

Institutsleitung
Prof. Dr.-Ing. Dr. h. c. J. Becker (Sprecher)
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Masterarbeit Nr. ID-2418

von Herrn cand. wing. Manuel **Kaiser**

**Künstliches Lernen: Trainingsdaten aus dem
virtuellen Fahrversuch für
Szenarienklassifizierung mit Deep Learning
Algorithmen;
“Artificial Learning“ - Training Data From a
Virtual Driving Test for Scenario Classification
With Deep Learning Algorithms**

Beginn: 01.05.2018
Abgabe: 31.12.2018

Betreuer: Dipl.-Wi.-Ing. Raphael Pfeffer
Institut für Technik der Informationsverarbeitung

Korreferent: Prof. Dr. Andreas Oberweis
AIFB

Prof. Dr.-Ing. Eric Sax

Abstract

Ab Stufe 3 des autonomen Fahrens kontrolliert nicht mehr der Fahrer, sondern das System die Umgebung in der sich ein Fahrzeug bewegt. Das Resultat ist, dass die Sicherheit von hochautomatisierten Fahrerassistenzsystemen (FAS) in Zukunft in allen potentiellen Fahrzonen garantiert sein muss. Da viele Szenarien bisher unbekannt sind, stellt diese Absicherung die Automobilindustrie vor große Herausforderungen. In dieser Arbeit wird ein Konzept für die Klassifizierung von Fahrszenarien entwickelt und umgesetzt. Dafür wird ein künstliches neuronales Netz (KNN) mit 95 Prozent synthetischen und 5 Prozent realen Videodaten aus fünf Szenarienklassen trainiert. Diese synthetischen Daten werden zuvor mit der Simulationssoftware CarMaker generiert und automatisch annotiert. Mit diesem Ansatz soll es in Zukunft möglich sein, einen Klassifikator mit allen bisher bekannten Szenarien zu trainieren. Darauf basierend kann dieser Klassifikator bekannte Szenarien erkennen und bisher unbekannte Szenarien identifizieren. Mit der Entwicklung des Ansatzes und einem Proof-of-Concept liefert diese Arbeit einen theoretischen und praktischen Beitrag zur Absicherung von hochautomatisierten FAS.

Erklärung

Ich versichere hiermit, dass ich meine Masterarbeit selbständig und unter Beachtung der Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT) in der aktuellen Fassung angefertigt habe.

Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und wörtlich oder inhaltlich übernommene Stellen als solche kenntlich gemacht.

Karlsruhe, 31. Dezember 2018

Manuel Kaiser

Inhaltsverzeichnis

Abkürzungsverzeichnis	ix
Abbildungsverzeichnis	xi
Tabellenverzeichnis	xiii
1 Einleitung	1
1.1 Problemstellung und Motivation	1
1.2 Zielsetzung	2
2 Grundlagen	5
2.1 Hochautomatisiertes Fahren	5
2.1.1 Entwicklung von Fahrerassistenzsystemen	6
2.1.2 Klassifizierung von Szenarien	9
2.2 Künstliche Neuronale Netze	16
2.2.1 Einordnung im maschinellen Lernen	17
2.2.2 Entwicklung von künstlichen neuronalen Netzen	19
2.2.3 Convolutional Neural Networks	25
2.2.4 Recurrent Neural Networks und LSTMs	27
2.2.5 Training mit synthetischen Daten	30
2.2.6 Klassifizierung von Videos	31
3 Konzept	35
3.1 Struktur	36
3.2 Ansätze, Methoden, Werkzeuge	38

4 Implementierung	39
4.1 Definition der Fahrszenarien	39
4.2 Generierung synthetischer Daten	42
4.2.1 Simulation	42
4.2.2 Daten Annotation	46
4.3 Generierung realer Daten	48
4.4 Training	52
4.4.1 Vorbereitung der Daten	52
4.4.2 Design der KNN-Architekturen	54
4.4.3 Wahl der Komponenten	62
5 Ergebnis	67
5.1 Variation der Komponenten	67
5.2 Synthetische und reale Testdaten	71
5.3 Genauigkeit der einzelnen Szenarien	72
6 Zusammenfassung	75
6.1 Ergebnis und Diskussion	75
6.2 Ausblick	76
Literaturverzeichnis	xv
Appendix	xxiii

Abkürzungsverzeichnis

CNN Convolutional Neural Network

DNN Deep Neural Network

FAS Fahrerassistenzsystem

FRC Fuzzy Rule-Based Classifier

HiL Hardware-in-the-Loop

HMM Hidden Markov Model

kNN k-Nearest-Neighbor

KNN künstliches neuronales Netz

LSTM Long Short-Term Memory

MiL Model-in-the-Loop

RF Random Forest

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

SiL Software-in-the-Loop

SVM Support Vector Machine

tanh Tangens Hyperbolicus

ViL Vehicle-in-the-Loop

Abbildungsverzeichnis

2.1	Norm SAE J3016 für die Stufen des autonomen Fahrens	6
2.2	V-Modell	7
2.3	Schritte zur Testfallerstellung	8
2.4	Zusammenhang zwischen Szene und Szenario	10
2.5	Beispiel für ein funktionales, logisches und konkretes Szenario	12
2.6	Beispiel einer Klassifizierung mit zwei Klassen	18
2.7	Modell eines Perzeptrons	19
2.8	Aktivierungsfunktionen für Neuronen	21
2.9	Mehrlagiges Perzeptron mit zwei versteckten Schichten	22
2.10	Beispiel einer Unter- und Überanpassung eines Klassifikators	24
2.11	Beispiel einer Convolution-Operation	26
2.12	Beispiel einer Max-Pooling-Operation	27
2.13	Beispiel eines Convolutional Neural Networks	27
2.14	Beispiel von zwei verschiedenen RNN-Architekturen	28
2.15	Long Short-Term Memory-Zelle	29
2.16	Klassifizierung von einzelnen Bildern mit anschließender Klassifizierung des gesamten Videos	32
2.17	Schematische Darstellung von Klassifizierungsarchitekturen mit frühen, späten und langsamen Fusionen von Feature Maps	32
2.18	Klassifizierung von Videos mit einer CNN-LSTM-Architektur	33
3.1	Konzept dieser Arbeit	36
3.2	Schema der funktionalen Dekomposition	37
4.1	Konfiguration der simulierten Straße	44

4.2	Schema der simulierten Strecken 1 und 2	45
4.3	Beispiel eines simulierten Szenarios der Klasse <i>lane change left</i>	49
4.4	Routen für die Aufnahme der realen Bilddaten	50
4.5	Beispiel eines realen Szenarios der Klasse <i>lane change right</i>	52
4.6	Zwei Ansätze für die Videoklassifizierung	55
4.7	Vergleich von vortrainierten Convolutional Neural Networks	56
4.8	Inception-Module der Inception-V3 Architektur	57
4.9	Architektur des Inception-V3 Netzes	58
4.10	Inception-Modul der Xception-Architektur	59
4.11	Xception-Architektur	60
4.12	Regularisierung mit Dropout	61
4.13	Grundlegende KNN-Architekturen in dieser Arbeit	64
5.1	Konfusionsmatrix der Klassifizierung von realen Testdaten mit der Architektur F (Inception-V3, Long Short-Term Memory (LSTM), Dropout)	69
5.2	Genauigkeit der Trainings- und Testdaten während dem Training von zwei verschiedenen Architekturen	70

Tabellenverzeichnis

2.1	Bisherige Arbeiten zur Szenarienerkennung	15
3.1	Ansätze, Methoden und Werkzeuge dieser Arbeit	38
4.1	Definitionen der Szenarien <i>free cruising, approaching, following, catching up, overtaking, lane change left</i> und <i>lane change right</i>	42
4.2	Aufgezeichnete Signaldaten in CarMaker	43
4.3	Variablen und Werte die in der Simulation verwendet werden	45
4.4	Bedingungen für die Szenarien <i>free cruising, approaching, following, catching up, overtaking, lane change left</i> und <i>lane change right</i>	47
4.5	Anzahl der simulierten Szenarien nach Klasse	48
4.6	Vergleich von synthetischen und realen Datensätzen	51
4.7	Aufteilung von synthetischen und realen Szenarios auf Trainings-, Validierungs- und Testdaten	54
4.8	Komponenten die in den Experimenten variiert werden	63
4.9	KNN-Architekturen für die Klassifizierung von Bildsequenzen	65
4.10	KNN-Architekturen für die Klassifizierung einzelner Bilder	66
5.1	Ergebnisse der verschiedenen Architekturen bei der Klassifizierung der realen Testdaten	68
5.2	Ergebnisse der Klassifizierung von synthetischen und realen Testdaten	72
5.3	Genauigkeit der Klassifizierung von realen Testdaten auf der Ebene von Szenarienklassen	73

Kapitel 1

Einleitung

1.1 Problemstellung und Motivation

Autonomes Fahren - kaum ein Trend ist aktuell ein stärkerer Treiber für die Entwicklung von hochautomatisierten Fahrerassistenzsystemen (FAS). Doch die Absicherung dieser Fahrfunktionen stellt die Automobilindustrie vor große Herausforderungen. Aktuell werden Fahrfunktionen auf der Stufe 2 des autonomen Fahrens entwickelt [Com14]. Zukünftige hochautomatisierte Fahrerassistenzsystem (FAS) ab Stufe 3 unterscheiden sich in einem zentralen Punkt von Funktionen der Stufe 2. Die Kontrolle der Funktionen wird nicht mehr vom Fahrer, sondern vom System übernommen. Das bedeutet, dass die Rückfallebene zum Fahrer nicht mehr existiert. Damit wird die Frage aufgeworfen, wie die Sicherheit garantiert werden kann, wenn die Kontrolle durch den Fahrer nicht mehr existiert.

Aktuell folgt die Absicherung von FAS dem Testkonzept [Sch13]. Dabei werden Funktionen auf Basis von relevanten und bekannten Szenarien abgeleitet und entsprechende Testfälle erstellt und durchgeführt. Da ab Stufe 3 des autonomen Fahrens die Kontrolle durch den Fahrer nicht mehr existiert, müssen Fahrfunktionen zukünftig in allen potentiellen Szenarien sicher sein. Das schließt alle bisher bekannten und unbekannten Szenarien ein. Durch die steigende Anzahl unbekannter Szenarien decken Testfälle nicht mehr alle Szenarien ab und die Absicherung von FAS kann nicht mehr garantiert werden.

Weiterhin spielt der Einsatz von Verfahren des maschinellen Lernens eine zu-

nehmend bedeutende Rolle. Eine große Herausforderung für diese Algorithmen ist, dass Trainingsdaten, sofern sie auf realen, aufgezeichneten Daten beruhen, manuell annotiert werden müssen, was diesen Prozess sehr aufwändig macht. Ein weiteres Problem von realen Daten ist Ihre geringe Varianz. Während Standardsituationen sehr häufig vorkommen und damit auch mit einem KNN erlernt werden können, gibt es andere Situationen die selten auftreten, allerdings sehr kritisch sind. Es ist daher schwieriger ein neuronales Netz für diese Situationen, wie z.B. das „schneiden“ eines anderen Fahrzeugs beim Spurwechsel, zu trainieren.

Genau hier setzt diese Arbeit an. Es soll ein Beitrag dazu geliefert werden, auf Basis von größtenteils synthetischen Bilddaten, bisher unbekannte Szenarien zu finden. Durch die Verwendung von überwiegend simulierten Bilddaten soll der Ansatz auch in der Praxis Anwendung finden.

1.2 Zielsetzung

Das Ergebnis der Arbeit soll ein Ansatz sein, um bisher unbekannte Testfälle zu finden. Dafür sollen Szenarien ausgewählt und definiert werden. Auf der Basis dieser Definitionen sollen Videodaten zu diesen Szenarien mit einer Simulationssoftware erzeugt und automatisch annotiert werden. Zusammen mit einem kleinen Anteil realer Videodaten soll ein neuronales Netz trainiert werden, um anschließend reale Fahrszenarien klassifizieren zu können. Das Ziel ist ein Proof-of-Concept dieses Ansatzes mit einigen beispielhaften Szenarien.

In einem nächsten Schritt kann der Ansatz auf alle bisher bekannten Szenarien übertragen werden. Dabei müssen für alle Szenarien synthetische Trainingsdaten mit einer Simulationssoftware generiert werden. Zusammen mit einem kleinen Anteil realer Trainingsdaten kann ein KNN trainiert werden. Dieses KNN kann im Anschluss alle bisher bekannten Szenarien erkennen und unbekannte Szenarien identifizieren, zum Beispiel mit einer Schwelle der Klassifizierungsgenauigkeit.

Diese Arbeit soll einen theoretischen und praktischen Beitrag zum Training von neuronalen Netzen im Bereich des automatisierten Fahrens liefern. Die oben genannten Probleme sollen mit der Verwendung von simulierten Trainingsdaten adressiert und weiter untersucht werden. Der Fokus liegt dabei auf den folgenden

Hauptpunkten:

- Trainingsdaten müssen nicht mehr aufwendig manuell annotiert werden.
- Szenarien werden auf der Basis von Bilddaten klassifiziert.
- Bisher unbekannte Szenarien können gefunden werden.

Kapitel 2

Grundlagen

In diesem Kapitel werden die zum Verständnis benötigten Grundlagen für diese Arbeit erklärt. Dabei wird in Abschnitt 2.1 der Stand der Technik von automatisierten Fahrfunktionen und deren Entwicklung beschrieben. Im Abschnitt 2.2 wird maschinelles Lernen im Allgemeinen und im Speziellen KNNs, die in der Umsetzung dieser Arbeit verwendet werden, beschrieben.

2.1 Hochautomatisiertes Fahren

Hochautomatisiertes Fahren wird in den vergangenen Jahren zunehmend von der Automobilindustrie vorangetrieben. Aktuelle FAS, wie der Spurhalteassistent oder die Abstandsregelung, sind nach der Norm SAE J3016 (Abbildung 2.1) bei Level 2 des autonomen Fahrens eingeordnet. Dabei kontrolliert der Fahrer die Funktion des Assistenzsystems und überwacht die Umgebung. Ab Stufe 3 des autonomen Fahrens kontrolliert das System die Umgebung und die Funktion der FAS [Com14]. Die Kontrollinstanz des Fahrers existiert nicht mehr und die Sicherheit von Fahrfunktionen muss in jedem potentiellen Szenario garantiert sein. Das betrifft sowohl alle bekannten Szenarien, als auch alle bisher unbekannten Szenarien. Da für zukünftige FAS nicht mehr alle Szenarien bekannt sind, kann die Absicherung der Funktionen nicht garantiert werden. Das stellt Automobilhersteller und Automobilzulieferer vor große Herausforderungen.

In den folgenden Abschnitten wird erläutert, wie diesen Herausforderungen

aktuell begegnet wird. In Abschnitt 2.1.1 wird ein allgemeiner Überblick über die aktuellen Entwicklungsmethodiken für FAS gegeben. Danach werden in Abschnitt 2.1.2 bisherige Ansätze für die Klassifizierung von Fahrszenarien vorgestellt.

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/Deceleration	Monitoring of Driving Environment	Fallback Performance of Dynamic Driving Task	System Capability (Driving Modes)
Human driver monitors the driving environment						
0	No Automation	the full-time performance by the <i>human driver</i> of all aspects of the <i>dynamic driving task</i> , even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	Driver Assistance	the <i>driving mode-specific</i> execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the <i>human driver</i> perform all remaining aspects of the <i>dynamic driving task</i>	Human driver and system	Human driver	Human driver	Some driving modes
2	Partial Automation	the <i>driving mode-specific</i> execution by one or more driver assistance systems of both steering and acceleration/deceleration using information about the driving environment and with the expectation that the <i>human driver</i> perform all remaining aspects of the <i>dynamic driving task</i>	System	Human driver	Human driver	Some driving modes
Automated driving system (“system”) monitors the driving environment						
3	Conditional Automation	the <i>driving mode-specific</i> performance by an <i>automated driving system</i> of all aspects of the <i>dynamic driving task</i> with the expectation that the <i>human driver</i> will respond appropriately to a <i>request to intervene</i>	System	System	Human driver	Some driving modes
4	High Automation	the <i>driving mode-specific</i> performance by an <i>automated driving system</i> of all aspects of the <i>dynamic driving task</i> , even if a <i>human driver</i> does not respond appropriately to a <i>request to intervene</i>	System	System	System	Some driving modes
5	Full Automation	the full-time performance by an <i>automated driving system</i> of all aspects of the <i>dynamic driving task</i> under all roadway and environmental conditions that can be managed by a <i>human driver</i>	System	System	System	All driving modes

Copyright © 2014 SAE International. The summary table may be freely copied and distributed provided SAE International and J3016 are acknowledged as the source and must be reproduced AS-IS.

Abbildung 2.1: Norm SAE J3016 für die Stufen des autonomen Fahrens, entnommen aus [Com14]

2.1.1 Entwicklung von Fahrerassistenzsystemen

FAS sind Funktionen im Kraftfahrzeug, die den Fahrer unterstützen. Diese Systeme nutzen Sensordaten, wie Radar-, Ultraschall-, oder Kameradaten, aus dem Fahrzeug, um den Fahrer dann auf Basis der abgeleiteten Informationen zu unterstützen. Beispielsweise erkennt ein Spurhalteassistent wenn das Fahrzeug die Spur verlässt und kann die Fahrlinie korrigieren.

Um FAS effizient zu entwickeln und schon früh einzelne Komponenten testen zu können, werden diese in der Automobilindustrie häufig mit dem V-Modell als Vorgehensmodell entwickelt. Damit soll die Planungssicherheit und die Qualität

der Entwicklung erhöht werden. Das V-Modell ist ein chronologisches Vorgehensmodell und aus der Softwareentwicklung adaptiert [Bun05]. Das V-Modell kann in einen linken absteigenden und einen rechten aufsteigenden Ast unterteilt werden. Der linke Ast enthält die Funktionsanforderungen, die nach unten weiter detailliert und aufgeschlüsselt werden. Der rechte Ast umfasst aufsteigend Funktionstests auf dem jeweiligen Detaillierungsgrad [HK15]. Das V-Modell ist schematisch in Abbildung 2.2 dargestellt.

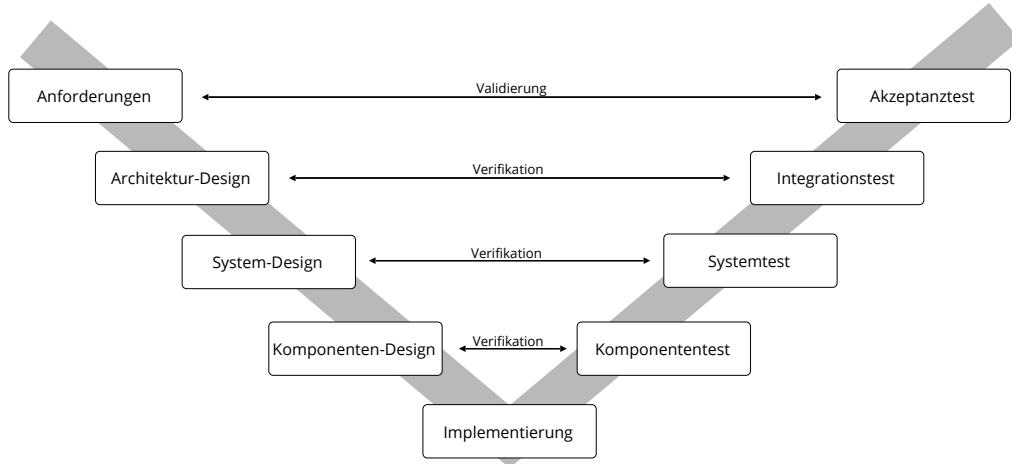


Abbildung 2.2: V-Modell [HK15]

Die Schritte auf dem absteigenden und aufsteigenden Ast haben jeweils eine Beziehung. Jeder Test auf dem aufsteigenden Ast verifiziert bzw. validiert den dazugehörigen Entwicklungsschritt auf dem absteigenden Ast. Dementsprechend werden oben im V-Modell die Kundenanforderungen auf dem absteigenden Ast erfasst und auf dem aufsteigenden Ast validiert. Unten im V-Modell werden einzelne Hardware- oder Softwarekomponenten entwickelt, die die entsprechenden Kundenanforderungen von oben lösen sollen. Diese werden anschließend auf dem aufsteigenden Ast verifiziert [HK15].

Testfälle für die Validierung und Verifikation

Die Validierung und Verifikation von FAS folgt dem Testkonzept. Ein Testkonzept umfasst die Analyse des Testobjektes, die Generierung von Testfällen, die Durchführung von Tests und schließlich die Testauswertung [Sch13]. Diese Schritte sind

in der Abbildung 2.3 dargestellt.

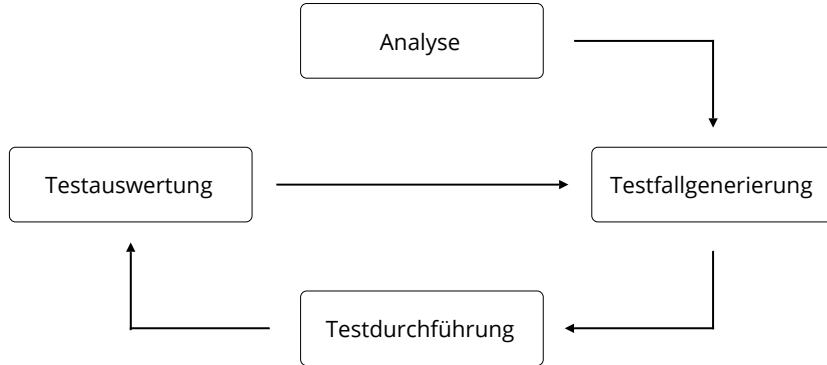


Abbildung 2.3: Schritte zur Testfallerstellung [Sch13]

Testfälle werden bereits möglichst früh im Entwicklungsprozess erstellt, um eine hohe Qualität von FAS und einzelnen Komponenten zu erreichen [WW15]. Hierfür werden in der Praxis virtuelle Fahrversuche eingesetzt. Die Idee ist eine stufenweise Digitalisierung von Komponenten aus dem realen Fahrversuch mit den Zielen die Reproduzierbarkeit zu steigern, den Aufwand zu reduzieren und insgesamt flexibler zu werden. Im virtuellen Fahrversuch werden in der frühen Konzeptphase alle Komponenten virtuell getestet und dann schrittweise durch Hardwarekomponenten ersetzt. Schließlich werden alle Komponenten im realen Fahrversuch auf der Straße mit einem realen Fahrer und anderen Verkehrsteilnehmern getestet [HK15].

Beim virtuellen Fahrversuch spielen die Konzepte Model-in-the-Loop (MiL), Software-in-the-Loop (SiL), Hardware-in-the-Loop (HiL) und Vehicle-in-the-Loop (ViL) eine wichtige Rolle. Mit MiL und SiL werden Funktionen auf Basis von Simulationsmodellen getestet [BF15], indem Hardwarekomponenten simuliert werden. Mit fortschreitender Entwicklung werden immer mehr Simulationskomponenten durch die entsprechende Hardware ersetzt und mit HiL getestet [HK15]. ViL schließt schließlich die Lücke zwischen virtuellem Fahrversuch und realem Fahrversuch. Für die Freigabe von FAS ist die Realfahrt die wichtigste Methode, da sie aktuell die beste Validierung bei annehmbaren ökonomischen Aufwand ist [WW15].

Mit steigender Automatisierung von FAS steigt die Anzahl möglicher Szenarien, in denen die Sicherheit der Funktionen auch ohne Fahrer garantiert sein muss.

Um alle Funktionen ausreichend testen zu können, steigt die Anzahl der benötigten Testfälle. Testfälle müssen alle potentiell eintretenden Szenarien, in denen das FAS zum Einsatz kommen kann, abdecken. Dadurch steigt mit hochautomatisierten Funktionen der Aufwand für die Validierung und Verifikation [Bac17]. Eine Möglichkeit für die Reduzierung von Testfällen ist es, kritische Situationen zu finden und Testfälle mit weniger kritischen Situationen zu entfernen [WW15].

Heute werden Testfälle auf der Basis von Anforderungen an die Fahrfunktionen und Standardsituationen abgeleitet und getestet. Dabei wächst heute schon, mit der Beschränkung auf Standardfälle, der Testaufwand [Sur18]. Mit zukünftigen hochautomatisierten FAS ab Stufe 3 des autonomen Fahrens muss das System in allen potentiellen Szenarien sicher sein und Testfälle dürfen nicht mehr auf die Standardsituationen begrenzt sein. Die Sicherheit des Gesamtsystems muss in breiterem Spektrum mit einer Vielzahl Szenarien ohne Eingriff des Fahrers garantiert werden können. Das bedeutet, dass die bisherigen Standardsituationen um neue Szenarien erweitert werden müssen [Sur18].

