerprojekt, mithilfe des *Robot Operating Systems* gelöst [4]. Es besteht die Möglichkeit Personen mithilfe von Positionskoordinaten in das *ROS*-Netzwerk zu veröffentlichen. Dies ermöglicht dem Roboter entsprechende Orte anzufahren und eine Kommunikation aufzubauen.

Für die Bestimmung der Positionskoordinaten werden ein zweidimensionales Bild und die dazugehörigen Tiefeninformationen genutzt. Die Kenntnis über die Pixelposition des Objekts im Bild und den horizontalen Sichtwinkel der Kamera ist für die Positionsbestimmung notwendig.

Abbildung 2.12 zeigt ein Fallbeispiel für die Positionserkennung. Die z-Koordinate ist als Höhenkoordinate zu interpretieren. Diese ist für die Berechnung der Position nicht weiter relevant und dient lediglich zum besseren Verständnis der Abbildung 2.12. Der Bildsensor der Kamera befindet sich im Ursprung des gezeigten Koordinatensystems. Zwischen der Objektivebene und dem Bildsensor besteht eine absolute Distanz $p_y{}'$. Der horizontale Sichtwinkel β der Kamera spannt sich wie dargestellt auf. Das Objekt, dessen longitudinalen p_y und lateralen p_x Positionskoordinaten im Bezug zum Bildsensor bestimmt werden sollen, ist in der Abbildung mit einem Punkt gekennzeichnet und entsprechend beschrieben. Durch die Variable d wird die absolute Distanz des Bildsensors zum Objekt beschrieben.

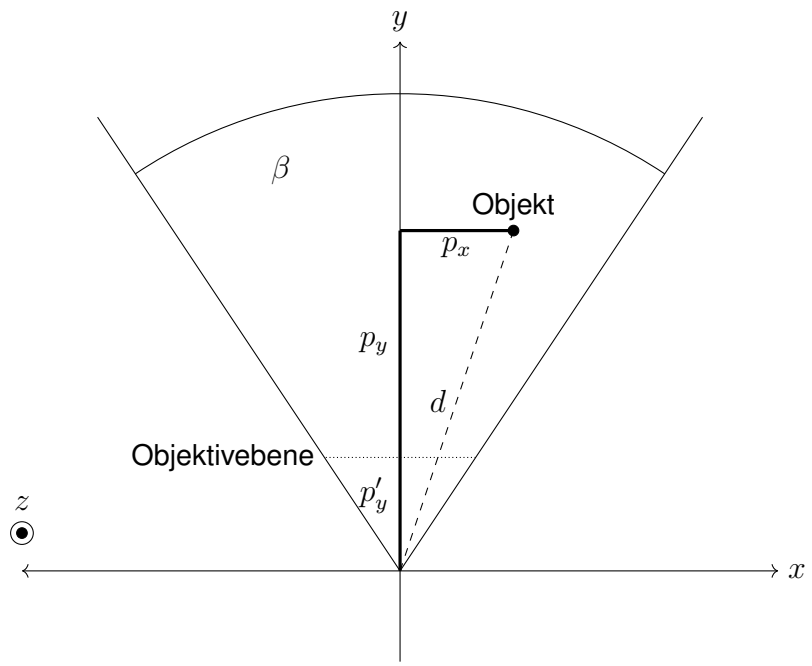


Abbildung 2.12: Funktionsprinzip der Positionsbestimmung durch ein zweidimensionales Bild und Tiefeninformationen von oben betrachtet. Der Bildsensor der Kamera befindet sich im Ursprung des Koordinatensystems. Der Abstand zwischen der als gepunktete Linie dargestellte Objektivebene und des Bildsensors wird mit $p_y{}'$ bezeichnet. Der Sichtwinkel der Kamera wird mit β beschrieben. Durch eine gestrichelte Linie ist hier die absolute Distanz d der Kamera zum Objekt gezeigt.

Zur Berechnung der longitudinalen Positionskoordinate p_y wird das mathematische Verhältnis aus Gleichung 2.14 mithilfe des zweiten Strahlensatzes definiert. Der Teil der Distanz d, der

durch den Bildsensor und der Objektivebene begrenzt wird, wird durch die Variable d' beschrieben. Da die Objektivebene im rechten Winkel zur y-Achse steht, kann d' durch Gleichung 2.15 beschrieben werden. Die Variable ${p_x}'$ steht für den Teil der Objektivebene, der durch die y-Achse und d eingegrenzt wird.

$$\frac{p_x}{d} = \frac{{p_x}'}{d'} \tag{2.14}$$

$$d' = \sqrt{({p_y}')^2 + ({p_x}')^2} \tag{2.15}$$

Das Verhältnis der halben Bildbreite $\frac{b}{2}$ zur Distanz ${p_y}'$ wird über den Tangens des halben Sichtwinkels der Kamera ausgedrückt. Gleichung 2.16 drückt das beschriebene mathematische Verhältnis aus. Durch die in Gleichung 2.17 aufgeführte Umstellung lässt sich die Distanz ${p_y}'$ berechnen. Diese wird in Gleichung 2.18 zur Berechnung der lateralen Positionskoordinate p_x eingesetzt.

$$\tan\left(\frac{\gamma}{2}\right) = \frac{\frac{b}{2}}{{p_y}'} \tag{2.16}$$

$${p_y}' = \frac{b}{2 \cdot \tan\left(\frac{\gamma}{2}\right)} \tag{2.17}$$

Gleichung 2.18 zeigt die Zusammensetzung der Berechnungen 2.14, 2.15 und 2.17. Die Zusammensetzung der Gleichungen erfolgte aufgrund der einzusetzenden Werte. Diese sind in der Praxis häufig bekannt oder können mit grundlegenden Mitteln berechnet oder ermittelt werden. Über den Hypotenusensatz erfolgt die Berechnung der longitudinalen Positionskoordinate p_y wie in Gleichung 2.19.

$$p_x = \frac{{p_x}'}{d'} \cdot d = \frac{{p_x}' \cdot d}{\sqrt{({p_y}')^2 + ({p_x}')^2}} = \frac{{p_x}' \cdot d}{\sqrt{\left(\frac{b}{2 \cdot \tan(\frac{\gamma}{2})}\right)^2 + ({p_x}')^2}} \tag{2.18}$$

$$p_y = \sqrt{d^2 - (p_x)^2} \tag{2.19}$$

3 Konzeptionierung

In diesem Kapitel wird die Konzeptionierung der Personenerkennung sowie die des Zustandsautomaten für das autonome Logistikfahrzeug erläutert. Wie auch in der vorangegangenen Bachelorarbeit wird die Anforderungserhebung mit der *Conceptual design specification technique for the engineering of complex Systems* (CONSENS)-Methode durchgeführt [4]. Die dadurch erhobenen Anforderungen an die beschriebenen Teilsysteme wurden im Lastenheft A.1.3 als Anforderungsliste festgehalten. Die Komponenten für die Entwicklung sind im Umfeldmodell und den entsprechenden Wirkstrukturen dargestellt.

3.1 Anforderungserhebung mit CONSENS

In Abbildung 3.1 ist das erweiterte Umfeldmodell des ALFs zu sehen. Das ursprüngliche Umfeldmodell ist in der entsprechenden Bachelorarbeit wiederzufinden [4]. Der Aufbau des Entwurfs besteht im Allgemeinen aus hellblauen und gelben Hexagonen, die Wirk- und Umfeldelemente repräsentieren. Der Informationsfluss zwischen den Elementen ist durch gestrichelte Pfeile gekennzeichnet.

Das ALF gilt als Kern des Gesamtsystems und ist deswegen im Modell mittig dargestellt. Es interagiert sowohl mit Wirk- als auch mit Umfeldelementen, wie in Abbildung 3.1 gezeigt. Zur Umsetzung von Fahraufgaben wird zusätzlich zur Personenerkennung ein Zustandsautomat entwickelt. Der Zustandsautomat ist ein Teilsystem, das als Erweiterung des Umfelds gilt. Im Laufe dieses Kapitels wird näher auf den technischen Hintergrund eingegangen. Mittels der von Herrn Dittmann entwickelten Sprachverarbeitung wird der Automat gesteuert. Diese kann, wie in Abbildung 3.1 gezeigt, sowohl auf dem Rechner des ALFs als auch auf dem integrierten *Raspberry Pi* ausgeführt werden [7].

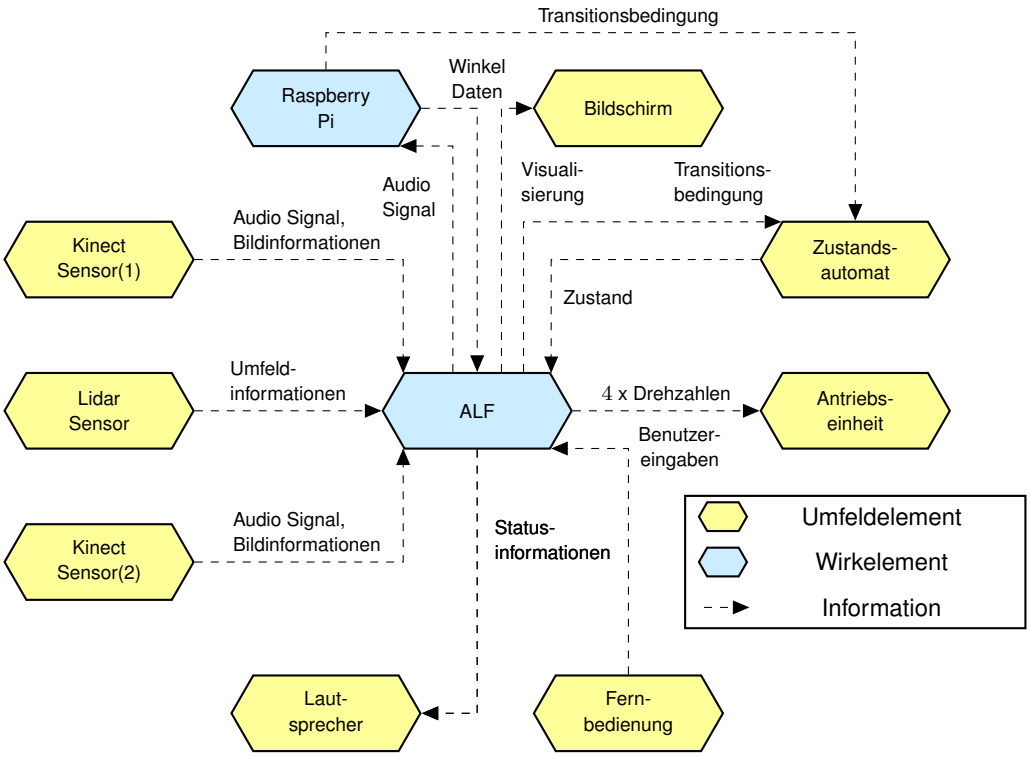


Abbildung 3.1: Weiterentwickeltes Umfeldmodell des Systems ALF. Hierbei wurde das System um die Umfeldelemente des Zustandsautomaten und der integrierten Lautsprecher erweitert. [4]

Die Erweiterung um die Personenerkennung ist in der in Abbildung 3.2 gezeigten Wirkstruktur dargestellt. Eine Wirkstruktur repräsentiert den Inhalt eines Wirkelements. In diesem Fall wird das Wirkelement mit dem Titel ALF aus dem vorangegangenem Umfeldmodell 3.1 aufgeschlüsselt. Die hier gezeigte Wirkstruktur wurde um die Personenerkennung und die Sprachverarbeitung erweitert. Elemente mit blassen Farben sind für diese Masterarbeit nicht relevant und werden weiterhin nicht behandelt.

Für den Betrieb der Personenerkennung sind die Bildinformationen der integrierten *Kinect*-Kameras notwendig. Als Ausgabe werden zweidimensionale Positionen von erkannten Personen veröffentlicht. Dies kann in zukünftigen Projekten beispielsweise als Einfluss auf eine Navigationsanwendung zur Vermeidung von Personenkontakt verwendet werden.

