

Inhaltsverzeichnis

Zusammenfassung	iii
Eidesstattliche Erklärung	v
Abkürzungsverzeichnis	viii
Abbildungsverzeichnis	x
Tabellenverzeichnis	xi
1 Einleitung	1
1.1 Problemstellung und Motivation	1
1.2 Zielsetzung	2
2 Grundlagen	3
2.1 Hochautomatisiertes Fahren	3
2.1.1 Entwicklung von Fahrerassistenzfunktionen	4
2.1.2 Klassifizierung von Szenarien	7
2.2 Künstliche Neuronale Netze	14
2.2.1 Einordnung im maschinellen Lernen	14
2.2.2 Entwicklung von künstlichen neuronalen Netzen	16
2.2.3 Convolutional Neural Networks	22
2.2.4 Recurrent Neural Networks und LSTMs	25
2.2.5 Training mit synthetischen Daten	27
2.2.6 Klassifizierung von Videos	29

3 Konzept	33
3.1 Struktur	34
3.2 Ansätze, Methoden, Werkzeuge	36
4 Umsetzung	37
4.1 Definition der Fahrszenarien	37
4.2 Generierung synthetischer Trainingsdaten	40
4.2.1 Simulation mit CarMaker	40
4.2.2 Daten Labeln	44
4.3 Generierung realer Trainings- und Testdaten	46
4.4 Training	50
4.4.1 Vorbereitung der Inputdaten	50
4.4.2 Architektur der künstlichen neuronalen Netze	52
4.4.3 Wahl der Parameter	60
5 Ergebnis	65
5.1 Variation der Parameter	65
5.2 Synthetische und reale Testdaten	69
5.3 Genauigkeit der einzelnen Szenarien	70
6 Zusammenfassung	73
6.1 Ergebnis und Diskussion	74
6.2 Ausblick	74
Literaturverzeichnis	82

Zusammenfassung

Abstract here. test mit Ö, ä und ß ...

Eidesstattliche Erklärung

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, 21. Dezember 2018

Manuel Kaiser

Abkürzungsverzeichnis

Items einrücken
wie andere Ver-
zeichnisse

MiL Model-in-the-Loop

SiL Software-in-the-Loop

HiL Hardware-in-the-Loop

ViL Vehicle-in-the-Loop

FAS Fahrerassistenzsysteme

RF Random Forest

SVM Support Vector Machine

FRC Fuzzy Rule-Based Classifier

kNN k-Nearest-Neighbor

HMM Hidden Markov Model

KNN künstliches neuronales Netz

DNN Deep Neural Network

RNN Recurrent Neural Network

CNN Convolutional Neural Network

LSTM Long Short-Term Memory

tanh Tangens Hyperbolicus

ReLU Rectified Linear Unit

Abbildungsverzeichnis

2.1	Norm SAE J3016 für die Level des autonomen Fahrens, entnommen aus [Com14]	4
2.2	V-Modell [HK15]	5
2.3	Schritte zur Testfallerstellung [Sch13]	6
2.4	Zusammenhang zwischen Szene und Szenario [Ulb15]	8
2.5	Beispiel für ein funktionales, logisches und konkretes Szenario [Bag17]	10
2.6	Beispiel einer Klassifizierung mit zwei Klassen	16
2.7	Modell eines Perzeptrons [Ros58]	17
2.8	Aktivierungsfunktionen für Neuronen	19
2.9	Mehrlagiges Perzeptron mit zwei versteckten Schichten und einer Ausgangsschicht	20
2.10	Beispiel einer Unter- und Überanpassung eines Klassifikators	22
2.11	Beispiel einer Convolution-Operation	23
2.12	Beispiel einer Max-Pooling-Operation	24
2.13	Beispiel eines Convolutional Neural Networks	25
2.14	Beispiel von zwei verschiedenen Recurrent Neural Network (RNN)-Architekturen	26
2.15	Long Short-Term Memory-Zelle [Ola15]	27
2.16	Klassifizierung von einzelnen Bildern mit anschließender Klassifizierung des gesamten Videos	29
2.17	Schematische Darstellung von Klassifizierungsarchitekturen mit frühen, späten und langsamen Fusionen von Feature Maps [Kar14]	30
2.18	Klassifizierung von Videos mit einer Convolutional Neural Network (CNN)-Long Short-Term Memory (LSTM)-Architektur [Don15]	31

3.1	Konzept dieser Arbeit	34
3.2	Schema der funktionalen Dekomposition [AW17]	35
4.1	Konfiguration der simulierten Straße [Gmb18]	43
4.2	Schema der simulierten Strecken 1 und 2 [Gmb18]	43
4.3	Beispiel eines simulierten Szenarios der Klasse <i>lane change left</i> [Gmb18]	47
4.4	Routen für die Aufnahme der realen Bilddaten [Goo18a; Goo18b] . .	48
4.5	Beispiel eines realen Szenarios der Klasse <i>lane change right</i>	50
4.6	Zwei Ansätze für die Videoklassifizierung	53
4.7	Vergleich von vortrainierten Convolutional Neural Networks, entnommen aus [CPC16]	54
4.8	Inception-Module aus der Inception-V3 Architektur [Sze15]	55
4.9	Architektur des Inception-V3 Netzes, entnommen aus [Clo18]	56
4.10	Inception-Modul der Xception-Architektur [Cho17]	57
4.11	Xception-Architektur, entnommen aus [Cho17]	58
4.12	Regularisierung mit Dropout [Sri14]	59
4.13	Grundlegende künstliche neuronale Netz (KNN)-Architekturen in dieser Arbeit	62
5.1	Konfusionsmatrix der Klassifizierung von realen Testdaten mit der Architektur F (Inception-V3, LSTM, Dropout)	67
5.2	Genauigkeit der Trainings- und Testdaten während dem Training von zwei verschiedenen Architekturen	68

Tabellenverzeichnis

2.1	Bisherige Arbeiten zur Szenarienerkennung	13
3.1	Ansätze, Methoden und Werkzeuge dieser Arbeit	36
4.1	Definition der Szenarien <i>free cruising, approaching, following, catching up, overtaking, lane change left</i> und <i>lane change right</i>	40
4.2	Aufgezeichnete Signaldaten in CarMaker	42
4.3	Variablen und Werte die in der Simulation verwendet werden	44
4.4	Bedingungen der Szenarien <i>free cruising, approaching, following, catching up, overtaking, lane change left</i> und <i>lane change right</i>	45
4.5	Anzahl der simulierten Szenarien nach Klasse	46
4.6	Vergleich von synthetischen und realen Datensätzen	49
4.7	Aufteilung von synthetischen und realen Szenarios auf Trainings-, Validierungs- und Testdaten	52
4.8	Parameter die in den Experimenten variiert werden	61
4.9	KNN-Architekturen für die Klassifizierung von Bildersequenzen	63
4.10	KNN-Architekturen für die Klassifizierung einzelner Bilder	64
5.1	Ergebnisse der verschiedenen Architekturen bei der Klassifizierung der realen Testdaten	66
5.2	Ergebnisse der Klassifizierung von synthetischen und realen Testdaten	69
5.3	Genauigkeit der Klassifizierung von realen Testdaten auf der Ebene von Szenarienklassen	70

Kapitel 1

Einleitung

1.1 Problemstellung und Motivation

Hochautomatisierte Fahrerassistenzsysteme werden zunehmend komplexer. Herkömmliche Testmethoden sind durch die Vielzahl an möglichen Szenarien nicht mehr praktisch testbar. Heutzutage wird schon vieles in Simulation getestet. Dabei gibt es aktuell noch Probleme..

Autonomes Fahren - kaum ein Trend ist aktuell ein stärkerer Treiber in der Automobilindustrie. Dabei spielt der Einsatz von Verfahren des maschinellen Lernens eine 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 neuronalen Netz erlernt werden können, gibt es einige 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 soll diese Arbeit ansetzen. Es soll ein Konzept entwickelt und umgesetzt werden, wie eine bereits existierende Simulationsumgebung eingesetzt werden kann, um neuronale Netze zu trainieren und zu testen.

Moreover, in the context of ADAS/AD the interest in using virtual scenarios is already increasing for the task of validating functionalities in the Lab, i.e. to

perform validation in the real world (which is very expensive) only once after extensive and well-designed simulations are passed [Ros16].

1.2 Zielsetzung

Das Ergebnis der Arbeit soll eine Methodik sein bisher unbekannte Testfälle zu finden. Dabei sollen Videodaten mit CarMaker erzeugt und mit diesen Daten ein neuronales Netz trainiert werden um Fahrszenarien zu klassifizieren. Mit einem trainierten neuronalen Netz sollen auch Fahrszenarien mit realen Daten erkannt und klassifiziert werden.

Die oben genannten Probleme sollen mit der Verwendung von simulierten Trainingsdaten adressiert und weiter untersucht werden:

- Trainingsdaten müssen nicht mehr aufwendig manuell annotiert werden.
- Die Umgebung ist bei der Simulation der Daten vollständig kontrollierbar und die Datenerfassung wird effizienter.
- Bisher unbekannte Testfälle können gefunden werden.

Diese Arbeit soll einen theoretischen und praktischen Beitrag zum automatisierten Training von neuronalen Netzen im Bereich automatisiertem Fahren liefern. Der Fokus liegt dabei auf den Möglichkeiten und Herausforderungen, die sich durch die Verwendung von simulierten Trainingsdaten ergeben.

Kapitel 2

Grundlagen

In diesem Kapitel werden die zum Verständnis nötigen Grundlagen für diese Arbeit erklärt. Dabei wird im 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 für die Umsetzung dieser Arbeit nötig sind, beschrieben.

2.1 Hochautomatisiertes Fahren

Hochautomatisiertes Fahren wird in den vergangenen Jahren zunehmend von der Automobilindustrie vorangetrieben. Aktuelle Fahrerassistenzsysteme (FAS) wie der Spurhalteassistent oder die Abstandsregelung sind nach der Norm SAE J3016 (Abbildung 2.1) bei Level 2 des autonomen Fahrens eingeordnet. Mit neuen Technologien werden immer mehr Funktionen für automatisiertes Fahren entwickelt und verknüpft. Es entstehen zunehmend komplexe Fahrfunktionen mit einer steigenden Anzahl möglicher Fahrsituationen und Szenarien [Kin17]. Das stellt Automobilhersteller und Automobilzulieferer vor eine große Herausforderung, da die Systemkomplexität wächst. Das schließt sowohl die Entwicklung von FAS als auch die dazu benötigten Testszenarien ein [PL16].

In den folgenden Abschnitten wird erläutert wie aktuell diesen Herausforderungen begegnet wird. In Abschnitt 2.1.1 wird ein allgemeiner Überblick über die aktuellen Entwicklungsmethodiken für FAS gegeben. Danach werden in Abschnitt

Zahlen, Daten
Fakten

erläutern

erläutern, warum
die Herausforderung
insbesondere ab
Level 3 existiert

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 Level des autonomen Fahrens, entnommen aus [Com14]

2.1.1 Entwicklung von Fahrerassistenzfunktionen

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.

pauschalisiert

FAS werden in der Automobilindustrie mit dem V-Modell entwickelt. Das V-Modell ist ein chronologischer Entwicklungsprozess 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.

2.1. Hochautomatisiertes Fahren

Der rechte Ast umfasst aufsteigend Funktionstests auf dem jeweiligen Detaillierungsgrad [HK15]. Das V-Modell ist in einer einfachen Version in Abbildung 2.2 dargestellt.

