Freie Universität Berlin
Fachbereich Mathematik und Informatik

Freie Universität Berlin

Masterarbeit

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

Analyse und Implementation von Methoden zur Spurmarkierungserkennung aus Bilddaten zum Generieren von virtuellen Teststrecken

Studiengang:	Informatik
Gutachter:	Prof. Dr. Daniel Göhring
	Prof. Dr. Raúl Rojas
Betrieb:	Carmeq GmbH
	Carnotstraße 4
	10587 Berlin
eingereicht von:	Jessica Lynn Concepcion <5152081>
eingereicht am:	29.11.2019

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Ort, Datum J. L. Concepcion

Abstract

Die Entwicklung von Fahrerassistenzsystemen und automatisierten Fahrfunktionen erfordert zunehmend einen hohen Testaufwand. Der virtuelle Fahrversuch mithilfe der Simulation ergänzt und entlastet die reale Fahrerprobung, darf sich aber in Qualität und Aussagekraft nicht wesentlich vom realen Test auf der Versuchsstrecke oder im öffentlichen Straßenverkehr unterscheiden.

Mit steigenden Anforderungen müssen die virtuellen Teststrecken immer realistischer das wirkliche Straßenbild nachstellen. Dazu gehört auch, möglichst wirklichkeitsnahe Teststrecken zu generieren, die nicht nur einzelene Testfälle abbilden. Reale Straßen und ihre Markierungen entsprechen häufig nicht den Idealvorstellungen und variieren in ihren Eigenschaften. Grundvoraussetzung, eine Teststrecke aus Bilddaten realer Straßen zu generieren, sind akkurate Bilderkennung und -unterscheidung. Die klassische Spurerkennung ist hier nur bedingt hilfreich.

Gegenstand dieser Arbeit ist es, verschiedene Ansätze zum Erkennen von Spurmarkierungen auf Bilddaten zu analysieren und zu evaluieren. Die beste erreichbare und effizienteste Lösung wurde mittels einer semantischen Segmentierung, die die verschiedenen Arten der Spurmarkierung erkennt und pixelweise klassifiziert, implementiert. Mithilfe der Segnet-Architektur konnte ein IOU Score von 0.3169 und ein Dice Koeffizient von 0.4790 erreicht werden. Darüber hinaus wurden weitere unabdingbare Arbeitsschritte diskutiert, die erforderlich sind, um Teststrecken generieren zu können.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Adam Adaptive Moment Estimation

API Application Programming Interface

ASPP Atrous Spatial Pyramid Pooling

CNN Convolutional Neural Networks

CRF Conditional Random Field

CT Computed Tomography

FAS Fahrerassistenzsysteme

FNN Feedforward Neural Network

GAN Generative Adversarial Networks

HiL Hardware in the Loop

IoU Intersection over Union

MSE Mean Squared Error

NN Neural Network

ReLU Rectified Linear Unit

R-FGÜ Richtlinien für die Anlage und Ausstattung von Fußgängerüberwegen

RGB Rot-Grün-Blau Farbskala

RNN Recurrent Neural Network

SCNN Spatial Convolutional Neural Network

SiL Software in the Loop

StVO Straßenverkehrsordnung

ViL Vehicle in the Loop

VwV Verwaltungsvorschrift

XML Extensible Markup Language

1 Einleitung

Ziel dieser Arbeit ist es, eine Methode zu finden, die geeignet ist, detaillierte Informationen über Spurmarkierungen aus Bildern zu extrahieren. Dazu zählen vorrangig der Verlauf der Markierungen, ihre Farbe, Breite (Schmalstrich, Breitstrich, Doppelstrich) und ob sie durchgängig oder unterbrochen sind. Die aus der Arbeit resultierende Spurmarkierungserkennung soll im Bereich der Simulation eingesetzt werden, wo die erkannten Daten zur Erstellung virtueller Teststrecken verwendet werden können. Kooperationspartner der Arbeit ist die Carmeq GmbH.

1.1 Betriebliches Umfeld

Die Carmeq GmbH ist ein Unternehmen im Volkswagen-Konzern und arbeitet für die internationale Automobil- und Zulieferindustrie. Wegweisende Lösungen für attraktive und sichere Mobilität sind das Kerngeschäft. Zu den aktuellen Themengebieten zählen E-Mobilität, autonomes Fahren und Connected Cars.[1]

Die Abteilung für Simulation und Testwerkzeuge gehört zu der Funktions- und Systementwicklung für Systeme im Komfort- und Fahrerassistenzbereich. Im Mittelpunkt steht die Simulation zum Testen von Fahrerassistenzsystemen (FAS) zur Absicherung der Funktionen. Um innovative FAS auf ihre Sicherheit und Funktionstüchtigkeit zu prüfen, sind Testfahrten von mehreren Millionen Kilometern nötig. Um diese immensen Anforderungen an Fahrer, Fahrzeug und Fahrstrecke zu minimieren, werden neue Funktionen vorab virtuell mit Computersimulationen getestet, bevor es zu realen Testfahrten kommt.[1]

1.2 Motivation

Neue FAS, hochautomatisches und autonomes Fahren stellen immer neue, höhere Anforderungen an die Testumgebung. Während der Entwicklung solcher Systeme sind unkomplizierte Alternativen zu Testfahrten unverzichtbar, um schnell, präzise, sicher, ökologisch und ökonomisch effizient zu Ergebnissen zu gelangen. Am besten eignet sich hier die Simulation, die in verschiedenen Abstufungen eingesetzt werden kann und nahe-

zu alle denkbaren Szenarien abdeckt. So gibt es neben kompletten Softwaresimulationen (SiL) auch Optionen wie Hardware in the Loop (HiL), wo Hardwarekomponenten mit simuliertem Input getestet werden.

Um Simulationen im Fahrerassistenzbereich durchführen zu können, werden digitale Teststrecken benötigt, auf denen die Fahrten getestet werden. Je realistischer diese virtuellen Straßen konzipiert sind, umso verlässlichere Ergebnisse können bei den Testläufen erzielt werden. Hier sollen die Ergebnisse dieser Arbeit dazu dienen, Möglichkeiten aufzuzeigen und Grenzen auszuloten, Teststrecken aus Bilddaten echter Straßen zu generieren. So können Bilder von tatsächlich existierenden Straßen in die Simulation integriert werden. Diese Vorgehensweise führt zweifellos zu realistischeren Teststrecken und somit auch zu wirklichkeitsnahen Simulationslösungen.

1.3 Aufgabenstellung

Ziel der Masterarbeit ist es, eine möglichst genaue Spurmarkierungserkennungsmethode für Bilddaten zu entwickeln. Dabei sollen Spurmarkierungen nicht nur erkannt, sondern auch nach Art klassifiziert werden. Ohne die notwendige Präzision bei der Bilderkennung sind Anpassungen von Hand erforderlich wie bei der Toolkette, die momentan für diesen Prozess genutzt wird.

Langfristig angestrebt wird, ein Tool zu entwickeln, das automatisch eine Teststrecke aus Bilddaten erstellt und die fertige Datei ausgibt. In dieser Arbeit soll eine geeignete Methode zur Spurmarkierungserkennung erarbeitet und implementiert werden, die das ermöglicht. Die Entscheidung, welche Vorgehensweise gewählt wird, um eine optimale Spurmarkierungserkennung vorzunehmen, soll Teil der Arbeit sein.

Beabsichtigt ist, eine Methode zu finden, die in der Lage ist, alle notwendigen Informationen zu den Markierungen zu erfassen, die benötigt werden, um eine schlüssige Strecke zu generieren. Für die Erstellung von Teststrecken sind bereits Bibliotheken vorhanden. Hier ist zunächst wichtig, die Angaben der Spurmarkierungen in eine Form zu bringen, die es ermöglicht, die Teststrecke mit ihnen nachzubilden. Vorrangig muss dafür festgestellt werden, welche Spurmarkierungsdaten hierfür benötigt werden.

1.4 Verwandte Arbeiten

Es gibt zahlreiche Veröffentlichungen zur Spurerkennung und -verfolgung im Bereich der Bilderkennung. Für die spezifische Aufgabenstellung eignen sich diese Lösungsansätze oft nicht, da die Markierungen nicht exakt genug erfasst werden. Der Einsatzbereich dieser Methoden dient häufig für die Funktionen von FAS, wie zum Beispiel Spurhalteassistenten. Ansätze und Erkenntnisse, die Impulse für die weitere, detaillierte Sacharbeit geben, sind aber durchaus in den Publikationen zu finden. Hier sei stellvertretend verwiesen auf das „Spatial As Deep: Spatial CNN for Traffic Scene Understanding“ von Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang und Xiaoou Tang.