Für die Erstellung von Testfällen müssen daher alle potentiell eintretenden kritischen Szenarien bekannt sein. Das Ziel dieser Arbeit ist es, bekannte Szenarien zu klassifizieren und auf diese Weise bisher unbekannte und möglicherweise kritische Szenarien zu finden. Dies soll mit Hilfe von simulierten Daten geschehen, um die Skalierbarkeit mit angemessenem Aufwand garantieren zu können. Das Konzept hierzu wird im Detail in Kapitel 3 vorgestellt.

Im nächsten Abschnitt werden andere Arbeiten, welche die Klassifizierung von Szenarien untersucht haben, vorgestellt und wichtige Grundbegriffe definiert.

2.1.2 Klassifizierung von Szenarien

In diesem Abschnitt werden zu Beginn die Terminologien von Szene, Situation und Szenario unterschieden und definiert. Im Anschluss wird auf bisherige Arbeiten zur Szenarienerkennung eingegangen.

In dieser Arbeit werden Szene, Situation und Szenario nach Ulbrich et al. [Ulb15] definiert. In Abbildung 2.4 wird die Beziehung zwischen Szene und Szenario dargestellt.

Szene

Eine Szene ist eine Momentaufnahme von der Umgebung einschließlich der räumlichen Szenerie, allen dynamischen Elementen, der Selbstdarstellung aller Akteure und Beobachter, sowie die Beziehung zwischen diesen Entitäten. Nur in einer Simulation kann eine Szene vollständig und allumfassend beobachtet und erfasst werden (Ground Truth). In der realen Welt dagegen ist die Beschreibung einer Szene immer unvollständig, fehlerhaft, unsicher und subjektiv von einem oder mehreren Beobachtern.

Situation

Eine Situation ist die Gesamtheit aller Umstände, die für die Auswahl einer angemessenen Entscheidung zu einem bestimmten Zeitpunkt berücksichtigt werden müssen. Sie umfasst alle relevanten Zustände, Möglichkeiten und Einflussgrößen für ein Verhalten. Eine Situation wird abgeleitet von einer Szene durch die Auswahl von Informationen basierend auf kurzzeitigen sowie langfristigen Zielen und Werten. Eine Situation ist daher per Definition immer subjektiv von einem Beobachter.

Szenario

Ein Szenario besteht aus mehreren aufeinander folgenden Szenen und beschreibt deren zeitliche Entwicklung. Handlungen, Ereignisse, Ziele und Werte können für eine Charakterisierung der zeitlichen Entwicklung eines Szenarios spezifiziert werden. Im Gegensatz zu einer Szene, umfasst ein Szenario einen definierten Zeitraum.

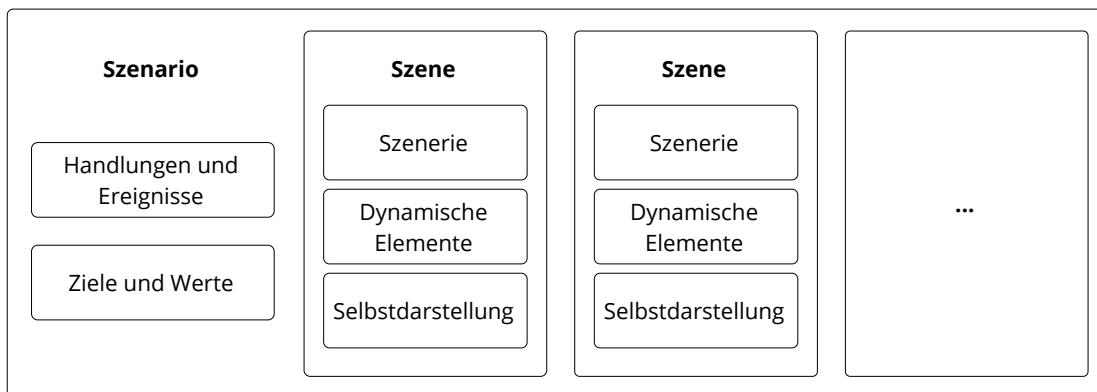


Abbildung 2.4: Zusammenhang zwischen Szene und Szenario [Ulb15]

Neben den Begriffen Szene, Situation und Szenario wird oft der Begriff Manöver verwendet. Bach et al. [BOS16] definieren ein Manöver als einen Zustand innerhalb eines Szenarios. Dabei sind Situationen jeweils Übergangsbedingungen zwischen einzelnen Manövern. So setzt sich beispielsweise das Szenario *Überholen* aus den Manövern *Spurwechsel nach links*, *Beschleunigung*, *Spurwechsel nach rechts* und *Bremsvorgang* zusammen. Zwischen den einzelnen Manövern gibt es Situation wie *langsameres Auto voraus* oder *langsameres Auto links überholt*, die jeweils ein neues Manöver einleiten.

Je nachdem, wie granular Szenarien bzw. wie grob Manöver definiert werden, können Szenarien und Manöver nicht klar voneinander abgegrenzt werden. So kann ein einzelnes Manöver auch bereits ein gesamtes Szenario darstellen. Zum Beispiel kann das Manöver *Spurwechsel* bereits als Szenario definiert werden. Aus diesem Grund wird in dieser Arbeit im Folgenden ausschließlich von Szenarien gesprochen.

Da sich diese Arbeit größtenteils auf die Klassifizierung von Szenarien fokussiert, wird in den folgenden Absätzen eine erweiterte Definition von Szenarien nach Bagschik et al. [Bag17] gegeben. Diese Definition unterteilt den Begriff in drei weitere Abstraktionsebenen: Funktionale, logische und konkrete Szenarien.

Funktionale Szenarien sind auf der semantischen Ebene formuliert. Entitäten und Beziehungen werden widerspruchsfrei in sprachlichen Texten beschrieben. Dabei ist das Vokabular klar definiert und wird eindeutig für alle zu beschreibenden Szenarien verwendet. Je nachdem wie detailliert ein Szenario beschrieben werden soll, muss ein geeignetes Vokabular definiert werden. Funktionale Szenarien können in einzelne oder mehrere logische Szenarien überführt werden.

Logische Szenarien sind detaillierter als funktionale Szenarien, indem Entitäten und Beziehung in quantitative Parameterbereiche übersetzt werden. Parameterbereiche können dabei mit statistischen Verteilungen (Normalverteilung, Gleichverteilung etc.) modelliert werden. Zusätzlich können Beziehungen zwischen Entitäten mit numerischen Bedingungen (e.g. Fahrzeug A muss auf derselben Spur fahren wie Fahrzeug B) oder Korrelationsfunktionen (e.g. Abstand zwischen Fahrzeug A und Fahrzeug B in Abhängigkeit der Geschwindigkeit) ausgedrückt werden.

Konkrete Szenarien haben den höchsten Detailgrad und Entitäten und Beziehungen werden mit festen Parametern definiert. Logische Szenarien können in

einzelne oder mehrere konkrete Szenarien überführt werden.

Ein Beispiel zu jeder Abstraktionsebene (funktional, logisch, konkret) ist in Abbildung 2.5 gegeben.

Funktionales Szenario	Logisches Szenario	Konkretes Szenario
<u>Basisstrecke</u> 3- streifige Autobahn in Kurve Begrenzung auf 100 km/h durch Verkehrszeichen rechts und links	<u>Basisstrecke</u> Breite Fahrstreifen [2,3..3,5] m Kurvenradius [0,6..0,9] km Pos_Verkerszeichen [0..200] m	<u>Basisstrecke</u> Breite Fahrstreifen [3,2] m Kurvenradius [0,7] km Pos_Verkerszeichen [150] m
<u>Stationäre Objekte</u> -	<u>Stationäre Objekte</u> -	<u>Stationäre Objekte</u> -
<u>Bewegliche Objekte</u> Ego, Stau; Interaktion: Ego in Manöver „Annähern“ auf mittleren Fahrstreifen, Stau zähfließend	<u>Bewegliche Objekte</u> Stauende_Pos [10..200] m Stau_Geschw [0..30] km/h Ego_Abstand [50..300] m Ego_Geschw [80..130] km/h	<u>Bewegliche Objekte</u> Stauende_Pos [40] m Stau_Geschw [30] km/h Ego_Abstand [300] m Ego_Geschw [100] km/h
<u>Umwelt</u> Sommer, Regen	<u>Umwelt</u> Temperatur [10..40] °C Tröpfchengröße [20..100] µm	<u>Umwelt</u> Temperatur [20] °C Tröpfchengröße [30] µm

The diagram consists of three columns of tables above a horizontal bar. The bar is labeled 'Abstraktionslevel' on the left and 'Szenarienanzahl' on the right. The first column has 4 rows, the second has 3 rows, and the third has 3 rows. This visualizes how the number of scenarios decreases as the abstraction level increases from functional to concrete.

Abbildung 2.5: Beispiel für ein funktionales, logisches und konkretes Szenario [Bag17]

Bisherige Arbeiten zur Klassifizierung von Fahrszenarien

Wie in Abschnitt 2.1.1 beschrieben ist die Klassifizierung von Szenarien ein wichtiges Element für die Erstellung von Testfällen und die Sicherung von FAS. In den vergangenen Jahren wurden bereits einige Arbeiten zur Klassifizierung von Szenarien veröffentlicht. In den folgenden Absätzen werden jeweils die Datenbasis, die Klassifizierungsmethode und die jeweiligen Fahrszenarien der bisherigen Arbeiten seit 2014 kurz vorgestellt.

Für die Klassifizierung wurden verschiedene Sensordaten aus dem Fahrzeug verwendet. Die Autoren von fünf Arbeiten haben Beschleunigungs-, Gyroskop-, GPS- und Magnetometer-Daten mithilfe eines Smartphones in einem Fahrzeug aufgezeichnet [XHK18; Cer16; WK16; CHK16; ABR16]. Die Verwendung von Smartphone-Daten wurde mit der einfachen und kostengünstigen Umsetzung be-

gründet. Vier andere Arbeiten verwendeten Sensordaten wie *Lenkwinkel*, *Fahrzeuggeschwindigkeit*, *laterale Geschwindigkeit*, *Giergeschwindigkeit* und die *Position des Gas- und Bremspedals*, die sie aus dem CAN-Bus des Fahrzeugs auslasen [ZH17; ZSH15; Li15; ZSH14]. Drei weitere Arbeiten basierten ihre Experimente auf Daten aus einem Fahrsimulator [Sun17; Zhe16] und realen Testfahrten [Gru17]. Die verwendeten Daten waren der *laterale Abstand zwischen Fahrzeug und Fahrbahnmarkierung*, *Spurabfahrtsbetrag*, *Beschleunigung*, *Lenkwinkel*, *Lenkgeschwindigkeit*, *Lenkmoment* und *Ort*, *Ausrichtung*, und *Geschwindigkeit* des Ego-Fahrzeugs und benachbarten Objekten. Dabei wurde nicht weiter spezifiziert, wie die Daten ausgelesen wurden.

Auf Basis der Sensordaten wurden verschiedene Klassifikatoren erstellt. Es wurden die Methoden Support Vector Machine (SVM) [Sun17; Cer16; WK16; CHK16; Zhe16; ZSH15], Random Forest (RF) [XHK18; Cer16; Zhe16], k-Nearest-Neighbor (kNN) [ZH17; CHK16; Zhe16], Hidden Markov Model (HMM) [ZH17; Li15], Fuzzy Rule-Based Classifier (FRC) [Cer16; ABR16], Bayesian Inference Model [Sun17], Convolutional Neural Network (CNN) [Gru17], Decision Tree [ZSH14] und Naive Bayes [CHK16] verwendet. Da die Methoden für diese Arbeit nicht weiter relevant sind, werden sie an dieser Stelle nicht im Detail erläutert, es wird lediglich auf die jeweiligen Quellen verwiesen.

In Tabelle 2.1 sind alle dem Autor bekannten Arbeiten seit 2014 zusammengefasst. Neben den verwendeten Methoden zur Klassifizierung sind die jeweils verwendeten Sensordaten und die klassifizierten Szenarien aufgeführt.

Quelle	Sensordaten	Klassifikator	Szenarien
[XHK18]	Beschleunigung, Gyroskop und GPS von einem Smartphone im Fahrzeug	RF	Abbiegen, links abbiegen, rechts abbiegen, beschleunigen, bremsen, stoppen, Spurwechsel nach links, Spurwechsel nach rechts
[ZH17]	Lenkwinkel und Fahrzeuggeschwindigkeit aus dem CAN-Bus	kNN, HMM	Spurwechsel nach links, Spurwechsel nach rechts, Spur halten

Quelle	Sensordaten	Klassifikator	Szenarien
[Sun17]	Lateraler Abstand zwischen Fahrzeug und Fahrbahnmarkierung von einem Fahrsimulator	SVM, Bayesian Inference Model	Spurwechsel nach links, Spurwechsel nach rechts, Spur halten
[Gru17]	Ort, Ausrichtung und Geschwindigkeit des Ego-Fahrzeugs und benachbarten Objekten von realen Testfahrten	CNN auf Basis von gestapelten Positionsgittern der Objekte	Frei fahren, anderes Fahrzeug voraus, anderes Fahrzeug überholt Ego-Fahrzeug, Querverkehr vor Ego-Fahrzeug
[Cer16]	Beschleunigung von einem Smartphone im Fahrzeug	RF, SVM, FRC	Einparken, geparkt, frei fahren, stoppen
[WK16]	Beschleunigung, Gyroskop, GPS und Magnetometer von einem Smartphone im Fahrzeug	SVM	Stoppen, beschleunigen, bremsen, links abbiegen, rechts abbiegen
[CHK16]	Beschleunigung, Gyroskop und GPS von einem Smartphone im Fahrzeug	SVM, kNN, Naive-Bayes	Stoppen, beschleunigen, frei fahren, bremsen, Spurwechsel nach links, Spurwechsel nach rechts, links abbiegen, rechts abbiegen, in Kreisverkehr eintreten, aus Kreisverkehr austreten
[Zhe16]	Spurabfahrtsbetrag, Beschleunigung, Lenkwinkel, Lenkgeschwindigkeit und Lenkmoment von einem Fahrsimulator	SVM, kNN, RF	Spurwechsel nach links, Spurwechsel nach rechts, Spur halten

Quelle	Sensordaten	Klassifikator	Szenarien
[ABR16]	Beschleunigung, Gyroskop und GPS von einem Smartphone im Fahrzeug	FRC	Lenken, beschleunigen, bremsen, Bodenwelle
[ZSH15]	Fahrzeuggeschwindigkeit und Lenkwinkel aus dem CAN-Bus	SVM	links abbiegen, rechts abbiegen, Spurwechsel nach links, Spurwechsel nach rechts, Kurve nach links, Kurve nach rechts, geradeaus fahren, stoppen
[Li15]	Fahrzeuggeschwindigkeit, Position des Gas- und Bremspedals, Lenkwinkel, Laterale Beschleunigung und Giergeschwindigkeit aus dem CAN-Bus	HMM	Spurwechsel nach links, Spurwechsel nach rechts, Spur halten
[ZSH14]	Fahrzeuggeschwindigkeit, Lenkwinkel, Drehzahl und Position des Gas- und Bremspedals aus dem CAN-Bus	Decision Tree auf Basis von Schwellenwerten	links abbiegen, rechts abbiegen, Spurwechsel nach links, Spurwechsel nach rechts, Kurve nach links, Kurve nach rechts, geradeaus fahren, stoppen

Tabelle 2.1: Bisherige Arbeiten zur Szenarienerkennung

Die Datenerhebungen in den vergangenen Arbeiten wurden vor dem Hintergrund durchgeführt, vordefinierte Szenarien zu erkennen. Von diesen Szenarien wurden die benötigten Sensordaten abgeleitet und dann mit bestimmten Methoden verschiedene Klassifikatoren erstellt. Bis auf in der Arbeit von Gruner [Gru17] werden für die Klassifizierung von Fahrszenarien bisher keine Deep Neural Networks (DNNs) verwendet. Ebenfalls verwendet Gruner in seiner Arbeit keine Kamerabilder, sondern gestapelte Matrizen, auf denen jeweils die Positionen aller

Verkehrsteilnehmer markiert sind. Nach Gruner [Gru17] wird sich die zukünftige Forschung mit tieferen Netzstrukturen wie Recurrent Neural Networks (RNNs) beschäftigen, um zeitabhängige Szenarien noch besser zu verstehen.

Mit den bisherigen Ansätzen können bekannte Szenarien sehr zuverlässig erkannt werden. Bisher unbekannte Szenarien werden allerdings nur sehr bedingt erkannt, weil die Datengrundlage auf Basis der bekannten Szenarien optimiert ist. Das bedeutet, dass Daten, die für die Erkennung von bisher unbekannten Szenarien möglicherweise relevant sind, nicht erfasst und daher nicht für die Erstellung des Klassifikators verwendet werden.

Im Gegensatz zu den bisherigen Arbeiten, soll in dieser Arbeit die Verwendung von Kameradaten für die Klassifizierung von Szenarien untersucht werden. Die Idee ist, dass Videodaten den gleichen Informationsgehalt besitzen wie das Sichtfeld eines realen Fahrers. Ein Fahrer kann auf Basis seines Sichtfeldes alle Szenarien erkennen und auf der Basis die Umgebung und alle Fahrfunktionen überwachen. Da in Zukunft das System diese Aufgabe übernehmen wird, soll in dieser Arbeit ein Klassifikator mit genau diesem Sichtfeld, mit Videodaten, trainiert werden. Damit sollen auch bisher unbekannte Einflüsse, die auf den Bildern zu sehen sind, für die Klassifizierung berücksichtigt werden. In einem weiteren Schritt kann ein Klassifikator, der auf diese Weise mit allen bisher bekannten Szenarien trainiert wurde, bekannte Szenarien erkennen und bisher unbekannte Szenarien identifizieren. Der Ansatz wird im Detail in Kapitel 3 vorgestellt.

2.2 Künstliche Neuronale Netze

In diesem Kapitel werden künstliche Neuronale Netze (KNNs) mit einem Schwerpunkt auf Bilderkennung mit Convolutional Neural Networks (CNNs) und Sequenzerkennung mit Long Short-Term Memorys (LSTMs) eingeführt. Im folgenden Abschnitt 2.2.1 werden KNNs in den Gesamtkontext von maschinellem Lernen gestellt. Im Anschluss werden in Abschnitt 2.2.2 die Grundlagen zu KNNs erläutert. Danach werden komplexe Architekturen von KNNs zur Bilderkennung in Abschnitt 2.2.3 und zur Sequenzerkennung in Abschnitt 2.2.4 erklärt. In Abschnitt 2.2.5 wird auf das Training mit synthetischen Daten eingegangen und im letzten

Abschnitt 2.2.6 wird die aktuelle Forschung zu Videoklassifizierung vorgestellt.

2.2.1 Einordnung im maschinellen Lernen

Maschinelles Lernen wird oft als ein Teil des Bereichs künstliche Intelligenz beschrieben. Dabei wird maschinelles Lernen nach Mitchell [Mit97] wie folgt definiert:

„Ein Computerprogramm lernt aus der Erfahrung E in Bezug auf eine Klasse von Aufgaben T und dem Leistungsmaß P, wenn seine Leistung, gemessen mit P, bei Aufgaben aus T sich mit Erfahrung E verbessert.“

Da maschinelles Lernen sehr viele Bereiche umfasst, wird hier nur auf die relevanten Teile für diese Arbeit eingegangen und auf [Mit97] verwiesen. Maschinelles Lernen kann in drei verschiedenen Kategorien eingeteilt werden. Diese werden in den folgenden Absätzen beschrieben.

Überwachtes Lernen (engl. supervised learning) beschreibt einen Lernprozess, in dem die Trainingsdaten sowohl Inputvektoren als auch die zugehörigen Zielvektoren enthalten [Bis06]. Ein Beispiel dafür ist ein Klassifizierungsproblem von Buchstaben, bei dem sowohl die Bilder der einzelnen Buchstaben, als auch deren zugehörige Klasse (abgebildeter Buchstabe) einem Trainingsalgorithmus übergeben werden. Neben Klassifizierungsproblemen fallen auch Regressionsprobleme in diese Kategorie.

Beim unüberwachten Lernen (engl. unsupervised learning) enthalten die Trainingsdaten ausschließlich die Inputvektoren, ohne die dazugehörigen Zielvektoren. Das Ziel dabei ist es, Muster in den gegebenen Daten zu erkennen, um beispielsweise Cluster zu bilden [Bis06]. Das Clustering von Kundengruppen, die bisher unbekannt waren, fällt in diese Kategorie des maschinellen Lernens.

Das verstärkende Lernen (engl. reinforcement learning) ist eine Methodik in welcher der Trainingsalgorithmus mit Situationen konfrontiert wird und jeweils aus einer Reihe von gegebenen Handlungen wählen kann. Das Ziel dabei ist es, das Endergebnis, das auf der Wahl aller Handlungen basiert, zu maximieren [SB98]. Ein Beispiel hierfür ist das selbstständige Erlernen des Brettspiels Schach.

Klassifizierung

In dieser Arbeit wird ein Konzept für die Klassifizierung von Fahrszenarien - damit in der Kategorie des überwachten Lernens - entwickelt und umgesetzt. Das Ziel von Klassifizierungsalgorithmen ist es, gegebene Objekte auf Basis ihrer Eigenschaften einer Klasse zuzuordnen. Dabei sollen die Objekte innerhalb einer Klasse eine möglichst geringe Varianz und zwischen verschiedenen Klassen eine möglichst hohe Varianz besitzen. Klassifizierungsalgorithmen arbeiten dafür mit Trainingsdaten, die aus Inputvektoren und Zielvektoren bestehen. Ein Inputvektor enthält alle Eigenschaften und der Zielvektor die jeweilige Klasse des Objekts. In Abbildung 2.6 ist beispielhaft die Klassifizierung eines Datensatzes mit zwei Klassen dargestellt. Die Objekte im Datensatz haben jeweils die Eigenschaften x_1 und x_2 .

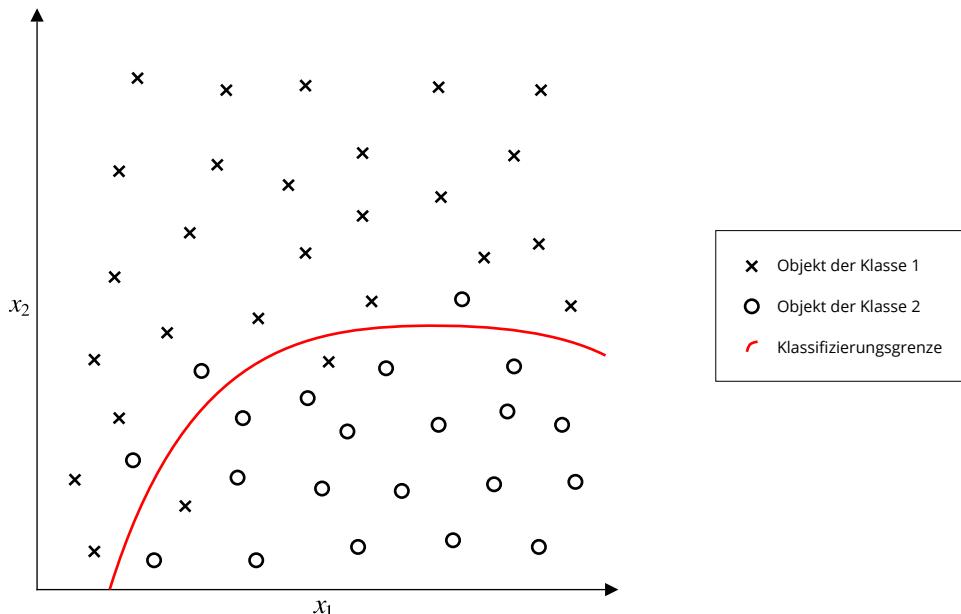


Abbildung 2.6: Beispiel einer Klassifizierung mit zwei Klassen

In den folgenden Abschnitten werden schrittweise CNNs und LSTMs eingeführt. Mit diesen Komponenten werden später verschiedene Klassifikatoren für die Erkennung von Fahrszenarien entwickelt und trainiert.

2.2.2 Entwicklung von künstlichen neuronalen Netzen

KNNs wurden ursprünglich als ein Modell der Informationsverarbeitung von biologischen Gehirnen entwickelt [MP43]. Dabei ist die kleinste Einheit in einem KNN ein einzelnes Neuron. Rosenblatt [Ros58] entwickelte ein Modell eines Neurons als binären Klassifikator. Dieses sogenannte Perzeptron setzt sich aus einem Eingangsvektor x_1, \dots, x_n , einem Vektor mit Gewichten w_1, \dots, w_n , einer Summenfunktion \sum , einer Aktivierungsfunktion φ mit einem Schwellenwert θ und einem Aktivierungswert $o(\vec{x})$ zusammen. Abbildung 2.7 zeigt das Modell eines Perzeptrons.