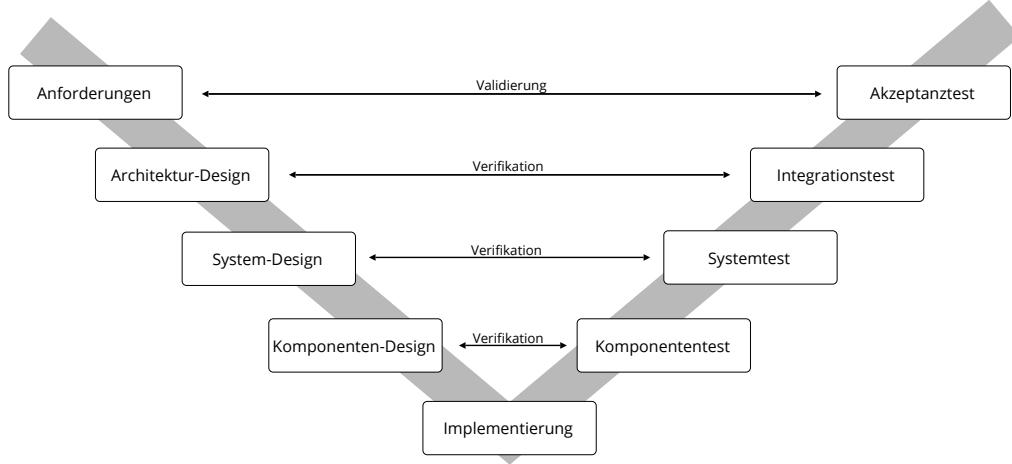


Abbildung 2.2: V-Modell [HK15]

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, und 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 der Abbildung 2.3 abgebildet.

Testfälle werden bereits möglichst früh im Entwicklungsprozess erstellt um die Qualität von FAS und einzelnen Komponenten möglichst hoch zu halten [WW15]. Hierfür werden in der Praxis virtuelle Fahrversuche eingesetzt. Die Idee ist eine

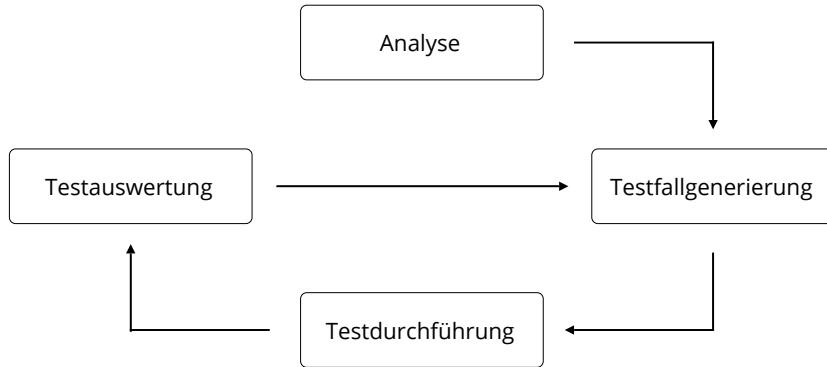


Abbildung 2.3: Schritte zur Testfallerstellung [Sch13]

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 realem 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. Dieses Testkonzept macht die Komplexität bei der Entwicklung von FAS beherrschbar und reduziert den Testaufwand [Sch14]. 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 auch die Anzahl möglicher Situationen in denen die Funktionen ohne Fahrer ablaufen müssen. Um alle Funktionen ausreichend testen zu können, steigt die Anzahl der benötigten Testfälle. Testfälle müssen alle potentiell möglichen Situationen, in denen das FAS zum Ein-

2.1. Hochautomatisiertes Fahren

satz kommen kann, abdecken. Dadurch steigt mit hochautomatisierten Funktionen der Aufwand für Validierung und Verifikation mit Testfällen [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 Szenarienkatalogen abgeleitet [Püt17]. Diese Kataloge enthalten alle bekannten Szenarien in denen sich ein Fahrzeug befinden kann. Sie sind jedoch vor dem Hintergrund erstellt worden, dass zu jeder Zeit ein Fahrer das Kraftfahrzeug überwacht und steuert [WW15]. Bei neuen hochautomatisierten FAS für Stufe 3 und 4 des autonomen Fahrens, ist dies nicht mehr gegeben. Die Sicherheit des Gesamtsystems muss in breiterem Spektrum mit einer Vielzahl hochkomplexer Szenarien ohne Eingriff des Fahrers garantiert werden können. Da bedeutet, dass die bisherigen Szenarienkataloge um neue Szenarien erweitert werden müssen [Sur18].

wo kommen die Szenarienkataloge her? Aus den Anforderungen?

Für die Erstellung von Testfällen müssen daher alle potentiell möglichen kritischen Szenarien bekannt sein. Der Ansatz in 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, die 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 mehrerer aufeinander folgenden Szenen und beschreibt diese zeitliche Entwicklung. Handlungen, Ereignisse, Ziele und Werte können für eine Charakterisierung der zeitlichen Entwicklung eines Szenarios spezifiziert werden. Anders als eine Szene, umfasst ein Szenario einen definierten Zeitraum.

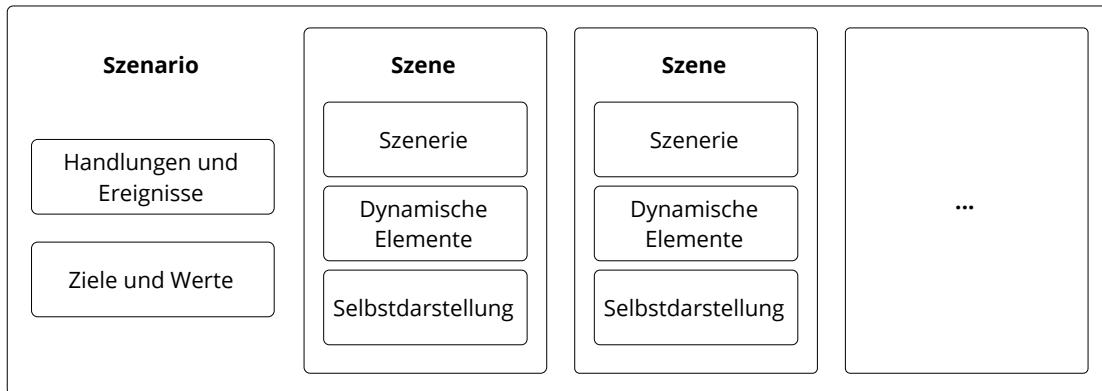


Abbildung 2.4: Zusammenhang zwischen Szene und Szenario [Ulb15]

Neben den Begriffen Szene, Situation und Szenario wird auch oft der Begriff Manöver verwendet. Bach et al. [BOS16] definieren ein Manöver als einen Status innerhalb eines Szenarios. Dabei sind Situation jeweils Übergangsbedingungen zwischen einzelnen Manövern. So setzt sich beispielsweise das Szenario *Überho-*

2.1. Hochautomatisiertes Fahren

len 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 Szenarios 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.

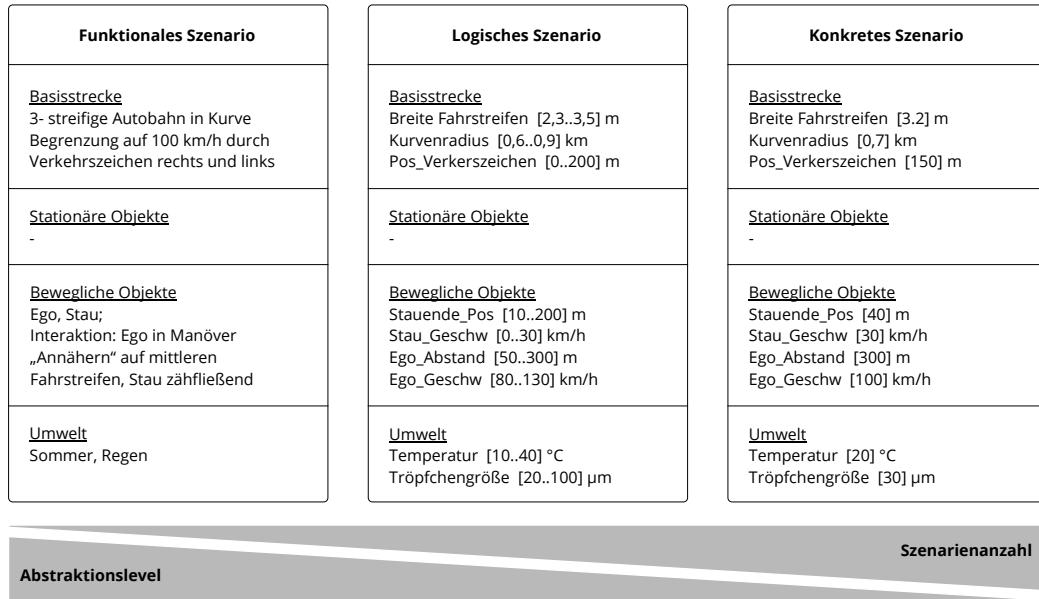


Abbildung 2.5: Beispiel für ein funktionales, logisches und konkretes Szenario [Bag17]

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 Methoden zur Klassifizierung von Szenarien veröffentlicht. In den folgenden Absätzen werden die relevanten Arbeiten (Anzahl 12) seit 2014 kurz vorgestellt.

Für die Klassifizierung wurden verschiedene Sensordaten aus dem Fahrzeug verwendet. Die Autoren von fünf Arbeiten haben ein Smartphone im Fahrzeug platziert und Beschleunigungs-, Gyroskop-, GPS- und Magnetometer-Daten für die Klassifizierung ausgelesen und verwendet [XHK18; Cer16; WK16; CHK16; ABR16]. Die Verwendung von Smartphone-Daten wurde mit der einfachen und kostengünstigen Umsetzung begründet. Vier andere Arbeiten verwendeten Sensor-daten wie *Lenkwinkel*, *Fahrzeuggeschwindigkeit*, *laterale Geschwindigkeit*, *Giergeschwindigkeit* und die *Position des Gas- und Bremspedals*, die sie aus dem CAN-Bus des Fahrzeugs ausgelesen haben [ZH17; ZSH15; Li15; ZSH14]. Drei weitere Arbeiten basierten ihre Experimente auf Daten aus einem Fahrsimulator [Sun17;

2.1. Hochautomatisiertes Fahren

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], CNN [Gru17], Decision Tree [ZSH14] und Naive Bayes [CHK16] verwendet. Da es für diese Arbeit nicht relevant ist, werden die Methoden 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 das im Fahrzeug platziert ist	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
[Sun17]	Lateraler Abstand zwischen Fahrzeug und Fahrbahnmarkierung von einem Fahrsimulator	SVM, Bayesian Inference Model	Spurwechsel nach links, Spurwechsel nach rechts, Spur halten

Quelle	Sensordaten	Klassifikator	Szenarien
[Gru17]	Ort, Ausrichtung und Geschwindigkeit des Ego-Fahrzeugs und benachbarten Objekten von realen Testfahrten	CNN auf Basis von gestapelten Positionsgittern der Objekte	Freifahren, anderes Fahrzeug voraus, anderes Fahrzeug überholt Ego-Fahrzeug, Querverkehr vor Ego-Fahrzeug
[Cer16]	Beschleunigung von einem Smartphone das im Fahrzeug platziert ist	RF, SVM, FRC	Einparken, geparkt, frei fahren, stoppen
[WK16]	Beschleunigung, Gyroskop, GPS und Magnetometer von einem Smartphone das im Fahrzeug platziert ist	SVM	Stoppen, beschleunigen, bremsen, links abbiegen, rechts abbiegen
[CHK16]	Beschleunigung, Gyroskop und GPS von einem Smartphone das im Fahrzeug platziert ist	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
[ABR16]	Beschleunigung, Gyroskop und GPS von einem Smartphone das im Fahrzeug platziert ist	FRC	Lenken, beschleunigen, bremsen, Bodenwelle

2.1. Hochautomatisiertes Fahren

Quelle	Sensordaten	Klassifikator	Szenarien
[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 wurde 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. Und in seiner Arbeit verwendet er keine Kamerabildern, sondern mit gespaltenen Matritzen, auf denen jeweils die Positionen aller Verkehrsteilnehmer markiert sind. Nach Grunder [Gru17] wird sich die zukünftige Forschung mit tieferen Netzstrukturen wie RNNs beschäftigen, um zeitabhängige Szenarien noch besser zu verstehen.