In dem 2017 veröffentlichten Paper wird ein Verfahren vorgestellt, das es ermöglicht, neben den bekannten Vorteilen des Convolutional Neural Networks (CNN) auch sein Potential zum Erkennen räumlicher Beziehungen von Pixeln über Zeilen und Spalten zu nutzen. Dafür haben Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang und Xiaoou Tang traditionelle Convolutions zu schichtweisen Convolutions innerhalb von Feature-Maps umgewandelt. Hiermit ist es ihnen gelungen, den Informationsaustausch zwischen Pixeln über Zeilen und Spalten innerhalb eines Layers zu ermöglichen. Die Anwendungsfälle, in denen das Spatial Convolutional Neural Network (SCNN) getestet wurde, waren Spurerkennung und die Semantische Segmentierung der Spurgrenzen. Mit einer Genauigkeit von 96,53% übertrafen die Ergebnisse bei der TuSimple Benchmark Lane Detection Challenge 2017 auf Hawaii die ihrer Mitbewerber.[2]

Auch die 2018 veröffentlichte Arbeit „EL-GAN: Embedding Loss Driven Generative Adversarial Networks for Lane Detection“ von Mohsen Ghafoorian, Cedric Nugteren, Nóra Baka, Olaf Booij und Michael Hofmann nutzt neue Lösungsansätze, um eine möglichst optimale Spurerkennung zu erzielen. Dafür verwenden sie Generative Adversarial Networks (GAN), die trainiert werden, indem man zwei verschiedene Netze gegeneinander antreten lässt: Generator und Discriminator. Wie der Name schon impliziert, wird der Generator darauf trainiert, die gewünschte Datenverteilung zu generieren. Als “quasi” seinem Gegenspieler wird dem Discriminator beigebracht, die produzierten Ergebnisse des Generators (Fälschungen) von den echten Labels (Originale) zu unterscheiden. Durch diesen doppelten Feedback-Kreislauf, in dem beide Netzte gegensätzliche Funktionen trainieren, entsteht ein sogenanntes „Zero-Sum-Game“. Dadurch müssen beide Netze stetig ihr Verhalten anpassen, um die Aufgabe erfolgreich zu bewältigen.[3]

In der Arbeit wird die Wahl, ein GAN zu benutzen, damit legimitiert, dass die Ergebnisse realistischer sind und bessere Strukturen aufweisen. An den Resultaten ist der Unterschied zu anderen Spurerkennungsverfahren gut wahrnehmbar - ein großer Vorteil, da bisher oft Segmentierungsverfahren benutzt wurden, die keine klaren Linien zum Ergebnis haben. Durch die Nutzung der EL-GAN-Architektur wurde nur die Mitte einer Spurmarkierung im Verlauf gekennzeichnet, was für die klassische Spurerkennung eine klarere Grenze darstellt. Auch diese Architektur konnte eine Genauigkeit über 96% bei dem TuSimple Benchmark Datenset erreichen.[3]

1.5 Aufbau der Arbeit

Die Struktur der Arbeit basiert auf den Arbeitsabläufen zur Bearbeitung der Aufgabenstellung. Im Kapitel 2 werden zunächst alle für das Verständnis benötigten Informationen bereitgestellt. Im Anschluss werden im Kapitel 3 explizite Lösungsansätze diskutiert und ihre Implementierung beschrieben. Darauffolgend werden im Kapitel 4 die genauen Testergebnisse analysiert und weitere notwendige Arbeitsschritte beschrieben. Kapitel 5 ist das Resümee der Arbeit. Offene Fragen werden angesprochen und auf weiterführende Überlegungen wird hingewiesen.

2 Grundlagen

In diesem Kapitel werden Informationen gegeben, die zum Verständnis der Aufgabe und möglicher Lösungsansätze notwendig sind. Umfang und Vielfalt von Gesetzen, Verordnungen, Durchführungsbestimmungen bedingen, dass in dieser Arbeit nur die wichtigsten Passagen erwähnt werden. Auf Quellen und weitere Literatur wird verwiesen.

2.1 Spurmarkierungen

Gemäß § 39 Abs. 5 Straßenverkehrsordnung (StVO) sind sowohl Markierungen als auch Radverkehrsführungsmarkierungen Verkehrszeichen und dienen in Zusammenhang mit anderen Leiteinrichtungen der Sicherheit im Straßenverkehr. Dieser Aufgabe können sie nur gerecht werden, wenn die richtigen Materialien verwendet, sie richtig appliziert werden und ihre Funktionstüchtigkeit über die gesamte Lebensdauer sichergestellt wird. In Deutschland sind sie grundsätzlich weiß; gelbe Markierungen gelten nur vorübergehend und heben weiße Markierungen auf. Markierungslinien können auch in Form von Markierungsknopfreihen, Markierungsleuchtknopfreihen oder als Leitschwellen oder Leitborde auftreten. Eine weitere Sonderregelung gilt in verkehrsberuhigten Geschäftsbereichen § 45 Abs. 1d StVO in Verbindung mit § 39 Abs. 5 Satz 7 StVO und Verwaltungsvorschrift (VwV) IV/8 zur StVO; hier können Spurmarkierungen auch Pflasterlinien sein und müssen in der Regel eine Breite von mindestens 0,1 Meter haben und einen deutlichen Kontrast zur Fahrbahn aufweisen.

Um gute Sichtbarkeit und eine lange Lebensdauer sicherzustellen, sind die Wahl der richtigen Materialien und eine ordnungsgemäße Applikation unersetzlich. Die genauen Anforderungen an Markierungen auf Straßen und Straßenmarkierungsmaterialien sind in der Norm DIN EN 1436:2018 festgelegt. Reflexion, Retroflexion sind ebenso exakt definiert wie Farbbereiche und Griffigkeit.[4]

Bei den Markierungszeichen werden Schmal- und Breitstrich in Abhängigkeit von der Straßenkategorie verwendet. So sind Schmalstrichlinien auf einer Autobahn 0,15 Meter breit und 0,12 Meter bei anderen Straßen. Die Breitstrich-Markierungen sind auf

Autobahnen 0,30 Meter breit, sonst 0,25 Meter. Für Haltlinien (StVO Anlage 2 zu § 41 Abs. 1 Verkehrszeichen 294) gibt es gesonderte Regelungen, da sich diese von normalen Spurmarkierungen abheben müssen. Sie sind überall dort anzubringen, wo Verkehrsteilnehmer ein Halte- bzw. Warteverbot beachten müssen und sind mittels eines durchgehenden Querstrichs mit einer Strichbreite von 0,5 Meter gekennzeichnet. Fußgängerüberwege (StVO, Anlage 2, Abschnitt 9, lfd.Nr. 66) bestehen aus parallel zur Fahrtrichtung angeordneten 0,5 Meter breiten Markierungsstreifen mit jeweils 0,5 Meter Abstand (Richtlinien für die Anlage und Ausstattung von Fußgängerüberwegen (R-FGÜ) 2001). Die Länge der Striche und damit die Breite des Fußgängerüberweges beträgt mindestens 3 Meter (§ 26 StVO in Verbindung mit VwV R-FGÜ 2001).

Im realen Straßenverkehr kommt es bei Spurmarkierungen allerdings immer wieder zu Abweichungen von den Richtlinien. Oft treten diese schon bei der Applikation auf, beispielsweise durch Verrutschen. Nach Entfernung von temporären Markierungen bleiben häufig Phantomspuren erhalten. Die Nichteinhaltung der Abmessungen ist ein weiterer Kritikpunkt. Auch der Verschleiß der Spurmarkierungen kann zur mangelhaften Sichtbarkeit der Markierungen führen. Der Deutsche Verkehrssicherheitsrat und die Deutsche Studiengesellschaft für Straßenmarkierungen empfehlen der Polizei, den Straßenverkehrs- und Straßenbaubehörden der Länder und Kommunen Fahrbahnmarkierungen mehr Aufmerksamkeit zu schenken. Korrekte und qualitativ hochwertige Fahrbahnmarkierungen sind eine gute Investition in die Verkehrssicherheit und eine unerlässliche Voraussetzung für die Verkehrskonzepte der Zukunft.[5]

2.2 Simulation in der Automobilbranche

„Simulation ist die Nachbildung eines dynamischen Prozesses in einem realen Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind.“[6][Blatt1]

Aus dem Zitat geht das Hauptziel der Simulation bereits hervor, zu Erkenntnissen, die auf die Realität übertragbar sind, zu gelangen. Dafür werden Modelle verwendet, die die Wirklichkeit möglichst detailgetreu widerspiegeln. Solche Imitationen der Realität bieten die Möglichkeit, quantitative Aussagen über den Nutzen von Einsatzmitteln zu treffen. So ist es möglich, die aussichtsreichsten Konzepte bereits ohne den Bau von Prototypen zu identifizieren. Dies hält die Materialkosten gering, da zu testende Elemente und Funktionen in die vorhandene, virtuelle Testumgebung integriert werden können. Weitere Vorteile sind schnellere Anpassung an neue Sicherheits- und Umweltbestimmungen und schnellere Reaktion auf Marktveränderungen.[7]

Im Bereich des Automotive Software Engineering können die Simulationsformen grob in drei Kategorien eingeteilt werden:

Software in the Loop (SiL)

Bei diesem Verfahren werden Softwarekomponenten in einer komplett simulierten Umgebung ausgeführt. Die Bezeichnung „in the Loop“ steht dabei für den Regelkreis (in sich geschlossener Wirkungskreis). Dieses Verfahren ist beschränkt auf Regelungsfunktionen.[8]

Hardware in the Loop (HiL)

Mit einem HiL-Prüfstand ist es möglich, die Funktionstüchtigkeit von elektronischen Steuergeräten zu testen. Dabei erhält das zu testende Gerät simulierten Input aus zuvor festgelegten Szenarien. Außerdem ist eine live-Steuerung des Fahrzeugs möglich.[8]

Vehicle in the Loop (ViL)

Bei den zuvor erwähnten Verfahren ist bereits ein hohes Maß an Sicherheit und Reproduzierbarkeit gegeben. Der nächste Schritt, der durch ViL ermöglicht wird ist, diese Vorteile mit realen Testfahrten zu kombinieren. Die Verbindung eines virtuellen Testszenarios mit Wahrnehmungen und Empfindungen bei der realen Fahrzeugbewegung bietet eine Chance, Fahrerassistenzsysteme effizient und mit hohem Sicherheitspotential zu entwickeln. Dieser Effekt wird durch ein „Optical see through Head Mounted Display“ erreicht, das durch transparente Darstellung die Wahrnehmung der echten Umgebung zulässt. So können Funktionen und Steuergeräte direkt in einem Fahrzeug erprobt werden. Die komplexe Materie erfordert spezielle Prüfgelände mit ausreichender Größe, so dass sämtliche Fahrsituationen realitätsnah und/oder unter extremen Bedingungen und Belastungen getestet werden können.[8]

2.3 Bilderkennung

Die Bilderkennung ist ein wesentlicher Teil der Computer Vision. Das Ziel ist es, eine Szene auf einer Abbildung zu analysieren, Objekte zu erkennen oder bestimmte Bereiche zu segmentieren. Durch die Vielzahl und Variabilität von Objekten, die in der realen Welt existieren, ist diese Aufgabe nicht durch einen simplen Vergleich zu lösen. Es gibt verschiedene Lösungsansätze, um diese Aufgabe zu bewältigen, wie das Suchen nach bestimmten, vorgegebenen Eigenschaften, die für das jeweilige Objekt stehen.[9]

