

Inhaltsverzeichnis

| | |
|---|-----------|
| 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 | 13 |
| 2.2.1 Einordnung im maschinellen Lernen | 13 |
| 2.2.2 Entwicklung von künstlichen neuronalen Netzen | 15 |
| 2.2.3 Convolutional Neural Networks | 20 |
| 2.2.4 Recurrent Neural Networks und LSTMs | 22 |
| 2.2.5 Training mit synthetischen Daten | 22 |
| 2.2.6 Klassifizierung von Videos | 24 |
| 3 Konzept | 25 |
| 3.1 Struktur | 26 |
| 3.2 Ansätze, Methoden, Werkzeuge | 28 |
| 4 Umsetzung | 29 |
| 4.1 Definition der Fahrszenarien | 29 |
| 4.2 Generierung synthetischer Trainingsdaten | 31 |
| 4.2.1 Simulation mit CarMaker | 31 |
| 4.2.2 Daten Labeling | 35 |
| 4.3 Generierung realer Trainings- und Testdaten | 38 |
| 4.4 Training | 39 |

| | | |
|----------|--|--------------|
| 4.4.1 | Inputdaten | 39 |
| 4.4.2 | Architektur künstlicher neuronaler Netze | 40 |
| 4.4.3 | Experimente | 41 |
| 5 | Ergebnis | 43 |
| 5.1 | Synthetische Daten | 43 |
| 5.2 | Reale Daten | 43 |
| 6 | Zusammenfassung | 45 |
| 6.1 | Ergebnis und Diskussion | 45 |
| 6.2 | Ausblick | 45 |
| | Literaturverzeichnis | xv |
| | Abkürzungsverzeichnis | xvii |
| | Tabellenverzeichnis | xix |
| | Abbildungsverzeichnis | xxi |
| A | Appendix | xxiii |
| A.1 | Python-Code für das Labeln der simulierten Fahrszenarien | xxiii |

Abstract

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, 23. Dezember 2018

Manuel Kaiser

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 künstliche neuronale Netze (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].

Zahlen, Daten Fakten

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 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</i> -specific 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</i> -specific 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</i> -specific 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</i> -specific 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 [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.

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

Die Schritte auf dem absteigenden und aufsteigenden Ast haben jeweils eine Beziehung. Jeder Test auf dem aufsteigenden Ast verifiziert bzw. validiert den dazugehö-

2.1. Hochautomatisiertes Fahren

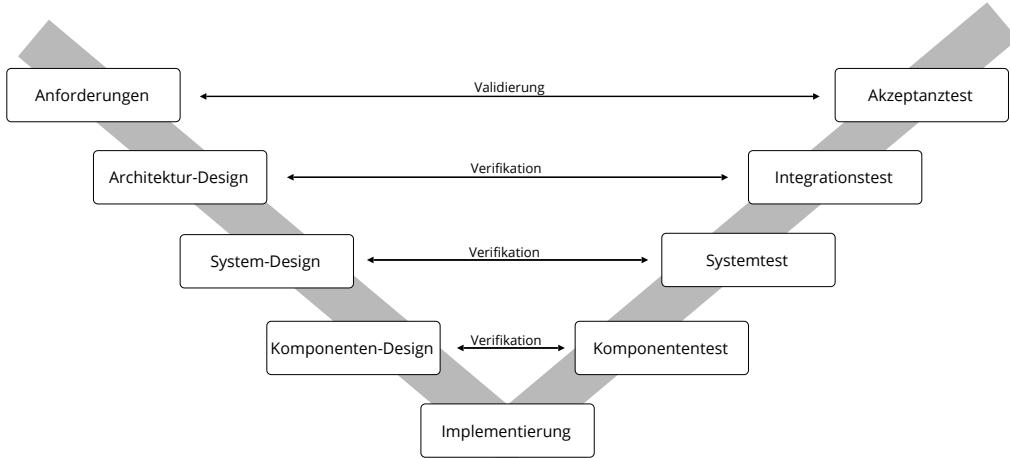


Abbildung 2.2: V-Modell [HK15]

riegen 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.

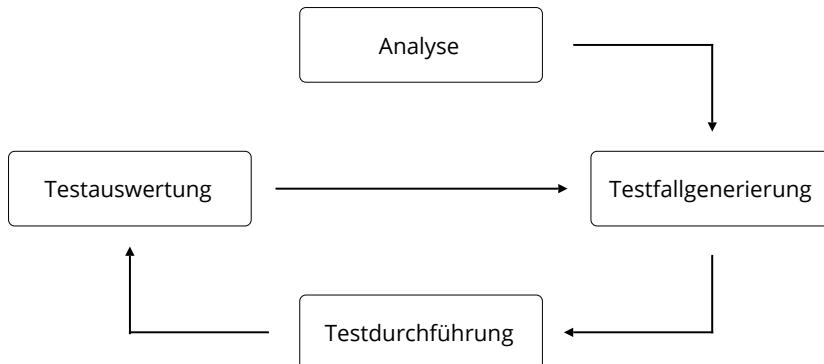


Abbildung 2.3: Schritte zur Testfallerstellung [Sch13]

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 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 Einsatz 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].

2.1. Hochautomatisiertes Fahren

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.

Neben den Begriffen Szene, Situation und Szenario wird auch oft der Begriff Manö-

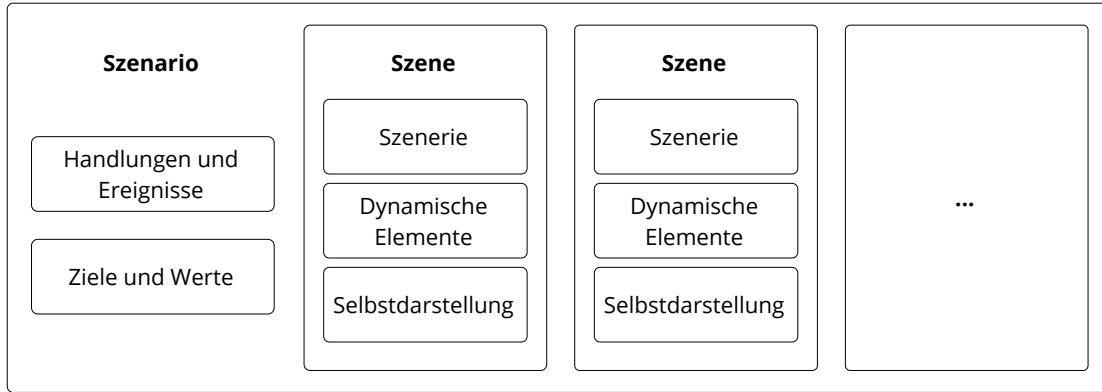


Abbildung 2.4: Zusammenhang zwischen Szene und Szenario [Ulb15]

ver 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 *Überholen* aus den Manövern *Spurwechsel nach links*, *Beschleunigung*, *Spurwechsel nach rechts* und *Bremsvorgang* zusammen. Zwischen den einzelnen Manövern gibt es Situation wie *langsameres Auto voraus* oder *langsameres Auto links überholt*, die jeweils ein neues Manöver einleiten.