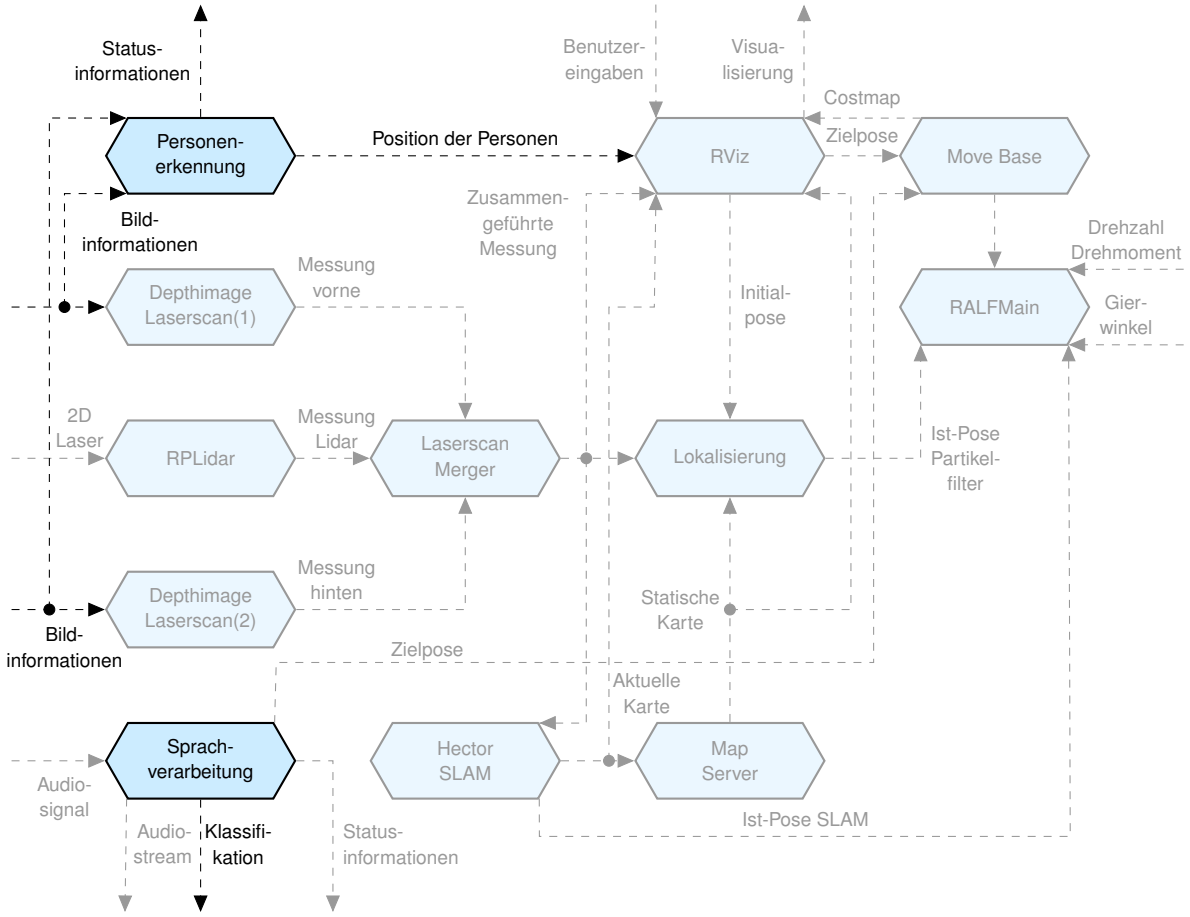


Abbildung 3.2: Wirkstruktur des ALFs. Die für diese Masterarbeit relevanten Wirkelemente und Informationsflüsse sind deckend dargestellt. Eine blasse Farbgebung deutet auf bereits implementierte Teilsysteme hin, die nicht weiter relevant sind, jedoch für ein besseres Verständnis des Gesamtsystems dienen.

Für eine Interaktion mit anwesenden Personen werden Statusinformationen ausgegeben, wie oben links in Abbildung 3.2 dargestellt. Im Umfeldmodell wird die Klassifikation als Transitionsbedingung interpretiert und ist dort als solche gekennzeichnet. Somit hat die Ausgabe der Sprachverarbeitung durch die Klassifikation der Sprache einen Einfluss auf den Zustandsautomaten. In diesem Kapitel wird genauer auf die Konzeptionierung des Automaten eingegangen.

3.2 Konzept und Aufbau der Personenerkennung

Die Personenerkennung gilt als eine von zwei in dieser Masterarbeit entwickelten Erweiterungen und gleichzeitig als Hauptthema. Ziel hierbei ist die eindeutige Unterscheidung und Wiedererkennung von Personen durch das System. In diesem Abschnitt wird die Personenerkennung in ihrer Struktur und Umsetzung näher erläutert. Weiterhin werden konkurrierende Softwarekomponenten gegenübergestellt.

Grundlage für eine Personenerkennung ist die eindeutige Identifikation eines Menschen. Folglich muss ein System in der Lage sein, äußerliche Merkmale festzustellen, die langfristig wiederzuerkennen sind. Wie bereits in Kapitel 1 beschrieben, sollen Personen langfristig identifiziert werden. Aufgrund dessen musste das äußere Erscheinungsbild der Person, wie zum Beispiel die Kleidung oder die Körperhaltung dieser als Identifikationsmerkmal ausgeschlossen werden. Derartige Merkmale ändern sich ständig und können somit nicht sicher zugeordnet werden. Das Gesicht eines Menschen verändert sich mit fortlaufendem Alter. Es wird angenommen, dass sich die Benutzergruppe des ALFs auf Studenten und Mitarbeiter des Hochschule Bochum beschränkt. Mit dieser Annahme besteht die Benutzergruppe des Fahrzeugs aus Personen mit einem Alter zwischen 20 und 65 Jahren. Durch diese Beschränkung könnten individuelle Personen schätzungsweise mindestens 5 Jahre durch das Gesicht erkannt werden. Die eindeutige Erkennung eines Menschen setzt also voraus, dass das Gesicht einer Person von der Kamera des ALFs erfasst wird.

3.2.1 Wirkstruktur der Personenerkennung

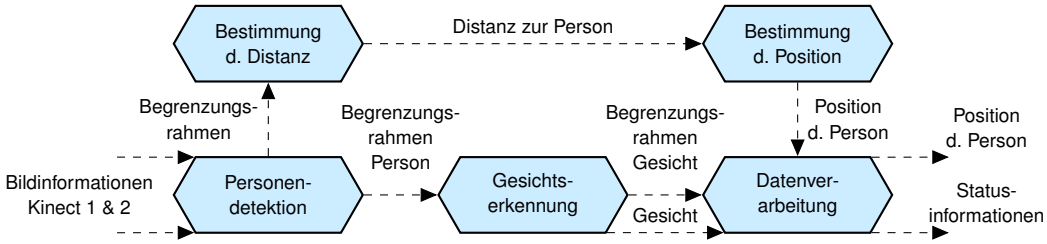


Abbildung 3.3: Wirkstruktur der Personenerkennung. Die Wirkelemente stellen hierbei Softwareelemente der Personenerkennung dar.

In Abbildung 3.3 ist die Wirkstruktur der Personenerkennung dargestellt. Die Bildinformationen der *Kinect*-Kameras korrespondieren mit den im Umfeldmodell 3.1 gezeigten Pfeilen. Eine der Gesichtserkennung vorangegangene Personendetektion gibt die Bildkoordinaten der erkannten Menschen aus. Dies wird in Abschnitt 3.2.4 näher erläutert.

In den Bildinformationen der integrierten *Kinect*-Kameras befinden sich unter anderem auch Tiefeninformationen passend zum gelieferten Farbbild. Somit wird die Distanz und die daraus resultierende Position einer Person ebenfalls errechnet. Die Abschnitte 2.5 und 3.2.4 führen die Berechnung und die Methodik genauer aus.

Eine Datenverarbeitung sorgt letztendlich für das Abspeichern verwertbarer Informationen einer erkannten Person. Zukünftige Projekte am ALF können diese Eigenschaften für weitere Entwicklungen nutzen.

3.2.2 Konzept der Bildverarbeitung

In Abschnitt 2.2.2 wurde bereits erwähnt, dass state-of-the-art Lösungen häufig auf künstliche, neuronale Netze zurückgreifen. Insbesondere werden Konvolutionsnetze bevorzugt, wenn es um eine Umsetzung einer Objekterkennung geht. Auch in dieser Masterarbeit liegt das Hauptaugenmerk auf CNNs. Jedoch werden die in Abschnitt 2.2.1 beschrieben Methoden weiterhin berücksichtigt, indem ihre Leistung mit der der Faltungsnetze verglichen wird. Im Folgenden werden Vergleiche zwischen den in Abschnitt 2.2.2 genannten künstlichen neuronalen Netzen gezogen. Weiterhin wird eine Auswahl für die praktische Anwendung am ALF getroffen.

Tabelle 3.1: Vergleich verschiedener Netzarchitekturen in den Eigenschaften Parameter, Speicherplatz, Tiefe, Top-5 Fehler und Top-1 Fehler. Es ist zu beachten, dass die Informationen teilweise je nach Trainings- und Testdatensatz variieren können. Die Top-x Genauigkeit wurde anhand der *ImageNet*-Datenbank festgestellt [27]. Die Tiefe beschreibt hierbei die topologische Tiefe des Netzes einschließlich jedes Layertyps [27].

Eigenschaften	Netze			
	VGG-16	ResNet50	InceptionV3	MobileNet
Parameter (in Millionen)	138.357	25.636	23.851	4.253
Speicherplatz (in MB)	528	98	92	16
Tiefe (in Layer)	23	-	159	88
Top-5 Fehler (in $\%$)	0.901	0.921	0.937	0.895
Top-1 fehler (in $\%$)	0.713	0.749	0.779	0.704

Tabelle 3.1 zeigt den Vergleich unterschiedlicher Netzarchitekturen. Hierbei ist zu beachten, dass die Angaben je nach Trainings- und Testdatensatz variieren können. Die nähere Auswahl der gezeigten CNNs *VGG-16*, *ResNet50*, *InceptionV3* und *MobileNet* wurde bereits in Abschnitt 2.3 begründet. Die Gegenüberstellung der Parameteranzahl und des daraus resultierenden Speicherplatzes zeigt deutlich, dass die *MobileNet*-Architektur hierbei die kleinsten Werte aufweist. Bei der Betrachtung der Genauigkeiten im Hinblick auf die Tiefe der Netze fällt auf, dass sich offensichtlich keinerlei Relation beschreiben lässt. *InceptionV3* weist bei den Top-x Fehlern die höchsten Werte auf und hat somit die höchste Genauigkeit.

Im Bezug auf die entwickelte Wirkstruktur 3.3 zielt der Anwendungsfall darauf ab, Personen sicher und schnell zu erkennen. Die Aufgabe der reinen Personendetektion erfordert jedoch keine sicherheitsrelevanten Eigenschaften des Netzes, sodass geringe Abstriche bei der Genauigkeit in Kauf genommen werden können. Die *MobileNet*-Architektur zeigt das gesuchte Zusammenspiel aus geringem Speicherbedarf und verhältnismäßig hoher Genauigkeit. Durch den Einsatz von *MobileNet* wird im Vergleich zu *InceptionV3* eine Einsparung des Speicherplatzes um circa 82 $\%$ bei einem Verlust der Top-1 Genauigkeit um lediglich 10 $\%$ erreicht. Für eine zukünftige Anwendung auf einem eingebetteten System eignet sich die Umsetzung mithilfe von *MobileNet*.

Tabelle 3.2: Darstellung unterschiedlicher Kombinationen von Merkmalsextraktoren und Detektoren. Die Werte wurden mithilfe der *Pascal VOC2007 test* Benchmark im Paper von Yuxi Li ermittelt [41]. Als *Backbone* wird in der Fachsprache der vorgeschaltete Merkmalsextraktor bezeichnet.

Netze (Backbone)	mAP	FPS	Bildgröße
Faster R-CNN (VGG-16)	73.2	7	600×1000
SSD (VGG-16)	77.2	46	300×300
SSD (MobileNet)	68.0	59	300×300

Bereits in Abschnitt 2.3 wurde die Kombination diverser Architekturen zur Merkmalsextraktion mit entsprechenden Detektoren beschrieben. In Tabelle 3.2 werden derartige Verknüpfungen und ihre Eigenschaften gezeigt. FPS steht für *frames per second* und besagt, wie viele Bilder pro Sekunde verarbeitet werden können. In der Fachsprache wird der Teil der Merkmalsextraktion eines Netzes häufig auch als *Backbone* bezeichnet. Die Tabelle zeigt die Architekturen *Faster R-CNN* und *SSD* mit jeweils einem *VGG-16* Netz als *Backbone* und das *MobileNet SSD* aus Zhangs Paper [42]. Zhang wendet dort ebenfalls die *MobileNet-SSD* Architektur auf einem eingebetteten System an und erreicht eine Rechenzeit von 1.13 FPS. Wie bereits in Abschnitt 2.3 beschrieben können am ALF maximal bis zu 60 Bilder in der Sekunde eingehen. Mit 7 FPS und einer geringeren Genauigkeit als der SSD (VGG-16) ist der Einsatz des Faster R-CNN ausgeschlossen. Die *MobileNet-SSD* Architektur erreicht laut der präsentierten Benchmark nahezu 60 FPS und nimmt aufgrund des *MobileNet Backbones* weniger Speicherplatz ein als das *SSD (VGG-16)* Netz [41].

Das Kapitel 4 wird sich mit der tatsächlich gemessenen Leistung der behandelten Netze im Feld beschäftigen. Hierbei wird der Einsatz auf verschiedenen Hardwareplattformen getestet. Die durch die wissenschaftlichen Paper erlangten Informationen zeigen jedoch, welche Netzarten und -architekturen hierfür in Betracht gezogen werden können. Über die *SSDLite*-Struktur gibt es keine aussagekräftigen Benchmarks. Auch diese Architektur wird in der Evaluation in Kapitel 4 berücksichtigt.

3.2.3 Entwicklung eines neuronalen Netzes

Die in Abschnitt 3.2.2 behandelten neuronalen Netze wurden auf ihre Leistungsfähigkeit passend zum Anwendungsfall am ALF untersucht. Weiterhin kann die Struktur eines gewählten KNNs optimiert werden. Der Vorgang wird in diesem Abschnitt beschrieben.