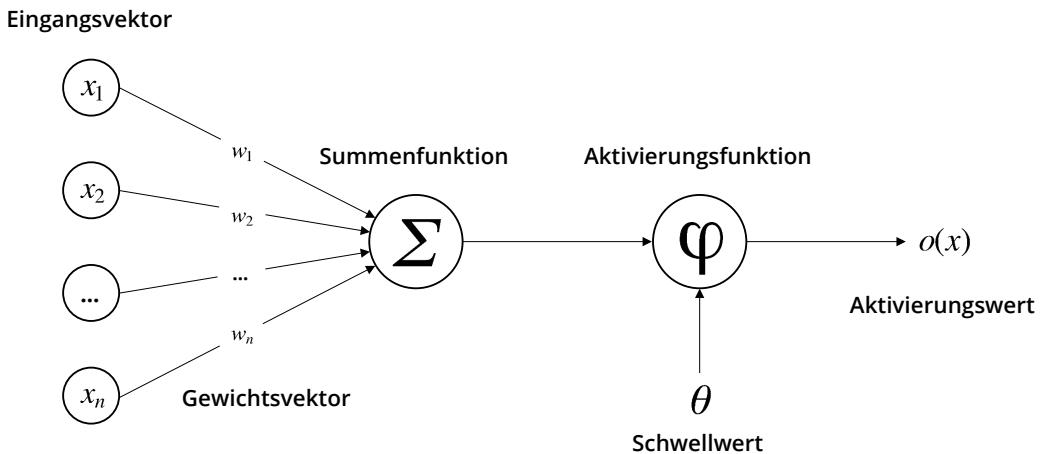


Abbildung 2.7: Modell eines Perzeptrons [Ros58]

Um den Aktivierungswert $o(\vec{x})$ zu berechnen wird zunächst die gewichtete Summe mit dem Eingangsvektor \vec{x} und den Gewichten \vec{w} gebildet. Im Anschluss wird mit der Aktivierungsfunktion φ eine Klasse bestimmt. Beim ursprünglichen Perzeptron handelt es sich dabei um eine Schwellenwertfunktion, welche die zwei Werte -1 und 1 annehmen kann. Damit ist das Perzeptron ein binärer Klassifikator und kann die logischen Operationen *AND*, *OR* und *NOT* ausführen. Die logische Operation *XOR* kann mit einem einzelnen Perzeptron nicht abgebildet werden [MP69].

$$\varphi(\vec{x}, \vec{w}) = \begin{cases} 1 & \text{wenn } \vec{x} * \vec{w} \geq \theta \\ 0 & \text{sonst} \end{cases} \quad (2.1)$$

Die Klassifizierung eines Objekts aus der Klasse t mit den Eigenschaften \vec{x} ist richtig, wenn das Ergebnis o der Aktivierungsfunktion $\varphi(\vec{x}, \vec{w})$ der tatsächlichen Klasse des Objekts t entspricht. Wenn die Klasse nicht richtig erkannt wurde $o(\vec{x}) \neq t$, werden die Gewichte \vec{w} entsprechend der Perzeptron-Lernregel angepasst. Diese Lernregel ist ein wichtiger Vorteil von Rosenblatts Perzeptron [Ros58] gegenüber des Neurons von McCulloch und Pitts [MP43], weil die Gewichte erlernt werden können.

Vor dem Training werden die Gewichte \vec{w} zufällig initialisiert. In jedem Trainingsschritt wird überprüft ob die berechnete Klasse $o(\vec{x})$ der tatsächlichen Klasse t entspricht. Wenn $o(\vec{x}) = t$ werden die Gewichte nicht verändert und der nächste Trainingsschritt wird ausgeführt. Wenn $o(\vec{x}) \neq t$ werden die Gewichte nach der Perzeptron-Lernregel aktualisiert:

$$w_i^{neu} = w_i^{alt} + \Delta w_i \quad (2.2)$$

$$\Delta w_i = \eta * (t - o) * x_i \quad (2.3)$$

Die Lernrate η kann angepasst werden und bestimmt wie stark die Gewichte in jedem Trainingsschritt verändert werden. Üblicherweise werden Lernraten zwischen 0,01 und 0,0001 gewählt.

Neben der Schwellenwertfunktion werden für die Aktivierung von einzelnen Neuronen verschiedene Aktivierungsfunktionen verwendet. Die am meisten verwendeten Funktionen hierfür sind die Tangens Hyperbolicus (\tanh)-Funktion

$$\varphi(x) = \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}, \quad (2.4)$$

die Sigmoid- oder S-Funktion

$$\varphi(x) = \frac{1}{1 + e^{-x}} \quad (2.5)$$

und die Rectified Linear Unit (ReLU)-Funktion

$$\varphi(x) = \max(0, x). \quad (2.6)$$

Diese Funktionen sind, zusammen mit der zuvor vorgestellten Schwellenwertfunk-

tion, in Abbildung 2.8 dargestellt.

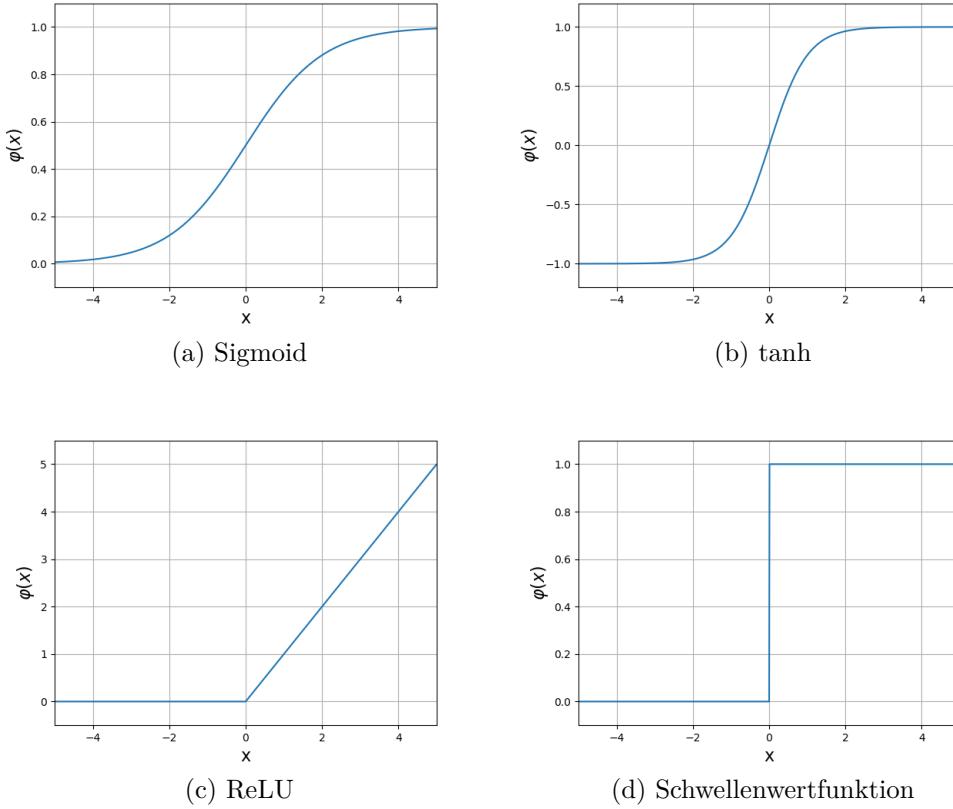


Abbildung 2.8: Aktivierungsfunktionen für Neuronen

Mit diesen Aktivierungsfunktionen und der Verwendung mehrerer Perzeptoren kann ein KNNs, oder auch mehrlagiges Perzeptron (engl. multilayer perceptron), modelliert werden. Diese Netze können auf unterschiedliche Probleme angewendet werden und überwinden die Schwächen eines einzelnen Perzeptrons (ausschließlich binäre Klassifizierung, keine *XOR*-Operation). Ein mehrschichtiges KNN besteht aus mindestens zwei Schichten, einer Ergebnis-Schicht und mindestens einer verborgenen Schicht. Der Eingangsvektor \vec{x} wird nicht als eine Schicht gezählt. Jede Schicht besteht aus $1, \dots, n$ Neuronen (Perzeptronen), die im Modell als Knoten modelliert sind. Die Neuronen einer Schicht bestehen jeweils aus einer gewichteten Summe und einer Aktivierungsfunktion und sind jeweils mit dem Eingangsvektor oder den Neuronen der vorherigen und nachfolgenden Schichten

verbunden. Diese Verbindungen werden als Kanten modelliert und repräsentieren die Gewichte mit denen die Neuronen verknüpft sind. Abbildung 2.9 zeigt das Schema eines dreischichtigen KNNs.

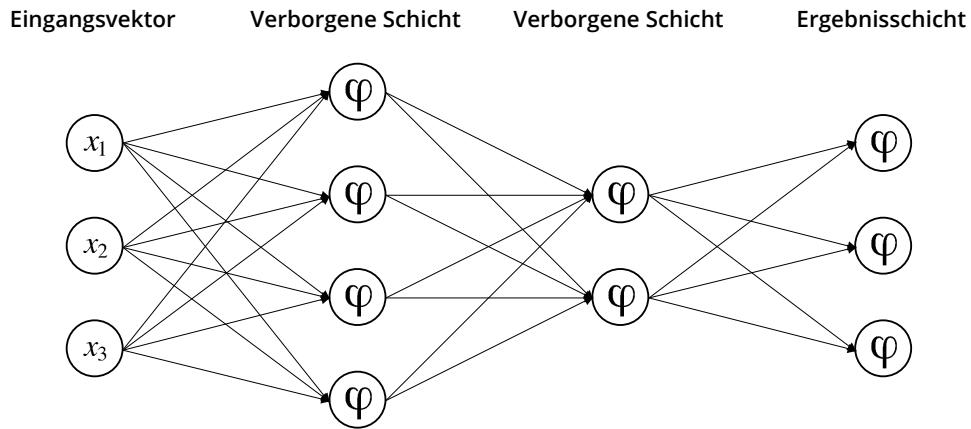


Abbildung 2.9: Mehrlagiges Perzeptron mit zwei versteckten Schichten

Das Training eines mehrschichtigen KNN funktioniert analog zu dem Training eines einzelnen Perzeptrons. In jedem Trainingsschritt wird ein Ergebnis $o(\vec{x})$ berechnet und mit dem richtigen Ergebnis t verglichen. Mit dem Eingangsvektor \vec{x} werden die Aktivierungswerte y_j des Vektors \vec{y} der ersten verborgenen Schicht wie folgt berechnet:

$$y_j = \varphi\left(\sum_i w_{ij} * x_i\right) \quad (2.7)$$

Dabei ist w_{ij} die Gewichtung der Kante zwischen dem Eingangswert x_i und dem verborgenem Neuron j . Alle weiteren verborgenen Schichten sowie der Ergebnisvektor werden auf die gleiche Weise, mit den Aktivierungswerten der vorherigen Schicht als Eingangsvektor, berechnet. Für binäre Klassifizierungsprobleme kann beispielsweise die Sigmoid-Funktion als Aktivierungsfunktion für die Ergebnisschicht gewählt werden. Dann ist das Ergebnis o eine Zahl zwischen 0 und 1 und beschreibt die Wahrscheinlichkeit, dass der Eingangsvektor \vec{x} zur Klasse c_1 gehört. Dementsprechend beschreibt $1 - o$ die Wahrscheinlichkeit der Klasse c_2 . Bei Klassifizierungsproblemen mit mehreren Klassen wird häufig die Softmax-Funktion als Aktivierungsfunktion φ gewählt, um die Wahrscheinlichkeiten der einzelnen Klas-

sen c_k zu bestimmen [Bri90]:

$$o_k = p(c_k|x) = \frac{e^{x^\top w}}{\sum_{j=1}^C e^{x_j^\top w}} \quad (2.8)$$

Dabei beschreibt x den Vektor mit Aktivierungswerten aus der vorherigen Schicht, w den Gewichtsvektor und C die Anzahl der Klassen c_k . Auf diese Weise können die Wahrscheinlichkeiten $p(c_k|x)$ für jede Klasse c_k , gegeben den Aktivierungswerten x , berechnet werden.

Für das Erlernen von Gewichten in mehrschichtigen KNNs wird die Fehlerrückführung (engl. error backpropagation) verwendet. Die Idee bei diesem Verfahren ist es, eine Fehlerfunktion, welche die Abweichung zwischen der berechneten und der tatsächlichen Klasse beschreibt, zu definieren und anschließend zu minimieren [Bis06]. Dafür wird häufig die mittlere quadratische Abweichung als Fehlerfunktion verwendet:

$$E = \frac{1}{n} \sum_{j=1}^n (t_j - o_j)^2 \quad (2.9)$$

Dabei beschreiben t_j die tatsächliche Klasse, o_j die errechnete Klasse und n die Anzahl der Klassen. Auf Basis dieser Fehlerfunktion läuft die Fehlerrückführung iterativ in den folgenden Schritten ab [Bis06]:

1. Auf Basis eines Eingangsvektors \vec{x} werden die Aktivierungswerte \vec{y}_j aller versteckten Schichten j und der Ergebnisvektor \vec{o} berechnet.
2. Der errechnete Ergebniswert o_j wird mit dem erwarteten Ergebnis t_j verglichen und die Differenz wird berechnet. Diese Differenz wird als Fehler bezeichnet.
3. Auf Basis dieser Differenz werden die Gewichte zwischen allen Neuronen geändert, mit dem Ziel die Differenz bei der nächsten Iteration zu verringern. Die Änderung wird wie folgt berechnet:

$$w_{ij}^{neu} = w_{ij}^{alt} + \Delta w_{ij} \quad (2.10)$$

mit

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \quad (2.11)$$

Dabei beschreibt w_{ij} das Gewicht zwischen Neuron i und Neuron j , η eine feste Lernrate die beeinflusst, wie stark Gewichte geändert werden und E die oben definierte Fehlerfunktion.

Mit diesem Algorithmus werden die Gewichte von mehrschichtigen KNNs bei jedem Trainingsschritt angepasst. Eine Herausforderung beim Trainieren von KNNs ist es, mit bisher unbekannten Daten gute Ergebnisse zu erzielen [Sri14]. Das bedeutet, dass im Laufe des Trainings die Gewichte so angepasst werden müssen, dass das Modell nach dem Training nicht nur mit den Trainingsdaten gute Ergebnisse erzielen kann, sondern auch mit Daten, die es zuvor nicht verarbeitet hat. Diese Fähigkeit eines Modells wird Generalisierbarkeit genannt.

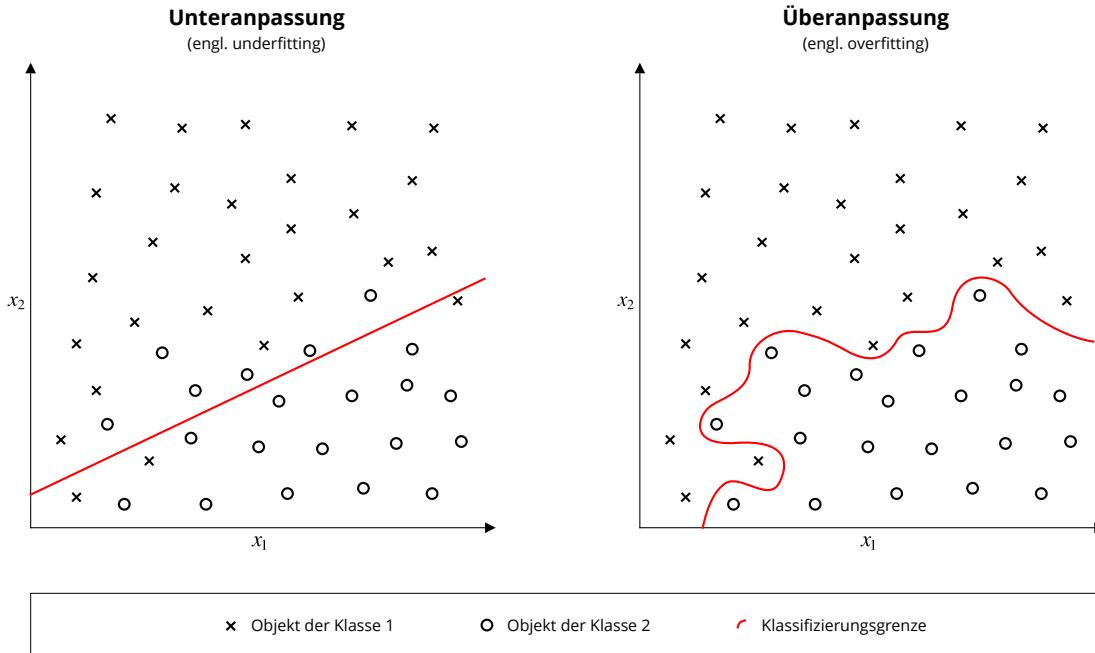


Abbildung 2.10: Beispiel einer Unter- und Überanpassung eines Klassifikators

Wenn ein Modell hingegen nur mit den Trainingsdaten gute Ergebnisse erzielt, spricht man von Überanpassung (engl. overfitting). Im Gegensatz dazu spricht man von Unteranpassung (engl. underfitting), wenn ein Modell schon mit den

Trainingsdaten sehr schlechte Ergebnisse erzielt. In Abbildung 2.10 ist beispielhaft eine Unter- und Überanpassung eines binären Klassifikators dargestellt.

Heute werden in Anwendungen häufig tiefen neuronale Netze (engl. deep neural networks) verwendet. Von einem tiefen neuronalen Netz spricht man, wenn es viele versteckte Schichten besitzt. Damit können Merkmale auf verschiedenen Abstraktionsebenen erkannt werden [LBH15]. In den folgenden Absätzen 2.2.3 und 2.2.4 werden zwei verschiedene Architekturen von tiefen neuronalen Netzen vorgestellt, die in Kapitel 4 für das Klassifizierungsproblem dieser Arbeit angewendet werden.

2.2.3 Convolutional Neural Networks

Ein Convolutional Neural Network (CNN) ist eine Architektur eines KNNs und gilt heute als Stand der Technik für Probleme in der Bilderkennung [KSH12]. Die Architektur eines CNNs besteht grundsätzlich aus einer oder mehreren Convolution- und Pooling-Schichten [LKF10]. Die Abfolge dieser Schichten kann sich beliebig oft wiederholen und wird am Ende mit einer oder mehreren Fully-Connected-Schichten für die Klassifizierung ergänzt. In den folgenden Absätzen werden die Funktionsweisen der Convolution- und der Pooling-Schicht erklärt. Eine Fully-Connected-Schicht entspricht einer Schicht im mehrlagigen Perzeptron wie sie im vorherigen Abschnitt 2.2.2 beschrieben wurde. In Abbildung 2.13 ist ein gesamtes CNN dargestellt.

Die Convolution-Schicht besteht aus einer Convolution-Operation gefolgt von einer Aktivierungsfunktion. Die Grundidee dieser Schicht ist die Extraktion von Merkmalen aus einem Bild oder anderen Inputdaten. Dabei werden in den ersten Schichten Merkmale auf einer niedrigen Ebene (engl. low-level features) und in späteren Schichten zunehmend abstraktere Merkmale (engl. high-level features) extrahiert. Bei der Convolution-Operation (oder auch Faltung) wird ein ausgewählter Filter mit einer festgelegten Schrittgröße (engl. stride) über das Bild bewegt und bei jedem Schritt der entsprechende Ausgabewert berechnet [LeC98]. Diese Operation ist in Abbildung 2.11 dargestellt.

In diesem Beispiel handelt es sich um ein Bild mit den Dimensionen $6 \times 6 \times 1$ Pixel, also einem zweidimensionalen Bild mit einem Farbkanal. Der Filter hat die Dimensionen $3 \times 3 \times 1$. In dem Beispiel ist die aktuell dargestellte Rechnung wie

33	15	1	67	84	73
93	84	17	38	49	28
36	72	83	94	82	84
59	29	40	18	16	2
33	33	8	76	69	33
32	41	62	53	12	25

*

0	0	1
0	0	0
1	0	0

=

37	139	167	167
76	67	89	46
116	127	90	160
72	59	78	55

Abbildung 2.11: Beispiel einer Convolution-Operation

folgt:

$$0 * 33 + 0 * 15 + 1 * 1 + 0 * 93 + 0 * 84 + 0 * 17 + 1 * 36 + 0 * 72 + 0 * 83 = 37$$

Nach dieser Berechnung bewegt sich der Filter einen Schritt weiter nach rechts und der nächste Wert wird berechnet. Am rechten Rand des Bildes angekommen, wird der Filter eine Schrittlänge weiter nach unten gesetzt und wieder an den linken Rand gesetzt. Dies wiederholt sich bis der Filter am rechten unteren Rand des Bildes angekommen ist. Bei einem dreidimensionalen Bild, mit den drei Farbkanälen als dritte Dimension, hat der Filter ebenfalls drei Dimensionen. Die Berechnung wird analog durchgeführt. Bei dieser Operation kann die Größe des Filters und die Schrittgröße variiert werden. Es können auch verschiedenen Filter verwendet werden, um unterschiedliche Merkmale zu extrahieren. Außerdem können sogenannte Padding-Methoden eingesetzt werden um den Rand der Bilder künstlich zu erweitern. Beim sogenannten Zero-Padding wird beispielsweise ein beliebig breiter Rand mit den Werten 0 um das Bild gelegt. Somit kann sich ein Filter auch über die existierenden Ränder hinweg bewegen und Muster an den Rändern besser erkennen. Das Ergebnis einer Convolution-Schicht ist eine sogenannte Feature Map, also eine Schicht, die aus extrahierten Merkmalen besteht [LB97].

Das Ziel der Pooling-Schicht ist es, die Größe von Feature Maps zu reduzieren und dabei die wichtigsten Merkmale beizubehalten [SMB10]. Wie bei der Convolution-Operation, gibt es auch beim Pooling verschiedene Operationen (z.B. Average-Pooling). In Abbildung 2.12 ist die oft verwendete Max-Pooling-Operation

dargestellt. Dabei werden Fenster mit einer festgelegten Breite und Höhe über das Bild bewegt und jeweils nur der maximale Wert innerhalb eines Fensters übernommen.

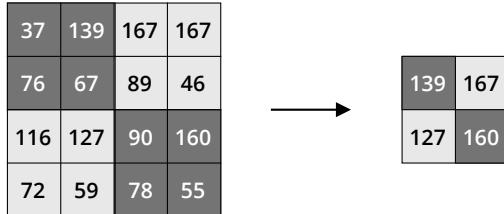


Abbildung 2.12: Beispiel einer Max-Pooling-Operation

Mit Convolution-, Pooling- und Fully-Connected-Schichten kann die Architektur eines CNNs zusammengesetzt werden. In Abbildung 2.13 ist beispielhaft die Architektur eines CNN abgebildet.

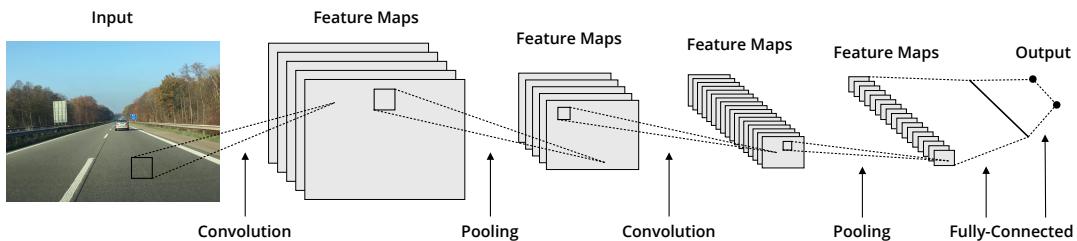


Abbildung 2.13: Beispiel eines Convolutional Neural Networks

2.2.4 Recurrent Neural Networks und LSTMs

In diesem Abschnitt wird die Architektur von Recurrent Neural Networks (RNNs) und einer modifizierten Version davon, Long Short-Term Memorys (LSTMs), vorgestellt. Ein RNN ist ein KNN, das für sequentielle Daten geeignet ist. Der Unterschied zu der Architektur eines mehrlagigen Perzeptrons ist, dass beim Training auch die Werte von Neuronen derselben Schicht berücksichtigt werden [Gra12]. Damit entsteht nicht nur eine Abhängigkeit zu den Aktivierungswerten der vorherigen Schicht, sondern auch zu den Werten der Neuronen derselben Schicht. Das bedeutet wiederum, dass ein RNN nicht von einem einzigen Eingangsvektor abhängig ist, sondern von einer Sequenz von Eingangsvektoren.