Als Schöpfer der Computer Vision gilt Larry Roberts, der sich 1960 mit der Extraktion von Informationen dreidimensionaler Quader aus zweidimensionalen Bilddaten beschäftigte. In den folgenden Jahren begannen Forscher, sich mit Bildern der realen Welt auseinanderzusetzen. Erste Aufgaben bezogen sich beispielsweise auf die Kantenerkennung und Segmentierung von Bildern. Im Jahr 1978 veröffentlichte David Marr einen bottom-up-Ansatz, der über viele Jahre genutzt wurde. Der Ansatz besteht aus mehreren Stufen und beginnt mit einer Bildvorverarbeitung, die beispielsweise das Bild schwarz-weiß konvertiert und die Kanten erkennt. Anschließend werden high-level Techniken verwendet, um die gewünschten Informationen zu extrahieren.[10]

In den letzten Jahren gewannen maschinelle Lernverfahren auch in der Bilderkennung immer mehr an Bedeutung. Vor allem Convolutional Neural Networks (CNN) eignen sich zur Bilderkennung, da sie es ermöglichen, Daten in Matrixform zu verarbeiten. Dadurch können Objekte unabhängig von ihrer Position lokalisiert werden, was zur steigenden Beliebtheit von CNN im Einsatz der Bilderkennung geführt hat. Durch diese Entwicklungen hat sich auch der Arbeitsablauf verändert; Bilder werden meist direkt verwendet und keiner Vorverarbeitung unterzogen.[11, S. 576 ff]

Auch in der Automobilbranche spielt die Bilderkennung eine wesentliche Rolle. Von Spurhalteassistenten bis hin zum autonomen Fahren ist es wesentlich, dass die Umgebung mithilfe von Sensoren und auch Kameras genau erfasst werden kann. Bei solchen Systemen, die in Echtzeit eingesetzt werden, ist die zeitliche Komponente der Bilderkennung und -auswertung ebenfalls ein wichtiges Kriterium. Mit der Hilfe von künstlichen Neuronalen Netzen (NN) können hier besonders gute Ergebnisse erzielt werden.

2.4 Künstliche Neuronale Netze

Die ursprüngliche Idee für künstliche Neuronale Netze basiert auf dem Versuch, biologische Neuronale Netze zu imitieren und zu abstrahieren. Die Anfänge neuronaler Netze gehen auf den Neurophysiologen Warren McCulloch und den Logiker Walter Pitts zurück. Bereits im Jahr 1943 beschrieben diese unter „A Logical Caluculus of the Ideas Immanent in Nervous Activity“ ein neuronales Netz, das in der Lage ist, jede arithmetische und logische Funktion zu berechnen.[12] Das erste auch kommerziell erfolgreiche neuronale Netz wird 1959 von Marcian E. Hoff und Bernhard Widrow vorgestellt, „ADALINE“ oder auch „MADALINE“ („Multiple ADAptive LINear Elements“). Es wurde entwickelt, um Bits einer Telefonleitung vorherzusagen und war somit das erste NN, das für ein real existierendes wirtschaftliches Interesse erforscht wurde. Als Loss-

funktion schlugen Widrow und Hoff den bekannten Least-Mean-Squares-Algorithmus vor, der auch als Delta-Regel oder Widrow-Hoff-Regel bezeichnet wird und angibt wie weit eine Vorhersage von der Wahrheit entfernt ist.[13] Die Entdeckung einiger fundamentaler Mängel solcher einfachen neuronalen Netzwerke veranlassten Seymour Papert und Marvin Minsky Ende der sechziger Jahre die Kritikpunkte in „Perceptrons“ zu publizieren. Sie nahmen eine systematische Analyse der prinzipiellen Leistungsfähigkeit vor und verifizierten die funktionalen Grenzen insbesondere zweischichtiger neuronaler Feed-Forward-Netze. Diese kritischen Untersuchungen führten dazu, dass viele Wissenschaftler ihre Forschungsarbeit auf dem Gebiet der neuronalen Netze einstellten.[14] Im Jahr 1974 entwickelte Paul Werbos mit dem Backpropagation-Algorithmus ein Lernverfahren für neuronale Netze, das in der Lage war, die bestehenden Probleme zu beheben.[15] In den folgenden Jahrzehnten gewannen künstliche neuronale Netze in den Bereichen Wissenschaft und Technik zunehmend an Bedeutung.

Ein künstliches Neural Network (NN) wird durch die Verbindung von Neuronen mit dem Input erstellt. Abhängig von der Verbindung der Neuronen verhält sich das NN unterschiedlich. Es kann eine Vielzahl verschiedener Strukturen besitzen. Prinzipiell ist eine Unterscheidung in Feedforward Neural Network (FNN) und Recurrent Neural Network (RNN) möglich. Der größte Unterschied zwischen ihnen liegt in der Verbindung ihrer Neuronen. Bei RNN formen die Verbindungen der Neuronen einen gerichteten Kreislauf mit Zyklen und Schleifen, wohingegen FNN geradlinig nur in einer Richtung vorwärts durchlaufen werden. Im Folgenden wird nur auf die Funktionsweise von FNN eingegangen.[16, S. 63 ff]

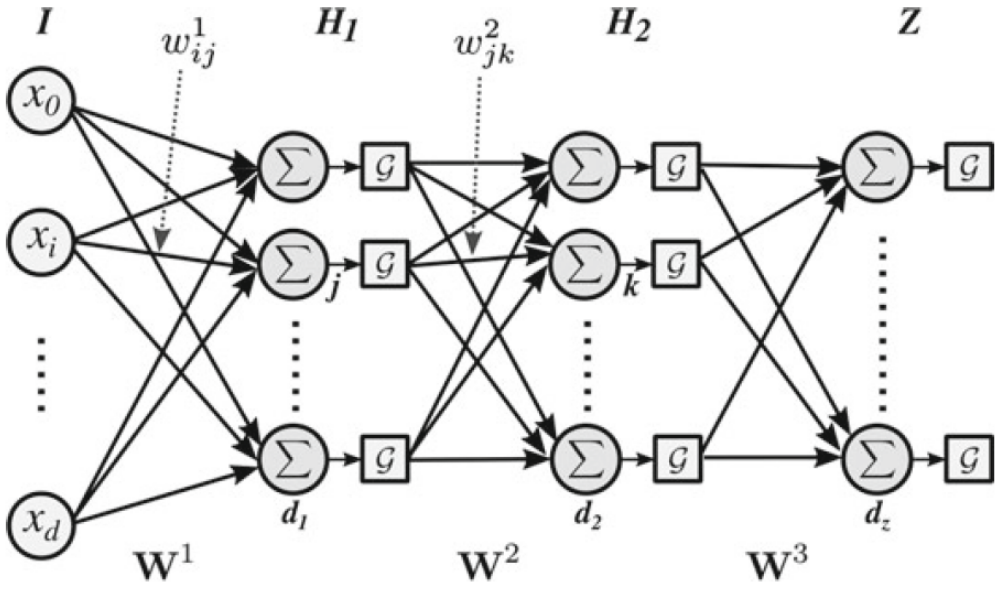


Abbildung 2.1: Feedforward Neural Network [16, S. 63]

Die generelle Struktur eines FNN ist in Abbildung 2.1 dargestellt. Zu sehen ist ein Netz mit einem Input Layer (I) und drei Layern mit Neuronen ($\sum$). Der letzte Layer wird Output Layer (Z) genannt und die zwei Layer zwischen dem Input- und Output Layer Hidden Layer (H_i). Die Anzahl der Neuronen pro Layer kann variieren; in der gegebenen Struktur wird die Anzahl durch d_j dargestellt. Jedes Neuron eines Hidden Layer oder Output Layer ist mit jedem des vorherigen Layer verbunden. In dem Beispiel sind es dann, zwischen den zwei Hidden Layern, $d_1 \times d_2$ Verbindungen.[16, S. 63 ff]

Normalerweise haben alle Neuronen eines Layers dieselbe Aktivierungsfunktion $\mathcal{G}$. Nicht ersichtlich in der Darstellung ist das Bias (b), das jedes der Neuronen besitzt. Mit W sind die Gewichte zwischen den Layern gekennzeichnet, die mit dem Input multipliziert werden. Somit kann das NN wie folgt beschrieben werden:

$$f(x) = \mathcal{G}(\mathcal{G}(\mathcal{G}(xW^1 + b^1)W^2 + b^2)W^3 + b^3) \tag{2.1}$$

Der Input-Vektor x wird also durch jeden Layer transformiert bis der Output Layer den transformierten d_2-dimensionalen Vektor klassifiziert. Die Besonderheit dieser NN ist, dass sie schon mit einem Hidden Layer jede kontinuierliche Funktion approximieren können. Diese Eigenschaft ist besonders für Klassifizierungsprobleme geeignet und kann auch Multiclass Klassifizierungen, die nicht linear trennbar sind, akkurat approximieren. Um solche Probleme zu lösen, ist es bei NN nicht erforderlich, eine Funktion zu entwerfen, es müssen nur die richtigen Hyperparameter gefunden werden. Hyperparameter sind von den Parametern zu unterscheiden, die während des Trainings gelernt/optimiert werden und müssen schon vorher festgelegt werden. Die Entscheidung, welche Hyperparameter benutzt werden, beeinflusst die Performance von dem NN. Um die optimalen Werte zu ermitteln, sind manuelle Anpassungen der Hyperparameter über viele Trainingsläufe vorzunehmen; erste Versuche können durch Erfahrungswerte beschlossen werden. Zu den Hyperparametern zählen unter anderem die Anzahl der Hidden Layer, die Anzahl der Neuronen pro Layer und die Aktivierungsfunktion. Das Vervollständigen der eingesetzten Funktion (2.1) wird dann durch das Trainieren des NN übernommen, dafür muss allerdings der Gradient der Lossfunktion berechnet werden. Der Vorgang zum Berechnen dieses Gradienten wird als Backpropagation bezeichnet.[16, S. 63 ff]