Mittlerweile gibt es auf Seiten wie zum Beispiel *Tensorflow.org* oder *Keras.io* viele Möglichkeiten zum Download vortrainierter Netze. Viele der angebotenen KNNs sind je nach Anwendungsfall diversen Kategorien zugeordnet. Unter anderem finden sich dort auch KNNs zur Objekterkennung. Die meisten sind darauf ausgelegt, multiple Objekte zu erkennen. Somit lassen sich beispielsweise mit einem Netz nicht nur Personen, sondern auch Flugzeuge, Autos und viele weitere Gegenstände des Alltags erkennen.

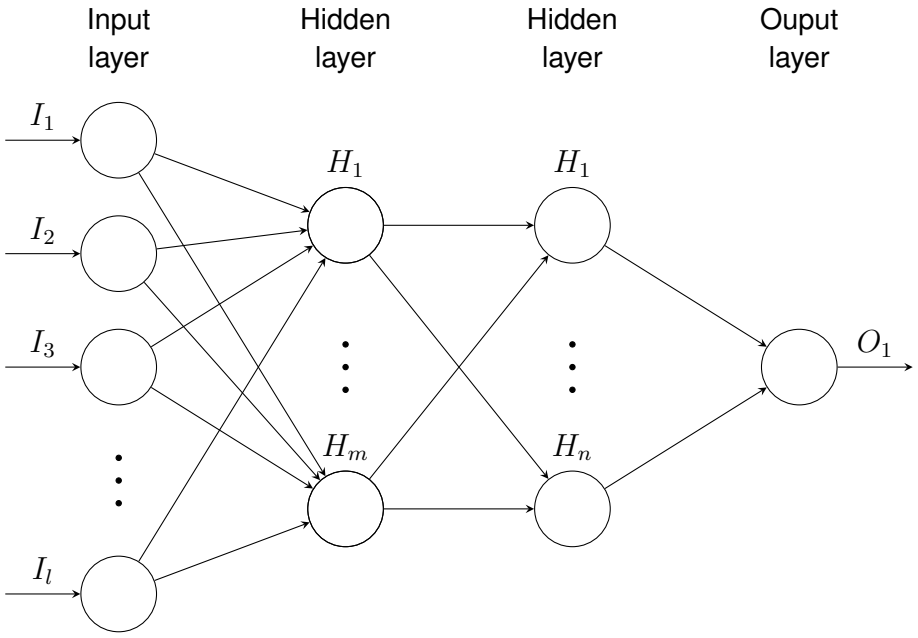


Abbildung 3.4: Darstellung des entwickelten neuronalen Netzes. Neuronen sind hier als Kreise gekennzeichnet. Fortsetzungen sind hier sinngemäß mit drei aufeinander folgenden Punkten dargestellt. Die Besonderheit ist die Reduktion der Ausgabeschicht auf lediglich einen Datenpunkt. Adaptiert aus [12]

Für den Einsatz am ALF ist die Erkennung anderer Objekte jedoch unerheblich. Zwei Lösungsansätze werden im Rahmen dieser Masterthesis genauer betrachtet. Für den ersten Lösungsansatz wird das entsprechende KNN unverändert eingesetzt. Ein Sortieralgorithmus

beschäftigt sich nach der Analyse des Bildes mit dem Ausschluss nicht relevanter Klassen. In Abbildung 3.4 ist eine weitere Lösung dargestellt. Die Ausgabeschicht eines Netzes wird so verändert, dass lediglich ein Ausgangsneuron vorhanden ist. Dieses ist bekanntlich in der Lage, auf eine Klasse trainiert zu werden. Die restlichen Klassen werden somit verworfen und das Neuron wird ausschließlich auf die Klasse *Person* trainiert. Diese Vorgehensweise spart nicht nur die Einbindung eines Sortieralgorithmus, sondern auch Speicherplatz ein. Durch die Elimination der unbedeutsamen Neuronen werden folglich auch Parameter gelöscht, die gewissen Speicher einnehmen und Rechenoperationen vorraussetzen. Somit wird auch eine schnellere Rechenzeit erwartet.

Zur Umsetzung des KNNs wird *Tensorflow* der Firma *Google* verwendet. *Tensorflow* bietet neben der Möglichkeit des Trainings vortrainierter Netze, Lösungen zur Implementierung auf eingebetteten Systemen [43]. Diesbezüglich haben die Entwickler einen eigenen Framework namens *Tensorflow Lite* ins Leben gerufen. Dieser Framework ist insbesondere für die Entwicklung mobiler Applikationen ausgelegt [44]. *Tensorflow Lite* bietet beispielsweise die Möglichkeit, Netze einer sogenannten Quantisierung zu unterziehen [44]. Dies ermöglicht einen noch kleineren FPS-Wert bei geringer Einsparung der Genauigkeit. *Xus* Paper zeigt, dass *Tensorflow* und *Tensorflow Lite* die meistgenutzten Frameworks sind, wenn es um mobile Applikationen geht. Weiterhin unterstützt der *Tensorflow* die *NVIDIA CUDA Deep Neural Network* (cuDNN) Bibliothek [43]. *NVIDIA cuDNN* ist für Grafikprozessoren und die Arbeit mit künstlichen neuronalen Netzen optimiert [43]. Außerdem unterstützt *Tensorflow* die Betriebssysteme *Linux*, *Mac OS X* und *Windows* [43]. Somit kann der integrierte Computer des autonomen Logistikfahrzeugs als ausführende Instanz verwendet werden. Für Trainingsprozesse eignet sich in vielen Fällen ein Rechner mit einer leistungsstarken Grafikeinheit. Die Praxiserfahrung dieser Masterarbeit zeigte, dass eine Grafikkarte den Trainingsprozess um den Faktor 12 beschleunigt. Jedoch gibt es Anwendungsfälle, die durch die Verwendung der zentralen Recheneinheit des Computers beschleunigt werden [45].

Als Datensatz für das Training der Netze wird der *Common Objects in Context* (COCO) Datensatz der Firma *Microsoft* verwendet. Dieser Datensatz enthält circa 330000 Bilder mit diversen alltäglichen Objekten [46, 47]. Davon sind laut eigener Aussagen über 200000 Bilder mit sogenannten *Annotations* oder auch *Labels* (deutsch: Anmerkungen/Etiketten) versehen [46]. Die Daten sind in Trainings-, Evaluations- und Testdaten unterteilt. Letzteres ist jedoch nicht etikettiert und somit für diese Masterarbeit nicht relevant. Die Trainingsdaten von dem

COCO-Datensatz aus dem Jahr 2017 stellt knapp 65000 etikettierte Bilder von Personen bereit. Circa 5000 Bilder sind im Evalutaionsdatensatz enthalten.

Für den Lernprozess wird somit der Trainingsdatensatz in Trainings- und Testdaten unterteilt. So kann ein Teil der eigentlichen Trainingsdaten durch die vorhandenen Etiketten als Testdatensatz verwendet werden. Als zweiten Test wird ein eigener Datensatz erstellt. Hierbei werden 160 Bilder aus dem Einsatzumfeld des ALFs aufgenommen und etikettiert. In Kapitel 4 werden bereits besprochene Netze mit dem eigenen und dem *COCO*-Datensatz evaluiert.

3.2.4 Funktionsweise der Personenerkennung

Der Kern dieser Arbeit ist die Personenerkennung im praktischen Kontext des im Kapitel 1 beschriebenen autonomen Logistikfahrzeugs. In Abschnitt 3.1 wurden bereits alle Schnittstellen zu verbauten Hardware- und Softwarekomponenten präsentiert. Das vollständige System der Personenerkennung ist im Folgenden erklärt.

Wie bereits in Kapitel 1 dieser Masterarbeit beschrieben, wird die Personenerkennung am ALF mithilfe der Bildinformationen von zwei *Kinect*-Kameras betrieben. Als Programmiersprache wird im Zuge dieser Masterarbeit *Python* verwendet. Mit Softwarepaketen wie beispielsweise *OpenCV* oder *Pillow* bietet *Python* ein großes Spektrum an Softwarelösungen für die Bildverarbeitung. Bekannte Frameworks wie *Tensorflow* oder *Keras* unterstützen *Python* [48, 49].

Wie auch in dem Projekt der Bachelorarbeit liefert das integrierte *ROS*-Netzwerk die Bildinformation der Kameras [4]. Die Schnittstelle zwischen *ROS* und *Python* bietet die Möglichkeit, eingehende Bilder sowohl parallel als auch seriell zu bearbeiten. Jedoch ist zu beachten, dass die Recheneinheit eines Computers mit der Bildverarbeitung eines Bildes stark belastet sein kann. Der beschriebene Effekt variiert je nach verwendeter Hardwareplattform. Demzufolge ist auf einem eingebetteten System eine deutlich höhere Belastung zu erwarten als auf dem Computer des ALFs. Aufgrund dessen wird bei der Bildverarbeitung der Personenerkennung auf eine serielle Bearbeitung gesetzt. So wird ausgeschlossen, dass zwei eingehende Bilder gleichzeitig analysiert werden.

Zu Beginn arbeitet das entwickelte System in einem reduzierten Modus. Hierbei werden Pausen mit einer gewünschten Dauer zwischen Bildverarbeitungsprozessen eingelegt, bis ein relevantes Bild erkannt wird. Als relevant werden Bilder eingestuft, die eine Person enthalten. Erst dann arbeitet die Personenerkennung mit der maximalen Geschwindigkeit. Der reduzierte Modus erspart weitere Rechenkapazität des Computers für parallel laufende Prozesse.

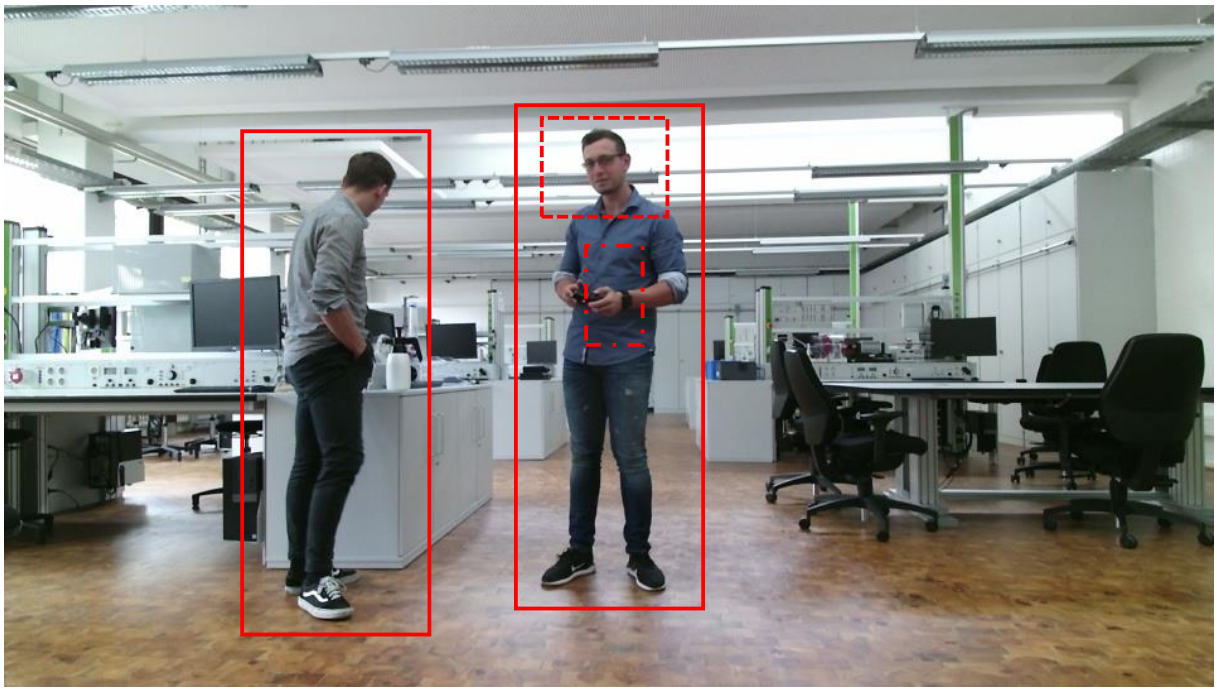


Abbildung 3.5: Konzept eines Fallbeispiels der Personenerkennung. Zu sehen sind zwei Personen im Labor für Antriebstechnik der Hochschule Bochum. Rote Rechtecke zeigen den Interessensbereich der Personendetektion. Der Begrenzungsrahmen der Gesichtsdetektion ist gestrichelt dargestellt. Das als Strichpunkt präsentierte Rechteck deutet den Bereich der Distanzmessung an.

In Abbildung 3.6 wird der Ablauf der Personenerkennung in Form eines Programmablaufplans dargestellt. Zu Beginn der Analyse gelangt jedes Bild in das verwendete künstliche neuronale Netz. Je nach Anzahl der erkannten Person werden korrespondierende Begrenzungsrahmen ausgegeben, die die Position des Interessensbereichs im Bild beschreiben. Dieser Vorgang ist beispielhaft in Darstellung 3.5 präsentiert. Die dort abgebildeten Personen werden in diesem Fall von einem roten Begrenzungsrahmen umrandet. Wird keine Person erkannt, verfällt das Programm wieder in den bereits beschriebenen, reduzierten Modus. Die extrahierten Begrenzungsrahmen gelten im Falle einer Personendetektion als Interessensbereich für die Gesichtserkennung.