Mit den bisherigen Ansätzen können bekannte Szenarien gut klassifiziert werden. Unbekannte bzw. neue Szenarien werden allerdings nur sehr bedingt erkannt, weil die Datengrundlage für die 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.

Inwieweit erkennen wir wirklich „neue“ Szenarien mit dem Ansatz dieser Arbeit?

Im Gegensatz zu den bisherigen Arbeiten, soll in dieser Arbeit die Verwendung von Kameradaten für die Klassifizierung von Szenarien untersucht werden. Als Klassifikator soll ein bild- und zeitsensitives DNN verwendet werden. Damit sollen auch bisher unbekannte Einflüsse, die auf den Bildern zu sehen sind, für die Klassifizierung berücksichtigt werden. Der Ansatz wird im Detail in Kapitel 3 erklärt.

2.2 Künstliche Neuronale Netze

In diesem Kapitel werden KNNs mit einem Schwerpunkt auf Bilderkennung mit CNNs und Sequenzerkennung mit 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. Dann 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.“

2.2. Künstliche Neuronale Netze

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 der 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 selbstständige Erlernen des Brettspiels Schach.

Klassifizierung

In dieser Arbeit wird ein Konzept für die Klassifizierung von Fahrszenarien - damit in der Kategorie überwachtes Lernen - 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 ein Datensatz mit zwei Klassen dargestellt. Die Objekte im

Datensatz haben jeweils die Eigenschaften x_1 und x_2 .

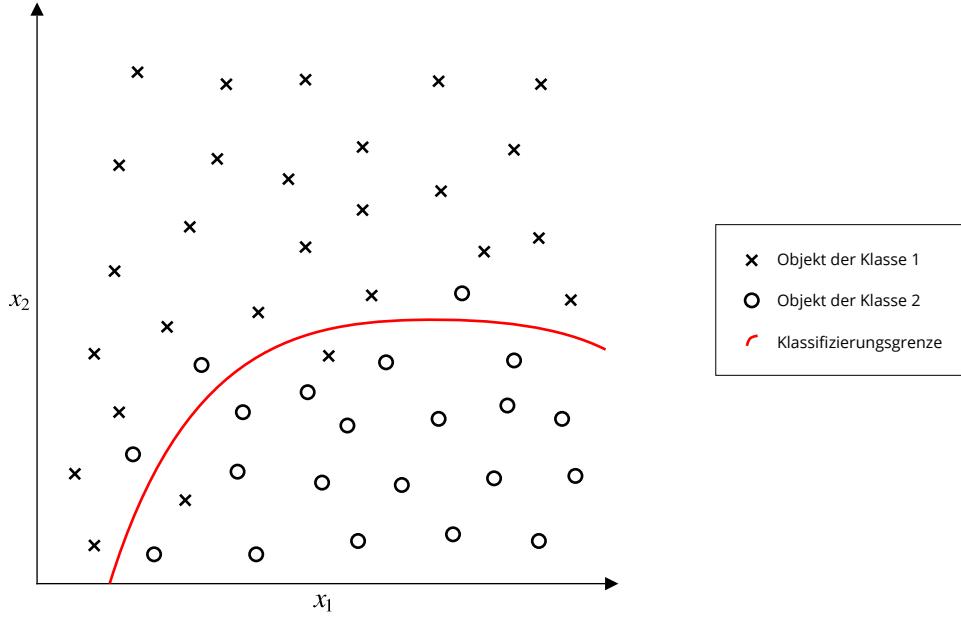


Abbildung 2.6: Beispiel einer Klassifizierung mit zwei Klassen

In den folgenden Abschnitten werden schrittweise CNNs und LSTMs eingeführt mit denen in dieser Arbeit ein Klassifikator für die Erkennung von Fahrszenarien entwickelt und trainiert wird.

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.

Um den Aktivierungswert $o(\vec{x})$ zu berechnen wird zunächst die gewichtete Summe mit dem Eingangsvektor \vec{x} und den Gewichten \vec{w} gebildet. Dann wird mit der Aktivierungsfunktion φ eine Klasse bestimmt. Beim ursprünglichen Perzeptron

2.2. Künstliche Neuronale Netze

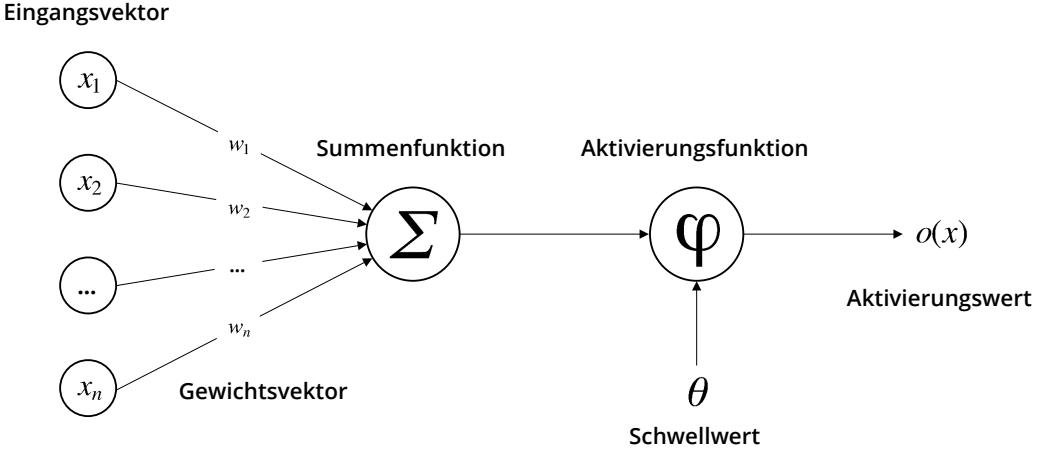


Abbildung 2.7: Modell eines Perzeptrons [Ros58]

handelt es sich dabei um eine Schwellenwertfunktion, die 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 bestimmt und 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 $1e - 2$ und $1e - 4$ 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 Schwellenwertfunktion, in Abbildung 2.8 dargestellt.

Mit diesen Aktivierungsfunktionen und der Verwendung mehrerer Perzeptronen werden mehrschichtige KNNs, oder auch mehrlagiges Perzeptron (engl. multi-layer perceptron), modelliert. Dadurch können diese 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, dem Ergebnisvektor oder den Neuronen der vorherigen und

2.2. Künstliche Neuronale Netze

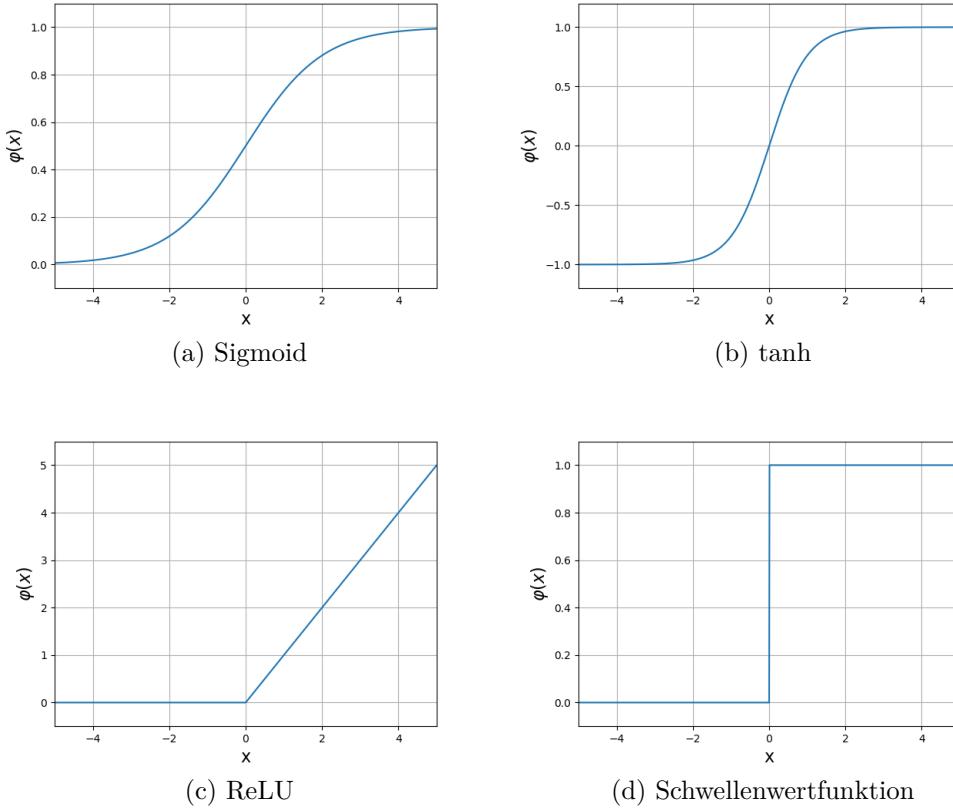


Abbildung 2.8: Aktivierungsfunktionen für Neuronen

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 KNN.

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

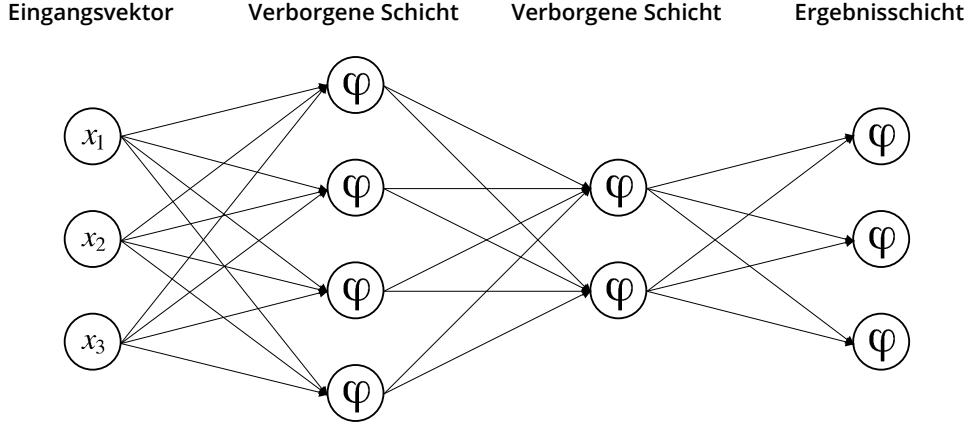


Abbildung 2.9: Mehrlagiges Perzeptron mit zwei versteckten Schichten und einer Ausgangsschicht

dem verborgenen Neuron j . Alle weiteren verborgenen Schicht und der Ergebnisvektor werden auf die gleiche Weise berechnet mit den Aktivierungswerten der vorherigen Schicht als Eingangsvektor. 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 Klassen 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, die die Abweichung zwischen der berechneten und der tat-

2.2. Künstliche Neuronale Netze

sächlichen Klasse beschreibt, zu definieren und dann 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 alle Aktivierungswerte \vec{y}_j aller versteckter 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.
3. Auf Basis dieser Differenz werden die Gewichte zwischen allen Neuronen geändert, mit dem Ziel diese 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 Training 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. Dabei spricht man auch von Generalisierbarkeit. Wenn ein Modell dagegen 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.