Es gibt verschiedene Architekturen von RNNs, zum Beispiel mit Ergebniswerten von jedem Input (engl. many to many) oder nur einem Ergebniswert am Ende einer Sequenz (engl. many to one). Die Architekturen mit Ergebniswerten von jedem Input werden beispielsweise für die automatische Übersetzung von Texten verwendet. Dabei ist jeder Input ein neues Wort, für das jeweils ein Ergebnis, wie zum Beispiel eine Übersetzung, produziert wird, und das auch Einfluss auf die Bedeutung der folgenden Wörter hat. Architekturen mit einem Ergebniswert werden für Klassifizierungsprobleme verwendet. Dabei wird einer Sequenz von Inputdaten eine Klasse zugeordnet. Diese beiden Varianten sind schematisch in Abbildung 2.14 dargestellt. In dieser Arbeit wird in Kapitel 4 die Variante mit einem Ergebnis basierend auf einer Sequenz von Inputdaten verwendet, i.e. die Berechnung einer Klasse auf Basis einer Bildsequenz.

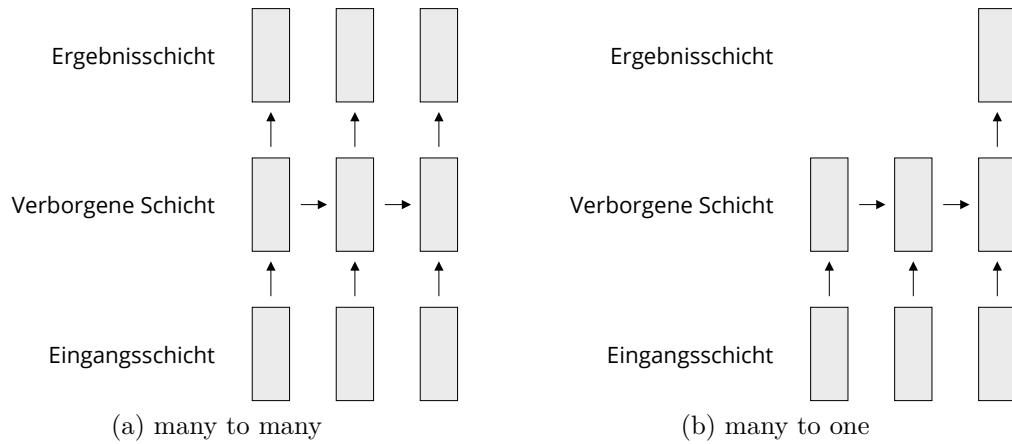


Abbildung 2.14: Beispiel von zwei verschiedenen RNN-Architekturen

Wenn nun in einem RNN die Fehlerrückführung (engl. error backpropagation) angewendet wird, kommt es schnell zu dem Problem des verschwindenden oder explodierenden Gradienten (engl. vanishing or exploding gradient). Das basiert auf der Multiplikation der Gradienten von mehreren Sequenzschritten. Wenn der Gradient größer als 1 ist, wächst er sehr stark (explodiert), wenn er kleiner als 1 ist, geht er schnell gegen 0 (verschwindet). In beiden Fällen kann es zu sehr langen Trainingszeiten, in Extremfällen zu keinem Lerneffekt während des Trainings, kommen [Gra12]. Die Lösung dieses Problems wurde von Hochreiter und

Schmidhuber [HS97] in Form der LSTM-Zelle vorgestellt.

Eine LSTM-Zelle ist mit drei Toren, dem Eingangstor (engl. input gate), dem Merk- und Vergesstor (engl. forget gate) und dem Ausgangstor (engl. output gate), und einem Zellenzustand aufgebaut. Die Idee ist, dass die LSTM-Zelle einen Zustand speichern und damit erhalten kann. Das verhindert den Effekt des verschwindenden oder explodierenden Gradienten. Die Abbildung 2.15 zeigt eine solche LSTM-Zelle.

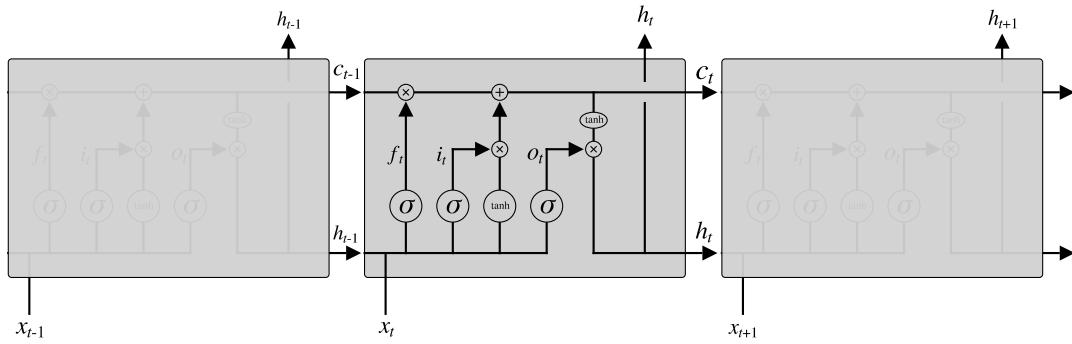


Abbildung 2.15: Long Short-Term Memory-Zelle [Ola15]

Das Vergesstor f_t , das Eingangstor i_t und das Ausgangstor o_t ist jeweils mit einer Sigmoid-Funktion σ modelliert und hat als Ergebnis einen Wert zwischen 0 und 1. Dieser Wert gibt an, wie durchlässig das jeweilige Tor ist. Der Zustand der Zelle c_t zum Zeitpunkt t wird von der Zelle verwendet, um Werte zu speichern oder zu ändern. x_t ist der Dateninput zum Zeitpunkt t . Die Tore f_t , i_t und o_t , der Zustand der Zelle c_t und der Ergebniswert h_t der Zelle werden mit der jeweiligen Gewichtsmatrix W und dem Bias b wie folgt berechnet [Ola15].

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad (2.12)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (2.13)$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (2.14)$$

$$c_t = f_t * c_{t-1} + i_t * \tanh((W_c[h_{t-1}, x_t]) + b_c) \quad (2.15)$$

$$h_t = o_t * \tanh(c_t) \quad (2.16)$$

2.2.5 Training mit synthetischen Daten

Ein großes Problem beim Training von neuronalen Netzen ist der manuelle Aufwand um reale Trainings- und Validierungsdaten zu annotieren [Ric16]. Ein Ansatz dem entgegenzuwirken ist das Training mit synthetischen Daten oder einer Kombination aus synthetischen und realen Daten. In diesem Abschnitt werden vier Arbeiten aus den letzten Jahren vorgestellt, die diesen Ansatz untersuchen. Die Datensätze dieser Arbeiten werden im Abschnitt 4.3 in Tabelle 4.6 zusammengefasst und mit dem Datensatz aus dieser Arbeit verglichen.

Ros et al. [Ros16] haben für die Simulation einer virtuellen Stadt die Unity Development Platform verwendet. Mit virtuellen Fahrten wurden insgesamt circa 200.000 Bilder generiert und diese mit 11 Klassen (e.g. *Sky*, *Buildings*, *Road*) auf Pixelebene annotiert. Für das Training wurden die Bilder auf eine Größe von 180 x 120 Pixel reduziert. Das kombinierte Training mit synthetischen und realen Daten auf den Datensätzen KITTI, CamVid, U-LabelMe und CBCL hat die Genauigkeit (engl. accuracy) auf den realen Testdaten, im Vergleich zum Training mit ausschließlich realen Daten, deutlich gesteigert.

Johnson-Roberson et al. [Joh17] und Richter et al. [Ric16] haben für die Generierung von Trainingsdaten das Computerspiel Grand Theft Auto V verwendet. Die Grafikleistung dieses Spiels übertrifft kommerzielle und Open-Source Simulationssoftware und ist daher sehr gut für die Simulation von Bildern geeignet. Johnson-Roberson et al. [Joh17] generierten zwei Datensätze mit 50.000 und 200.000 Bildern mit Begrenzungsboxen (engl. bounding boxes) für Fahrzeuge. Mit ihrem kombinierten Training zusammen mit dem KITTI Datensatz konnten sie die Genauigkeit auf den Testdaten, im Vergleich zum Training auf ausschließlich realen Daten, verbessern. Richter et al. [Ric16] generierten 25.000 Bilder mit einer Annotation auf Pixelebene von 19 Klassen (e.g. *Road*, *Sky*, *Car*). Mit diesen

Bildern und einem kombinierten Training zusammen mit dem KITTI Datensatz konnten auch sie die Genauigkeit verbessern.

Tremblay et al. [Tre18] verfolgten bei der Generierung von Bildern einen anderen Ansatz. Sie nutzten 3D Modelle von Fahrzeugen und platzierten diese mit zufälliger Position und Ausrichtungen auf verschiedenen realen Szenen. Die 100.000 Bilder wurden mit Begrenzungsboxen (engl. bounding boxes) annotiert. Beim Training wurden ausschließlich die synthetischen Daten verwendet und es wurde eine Genauigkeit von circa 80 Prozent erreicht.

Generell zeigt das Training von neuronalen Netzen mit einer Kombination von synthetischen und realen Daten bereits vielversprechende Ergebnisse, auf denen dieses Arbeit aufbauen will. Im Gegensatz zu den vergangenen Arbeiten, werden in dieser Arbeit ganze Szenarien generiert und annotiert und nicht einzelne Bilder mit Begrenzungsboxen oder semantischer Annotation auf Pixelebene.

2.2.6 Klassifizierung von Videos

Grundsätzlich kann zwischen drei verschiedenen Ansätzen bei der Klassifizierung von Videos mit KNNs unterschieden werden. Beim ersten Ansatz werden die einzelnen Bilder aus einem Video mit einem CNN klassifiziert und anschließend wird das Video der Klasse zugeordnet, zu der die meisten Bilder zugeordnet wurden [Kar14]. Mit diesem Ansatz werden die räumlichen Merkmale (engl. spatial features) mit dem CNN sehr gut extrahiert und bei der Klassifizierung berücksichtigt. Das Problem ist, dass zeitliche Merkmale (engl. temporal features) keine Beachtung finden und die Reihenfolge der einzelnen Bilder ignoriert wird. Dieser Ansatz ist in Abbildung 2.16 dargestellt.

Ein weiterer Ansatz basiert auf der Idee, Feature Maps von mehreren Bildern zu kombinieren und damit die zeitlichen Merkmalen mit 3D-Filters eines CNNs zu extrahieren. In früheren Arbeiten wurde für diese Kombination eine Pooling-Operation verwendet [Kar14; Yue15] und es wurden verschiedene Architekturen mit früher, später oder langsamer Fusion verwendet (engl. early, late, and slow fusion). Bei der frühen Fusion werden schon zu Beginn die Eingangswerte von mehreren Bildern kombiniert. Bei der späten Fusion werden von allen Bildern Merkmale extrahiert und am Ende mit einer Pooling-Schicht vereint. Beim Ansatz

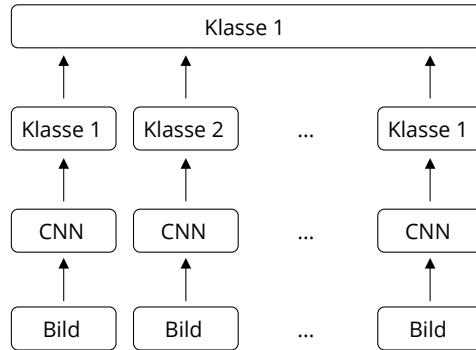


Abbildung 2.16: Klassifizierung von einzelnen Bildern mit anschließender Klassifizierung des gesamten Videos

der langsamen Fusion werden schrittweise die Feature Maps von immer mehr Bildern zusammengefasst. Diese Architekturen sind schematisch in Abbildung 2.17 dargestellt.

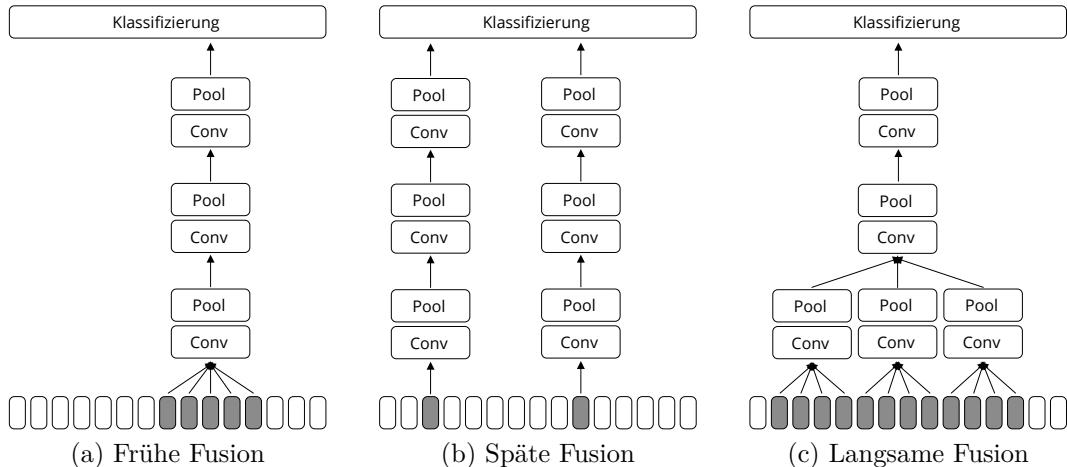


Abbildung 2.17: Schematische Darstellung von Klassifizierungsarchitekturen mit frühen, späten und langsamen Fusionen von Feature Maps [Kar14]

Später wurden neben der Pooling-Operation andere Operationen für die Fusion entwickelt [FPZ16]. Obwohl damit teilweise sehr gute Ergebnisse erzielt werden konnten [CZ17], haben die Fusions-Architekturen einen entscheidenden Nachteil. Zeitliche Merkmale werden berücksichtigt, allerdings nicht die Reihenfolge der Bilder. Aus diesem Grund gibt es einen dritten Ansatz für die Klassifizierung von

Videos, die Kombination aus CNNs, die räumliche Merkmale extrahieren, und LSTMs, die zeitliche Merkmale extrahieren. Eine Architektur mit diesem Ansatz wurde von Donahue et al. [Don15] vorgestellt. Dabei wird ein vortrainiertes CNN verwendet, um die räumlichen Merkmale aus allen Bildern zu extrahieren. Anschließend werden diese Feature Maps einer LSTM-Schicht übergeben, welche die zeitlichen Merkmale extrahiert. In Abbildung 2.18 ist diese Architektur schematisch dargestellt.

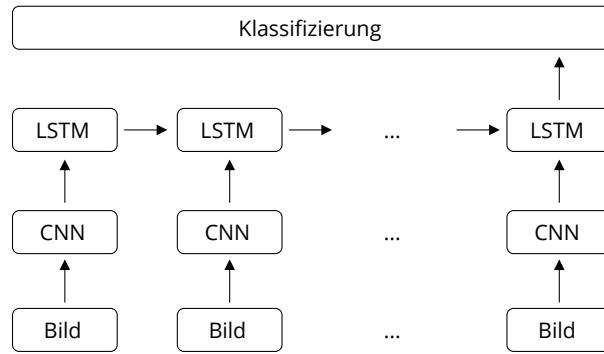


Abbildung 2.18: Klassifizierung von Videos mit einer CNN-LSTM-Architektur [Don15]

In dieser Arbeit wird der erste Ansatz, die Klassifizierung von einzelnen Bildern mit anschließender Klassifizierung des gesamten Videos, und der dritte Ansatz, die kombinierte CNN-LSTM-Architektur verwendet. Die Architekturen dieser Arbeit werden im Detail in Abschnitt 4.4.2 vorgestellt.

Kapitel 3

Konzept

Wie in Abschnitt 2.1 beschrieben, stellt die Absicherung von hochautomatisierten FAS die Automobilindustrie vor große Herausforderungen. Die Menge der bekannten Fahrszenarien ist nur eine Teilmenge aller Szenarien, die zukünftige FAS abdecken müssen. Die Folge ist eine steigende Anzahl benötigter Testkilometer, die in Zukunft mit ökonomischem Aufwand nicht mehr umsetzbar sein wird. Es müssen neue Methoden gefunden werden, relevante Szenarien für die Generierung von Testfällen zu identifizieren, um die Sicherung von hochautomatisierten FAS mit ökonomischen Aufwand garantieren zu können.

Genau hier soll diese Arbeit einen Beitrag liefern. Das Ziel, wie bereits in Abschnitt 1.2 erklärt, ist die Klassifizierung von realen Fahrszenarien. Die Grundidee ist es, einen Klassifikator mit einem großen Anteil synthetischer Daten und einem kleinen Anteil realer Daten von bisher bekannten Szenarien zu trainieren. Es wird mit einem großen Teil synthetischer Daten gearbeitet, weil es in der Praxis einfacher ist, synthetische Daten zu generieren und automatisch zu annotieren als große Mengen realer Daten zu annotieren. Die Idee ist, dass auf diese Weise auch unbekannte Szenarien herausgefiltert werden können, weil diese von dem trainierten Klassifikator nicht erkannt werden. Diese bisher unbekannten Szenarien können dann wiederum als Basis für neue Testfälle für die Sicherung hochautomatisierter Fahrfunktionen verwendet werden.

In dieser Arbeit soll ein Proof-of-Concept für diese Methodik entwickelt werden. Dafür wird im folgenden Abschnitt 3.1 das Konzept im Detail und die Vorgehens-

weise vorgestellt. Anschließend werden in Abschnitt 3.2 die Ansätze, Methoden und Werkzeuge für die Umsetzung aufgelistet.

3.1 Struktur

Das Konzept dieser Arbeit lässt sich in vier Teile untergliedern. Im ersten Teil werden beispielhaft einige Szenarien ausgewählt und auf der Ebene der *logischen Szenarien* definiert. Auf dieser Basis werden im zweiten Teil synthetische und reale Trainingsdaten generiert. Im dritten Teil wird ein KNN als Klassifikator trainiert und evaluiert. In einem vierten Schritt können die Schritte 1 bis 3 mit allen bekannten Szenarien wiederholt werden um bisher unbekannte Szenarien zu finden. Die ersten drei Schritte, die in dieser Arbeit als Proof-of-Concept umgesetzt werden, ist schematisch in Abbildung 3.1 dargestellt. In den folgenden Absätzen werden die Schritte im Detail beschrieben.

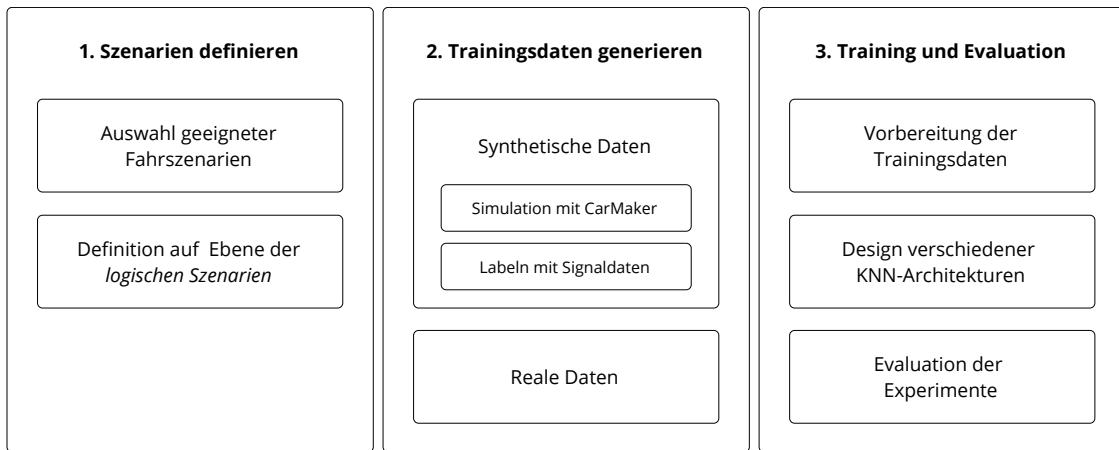


Abbildung 3.1: Konzept dieser Arbeit

Im ersten Schritt werden beispielhaft Fahrszenarien ausgewählt und wie in Abschnitt 2.1.2 definiert. In dieser Arbeit werden Szenarien auf der Ebene der *logischen Szenarien* definiert. Nachdem Szenarien ausgewählt und definiert sind, werden synthetische und reale Daten für das Training eines Klassifikators benötigt.

Für die Generierung von synthetischen Daten wird mit einer Simulationssoftware gearbeitet. Die Idee ist es, Fahrten eines Ego-Fahrzeugs zu simulieren, ent-

sprechende Bild- und Signaldaten aufzuzeichnen und die Bilddaten anhand der Signaldaten zu annotieren. Auf diese Weise können ohne großen Aufwand beliebig viele synthetische Daten erzeugt und annotiert werden. Amersbach und Winner [AW17] stellen einen Ansatz für die funktionale Dekomposition von hochautomatisierten FAS vor. In diesem Ansatz werden Informationen über sechs Schichten, von den Ground Truth Daten über die Szenenerkennung bis zur entsprechenden Aktion des Ego-Fahrzeugs, abgeleitet. Ein Schema dieses Ansatzes ist in Abbildung 3.2 dargestellt. In dieser Arbeit werden für die Annotation der Bilddaten Signaldaten generiert, die nach Schicht 1 (e.g. Geschwindigkeit des Ego-Fahrzeugs) und Schicht 2 (e.g Position des vorausfahrenden Fahrzeugs) eingeordnet werden können. Jeder generierte Zeitpunkt stellt eine Szene, wie in Abschnitt 2.1.2 beschrieben, dar. Jede Szene wird separat auf Basis der entsprechenden Signaldaten nach festgelegten Regeln annotiert. Die Aneinanderreihung von mehreren Szenen ergibt schließlich ein Szenario. Für die Generierung von realen Trainingsdaten werden aufgezeichnete Videos manuell annotiert.

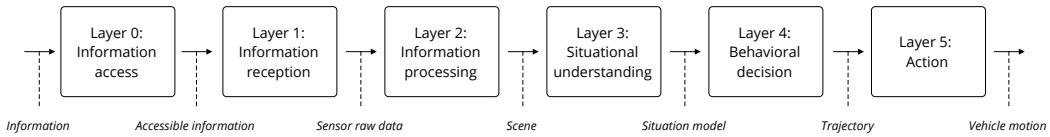


Abbildung 3.2: Schema der funktionalen Dekomposition [AW17]

In dieser Arbeit sollen KNNs als Klassifikatoren verwendet werden. Wie in Abschnitt 2.2.6 beschrieben, gibt es verschiedene Ansätze wie ein Klassifikator für Videos trainiert werden kann. In dieser Arbeit werden zwei verschiedene Ansätze angewendet und miteinander verglichen. Beim ersten Ansatz werden CNNs verwendet, um einzelne Bilder zu klassifizieren. Mit dem zweiten Ansatz wird ein Klassifikator aus einer Kombination von CNNs und LSTMs erstellt. Wie oben beschrieben wird für das Training nur ein kleiner Teil realer und ein großer Teil synthetischer Daten verwendet, um die Skalierbarkeit dieses Ansatzes in der Praxis zu gewährleisten.

Der Proof-of-Concept in dieser Arbeit ist erfolgreich, wenn ein Klassifikator mit diesen drei Schritte trainiert werden und im Anschluss Szenarien auf Basis von Bilddaten klassifizieren kann. Danach kann in einem vierten Schritt ein wei-

terer Klassifikator mit allen bisher bekannten Szenarien nach demselben Prinzip trainiert werden. Dieser Klassifikator kann im Anschluss alle bisher bekannten Szenarien klassifizieren und bisher unbekannte Szenarien herausfiltern, wenn beispielsweise eine Schwelle für die Klassifizierungsgenauigkeit unterschritten wird. Damit wird es möglich die Menge der bisher bekannten Szenarien zu erweitern und die Absicherung von hochautomatisierten FAS weiter voranzutreiben.

3.2 Ansätze, Methoden, Werkzeuge

In diesem Abschnitt wird ein Überblick gegeben, welche Ansätze, Methoden und Werkzeuge in den jeweiligen Teilen der Umsetzung verwendet werden. Diese Zuordnung ist in der folgenden Tabelle 3.1 dargestellt. Die detaillierte Beschreibung folgt in Kapitel 4.

Teil der Umsetzung	Ansätze, Methoden, Werkzeuge	Quellen
Auswahl und Definition geeigneter Fahrszenarien	Konzept der <i>logischen Szenarien</i>	[Ulb15], [Bag17]
Simulation und Annotation synthetischer Trainingsdaten	Generierung von Bild- und Signaldaten mit CarMaker, regelbasierte Klassifizierung auf Basis von vorher festgelegten Signaldaten	[Gmb18]
Generierung realer Trainingsdaten	Auswahl geeigneter Videosequenzen von YouTube, manuelle Annotation	[Nut18], [Goo18a], [Goo18b]
Vorbereitung der Daten, Erstellung des Klassifikators und Evaluation der Experimente	Verschiedene Architekturen mit CNNs und LSTMs, implementiert mit Python und Keras	[Cho15], [LKF10], [HS97]

Tabelle 3.1: Ansätze, Methoden und Werkzeuge dieser Arbeit

Kapitel 4

Implementierung

In diesem Kapitel wird die Umsetzung des Konzepts aus dem vorherigen Kapitel beschrieben. Zu Beginn werden die dafür notwendigen Fahrszenarien in Abschnitt 4.1 definiert. Danach wird in den Abschnitten 4.2 und 4.3 die Methodik für die Generierung von synthetischen und realen Daten erläutert. Im Anschluss wird in Abschnitt 4.4 die Architektur und die Experimente der Klassifikatoren für die Szenarienerkennung beschrieben.