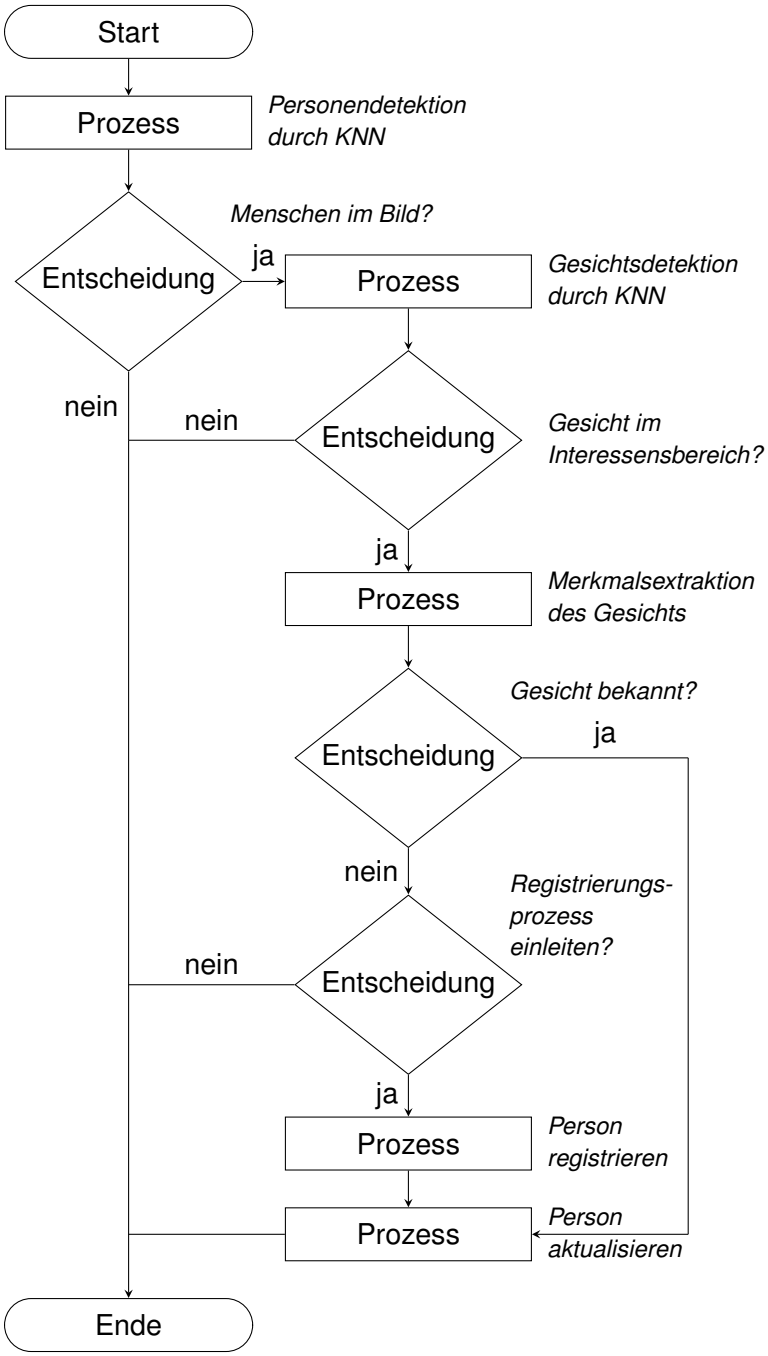


Abbildung 3.6: Prozessablaufplan der Personenerkennung. Der Start sowie das Ende des Programms sind als Boxen mit Abrundungen dargestellt. Rechtecke zeigen Prozesse und Parallelogramme deuten eine Entscheidung im Programmablauf an. Die Flussrichtung der entsprechenden Informationen werden durch Pfeile präsentiert.

Das Pythonpaket *face-recognition* wird in diesem Projekt für die Gesichterkennung verwendet. Prinzipiell arbeitet das Paket in drei Schritten. Die in Abschnitt 2.2.1 erklärte *HOG*-Methode wird zur Extraktion des Begrenzungsrahmens des jeweiligen Gesichts angewendet [50]. Damit eine Person nicht unbedingt gerade in das entsprechende Aufnahmegerät schauen muss, werden durch die Software sogenannte Landmarken auf dem Gesicht verteilt [50]. Diese werden zur Rotation des Gesichts im Bild verwendet [50]. Das bearbeitete Bild des Gesichts wird mit einem neuronalen Netz analysiert, das von den Entwicklern von *OpenFace* bereitgestellt wird [50]. Entwickler *Adam Geitgey* gibt für das im Paket verwendete Netz eine Genauigkeit von 99,38% an [51]. Hinsichtlich der in Kapitel 2 besprochenen, möglichen Nutzungsszenarien der Personenerkennung ist eine hohe Genauigkeit notwendig. Nach eigener Recherche besitzen die von *OpenFace* bereitgestellten, neuronalen Netze zwischen 3,7 - 7,4 Millionen Parameter [52]. Die Größenordnung ähnelt folglich der einer herkömmlichen *MobileNet*-Architektur.

Außerdem bietet das *face-recognition* Paket eine Funktion zum Vergleichen und Unterscheiden von Gesichtsmerkmalen. Aufgrund der Genauigkeit und der verwendeten Netze wird das *face-recognition* Paket in dieser Masterarbeit für die Gesichtserkennung eingesetzt.

Es kann davon ausgegangen werden, dass sich das Gesicht einer Person im oberen Teil eines extrahierten Begrenzungsrahmens befindet. Aufgrund dessen wird der Interessensbereich entsprechend verkleinert. Somit muss die Gesichtserkennung nicht den vollständigen Bereich durchsuchen und es werden weitere Rechenoperationen eingespart. Sollte sich kein Gesicht im relevanten Bereich befinden, wird davon ausgegangen, dass die Person stark von der Kamera abgewandt ist. Somit ist keine eindeutige Identifikation möglich und es erfolgt ein Neustart des Programms.

Detektiert das Netz jedoch ein Gesicht, wird eine Merkmalsextraktion durchgeführt. Im Anschluss werden die extrahierten Merkmale mit denen der bereits abgespeicherten Gesichter verglichen. Wird kein übereinstimmendes Gesicht gefunden, leitet das System einen Registrierungsprozess ein. Je nach Einstellung der Häufigkeit wird ein Gesicht registriert, wenn es entsprechend oft hintereinander erkannt worden ist. Im Falle der Erkennung eines bekannten Gesichts werden Eigenschaften der erkannten Person, wie zum Beispiel die Gesichtsmerkmale oder die Position in der aktuellen Karte aktualisiert.

3.3 Konzept und Aufbau des Zustandsautomaten

Bisher wurden für den Aufruf der Fahrfunktionen in *ROS* ausführbare Dateien aufgerufen [4]. Hierbei musste der Benutzer darauf achten, diese Dateien nicht im geringen Zeitabstand und vor allem in der richtigen Reihenfolge aufzurufen [4]. Zu jedem Anwendungsfall gehören entsprechende Dateien.

Ein hierarchischer Zustandsautomat nach Mealy unterbindet die Probleme, indem sich der Endzustand durch zuvor aufgerufene Zustände zusammensetzt. Der Aufruf verschiedner *ROS*-Knoten in der korrekten Reihenfolge wird somit autark geregelt. Für die Steuerung des EAs wird die Spracherkennung aus der parallel laufenden Masterarbeit verwendet. Die Auswahl des Zustandsautomats wird im folgendem Abschnitt näher ausgeführt.

Die Problematik der Steuerung über Sprache ist die Extraktion der eigentlichen Aussage eines Satzes [7]. In der Masterarbeit von *Hannes Dittmann* werden aufgrund dessen Sprachbefehle kategorisiert [7]. Die KI ist in der Lage verschiedene Sätze einer für den Roboter relevanten Kategorie zuzuordnen. Diese werden als Transitionsbedingung für den EA genutzt. Weiterhin wird zwischen einer manuellen und einer autonomen Fahraufgabe unterschieden. Die Information wird dann als zweite Bedingung für den Zustandsautomaten verwendet. Anhand der technischen Fähigkeiten des Roboters wurden die Kategorienamen so gewählt, dass alle möglichen Handlungen abgedeckt sind. Der Zustandsautomat wurde so entworfen, dass er trotz der umfangreichen Fahrfunktionen des Roboters mit möglichst wenig Zuständen arbeitet. Außerdem deckt der Automat die Richtlinien nach Level 5 der in Kapitel 1 gezeigten Tabelle ab. Für die größtmögliche Effizienz hinsichtlich der Dimension und Funktionsweise des EAs wurde ein mathematisches Modell entworfen.

$$\vec{\epsilon} = \sum_{\epsilon_0}^{\epsilon_f} \begin{bmatrix} k_0 \\ k_1 \\ \dots \\ k_8 \end{bmatrix} \circ \begin{bmatrix} g_0(\epsilon) \\ g_1(\epsilon) \\ \dots \\ g_8(z) \end{bmatrix} \circ \begin{bmatrix} 1 \\ 1 \\ m_2(\epsilon_f) \\ m_3(\epsilon_f) \\ 1 \\ m_5(\epsilon_f) \\ 1 \\ m_7(\epsilon_f) \\ m_8(\epsilon_f) \end{bmatrix} \quad \text{für } \epsilon_f \in [0;8] \text{ und } \epsilon_0 = 0 \vee 1 \tag{3.1}$$

Anhand der Gleichungen 3.1 und 3.2 wird der mathematische Hintergrund des EAs erläutert. Der hierarchische Zustandsautomat ermöglicht einen finalen, sich aufbauenden Zustand $\vec{\epsilon}$. Dieser wird anhand der in Gleichung 3.1 dargestellten Summe berechnet. Beginnend von dem Anfangszustand ϵ_0 wird jeder mögliche ϵ bis zum finalen Zustand ϵ_f addiert. Bei der Addition geht es hauptsächlich darum, dem Anwendungsfall entsprechende *ROS*-Knoten aufzurufen. Zu einem Zustand fest zugehörige Knotengruppen werden in Gleichung 3.1 mit k bezeichnet. Dessen Index deutet auf die Zugehörigkeit des Knotens zum jeweiligen Zustand an. Hierbei werden die möglichen Endzustände wie in Abbildung 3.7 nummeriert.

Die Knotengruppe k_0 gehört zu dem Zustand *Stop* und leitet den risikominimalen Zustand ein. Dieser wird erreicht, indem alle *ROS*-Knoten inklusive des *ROS*-Netzwerks heruntergefahren werden. So kann sichergestellt werden, dass keine Nachrichten an die Motorsteuergeräte der vier verbauten Motoren gesendet werden. Der Zustand *Stop* kann nur verlassen werden, indem ein sicherer Wechsel in den Folgezustand *Warten* durch den Benutzer zugesichert wird. Die Abfrage erfolgt sowohl über die Tastatur als auch über Sprache. Alle folgenden Zustände enthalten der jeweiligen Anwendung entsprechende Knottengruppen. Eine Auflistung aller Zustände inklusive der Knotengruppen ist im Anhang A.1 in der Abbildung A.1 aufgeführt.

$$g(\epsilon) = \begin{cases} 1 \text{ für } \epsilon = n \\ 0 \text{ für } \epsilon \neq n \end{cases} \tag{3.2}$$

Gleichung 3.2 zeigt die verwendete Binärfunktion $g(\epsilon)$. Je nach Iterationsschritt de- und aktiviert die Funktion $g(\epsilon)$ rein binär die entsprechende Zeile in Gleichung 3.1. Nicht aktive Zustände werden mit 0 multipliziert. Aktive Zustände werden mit einer einfachen Verstärkung beaufschlagt.