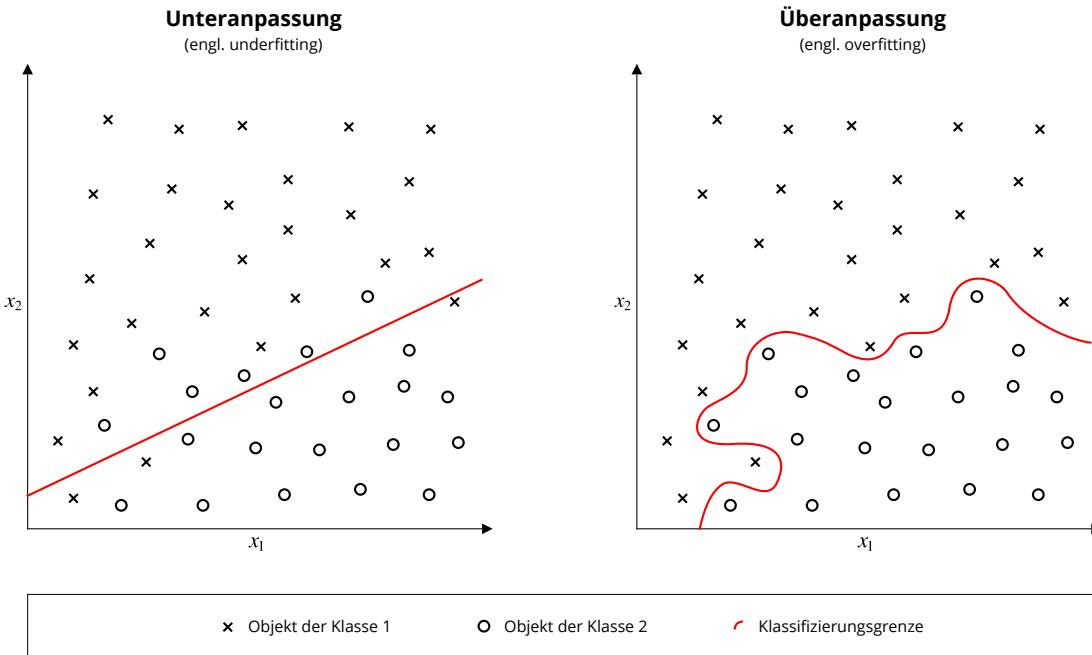


Abbildung 2.10: Beispiel einer Unter- und Überanpassung eines Klassifikators

Heute werden in Anwendungen häufig tiefe 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-Schicht und einer Pooling-Schicht [LKF10]. Diese Abfolge kann sich beliebig oft

2.2. Künstliche Neuronale Netze

wiederholen und wird am Ende mit einer oder mehreren Fully-Connected-Schicht 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 Idee dieser Schicht ist es Merkmale aus einem Bild (oder anderen Inputdaten) zu extrahieren. Dabei werden in den ersten Schichten Merkmalen 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.

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

In diesem Beispiel handelt es sich um ein Bild mit den Dimensionen 6x6x1 Pixel, also einem zweidimensionalen Bild mit einem Farbkanal. Der Filter hat die Dimensionen 3x3x1. In dem Beispiel ist die aktuell dargestellte Rechnung wie 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 Bilder angekommen,

wir der Filter eine Schrittänge weiter nach unten gesetzt und wieder am 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 eingesetzt 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 eine 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 der Feature Map zu reduzieren und dabei die wichtigsten Merkmale beizubehalten [SMB10]. Wie bei der Convolution-Operation, gibt es auch beim Pooling verschiedene Operationen (e.g. Average-Pooling). Es können auch die Größe des Pooling-Fensters und die Schrittgröße variiert werden. In Abbildung 2.12 ist die oft verwendete Max-Pooling-Operation dargestellt.

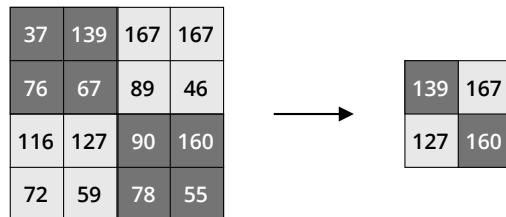


Abbildung 2.12: Beispiel einer Max-Pooling-Operation

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.

2.2. Künstliche Neuronale Netze

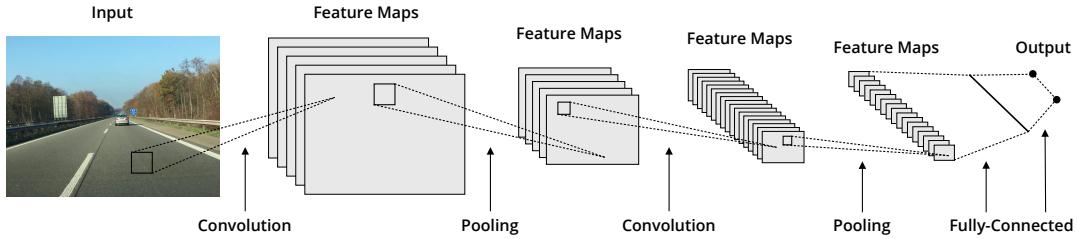


Abbildung 2.13: Beispiel eines Convolutional Neural Networks

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 besonders für sequentielle Daten geeignet ist. Der Unterschied zu der Architektur eines mehrlagigen Perzeptrons ist, dass die Neuronen einer versteckten Schicht bei einem RNN miteinander und mit vorherigen Schichten verbunden sind [Gra12]. Damit entsteht nicht nur eine Abhängigkeit zu den Aktivierungswerten der vorherigen Schicht, sondern auch zu den Werten von anderen Neuronen derselben Schicht. Das bedeutet wiederum, dass das RNN nicht von einem einzigen Eingangsvektor abhängig ist, sondern von einer Sequenz von Eingangsvektoren.

Es gibt verschiedene Architekturen von RNNs, e.g. 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, eine Übersetzung, produziert wird, und das auch Einfluss auf die Bedeutung der folgenden Wörter hat. Architekturen mit einem Ergebniswert werden für Klassifizierungsprobleme verwendet, in denen einer Sequenz von Inputdaten eine Klasse zugeordnet werden soll, beispielsweise eine Videoklassifizierung wie in dieser Arbeit. Diese beiden Varianten sind beispielhaft 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.

Wenn nun in einem RNN die Fehlerrückführung (engl. error backpropagation)

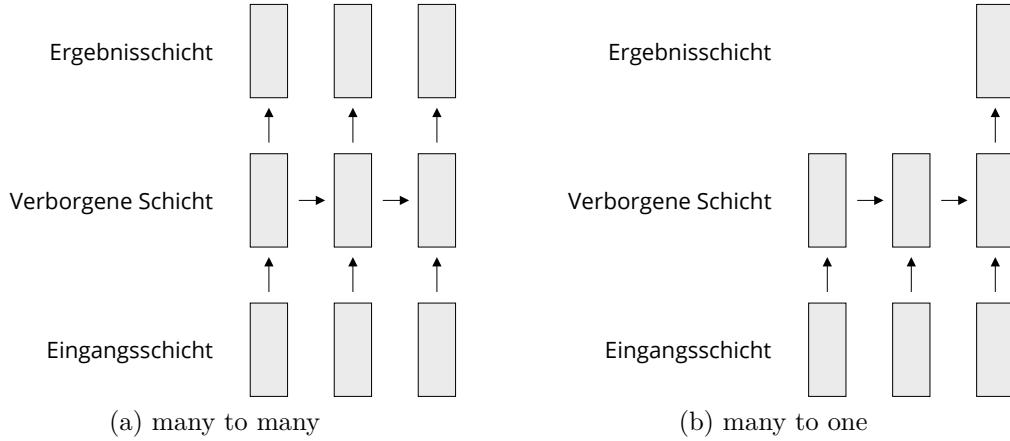


Abbildung 2.14: Beispiel von zwei verschiedenen RNN-Architekturen

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 schnell (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 Training, 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 Vergessstor (engl. forget gate) und dem Ausgangstor (engl. output gate), und einem Zellenzustand aufgebaut. Die Idee ist, dass die LSTM-Zelle ein Kurzzeitgedächtnis an frühere Werte hat, das lange hält (engl. Long Short-Term Memory). Die Abbildung 2.15 zeigt eine solche LSTM-Zelle.

Das Vergessstor 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].

2.2. Künstliche Neuronale Netze

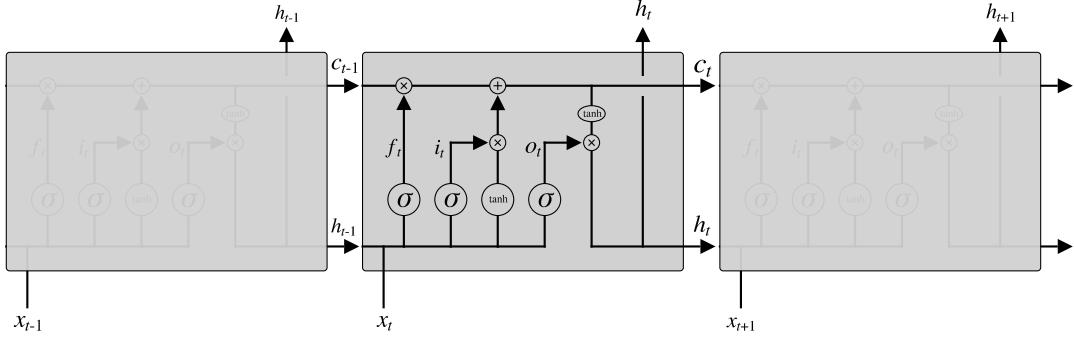


Abbildung 2.15: Long Short-Term Memory-Zelle [Ola15]

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

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

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

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

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

2.2.5 Training mit synthetischen Daten

Ein großes Problem beim Training von neuronalen Netzen ist der Aufwand um die Trainings- und Validierungsdaten zu labeln [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. Diese 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 Pixeln 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 GTA5 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) als Label 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 konnte auch sie die Genauigkeit verbessern.

Tremblay et al. [Tre18] verfolgten bei der Generierung von Bildern einen etwas anderen Ansatz. Sie nutzen 3D Modelle von Fahrzeugen und platzierten diese mit zufälligen Positionen und Ausrichtungen auf verschiedene reale Szenen. Die 100.000 Bilder wurden mit Begrenzungsboxen (engl. bounding boxes) um Fahrzeuge gelabelt. Beim Training wurden ausschließlich die synthetischen Daten verwendet und es wurde eine Genauigkeit von circa 80% erreicht.

Insgesamt 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 gelabelt 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 einzeln 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.

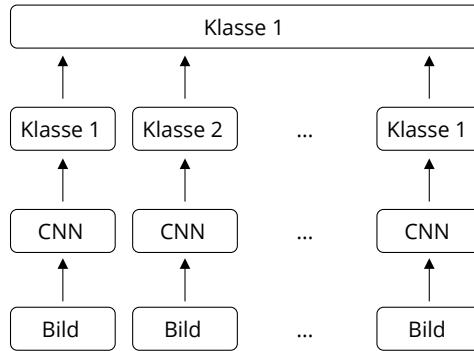


Abbildung 2.16: Klassifizierung von einzelnen Bildern mit anschließender Klassifizierung des gesamten Videos

Ein weiterer Ansatz basiert auf der Idee Feature Maps von mehreren Bildern zu kombinieren und damit die zeitlichen Merkmalen mit 3D-Filtern eines CNNs zu extrahieren. In frühen Arbeiten dazu wurde für diese Kombination eine Pooling-Operation verwendet [Kar14; Yue15] und es wurden verschiedene Architekturen mit frühen, späten oder langsamen Fusion verwendet (engl. early, late, and slow fusion). Bei der frühen Fusion werden schon zu Beginn die Eingangswerte von mehreren Bildern kombiniert und dann zusammen die Feature Maps berechnet. Bei der späten Fusion werden von allen Bildern einzeln die Merkmale extrahiert und erst am Ende mit einer Pooling-Schicht vereint. Beim Ansatz der langsamen Fusion werden schrittweise die Feature Maps von immer mehr Bildern zusammengefasst. Diese Architekturen sind schematisch in Abbildung 2.17 dargestellt.