2.4.1 Lossfunktionen

Lossfunktionen werden dazu verwendet, den Fehler von der Vorhersage des NN im Vergleich zu dem Label zu berechnen. Sie dienen also als Maß, das angibt, wie gut die Vorhersagen des NN sind. Eine ihrer essenziellen Eigenschaften ist es, dass der Loss immer kleiner wird je besser die Vorhersagen sind; sonst kann das Netz nicht mittels Backpropagation optimiert werden. Beispiele für Lossfunktionen sind:

Mean Squared Error (MSE)

Um den MSE zu berechnen, wird der Unterschied zwischen Vorhersage und Label quadriert und anschließend durch die Anzahl der Daten geteilt[17]:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

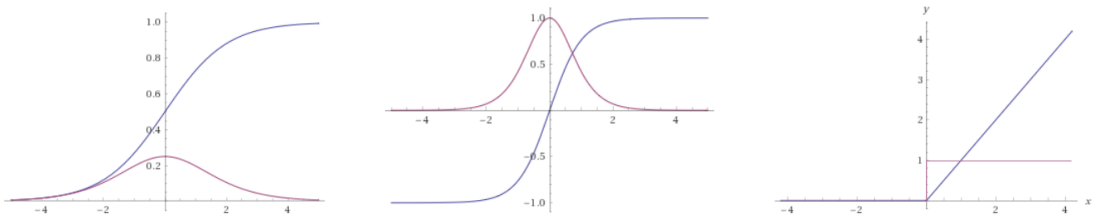
Cross-Entropy-Loss

Der Crossentropy Loss oder auch log-Loss wird zur Bewertung von Wahrscheinlichkeitswerten zwischen Null und Eins verwendet. Mit der Entfernung von der Prediction $\hat{y}$ zum Label y wird auch der Loss immer größer. Wird beispielsweise ein Pixel mit Label 1 als Spurmarkierung deklariert, so wird der Loss immer kleiner je höher die Wahrscheinlichkeit der Vorhersage steigt. Der Crossentropy Loss bestraft beide Arten von Fehlern (false positives und false negatives) aber besonders die false positives mit hohen Wahrscheinlichkeiten.[17]

Bei zwei Klassen: $loss = -(ylog(\hat{y}) + (1-y)log(1-\hat{y}))$
Bei mehr (N) Klassen: $loss = -\sum_{i=1}^{N} y_i log(\hat{y_i})$

2.4.2 Aktivierungsfunktionen

Die Aktivierungsfunktion überprüft den Output eines Neurons und entscheidet, ob es als aktiviert betrachtet wird. Damit das NN nicht linear wird, muss mindestens eine der genutzten Aktivierungsfunktionen nicht linear sein. Eine weitere wichtige Eigenschaft der Aktivierungsfunktion ist Differenzierbarkeit, da diese für die Anwendung von Gradient Descent Voraussetzung ist. Zudem ist es von Vorteil, dass sie das Mapping in der Nähe des Ursprungs approximiert, da die Gradient Descent Methode das Gewicht w und das Bias b normalerweise nahe Null initialisiert. Im Folgenden werden einige häufig eingesetzte Aktivierungsfunktionen genauer erläutert.[16, S. 71 ff]



(a) Sigmoid (b) tanh (c) Rectified Linear Unit (ReLU)

Abbildung 2.2: Aktivierungsfunktionen (blau) und ihre Ableitungen (lila)

Sigmoid

Die Sigmoid-Funktion (Abbildung 2.2a, Formel 2.2) ist eine S-förmige, nicht lineare Funktion und ihr Output ist immer zwischen Null und Eins. Ein entscheidender Vorteil ist ihre Differenzierbarkeit.[16, S. 71 ff]

$$\mathcal{G}_{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

$$\mathcal{G}'_{sigmoid}(x) = \mathcal{G}_{sigmoid}(x)(1 - \mathcal{G}_{sigmoid}(x)) \tag{2.3}$$

Die Funktion ist leicht zu verstehen, aber Probleme wie „vanishing gradients“, langsame Konvergenz und die Zentrierung auf 0.5 sind Gründe für ihren schwindenden Einsatz. Je mehr Layer zu dem NN hinzugefügt werden, umso mehr nähert sich der Gradient der Lossfunktion Null. Besonders in tiefen NN ist das ein Problem und macht das Training fast unmöglich. Die Sigmoidfunktion staucht einen großen Input-Raum zwischen Null und Eins, was zur Folge hat, dass große Veränderungen im Input nur sehr kleine Veränderungen im Output bewirken. Das ist für die Gradient Descent Methode ein Problem, da die Ableitung (Formel 2.3), wie in Abbildung 2.2a zu sehen ist, sehr klein wird. Bei **n** Hidden Layern führt das dazu, dass **n** kleine Ableitungen multipliziert werden, was den Gradienten exponentiell schrumpfen lässt, bis er vernachlässigbar klein ist. Eine mögliche Lösung für dieses Problem ist die Verwendung von anderen Aktivierungsfunktionen, wie beispielsweise ReLU, die dieses Problem nicht verursacht.[16, S. 71 ff]

Tanh

Die Tanh oder auch hyperbolic tangent Funktion (Abbildung 2.2b, Formel 2.4) ist eine skalierte Variante der Sigmoid-Funktion. Der Unterschied besteht darin, dass die Werte des Outputs zwischen minus Eins und plus Eins liegen.[16, S. 72 ff]

$$\mathcal{G}_{tanh}(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.4}$$

$$\mathcal{G}'_{tanh}(x) = 1 - \mathcal{G}_{tanh}(x)^2 \tag{2.5}$$

Eine wichtige Eigenschaft der Tanh-Funktion ist, dass sie negativen Input auch negativ darstellt und die Ausgabe um Null zentriert ist. Ein weiterer Vorteil besteht darin, dass sie beim Gradient Descent Algorithmus schneller konvergiert. Allerdings hat sie wie auch schon die Sigmoid-Funktion das Problem, dass sie beim Anstieg von $|x|$ schnell gesättigt ist, wodurch „vanishing gradients“ auftreten können. [16, S. 72 ff]

ReLU

Anders als die bisherigen Aktivierungsfunktionen eignet sich die ReLU-Funktion (Abbildung 2.2c, Formel 2.6) für tiefe NN, die mehrere Hidden-Layer haben. Das liegt daran, dass der Gradient bei dieser Aktivierungsfunktion nicht dazu neigt zu verschwinden. Sie wird wie folgt definiert: [16, S. 73 ff]

$$\mathcal{G}_{relu}(x) = max(0, x) \tag{2.6}$$

$$\mathcal{G}'_{relu}(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{2.7}$$

Eine weitere Eigenschaft der ReLU-Funktion ist, dass Neuronen während des Trainings „sterben“ können; sie geben immer Null zurück. Das kann durch die Gewichtsanpassungen verursacht werden. Dieses Phänomen bringt Vor- und Nachteile mit sich. Zum einen wird das NN recheneffizienter, da Ausgaben weggelassen werden; zum anderen kann die Genauigkeit des Netzes dadurch auch beeinträchtigt werden.[16, S. 73 ff]

Softmax

Die Softmax-Funktion (Formel 2.9) ist ähnlich wie die Sigmoid-Aktivierung auch eine logistische Funktion. Ihre Besonderheit liegt in der Eigenschaft, dass sie positive Vorhersagen ausgibt, deren Summe Eins ergibt.[17, S. 393 ff]

$$\mathcal{G}_{softmax}(X_i) = \frac{e^{X_i}}{\sum_{j=1}^{i} e^{X_j}} \tag{2.8}$$

$$\text{mit } X = \text{Vektor der Outputs} \tag{2.9}$$

2.4.3 Backpropagation

Backpropagation, auch Fehlerrückführung oder Backprop genannt, ist das Verfahren, das zum Trainieren von künstlichen NN verwendet wird. Ziel ist es, den Loss, der mit der Lossfunktion berechnet wurde, durch Gradientenabstieg zu minimieren, indem man die Parameter optimiert. Dafür wird der Gradient des Outputfehlers berechnet und zurück auf die verschiedenen Layer verteilt. Durch den komponentenweisen Aufbau des Modells kann der Gradient mit der Kettenregel einfach abgeleitet werden.[17, S. 395 ff] Im Folgenden wird der Ablauf des Backpropagation-Algorithmus basierend auf [18, S. 242 ff] genauer erläutert:

Für das Beispiel berechnet sich der Output des Netzes als eine lineare Kombination des Inputs x_i:

$$\hat{y}_k = \sum_i w_{ik} x_i \tag{2.10}$$

Zur Berechnung des Fehlers wird die Square-Error-Lossfunktion verwendet. Zusätzlich wird diese mit $\frac{1}{2}$ multipliziert, was die spätere Berechnung der Ableitung erleichtert. Der Fehler wird mit folgender Funktion berechnet:

$$E_n = \frac{1}{2} \sum_k (\hat{y}_{nk} - y_{nk})^2 \tag{2.11}$$

$$\text{mit } \hat{y}_{nk} = \hat{y}_k(x_n, w) \tag{2.12}$$

Der Gradient dieser Fehlerfunktion unter Beachtung des Gewichtes w_{ik} ergibt sich aus:

$$\frac{\partial E_n}{\partial w_{ij}} = (\hat{y}_{nj} - y_{nj}) x_{ni} \tag{2.13}$$

Jede Einheit (Neuron) des NN berechnet die gewichtete Summe der Inputs mit

$$a_j = \sum_i w_{ij} z_i \tag{2.14}$$

wobei z_i die Aktivierungsfunktion ist, die das Ergebnis weiter zur Einheit j sendet. Die Verbindung der beiden Einheiten wird mit w_{ij} gewichtet. Die Summe in 2.14 wird durch eine nicht-lineare Aktivierungsfunktion h transformiert und gibt die Aktivierung von z_j der Einheit j durch:

$$z_j = h(a_j) \tag{2.15}$$

Nun kann mithilfe der Kettenregel die Ableitung für E_n unter Beachtung des Gewichtes w_{ij} gebildet werden:

$$\frac{\partial E_n}{\partial w_{ij}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} \tag{2.16}$$

Mithilfe der Notation $\delta_j = \frac{\partial E_n}{\partial a_j}$ und einer Umformung der Formel 2.14 zu $\frac{\partial a_j}{\partial w_{ij}} = z_i$ können wir die Ableitung zu:

$$\frac{\partial E_n}{\partial w_{ij}} = \delta_j z_i \tag{2.17}$$

umformen. Aus dieser Formel wird ersichtlich, dass die benötigte Ableitung dadurch erzielt wird, dass der Wert von δ am Output-Ende des Gewichtes mit dem Wert von z am Input-Ende des Gewichtes multipliziert wird. Im Fall eines Bias ist $z = 1$. Um die Ableitung auswerten zu können, muss also der Wert von δ_j für jedes Neuron der Hidden- und Output-Layer berechnet und angewendet werden (Formel 2.18). Indem die Kettenregel erneut verwendet wird, ist es möglich, die δ's der Hidden-Layer zu berechnen:

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{2.18}$$

was für alle Einheiten k errechnet wird, zu denen Einheit j Verbindungen hat. Dabei kann k ein weiterer Hidden- oder Output-Layer sein. Mithilfe der Formeln 2.14, 2.15 und 2.17 kann die Gleichung 2.18 für die Backpropagation umgeformt werden:

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \tag{2.19}$$

Somit lässt sich der Wert von δ für ein bestimmtes Neuron dadurch berechnen, dass man die δ's der späteren Einheiten des NN rückwärts durch das Netz propagiert.