Die ebenfalls binären Vektorelemente $m_n(\epsilon_f)$ sind mathematisch von der Zusammensetzung des Endzustands abhängig. Dieser Sachverhalt wird beispielsweise mithilfe der manuellen Fahrfunktion erklärt. Diese wird automatisiert mit einer statischen Karte gestartet, in der sich der Roboter durch einen Partikelfilter selbst findet. Hat der Benutzer jedoch zuvor eine Lokalisierung mithilfe der *SLAM*-Methode gefordert, kann dieselbe Fahraufgabe mit einer sich aufbauenden Karte vollzogen werden. Widerrum schließen sich diverse Zustände gegenseitig aus. Eine bestimmte Zielpose ist von dem Ursprung einer statischen Karte abhängig. Diese ist beim *SLAM*-Algorithmus jeodch nicht vorhanden. So kann folglich kein Ziel angefahren werden, wenn zuvor die Lokalisierung mithilfe von *SLAM* abgeschlossen wurde.

$$m_2(\epsilon_f) = \begin{cases} 1 \text{ für } \epsilon_f = 2,7 \\ 0 \text{ für } \epsilon_f \neq 2,7 \end{cases} \tag{3.3}$$

$$m_3(\epsilon_f) = \begin{cases} 1 \text{ für } \epsilon_f \neq 2,7 \wedge m_2(\epsilon_f) = 0 \\ 0 \text{ für } \epsilon_f = 2,7 \end{cases} \tag{3.4}$$

Nur eines der Vektorelemente $m_2(\epsilon_f)$ und $m_3(\epsilon_f)$ kann aktiv sein. Der Modus entspricht auch hier dem Index und sagt aus, ob die Fahraufgabe mit einer statischen oder einer sich aufbauenden Karte bearbeitet werden soll. Hierfür gelten die bereits angesprochenen Restriktionen. Die drei fahrfähigen Endzustände *Manuell*, *Erkunden* und *Ziel* werden durch m_5, m_7 und m_8 aktiviert. Der Zustandsautomat besitzt die Fähigkeit, diverse Zwischenzustände, als temporäre Endzustände ϵ_f auszuführen. Somit kann sich das ALF beispielsweise im Modus *Statische Karte* $\epsilon_f = 3$ ohne eine bestimmte Fahraufgabe selbst lokalisieren. In einem weiteren Schritt ist es dem Benutzer erlaubt, vollständige und fahrfähige Endzustände auszuwählen, wie zum

Beispiel den Modus *Ziel* $\epsilon_f = 8$.

$$m_5(\epsilon_f) = \begin{cases} 1 \text{ für } \epsilon_f = 5 \\ 0 \text{ für } \epsilon_f \neq 5 \end{cases} \tag{3.5}$$

Auch bei den genannten, fahrfähigen Endzuständen gibt es verschiedene Einschränkungen. So kann der manuelle Fahrmodus nur ausgewählt werden, wenn er als Endzustand ϵ_f definiert wurde. Die entsprechende Logik ist in Gleichung 3.5 gezeigt.

$$m_7(\epsilon_f) = \begin{cases} 1 \text{ für } \epsilon_f = 7 \wedge m_2(\epsilon_f) = 1 \wedge m_5(\epsilon_f) = 0 \\ 0 \text{ für } \epsilon_f \neq 7 \end{cases} \tag{3.6}$$

Gleichung 3.6 stellt die Restriktionen des Zustands *Erkunden* dar. Wenn der Modus als Endzustand gewählt wurde, die Lokalisierung ausschließlich durch die *SLAM*-Methode realisiert wird und zuvor kein fahrfähiger Endzustand aktiviert wurde, darf der Zustand *Erkunden* in Kraft treten. Für den Zustand *Ziel* gilt grundlegend dieselbe Logik für die Aktivierung. Jedoch verlangt der Fahrmodus eine statische Karte. In Gleichung 3.7 sind die Abhängigkeiten genauer verdeutlicht.

$$m_8(\epsilon_f) = \begin{cases} 1 \text{ für } \epsilon_f = 8 \wedge m_3(\epsilon_f) = 1 \wedge m_5(\epsilon_f) = 0 \wedge m_7(z_f) = 0 \\ 0 \text{ für } \epsilon_f \neq 8 \end{cases} \tag{3.7}$$

Für eine übersichtliche Darstellung eines Zustandsautomats wird *Unified Modeling Language* (UML) verwendet. Das dem entworfenen Zustandsautomaten entsprechende UML-Diagramm ist in Abbildung 3.7 gezeigt. Zustände sind als Boxen mit Abrundungen dargestellt und durch den Namen sowie den dazugehörigen Index gekennzeichnet. Pfeile stehen für Transitionen zwi-

schen den Zuständen. Je nach Hierarchieebene legen schwarze Punkte den Startpunkt fest, der durchlaufen werden muss.

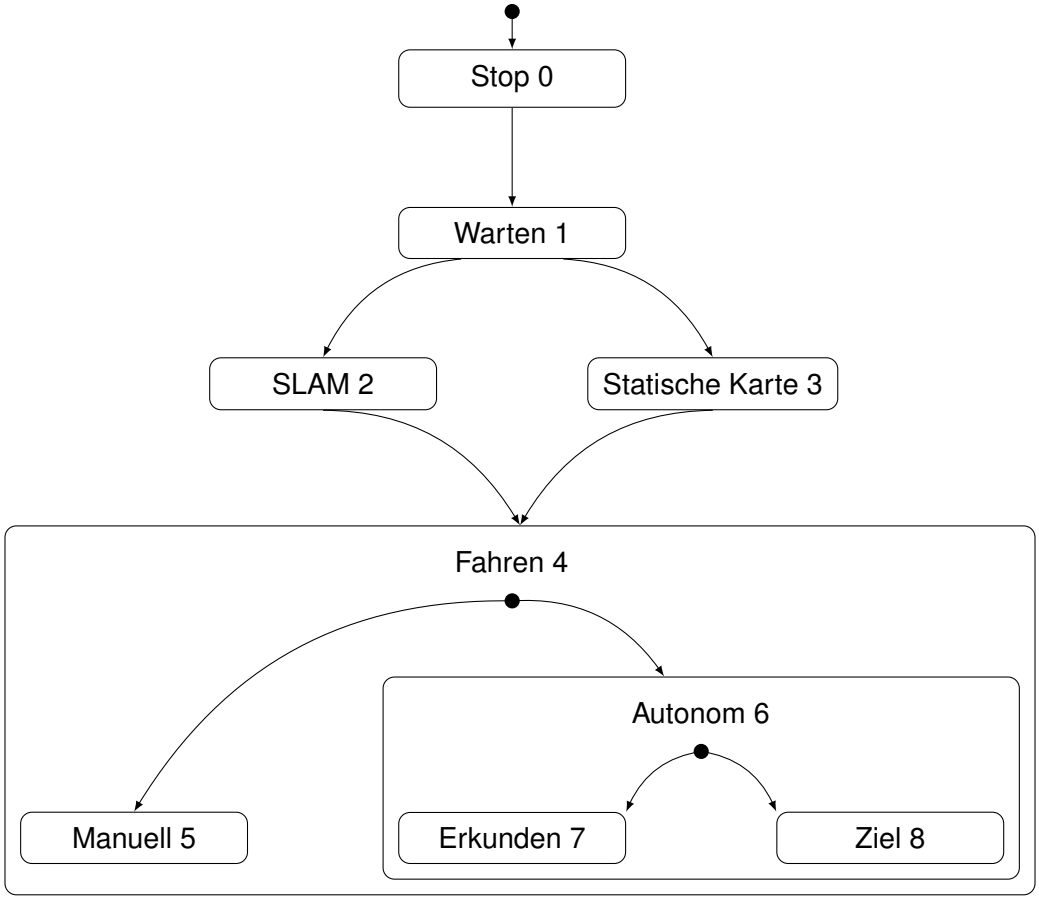


Abbildung 3.7: Konzept eines hierarchischen Zustandsautomaten in Form eines UML-Diagramms. Zustände sind als Boxen mit Abrundungen dargestellt. Ineinander verschachtelte Zustände erben Funktionen von dem jeweils größeren Zustand. Pfeile deuten mögliche Transitionen an. Startzustände werden durch schwarze Punkte gezeigt.

Darüber hinaus gilt es, den Zustandsautomaten und das ALF nach gegebenen Standards auszulegen. Die *Society of Automotive Engineers* (SAE) ist eine Organisation, die aus Ingenieuren und technischen Experten besteht [8]. Ziel dieser ist im Allgemeinen die Weiterentwicklung im Thema Mobilität [8]. Unter anderem veröffentlicht *SAE International* Normen wie die *SAE J3016*. Diese beschäftigt sich mit verschiedenen Stufen der Automatisierung von Fahrzeugen. Eine entsprechende Tabelle ist in Abbildung 3.3 dargestellt.

Das ALF wird zum Zeitpunkt des Entwicklungsbeginns dieser Masterarbeit auf dem *SAE-Level* 3 eingeschätzt. Hierbei werden lediglich die autonomen Fahrmodi des Fahrzeugs berücksichtigt. Die Fahrbefehle konnten selbstständig durch das System ausgeführt werden und die Umgebung über die bereits genannten Sensoren erfasst werden. Die *Fallback Performance* beschreibt den Übergang des automatisierten oder autonomen in einen risikominimalen Zustand. Bisher musste die laufende Software des ALFs auch im Fehlerfall manuell am Fahrzeug beendet werden. Dies bestärkte die Einschätzung, dass das Fahrzeug bedingt automatisiert sei.

Tabelle 3.3: Übersicht der sechs Automatisierungsstufen nach SAE Standard J3016. Die Angaben wurde von der Quelle übernommen und sind sinngemäß übersetzt. [9]

SAE Level	Name	Ausführung von Fahrbefehlen	Erfassung der Umgebung	Fallback Performance	Systemauswirkung
Benutzer überwacht die Fahrumgebung					
0	Keine Automatisiering	Benutzer	Benutzer	Benutzer	k.A.
1	Assisitiert	Benutzer & System	Benutzer	Benutzer	Einige Fahrmodi
2	Teilautomatisiert	System	Benutzer	Benutzer	Einige Fahrmodi
Automatisiertes System - System überwacht Fahrumgebung					
3	Bedingte Automatisierung	System	System	Benutzer	Einige Fahrmodi
4	Hochautomatsisiert	System	System	System	Einige Fahrmodi
5	Vollautomatisiert (autonom)	System	System	System	Alle Fahrmodi

Das ALF wird im Rahmen der technischen und sicherheitsrelevanten Möglichkeiten auf das *SAE-Level* 5 angehoben. Dies betrifft insbesondere den zu entwickelnden Zustandsautomat, da dieser den Aufruf der Fahraufgaben realisiert.

4 Evaluation

Zur Evaluation der künstlichen neuronalen Netze wird eine anwendungsorientierte *Benchmark* durchgeführt. Hierbei wird anhand der in Abschnitt 3.2.3 beschriebenen Datensätze die *Precision-Recall*-Methode angewendet. Weiterhin werden die Benchmarks auf dem integrierten Computer des ALFs und einem eingebetteten System ausgeführt. Als eingebettetes System wird ein *Raspberry Pi 3 Model B* verwendet. Die Eckdaten des integrierten Computers sind in der Masterthesis von M.Sc. *Dominik Eickmann* und M.Sc. *Dennis Hotze* dargestellt [3]. Es ist keine geräteübergreifende Veränderung der Genauigkeit je Netz zu erwarten, da die Eingangsdaten und die Rechenoperationen identisch sind. Jedoch können so die Bearbeitungszeiten pro Bild für unterschiedliche Hardwareplattformen verglichen werden. Eine präzise Auflistung aller gemessenen Analysezeiten ist in Tabelle 4.1 präsentiert.

Insgesamt werden für den Test 12755 Bilder von Personen aus dem Trainingsdatensatz entnommen. Der technische Hintergrund hierfür ist in Abschnitt 3.2.3 zu finden. Die Unterteilung des Datensatzes in Trainings- und Testdaten geschah vor dem Training. Andernfalls würde ein Netz während des Trainings mit bereits bekannten Eingangsdaten rechnen. Dies würde das Testergebnis verfälschen. Als Vergleich analysiert jedes Netz auch den eigenen Datensatz. So kann die Performance am Einsatzort des ALFs an der Hochschule Bochum evaluiert werden. Jedes Bild wird für die verwendeten, neuronalen Netze auf eine Pixelgröße von 300×300 skaliert. Für die Evaluation des *HOG-SVM*-Systems wird eine höhere Auflösung gewählt. Hierbei wird eine Seite des Bildes softwareseitig auf 400 Pixel begrenzt.

Der im Grundlagenabschnitt 2.1.3 beschriebene *mAP*-Wert wird häufig auf Objekterkennungssystemen mit multiplen Klassen angewendet. Die hier entwickelte Personenerkennung soll jedoch lediglich die Klasse *Person* erkennen. Somit ist der *mAP*-Wert in diesem Fall der Mittelwert eines Messwerts und kann als Integral der *Precision-Recall*-Kurve angesehen werden. Im Verlauf der Evaluation der angewendeten Systeme wird mithilfe einer Berechnungssoftware jeweils der *mAP*-Wert berechnet.