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

2.1. Hochautomatisiertes Fahren

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.

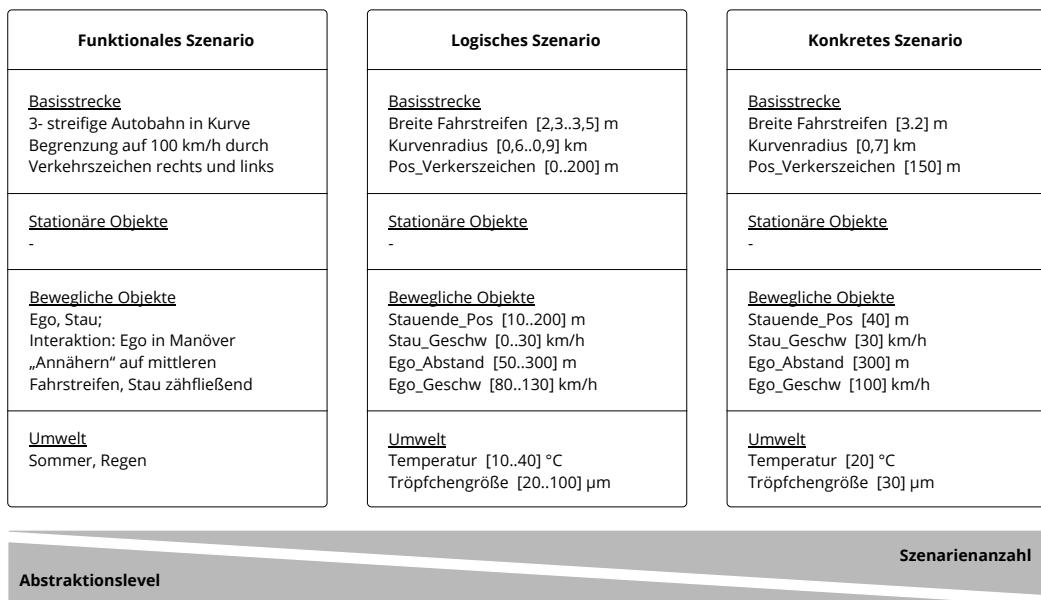


Abbildung 2.5: Beispiel für ein funktionales, logisches und konkretes Szenario [Bag17]

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 Sensordaten 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; Zhe16] und realen Testfahrten [Gru17]. Die verwendeten Daten waren der *laterale Abstand zwischen Fahrzeug und Fahrbahnmarkierung*, *Spurabfahrtsbetrag*, *Beschleunigung*, *Lenkwinkel*, *Lenkgeschwindigkeit*, *Lenkmoment* und *Ort, Ausrichtung*, und *Geschwindigkeit* des Ego-Fahrzeugs und benachbarten Objekten. Dabei wurde nicht weiter spezifiziert wie die Daten ausgelesen wurden.

Auf Basis der Sensordaten wurden verschiedene Klassifikatoren erstellt. Es wurden die Methoden Support Vector Machine (SVM) [Sun17; Cer16; WK16; CHK16; Zhe16; ZSH15], Random Forest (RF) [XHK18; Cer16; Zhe16], k-Nearest-Neighbor (kNN) [ZH17; CHK16; Zhe16], Hidden Markov Model (HMM) [ZH17; Li15], Fuzzy Rule-Based Classifier (FRC) [Cer16; ABR16], Bayesian Inference Model [Sun17], Convolutional Neural Network (CNN) [Gru17], Decision Tree [ZSH14] und Naive Bayes [CHK16] verwendet. Da 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 Arbeiten 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 |

2.1. Hochautomatisiertes Fahren

| 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 | Frei fahren, 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 |
| [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 |

| Quelle | Sensordaten | Klassifikator | Szenarien |
|---------|---|---|--|
| [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 Gruner [Gru17] wird sich die zukünftige Forschung mit tieferen Netzstrukturen wie Recurrent Neural Networks (RNNs) beschäftigen, um zeitabhängige Szenarien noch besser zu verstehen.

Mit den bisherigen Ansätzen können bekannte Szenarien 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.

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 Long Short-Term Memorys (LSTMs) eingeführt. Im folgenden Abschnitt 2.2.1 werden KNNs in den Gesamtkontext von maschinellem Lernen gestellt. Im Anschluss werden in Abschnitt 2.2.2 die Grundlagen zu KNNs erläutert. 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.“

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 .

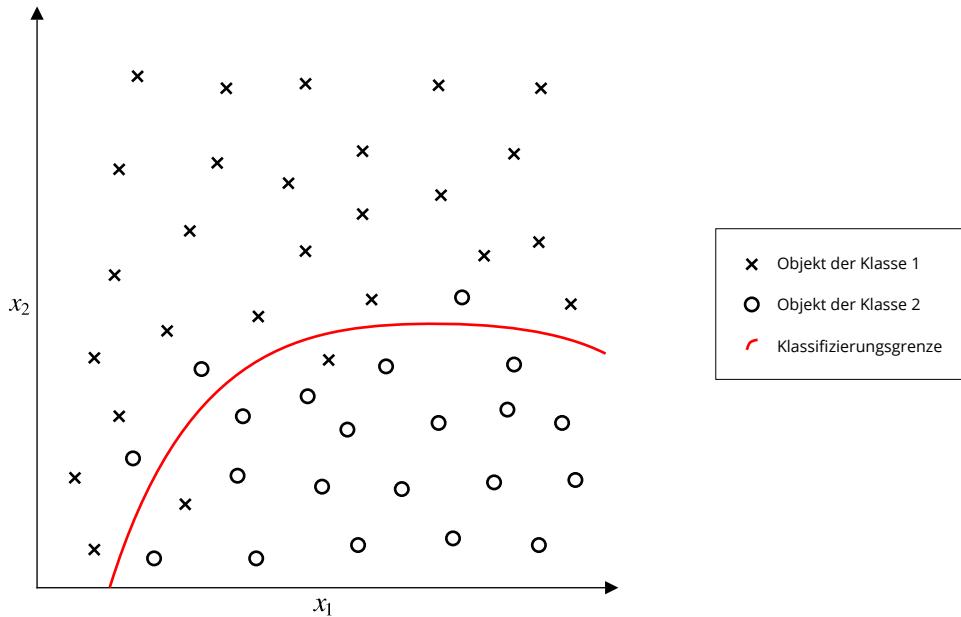


Abbildung 2.6: Beispiel einer Klassifizierung mit zwei Klassen

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.