Neben dem klassischen Gradientenabstieg gibt es noch weitere Optimierungsfunktionen, die zum Minimieren des Fehlers (Loss) und Optimieren der Gewichte genutzt werden können. Dabei basieren sie alle auf dem Prinzip der Backpropagation. Die optimalen Parameter wurden gefunden, wenn der Loss nicht weiter minimiert werden kann.

2.4.4 Optimierungsfunktionen

NN werden prinzipiell mit dem Backpropagation-Algorithmus trainiert, allerdings wird dabei nicht immer der klassische Gradientenabstieg verwendet. Im Folgenden werden einige Erweiterungen des Algorithmus aufgeführt:

Stochastischer Gradientenabstieg

Der stochastische Gradientenabstieg berechnet die Gradienten in Bezug auf die individuellen Datenpunkte statt des gesamten Trainingssets. Es ist erwiesen, dass diese Version des Gradientenabstiegs besser generalisiert, aber auch nur sehr langsam konvergiert.[19]

Momentum

Momentum ist eine Methode, die versucht, die Konvergenz des stochastischen Gradientenabstiegs zu beschleunigen. Um das zu realisieren, wird ein Anteil der Richtung des vorangegangenen Schritts (von dem Aktualisierungsvektor) in den nächsten mit einbezogen, damit nicht nur der aktuelle Datenpunkt bei der Berechnung der optimalen Richtung beachtet wird. Neben dem schnelleren Erreichen des Minimums, werden auch die starken Schwankungen auf dem Weg dorthin vermieden.[20]

AdaGrad

Der AdaGrad-Algorithmus passt die Lernrate für die unterschiedlichen Parameter an und benutzt, anders als bisher, nicht immer dieselbe. Parameter, die zu häufig auftretenden Features gehören, bekommen kleinere Updates, als die, die selten auftreten. Somit werden schwache Gradienten gestärkt und starke abgeschwächt. Es wird also nur eine generelle Lernrate für das Training angegeben (meistens 0.01), die individuell angepasst wird. Ein Problem, das bei Adagrad auftreten kann ist, dass die Lernrate durch die Akkumulation aller vergangenen Gradienten immer kleiner wird. Zum Problem wird das, sobald sie vernachlässigbar klein geworden ist.[20]

Adadelta

Adadelta ist eine Erweiterung von Adagrad, die den monoton fallenden Lernraten entgegenwirken soll. Um dies zu erreichen, wird statt allen vergangenen Gradienten nur ein Anteil dieser verwendet. Es wird eine Art Fenster mit einer festgelegten Größe benutzt; nur die sich darin befindenden vergangenen Gradienten werden verwendet. Dadurch ist es möglich, dass die Summe insgesamt kleiner wird. Bei diesem Algorithmus muss keine generelle Lernrate angegeben werden.[20]

Adaptive Moment Estimation (Adam)

Adam ist eine erneute Erweiterung des stochastischen Gradientenabstiegs und berechnet angepasste Lernraten für jeden Parameter. Ähnlich wie beim Adadelta-Algorithmus werden die vorangegangenen Schritte mit einbezogen. Dafür behält Adam einen exponentiell verfallenden Durchschnitt der vergangenen Gradienten.[20]

2.4.5 Regularisierung

Eines der Probleme, das beim Training von NN auftreten kann, ist Overfitting. Dabei handelt es sich um ein Netz, das die Trainingsdaten zu gut abbildet, was einen negativen Einfluss auf die Beurteilung neuer Daten hat. Es gibt allerdings Möglichkeiten, mit denen man die Komplexität eines NN kontrollieren kann, um Overfitting zu vermeiden.

Diese Methoden werden unter dem Begriff der Regularisierung zusammengefasst. Im Folgenden werden einige dieser Regularisierungsmethoden genauer beschrieben.[18]

Early Stopping

Das Problem des Overfitting entsteht, wenn das NN immer mehr auf die Trainingsdaten optimiert wird und letztendlich nur noch diese gut beurteilen kann. Early Stopping ist in der Lage dem entgegenzuwirken, indem ein weiterer Validierungs-Datensatz verwendet wird. So kann das Training an dem Punkt gestoppt werden, an dem der kleinste Fehler auf den Validierungsdaten, nicht auf den Trainingsdaten erreicht wurde. Dadurch kann das NN besser generalisieren und erreicht auch bei neuen Daten bessere Ergebnisse.[18]

Dropout

Dropout vermeidet Overfitting dadurch, dass immer nur eine Untermenge des Netzes trainiert wird. Diese Wirkung kann erzielt werden, wenn mit einer zuvor festgelegten Wahrscheinlichkeit zufällige Neuronen eliminiert werden. Mit dieser Vorgehensweise soll erreicht werden, dass das NN sich nicht zu sehr auf zufällige Zusammenhänge konzentriert. Es ist üblich, dass die Dropout Layer direkt nach den Aktivierungslayern positioniert werden.[16]

Batch Normalization

Anders als bei den vorab erwähnten Regularisierungsmethoden tritt dieser Effekt bei der Batch Normalization nur als ein Nebeneffekt auf. Die eigentliche Aufgabe dieser Methode ist es, die Inputs der Layer zu stabilisieren. Um dies zu erreichen, wird ein Verfahren angewendet, das bei den Input Daten von NN ebenfalls üblich ist: Normalisierung. Das Ziel der Normalisierung ist es, die unterschiedlichen Daten in einem einheitlichen Verhältnis zu halten (zum Beispiel Null bis Eins), um Redundanzen, Inkonsistenzen und Anomalien zu vermeiden. Die Batch Normalization wird für jedes Batch des Datensatzes durchgeführt, und die Backpropagation der Gradienten wird mit den normalisierten Parametern vollzogen. Es konnte nachgewiesen werden, dass dieses Verfahren die Trainingszeit stark verringern kann und zusätzlich als Regularisierung dient, da ein gewisser Anteil an Noise (Rauschen) entsteht. In der Regel wird der Batch Normalization Layer zwischen dem Fully Connected/Convolutional Layer und der Aktivierungsfunktion positioniert.[16][19]

2.4.6 Convolutional Neural Networks

Convolutional Neural Networks (auch ConvNet genannt) gehören zu den FNN. Sie zeichnen sich dadurch aus, dass sie in der Lage sind, Input in Form einer Matrix zu verarbeiten. Diese Vorgehensweise ermöglicht es, als Matrix dargestellte Bilder als Input zu verwenden, ohne zuvor Flattening anwenden zu müssen. Dadurch können Objekte in einem Bild unabhängig von ihrer Position gefunden werden. Um diesen Effekt zu erreichen, nutzen CNN die Eigenschaft von Bildern, dass Pixel in der Regel stärker mit ihren unmittelbaren Nachbarn als mit weiter entfernten Pixeln korrelieren. Somit benötigen die Neuronen nur Informationen von dem betroffenen Pixel und seinen Nachbarn, um Informationen über eine bestimmte Region zu sammeln. Dieses Resultat wird durch Filter erreicht, die im Sliding-Window-Prinzip das Bild abtasten. Die Filter haben eine zuvor festgelegte Kernel-Größe, die besagt, wie viele Pixel auf einmal betrachtet werden (z. B. 2x2 oder 3x3 Pixel). Dabei bleiben die Gewichte für den Filter auf jeder Position im Bild gleich. Als Output erhält man eine zweidimensionale Matrix für jeden der Filter. Durch die Anwendung dieses Verfahrens ist es möglich, die Zahl der Parameter im Vergleich zu einem FNN drastisch zu reduzieren.[16, S. 85 ff]

Ein tiefes CNN besteht meistens aus mehreren Convolutional Layern. Jeder von diesen Layern hat eine Zahl von Bildern als Output, die der Zahl der Filter entspricht. Bei RGB-Bildern können die Filter auch zweidimensional sein, werden dann aber für jeden Farbkanal einzeln angewendet. Meist werden Convolutional Layer in Verbindung mit Pooling Layern eingesetzt, die die Ergebnisse der Convolution aggregieren, indem sie nur die stärksten Signale weitergeben.[16, S. 85 ff]

Padding

Es gibt zwei verschiedene Padding Varianten, die bei CNN verwendet werden können. Sie haben jeweils einen unterschiedlichen Einfluss auf die Output-Größe und die Abtastungen der Randpixel. Im Folgenden werden beide Varianten kurz erläutert.