4.1 Definition der Fahrszenarien

In diesem Abschnitt werden Szenarien, wie in Abschnitt 2.1.2 beschrieben, als *logische Szenarien* für das weitere Vorgehen in dieser Arbeit definiert. In Anlehnung an bestehende Arbeiten zur Erkennung von Fahrszenarien und auf Basis von Machbarkeitsabschätzungen für die Umsetzung werden in dieser Arbeit die Szenarien *free cruising, approaching, following, catching up, overtaking, lane change left* und *lane change right* auf der Autobahn betrachtet. Die Autobahn wurde ausgewählt, weil es weniger Details zu betrachten gibt als auf anderen Straßen wie beispielsweise in der Stadt. In der folgenden Tabelle 4.1 werden diese Szenarien auf *funktionaler* und *logischer Ebene* definiert.

Um die Darstellung in der Tabelle zu erleichtern, werden folgende Abstände zwischen Ego-Fahrzeug und Fahrzeug 2 definiert. Dabei beschreibt ego_v die Geschwindigkeit des Ego-Fahrzeugs in [m/s].

$$\begin{aligned}
 s_0 &= ego_v * 3,6 & [m] \\
 s_1 &= ego_v * 3,6 * \frac{2}{3} & [m] \\
 s_2 &= ego_v * 3,6 * \frac{1}{2} & [m] \\
 s_3 &= ego_v * 3,6 * \frac{1}{3} & [m]
 \end{aligned}$$

Szenario	Funktionale Definition	Logische Definition	
Alle	2-spurige Autobahn geradeaus oder in einer Kurve, Geschwindigkeitsbegrenzung ist größer als 80 km/h	Breite Fahrstreifen [2,3..3,5] m Geschwindigkeitsbegrenzung [80..keine] km/h	
Alle	Tageslicht, keine Wolken bis leicht bewölkt, kein Niederschlag, gute Sichtbedingungen	Tageszeit Bewölkung [leicht bewölkt..wolkenlos]	[Sonnenaufgang..Sonnenuntergang]
Free cruising	Ego, keine anderen Verkehrsteilnehmer <u>Interaktion:</u> Ego fährt frei auf linker oder rechter Fahrspur, andere Fahrzeuge sind weit entfernt und haben keinen Einfluss auf die Manöver des Ego	Geschwindigkeit Ego [60..200] km/h Abstand zu anderen Verkehrsteilnehmern [$> s_0$] m	
Approaching	Ego, andere Verkehrsteilnehmer <u>Interaktion:</u> Ego nähert sich auf linker oder rechter Fahrspur in mittlerem Abstand dem Fahrzeug 2	Geschwindigkeit Ego abnehmend [60..200] km/h Geschwindigkeit Ego < Geschwindigkeit Fahrzeug 2 Abstand Ego zu Fahrzeug 2 [$s_2..s_0$] m	

Szenario	Funktionale Definition	Logische Definition
Following	Ego, andere Verkehrsteilnehmer <u>Interaktion:</u> Ego fährt auf linker oder rechter Fahrspur in sicherem Abstand hinter Fahrzeug 2	Geschwindigkeit Ego [60..200] km/h Geschwindigkeitsdifferenz zwischen Ego und Fahrzeug 2 $< \text{Geschwindigkeit Ego} * 0,05 \text{ km/h}$ Abstand Ego zu Fahrzeug 2 $[s_3..s_1] \text{ m}$ Ego befindet sich auf gleicher Fahrspur hinter Fahrzeug 2
Catching up	Ego, andere Verkehrsteilnehmer <u>Interaktion:</u> Ego fährt auf der linken Fahrspur und verringert den vertikalen Abstand zu Fahrzeug 2, das sich vor dem Ego-Fahrzeug auf der rechten Fahrspur befindet	Geschwindigkeit Ego [60..200] km/h Geschwindigkeit Fahrzeug 2 $< \text{Geschwindigkeit Ego}$ Vertikaler Abstand Ego zu Fahrzeug 2 $[0..s_0] \text{ m}$ Ego fährt auf linker Fahrspur hinter Fahrzeug 2 das auf rechter Fahrspur fährt
Overtaking	Ego, andere Verkehrsteilnehmer <u>Interaktion:</u> Ego fährt auf der linken Fahrspur und vergrößert den vertikalen Abstand zu Fahrzeug 2, das sich hinter dem Ego-Fahrzeug auf der rechten Fahrspur befindet	Geschwindigkeit Ego [60..200] km/h Geschwindigkeit Fahrzeug 2 $< \text{Geschwindigkeit Ego}$ Vertikaler Abstand Ego zu Fahrzeug 2 $[0..s_0] \text{ m}$ Ego fährt auf linker Fahrspur vor Fahrzeug 2 das auf rechter Fahrspur fährt

Szenario	Funktionale Definition	Logische Definition
Lane change left	Ego, andere Verkehrsteilnehmer sind optional <u>Interaktion:</u> Ego fährt auf rechter Fahrspur und wechselt auf linke Fahrspur	Geschwindigkeit Ego [60..200] km/h Ego befindet sich auf rechter Fahrspur und wechselt auf linke Fahrspur
Lane change right	Ego, andere Verkehrsteilnehmer sind optional <u>Interaktion:</u> Ego fährt auf linker Fahrspur und wechselt auf rechte Fahrspur	Geschwindigkeit Ego [60..200] km/h Ego befindet sich auf linker Fahrspur und wechselt auf rechte Fahrspur

Tabelle 4.1: Definitionen der Szenarien *free cruising, approaching, following, catching up, overtaking, lane change left* und *lane change right*

4.2 Generierung synthetischer Daten

Auf Basis der Definitionen aus dem vorherigen Abschnitt 4.1 werden in diesem Abschnitt die benötigten Signal- und Bilddaten simuliert und entsprechend annotiert. Dafür werden in Abschnitt 4.2.1 die Signaldaten, die für die eindeutige Klassifizierung der Szenarien benötigt werden, simuliert. In Abschnitt 4.2.2 werden diese Signaldaten verwendet, um die parallel simulierten Bilddaten entsprechend zu annotieren.

4.2.1 Simulation

Für die Simulation der Signal- und Bilddaten wird die kommerzielle Software CarMaker von IPG Automotive [Gmb18] verwendet. Diese Simulationssoftware wird für den virtuellen Fahrversuch und HiL-Tests eingesetzt, um Komponenten in unterschiedlichen Szenarien zu testen. In dieser Arbeit wird CarMaker verwendet, um die Szenarien *free cruising, approaching, following, catching up, overtaking, lane change left* und *lane change right* zu simulieren.

Für die Aufnahme der benötigten Bilddaten wird im simulierten Fahrzeug ein

entsprechender Kamerasensor konfiguriert. Die Konfiguration des Sensors orientiert sich an der Konfiguration von realen Frontview-Kameras im Fahrzeug nach Punkte [Pun15]. So ist der Kamerasensor an der Stelle des Rückfahrspiegels platziert, hat eine Auflösung von 640 x 480 Pixel und ein Sichtfeld von 20°.

Variable in CarMaker	Beschreibung
Car.v	Geschwindigkeit des Ego-Fahrzeugs in [m/s]
Car.Road.sRoad	Position des Ego-Fahrzeugs auf der Strecke in [m]
Car.Road.Lane.ActLaneId	Fahrspur-ID des Ego-Fahrzeugs
Sensor.Object.OB01.TX.NearPnt.dv_p	Geschwindigkeitsdifferenz zwischen Fahrzeug TX und dem Ego-Fahrzeug in [m/s]
Sensor.Object.OB01.TX.NearPnt.ds_p	Abstand zwischen Fahrzeug TX und dem Ego-Fahrzeug in [m]
Traffic.TX.sRoad	Position des Fahrzeugs TX auf der Strecke in [m]
Traffic.TX.Lane.ActLaneId	Fahrspur-ID des Fahrzeugs TX

Tabelle 4.2: Aufgezeichnete Signaldaten in CarMaker

Die benötigten Signaldaten für das Annotieren werden von den Definitionen aus Abschnitt 4.1 abgeleitet. Für die eindeutige Identifikation der *logischen Szenarien* werden die folgenden Werte benötigt: Geschwindigkeit des Ego-Fahrzeugs, Abstand und Geschwindigkeitsdifferenz des Ego-Fahrzeugs zu allen anderen Fahrzeugen, aktuelle Fahrspur des Ego-Fahrzeugs und allen anderen Fahrzeugen und die relative Position des Ego-Fahrzeugs, i.e. ob sich das Ego-Fahrzeug vor oder hinter einem anderen Fahrzeug befindet. Um den Abstand und die Geschwindigkeitsdifferenz des Ego-Fahrzeugs zu allen anderen Fahrzeugen aufzuzeichnen, wird ein Objektsensor im Ego-Fahrzeug konfiguriert. Mit diesem Sensor können im festgelegten Radius alle Fahrzeuge und ihr Abstand und ihre relative Geschwindigkeit zum Ego-Fahrzeug erfasst und über die Funktion *OutputQuantities* in CarMaker aufgezeichnet werden. Die Geschwindigkeit des Ego-Fahrzeugs und die Fahrspur-

ID und Position aller Fahrzeuge können direkt, ohne zusätzlichen Sensor, über die Funktion *OutputQuantities* aufgezeichnet werden. Die jeweiligen Variablen in CarMaker sind in der Tabelle 4.2 zusammengefasst.

Für die Simulation werden zwei Strecken der Länge 6.000 m und 10.000 m mit dem *CarMaker - Scenario Editor* erstellt. Bei beiden Strecken handelt es sich um eine 4-spurige Autobahn mit zwei Fahrspuren in jede Richtung. Die Fahrtrichtungen sind in der Mitte von einer Leitplanke getrennt und am Rand der Fahrbahn sind jeweils Standstreifen vorhanden. Abschnittsweise stehen neben der Fahrbahn einige Bäume, was in Abbildung 4.2 mit grünen Streifen gekennzeichnet ist. Abbildung 4.1 zeigt die Konfiguration der simulierten Straße.



Abbildung 4.1: Konfiguration der simulierten Straße [Gmb18]

Auf beiden Strecken wird autonomer, stochastisch verteilter Verkehr erzeugt, was CarMaker mit einer gesonderten Funktion unterstützt. Der Verkehr wird in einer niedrigen Dichte (10 Prozent) und einem 80-prozentigen Anteil Autos erzeugt. Andere Fahrzeuge sind Motorräder, Lastkraftwagen und Busse. In CarMaker ist eine Vielzahl an unterschiedlichen Fahrzeugen verfügbar, was wichtig ist, um möglichst viele unterschiedliche Szenarien zu generieren. Mit dieser Konfiguration werden auf der 10.000m-Strecke 131 Fahrzeuge und auf der 6.000m-Strecke 89 Fahrzeuge generiert.

Die Simulation und Generierung von Bild- und Signaldaten wird mit dem *CarMaker - Test Manager* durchgeführt. Mit diesem Modul lassen sich Fahrten mit unterschiedlichen Konfigurationen simulieren. In dieser Arbeit werden die Variablen *Geschwindigkeit*, *Mindestabstand zu vorausfahrendem Fahrzeug*, *Minimale*

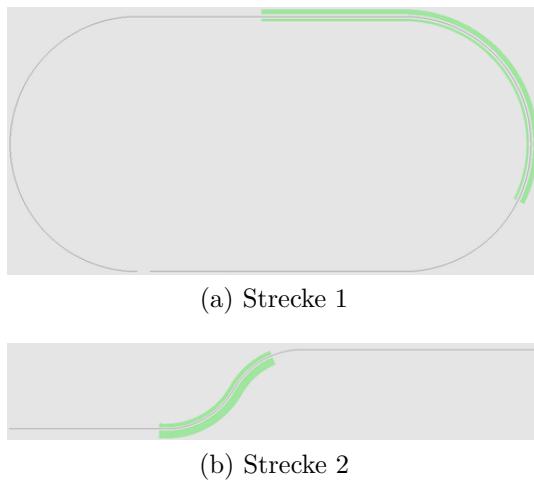


Abbildung 4.2: Schema der simulierten Strecken 1 und 2 [Gmb18]

Geschwindigkeitsdifferenz beim Überholen und *Aggressivität beim Überholen* auf beiden oben beschriebenen Strecken variiert. Die Werte der Variablen, die simuliert werden, sind in Tabelle 4.3 aufgelistet und sind aus der Sicht des Ego-Fahrzeugs zu verstehen.

Variablen	Werte
Geschwindigkeit in [km/h]	100 120 140 160 180
Mindestabstand zu vorausfahrendem Fahrzeug in [s]	1,0 1,5 2,0
Minimale Geschwindigkeitsdifferenz beim Überholen in [km/h]	5 15 25
Aggressivität beim Überholen	0,2 0,6 1,0

Tabelle 4.3: Variablen und Werte die in der Simulation verwendet werden

Die ersten drei Variablen sind selbsterklärend und werden hier nicht weiter erläutert. Die Variable *Aggressivität beim Überholen* (in CarMaker *Overtaking Rate*) ist eine Zahl zwischen 0 und 1. Dabei beschreibt 0 einen Fahrstil, bei dem sich der Fahrer sehr risikoavers beim Überholen verhält, i.e. Überholen nur in sehr sicheren Situationen. Je größer die Zahl wird, desto aggressiver wird der Überholvorgang und dementsprechend sinkt die Risikoaversion beim Überholen und der Fahrer

überholt auch bei kritischen oder schlecht einsehbaren Situationen.

Mit diesen vier Variablen mit jeweils drei beziehungsweise fünf Werten ergeben sich 135 verschiedene Kombinationsmöglichkeiten. Somit werden auf beiden Strecken in Summe 270 Fahrten mit insgesamt 2.160 km simuliert. Signal- und Bilddaten werden mit einer Frequenz von 5 Hz aufgezeichnet, was in 326.108 aufgezeichneten Szenen resultiert. Diese Szenen werden im folgenden Abschnitt 4.2.2 annotiert.

4.2.2 Daten Annotation

Für das Annotieren der Szenarien wird jede Szene auf Basis der Definition aus Abschnitt 4.1 mithilfe der Signaldaten klassifiziert. Die logischen Bedingungen für jedes Szenario sind dafür in Tabelle 4.4 zusammengefasst. Auf Basis der CarMaker-Variablen aus Tabelle 4.3 werden folgende Variablen definiert, um die nachfolgenden Bedingungen übersichtlicher darstellen zu können. Dabei beschreibt $v2$ jeweils das Fahrzeug, auf Basis dessen das jeweilige Szenario klassifiziert wird.

$$\begin{aligned}
 ego_v &= \text{Car.v} & [\text{m/s}] \\
 ego_{sRoad} &= \text{Car.Road.sRoad} & [\text{m}] \\
 ego_{laneID} &= \text{Car.Road.Lane.Act.LanId} & [1, 2] \\
 v2_{dv} &= \text{Sensor.Object.OB01.TX.NearPnt.dv_p} & [\text{m/s}] \\
 v2_{ds} &= \text{Sensor.Object.OB01.TX.NearPnt.dv_s} & [\text{m}] \\
 v2_{sRoad} &= \text{Traffic.TX.sRoad} & [\text{m}] \\
 v2_{laneID} &= \text{Traffic.TX.Lane.Act.LaneId} & [1, 2]
 \end{aligned}$$

Szenario	Bedingungen
Free cruising	$ego_v > 17$ $s_0 < v2_{ds}$

Szenario	Bedingungen
Approaching	$s_2 < v2_{ds} < s_0$ $ego_{sRoad} < v2_{sRoad}$ $ego_{laneID} = v2_{laneID}$ $ego_v <$ Durchschnitt von ego_v der letzten 3 Sekunden
Following	$v2_{dv} < ego_v * 0,05$ $s_3 < s_1$ $ego_{sRoad} < v2_{sRoad}$ $ego_{laneID} = v2_{laneID}$
Catching up	$v2_{dv} < 0$ $0 \leq v2_{ds} < s_0$ $ego_{sRoad} \leq v2_{sRoad}$ $ego_{laneID} = v2_{laneID} - 1$
Overtaking	$0 \leq v2_{ds} < s_0$ $v2_{sRoad} < ego_{sRoad}$ $ego_{laneID} = v2_{laneID} - 1$
Lane change left	$ego_{laneID}^{before} = ego_{laneID}^{after} + 1$ <p>Als Spurwechsel wird ein Intervall von 4 Sekunden betrachtet, in dessen Mitte die Variable ihren Wert wechseln muss</p>
Lane change right	$ego_{laneID}^{before} = ego_{laneID}^{after} - 1$ <p>Als Spurwechsel wird ein Intervall von 4 Sekunden betrachtet, in dessen Mitte die Variable ihren Wert wechseln muss</p>

Tabelle 4.4: Bedingungen für die Szenarien *free cruising*, *approaching*, *following*, *catching up*, *overtaking*, *lane change left* und *lane change right*

Im Anschluss an die Klassifizierung einzelner Zeitpunkte werden diese zu Szenarien zusammengefasst, wenn mindestens 15 Zeitpunkte (3 Sekunden) in Folge mit der gleichen Klasse annotiert wurden. Drei Sekunden wird aus den folgenden zwei Gründen als Länge für Szenarien in dieser Arbeit gewählt. Erstens orientiert sich diese Zeitspanne an den vorherigen Arbeiten zur Szenarienerkennung.

Zweitens haben die zwei Szenarien *lane change left* und *lane change right* jeweils eine natürliche Länge von 3-4 Sekunden. Alle anderen Szenarien können länger sein, lassen sich aber in Blöcke von jeweils 3 Sekunden einteilen. Insgesamt werden 326.108 Zeitpunkte und 23.972 Szenarien klassifiziert. Die Anzahl der simulierten Szenarien nach Klasse ist in der Tabelle 4.5 dargestellt.

Szenario	Anzahl
Free cruising	2.545
Approaching	3.512
Following	3.601
Catching up	5.563
Overtaking	5.149
Lane change left	957
Lane change right	975
Unknown	1.670
Summe	23.972

Tabelle 4.5: Anzahl der simulierten Szenarien nach Klasse

Es sind auch einige Szenarien als *unknown* klassifiziert, da zwischen den definierten Szenarien andere Situationen auftreten können oder nicht die Mindestanzahl der konsekutiven Szenen erreicht wird. In manchen Zeitpunkten wird mehr als eine einzelne Szene annotiert. Beispielsweise kann sich das Ego-Fahrzeug gleichzeitig in der Szene *catching up* und *overtaking* befinden, wenn es auf der linken Fahrspur fährt und sich in vertikalem Abstand ein anderes Fahrzeug jeweils vor und hinter dem Ego-Fahrzeug befindet. Abbildung 4.3 zeigt ein Beispiel des Szenarios *lane change left* mit allen zugehörigen 15 Bildern.

4.3 Generierung realer Daten

Für den Proof-of-Concept dieser Arbeit, die Erkennung von realen Fahrszenarien, werden neben den synthetischen Trainingsdaten auch reale Daten für das Training

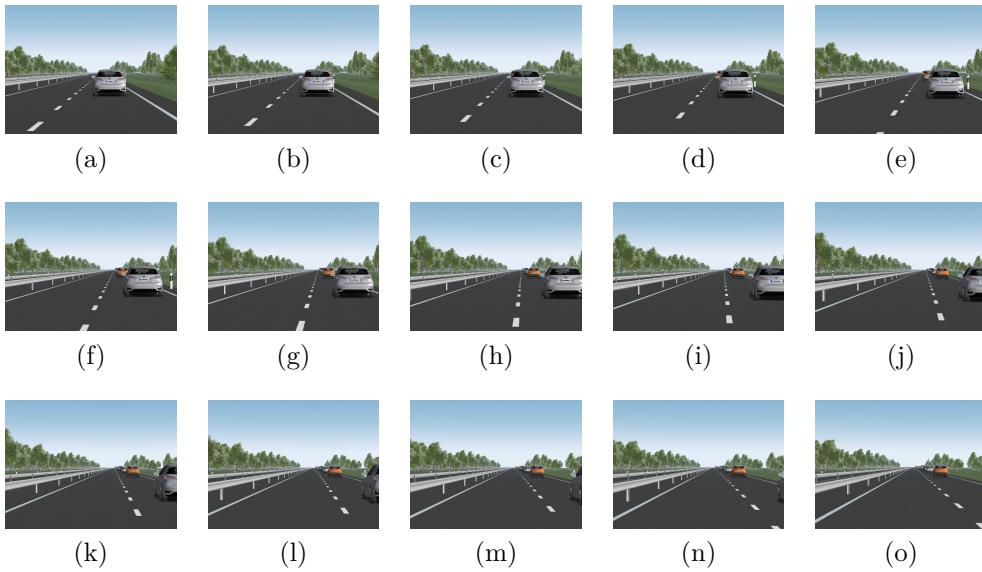


Abbildung 4.3: Beispiel eines simulierten Szenarios der Klasse *lane change left* [Gmb18]

und die anschließenden Tests benötigt. Dafür werden im ersten Schritt bestehende Datensätze nach ihrer Nutzbarkeit untersucht.

Die bekanntesten Datensätze sind KITTI [Gei13], BDD100K [Yu18], Cityscapes [Cor16] und Oxford RobotCar [Mad17]. Der Cityscapes und Oxford RobotCar Datensatz umfasst lediglich Bilder von Szenen in Städten und ist daher nicht nutzbar für diese Arbeit. Die Datensätze KITTI und BDD100K umfassen auch Videos von Autobahnfahrten, allerdings liegt der Fokus auf Objekterkennung in einzelnen Bildern oder semantischer Segmentation. Und es gibt jeweils nur sehr begrenzt Videos, die auf einer 2-spurigen Autobahn aufgenommen wurden. Daher können diese Datensätze in dieser Arbeit nicht verwendet werden. Als Alternative werden die Aufnahmen von zwei Fahrten auf der Autobahn und der Bundesstraße verwendet. Die Strecken sind in der Abbildung 4.4 abgebildet.

Die Aufnahme der Route 1 umfasst über sechs Stunden Videomaterial mit über einer Stunde Fahrt auf einer 2-spurigen Autobahn [Nut18]. Davon werden manuell zwischen 50 und 180 Szenarien aus jeder Klasse annotiert. Da von den Szenarien *lane change left* und *lane change right* jeweils nur 50 Szenarien vorhanden sind, wird vom Autor eine Fahrt auf Route 2 mit einigen Spurwechseln aufgenommen.

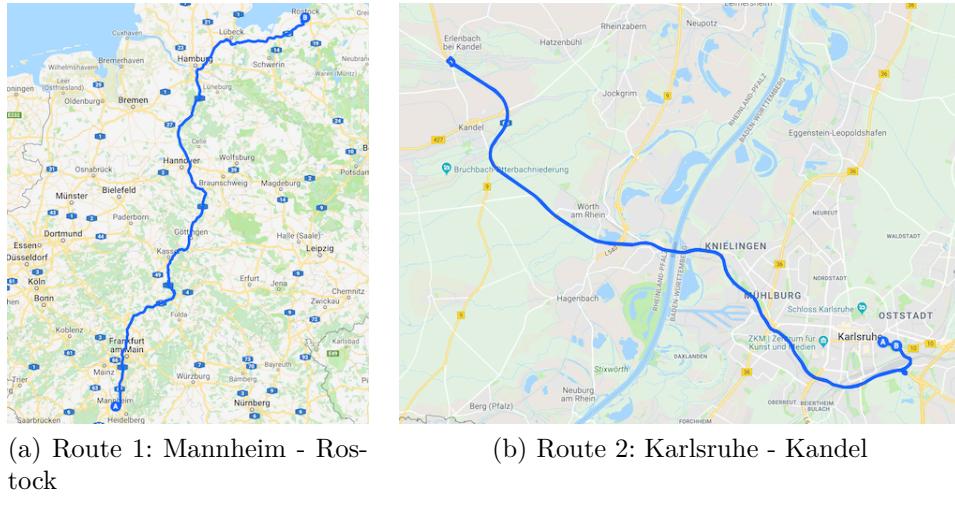


Abbildung 4.4: Routen für die Aufnahme der realen Bilddaten [Goo18a; Goo18b]

Das Ergebnis sind jeweils 17 weitere Szenarien in den Klassen *lane change left* und *lane change right*. Abbildung 4.5 zeigt ein Beispiel des realen Szenarios *lane change right* mit allen zugehörigen 15 Bildern.