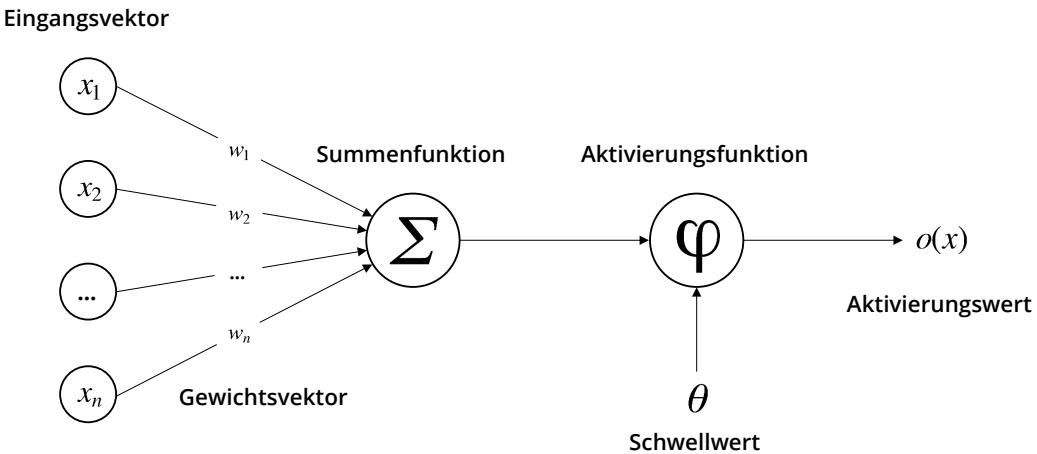


Abbildung 2.7: Modell eines Perzeptrons [Ros58]

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

2.2. Künstliche Neuronale Netze

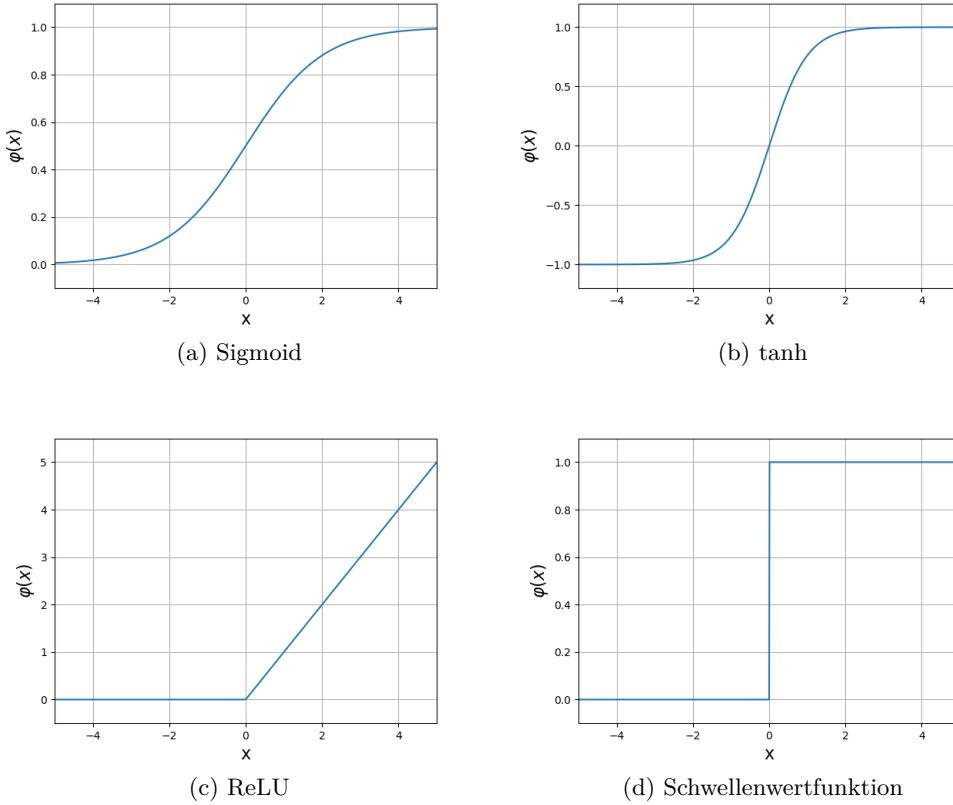


Abbildung 2.8: Aktivierungsfunktionen für Neuronen

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 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:

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

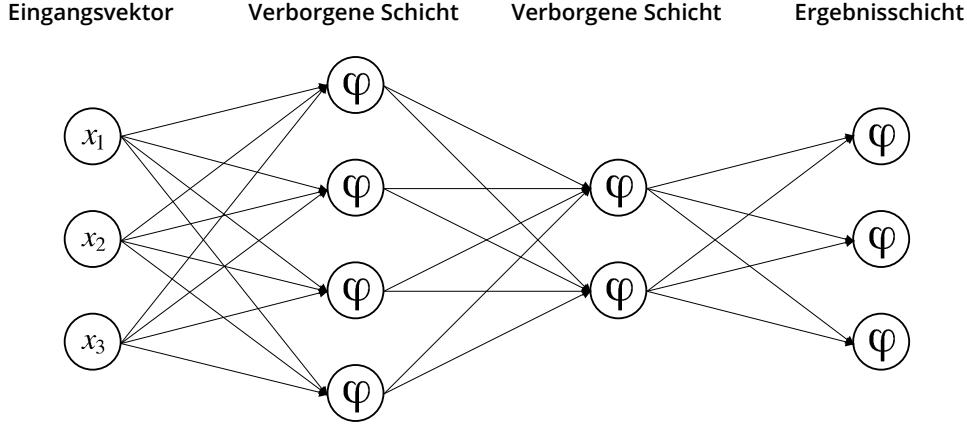


Abbildung 2.9: Dreischichtiges künstliches neuronales Netz

Dabei ist w_{ij} die Gewichtung der Kante zwischen dem Eingangswert x_i und 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 tatsächlichen Klasse beschreibt, zu definieren und dann zu minimieren [Bis06]. Dafür wird die Funktion

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

verwendet. 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.