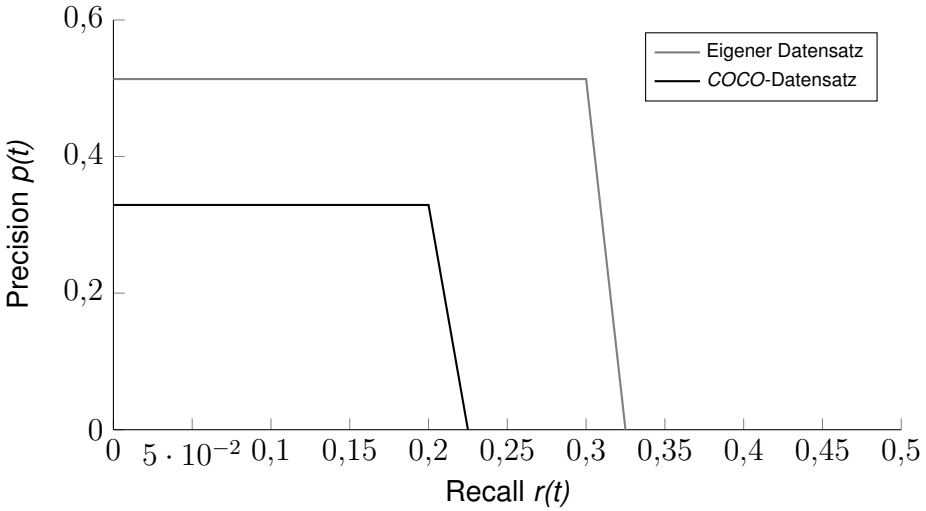


Abbildung 4.1: *Precision-Recall*-Kurven der *HOG-SVM*-Kombination. [53]

Die Kombination aus *HOG* und *SVM* erreicht in der Benchmark die in Abbildung 4.1 präsentierten Ergebnisse. Der *mAP*-Wert liegt für den eigenen Datensatz bei 0,16 und für den *COCO*-Datensatz bei 0,07. Die eingetragenen Begrenzungsrahmen stechen bei der Durchsicht der Ergebnisse durch ihren verhältnismäßig großen Umfang heraus. Dies lässt sich auf die Dimension der Zellen zurückführen. Eine Reduktion der Zellengröße könnte wiederum die Erkennung weiter verschlechtern. Der *IoU*-Wert ist hierdurch entsprechend niedrig und führt zu diesem Ergebnis. Mit den entsprechende Optimierungen wurden die *Precision-Recall* Werte für dieses System maximiert und sind in der obigen Abbildung dargestellt.

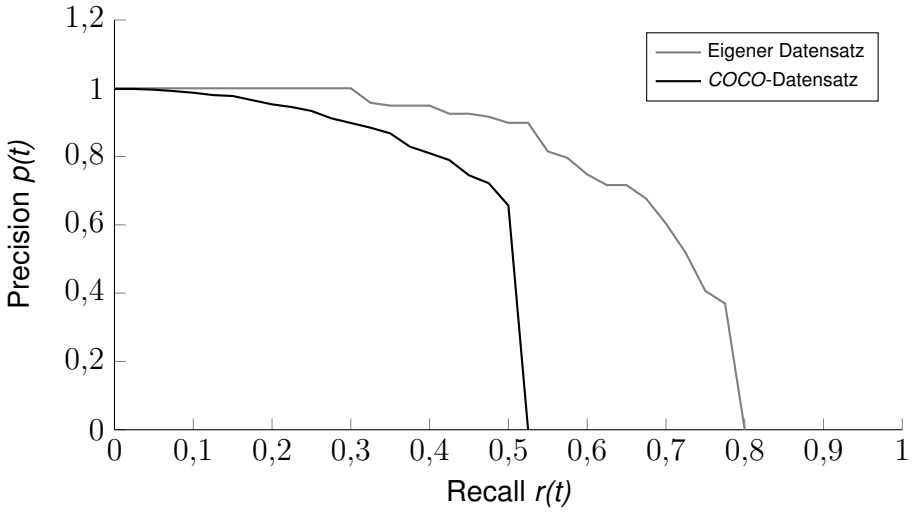


Abbildung 4.2: Gegenüberstellung der *Precision-Recall* Kurven eines quantisierten *SSD MobileNet V1* Netzes [54].

Eine deutliche Steigerung hinsichtlich der Geschwindigkeit im Vergleich zur *HOG-SVM*-Methode wird durch das quantisierte *MobileNet V1 SSD* Netz erreicht. Die Berechnung des Integrals der *Precision-Recall*-Kurve ergab für den eigenen Datensatz einen Wert von 0,68 und für den *COCO*-Datensatz 0,46. Dieses Netz ist in der Lage 90 verschiedene Klassen zu erkennen. Hierbei nimmt das Netz jedoch lediglich 4 MB Speicherplatz ein.

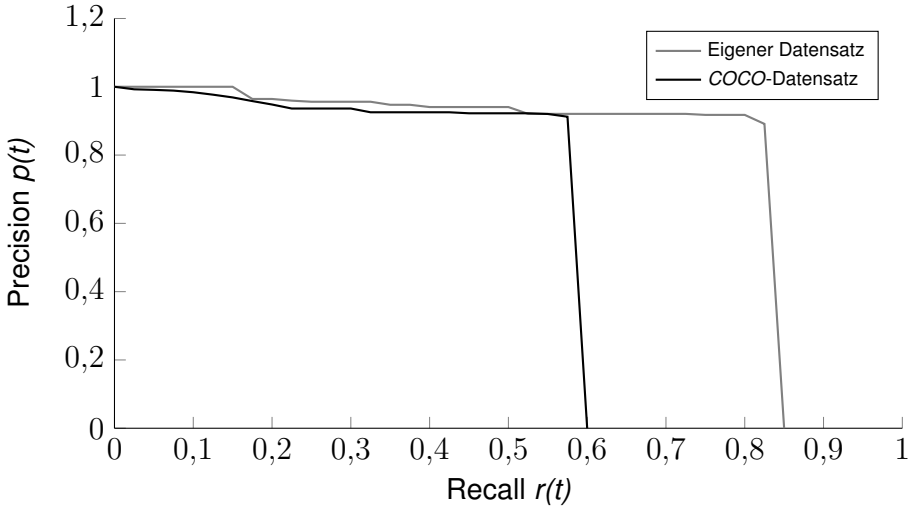


Abbildung 4.3: *Precision-Recall*-Kurven des entwickelten *MobileNet V1 SSD* Netzes.

Das in Abbildung 4.3 gezeigte *MobileNet V1 SSD*-Netz ist darauf trainiert, Personen zu erken-

nen. Die dargestellte Architektur weist einen *mAP*-Wert von 0,79 für den eigenen Datzensatz und 0,56 für den *COCO*-Datensatz auf.

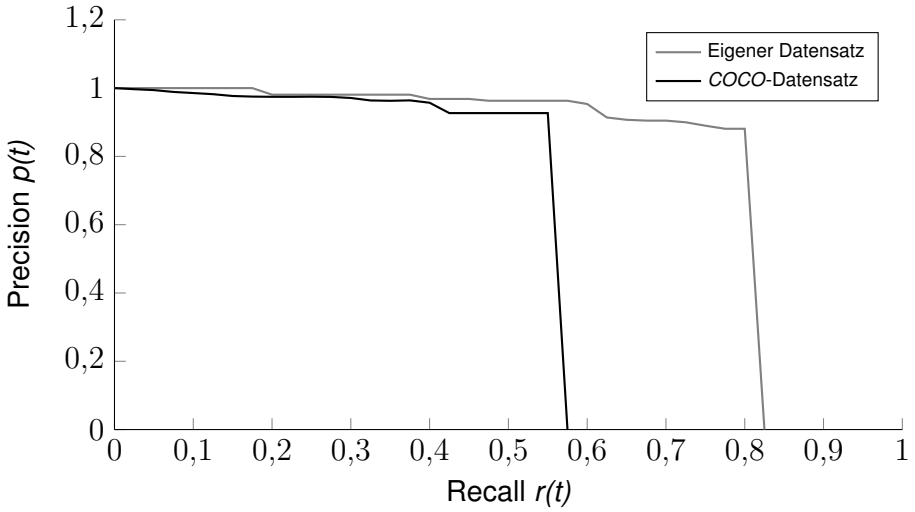


Abbildung 4.4: *Precision-Recall*-Kurven des enwtickelten *SSD MobileNet V2* Netz.

In Darstellung 4.4 werden die *Precision-Recall*-Kurven eines *MobileNet V2 SSD* gezeigt. Das Verhaltensmuster dieses Netzes ist ebenfalls auf eine Personenerkennung beschränkt, um enthaltene Parameter und den damit verbundenen Speicherplatz zu reduzieren. Die *mAP*-Werte liegen bei 0,78 für den eigenen Datensatz und 0,54 für den *COCO*-Datensatz.

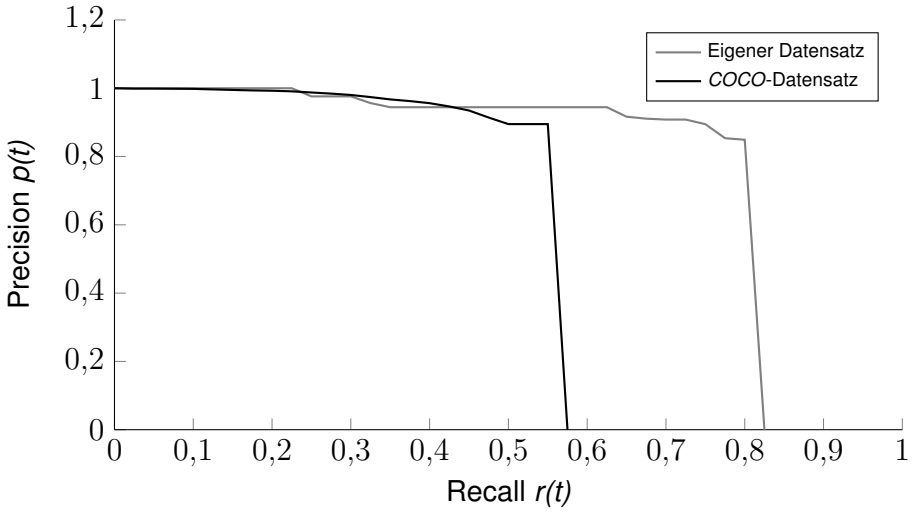


Abbildung 4.5: *Precision-Recall*-Kurven eines *MobileNet V2 SSDLite* Netzes [55].

Abbildung 4.5 zeigt die *Precision* und *Recall* Kurve einer *MobileNet V2 SSDLite* Architektur. Auffällig hierbei ist die konstant hohe Genauigkeit $p(t)$. Diese liegt bis zu einem *Recall*-Wert von circa 0,8 zwischen 0,9 und 1. Für die Anwendung auf den eigenen Datensatz erreicht diese Architektur eine mittlere Durschnittsgenauigkeit von 0,77. Angewendet auf den *COCO*-Datensatz liegt der Wert bei 0,54.

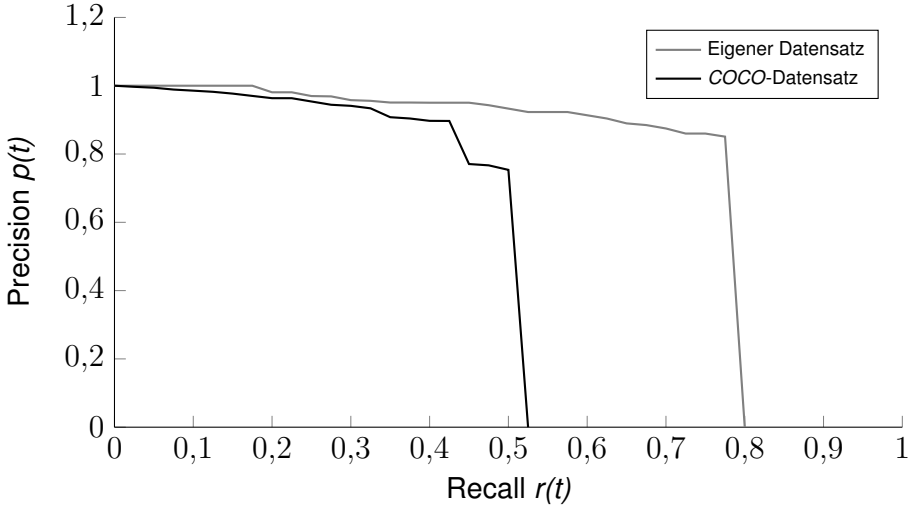


Abbildung 4.6: Darstellung der *Precision-Recall*-Verläufe des enwtickelten *MobileNet V2 SSDLite* Netz.

Die *Precision-Recall*-Kurven der *MobileNet V2 SSDLite* Architektur sind in der Grafik 4.6 abgebildet.

Nachfolgend werden die Benchmarkergebnisse der untersuchten Objekterkennungssysteme gegenübergestellt. Der Vergleich dient zur Veranschaulichung der gemessenen *Precision-Recall*-Leistung je Datensatz. In Abbildung 4.7 werden die untersuchten *MobileNet* Architekturen anhand des eigenen Datensatzes verglichen. So kann eine Aussage darüber getroffen werden, ob die Systeme ortsabhängig ein anderes Verhaltensmuster aufzeigen. Es kann beispielsweise vorkommen, dass die Netze aufgrund prägnater Eigenschaften der Umgebung am Standort der Hochschule Bochum verschieden reagieren. Eine allgemeine Aussage über die Genauigkeiten kann anhand der Darstellung 4.8 getroffen werden. Für die Verallgemeinerung sorgt hierbei der *COCO*-Datensatz. Dieser enthält anders als der eigene Datensatz Bilder von verschiedenen Orten.