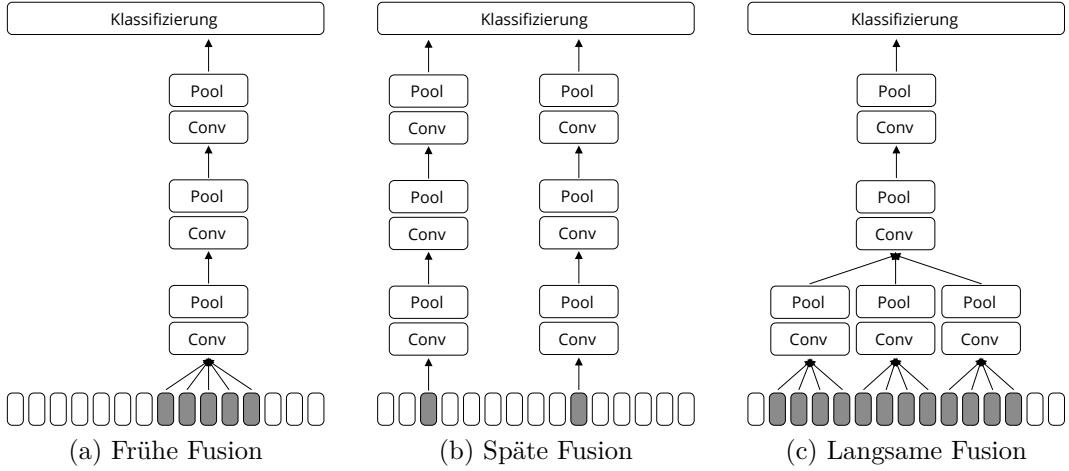


Abbildung 2.17: Schematische Darstellung von Klassifizierungsarchitekturen mit frühen, späten und langsamen Fusionen von Feature Maps [Kar14]

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, die die zeitlichen Merkmale extrahiert. In Abbildung 2.18 ist diese Architektur schematisch dargestellt.

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.

2.2. Künstliche Neuronale Netze

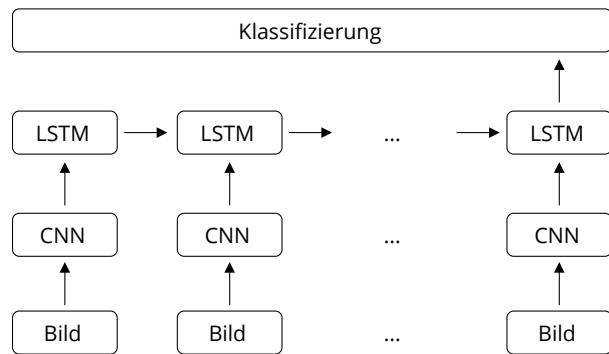


Abbildung 2.18: Klassifizierung von Videos mit einer CNN-LSTM-Architektur [Don15]

Kapitel 3

Konzept

Wie in Abschnitt 2.1 beschrieben, stellt das Testen 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 leisten. 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 (circa 95%) und einem kleinen Anteil realer Daten (circa 5%) von bisher bekannten Szenarien zu trainieren. Es wird mit einem großen Teil synthetischer Daten gearbeitet, weil es in der Praxis um ein Vielfaches einfacher ist synthetische Daten zu generieren und automatisch zu Labeln als große Mengen realer Daten zu Labeln. Der trainierte Klassifikator kann dann bekannte Szenarien sehr gut klassifizieren. 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 Vorgehensweise vorgestellt. Anschließend wird in Abschnitt 3.2 die Methodik erklärt mit welcher dieses Konzept umgesetzt werden soll.

3.1 Struktur

Die Umsetzung in dieser Arbeit lässt sich in drei Teile untergliedern. Im ersten Teil werden die zu klassifizierenden Szenarien als *logische Szenarien* definiert. Auf der Basis werden im zweiten Teil synthetische und reale Trainingsdaten generiert. Im dritten Teil wird schließlich ein KNN als Klassifikator trainiert und evaluiert. Diese Struktur ist schematisch in Abbildung 3.1 abgebildet und wird in den folgenden Absätzen weiter beschrieben.

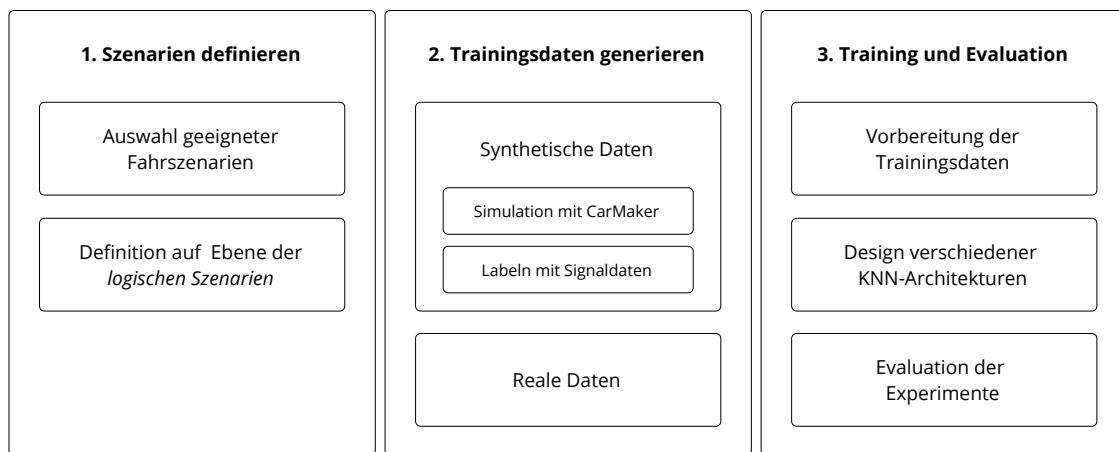


Abbildung 3.1: Konzept dieser Arbeit

Im ersten Schritt der Umsetzung werden bestimmte 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 der Simulationssoftware CarMaker gearbeitet. Mit dieser Software können das Ego-Fahrzeug, Straßen, Verkehr und die Trajektorien aller Fahrzeuge generiert und beliebig verändert werden.

3.1. Struktur

Die Idee ist es, Fahrten des Ego-Fahrzeugs zu simulieren, entsprechende Bild- und Signaldaten aufzuzeichnen und die Bilddaten anhand der Signaldaten zu labeln. Auf diese Weise können ohne großen Aufwand beliebig viele synthetische Daten erzeugt und gelabelt werden. Mit CarMaker können sowohl Rohdaten, wie zum Beispiel Radarsignale des Ego-Fahrzeugs, als auch abstrakte Informationen, wie die Position und Geschwindigkeit von anderen Objekten, simuliert 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 das Labeln 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 klassifiziert. Die Aneinanderreihung von mehreren Szenen ergibt schließlich ein Szenario. Für die Generierung von realen Trainingsdaten werden Videosequenzen verwendet und manuell gelabelt.

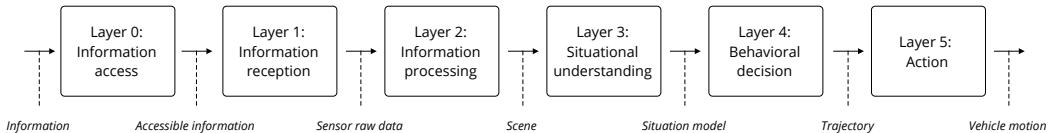


Abbildung 3.2: Schema der funktionalen Dekomposition [AW17]

Für das Training und die Evaluation eines Klassifikators werden KNNs verwendet. Wie Abschnitt 2.2.6 gibt es verschiedene Ansätze einen Klassifikator für Videos zu trainieren. In dieser Arbeit werden zwei verschiedene Ansätze angewendet und miteinander verglichen. Beim ersten Ansatz werden CNNs verwendet um einzelne Bilder zu klassifizieren. CNNs sind nach dem Stand der Technik die besten Architekturen um Merkmale aus Bildern zu extrahieren. Im Anschluss werden Videos mit dem Szenario klassifiziert, mit dem die meisten Bilder im Video klassifiziert wurden. Mit dem zweiten Ansatz wird ein Klassifikator aus einer Kombination von CNNs und LSTMs erstellt. Damit können räumlichen und zeitlichen

Merkmale aus ganzen Videos extrahiert und diese dann auf der Basis klassifiziert werden. 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.

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 Labeln 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, manuelles Labeln	[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

Umsetzung

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 synthetische Trainingsdaten und reale Trainings- und Testdaten erläutert. Im Anschluss wird in Abschnitt 4.4 die Architektur und die Experimente des Klassifikators 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 zu 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 Parameter 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 [Sonnenaufgang..Sonnenuntergang] Bewölkung [leicht bewölkt..wolkenlos]	
Free cruising	Ego, andere 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	

4.1. Definition der Fahrszenarien

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

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

4.2 Generierung synthetischer Trainingsdaten

Auf Basis der Definitionen aus dem vorherigen Abschnitt 4.1 werden in diesem Abschnitt die benötigten Signal- und Bilddaten simuliert und entsprechend gelabelt. 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 labeln.

4.2.1 Simulation mit CarMaker

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

4.2. Generierung synthetischer Trainingsdaten

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 und hat eine Auflösung von 640 x 480 Pixeln und ein Sichtfeld von 20°.

Die benötigten Signaldaten für das Labeln 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 konfigurierten Radius alle Fahrzeuge und ihr Abstand und ihre relative Geschwindigkeit zum Ego-Fahrzeug erfasst und über die *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 *OutputQuantities* aufgezeichnet werden. Die jeweiligen Variablen in CarMaker sind in der Tabelle 4.2 zusammengefasst.

Für die Simulation werden zwei Strecken der Länge 6.000m und 10.000m 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 auch einige Bäume, was in Abbildung 4.2 mit grünen Streifen gekennzeichnet ist. Abbildung 4.1 zeigt die Konfiguration der simulierten Straße.

Auf beiden Strecken wird autonomer, stochastisch verteilter Verkehr erzeugt, was CarMaker mit einer gesonderten Funktion unterstützt. Der Verkehr wird in einer niedrigen Dichte (10%) und einem 80%-igen 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

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

auf der 10.000m-Strecke 131 Fahrzeuge und auf der 6.000m-Strecke 89 Fahrzeuge generiert.

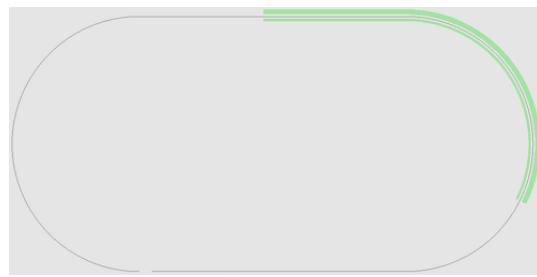
Die Simulation und Generierung von Bild- und Signaldaten wird mit dem *CarMaker - Test Manager* durchgeführt. Mit diesem Modul lassen sich Fahrten mit unterschiedlichen Konfigurationen simulieren. In dieser Arbeit werden die Variablen *Geschwindigkeit*, *Mindestabstand zu vorausfahrendem Fahrzeug*, *Minimale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

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 markiert die 0 ein 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.

4.2. Generierung synthetischer Trainingsdaten



Abbildung 4.1: Konfiguration der simulierten Straße [Gmb18]



(a) Strecke 1



(b) Strecke 2

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

Mit diesen vier Variablen mit jeweils drei bzw. 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 (Bilder und Signaldaten) resultiert. Diese Szenen werden im folgenden Abschnitt 4.2.2 gelabelt.