In Tabelle 4.6 wird der gesamte Datensatz dieser Arbeit mit synthetischen und realen Daten, mit den synthetischen Datensätzen, die in Abschnitt 2.2.5 vorgestellt wurden, und realen Datensätzen, die in diesem Abschnitt vorgestellt werden, verglichen.

Datensatz	Generierung	Umfang	Annotation
[Ros16]	synthetisch mit der Unity Development Platform	200.000 Bilder	semantische Annotation auf Pixelebene von 11 Klassen
[Joh17]	synthetisch mit dem Computerspiel GTA5	250.000 Bilder	Begrenzungsboxen für die Klasse Fahrzeuge
[Ric16]	synthetisch mit dem Computerspiel GTA5	25.000 Bilder	semantische Annotation auf Pixelebene von 19 Klassen

Datensatz	Generierung	Umfang	Annotation
[Tre18]	synthetisch mit 3D Modelle von Fahrzeugen und realen Szenen	100.000 Bilder	Begrenzungsboxen für die Klasse Fahrzeuge
KITTI, [Gei13]	real mit Fahrten in der Region Karlsruhe	22 Videos und 15.000 Bilder	Begrenzungsboxen mit 8 Klassen
BDD100K, [Yu18]	real mit Fahrten in der Städten New York, Berkeley, San Francisco und der Bay Area	100.000 Videos (40 Sekunden) mit insgesamt 120 Mio. Bilder	für jeweils das zehnte Bild aus jedem Video: Fahrbahnbegrenzungslinien und Begrenzungsboxen für 10 Objekte; Für 10.000 Bilder semantische Annotation auf Pixelebene von 19 Klassen
Cityscapes, [Cor16]	real mit Fahrten in 50 Städten in Deutschland	25.000 Bilder	semantische Annotation auf Pixelebene von 30 Klassen
Oxford RobotCar, [Mad17]	real mit Fahrten in Oxford	knapp 20 Mio. Bilder	keine Annotation
Diese Arbeit	synthetisch mit CarMaker und real mit zwei Autobahnfahrten in Deutschland	24.307 Szenarien (3 Sekunden) mit insgesamt 331.133 Bilder	7 Klassen

Tabelle 4.6: Vergleich von synthetischen und realen Datensätzen

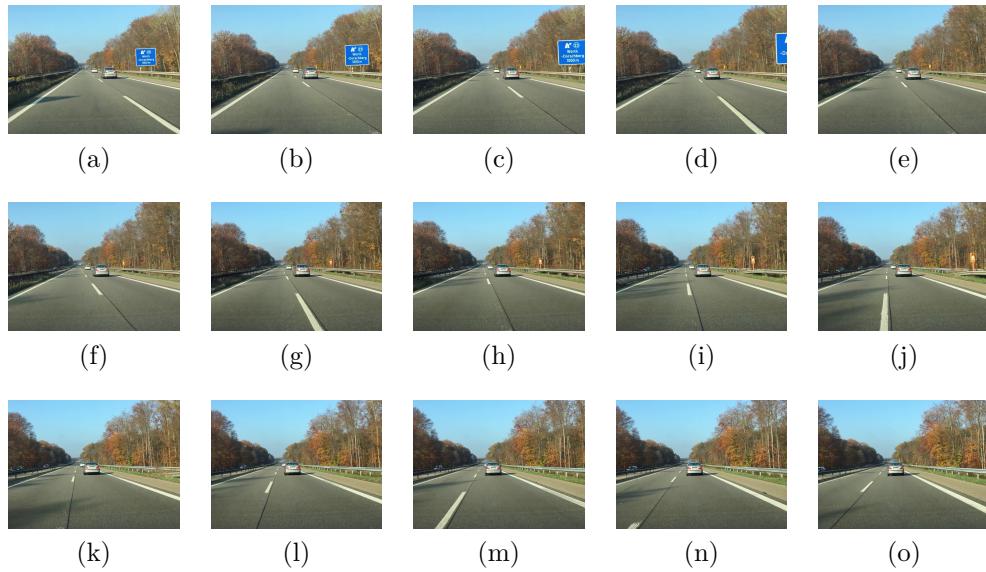


Abbildung 4.5: Beispiel eines realen Szenarios der Klasse *lane change right*

4.4 Training

In diesem Abschnitt wird zu Beginn das Format und der Import der Trainings- und Testdaten in das KNN erklärt. Danach werden verschiedene Architekturen von KNNs, die in dieser Arbeit zum Einsatz kommen, vorgestellt und schließlich werden die durchgeführten Experimente mit diesen Architekturen erläutert. Die Vorbereitung der Daten und das Training wird mit Python und der Deep-Learning-Bibliothek Keras [Cho15] implementiert.

4.4.1 Vorbereitung der Daten

Die Generierung der synthetischen und realen Trainings-, Validierungs- und Testdaten wurde bereits in den vorherigen Abschnitten beschrieben. In diesem Abschnitt wird darauf eingegangen, wie diese Daten vorbereitet werden.

In dieser Arbeit werden CNNs für die Erkennung einzelner Bilder und Kombinationen aus CNNs und LSTMs für die Klassifizierung von Videos eingesetzt. Daher müssen sowohl einzelne Bilder, als auch Bildsequenzen in das jeweilige KNN importiert werden. Da es sich um Datenmengen von insgesamt über 50 GB

handelt, müssen die Daten mit einem Datenstrom (engl. data stream) eingespeist werden, da nicht alle Daten zu Beginn in den Arbeitsspeicher geladen werden können.

Die Deep-Learning-Bibliothek Keras besitzt bereits verschiedene Klassen, sogenannte *DataGenerators*, für der Import von einzelnen Bildern und für sequenzielle Daten mit zwei Dimensionen (e.g. CSV-Dateien). Es gibt allerdings keine bereits nutzbare Lösung für den Import von sequentiellen Bildern, was 4-dimensionalen Daten entspricht (Anzahl Bilder, Höhe, Breite, Farbkanal). Aus diesem Grund wird die Lösung von Amidi [Ami17] adaptiert um höher-dimensionale Datensätze importieren zu können.

Von den generierten synthetischen und realen Szenarien werden *free cruising*, *following*, *catching up*, *lane change left* und *lane change right* für das Training ausgewählt. Das Szenario *approaching* wird für den Proof-of-Concept in dieser Arbeit bewusst entfernt, weil es Ähnlichkeiten zu dem Szenario *following* gibt. Diese Überschneidungen und Ähnlichkeiten zwischen Szenarien und ihren Einfluss auf das Training eines Klassifikators können in weiteren Arbeiten untersucht werden. Das Szenario *overtaking* wird entfernt, weil es mit der konfigurierten Frontkamera nicht erfasst werden kann. In folgenden Arbeiten kann dieses Szenario mithilfe weiterer Kameraperspektiven oder anderen Signaldaten berücksichtigt werden.

Um das Ergebnis nicht zu verzerren, sollten für das Training mit KNNs in jeder Klasse jeweils die gleiche Anzahl an Trainings-, Validierungs- und Testdaten vorhanden sein. Aus diesem Grund werden für das Training aus jeder Klasse nur die Anzahl der Szenarien verwendet, die mindestens in jeder Klasse verfügbar sind. Damit dieser Ansatz auch in der Praxis verwendet werden kann wird außerdem darauf geachtet, dass 95 Prozent synthetische Daten und 5 Prozent reale Daten für das Training verwendet werden. Mit dieser Verteilung bleibt der manuelle Aufwand für die Annotation der Trainingsdaten überschaubar. Die daraus resultierende Aufteilung von synthetischen und realen Szenarien auf Trainings-, Validierungs- und Testdaten ist in Tabelle 4.7 aufgeschlüsselt. Beim Training mit einzelnen Bildern wird die gleiche Verteilung angewendet und da jedes Szenario aus 15 Bildern besteht, können die Zahlen aus Tabelle 4.7 dafür mit dem Faktor 15 multipliziert werden.

	Synthetische Daten			Reale Daten		
	Training	Validierung	Test	Training	Validierung	Test
Szenario	85%	10%	5%	65%	10%	25%
free cruising	665	190	95	33	17	17
following	665	190	95	33	17	17
catching up	665	190	95	33	17	17
lane change left	665	190	95	33	17	17
lane change right	665	190	95	33	17	17
Summe	3.325	950	475	165	85	85

Tabelle 4.7: Aufteilung von synthetischen und realen Szenarios auf Trainings-, Validierungs- und Testdaten

Die Bilddaten werden für das Training in die Form 299 x 299 x 3 Pixel transformiert. Diese Bildgröße ist ein Kompromiss zwischen Detailgrad und Trainingszeit und wird in Keras für die Inception-V3 und die Xception Architektur als Standardgröße vorgeschlagen [Cho15].

4.4.2 Design der KNN-Architekturen

Wie in Abschnitt 2.2.6 erläutert, gibt es verschiedenen Ansätze, um Bildsequenzen mit KNNs zu klassifizieren. In dieser Arbeit werden zwei unterschiedliche Ansätze umgesetzt und miteinander verglichen. Im ersten Ansatz werden die Bilder aus den Bildsequenzen einzeln klassifiziert. Anschließend wird den Sequenzen die Klasse zugeordnet, mit der die meisten Bilder aus der jeweiligen Sequenz klassifiziert wurden (relative Mehrheit). Für den zweiten Ansatz wird eine Kombination aus Bild- und Sequenzerkennung verwendet, um die Bildsequenzen als Ganzes zu klassifizieren. Die Architekturen für den jeweiligen Ansatz sind schematisch in Abbildung 4.6 dargestellt und werden im Detail in den folgenden Absätzen erklärt. Dazu werden zu Beginn die Architekturen und Konfigurationen der verwendeten CNNs vorgestellt. Anschließend werden die verwendeten Fully-Connected- und LSTM-Schichten mit den verwendeten Hyperparametern vorgestellt. Am Ende

dieses Abschnitts werden alle Komponenten zusammengefügt und acht Architekturen vorgestellt.

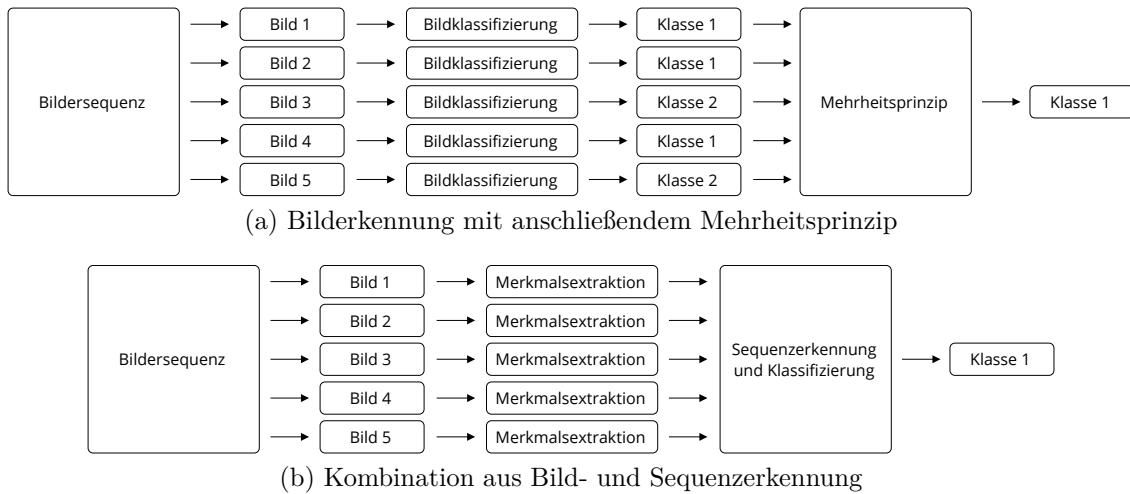


Abbildung 4.6: Zwei Ansätze für die Videoklassifizierung

Auswahl von Convolutional Neural Networks

Das Training mit CNNs hat in den vergangenen Jahren durch die Verfügbarkeit von großen Datenmengen, steigender Rechenleistung und die Wiederverwendung von bereits vortrainierten CNNs große Fortschritte gemacht. Die Verwendung von bereits trainierten Netzen wird Transferlernen (engl. transfer learning) genannt [Oqu14]. Die Idee von einem CNN ist es, in den frühen Schichten sehr grundlegende Merkmale (engl. low-level features) und in späteren Schichten abstraktere Merkmale (engl. high-level features) zu extrahieren. Schließlich wird Bilder auf Basis dieser abstrakten Merkmale mithilfe von einer oder mehreren Fully-Connected-Schichten klassifiziert.

Diese Funktionsweise macht man sich beim Transferlernen zunutze und verwendet ein CNN das bereits auf vielen Millionen Bildern trainiert wurde. Dieses CNN hat dann bereits gelernt unterschiedliche Merkmale aus Bildern zu extrahieren. Um ein solches neuronales Netz für die eigene Arbeit verwenden zu können, muss man lediglich die letzten Fully-Connected-Schichten ersetzen und mit den eigenen Bilddaten und entsprechenden Klassen trainieren [Oqu14]. Mit diesem Verfahren

kann viel Zeit und Rechenleistung gespart werden, weil die grundlegende Merkmalsextraktion nicht neu gelernt werden muss. Ein Datensatz der häufig für das Training von CNN-Architekturen verwendet wird ist ImageNet. ImageNet umfasst 14.197.122 Bildern und 21.841 sogenannte Synonymmengen (engl. synonym sets) [Den09]. In Synonymmengen sind Wörter der gleichen Bedeutung zusammengefasst. Zusätzlich sind alle Wörter auch hierarchisch nach dem WordNet-Projekt eingeordnet. In dieser Arbeit werden CNN-Architekturen verwendet, die mit dem ImageNet-Datensatz vorgenutzt wurden.

Eine Übersicht von bekannten CNN-Architekturen ist in Abbildung 4.7 dargestellt. In dieser Darstellung werden die neuronalen Netze anhand ihrer erreichten Genauigkeit (engl. accuracy) und der benötigten Operationen bei der Berechnung einer Klassifizierung eingeordnet [CPC16]. Die Größe der Kreise zeigt die Anzahl der Parameter der jeweiligen Architektur.

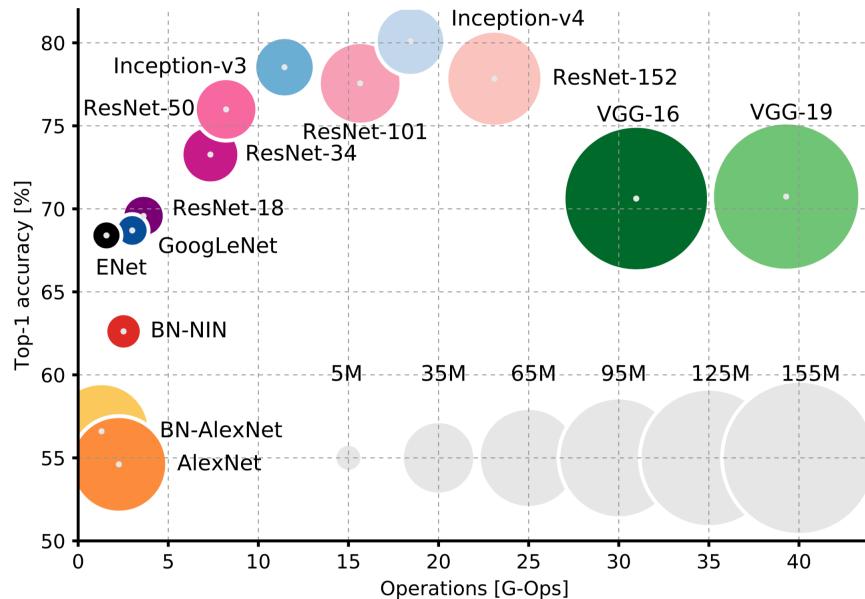


Abbildung 4.7: Vergleich von vortrainierten Convolutional Neural Networks, entnommen aus [CPC16]

Von diesen Architekturen wird in dieser Arbeit die Inception-V3-Architektur [Sze15] als Basis verwendet. Diese Architektur hat wenige Parameter, was eine geringe Anzahl benötigter Operationen zur Folge hat, bei gleichzeitig sehr guter Genauigkeit [CPC16]. Das Inception-V4 Modell ist zur Zeit dieser Arbeit noch nicht

bei Keras verfügbar und kann daher nicht verwendet werden. Die zweite Architektur in dieser Arbeit ist die Xception-Architektur, welche auf den gleichen Prinzipien wie das Inception-V3-Netz entwickelt wurde. Diese Xception-Architektur wird in dieser Arbeit verwendet, weil sie dieselbe Anzahl Parameter wie die Inception-V3-Architektur hat, aber noch höhere Genauigkeit erzielt [Cho17]. In den folgenden Absätzen wird der Vorteil der Inception-Architekturen im Vergleich zu normalen tiefen CNNs erklärt und die beiden Architekturen Inception-V3 und Xception werden vorgestellt.

Einige Jahre waren tiefe CNNs mit vielen einfach gestapelten Convolution- und Pooling-Schichten, wie beispielsweise die VGG-16-Architektur [SZ14], der Stand der Technik und erreichten gute Genauigkeiten bei der Klassifizierung von Bildern. Der Nachteil waren viele Parameter und ein sehr hoher Rechenaufwand [CPC16]. Genau an diesen Problemen setzen Architekturen wie das Inception-V3 Netz an. Es werden verschiedenen Prinzipien angewendet, um den benötigten Rechenaufwand zu senken und dennoch die Genauigkeit zu verbessern. Dabei werden Convolution-Operationen nicht nur gestapelt, sondern auch parallel zu anderen Convolution-Operationen ausgeführt [Sze15].

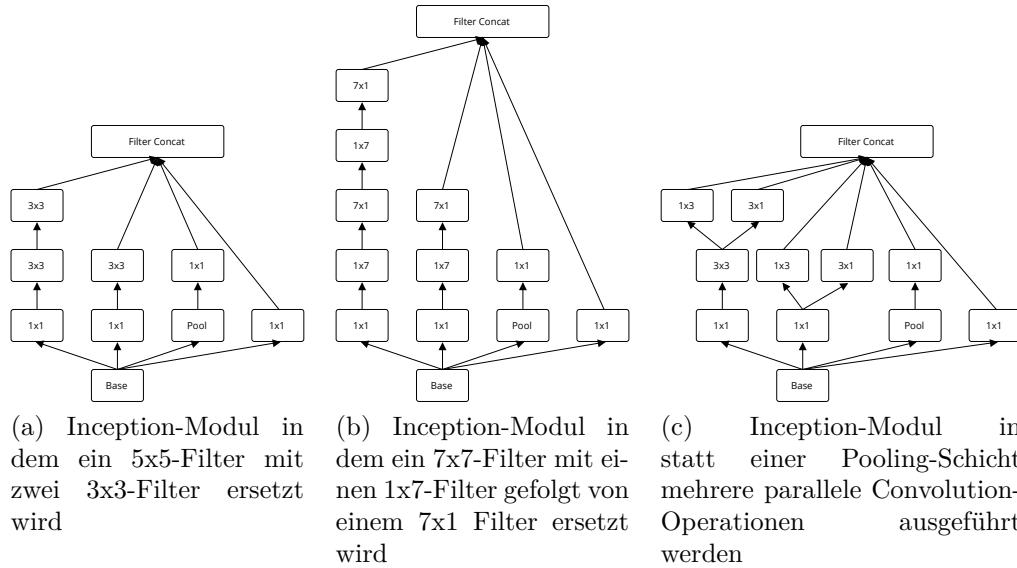


Abbildung 4.8: Inception-Module der Inception-V3 Architektur [Sze15]

Es wird argumentiert [Sze15], dass ein Filter mit der Dimension 5x5 durch

zwei aufeinanderfolgende Filter der Dimension 3x3 ersetzt werden kann, was die benötigte Rechenleistung deutlich reduziert [Sze15]. Diese Funktionalität wird in einem sogenannten Inception-Modul umgesetzt, was in Abbildung 4.8a zu sehen ist. Ein weiteres Prinzip ist das Ersetzen eines nxn-Filters durch einen 1xn-Filter gefolgt von einem nx1-Filter. Damit wird der benötigte Rechenaufwand, bei gleichbleibender Leistung, um 33 Prozent gesenkt. Dieses Prinzip ist in Abbildung 4.8b dargestellt. Für das dritte Prinzip wird argumentiert, dass bei einer starken Dimensionsreduzierung durch Pooling-Schichten viel Information aus dem Bild verloren gehen kann. Um dies zu verhindern, werden statt einer Pooling-Schicht mehrere parallele Convolution-Operationen ausgeführt. Dieses Prinzip wird mit dem Inception-Modul aus Abbildung 4.8c umgesetzt.

Die Architektur des gesamten Netzes ist in Abbildung 4.9 dargestellt. In dieser Arbeit wird ein Inception-V3 Netz verwendet, das mit dem Datensatz ImageNet [Den09] trainiert wurde. Außerdem werden die letzten Schichten der Inception-V3-Architektur (in der Abbildung der *final part*) am Ende des Netzes entfernt, weil sie für die Klassifizierung des ImageNet Datensatzes angepasst sind. Sie werden durch anderen Schichten ersetzt, die in den nachfolgenden Absätzen erklärt werden und der Klassifizierungsaufgabe von dieser Arbeit angepasst sind.

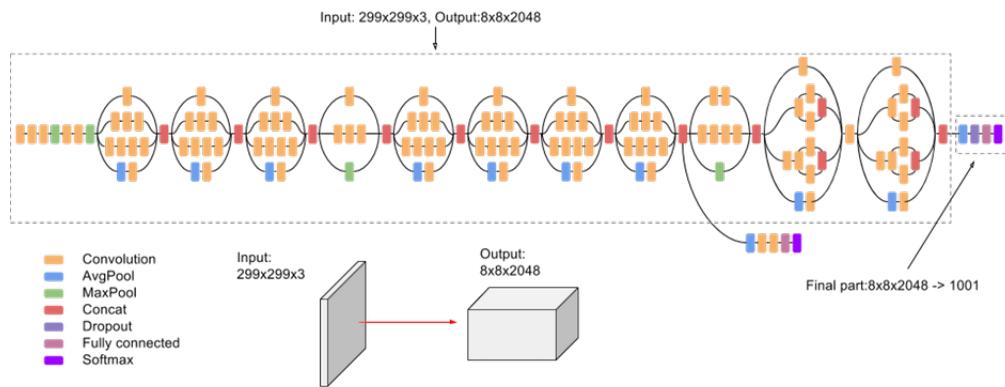


Abbildung 4.9: Architektur des Inception-V3 Netzes, entnommen aus [Clo18]

Die Grundidee bestehende Filter zu vereinfachen und um eine Dimension zu reduzieren ($7 \times 7 \rightarrow 1 \times 7$ und 7×1), wird mit der Xception-Architektur aufgegriffen und noch weiter vorangetrieben [Cho17]. Die Idee ist es, mit Inception-Modulen die

räumliche Convolution-Operation (engl. spatial convolution) und die Convolution-Operation über die Kanäle (engl. depthwise convolution) zu trennen. Das heißt, dass es einen Filter mit den Dimensionen $1 \times 1 \times c$ gibt, der auf ein Bild oder eine Feature Map der Tiefe c angewendet wird, gefolgt von einem Filter mit den Dimensionen $3 \times 3 \times 1$, der auf jede Schicht der Feature Map angewendet wird. Diese getrennten Convolution-Operationen (engl. depthwise separable convolution) sind in dem Inception-Modul in Abbildung 4.10 dargestellt.

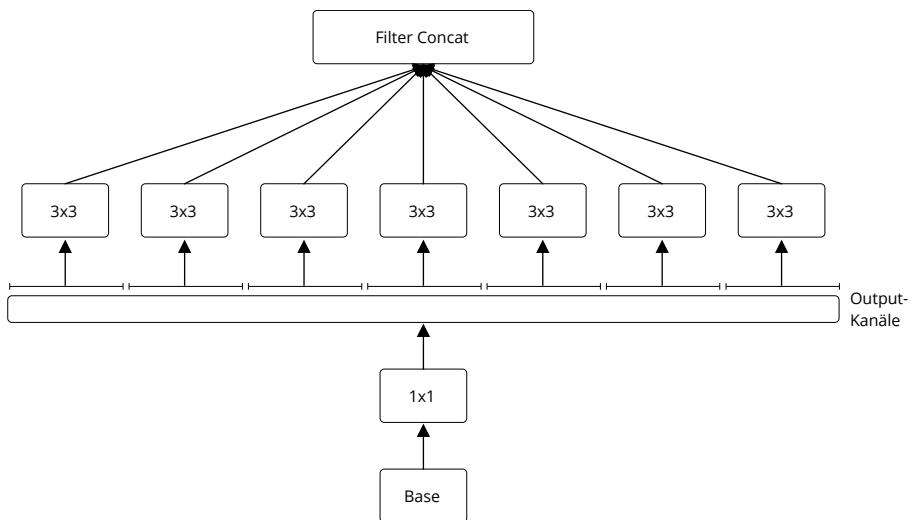


Abbildung 4.10: Inception-Modul der Xception-Architektur [Cho17]

Die gesamte Xception-Architektur besteht aus mehreren dieser getrennten Convolution-Operationen und ist in Abbildung 4.11 dargestellt. Wie die Gewichte des Inception-V3 Netzes, sind auch die Gewichte des Xception Netzes mit dem Imagenet Datensatz vorgenutzt. Auch hier werden die letzten Fully-Connected-Schichten entfernt und mit passenden Schichten für die Klassifizierung in dieser Arbeit ersetzt.