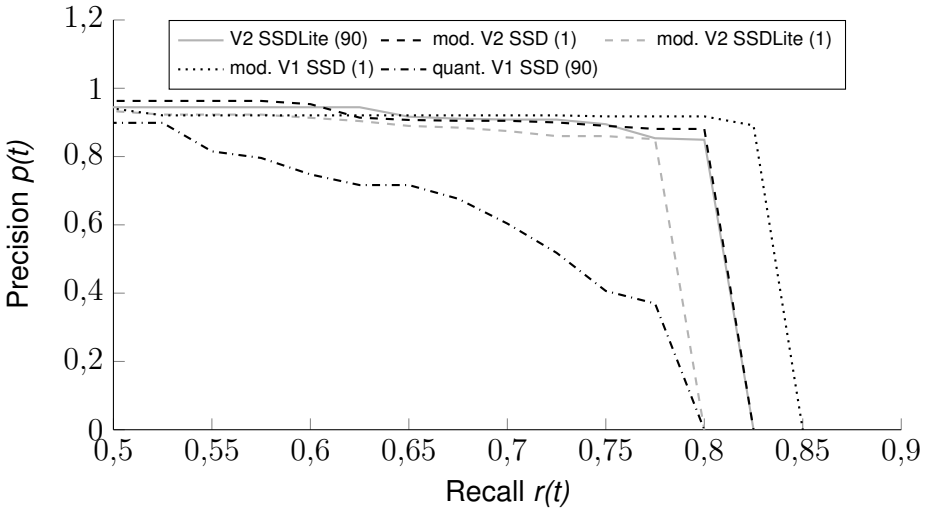


Abbildung 4.7: *Precision-Recall*-Kurven aller *MobileNet* Architekturen in Anwendung auf den eigenen Datensatz. Die Abkürzungen *mod.* und *quant.* stehen für *modifiziert* und *quantisiert*.

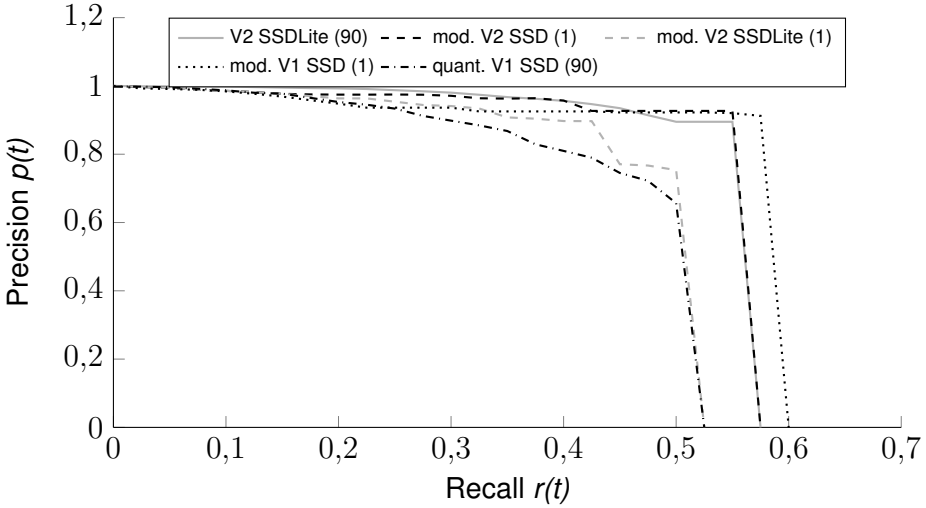


Abbildung 4.8: *Precision-Recall*-Kurven aller *MobileNet* Netze in Anwendung auf den *COCO*-Datensatz. Die Abkürzungen *mod.* und *quant.* stehen für *modifiziert* und *quantisiert*.

Bei den Auswertungen der *Precision-Recall*-Werte erreichte die Kombination aus *HOG* und *SVM* im Vergleich zu den *MobileNet* Netzen niedrige Ergebnisse. In den Vergleichsgrafiken 4.7 und 4.8 wird das System aufgrund dessen nicht weiter betrachtet. Es ist zu beachten, dass die verwendete Software für die Messung den *Recall* in einer Schrittweite von 0,25 ausgibt. Überwiegend fällt das quantisierte *MobileNet V1 SSD* Netz in Abbildung 4.7 auf. Die Kurve des Netzes zeigt einen anderen Verlauf als die anderen Architekturen. Das Verhältnis aus *Precision* und *Recall* reduziert sich bei derartigen Strukturen durch eine Quantisierung. Netzoptimierungen wie die zweite Version des *MobileNets* oder die Weiterentwicklung des *SSDs* zeigen vorwiegend geringere *Recall*-Werte. Jedoch gibt es im Verlauf der *Precision*-Werte keine nenneswerte Unterschiede.

Tabelle 4.1: Vergleich der Rechenzeiten pro Bild auf verschiedenen Hardwareplattformen gemessen an 12755 Bildern des *COCO*-Datensatzes. Die präsentierten Zeiten wurden für alle Analyseschritte addiert und durch die Anzahl aller Bilder geteilt. Ein Analyseschritt bedeutet in diesem Fall die reine Berechnung der Klassifikation und exkludiert beispielsweise die Zeit für eine Anpassung des Bildes für das entsprechende Netz. Die Abkürzung *mod.* steht für ein modifiziertes Netz. Eine quantisierte Architektur wird mit *quant.* gekennzeichnet. In runden Klammern wird die Anzahl der verschiedenen Klassen beziffert, die das jeweilige Netz erkennen kann.

Hardware	Computer ALF	Raspberry Pi 3 Model B
HOG & SVM (1)	37 ms	37 ms
MobileNet V1 SSD quant. (90)	39 ms	47 ms
MobileNet V1 SSD mod. (1)	25 ms	54 ms
MobileNet V2 SSD mod. (1)	24 ms	47 ms
MobileNet V2 SSDLite (90)	20 ms	42 ms
MobileNet V2 SSDLite mod. (1)	18 ms	39 ms

Tabelle 4.2: Gegenüberstellung der errechneten *mean Average Precision*-Werte. Die Hardwareplattformen sind für die Genauigkeit eines Netzes nicht relevant, da diese auf jeder Plattform denselben Wert hat.

Objekterkennungssystem	Eigener Datensatz	COCO-Datensatz
HOG & SVM (1)	0,16	0,07
MobileNet V1 SSD quant. (90)	0,68	0,46
MobileNet V1 SSD mod. (1)	0,79	0,56
MobileNet V2 SSD mod. (1)	0,78	0,54
MobileNet V2 SSDLite (90)	0,77	0,54
MobileNet V2 SSDLite mod. (1)	0,74	0,47

Die Berechnungszeit pro Bild des eigenen Datensatzes aller untersuchten Objekterkennungssysteme ist in Abbildung 4.1 präsentiert. Es ist die Benchmark auf dem Rechner des ALFs in Anwendung auf den eigenen Datensatz dargestellt. Bereits aus den beschriebenen Grundlagen aus Abschnitt 2.2.2 geht eine grobe Schätzung der Rechengeschwindigkeiten hervor. Die gezeigten Messergebnisse spiegeln die Erwartungen aus den Grundlagen wider. Mit 18 ms Rechenzeit und einer mittleren Durchschnittsgenauigkeit am eigenen Datensatz von 0,74 ist das modifizierte *MobileNet V2 SSDLite* Netz das schnellste in der Benchmark. Jedoch ist eine leichte Regression hinsichtlich der Genauigkeit im Vergleich zu den anderen Architekturen in Tabelle 4.2 zu erkennen. Aufgrund der beschriebenen, hohen Auflösung der Messergebnisse könnten tatsächliche *mAP*-Werte mit einer feineren Messung noch leicht variieren.

Für eine vollständige Übersicht der Evaluationsergebnisse aller *MobileNet* Netze ist der Graph in Abbildung 4.9 gezeigt. Der belegte Speicherplatz durch die Netze wird durch den Durchmesser der grauen Kreise angedeutet. Im Beispiel der *SSDLite* Modifikation ist zu erkennen, dass die Änderung der Ausgabeschicht den Speicherplatz minimiert. Dies bestätigt die These aus Abschnitt 3.2.3. Das quantisierte *MobileNet V1 SSD* Netz weist im Verhältnis zu den anderen Netzen eine langsame Rechenzeit und eine niedrige Genauigkeit auf. Letzteres wird durch die Quantisierung verursacht. Die Größe der Ausgabeschicht ist ein Indiz für eine lange Rechenzeit. Dieses Phänomen tritt bei der Verwendung von *SSDLite* laut der Benchmarkergebnisse nicht derartig ausgeprägt auf. Die Benchmark Ergebnisse zeigen einen stärkeren Anstieg der

Rechengeschwindigkeit bei der Verwendung des *SSDLite* Klassifikators. Zwischen den Versionen der *MobileNet* Architekturen ist keine relevante Leistungsänderung für diese Benchmark zu erkennen.

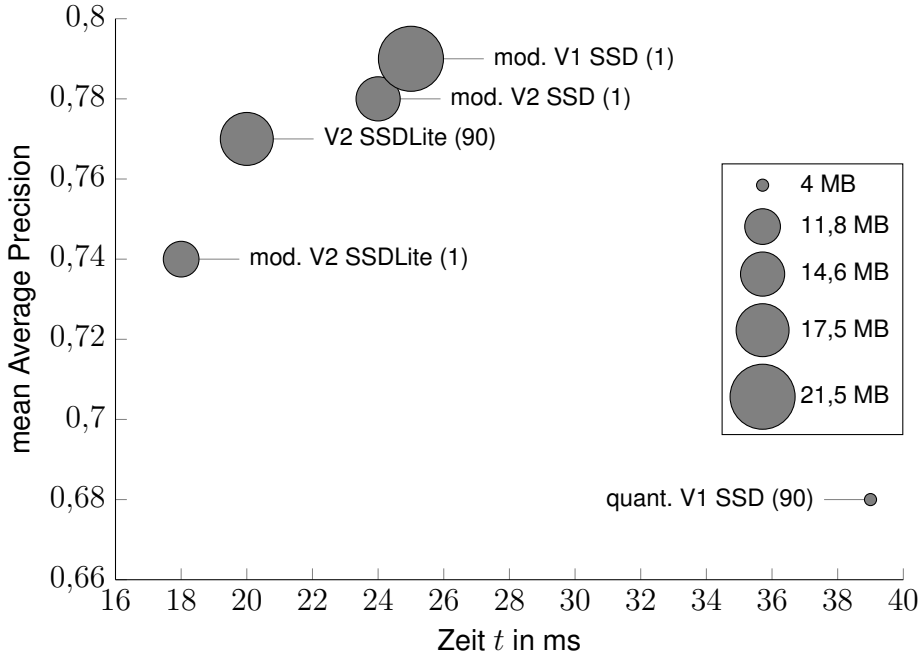


Abbildung 4.9: Vergleich aller untersuchten *MobileNet* Architekturen. Das Diagramm stellt die *mean Average Precision* aufgetragen über die gemessene Rechenzeit dar. Der Durchmesser der grauen Kreise deutet verhältnismäßig den Speicherplatz der Netze an. Gezeigt ist der Test auf dem integrierten Rechner des ALFs in Anwendung auf den eigenen Datensatz.

5 Zusammenfassung und Ausblick

Im Rahmen dieser Masterarbeit wurde eine Personenerkennung für die Anwendung am autonomen Logistikfahrzeug ALF entwickelt und evaluiert. Die entwickelte Software ist darauf optimiert, eine ausgewogene Balance zwischen einer hohen Genauigkeit bei kurzer Rechenzeit zu erzielen. Dies wird durch eine Unterteilung des Gesamtsystems in Teilsysteme auf verschiedenen Ebenen erreicht. Implementierte Softwarekomponenten eignen sich für den Einsatz auf eingebetteten Systemen und wurden bewusst danach ausgelegt. Eine dem Erkennungssystem übergeordnete Personendetektion erkennt Menschen im unmittelbaren Sichtkegel der verbauten *Kinect*-Kameras. Die Bildverarbeitung nutzt ein künstliches neuronales Netzwerk zur Analyse von Bildern. Für die Auswahl der Netze wurden state-of-the-art Lösungen verglichen und mögliche Netztypen evaluiert.

Im Hinblick auf die in Kapitel 4 erarbeiteten Genauigkeitswerte eignen sich die *HOG-SVM*-Kombination und das quantisierte *MobileNet V1 SSD* Netz für beide Hardwareplattformen nicht. Die *MobileNet V1 SSD* Architektur zeigte die besten Ergebnisse hinsichtlich der Genauigkeit. Das *MobileNet V2 SSDLite* Netz verarbeitet die Bilder der Benchmark am schnellsten. Je nach Anwendungsfall und Hardwareplattform werden die genannten KNNs angewendet.

Der Personendetektion untergeordnet, wurde eine Gesichtsdetektion und -erkennung implementiert. Diese sucht bei einer Personendetektion im entsprechenden Interessensbereich nach vorhandenen Gesichtern. Wird ein Gesicht detektiert, erfolgt eine Merkmalsextraktion. Gesichter dienen hierbei als langfristig einzigartiges Unterscheidungsmerkmal von Personen. Nach der Merkmalsextraktion werden im System bekannte Personen wiedererkannt oder durch ein Registrierungsprozess in eine Datenbank eingepflegt.