Variable	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

4.2.2 Daten Labeln

Für das Labeln 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 aufgelistet. Auf Basis der CarMaker-Variablen aus Tabelle 4.3 werden folgende zusätzliche Variablen definiert, um nachfolgende Bedingungen übersichtlicher darzustellen. 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	$ego_v > 17$ $s_0 < v2_{ds}$

4.2. Generierung synthetischer Trainingsdaten

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	$v2_{dv} < 0$ $0 <= v2_{ds} < s_0$ $ego_{sRoad} \leq v2_{sRoad}$ $ego_{laneID} = v2_{laneID} - 1$
Overtaking	$0 \leq v2_{ds} < s_0$ $v2_{sRoad} < ego_{sRoad}$ $ego_{laneID} = v2_{laneID} - 1$
Lane change left	$ego_{laneID}^{before} = ego_{laneID}^{after} + 1$ Als Spurwechsel wird ein Intervall von 4 Sekunden betrachtet in dessen Mitte die Variable ihren Wert wechseln muss
Lane change right	$ego_{laneID}^{before} = ego_{laneID}^{after} - 1$ Als Spurwechsel wird ein Intervall von 4 Sekunden betrachtet in dessen Mitte die Variable ihren Wert wechseln muss

Tabelle 4.4: Bedingungen der 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 dem gleichen Label klassifiziert wurden. Drei Sekunden wird 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. Zu manchen Zeitpunkten werden mehr als eine einzelne Szene klassifiziert. 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 Trainings- und Testdaten

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

4.3. Generierung realer Trainings- und Testdaten

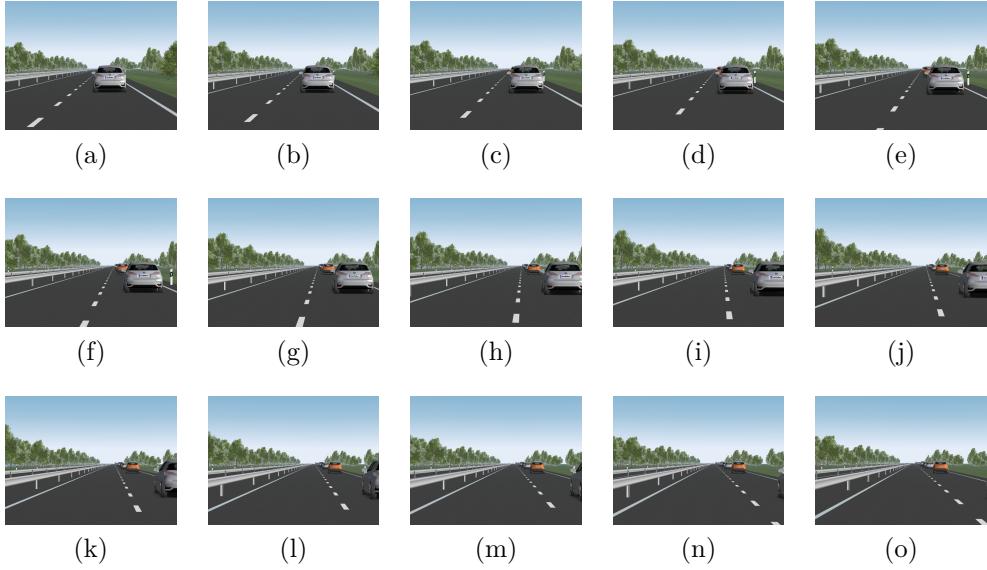


Abbildung 4.3: Beispiel eines simulierten Szenarios der Klasse *lane change left* [Gmb18]

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 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 gelabelt. 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.

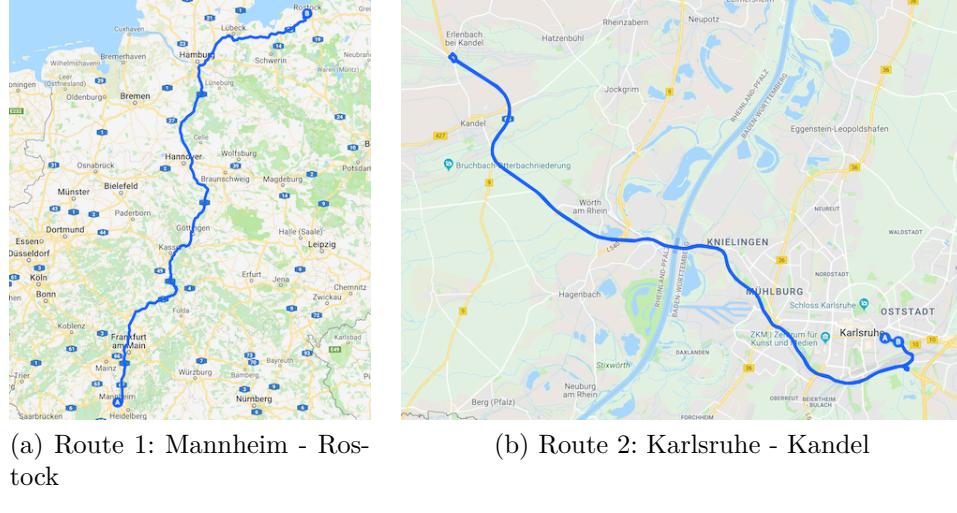


Abbildung 4.4: Routen für die Aufnahme der realen Bilddaten [Goo18a; Goo18b]

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 dieser Datensatz, mit synthetischen und realen Daten, mit anderen synthetischen Datensätzen, die in Abschnitt 2.2.5 vorgestellt wurden, und realen Datensätzen, die in diesem Abschnitt vorgestellt werden, verglichen.

Datensatz	Generierung	Umfang	Label
[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

4.3. Generierung realer Trainings- und Testdaten

Datensatz	Generierung	Umfang	Label
[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 Label
Diese Arbeit	synthetisch mit CarMaker und real mit zwei Autobahnfahrten in Deutschland	24.307 Szenarien (3 Sekunden) mit insgesamt 331.133 Bildern	7 Szenario-Klassen

Tabelle 4.6: Vergleich von synthetischen und realen Datensätzen

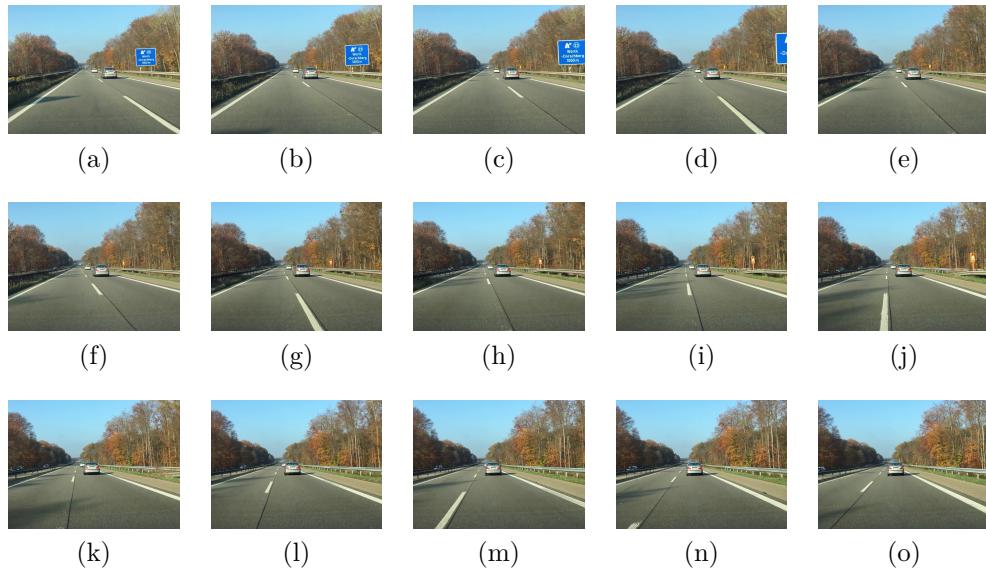


Abbildung 4.5: Beispiel eines realen Szenarios der Klasse *lane change right*

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 Inputdaten

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

In dieser Arbeit werden CNNs für die Erkennung einzelner Bilder und Kombinationen aus CNNs und LSTM für die Klassifizierung von Videos eingesetzt. Daher müssen sowohl einzelne Bilder, als auch Bildsequenzen in das jeweilige

4.4. Training

KNN importiert werden. Da es sich um große Datenmengen von insgesamt über 50GB 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 Klasse, sogenannte *Data-Generators*, 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 ?? adaptiert um höher-dimensionale Datensets 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 oft Überscheidungen mit 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% synthetische Daten und 5% reale Daten für das Training verwendet werden. Mit dieser Verteilung bleibt der manuelle Aufwand für das Labeln 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 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 Architektur der künstlichen neuronalen Netze

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 Bildersequenzen einzeln klassifiziert. Anschließend wird den Sequenzen die Klasse zugeordnet, mit der die meisten Bilder aus der jeweiligen Sequenz klassifiziert wurden (Mehrheitsprinzip). Für den zweiten Ansatz wird eine Kombination aus Bild- und Sequenzerkennung verwendet um die Bildersequenzen 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-, Dropout- und LSTM-Schichten mit den verwendeten Parametern vorgestellt. Am Ende dieses

4.4. Training

Abschnitts werden alle Komponenten zusammengefügt und acht Architekturen vorgestellt, die für die Experiment in Abschnitt 4.4.3 verwendet werden.

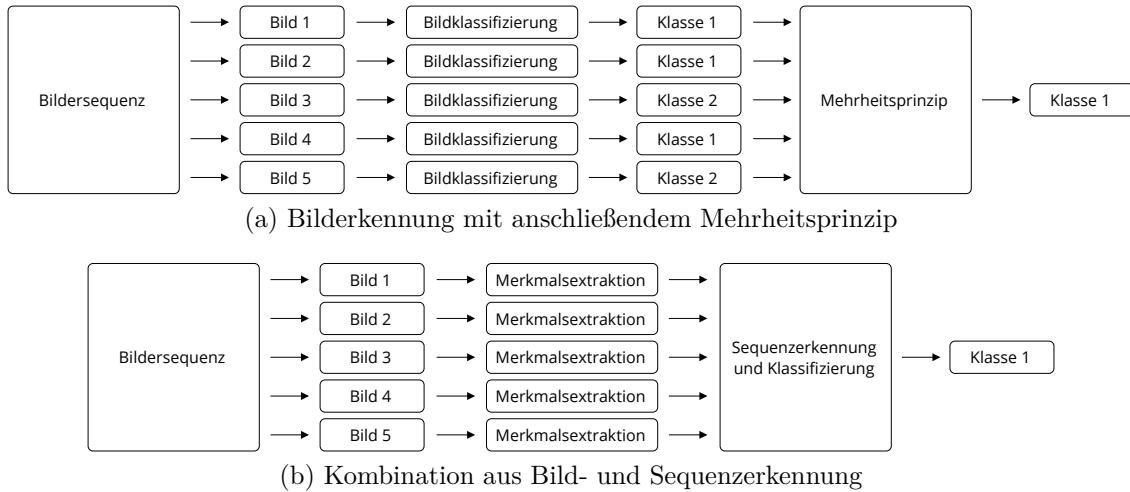


Abbildung 4.6: Zwei Ansätze für die Videoklassifizierung

Convolutional Neural Networks

Das Training mit CNNs hat in den vergangenen Jahren große Fortschritte gemacht durch die Verfügbarkeit von großen Datenmengen, steigender Rechenleistung und die Wiederverwendung von bereits vortrainierten CNNs. Diese Verwendung von bereits trainierten Netzen nennt sich Transferlernen (engl. transfer learning) [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 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 grundlegend 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. Dabei 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 CNNs-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.

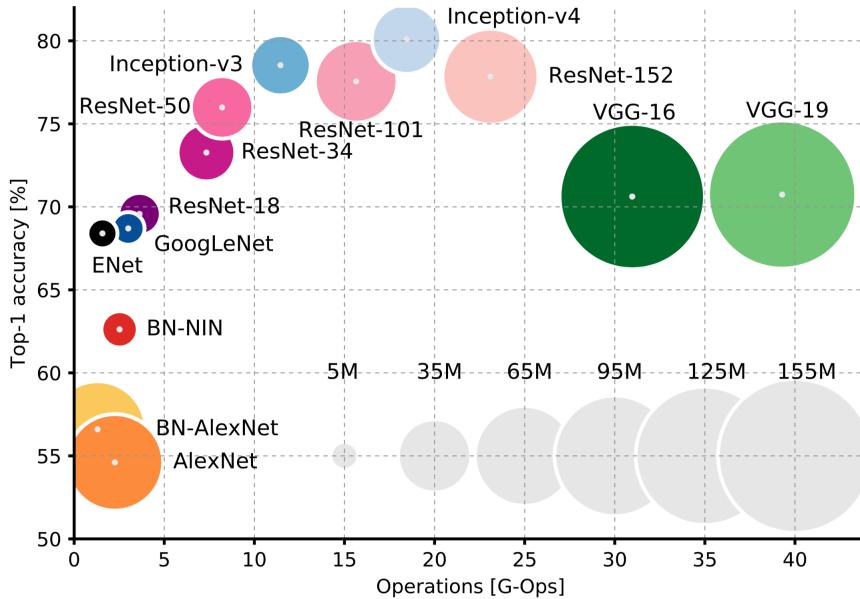


Abbildung 4.7: Vergleich von vortrainierten Convolutional Neural Networks, entnommen aus [CPC16]

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

4.4. Training

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 hat dieselbe Anzahl Parameter wie die Inception-V3-Architektur, aber eine höhere Genauigkeit [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-Architettur [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 die Rechenleistung zu senken und die dennoch Genauigkeit zu verbessern. Dabei werden Convolution-Operationen nicht nur gestapelt, sondern auch parallel zu anderen Convolution-Operationen berechnet [Sze15].