Heute werden in Anwendungen häufig tiefe neuronale Netze (engl. Deep Neural Networks) verwendet. Von einem tiefen neuronalen Netz spricht man, wenn es viele verdeckte Schichten besitzt. Damit können Merkmale auf verschiedenen Abstraktionsebenen erkannt werden [LBH15]. In den folgenden Absätzen 2.2.3 und 2.2.4 werden zwei

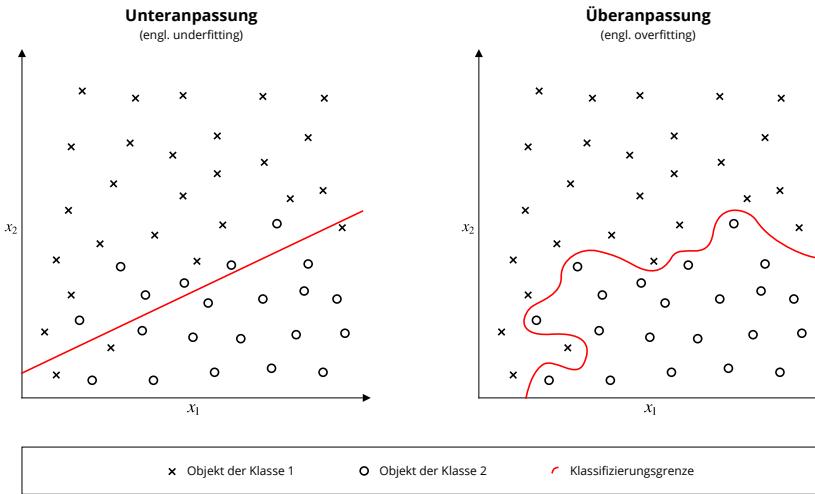


Abbildung 2.10: Beispiel einer Unter- und Überanpassung eines Klassifikators

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 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 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 und in späteren Schichten zunehmend abstraktere Merkmale extrahiert. Bei der Convolution-Operation (oder auch Faltung) wird ein ausgewählter Filter mit einer festgelegten Schrittgröße über das Bild bewegt und bei jedem Schritt der entsprechende Ausgabewert berechnet. Diese Operation ist in Abbildung 2.11 dargestellt.

In diesem Beispiel handelt es sich um ein Bild mit den Dimensionen 5x5x1 Pixel,

2.2. Künstliche Neuronale Netze

| | | | | | |
|----|----|----|----|----|----|
| 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

also einem zweidimensionalen Bild mit einem Farbkanal. Der Filter hat die Dimensionen 3x3x1. In dem Beispiel ist die aktuell dargestellte Rechnung wie folgt:

$$0 * 33 + 0 * 15 + 1 * 1 + 0 * 93 + 0 * 84 + 0 * 17 + 1 * 36 + 0 * 72 + 0 * 83 = 37 \quad (2.12)$$

Nach dieser Berechnung bewegt sich der Filter einen Schritt weiter nach rechts und der nächste Wert wird berechnet. 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 und 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. 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.

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

→

| | |
|-----|-----|
| 139 | 167 |
| 127 | 160 |

Abbildung 2.12: Beispiel einer Max-Pooling-Operation

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

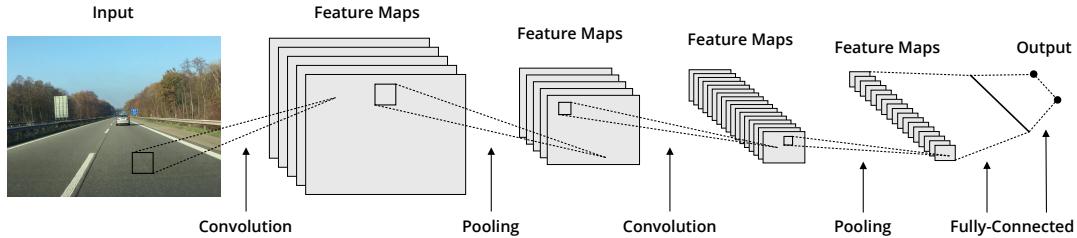


Abbildung 2.13: Beispiel eines CNNs

2.2.4 Recurrent Neural Networks und LSTMs

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. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. 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. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

2.2.5 Training mit synthetischen Daten

[Ric16]

Creating large datasets with pixelwise semantic labels is known to be very challenging due to the amount of human effort required to trace accurate object boundaries.

We extracted 24,966 frames from GTA5. Each frame has a resolution of 1914×1052 pixels. 19 semantic classes (e.g. road, sky, car)

The crop size is 628×628 and the receptive field is 373×373 pixels.

The results show that when we train on 1/3 of the CamVid training set along with the game data, we surpass the accuracy achieved when training on the full CamVid training set without game data. The table shows that using the synthetic data during training increases the mean IoU by 3.9 percentage points.

Our strongest baseline is the state-of-the-art system of Kundu et al. [25], which used a bigger ConvNet and analyzed whole video sequences. By using synthetic data

2.2. Künstliche Neuronale Netze

during training, we outperform this baseline by 2.8 percentage points, while considering individual frames only.

KITTI: The model trained with game data outperforms the model trained without game data by 2.6 percentage points.

[Ros16]

SYNTHIA consists of photo-realistic frames rendered from a virtual city and comes with precise pixel-level semantic annotations for 13 classes, i.e., sky, building, road, sidewalk, fence, vegetation, lane-marking, pole, car, traffic signs, pedestrians, cyclists and miscellaneous

SYNTHIA has been generated by rendering a virtual city created with the Unity development platform

(SYNTHIA-Seqs simulates four video sequences of approximately 50,000 frames each one up to a total of 200,000 frames, acquired from a virtual car across different seasons (one sequence per season).)

SYNTHIA-Rand consists of 13,400 frames of the city taken from a virtual array of cameras moving randomly through the city, with its height limited to the range [1.5m, 2m] from the ground.

Both sequences share standard properties as frame resolution of 960×720 pixels and horizontal field of view of 100 degrees.

All images are resized to a common resolution of 180×120 .

Observe that, for all the datasets and architectures, the inclusion of synthetic data systematically helps to boost the average per-class accuracy.

Test on CamVid, KITTI, U-LabelMe, CBCL

[Joh17]

Specifically, we leverage Grand Theft Auto V (GTA V). Sim 10k with 10,000 images, Sim 50k with 50,000 images, and Sim 200k with 200,000 images.

The performance of each of the trained networks was evaluated using a standard intersection over union (IoU) criteria for produced bounding boxes [6].

Networks trained using the simulated data were capable of achieving high levels of performance on real-world data without requiring any mixing with real-world training data set imagery.

depending on type of car (easy, medium hard) between 42 and 69 acc (IoT > IoU (Intersection over Union))

Tested on KITTI

[Tre18]

We begin with 3D models of objects of interest (such as cars). A random number of these objects are placed in a 3D scene at random positions and orientations.

The approach is evaluated on bounding box detection of cars on the KITTI dataset.

For our DR dataset, we generated 100K images containing no more than 14 cars each. As described in the previous section, each car instance was randomly picked from a set of 36 models, and a random texture from 8K choices was applied.