Zero Padding oder auch **Same Padding** zeichnet sich dadurch aus, dass am Bildrand Nullen angefügt werden, so dass die Pixel im Randbereich genauso oft abgetastet werden, wie die in der Mitte des Bildes. Ein Nebeneffekt ist, dass die Bildgröße (Höhe x Breite) des Inputs gleich bleibt. Allerdings verändert sich die Zahl der Kanäle, wie zuvor beschrieben, zu der Anzahl der Filter, da es pro Filter eine Auswertung des Bildes gibt.[16, S. 140]

Valid Padding ist die Bezeichnung, wenn kein Padding angewendet wird. Das bedeutet, dass der Filter nur auf den Pixeln des Input-Bildes benutzt wird, auch wenn Pixel am Rand des Bildes dadurch seltener abgetastet werden. Dieses Vorgehen hat den zusätzlichen Effekt, dass der Output kleiner als der Input wird.[21] Die Output Dimension kann mit folgender Formel berechnet werden:

$$\text{output dimension} = \frac{\text{input width} - n}{s+1} * \frac{\text{input height} - n}{s+1} \tag{2.20}$$

Mit der Kernel-Größe $n * n$ und einer Schrittweite s, in der der Kernel über das Bild geschoben wird. Die Tiefe des Outputs entspricht wieder der Anzahl der Filter.

Max Pooling

Max Pooling wird typischerweise in Verbindung mit ConvNets verwendet. Es fasst die Feature-Aktivierungen von benachbarten Pixeln zusammen, wodurch das CNN Merkmale ortsunabhägig auf dem Bild erkennen kann.[22]

Um dies zu erreichen, wird ein Max-Filter verwendet, der jede $n * n$ Region des Inputs durch den maximalen Wert in diesem Bereich repräsentiert. Dadurch wird der höchste Aktivierungswert in jeder Region gewählt, was zu einem kleinen Grad an räumlicher Invarianz führt. Außerdem wird die Größe der Aktivierung um einen Faktor von n^2 verringert, wodurch die nächsten Layer weniger Parameter lernen müssen. Max Pooling wird in der Regel im Anschluss von Convolutional-Layern verwendet.[19]

2.5 Klassische Spurerkennung

Für den Anwendungsfall der Spurmarkierungserkennung ist es am naheliegendsten, zunächst die Methoden der klassischen Spurerkennung genauer zu betrachten. Die Spurerkennung verfügt bereits über viele wissenschaftliche und technologische Quellen, Tools, Lösungsansätze und Realisationen, da sie sowohl im Bereich der Fahrerassistenzsysteme (FAS) als auch des autonomen Fahrens essenziell ist. Das Einhalten der Spur beim Fahren ist ein wesentlicher Bestandteil eines funktionierenden Straßenverkehrs. Das Verlassen der Spur – ob beabsichtigt oder unbeabsichtigt – ist generell risikobehaftet. Die Zahl der durch Spurwechsel verursachten Unfälle ist sehr hoch. Eine akkurate Spurführung ist nicht nur bei Spurhalteassistenzsystemen von eminenter Bedeutung, sondern ist auch im Bereich des autonomen Fahrens von größter Wichtigkeit, wenn Abstand zum vorausfahrenden Fahrzeug, Spurhalten und Spurwechsel in gewissen limitierten Geschwindigkeitsfenstern automatisiert sind.

Ein wesentlicher Bestandteil der Funktionsfähigkeit solcher Systeme ist das genaue Erkennen der Spurmarkierung. Die Methoden, mit denen Wirtschaft und Wissenschaft daran arbeiten, die Herausforderung Spurerkennung und Spurverfolgung zu bewältigen, sind ebenso vielschichtig und komplex wie die bereits eingesetzten aktiven und passiven Systeme von unterschiedlichen Herstellern/Unternehmen. Im Folgenden werden verschiedene Methoden nach der Unterteilung von Nadra Ben Romdhane, Mohamed Hammami und Hanene Ben-Abdallah aus dem Paper „A Lane Detection and Tracking Method for Driver Assistance System“[23] genauer erläutert.

2.5.1 Modellbasierte Methoden

Modellbasierte Methoden benutzen wenige Parameter, um Spuren anhand ihrer Form zu repräsentieren. Es wird die Annahme getroffen, dass sie entweder gerade Linien oder parabolische Kurven sind. Das Erkennen der Spuren kann nur im Rahmen dieser zuvor festgelegten Muster erfolgen. Diese Herangehensweise bietet den Vorteil, dass sie deutlich robuster gegen Störfaktoren wie Regen oder Schnee ist. Ein Nachteil dieser Methode ist die fehlende Flexibilität, da nur bestimmte Spur-/Straßenformen erkannt werden können.[24] Modelllbasierte Methoden lassen sich in zwei Kategorien einteilen:

Parametrisch

Parametrische Methoden repräsentieren die Spurgrenzen mithilfe von wenigen Parametern. Sie sind robust, können aber nur Spuren erkennen, deren Form in dem Modell beschrieben wurde. Versionen dieser Methode sind zum Beispiel die Hough-Transformation oder die Nutzung von linearen Funktionen. Beide Verfahren eignen sich besonders für das Detektieren von geraden Spuren.[23]

Ein Beispiel für parametrische Methoden wird in dem Paper „Model-based Lane Detection and Lane Following for Intelligent Vehicles“ von Jianzhuang Wang, Youping Chen, Jingming Xie und Haiping Lin beschrieben. Um die Herausforderung der Spurerkennung zu meistern, haben die Autoren dynamisches Processing in Verbindung mit Hough Transform genutzt. Der Lösungsansatz besteht aus mehreren Schritten, beginnend mit dem Extrahieren von möglichen Spurmarkierungs-Kandidaten durch klassische Kantenerkennung. Im nächsten Schritt wird mithilfe von dem Dynamic-Programming Optimisierungs-Algorithmus (Algorithmus zum Ermitteln eines optimalen Pfades von einem Start- zu einem Endpunkt) der optimale Pfad der Spurmarkierungen ermittelt und entfernt schwache Spurmarkierungs-Kandidaten. Im letzten Schritt wird ein verbesserter Hough Transform Algorithmus auf die gefilterten Kanten vom Bild angewendet,

um die Schätzung der Spur festzulegen. Dabei werden nur die Spurmarkierungen, die die eigene Spur begrenzen, ermittelt. Dieses Verfahren eignet sich gut für Autobahnen und andere gerade Strecken, ist für Kurven allerdings ungeeignet.[25]

Explizit

Bei dieser Methode können verschiedene Straßentypen erkannt werden, allerdings ist das Segmentierungsergebnis stark von der Initialisierung des Modells abhängig. Diese Verfahren sind sehr anfällig für Unregelmäßigkeiten und haben lange Ausführungszeiten, wodurch sie für Echtzeit-Anwendungsfälle nicht die geeignetsten Methoden sind. Eine Technik für dieses Verfahren sind B-Snake Ansätze, die B-Splines nutzen, um die Spur-Struktur zu beschreiben. Dadurch können beliebige Formen mittels eines Sets von Kontrollpunkten beschrieben werden.[23]

In dem Paper „Lane detection and tracking using B-Snake“ von Yue Wang, Eam Khwang Teoh und Dinggang Shen wird eine Implementierung dieses Verfahrens beschrieben, die neben geraden auch parabolische Spurverläufe erkennen kann. Zudem ist diese Vorgehensweise robust gegen Schatten und Lichter und kann auch für 3D-Modelle genutzt werden.[24]

2.5.2 Feature-basierte Methoden

Feature-basierte Methoden lokalisieren die Spurmarkierungen unter Einsatz von low-level Features, wie Linien oder Kanten. Eine wichtige Voraussetzung für diese Verfahren sind folglich gut erkennbare Spurmarkierungen. Ein Nachteil dieser Methoden ist, dass es keine globalen Einschränkungen der Spurformen gibt, was zu Ungenauigkeiten führen kann.[24] Feature-basierte Methoden lassen sich in drei Kategorien einteilen:

Regelbasiert

Regelbasierte Methoden suchen im Bild nach Formen, ähnlich denen von Spurmarkierungen. Die Regeln, nach denen klassifiziert wird, werden zuvor festgelegt. Bei diesen Verfahren werden also Nutzer-definierte Kriterien verwendet, mit denen die Spurmarkierungen auf den Bildern ermittelt werden sollen. So werden beispielsweise Schwellwerte benutzt, um die Spurmarkierungen vom Hintergrund zu extrahieren. Hier kann intuitiv vorgegangen werden, indem überlegt und festgelegt wird, was Spurmarkierungen ausmacht.[23] Ein Beispiel für eine auf einem Regelset basierende Spurerkennung ist in dem 2006 erschienen Paper „Rule-based tracking of multiple lanes using particle filters“ von Stefan Vacek, Stephan Bergmann, Ulrich Mohr und Rüdiger Dillmann beschrieben.[26]

Tracking

Tracking Methoden entscheiden ausgehend von einem Startpunkt iterativ, welches nächste Segment geeignet ist, eine geforderte Struktur einzuhalten. Ein hierfür eingesetztes Modell sind Markovsche Ketten, stochastische Prozesse, die zur Untersuchung komplexer Systeme verwendet werden.[23]

Markov-Prozesse sind ein Zufalls-Prozess, dessen Zustände von einem in den nächsten wandern. Diese Prozesse können dann von einer Markov-Übergangsmatrix repräsentiert werden. Bei einem Markov-Prozess erster Ordnung beeinflusst nur der aktuellste Zustand die Verteilung des nächsten. In der Spurerkennung können diese Verfahren dafür genutzt werden, die richtigen Spursegmente als gewünschte Markierungslinien zu suchen.[27]

Supervised

Auch bei Supervised Trainingsmethoden wird anhand von Regeln klassifiziert. In diesem Fall werden die Vorschriften allerdings, basierend auf einem Trainings-Datenset, automatisch generiert. Anschließend wird dann nach Formen ähnlich denen, die trainiert wurden, gesucht und ist somit auf die gelernten Eigenschaften/Regeln beschränkt.[23]

Zu diesen Methoden zählen auch moderne Machine Learning-Ansätze, mit denen versucht wird, mithilfe von künstlichen Neuronalen Netzen Spurmarkierungen oder Spuren zu erkennen. Die neueren wissenschaftlichen Arbeiten, die sich mit der Thematik Spurerkennung beschäftigen, präferieren diese Herangehensweise.