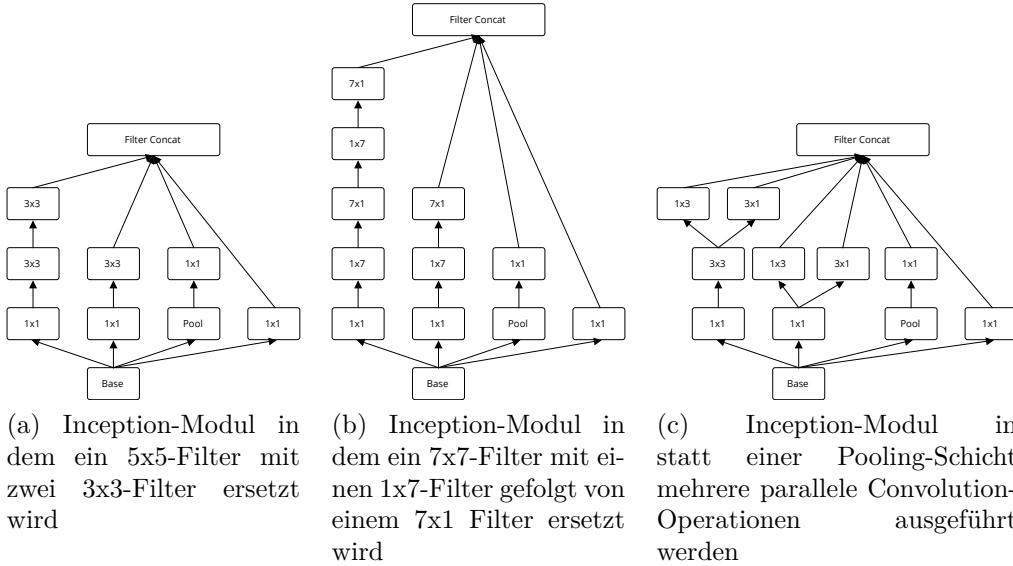


Abbildung 4.8: Inception-Module aus der Inception-V3 Architektur [Sze15]

Es wird argumentiert, dass ein Filter mit der Dimension 5x5 durch zwei aufeinanderfolgende Filter der Dimension 3x3 ersetzt werden kann, was die Rechenleis-

tung deutlich reduziert [Sze15]. Diese Funktionalität wird in einem sogenannten Inception-Modul umgesetzt, was in Abbildung 4.8a zu sehen ist. Ein weiteres Prinzip ist, dass ein $n \times n$ -Filter mit einem $1 \times n$ -Filter gefolgt von einem $n \times 1$ Filter ersetzt werden kann, bei gleichbleibender Leistung und 33% weniger Rechenaufwand. 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 Datenset ImageNet [Den09] trainiert wurde. Außerdem werden die letzten Schichten der Inception-V3-Architektur (in der Abbildung der *final part*) am Ende des Netzes entfernt und andere Schichten eingefügt, die in den nachfolgenden Absätzen erklärt werden.

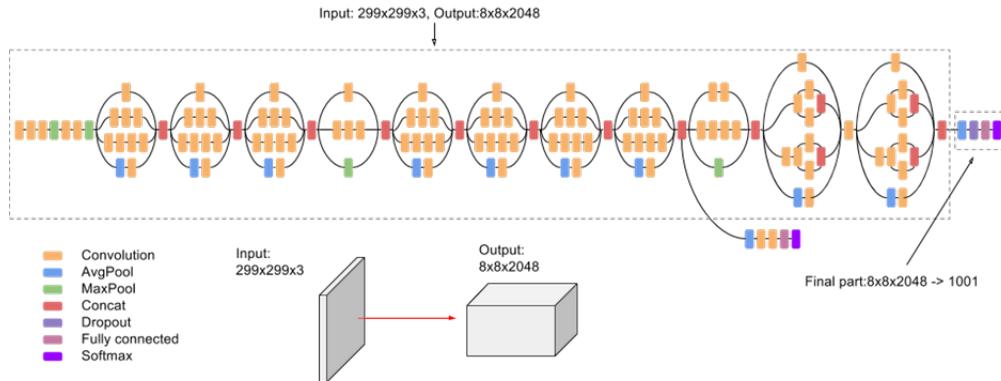


Abbildung 4.9: Architektur des Inception-V3 Netzes, entnommen aus [Clo18]

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) komplett 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

4.4. Training

Dimensionen $3 \times 3 \times 1$, der auf jede Schicht der Feature Map angewendet wird. Diese getrennten Convolution-Operation (engl. depthwise separable convolution) ist in dem Inception-Modul in Abbildung 4.10 dargestellt.

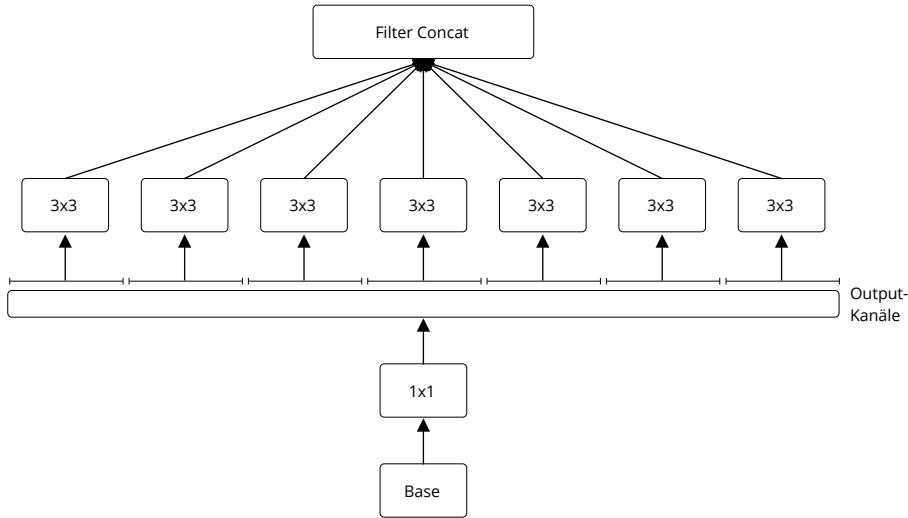


Abbildung 4.10: Inception-Modul der Xception-Architektur [Cho17]

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 vortrainiert. Für die Anwendung in dieser Arbeit werden ebenfalls die letzten Fully-Connected-Schichten entfernt.

Fully-Connected-, Dropout- und LSTM-Schicht

Eine Fully-Connected-Schicht ist, wie in Abschnitt 2.2.2 beschrieben, eine normale 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 ebenfalls immer eine Fully-Connected-Schicht. In dieser Arbeit werden in jeder Architektur zwei dieser Schichten verwendet, eine nach dem 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

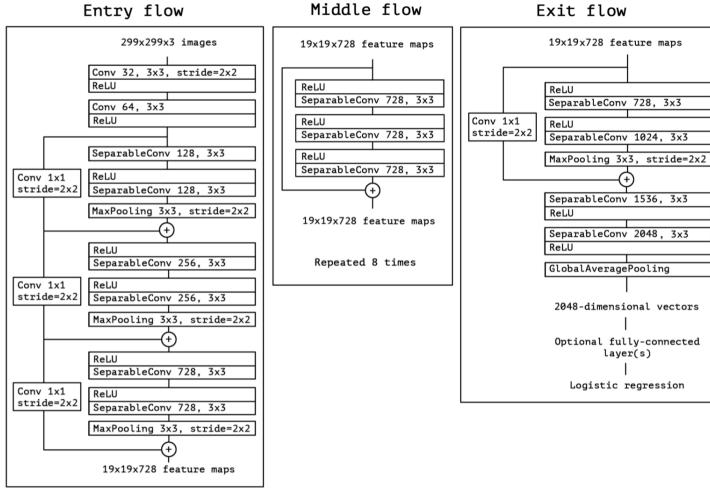


Abbildung 4.11: Xception-Architektur, entnommen aus [Cho17]

werden alle zuvor extrahierten Merkmale auf die Anzahl der Klassen heruntergebrochen und 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 Softmax als Aktivierungsfunktion verwendet [Bis06].

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.

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, indem bei jedem Trainingsschritt zufällig einige Neuronen und die zugehörigen Kanten im Netz, die Gewichte, entfernt. Dafür wird einer Schicht im Netz 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% 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

4.4. Training

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.

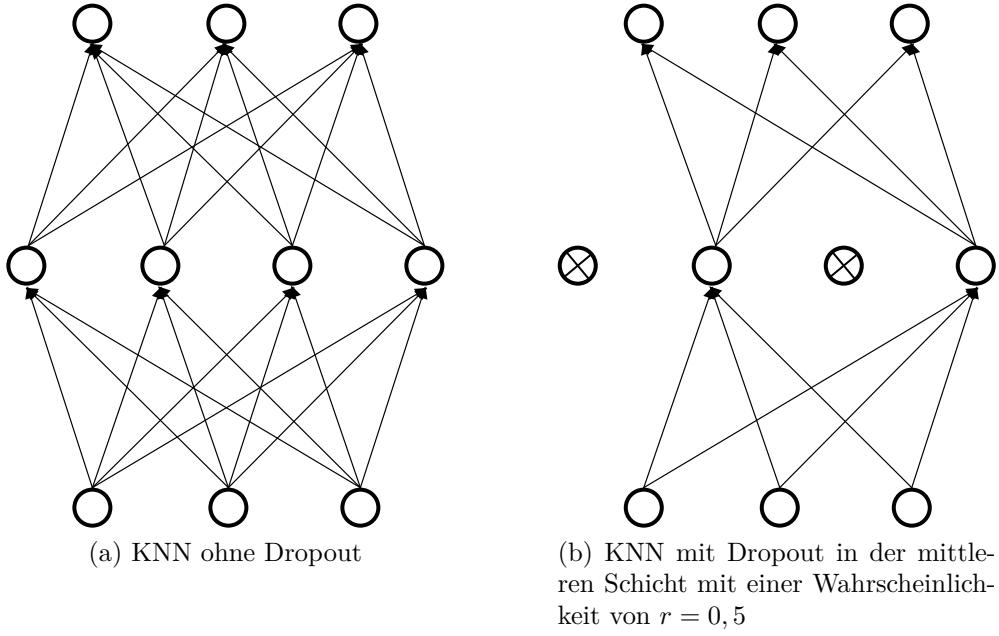


Abbildung 4.12: Regularisierung mit Dropout [Sri14]

Early Stopping ist eine Methode bei der das Training eines KNNs abgebrochen wird, wenn die Genauigkeit oder 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.

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 Parameter und keinen Einfluss auf die Klassifizierung. Die eigentliche LSTM-Schicht wird mit der Ausgangsdimension *units* = 256 und der tanh-Aktivierungsfunktion konfiguriert.

4.4.3 Wahl der Parameter

Bei der Konfiguration von KNNs und den einzelnen Schichten können viele unterschiedliche Parameter konfiguriert werden. Da es sich bei dieser Arbeit um einen Proof-of-Concept handelt, werden sowohl im Abschnitt 4.4.2 bei der Konfiguration der KNN-Architekturen, als auch in diesem Abschnitt, oftmals auf die Standardkonfiguration von Keras zurückgegriffen. Diese Standardkonfiguration basiert 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 der 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 eine möglichst hohe 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

4.4. Training

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 Parametern mit jeweils zwei Ausprägungen ergeben sich insgesamt acht Experimente. Die Parameter mit ihren Ausprägungen sind in Tabelle 4.8 zusammengefasst.