For testing, we used 500 images taken at random from the KITTI dataset [8].

With fine-tuning on all 6000 real images, our DR-based approach achieves an AP score of 98.5, which is better than VKITTI by 1.6% and better than using only real data by 2.1%.

2.2.6 Klassifizierung von Videos

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

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. Diese Beziehung ist schematisch in Abbildung 3.1 dargestellt. 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 erläutert, ist die Identifikation von bisher unbekannten Fahrszenarien. Die Grundidee ist es einen Klassifikator mit einem großen Anteil synthetischer Daten und einem kleinen Anteil realer Daten von bisher bekannten Szenarien zu trainieren. Dieser Klassifikator kann dann bekannte Szenarien erkennen, liefert aber keine eindeutigen Ergebnisse bei bisher unbekannten Szenarien. Mit dieser Methodik soll es möglich sein bisher unbekannte Fahrszenarien zu identifizieren, um auf der Basis neue Testfälle für die Sicherung hochautomatisierter Fahrfunktionen zu generieren.

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.

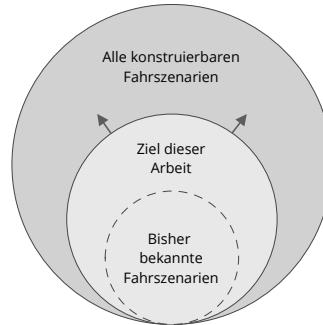


Abbildung 3.1: Beziehung zwischen bekannten und unbekannten Fahrszenarien

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 3.2 abgebildet und wird in den folgenden Absätzen weiter beschrieben.

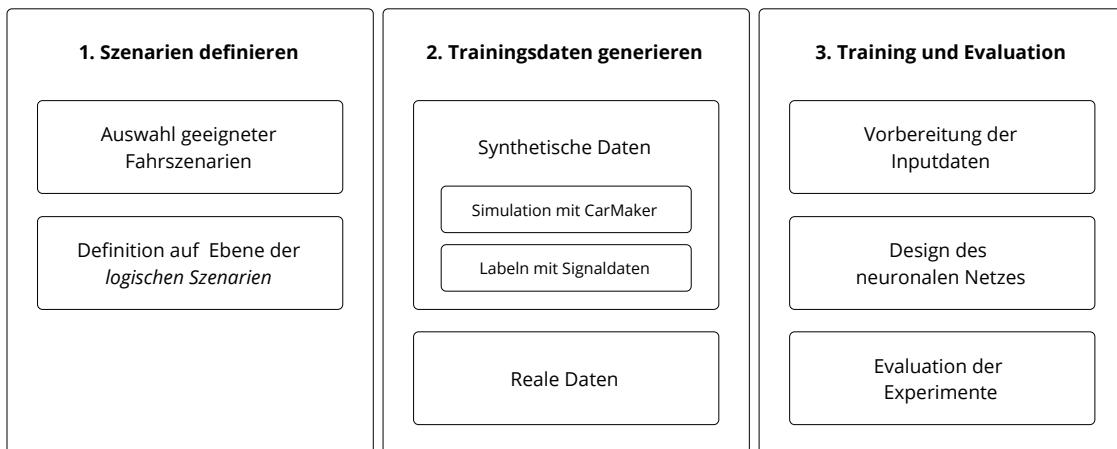


Abbildung 3.2: Konzept dieser Arbeit

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 Car-

3.1. Struktur

Maker gearbeitet. Mit dieser Software können das Ego-Fahrzeug, Straßen, Verkehr und die Trajektorien aller Fahrzeuge generiert und beliebig verändert werden. 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 3.3 dargestellt. In dieser Arbeit werden für das Labeln der Bild- und Signaldaten Bilddaten 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.

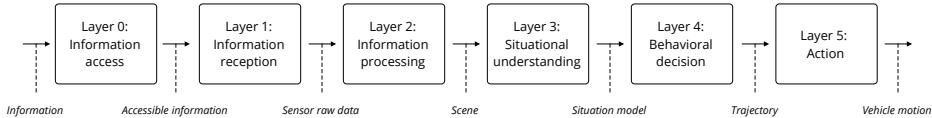


Abbildung 3.3: Schema der funktionalen Dekomposition [AW17]

Für das Training und die Evaluation eines Klassifikators werden KNNs verwendet. Wie in Abschnitt 2.2 beschrieben, sind CNNs nach dem Stand der Technik die besten Architekturen um Merkmale aus Bildern zu extrahieren. LSTMs sind besonders für Zeitreihen geeignet. Daher wird ein Klassifikator aus einer Kombination dieser Architekturen erstellt. Um die Skalierbarkeit dieses Ansatzes zu gewährleisten, wird für das Training nur ein kleiner Teil realer und ein großer Teil synthetischer Daten verwendet. Damit wird auch für die Zukunft der Aufwand des manuellen Labelns gering gehalten.

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 | Gegenstand dieser Arbeit |
| Generierung realer Trainingsdaten | Auswahl geeigneter Videosequenzen von YouTube, manuelles Labeln | Gegenstand dieser Arbeit |
| Vorbereitung der Daten, Erstellung des Klassifikators und Evaluation der Experimente | Architektur eines KNN mit einer Kombination von CNN und LSTM, implementiert mit Python und Keras | [Cho15], CNN, LSTM |

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 |
| Following | Ego, andere Verkehrsteilnehmer <u>Interaktion:</u> Ego fährt auf linker oder rechter Fahrspur in sicherem Abstand hinter Fahrzeug 2 | Geschwindigkeit Ego [60..200] km/h Geschwindigkeitsdifferenz zwischen Ego und Fahrzeug 2 $<$ Geschwindigkeit Ego *0,05 km/h Abstand Ego zu Fahrzeug 2 [$s_3..s_1$] m Ego befindet sich auf gleicher Fahrspur hinter Fahrzeug 2 |

4.2. Generierung synthetischer Trainingsdaten

| Szenario | Funktionale Definition | Logische Definition |
|--------------------------|---|---|
| Catching up | Ego, andere Verkehrsteilnehmer <u>Interaktion:</u> Ego fährt auf linker Fahrspur und verringert den vertikalen Abstand zu Fahrzeug 2 auf der rechten Fahrspur (auf-/überholen) | Geschwindigkeit Ego [60..200] km/h Geschwindigkeit Fahrzeug 2 < Geschwindigkeit Ego Vertikaler Abstand Ego zu Fahrzeug 2 [0..s ₀] m Ego fährt auf linker Fahrspur hinter Fahrzeug 2 das auf rechter Fahrspur fährt |
| Overtaking | | |
| 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 entsprechender Kameratasensor konfiguriert. Die Konfiguration des Sensors orientiert sich an der Konfiguration von realen Frontview-Kameras im Fahrzeug nach Punke [Pun15]. So ist der Kameratasensor an der Stelle des Rückfahrspiegels platziert und hat eine Auflösung von 640x480 Pixeln, ein vertikales Blickfeld von XX° und ein horizontales Blickfeld von XX°. Die Konfiguration und Position des Sensors im Modul *CarMaker - Vehicle Data Set* ist in Abbildung 4.1 zu sehen.