Der entwickelte Zustandsautomat steuert das Fahrzeug mithilfe der Ausgabe einer Sprachverarbeitung. Diese klassifiziert vom Benutzer eingesprochene, anwendungsorientierte Sprache. Die Klassifikation wird als Übergabewert und somit zur Eingabe des Zustandsautomats

verwendet. Der EA ist durch seinen hierarchischen Aufbau in der Lage, Fahraufgaben in Form von Zuständen nacheinander aufzubauen. Weiterhin könnten in Zukunft vollständige Logistikanwendungen mithilfe dieses Systems realisiert werden. Eine Weiterentwicklung des Zustandsautomats ist möglich, indem ein übergeordnetes System implementiert wird. Dieses könnte Fahraufgaben gesondert klassifizieren. So kann zum Beispiel die Fahraufgabe *Ziel* durch einen Anwendungsfall *Paket abholen* oder auch *Ware ausliefern* in Anspruch genommen werden.

In Kapitel 1 wurde als Beispiel die Freischaltung des ALFs für autorisierte Personen oder eine gesonderte Gefahreneinschätzung genannt. Letzteres könnte in Verbindung mit dem *Robot Operating System* dazu führen, dass das Fahrzeug einen besonders großen Abstand zu Personen hält oder Bereiche mit hohem Personenaufkommen meidet. Als Hilfestellung und Grundlage für ein derartiges Projekt können die aus der Personenerkennung extrahierten Informationen dienen.

Bezüglich der festgestellten Werte der Bildverarbeitungsalgorithmen in Kapitel 4 sind weitere Optimierungen möglich. Der in dieser Masterarbeit erstellte Datensatz kann mit Bildern passend zum Einsatzort des ALFs erweitert werden. Eine Steigerung der Genauigkeit ist hierbei zu erwarten.

Quellenverzeichnis

[1] Canalys. *Smart speaker market booms in 2018, driven by Google, Alibaba and Xiaomi*. https://www.canalys.com/newsroom/smart-speaker-market-booms-in-2018-driven-by-google-alibaba-and-xiaomi - zugegriffen am 24.09.2020.

[2] Sopra Steria GmbH. *Potenzialanalyse Künstliche Intelligenz*. https://www.soprasteria.de/docs/librariesprovider2/sopra-steria-de/publikationen/studien/potenzialanalyse-kuenstliche-intelligenz-2017.pdf?sfvrsn=190f45dc_4 - zugegriffen am 01.10.2020. Feb. 2017.

[3] Eickmann, D. und Hotze, D. *Entwicklung und Verfikation eines autonomen Logistik-Fahrzeugs. Hochschule Bochum - Bochum University of Applied Sciences*. Masterthesis. Feb. 2018.

[4] Montorio, G. und Dittmann, H. *Implementierung einer Schlupfregelung per Model-Based Design sowie einer SLAM-Kartografierung für ein autonomes Logistik-Fahrzeug. Hochschule Bochum - Bochum University of Applied Sciences*. Bachelorthesis. Feb. 2019.

[5] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. und Adam, H. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications - Computing Research Repository (CoRR)*. Apr. 2017.

[6] Dalal, N. und Triggs, B. *Histograms of Oriented Gradients for Human Detection - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 2005.

[7] Dittman, H. *Entwicklung und Verfikation einer Sprachverarbeitung für das autonome Logistik-Fahrzeug ALF. Hochschule Bochum - Bochum University of Applied Sciences*. Masterthesis. Okt. 2020.

[8] SAE International. *About SAE International*. https://www.sae.org/about - zugegriffen am 22.09.2020. 2020.

[9] SAE International. *Automated Driving - Levels of driving are defined in new SAE International standard J3016*. https://saemobilus.sae.org/content/j3016_201806 - zugegriffen am 02.10.2020. 2014.

[10] Mishra, M. und Srivastava, M. *A view of Artificial Neural Network - 2014 International Conference on Advances in Engineering Technology Research (ICAETR - 2014)*. 2014. S. 1-3.

[11] Manickam, M., Mohanapriya, M., Kale, S., Uday, M., Kulkarni, P., Khandagale, Y. and Patil, S.P. *Research study on applications of artificial neural networks and e-learning personalization - International Journal of Civil Engineering and Technology*. Aug. 2017. S. 1422-1432.

[12] Medina, G. *Diagram of an artificial neural network*. https://tex.stackexchange.com/questions/132444/diagram-of-an-artificial-neural-network - zugegriffen am 24.09.2020.

[13] Nischwitz, A., Fischer, M., Haberäcker, P., und Socher, G. *Bildverarbeitung: Band II des Standardwerks Computergrafik und Bildverarbeitung*. 4. Auflage. Springer Vieweg. Jan. 2020.

[14] Kriesel, D. *Ein kleiner Überblick über Neuronale Netze*. http://www.dkriesel.com/science/neural_networks - zugegriffen am 01.10.2020. 2007.

[15] R. Bibel, W., Ertel, W. und Kruse. *Grundkurs Künstliche Intelligenz - Eine praxisorientierte Einführung*. 3. Auflage. Springer Vieweg. 2013.

[16] Carterette, B. *Encyclopedia of Database Systems - Precision and Recall. Springer US*. 2009. S. 2126-2127, S. 2779–2779.

[17] Süße, H. und Rodner, A. *Bildverarbeitung und Objekterkennung - Computer Vision in Industrie und Medizin. Springer Vieweg*. 2014.

[18] Canziani, A., Adam, P. und Culurciello, E. *An Analysis of Deep Neural Network Models for Practical Applications - Computing Research Repository (CoRR)*. 2016.

[19] simongeek. *Pedestrian classification from photos using the Linear SVM and HOG features*. https://github.com/Ermlab/hog-svm-inria - zugegriffen am 24.09.2020.

[20] Hearst, M., Dumais, S.T., Osman, E., Platt, J. und Scholkopf, B. *Support vector machines - Intelligent Systems and their Applications, IEEE*. Aug. 1998, S. 18–28.

[21] Kibria, S. und Hasan, M. *An analysis of Feature extraction and Classification Algorithms for Dangerous Object Detection - 2017 2nd International Conference on Electrical & Electronic Engineering (ICEEE)*. Dez. 2017.

[22] Goodfellow, I. , Bengio, Y. und Courville, A. *Deep Learning*. http://www.deeplearningbook.org - zugegriffen am 21.09.2020. The MIT Press. 2016.

[23] SVG Repo. *Car Compact SVG Vector*. https://www.svgrepo.com/svg/178/car-compact - zugegriffen am 24.09.2020.

[24] Saha, S. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 - zugegriffen am 24.09.2020.

[25] estamos. *Write Convolutional Neural network using TikZ*. https://tex.stackexchange.com/questions/519268/write-convolutional-neural-networks-using-tikz - zugegriffen am 24.09.2020.

[26] Simonyan, K. und Zisserman, A. *Very Deep Convolutional Networks for Large-Scale Image Recognition - Computing Research Repository (CoRR)*. 2015.

[27] *Keras Applications*. https://keras.io/api/applications/ - zugegriffen am 17.09.2020.

[28] He, K., Zhang, X., Ren, S. und Sun, J. *Deep Residual Learning for Image Recognition - Computing Research Repository (CoRR)*. 2015.

[29] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. und Rabinovich, A. *Going deeper with convolutions - 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015. S. 1-9.

[30] Tsang, S.-H. *Review: MobileNetV1 - Depthwise Separable Convolution (Light Weight Model)*. https://towardsdatascience.com/review-mobilenetv1-depthwise-separable-convolution-light-weight-model-a382df364b69 - zugegriffen am 24.09.2020.

[31] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. und Chen, L. *MobileNetV2: Inverted Residuals and Linear Bottlenecks - 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018. S. 4510-4520.

[32] Girshick, R., Donahue, J., Darrell, T. und Malik, J. *Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation - 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Juni 2014. S. 580-587.

[33] Deng, Z., Sun, H., Zhou, S., Zhao, J., Lei, L. and Zou, H. *Multi-scale object detection in remote sensing imagery with convolutional neural networks - ISPRS Journal of Photogrammetry and Remote Sensing*. Mai 2018.

[34] Girshick, R. *Fast R-CNN - Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Dez. 2015.

[35] Ren, S., He, K., Girshick, R. und Sun, J. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks - Advances in Neural Information Processing Systems 28. Curran Associates, Inc.* 2015. S. 91-99.

[36] Liu, W. , Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y. und Berg, A. C. *SSD: Single Shot MultiBox Detector - European conference on computer vision*. pp. 21-37. 2016.

[37] Szegedy, C., Reed, S., Erhan, D. und Anguelov, D. *Scalable, high-quality object detection - Computing Research Repository (CoRR)*. 2014.

[38] Staud, J. L. *Unternehmensmodellierung - Objektorientierte Theorie und Praxis mit UML 2.5*. 2. Auflage. Springer Gabler.

[39] Solov'ev, V. »Implementation of finite-state machines based on programmable logic ICs with the help of the merged model of Mealy and Moore machines«. In: (Feb. 2013).

[40] Yannakakis, M. *Hierarchical State Machines*. Jan. 2000. S. 315-330.

[41] Li, Y., Li, J., Lin, W., und Jianguo, L. *Tiny-DSOD: Lightweight Object Detection for Resource-Restricted Usages*. Juli 2018.

[42] Zhang, Y., Bi, S., Dong, M. und Liu, Y. *The Implementation of CNN-based Object Detector of ARM Embedded Platforms. 2018 IEEE 16th Int. Conf. on Dependable, Autonomic & Secure Comp., 16th Int. Conf. on Pervasive Intelligence & Comp., 4th Int. Conf. on Big Data Intelligence & Comp., and 3rd Cyber Sci. and Tech. Cong.* 2018.

[43] Parvat, A., Chavan, J., Kadam, S., Dev, S., und Pathak, V. *A survey of deep-learning frameworks - 2017 International Conference on Inventive Systems and Control (ICISC)*. 2017. S. 1-7.

[44] Farhoodfar, A. *Machine Learning for Mobile Developers: Tensorflow Lite Framework*. Apr. 2019.

[45] Wang, Y., Wei, G.-Y., und Brooks, D. *Benchmarking TPU, GPU, and CPU Platforms for Deep Learning*. 2019.

[46] Lin, T.-Y. *COCO - Common Objects in Context*. https://cocodataset.org/ - zugegriffen am 24.09.2020.

[47] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., und Zitnick, C. L. *Microsoft COCO: Common Objects in Context - Computing Research Repository (CoRR)*. 2014.

[48] Abadi, M. u. a. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems - Computing Research Repository (CoRR)*. 2016.

[49] Arora, S., und Bhatia, M. P. S. *Handwriting recognition using Deep Learning in Keras - 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*. 2018, S. 142–145.

[50] Geitgey, A. *Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning*. https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78 - zugegriffen am 24.09.2020.

[51] Geitgey, A. *Face Recognition*. https://pypi.org/project/face-recognition/ - zugegriffen am 24.09.2020.

[52] Amos, B. *Models and Accuracies*. https://cmusatyalab.github.io/openface/models-and-accuracies/ - zugegriffen am 24.09.2020.

[53] Rosebrock, A. *Pedestrian Detection OpenCV*. https://www.pyimagesearch.com/2015/11/09/pedestrian-detection-opencv/ - zugegriffen am 24.09.2020.

[54] *Object detection*. https://www.tensorflow.org/lite/models/object_detection/overview - zugegriffen am 24.09.2020.

[55] kaka lin. *SSDLite-MobileNet v2 (tflite)*. https://github.com/kaka-lin/object-detection#sdlite-mobilenet-v2 - zugegriffen am 24.09.2020.

A Anhang

A.1 Abbildungen

Tabelle A.1: Auflistung der Zustände und der dazugehörigen Knotenmengen. Die Funktion der aufgerufenen Knoten wird in deskriptiver Form wiedergegeben.

Zustand und Knotengruppe	Deskriptive Funktion
Stop (k_0)	Risikominimaler Zustand. Ausschalten des *ROS*-Netzwerks. Wechsel in den Zustand *Warten* erfolgt nur nach manueller Quittierung.
Warten (k_1)	Grundfunktionen werden hochgefahren. Der *LiDAR*-Sensor, die *Kinect*-Sensoren, die *ROS* Visualisierungssoftware und die Sprachausgabe werden eingeschaltet.
SLAM (k_2)	Knoten für den *SLAM*-Algorithmus und statische Koordinatensysteme werden aufgerufen.
Statische Karte (k_3)	Einschalten des Partikelfilters für die Posenfindung durch gegebene Sensoren.
Fahren (k_4)	Übergeordneter Zustand
Manuell (k_5)	Das aus der Bachelorarbeit entwickelte Fahrprogramm wird im manuellen Modus ausgeführt [4].
Autonom (k_6)	Übergeordneter Zustand
Erkunden (k_7)	Das aus der Bachelorarbeit entwickelte Fahrprogramm wird im autonomen Modus ausgeführt [4]. Außerdem setzt eine entsprechende Applikation eine Erkundungsanwendung um.
Ziel (k_8)	Das aus der Bachelorarbeit entwickelte Fahrprogramm wird im autonomen Modus ausgeführt [4]. Wird ein Ziel im *ROS*-Netzwerk veröffentlicht, fährt der Roboter dieses an.

Abbildung A.1: Abbildung des autonomen Logistikfahrzeugs ALF.

A.2 Inhalt Datenträger

1. Masterthesis in digitaler Form
2. Entwickelte Software
3. Lastenheft
4. Literatur