Fully-Connected- und LSTM-Schicht

Eine Fully-Connected-Schicht ist, wie in Abschnitt 2.2.2 beschrieben, eine Schicht mit n Neuronen in einem Multilayer-Perzeptron. Diese Schicht muss mit einer Anzahl Neuronen n und einer Aktivierungsfunktion φ konfiguriert werden. Die Ergebnisschicht in einem KNN ist immer eine Fully-Connected-Schicht. In dieser Arbeit werden in jeder Architektur zwei dieser Schichten verwendet, eine nach dem

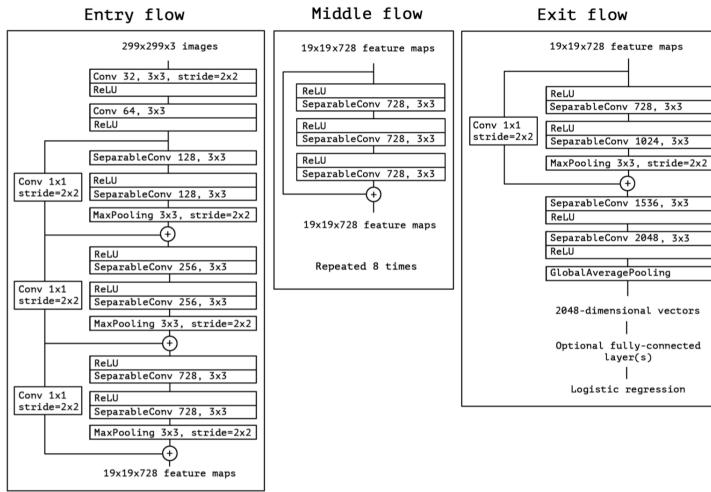


Abbildung 4.11: Xception-Architektur, entnommen aus [Cho17]

vortrainierten CNN bzw. der LSTM-Schicht und die andere als Ergebnisschicht. Die erste Fully-Connected Schicht besteht in dieser Arbeit immer aus 128 Neuronen und der ReLU-Aktivierungsfunktion [NH10]. In der Ergebnisschicht werden alle zuvor extrahierten Merkmale auf die Anzahl der Klassen heruntergebrochen. Daher besteht die Ergebnisschicht aus der gleichen Anzahl Neuronen wie es Klassen gibt. Als Aktivierungsfunktion für ein Multiklassen-Problem ($C > 2$), wie in dieser Arbeit, wird häufig die Softmax-Funktion verwendet [Bis06].

Wie in Abschnitt 2.2.4 beschrieben, werden LSTM-Schichten für die Sequenzerkennung eingesetzt. Da die LSTM-Schicht in dieser Arbeit nach der Merkmalsextraktion mit einem CNN verwendet wird, ist in Keras eine sogenannte *TimeDistributed*-Schicht nötig, um die Merkmale aller Bilder einer Sequenz der LSTM-Schicht übergeben zu können. Diese Umhüllung (engl. wrapper) hat keine zu konfigurierenden Parameter. Die LSTM-Schicht wird mit der Dimension $units = 256$ und der tanh-Aktivierungsfunktion konfiguriert.

Regularisierung

Das Ziel jedes KNNs ist es, nach dem Training gute Ergebnisse auf bisher unbekannten Daten zu erzielen. Dies wird Generalisierbarkeit eines Netzes genannt [Bis06]. Regularisierungsmethoden werden oft während des Trainings eingesetzt,

um die Generalisierbarkeit nach dem Training zu erhöhen. In dieser Arbeit werden die Methoden Dropout [Hin12] und Early Stopping [Pre98] verwendet.

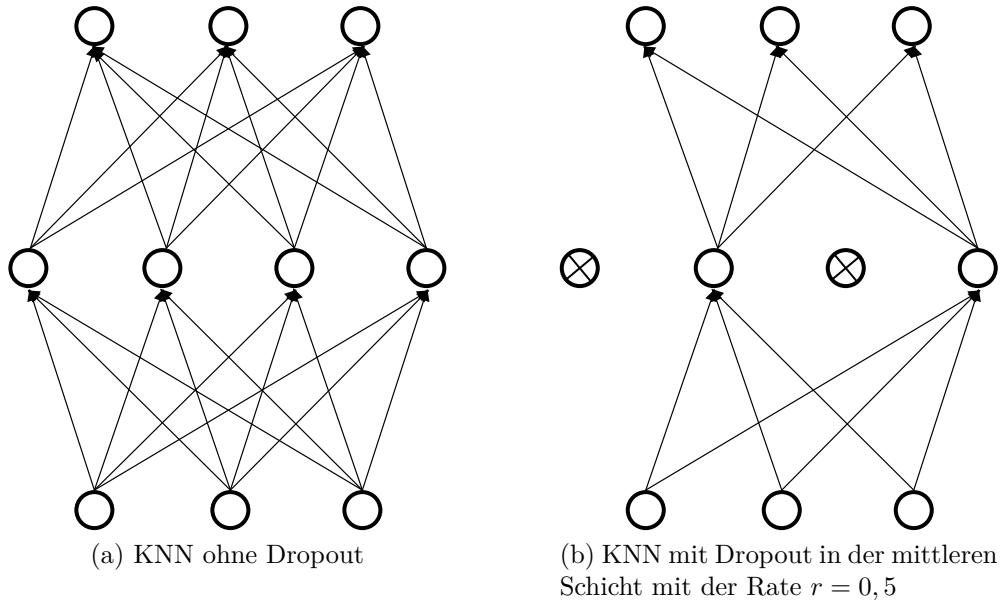


Abbildung 4.12: Regularisierung mit Dropout [Sri14]

Dropout ist eine Methode, um beim Training von KNNs Überanpassung (engl. Overfitting) zu vermeiden und insgesamt bessere Leistung bei der Kombination von verschiedenen Architekturen zu erreichen [Hin12]. Die Grundidee von Dropout ist es, die Komplexität eines KNNs zu reduzieren. Die Komplexität eines KNN beschreibt die Anzahl der zu trainierenden Parameter. Je mehr Parameter ein KNN besitzt, desto komplexer ist es. Dropout reduziert die Komplexität indem bei jedem Trainingsschritt zufällig einige Neuronen und die zugehörigen Kanten im Netz, die Parameter, entfernt. Dafür wird einer Schicht eine Rate r zugeordnet. Diese Rate r gibt an, welcher Anteil der Neuronen aus dieser Schicht zufällig entfernt wird. Bei $r = 0,5$ werden beispielsweise 50 Prozent aller Neuronen in jedem Trainingsschritt zufällig entfernt. Das Ergebnis ist, dass das KNN nicht zu abhängig von einzelnen Neuronen oder Gewichten wird und insgesamt bessere Ergebnisse erzielt werden können [Sri14]. In dieser Arbeit wird Dropout mit der Rate $r = 0,5$ verwendet. Obwohl Dropout keine eigenständige Schicht ist, wird es in Keras als Schicht modelliert. Dabei bezieht sich Dropout aber stets auf die

jeweilige Schicht davor. Ein Beispiel von Dropout ist in Abbildung 4.12 dargestellt.

Early Stopping ist eine Methode, bei der das Training eines KNNs abgebrochen wird, wenn die Genauigkeit oder der Fehler sich nicht weiter verbessern [Pre98]. In dieser Arbeit wird die Genauigkeit der Validierungsdaten (engl. validation accuracy) als Metrik verwendet und das Training wird beendet, wenn sich diese Metrik innerhalb von 20 Epochen nicht mindestens um den Wert 0,01 verbessert.

4.4.3 Wahl der Komponenten

Bei der Konfiguration von KNNs und den einzelnen Schichten können viele unterschiedliche Komponenten ausgewählt werden. Da es sich bei dieser Arbeit um einen Proof-of-Concept handelt, werden oftmals die Standardkonfigurationen von Keras verwendet. Diese Standardkonfigurationen basieren nach Chollet stets auf aktuellen Forschungsergebnissen und dem Stand der Technik bei KNNs [Cho15]. Diese Vorgehensweise erlaubt einerseits eine schnelle Umsetzung des Proof-of-Concept dieser Arbeit und bietet andererseits in nachfolgenden Arbeiten viel Potential die Ergebnisse weiter zu verbessern.

Vor jedem Training werden die KNNs mit einem Optimierer (engl. optimizer) und einer Fehlerfunktion (engl. loss function) kompiliert. Als Optimierer wird in dieser Arbeit Adam (adaptive moment estimation) verwendet. Dieser Optimierer vereint die Vorteile von den Optimierern AdaGrad (adaptive gradient algorithm) und RMSProp (root mean square propagation). Das Grundprinzip ist es, die Lernrate dynamisch anzupassen, was zu weniger benötigtem Speicher und einer hohen Konvergenzrate führt [KB15].

Neben der mittleren quadratischen Abweichung, die in Abschnitt 2.2.2 vorgestellt wurde, gibt es weitere Fehlerfunktionen. In dieser Arbeit wird die Kreuzentropie (engl. cross entropy) für Mehrklassenprobleme mit der folgenden Fehlerfunktion verwendet [Bis06]:

$$E(w_1, \dots, w_K) = - \sum_{n=1}^N \sum_{k=1}^K t_{nk} * \ln(y_{nk}) \quad (4.1)$$

Dabei beschreiben w_1, \dots, w_K die zu optimierenden Gewichte, N ist die Anzahl der Trainingsdaten und K die Anzahl der Klassen. t_{nk} gibt mit dem Wert 1 an, ob

der Dateninput n zur Klasse k gehört. Wenn n nicht zur Klasse k gehört, nimmt t_{nk} den Wert 0 an. y_{nk} ist die berechnete Wahrscheinlichkeit des Klassifikators, dass der Dateninput n zur Klasse k gehört.

Das Ziel der Experimente ist das Erreichen einer möglichst hohen Genauigkeit (engl. accuracy) bei der Klassifizierung von Bildsequenzen. Bei Mehrklassenproblemen (engl. multi-class problem) werden häufig die Top-1 und Top-5 Genauigkeit bzw. der Top-1 und Top-5 Fehler untersucht [Sze15; SZ14; Cho17]. Das Ergebnis einer Multiklassen-Klassifikation für jeden Input ist ein Vektor mit Wahrscheinlichkeiten für jede Klasse. Bei der Top-1 Genauigkeit bzw. dem Top-1 Fehler wird jeweils nur die Klasse mit der höchsten Genauigkeit betrachtet und mit der richtigen Klasse abgeglichen. Bei der Top-5 Genauigkeit bzw. dem Top-5 Fehler werden jeweils die fünf Klassen mit den höchsten Wahrscheinlichkeit mit der richtigen Klasse abgeglichen. Da in dieser Arbeit nur insgesamt fünf Klassen betrachtet werden, wird nur die Top-1 Genauigkeit als Metrik verwendet.

Anhand dieser Metrik werden die unterschiedlichen Architekturen aus dem Abschnitt 4.4.2 mit 100 Epochen und Early Stopping trainiert und dann getestet. Aus der Variation von drei Komponenten mit jeweils zwei Variationen ergeben sich acht Architekturen. Die Komponenten mit ihren Variationen sind in Tabelle 4.8 zusammengefasst.

Komponenten	Variationen
Prinzip der Klassifizierung	einzelne Bilder, Bildsequenzen
Convolutional Neural Network	Inception-V3, Xception
Regularisierung	ohne Dropout, mit Dropout

Tabelle 4.8: Komponenten die in den Experimenten variiert werden

Komponenten zusammenfügen

Auf Basis der oben beschriebenen Komponenten werden acht verschiedene KNN-Architekturen erstellt. Dabei wird zwischen den Architekturen für die Klassifizierung einzelner Bilder und den Architekturen für die Klassifizierung von Bildsequenzen unterschieden. Beide grundlegenden Architekturen sind schematisch in

Abbildung 4.13 dargestellt.

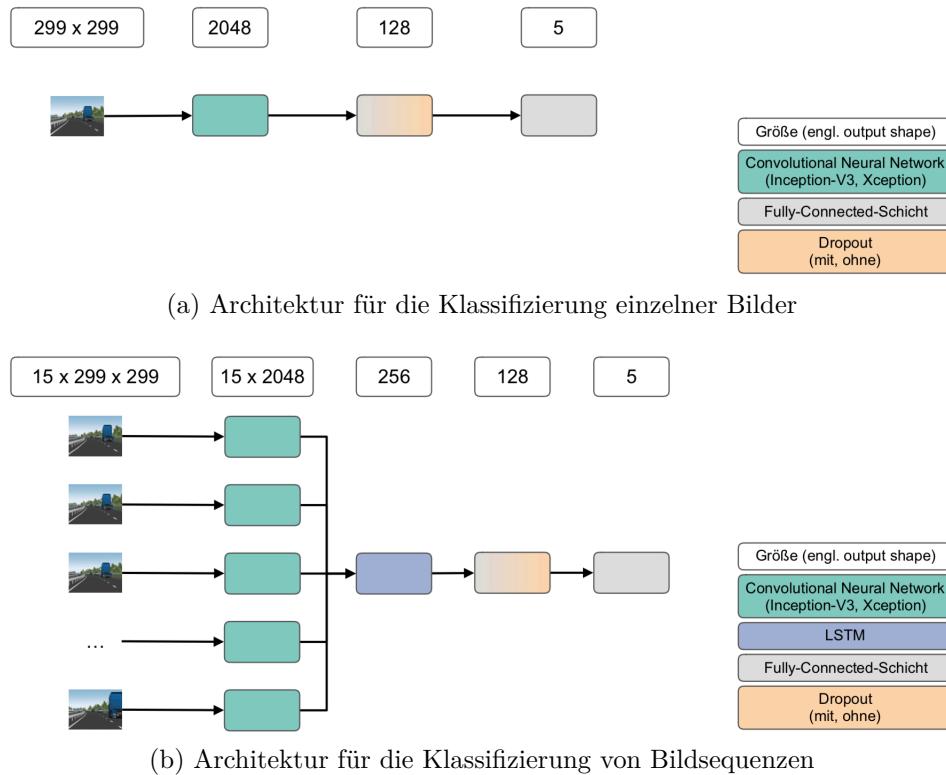


Abbildung 4.13: Grundlegende KNN-Architekturen in dieser Arbeit

Ausgehend von der Darstellung in Abbildung 4.13 wird bei den Architekturen für die Klassifizierung von einzelnen Bildern zwischen den zwei CNNs Inception-V3 [Sze15] und Xception [Cho17] und zwischen einer Version mit und ohne Dropout [Sri14] unterschieden. Damit ergeben sich vier verschiedene Architekturen. Als Basis wird dafür bei jeder Architektur eines der oben beschriebenen CNNs verwendet. Nach den Convolution- und Pooling-Schichten des jeweiligen CNNs wird eine Fully-Connected-Schicht mit 128 Neuronen und der ReLU-Aktivierungsfunktion eingefügt. Diese Schicht wird in zwei der vier Architekturen mit Dropout, mit $r = 0, 5$, konfiguriert. Abschließend wird eine weitere Fully-Connected-Schicht mit fünf Neuronen, was der Anzahl der zu klassifizierenden Szenen entspricht, und der Softmax-Aktivierungsfunktion angehängt. Diese vier Konfigurationen sind in der Tabelle 4.9 mit den jeweiligen Komponenten und Hyperparametern zusammenge-

fasst.

Architekturen					
A	B	C	D		
Input shape: 299x299x3					
Inception-V3		Xception			
Output shape: 2048		Output shape: 2048			
Fully-Connected-Schicht $units = 128$, $activation = \text{ReLU}$					
Output shape: 128					
ohne Dropout -	Dropout $r = 0,5$	ohne Dropout -	Dropout $r = 0,5$		
Fully-Connected-Schicht (Ergebnisschicht) $units = 5$, $activation = \text{Softmax}$					
Output shape: 5					

Tabelle 4.9: KNN-Architekturen für die Klassifizierung von Bildsequenzen

Analog zu den Architekturen für die Klassifizierung von einzelnen Bildern, werden bei den Architekturen für die Klassifizierung von Bildsequenzen ebenfalls die zwei verschiedene CNN-Architekturen variiert. Der Unterschied ist, dass diese CNNs jeweils in eine *TimeDistributed*-Schicht eingehüllt werden, um danach eine LSTM-Schicht mit 256 Neuronen anzuhängen. Danach wird wie bei den anderen Architekturen eine Fully-Connected-Schicht mit 128 Neuronen und der ReLU-Aktivierungsfunktion eingefügt und bei zwei von vier Architekturen mit der Regularisierungsmethode Dropout ($r = 0,5$) konfiguriert. Die Ergebnisschicht ist ebenfalls identisch mit 5 Neuronen und der Softmax-Aktivierungsfunktion. Die resultierenden vier Architekturen für die Klassifizierung von Bildsequenzen sind in Tabelle 4.10 mit ihren Komponenten und Hyperparametern zusammengefasst.

Architekturen					
E	F	G	H		
Input shape: 15x299x299x3					
$TimeDistributed$ mit Inception-V3 Output shape: 15x2048		$TimeDistributed$ mit Xception Output shape: 15x2048			
LSTM-Schicht $units = 256, activation = \tanh$ Output shape: 256					
Fully-Connected-Schicht $units = 128, activation = \text{ReLU}$ Output shape: 128					
ohne Dropout -	Dropout $r = 0,5$	ohne Dropout -	Dropout $r = 0,5$		
Fully-Connected-Schicht (Ergebnisschicht) $units = 5, activation = \text{Softmax}$ Output shape: 5					

Tabelle 4.10: KNN-Architekturen für die Klassifizierung einzelner Bilder

Kapitel 5

Ergebnis

In diesem Kapitel werden die Ergebnisse der trainierten KNNs aus Abschnitt 4.4.3 vorgestellt. Jede Architektur wird mit einer Maximalanzahl von 100 Epochen trainiert, was einem Kompromiss aus benötigtem Rechenaufwand und der Maximierung von Trainingsschritten entspricht. Aufgrund der Regularisierungsmethode Early Stopping wird diese Anzahl jedoch nie erreicht. Das Training bricht jeweils ab, wenn sich die Klassifizierungsgenauigkeit der Validierungsdaten innerhalb von 20 Epochen nicht um mindestens 0,01 verbessert. Nach Abbruch werden jeweils die Gewichte der Epoche mit der höchsten Validierungsgenauigkeit wiederhergestellt.

In Abschnitt 5.1 wird die Genauigkeit der jeweiligen Netzarchitekturen bei der Klassifizierung von realen Testdaten und die Anzahl der trainierten Epochen hinsichtlich der unterschiedlichen Komponenten aus Abschnitt 4.4.3 verglichen. Danach wird in Abschnitt 5.2 die Genauigkeit der Klassifizierung zwischen realen und synthetischen Testdaten untersucht. Im Anschluss werden in Abschnitt 5.3 die Unterschiede bei der Erkennung von verschiedenen Klassen erörtert.

5.1 Variation der Komponenten

In diesem Abschnitt werden die Unterschiede der verschiedenen Architekturen aus Abschnitt 4.4.2 noch einmal kurz aufgegriffen und dann die dazugehörigen Ergebnisse vorgestellt. Die Ergebnisse sind in Tabelle 5.1 zusammengefasst.

Architektur	Beschreibung	Genauigkeit	Epochen
A	Inception-V3	0,73	4
B	Inception-V3 Dropout	0,64	1
C	Xception	0,54	1
D	Xception Dropout	0,48	9
E	Inception-V3 LSTM	0,36	80
F	Inception-V3 LSTM Dropout	0,95	12
G	Xception LSTM	0,38	26
H	Xception LSTM Dropout	0,61	9

Tabelle 5.1: Ergebnisse der verschiedenen Architekturen bei der Klassifizierung der realen Testdaten

Prinzip der Klassifizierung

Bei dem Prinzip der Klassifizierung wird in dieser Arbeit zwischen KNNs für die Erkennung von einzelnen Bildern und für die Erkennung von 3-Sekunden-Videos mit jeweils 15 Bildern unterschieden. Die detaillierten Architekturen sind in Abschnitt 4.4.2 vorgestellt.

Insgesamt wird die höchste Genauigkeit bei der Klassifizierung von realen Testdaten von 0,95 mit der Architektur F für Videoerkennung erreicht. Das Ergebnis dieser Architektur nach einzelnen Klassen ist auch in Abbildung 5.1 in der Konfusionsmatrix (engl. confusion matrix) dargestellt. Auf die Unterschiede der Genauigkeiten zwischen den einzelnen Klassen wird in Abschnitt 5.3 genauer eingegangen. Die niedrigste Genauigkeit von 0,36 wird mit der Architektur E für Videoerkennung erreicht. Architekturen für die reine Bilderkennung erreichen Genauigkeiten zwischen 0,48 (Architektur D) und 0,73 (Architektur A). Damit erzielen diese Ar-

chitekturen Ergebnisse mit weniger Varianz im Vergleich zu den Architekturen mit LSTM-Schicht. Eine Erklärung hierfür kann die niedrigere Komplexität der KNNs sein. Bei einer höheren Komplexität, also mehr trainierbaren Parametern, kann es sein, dass die Genauigkeiten zwischen verschiedenen Architekturen stärker schwankt, weil mehr Parameter für die richtige Klassifizierung benötigt werden.

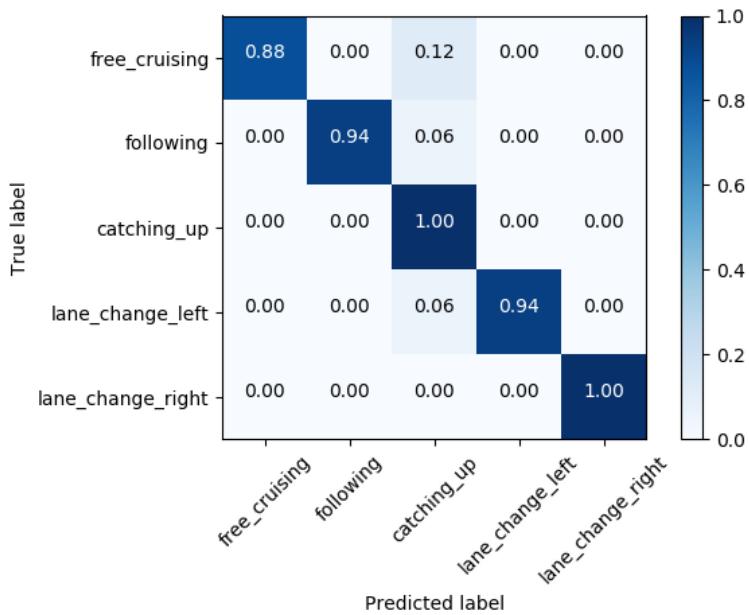


Abbildung 5.1: Konfusionsmatrix der Klassifizierung von realen Testdaten mit der Architektur F (Inception-V3, LSTM, Dropout)

Convolutional Neural Networks

Der Vergleich zwischen den zwei CNN-Architekturen Inception-V3 und Xception zeigt ein klares Bild. In drei von vier Fällen kann die Architektur mit der Inception-V3-Architektur als Basis bessere Ergebnisse erzielen. Die Differenz bei der Genauigkeit liegt dabei zwischen 0,16 und 0,34. Nur die Architektur G erzielt eine höhere Genauigkeit als Architektur E mit einem Unterschied von 0,02. Dieser Unterschied überrascht, da die Xception-Architektur in anderen Arbeiten [Cho17] höhere Genauigkeiten bei der Klassifizierung von Bildern erreichte.

Regularisierung mit Dropout

Es ist auffällig, dass die Architekturen für Videoklassifizierung mit Dropout in der vorletzten Fully-Connected-Schicht deutlich bessere Ergebnisse erzielen können im Vergleich zu denselben Architekturen ohne Dropout. Im Gegensatz dazu erzielen die Architekturen für die Erkennung einzelner Bilder die besseren Genauigkeit ohne Regularisierung mit Dropout in der vorletzten Schicht. Die Erklärung kann auf die Grundidee von Dropout zurückgeführt werden. Dropout ist eine Methode um die Komplexität eines KNNs während dem Training zu reduzieren. Daher hat Dropout einen sehr positiven Effekt auf die Architekturen für Videoklassifizierung, die mit der LSTM-Schicht deutlich komplexer sind, also mehr Parameter besitzen, als die Architekturen für Bildklassifizierung.