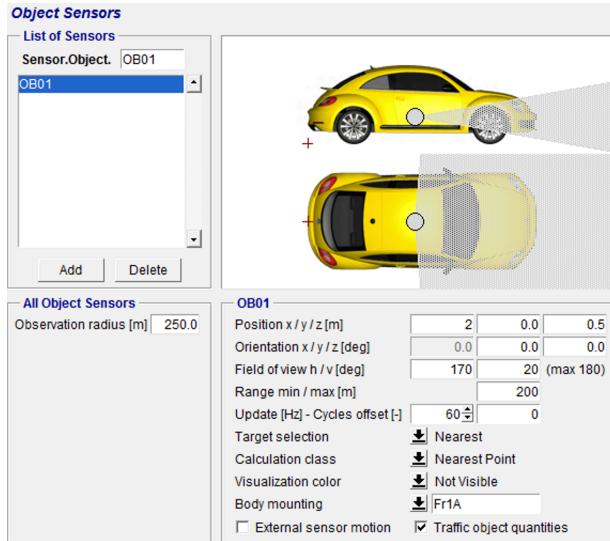


Abbildung 4.1: Konfiguration des Kameratasensors im Modul *CarMaker - Vehicle Data Set* [Gmb18]

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

4.2. Generierung synthetischer Trainingsdaten

die *OutputQuantities* aufgezeichnet werden. Die jeweiligen Variablen in CarMaker sind in der Tabelle 4.2 zusammengefasst.

| 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.Act.LaneId | 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.Act.LaneId | Fahrspur-ID des Fahrzeugs TX |

Tabelle 4.2: Aufgezeichnete Signaldaten in CarMaker

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 4.3 mit grünen Streifen gekennzeichnet ist. Abbildung 4.2 zeigt die Konfiguration der simulierten Straße .

Bild einfügen



Abbildung 4.2: Konfiguration der simulierten Straße [Gmb18]

Auf beiden Strecken wird autonomer, stochastisch verteilter Verkehr erzeugt, was CarMaker mit einer gesonderten Funktion unterstützt. Der Verkehr wird in einer niedrigen Dichte (10%) 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 auf der 10.000m-Strecke 131 Fahrzeuge und auf der 6.000m-Strecke 89 Fahrzeuge generiert.

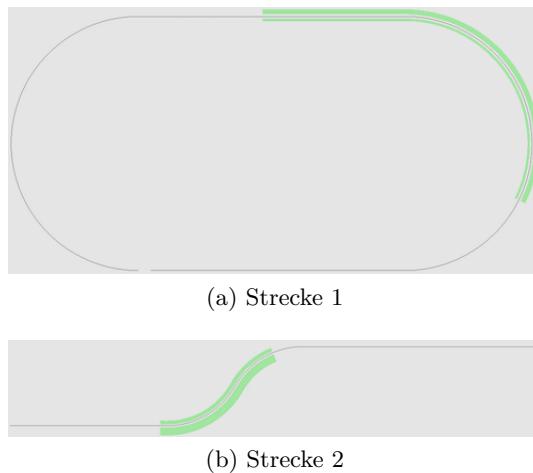


Abbildung 4.3: Schema der simulierten Strecken 1 und 2 [Gmb18]

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

| 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

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 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.

4.2.2 Daten Labeling

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.

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

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

| Szenario | Bedingungen |
|--------------------------|---|
| Approaching | $s_2 < v2_{ds} < s_0$ $ego_{sRoad} < v2_{sRoad}$ $ego_{laneID} == v2_{laneID}$ $ego_v < \text{Durchschnitt von } ego_v \text{ der letzten 3 Sekunden}$ |
| Following | $v2_{dv} < ego_v * 0,05$ $s_3 < s_1$ $ego_{sRoad} < v2_{sRoad}$ $ego_{laneID} == v2_{laneID}$ |
| Catching up | $v2_{dv} < 0$ $0 <= v2_{ds} < s_0$ $ego_{sRoad} <= v2_{sRoad}$ $ego_{laneID} == v2_{laneID} - 1$ |
| Overtaking | $0 <= v2_{ds} < s_0$ $v2_{sRoad} < ego_{sRoad}$ $ego_{laneID} == v2_{laneID} - 1$ |
| 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.

Es sind auch einige Szenarien als *unknown* klassifiziert, da zwischen den definierten Szenarien andere Situationen auftreten können oder nicht die Mindestanzahl der konse-

4.2. Generierung synthetischer Trainingsdaten

| 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

kutiven 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 4.4 zeigt ein Beispiel des Szenarios *lane change left* mit allen zugehörigen 15 Bildern.

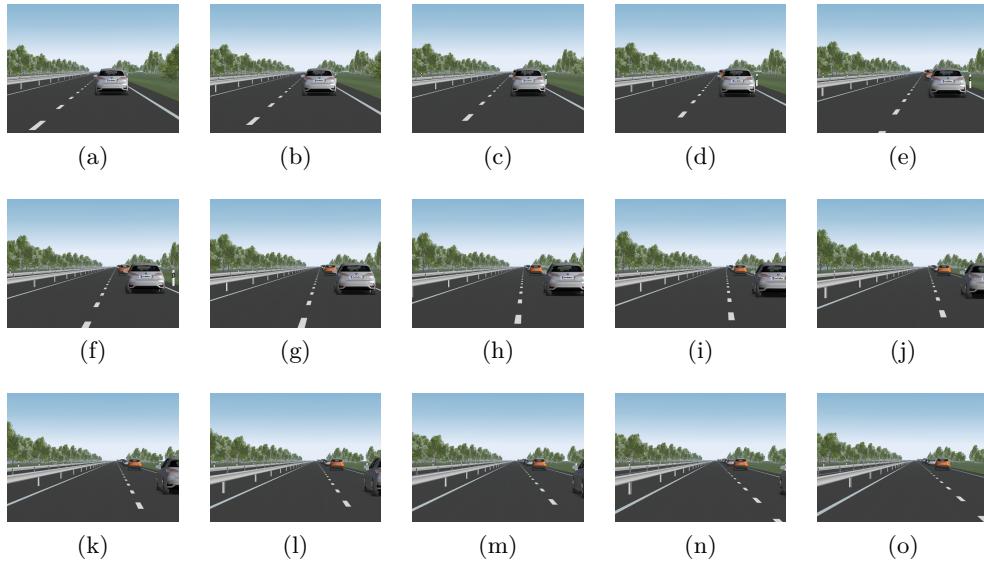


Abbildung 4.4: Beispiel eines simulierten Szenarios der Klasse *lane change left* [Gmb18]

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 und die anschließenden Tests benötigt. Dafür werden im ersten Schritt bestehen 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 4.5 abgebildet.