3 Lösungsansätze

Nach gründlicher Recherche im Hinblick auf die Methoden der klassischen Spuerkennung ist aufgefallen, dass in den letzten zwei bis drei Jahren der Einfluss und Einsatz von Machine Learning-Verfahren stark zugenommen hat. Darum fiel die Wahl für diese Arbeit ebenfalls auf die Nutzung von CNN, die speziell für das Verarbeiten von Bildern entwickelt wurden. Entscheidend bei der Problemstellung und -lösung ist, dass es sich bei dem Input um Bilder handelt. In diesem Kapitel werden alle verwendeten Methoden, Frameworks und Voraussetzungen aufgeführt und beschrieben. Dazu gehört auch die Auswahl des Datensets, das essenziell für das Training von NN ist. Zudem wird hier der Workflow beschrieben, mit dem die Implementierungen vorgenommen wurden.

3.1 Datenset Auswahl

Da es im Rahmen der Abschlussarbeit nicht möglich ist, ein exklusives Datenset mit Labeln, die auf die Aufgabenstellung zugeschnitten sind zu erstellen, muss ein passendes öffentlich verfügbares Datenset gefunden werden. Die wesentlichen Kriterien hierfür sind die Anzahl der gelabelten Daten und die Qualität der Spurmarkierungs-Label. Die nachstehende tabellarische Übersicht listet die verschiedenen Datensets und ihre Eigenschaften auf.

Datenset	**Markierungsverlauf**	**Markierungen**	**Datenanzahl**
BDD100k[28]	Geraden und Bézier-Kurven	14 Arten	100.000
Unsupervised Llamas[29]	Segmentierung und Verlauf	keine	100.000
KITTI Road[30]	nur Spuren keine Spurmarkierungen	keine	579
TuSimple[31]	Polylines für max. 5 Markierungen	keine	3626
CULane[32]	cubic splines	keine	133.235
ApolloScape[33]	Segmentierung	36 Arten	110.000

Tabelle 3.1: Übersicht der öffentlichen Datensets

Aus der Tabelle 3.1 ist ersichtlich, dass von den Datensets nur zwei wirklich umfangreiche Label zu den Spurmarkierungen aufweisen. Die meisten verfügen lediglich über eine Segmentierung ohne Unterscheidung zwischen den verschiedenen Markierungstypen. Dadurch kommen nur Lane Segmentation von ApolloScape[33] und BDD100k[28] in Frage.

Das Datenset BDD100k von Berkeley DeepDrive[28], hat den Vorteil, dass viele Arten von Spurmarkierungen vorhanden sind. Unter anderem gibt es Label für durchgängige Linien, unterbrochene Linien, doppelte Linien, Bordsteinkanten und Fußgängerüberwege. Zudem gibt es eine farbliche Unterscheidung zwischen weiß, gelb und anderen Spurmarkierungsfarben. Ein weiteres positives Merkmal dieses Datensatzes ist die Beschreibung des Markierungsverlaufes, was die Möglichkeit bietet, ein Netz zu trainieren, das direkt den Verlauf lernen kann. Dadurch könnte ein zusätzlicher Schritt zur Ermittlung des genauen Verlaufs eventuell umgangen werden. Zu den Nachteilen von diesem Datenset zählen die unregelmäßige Reihenfolge der Label und der Umstand, dass nur eine begrenzte Anzahl von Markierungen für die linke- und rechte Spurmarkierungskante ein Label haben und dass das Strich-Lückenverhältnis teilweise nicht genau aufgeführt ist. Als Negativum ist anzuführen, dass die Spurmarkierungen bei Hindernissen nicht durchgängig verlaufen.[28]

Auch das Lane Segmentation Datenset von ApolloScape[33] bietet viele Vorteile. Die Label umfassen durchgängige-, unterbrochene- und doppelte Linien. Auch Sperrflächen, Haltelinien, Fußgängerüberwege und Markierungspfeile werden unterschieden. Das Spektrum der verschiedenen Farben umfasst weiße und gelbe Markierungen. Die Label sind alle in Form von Segmentierungsmasken bereitgestellt ohne Verlaufsdaten in Form von Kurven oder Ähnlichem. Ein weiterer Pluspunkt ist, dass die genauen Informationen zur Montage der zwei verschiedenen Kameras verfügbar sind. Auch die Reihenfolge der Bilder ist hier genau gekennzeichnet.[33]

3.2 Arbeitsumgebung und Frameworks

Von der Carmeq GmbH wurde eine DevCube Workstation zum Berechnen der Algorithmen zur Verfügung gestellt. Diese enthält vier Nvidia Grafikkarten der GeForce GTX TITAN X Serie mit 12GB GDDR5 (Dynamic Random Access Memory mit doppelter Grafikdatenrate vom Typ 5). Bei dem Prozessor handelt es sich um einen Intel i7-5960X mit 3.00GHz und 8 Kernen. Der Arbeitsspeicher hat eine Kapazität von 64GB.

Als Arbeitsumgebung steht JupyterLab[34] zu Verfügung, ein webbasiertes, interaktives Nutzerinterface des Jupyter Projektes. Zum Implementieren der verschiedenen Lösungsansätze wurden teilweise Jupyter Notebooks[34] verwendet, die es ermöglichen, Quellcode, Ausgaben und Kommentare komfortabel und übersichtlich zu gestalten; ansonsten klassische Python-Dateien. Die gewählte Entwicklungssprache ist Python3[35],

die im Bereich des Machine Learning weit verbreitet ist und über ein breites Spektrum an Frameworks verfügt. Die Jupyter-Umgebung unterstützt Python als Programmiersprache mit entsprechenden Kerneln.

Zur Versionsverwaltung wurde unter anderem GitLab[36] verwendet. Ein entscheidender Vorteil des GitLab sind die Möglichkeiten, Dokumentation und Planung der Arbeitsschritte dort übersichtlich durchführen zu können. Auch die Ausarbeitung von Meilensteinen und Unteraufgaben lässt sich darüber problemlos vornehmen; darüber hinaus ist der Zeitplan jederzeit verfügbar.

Für die Implementierung ist die Wahl auf Tensorflows[37] Implementierung der Keras Application Programming Interface (API)[38] gefallen. Keras ist durch sein konsistentes und leicht verständliches Interface geeignet, schnell Ergebnisse zu erzielen. Ein weiterer Vorteil ist, dass User problemlos benutzerdefinierte eigene Funktionen einbinden können. Die Entscheidung für Tensorflows Implementierung von Keras ermöglicht die Unterstützung von Tensorflow-spezifischen Funktionen. Nur so können TFRecords und TFData benutzt werden. TFRecords ist ein Format, mit dem Daten als binäre Datensequenzen gespeichert werden und kann in Form eines TFData-Datasets verwendet werden.[39]

Im Vergleich zum reinen Tensorflow ist der Einstieg in Keras unkomplizierter, da die Architektur einfacher strukturiert ist und eine eigene Netzwerkarchitektur schneller implementiert werden kann. Weitere Vorteile von Keras sind, dass viele Layer, Lossfunktionen und andere notwendige Komponenten bereits als fertige Funktionen zur Verfügung stehen und nicht selbst implementiert werden müssen. Die zeitliche Komponente war bei der Wahl des Frameworks ausschlaggebend, da der Bearbeitungszeitraum limitiert ist.

3.3 Workflow

Im Folgenden wird der Workflow zum Implementieren der Spurmarkierungserkennung dargestellt. Dieser ist für beide Lösungsansätze identisch und besteht aus jeweils drei Teilen. Im ersten Schritt müssen die Trainingsdaten entsprechend vorverarbeitet werden, damit sie zu den jeweiligen Architekturen passen. Zudem müssen sie in eine Form gebracht werden, damit es keine Unterschiede in den Ergebnissen zwischen mehreren Architekturen zum selben Ansatz gibt (Format und Pixelwerte sollten übereinstimmen). Dafür wird bereits in der Vorverarbeitung eine Unterteilung in Trainings-, Validierungs-

und Testdaten vorgenommen. Die Trainingsdaten werden genutzt, um die Gewichte für die Neuronalen Netze zu trainieren, was in mehreren Epochen durchgeführt wird. Am Ende jeder Epoche dient das Validierungsset dazu zu prüfen, wie sich der Loss verändert hat. Dafür eignet sich das Trainingsset nicht, da man sonst bei Problemen wie Overfitting nicht eingreifen könnte. Nach Abschluss des Trainings kann das Testset dazu verwendet werden, um Vorhersagen zu neuen Daten zu erstellen; für dieses Datenset werden keine Label benötigt.

Zur Handhabung der Daten eignet sich das TFRecord Format von Tensorflow [40], mit dem Daten als eine binäre Datensequenz gespeichert werden, was sich für Bilder gut eignet. Beachtet werden muss allerdings, die Bilder direkt im bytes-Format einzulesen, da sonst durch die benötigten Umwandlungen die Speichergröße der TFRecord-Datei schnell sehr groß wird. Tensorflow stellt dafür die Funktion $tf.gfile.FastGFile$ zur Verfügung. Für Trainings-, Validierungs- und Testdaten wird jeweils eine TFRecord-Datei benötigt, die im letzten Schritt der jeweiligen Datenvorverarbeitung erstellt wird.

Der darauf folgende Schritt besteht darin, verschiedene Netzwerkarchitekturen aus anderen Veröffentlichungen für den Anwendungsfall der Spurmarkierungserkennung zu testen. Dafür müssen diese Netze zunächst implementiert werden. Um für alle Trainingsläufe gleiche Voraussetzungen zu schaffen, wird immer eine identische Zusammenstellung an Callbacks[41](siehe Tabelle 3.2) verwendet. Callbacks sind Funktionen, die während des Trainings angewendet werden und verschiedene Funktionen erfüllen können. In Tabelle 3.2 sind die eingesetzten Callbacks und ihre jeweilige Funktion aufgelistet.

Bezeichnung	**Funktion**
EarlyStopping	Stoppt das Training, wenn der Validierungsloss sich nicht mehr verbessert.
ReduceLROnPlateau	Reduziert die Lernrate, wenn es keine Verbesserung mehr gibt.
ModelCheckpoint	Speichert das Netz oder die Gewichte nach jeder Epoche.
TensorBoard	Generiert eine log-Datei zur Visualisierung mit TensorBoard.