In Abbildung 5.2 ist beispielhaft die Entwicklung der Genauigkeit der Trainings- und Testdaten während dem Training von den Architekturen E und F dargestellt. Es ist deutlich zu sehen, dass die Anzahl der Epochen bei Architekturen mit LSTM-Schicht mit Dropout stark reduziert werden kann und die Genauigkeit weniger sprunghaft ist und schneller konvergiert.

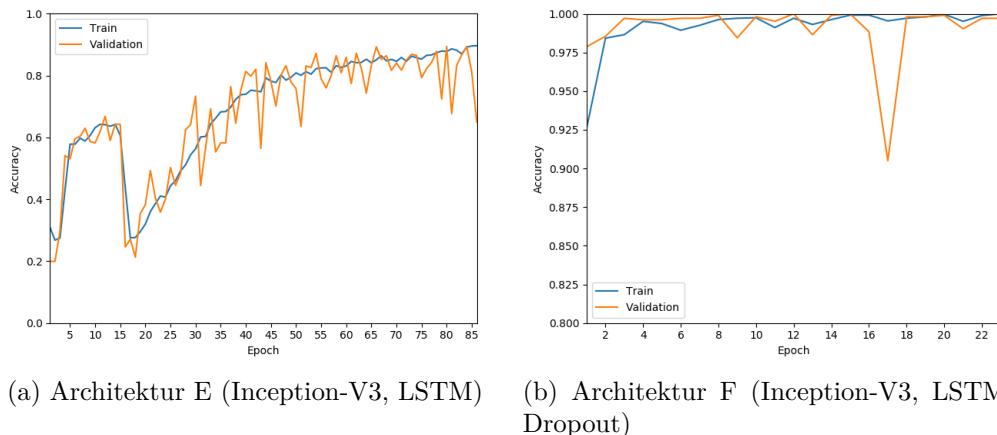


Abbildung 5.2: Genauigkeit der Trainings- und Testdaten während dem Training von zwei verschiedenen Architekturen

Eine Erklärung dafür ist die Komplexität der Architekturen mit LSTM-Schicht. Diese Architekturen haben mehr trainierbare Parameter und sind daher anfälliger für eine Überanpassung [Hin12]. Dropout reduziert die Anzahl der Parameter und

damit die Komplexität, was zu weniger benötigten Trainingsschritten und einer schnelleren Konvergenz der Gewichtsanpassung führt. Die Architekturen mit reinen CNNs sind weniger komplex und daher hat Dropout keine oder eine negative Auswirkung auf das Ergebnis.

Anzahl der Epochen

Es lässt sich beobachten, dass die Architekturen für die Klassifizierung von Videos mit bis zu 86 Epochen deutlich mehr Trainingsschritte benötigen als die Architekturen für reine Bilderkennung. Eine mögliche Erklärung dafür ist, dass reine CNNs eine weniger komplexe Architektur und weniger Parameter haben als die Architekturen mit einer LSTM-Schicht. Mehr Gewichte, die angepasst werden können, bedeutet eine höhere Komplexität und dementsprechend mehr benötigte Epochen.

Zu den Unterschieden der Epochenzahl zwischen der Inception-V3- und der Xception-Architektur lässt sich in dieser Arbeit keine eindeutige Aussage treffen. Die benötigte Epochenzahl bei beiden Architekturen variiert sehr stark.

5.2 Synthetische und reale Testdaten

In diesem Abschnitt werden die Unterschiede der Genauigkeiten bei der Klassifizierung zwischen realen und synthetischen Testdaten untersucht. Die Ergebnisse sind in Tabelle 5.2 zusammengefasst. Es ist nicht überraschend, dass die Genauigkeiten bei der Klassifizierung von synthetischen Testdaten überwiegend deutlich höher ist, weil die KNNs mit 95 Prozent synthetischer und nur 5 Prozent realer Daten trainiert wurden. Das bestätigt auch die Ergebnisse der vorangegangenen Arbeit, die in Abschnitt 2.2.5 diskutiert werden.

Es ist auffällig, dass die Klassifizierungsgenauigkeit von einzelnen synthetischen Bildern mit der Xception Architektur niedriger ist als die Klassifizierungsgenauigkeit von realen Testbildern. Eine Untersuchung dieser Ursache sollte in zukünftigen Arbeiten stattfinden.

Architektur	Beschreibung	Genauigkeit	
		Reale Testdaten	Synthetische Testdaten
A	Inception-V3	0,73	0,96
B	Inception-V3 Dropout	0,64	0,96
C	Xception	0,54	0,2
D	Xception Dropout	0,48	0,36
E	Inception-V3 LSTM	0,36	0,94
F	Inception-V3 LSTM Dropout	0,95	0,99
G	Xception LSTM	0,38	0,99
H	Xception LSTM Dropout	0,61	0,91

Tabelle 5.2: Ergebnisse der Klassifizierung von synthetischen und realen Testdaten

5.3 Genauigkeit der einzelnen Szenarien

Die Betrachtung von Klassifizierungsgenauigkeiten der einzelnen Klassen ergibt kein eindeutiges Bild. Bei verschiedenen Architekturen werden verschiedene Szenen teilweise besser oder schlechter erkannt. Hervorzuheben ist, dass es deutliche Unterschiede gibt, diese aber keinem erkennbaren Muster folgen. Eine Übersicht der Genauigkeit der Klassifizierung von realen Testdaten auf der Ebene von Szenarienklassen ist in Tabelle 5.3 dargestellt.

Ausgehend von der Architektur mit der höchsten Gesamt-Genauigkeit, Architektur F, können die folgenden Überlegungen zu unterschiedlichen Genauigkeiten zwischen Klassen angestellt werden. Bei Architektur F wurden die Klassen *catching up* und *lane change left* sehr gut erkannt. Die Klasse *free cruising* wurde am schlechtesten erkannt. Da die Erkennung auf reinen Bilddaten basiert, kann argumentiert

werden, dass Szenarien der Klasse *free cruising* den Szenarien aus den Klassen *following* und *catching up* ähneln, weil der Unterschied nur im Abstand zu anderen Fahrzeuge liegt. Dagegen sind die Klassen *lane change left* und *lane change right* eindeutiger, weil jeweils die Fahrbahnmarkierung überquert wird und damit ein eindeutiges Merkmal existiert. Diese Erklärung ist eine Interpretation des Autors und kann bisher nicht weiter belegt werden.

Architektur	Free	Following	Catching	Lane change	Lane change
	cruising		up	left	right
A	0,65	0,76	1,00	0,35	0,88
B	0,94	0,47	0,88	0,29	0,59
C	0,76	0,94	0,94	0,00	0,06
D	0,00	0,82	0,82	0,76	0,00
E	0,59	0,47	0,41	0,06	0,29
F	0,88	0,94	1,00	0,94	1,00
G	0,47	0,06	0,18	0,59	0,06
H	0,71	1,00	0,71	0,35	0,29
Durchschnitt	0,62	0,68	0,74	0,42	0,40

Tabelle 5.3: Genauigkeit der Klassifizierung von realen Testdaten auf der Ebene von Szenarienklassen

Eine weitere interessante Beobachtung ist, dass es teilweise zu Klassifizierungs- genauigkeiten von unter 0,2 kommt. In dem Fall sind die Ergebnisse des Klassifikators schlechter als eine zufällige Bestimmung der Klasse.

Um an dieser Stelle besser verstehen zu können auf welcher Basis die KNNs die Bilder und Bildsequenzen erkennen, sollten in nachfolgenden Arbeiten weitere Untersuchungen dazu angestellt werden. So ist es beispielsweise interessant, welche Teile der einzelnen Bilder stärker oder auch weniger stark bei der Klassifizierung berücksichtigt werden.

Kapitel 6

Zusammenfassung

In dieser Arbeit wurde ein Konzept für die Klassifizierung von Szenarien entwickelt und umgesetzt. Dafür wurden im ersten Schritt beispielhaft sieben Fahrszenarien auf *funktionaler und logischer Ebene* definiert. Auf Basis der *logischen Definitionen* wurden Variablen abgeleitet, die das jeweilige Szenario regelbasiert annotieren können.

Im zweiten Schritt wurden mit der Simulationssoftware CarMaker 2.160 Kilometer und damit 326.108 Szenen simuliert. Zu jeder Szene wurden Bilddaten und die zuvor abgeleiteten Variablen, sogenannte Signaldaten, erzeugt. Die Bilddaten wurde mit diesen Signaldaten regelbasiert, nach der *logischen Definition*, annotiert und schließlich zu 3-Sekunden-Szenarien zusammengefasst. Im Anschluss wurden reale Szenarien manuell annotiert.

Im dritten Schritt wurde für das Training von KNNs ein Trainingsdatensatz mit fünf Klassen aus 95 Prozent synthetischen und 5 Prozent realen Szenarien erstellt. Insgesamt wurden acht unterschiedliche KNN-Architekturen designed, trainiert und getestet. Das beste Modell erreicht Genauigkeiten von 95 Prozent bei der Klassifizierung von realen Testszenarien.

6.1 Ergebnis und Diskussion

Das Ziel dieser Arbeit war es, ein Proof-of-Concept für die Erkennung von Fahrszenarien auf Basis von überwiegend synthetischen Bilddaten zu erstellen. Der Fokus

lag dabei auf den folgenden Punkten:

- Trainingsdaten müssen nicht mehr aufwendig manuell annotiert werden.
- Szenarien werden auf der Basis von Bilddaten klassifiziert.
- Bisher unbekannte Szenarien können gefunden werden.

Im zweiten Schritt des Ansatzes dieser Arbeit wurde eine Methodik entwickelt und umgesetzt, mit der synthetische Bilddaten simuliert und annotiert werden können. Damit ist es in Zukunft möglich große Datenmengen für das Training von neuronalen Netzen für die Erkennung von Fahrszenarien zu generieren. Diese Methodik ist nicht auf bestimmte Szenarien festgelegt und kann in nachfolgenden Arbeiten und in der Praxis für weitere Fahrszenarien angewandt werden. Im dritten Schritt dieser Arbeit wurde zusätzlich gezeigt, dass sehr gute Klassifizierungsgenauigkeiten mit 95 Prozent synthetischen und nur 5 Prozent realen Trainingsdaten erreicht werden können. Damit ist es möglich den manuellen Aufwand für die Annotation von realen Trainingsdaten stark zu reduzieren.

In vorherigen Arbeiten, wie in Abschnitt 2.1.2 beschrieben, wurden Fahrszenarien auf der Basis von Signaldaten klassifiziert. Nach bestem Wissen des Autors wurden in dieser Arbeit zum ersten Mal Fahrszenarien mit ausschließlich Bilddaten klassifiziert. Damit soll die Grundlage gelegt werden, um in weiteren Arbeiten zusätzliche Fahrszenarien klassifizieren zu können und schließlich bisher unbekannte Szenarien für die Absicherung von hochautomatisierten FAS zu finden. Die Grundidee ist es, dass ein Klassifikator, der mit allen bisher bekannten Szenarien trainiert wurde, bekannte Szenarien klassifizieren und unbekannte Szenarien identifizieren kann. Dieses Vorgehen wurde in dieser Arbeit mit fünf beispielhaften Szenarien in einem Proof-of-Concept gezeigt.

6.2 Ausblick

Der nächste Schritt ist, wie in Abschnitt 6.1 beschrieben, die Übertragung dieses Ansatzes auf alle bisher bekannten Szenarien um unbekannte Fahrszenarien zu identifizieren. Damit kann ein Beitrag zur Absicherung von FAS geliefert werden.

Eine zweite Möglichkeit für nachfolgende Arbeiten ist es, die KNN-Architekturen weiter zu verbessern. In dieser Arbeit wurden meist Standardkonfigurationen von Keras [Cho15] für die einzelnen Schichten verwendet und keine umfangreichen empirischen Untersuchungen durchgeführt. Es ist anzunehmen, dass die Genauigkeit noch gesteigert und die Trainingszeit reduziert werden kann.

Eine zusätzlicher, wichtiger nächster Schritt ist die Vergrößerung der Datenvarianz. Bisher wurden Szenarien ausschließlich auf einer 4-spurigen Autobahn bei Tageslicht simuliert. Für die Anwendung in der Praxis müssen auch Fahrten unter anderen Umständen (bei Nacht, in der Stadt etc.) simuliert und für das Training eines Klassifikators verwendet werden. Nur so kann die Anwendbarkeit in der Praxis sichergestellt werden.

Eine vierte Möglichkeit ist die Erweiterung des Klassifikators. Wie zuvor beschrieben, fokussierten sich bisherige Arbeiten auf die Klassifizierung auf Basis von Signaldaten. In dieser Arbeit werden ausschließlich Bilddaten verwendet. Ein möglicher nächster Schritt ist die Kombination beider Ansätze. Es könnten KNN-Architekturen entwickelt werden, die sowohl Bilddaten als auch Signaldaten für die Klassifizierung verwenden. Dadurch könnten möglicherweise einzelne Klassen, die bisher weniger gut erkannt werden, mit einer höheren Genauigkeit klassifiziert werden.

In dieser Arbeit wurde ein Ansatz für die Klassifizierung von Fahrszenarien auf Basis von simulierten Bilddaten entwickelt und umgesetzt. Dieser Ansatz kann als Grundlage für verschiedene weitere Untersuchungen herangezogen werden, um in Zukunft autonome Fahrfunktionen abzusichern.

Literaturverzeichnis

- [ABR16] Cesar Arroyo, Luis M. Bergasa und Eduardo Romera. “Adaptive fuzzy classifier to detect driving events from the inertial sensors of a smartphone”. In: *IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. 2016.
- [Ami17] Shervine Amidi. *A detailed example of how to use data generators with Keras*. 2017. URL: <https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly> (besucht am 21.10.2018).
- [AW17] Christian Amersbach und Hermann Winner. “Functional decomposition: An approach to reduce the approval effort for highly automated driving”. In: *8. Tagung Fahrerassistenz*. 2017.
- [Bac17] Johannes Bach u. a. “Reactive-replay approach for verification and validation of closed-loop control systems in early development”. In: *SAE World Congress* (2017).
- [Bag17] Gerrit Bagschik u. a. “Szenarien für Entwicklung, Absicherung und Test von automatisierten Fahrzeugen”. In: *11. Workshop Fahrerassistenzsysteme und automatisiertes Fahren*. 2017.
- [BF15] Guy Berg und Berthold Färber. “Vehicle in the Loop”. In: *Handbuch Fahrerassistenzsysteme*. 2015.
- [Bis06] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
- [BOS16] Johannes Bach, Stefan Otten und Eric Sax. “Model based scenario specification for development and test of automated driving functions”. In: *Intelligent Vehicles Symposium (IV)*. 2016.

- [Bri90] John S. Bridle. "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition". In: *Neurocomputing*. 1990.
- [Bun05] Der Beauftragte der Bundesregierung für Informationstechnik. *V-Modell XT*. 2005. URL: https://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/vmodell_xt_node.html (besucht am 19. 10. 2018).
- [Cer16] Javier Cervantes-Villanueva u. a. "Vehicle maneuver detection with accelerometer-based classification". In: *Sensors* (2016).
- [CHK16] Zehra Camlica, Allaa Hilal und Dana Kulic. "Feature abstraction for driver behaviour detection with stacked sparse auto-encoders". In: *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2016.
- [Cho15] François Chollet. *Keras*. 2015. URL: <https://keras.io> (besucht am 15. 11. 2018).
- [Cho17] François Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.
- [Clo18] Google Cloud. *Advanced guide to Inception V3*. 2018. URL: <https://cloud.google.com/tpu/docs/inception-v3-advanced> (besucht am 23. 11. 2018).
- [Com14] SAE On-Road Automated Vehicle Standards Committee u. a. "Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems". In: *SAE Standard* (2014).
- [Cor16] Marius Cordts u. a. "The Cityscapes dataset for semantic urban scene understanding". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
- [CPC16] Alfredo Canziani, Adam Paszke und Eugenio Culurciello. "An analysis of deep neural network models for practical applications". In: *Computing Research Repository (CoRR)* (2016).

- [CZ17] Joao Carreira und Andrew Zisserman. “Quo vadis, action recognition? A new model and the Kinetics dataset”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.
- [Den09] Jia Deng u. a. “ImageNet: A large-scale hierarchical image database”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009.
- [Don15] Jeffrey Donahue u. a. “Long-term recurrent convolutional networks for visual recognition and description”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.
- [FPZ16] Christoph Feichtenhofer, Axel Pinz und Andrew Zisserman. “Convolutional two-stream network fusion for video action recognition”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
- [Gei13] Andreas Geiger u. a. “Vision meets robotics: The KITTI dataset”. In: *The International Journal of Robotics Research* (2013).
- [Gmb18] IPG Automotive GmbH. *CarMaker*. 2018. URL: <https://ipg-automotive.com/de/produkte-services/simulation-software/carmaker/> (besucht am 20.11.2018).
- [Goo18a] Google. *Route 1: Mannheim nach Rostock*. 2018. URL: <https://drive.google.com/open?id=1MpI5teZVqANhFJ6YB-n9gWijlOB0-mZ9&usp=sharing> (besucht am 21.11.2018).
- [Goo18b] Google. *Route 2: Karlsruhe nach Kandel*. 2018. URL: <https://drive.google.com/open?id=1oEwdcaIl8bY4W77dTlR6U68UAfOP1TxY&usp=sharing> (besucht am 21.11.2018).
- [Gra12] Alex Graves. *Supervised sequence labelling with recurrent neural networks*. Springer, 2012.
- [Gru17] Richard Gruner u. a. “Spatiotemporal representation of driving scenarios and classification using neural networks”. In: *Intelligent Vehicles Symposium*. 2017.

- [Hin12] Geoffrey E. Hinton u. a. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *Neural and Evolutionary Computing* (2012).
- [HK15] Stephan Hakuli und Markus Krug. “Virtuelle Integration”. In: *Handbuch Fahrerassistenzsysteme*. 2015.
- [HS97] Sepp Hochreiter und Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural computation* (1997).
- [Joh17] Matthew Johnson-Roberson u. a. “Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?” In: *International Conference on Robotics and Automation (ICRA)*. 2017.
- [Kar14] Andrej Karpathy u. a. “Large-scale video classification with convolutional neural networks”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.
- [KB15] Diederik P. Kingma und Jimmy Ba. “Adam: A method for stochastic optimization”. In: *International Conference for Learning Representations (ICLR)* (2015).
- [KSH12] Alex Krizhevsky, Ilya Sutskever und Geoffrey E. Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in Neural Information Processing Systems*. 2012.
- [LB97] Yann LeCun und Yoshua Bengio. “Convolutional networks for images, speech, and time series”. In: *The handbook of brain theory and neural networks* (1997).
- [LBH15] Yann LeCun, Yoshua Bengio und Geoffrey Hinton. “Deep learning”. In: *nature* (2015).
- [LeC98] Yann LeCun u. a. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* (1998).
- [Li15] Guofa Li u. a. “Lane change maneuver recognition via vehicle state and driver operation signals—Results from naturalistic driving data”. In: *Intelligent Vehicles Symposium (IV)*. 2015.

- [LKF10] Yann LeCun, Koray Kavukcuoglu und Clément Farabet. “Convolutional networks and applications in vision”. In: *International Symposium on Circuits and Systems*. 2010.
- [Mad17] Will Maddern u. a. “1 year, 1000 km: The Oxford RobotCar dataset”. In: *The International Journal of Robotics Research* (2017).
- [Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [MP43] Warren S. McCulloch und Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *Bulletin of Mathematical Biophysics* (1943).
- [MP69] Marvin L. Minski und Seymour A. Papert. *Perceptrons: an introduction to computational geometry*. The MIT Press, 1969.
- [NH10] Vinod Nair und Geoffrey E. Hinton. “Rectified linear units improve restricted Boltzmann machines”. In: *Proceedings of the 27th International Conference on Machine Mearning (ICML-10)*. 2010.
- [Nut18] Anonymer YouTube Nutzer. *Drivers View - Autobahn von Mannheim nach Rostock in Echtzeit 6 Std.* 2018. URL: <https://www.youtube.com/watch?v=uX6xSb6v6Pc> (besucht am 13.11.2018).
- [Ola15] Christopher Olah. *Understanding LSTM networks*. 2015. URL: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> (besucht am 22.11.2018).
- [Oqu14] Maxime Oquab u. a. “Learning and transferring mid-level image representations using convolutional neural networks”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.
- [Pre98] Lutz Prechelt. “Early stopping - but when?” In: *Neural Networks: Tricks of the trade*. 1998.
- [Pun15] Martin Punke u. a. “Kamera-Hardware”. In: *Handbuch Fahrerassistenzsysteme*. 2015.
- [Ric16] Stephan R Richter u. a. “Playing for data: Ground truth from computer games”. In: *European Conference on Computer Vision (ECCV)*. 2016.

- [Ros16] German Ros u. a. “The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
- [Ros58] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological Review* (1958).
- [SB98] Richard S. Sutton und Andrew G. Barto. *Introduction to reinforcement learning*. The MIT Press, 1998.
- [Sch13] Fabian Schuldt u. a. “Effiziente systematische Testgenerierung für Fahrerassistenzsysteme in virtuellen Umgebungen”. In: *AAET 2013*. 2013.
- [SMB10] Dominik Scherer, Andreas Müller und Sven Behnke. “Evaluation of pooling operations in convolutional architectures for object recognition”. In: *Artificial Neural Networks – ICANN 2010*. 2010.
- [Sri14] Nitish Srivastava u. a. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The Journal of Machine Learning Research* (2014).
- [Sun17] Hao Sun u. a. “Robust traffic vehicle lane change maneuver recognition”. In: *SAE Technical Paper*. 2017.
- [Sur18] Sebastian Surmund u. a. “Neue Szenarien für autonome Fahrsysteme”. In: *ATZ extra* (2018).
- [SZ14] Karen Simonyan und Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *Computing Research Repository (CoRR)* (2014).
- [Sze15] Christian Szegedy u. a. “Rethinking the inception architecture for computer vision”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.
- [Tre18] Jonathan Tremblay u. a. “Training deep networks with synthetic data: Bridging the reality gap by domain randomization”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

- [Ulb15] Simon Ulbrich u. a. “Defining and substantiating the terms scene, situation, and scenario for automated driving”. In: *18th IEEE International Conference on Intelligent Transportation Systems*. 2015.
- [WK16] Christopher Woo und Dana Kulić. “Manoeuvre segmentation using smartphone sensors”. In: *Intelligent Vehicles Symposium (IV)*. 2016.
- [WW15] Walther Wachenfeld und Hermann Winner. “Die Freigabe des autonomen Fahrens”. In: *Autonomes Fahren*. 2015.
- [XHK18] Jie Xie, Allaa R Hilal und Dana Kulić. “Driving maneuver classification: A comparison of feature extraction methods”. In: *IEEE Sensors Journal* (2018).
- [Yu18] Fisher Yu u. a. “BDD100K: A diverse driving video database with scalable annotation tooling”. In: *Computing Research Repository (CoRR)* (2018).
- [Yue15] Joe Yue-Hei Ng u. a. “Beyond short snippets: Deep networks for video classification”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.
- [ZH17] Yang Zheng und John H. L. Hansen. “Lane-change detection from steering signal using spectral segmentation and learning-based classification”. In: *IEEE Transactions on Intelligent Vehicles* (2017).
- [Zhe16] Zhixiao Zheng u. a. “Drivers’ lane-changing maneuvers detection in highway”. In: *Man-Machine-Environment System Engineering*. 2016.
- [ZSH14] Yang Zheng, Amardeep Sathyanarayana und John HL Hansen. “Threshold based decision-tree for automatic driving maneuver recognition using CAN-Bus signal”. In: *IEEE 17th International Conference on Intelligent Transportation Systems (ITSC)*. 2014.
- [ZSH15] Yang Zheng, Amardeep Sathyanarayana und John Hansen. “Non-Uniform time window processing of in-vehicle signals for maneuvers recognition and route recovery”. In: *SAE Technical Paper*. 2015.

Appendix

Generierte Daten dieser Arbeit

Unter dem nachfolgenden Link sind alle generierten Daten dieser Arbeit verfügbar:

<https://bit.ly/2A6hMKH>

Im Ordner *Raw Data CarMaker* sind die generierten Signal- und Bilddaten aus der Simulationssoftware CarMaker verfügbar. Im Ordner *Processed Data* sind alle synthetischen und realen annotierten Szenarien als Numpy-Arrays gespeichert. Die trainierten KNNs-Modelle und alle dazugehörigen Ergebnisdaten sind im Ordner *Trained Neural Networks* zu finden.

Python-Code dieser Arbeit

Unter dem nachfolgenden Link ist der gesamte Python-Code, der in dieser Arbeit geschrieben wurde, zu finden:

<https://github.com/manuelhk/master-thesis-python>

Der gesamte Code ist in der Datei *README.md* beschrieben.