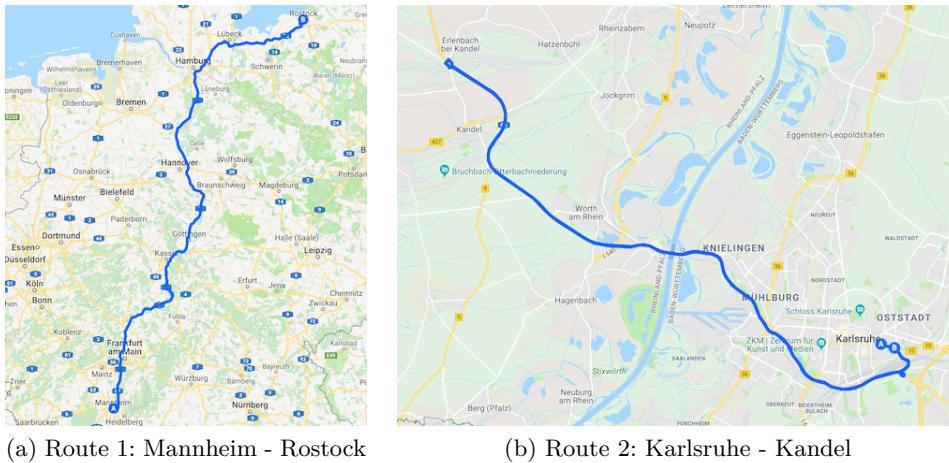


Abbildung 4.5: Routen für die Aufnahme der realen Bilddaten [Goo18a; Goo18b]

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. Das Ergebnis sind jeweils 17 weitere Szenarien in den Klassen *lane change left* und *lane change right*. Abbildung

4.4. Training

4.6 zeigt ein Beispiel des realen Szenarios *lane change right* mit allen zugehörigen 15 Bildern.

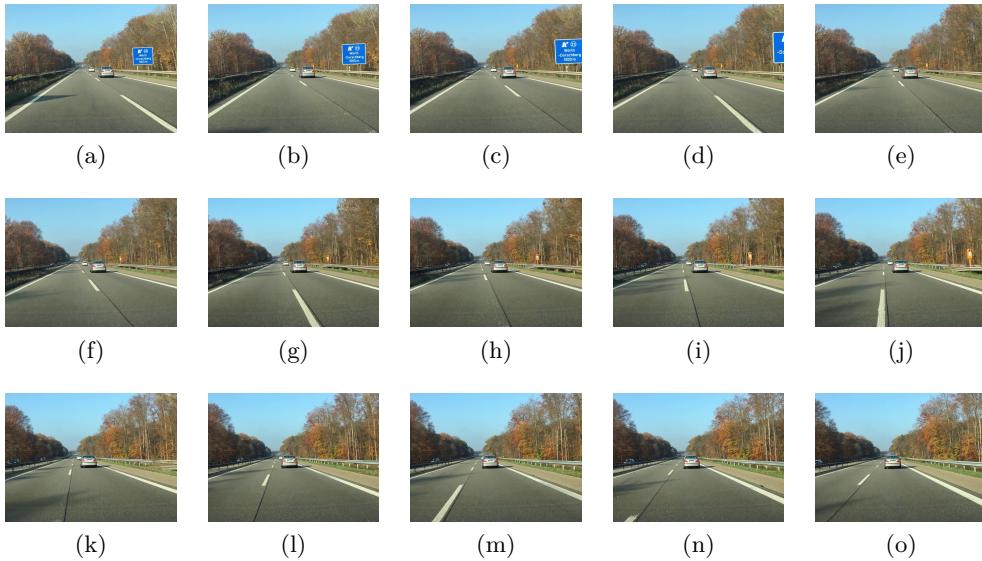


Abbildung 4.6: Beispiel eines realen Szenarios der Klasse *lane change right*

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 Inputdaten

Die Generierung der synthetischen und realen Trainings- und Testdaten wurde bereits in den vorherigen Abschnitten beschrieben. In diesem Abschnitt soll kurz darauf eingegangen werden 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 KNN importiert werden.

Da es sich um große Datenmengen von 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 hat bereits sogenannte *DataGenerators* für der Import von einzelnen Bildern und für sequenzielle Daten mit zwei Dimensionen (e.g. CSV-Dateien). Es gibt allerdings keine bereits nutzbare Lösung für den Import von sequentiellen Bildern, was 4-dimensionalen Daten entspricht (Anzahl Bilder, Höhe, Breite, Farbkanal). Aus diesem Grund wird die Lösung von Amidi ?? 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* wurde 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* wurde entfernt, weil es mit der konfigurierten Frontkamera allein nicht erfasst werden kann. In folgenden Arbeiten kann dieses Szenario mithilfe weiterer Kameraperspektiven zusätzlich berücksichtigt werden.

Um das Ergebnis nicht zu verzerren, sollten für das Training mit neuronalen Netzen in jeder Klasse jeweils die gleiche Anzahl an Trainings- 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 4.6 aufgeschlüsselt. Beim Training mit einzelnen Bildern wird die gleiche Verteilung angewendet und da jedes Szenario aus 15 Bildern besteht, können die Zahlen aus Tabelle 4.6 einfach mit 15 multipliziert werden.

4.4.2 Architektur künstlicher neuronaler Netze

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit

4.4. Training

| Szenario | Synthetische Daten | | | Reale Daten | | | Summe |
|-------------------|--------------------|-------------|------------|-------------|-------------|-----------|-------|
| | Training | Validierung | Test | Training | Validierung | Test | |
| | 85% | 10% | 5% | 65% | 10% | 25% | |
| free cruising | 807 | 95 | 48 | 43 | 7 | 17 | 1.017 |
| following | 807 | 95 | 48 | 43 | 7 | 17 | 1.017 |
| catching up | 807 | 95 | 48 | 43 | 7 | 17 | 1.017 |
| lane change left | 807 | 95 | 48 | 43 | 7 | 17 | 1.017 |
| lane change right | 807 | 95 | 48 | 43 | 7 | 17 | 1.017 |
| Summe | 4.035 | 475 | 240 | 215 | 35 | 85 | 5.085 |

Tabelle 4.6: Aufteilung von synthetischen und realen Szenarios auf Trainings-, Validierungs- und Testdaten

augue duis dolore te feugait nulla facilisi.