Parameter	Ausprägungen
Prinzip der Klassifizierung	einzelne Bilder, Bildersequenzen
Convolutional Neural Network	Inception-V3, Xception
Regularisierung	ohne Dropout, mit Dropout

Tabelle 4.8: Parameter die in den Experimenten variiert werden

Komponenten zusammenfügen

Auf Basis der oben beschriebenen Komponenten und Parameter werden jetzt acht verschiedene KNN-Architekturen designed. Dabei wird zwischen den Architekturen für die Klassifizierung einzelner Bilder und den Architekturen für die Klassifizierung von Bildersequenzen unterschieden. Beide grundlegenden Architekturen sind schematisch in Abbildung 4.13 dargestellt.

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 verwen-

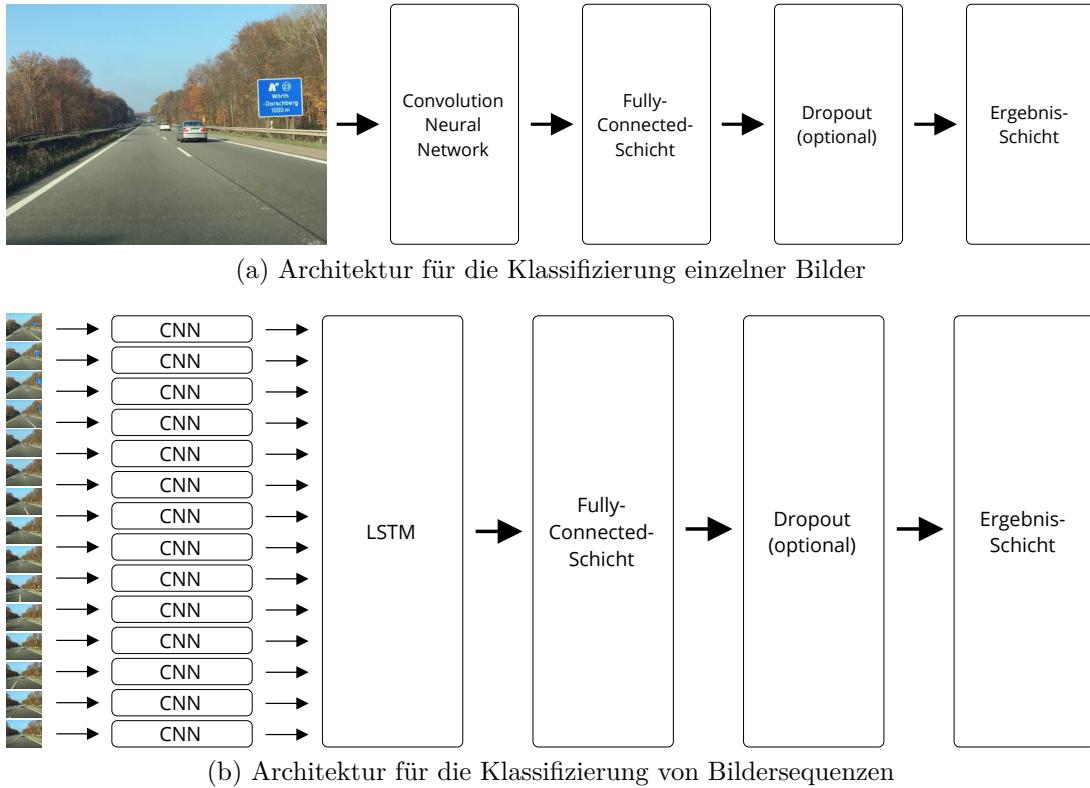


Abbildung 4.13: Grundlegende KNN-Architekturen in dieser Arbeit

det. 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 Parametern zusammengefasst.

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

4.4. Training

Architekturen			
A	B	C	D
Input shape: 299x299x3			
Inception-V3 Output shape: 2048		Xception 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 Bildersequenzen

ReLU-Aktivierungsfunktion eingefügt und zweimal mit der Regularisierungsme thode 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 Bildersequenzen sind in Tabelle 4.10 mit ihren Parametern 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 4.4.2 mit den Parametern aus Abschnitt 4.4.3 vorgestellt. Jede Architektur wird mit einer Maximalanzahl von 100 Epochen trainiert. 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 Genauigkeit wiederhergestellt.

In Abschnitt 5.1 wird die Genauigkeit der jeweiligen Netzarchitekturen bei der Klassifizierung der realen Testdaten und die Anzahl der trainierten Epochen hinsichtlich der unterschiedlichen Parameter 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 Parameter

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 der realen Testdaten von 0,95 mit der Architektur F für Videoerkennung erreicht. Dieses Ergebnis ist auch in Abbildung 5.1 in der Konfusionsmatrix (engl. confusion matrix) dargestellt. Die niedrigste Genauigkeit von 0,28 wird mit der Architektur H für Videoerkennung erreicht. Architekturen für die reine Bilderkennung erreichen Genauigkeiten zwischen 0,48 (Architektur D) und 0,73 (Architektur A). Damit erzielen diese Architekturen Ergebnisse mit weniger Varianz im Vergleich zu den Architekturen mit LSTM-Schicht.

5.1. Variation der Parameter

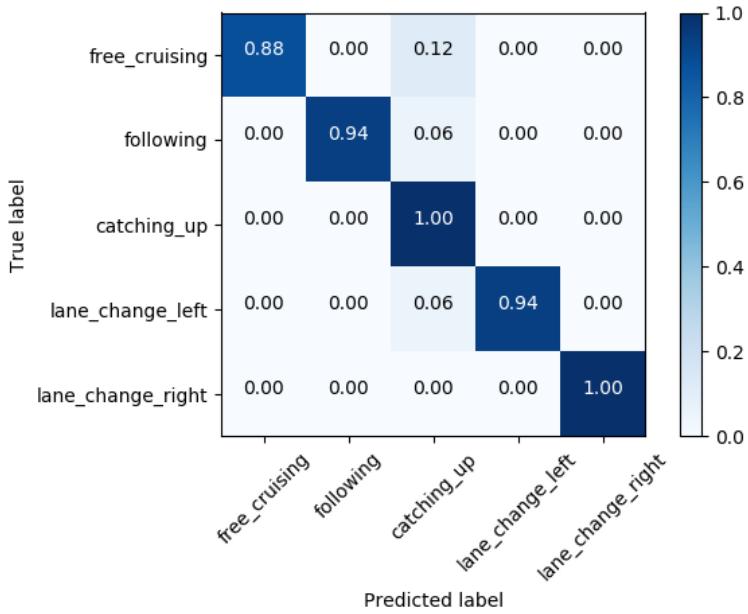


Abbildung 5.1: Konfusionsmatrix der Klassifizierung von realen Testdaten mit der Architektur F (Inception-V3, LSTM, Dropout)

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 0,67. Nur die Architektur G erreicht eine höhere Genauigkeit als Architektur E mit einem Unterschied von 0,02. Insgesamt überrascht dieser Unterschied, da die Xception-Architektur in anderen Arbeiten [Cho17] höhere Genauigkeiten bei der Klassifizierung von Bildern erreicht hat.

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 bessere Genauigkeit ohne

Regularisierung mit Dropout in der vorletzten Schicht.

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 und schneller konvergiert.

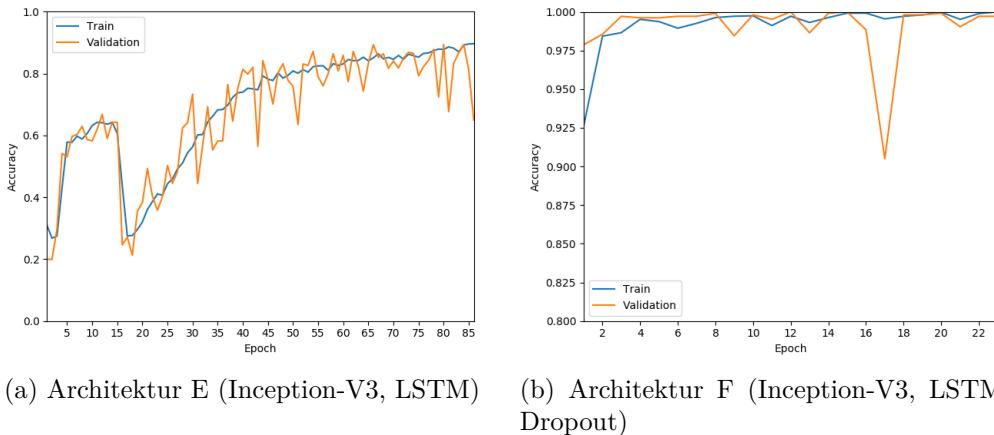


Abbildung 5.2: Genauigkeit der Trainings- und Testdaten während dem Training von zwei verschiedenen Architekturen

Eine mögliche Erklärung dafür könnte die Komplexität der Architekturen mit LSTM-Schicht sein. Diese Architekturen haben mehr trainierbare Parameter und sind daher anfälliger für eine Überanpassung [Hin12]. Regularisierung mit Dropout kann die Komplexität reduzieren und führt zu höheren Genauigkeiten [Hin12]. Die Architekturen mit reinen CNNs sind weniger komplex und daher könnte Dropout keine oder eine negative Auswirkung auf das Ergebnis haben.

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,

5.2. Synthetische und reale Testdaten

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

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

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% synthetischer und nur 5% 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 Klassifizierung von einzelnen synthetischen Bildern mit der Xception Architektur niedriger ist als die Klassifizierung von realen Testbildern. Aktuell hat der Autor keine Erklärung hierfür. Dieses Ergebnis kann als Ausgangspunkt für weitere Arbeiten dienen.

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.

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

Um an dieser Stelle besser verstehen zu können auf welcher Basis die KNNs die Bilder beziehungsweise die Bildsequenzen erkennen, sollten in nachfolgenden

5.3. Genauigkeit der einzelnen Szenarien

Arbeiten weitere Untersuchungen dazu angestellt werden. So wäre es beispielsweise interessant, welche Teile der einzelnen Bilder stärker oder auch weniger stark bei der Klassifizierung berücksichtigt werden.

Kapitel 6

Zusammenfassung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum.

At vero eos et accusam et justo duo dolores et ea rebum. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum.

6.1 Ergebnis und Diskussion

Die definierten Ziele aus Abschnitt 1.2 waren:

- Datensatz erstellen
- Architekturen designen
- Training mit synthetischen und realen Szenarien und Klassifizierung von realen Szenarien

6.2 Ausblick

Videos erkennen und Beschreibung dazu generieren, wie in [Don15]

Literatur

- [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.
- [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.

LITERATUR

- [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 - Rostock*. 2018. URL: <https://drive.google.com/open?id=1MpI5teZVqANhFJ6YB-n9gWijlOB0-mZ9&usp=sharing> (besucht am 21.11.2018).
- [Goo18b] Google. *Route 2: Karlsruhe - Kandel*. 2018. URL: <https://drive.google.com/open?id=1oEwdcaIl8bY4W77dTlR6U68UAf0P1TxY&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).
- [Kin17] Christian King u. a. “Identifikation von Fahrszenarien während einer virtuellen Testfahrt”. In: *INFORMATIK 2017*. 2017.
- [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.

LITERATUR

- [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.
- [PL16] Raphael Pfeffer und Tobias Leichsenring. “Continuous development of highly automated driving functions with vehicle-in-the-loop using the example of Euro NCAP scenarios”. In: *Simulation and Testing for Vehicle Technology*. 2016.
- [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.
- [Püt17] Andreas Pütz u. a. “System validation of highly automated vehicles with a database of relevant traffic scenarios”. In: *12th ITS European Congress* (2017).
- [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.
- [Sch14] Sebastian Schwab u. a. “Durchgängige Testmethode für Assistenzsysteme”. In: *ATZ-Automobiltechnische Zeitschrift* (2014).
- [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.

LITERATUR

- [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 Kulic. “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 Kulic. “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.