4.4.3 Experimente

Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, At accusam aliquyam diam diam dolore dolores duo eirmod eos erat, et nonumy sed tempor et et invidunt justo labore Stet clita ea et gubergren, kasd magna no rebum. sanctus sea sed takimata ut vero voluptua. est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur

Dropout - [Hin12]

Kapitel 5

Ergebnis

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

5.1 Synthetische Daten

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

5.2 Reale Daten

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

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.

6.1 Ergebnis und Diskussion

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat,
vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim
qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lo-
rem ipsum dolor sit amet, consectetur adipisciing elit, sed diam nonummy nibh euismod
tincidunt ut laoreet dolore magna aliquam erat volutpat.

6.2 Ausblick

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id
quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetur adi-
pisciing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam
erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper
suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie conse-
quat, vel illum dolore eu feugiat nulla facilisis.

Literaturverzeichnis

- [ABR16] Cesar Arroyo, Luis M. Bergasa und Eduardo Romera. “Adaptive fuzzy classifier to detect driving events from the inertial sensors of a smartphone”. In: *IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. 2016.
- [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 Kulić. “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).
- [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: *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=1oEwdcaIl8bY4W77dTLR6U68UAfOP1TxY&usp=sharing> (besucht am 21.11.2018).
- [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 Fahrrerassistenzsysteme*. 2015.

LITERATURVERZEICHNIS

- [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.
- [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).
- [Li15] Guofa Li u. a. “Lane change maneuver recognition via vehicle state and driver operation signals—Results from naturalistic driving data”. In: *Intelligent Vehicles Symposium (IV)*. 2015.
- [LKF10] Yann LeCun, Koray Kavukcuoglu und Clément Farabet. “Convolutional Networks and Applications in Vision”. In: *International Symposium on Circuits and Systems*. 2010.
- [Mad17] Will Maddern u. a. “1 year, 1000 km: The Oxford RobotCar dataset”. In: *The International Journal of Robotics Research* (2017).
- [Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [MP43] Warren S. McCulloch und Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *Bulletin of Mathematical Biophysics* (1943).
- [MP69] Marvin L. Minski und Seymour A. Papert. *Perceptrons: an introduction to computational geometry*. The MIT Press, 1969.
- [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).
- [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.

- [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: (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: *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.
- [Sur18] Sebastian Surmund u. a. “Neue Szenarien für autonome Fahrsysteme”. In: *ATZ extra* (2018).
- [Tre18] Jonathan Tremblay u. a. “Training deep networks with synthetic data: Bridging the reality gap by domain randomization”. In: *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.

LITERATURVERZEICHNIS

- [WK16] Christopher Woo und Dana Kulić. “Manoeuvre segmentation using smartphone sensors”. In: *Intelligent Vehicles Symposium (IV)*. 2016.
- [WW15] Walther Wachenfeld und Hermann Winner. “Die freigabe des autonomen Fahrens”. In: *Autonomes Fahren*. 2015.
- [XHK18] Jie Xie, Allaa R Hilal und Dana Kulić. “Driving Maneuver Classification: A Comparison of Feature Extraction Methods”. In: *IEEE Sensors Journal* (2018).
- [Yu18] Fisher Yu u. a. “BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling”. In: *Computing Research Repository (CoRR)* (2018).
- [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.

Abkürzungsverzeichnis

Items einrücken wie
andere Verzeichnisse

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

Tabellenverzeichnis

| | | |
|-----|--|----|
| 2.1 | Bisherige Arbeiten zur Szenarienerkennung | 12 |
| 3.1 | Ansätze, Methoden und Werkzeuge dieser Arbeit | 28 |
| 4.1 | Definition der Szenarien <i>free cruising, approaching, following, catching up, overtaking, lane change left</i> und <i>lane change right</i> | 31 |
| 4.2 | Aufgezeichnete Signaldaten in CarMaker | 33 |
| 4.3 | Variablen und Werte die in der Simulation verwendet werden | 35 |
| 4.4 | Bedingungen der Szenarien <i>free cruising, approaching, following, catching up, overtaking, lane change left</i> und <i>lane change right</i> | 36 |
| 4.5 | Anzahl der simulierten Szenarien nach Klasse | 37 |
| 4.6 | Aufteilung von synthetischen und realen Szenarios auf Trainings-, Validierungs- und Testdaten | 41 |

Abbildungsverzeichnis

| | | |
|------|---|----|
| 2.1 | Norm SAE J3016 für die Level des autonomen Fahrens [Com14] | 4 |
| 2.2 | V-Modell [HK15] | 5 |
| 2.3 | Schritte zur Testfallerstellung [Sch13] | 5 |
| 2.4 | Zusammenhang zwischen Szene und Szenario [Ulb15] | 8 |
| 2.5 | Beispiel für ein funktionales, logisches und konkretes Szenario [Bag17] . . | 9 |
| 2.6 | Beispiel einer Klassifizierung mit zwei Klassen | 14 |
| 2.7 | Modell eines Perzeptrons [Ros58] | 15 |
| 2.8 | Aktivierungsfunktionen für Neuronen | 17 |
| 2.9 | Dreischichtiges künstliches neuronales Netz | 18 |
| 2.10 | Beispiel einer Unter- und Überanpassung eines Klassifikators | 20 |
| 2.11 | Beispiel einer Convolution-Operation | 21 |
| 2.12 | Beispiel einer Max-Pooling-Operation | 21 |
| 2.13 | Beispiel eines CNNs | 22 |
| 3.1 | Beziehung zwischen bekannten und unbekannten Fahrszenarien | 26 |
| 3.2 | Konzept dieser Arbeit | 26 |
| 3.3 | Schema der funktionalen Dekomposition [AW17] | 27 |
| 4.1 | Konfiguration des Kamerasensors im Modul <i>CarMaker - Vehicle Data Set</i> [Gmb18] | 32 |
| 4.2 | Konfiguration der simulierten Straße [Gmb18] | 33 |
| 4.3 | Schema der simulierten Strecken 1 und 2 [Gmb18] | 34 |
| 4.4 | Beispiel eines simulierten Szenarios der Klasse <i>lane change left</i> [Gmb18] . | 37 |
| 4.5 | Routen für die Aufnahme der realen Bilddaten [Goo18a; Goo18b] | 38 |
| 4.6 | Beispiel eines realen Szenarios der Klasse <i>lane change right</i> | 39 |

Anhang A

Appendix

A.1 Python-Code für das Labeln der simulierten Fahrszenarien

```
1 import numpy as np
2 # This is a comment
3 a = 5
4 for x in range(a):
5     print("Hello")
```