Tabelle 3.2: verwendete Callbacks

Der dritte Arbeitsschritt beinhaltet das Anpassen der Architekturen, um die bestmöglichen Ergebnisse zu erhalten. Dafür werden verschiedene Layer, Methoden und Parameter getestet, wie beispielsweise die Output-Aktivierungsfunktion, Lossfunktionen, Optimierungsfunktionen, Filteranzahlen für die CNN-Layer, Einfügen von Regularisierungsmethoden und weitere Optionen, die bei den einzelnen Lösungsansätzen genauer erläutert werden.

3.4 Landmark detection

Der erste Lösungsansatz, der getestet wurde, basiert auf der Landmark detection. Die Landmark detection wird dafür genutzt, prägnante Punkte auf einem Bild zu lokalisieren. Verbreitete Anwendungsfälle sind unter anderem das Erkennen von eindeutigen Gesichtspunkten, Körperteilen oder Gelenken, Organpositionen auf Computed Tomography (CT) Bildern, Sprachelementen und Umgebungsobjekte für beispielsweise die Navigation von Robotern. Auch wenn diese Ansätze in der Standard-Literatur zur Spurerkennung nicht genutzt werden, ähneln die Anforderungen stark der Aufgabe zur Spurmarkierungserkennung. So werden für Gesichter jeweils mehrere Positionen (x,y) ausgegeben und teilweise auch mit einer bestimmten Klassifizierung zur Gesichtshaltung versehen. Das Ziel dieser Masterarbeit ist es, sowohl Art als auch Verlauf/Position von Spurmarkierungen feststellen zu können, was durchaus den Anspruchsvoraussetzungen der Landmark detection entspricht.

Aufgrund der Übereinstimmung vieler Elemente der Aufgabenstellung mit denen der Landmark detection wird als erster Lösungsansatz ein solches Verfahren getestet. Da die meisten aktuellen Arbeiten in diesem Bereich für die Facial Landmark detection implementiert wurden, fiel die Wahl für eine Architektur auf die Veröffentlichung von Rajeev Ranjan, Vishal M. Patel und Rama Chellappa: „HyperFace: A Deep Multi-Task Learning Framework for Face Detection, Landmark Localization, Pose Estimation, and Gender Recognition“[42], da diese Übereinstimmungen zu einer Arbeit zur Erstellung von High Definition Karten für beispielsweise autonomes Fahren („LineNet: a Zoomable CNN for Crowdsourced High Definition Maps Modeling in Urban Environments“[43] von Dun Liang, Yuanchen Guo, Shaokui Zhang, Song-Hai Zhang, Peter Hall, Min Zhang und Shimin Hu) aufweist. Die Arbeit wurde 2017 publiziert und besitzt die meisten Übereinstimmunen mit der Zielsetzung. Zunächst müssen aber die Trainingsdaten in eine passende Form gebracht werden.

3.4.1 Datenvorverarbeitung

Aus der vorher erwähnten Analyse vorhandender Datensets 3.1 geht hervor, dass BDD-100K [28] sich am besten für diesen Ansatz eignet, da die Klassen und Punkte der Spurmarkierungen bereits zur Verfügung gestellt werden. Neben den vielen unterschiedlichen Klassen (siehe Tabelle 3.3) für die Spurmarkierungen unterscheiden sich die Label auch in der Anzahl der Koordinaten, mit denen sie beschrieben werden. Der Großteil der Markierungen hat einen geraden Verlauf und wird durch den Start- und Endpunkt dargestellt. Zusätzlich gibt es noch Markierungen, die von mehreren Punkten definiert

werden und teilweie auch solche, die von Bézierkurven beschrieben werden. Bei den Kurven gibt es jeweils den Start- und Endpunkt sowie weitere Kontrollpunkte, die die Kurve beschreiben.

Da sich in der Json-Datei mehr Label als nur die der Spurmarkierungen befinden, müssen zunächst die relevanten extrahiert und den Bildern zugeordnet werden. Dafür werden die Spurmarkierungsklassen und Koordinaten ausgelesen und zunächst jeweils in eine einzele Datei im CSV-Format geschrieben. Im Anschluss hat jedes Bild eine Label-Datei mit einer Zeile pro Spurmarkierung. Jede Zeile umfasst den Index der Klasse (siehe Tabelle 3.3) und danach die Koordinaten der Spurmarkierung. Die Reihenfolge der Spurmarkierungen ist in der Json-Datei nicht beachtet worden. Damit die Daten leichter zu handhaben sind, sollten sie zunächst sortiert werden. Die Spurmarkierungen wurden vom linken zum rechten Bildrand sortiert und die Koordinaten wurden alle vom unteren Bildrand zum oberen angeordnet.

Index	**Name**	**Beschreibung**
0	none	keine Spurmarkierung
1	road curb	Bordsteinkante
2	crosswalk	Fußgängerüberweg
3	solid, single white	durchgängige weiße Spurmarkierung
4	solid, double white	durchgängige, doppelte weiße Spurmarkierung
5	solid, single yellow	durchgängige gelbe Spurmarkierung
6	solid, double yellow	durchgängige, doppelte gelbe Spurmarkierung
7	solid, single other	durchgängige andere Spurmarkierung
8	solid, double other	durchgängige, doppelte andere Spurmarkierung
9	dashed, single white	unterbrochene weiße Spurmarkierung
10	dashed, double white	unterbrochene, doppelte weiße Spurmarkierung
11	dashed, single yellow	unterbrochene gelbe Spurmarkierung
12	dashed, double yellow	unterbrochene, doppelte gelbe Spurmarkierung
13	dashed, single other	unterbrochene andere Spurmarkierung
14	dashed, double other	unterbrochene, doppelte andere Spurmarkierung

Tabelle 3.3: Übersicht der Label

Da der Datensatz bis zu 49 Spurmarkierungen pro Bild, mit unterschiedlich vielen Koordinaten zur Verfügung stellt, ist es sinnvoll, die Label zunächst etwas einzugrenzen. Auch der Unterschied zwischen Punkten auf den Spurmarkierungen und Kontrollpunkten der Kurven könnte ein Problem darstellen, deswegen sollen zunächst nur die Punkte, die wirklich auf der Spurmarkierung liegen, gelernt werden. Die Anzahl der Spurmarkierungen pro Bild und Spurmarkierungsklassen werden zunächst beibehalten. Die Bilder, die weniger Spurmarkierungen aufweisen, werden mithilfe von padding (Auffüllen der fehlenden Einträge mit Nullen) auf dieselbe Größe (49 x 18) gebracht. Die Label

mit den One-Hot-Encodeten Spurmarkierungsklassen und Start- und Endkoordinaten werden zusammen mit den Bildern im TFRecord-Format, das während des Trainings verwendet wird, gespeichert.

3.4.2 Architektur

Der Aufbau des Hyperface-Netzes besteht aus mehreren Convolutional- und Pooling-Layern und basiert auf der Architektur von AlexNet[44], wie in Abbildung 3.1 in der oberen Hälfte zu sehen ist. Alle Fully-Connected-Layer wurden zunächst entfernt, da sie klassifizierungsspezifisch sind. Folglich werden sie nicht benötigt, da die Ziele Landmark Extrahierung und Positionsklassifizierung sind. Um die verschiedenen Aufgaben simultan zu erfüllen, werden verschiedene Layer vom Alexnet fusioniert. Mithilfe von weiteren Convolutional-Layern werden sie zunächst in dieselbe Dimension gebracht. Erst dann können die verschiedenen Stufen des AlexNet miteinander konkateniert werden. Die Lower-Layer-Features eignen sich besonders für die Landmark localisation und Pose-estimation. Für komplexere Aufgaben wie detection oder Klassifizierung sind die Higher-Layer-Features prädestiniert. Im Anschluss der Konkatenierung hat das Netz verschiedene Output-Layer für die verschiedenen Aufgaben (Gesicht, Landmarks, Position, Geschlecht). Durch die verschiedenen Netzausgänge ist es möglich, Klassifizierung und Regression mit einem Netz zu lösen.[42]

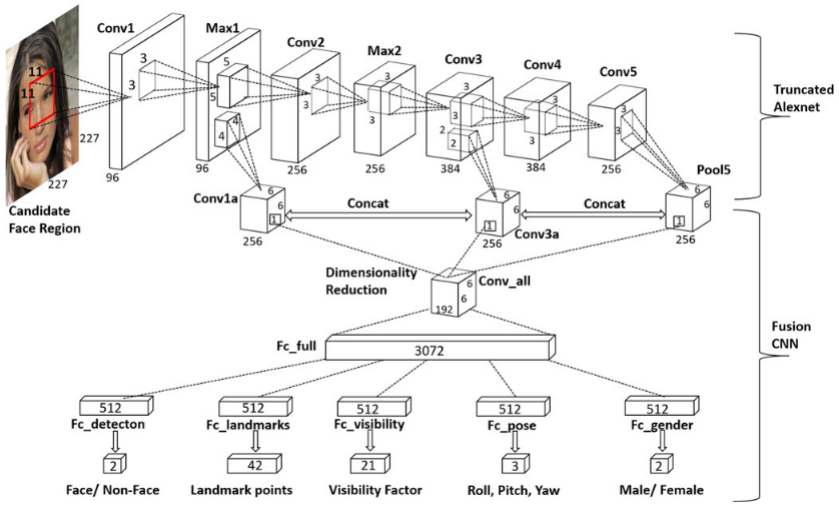


Abbildung 3.1: Hyperface Architektur[42]

Wie bereits am Anfang dieses Abschnittst 3.4.1 erwähnt wurde bestehen zwischen dieser Architektur und der von „LineNet: a Zoomable CNN for Crowdsourced High Definition Maps Modeling in Urban Environments“[43] von Dun Liang, Yuanchen Guo, Shaokui Zhang, Song-Hai Zhang, Peter Hall, Min Zhang und Shimin Hu Übereinstimmungen, die darauf hinweisen, dass dieser NN-Aufbau sich auch für Spurmarkierungen gut eignet. In der Arbeit sollten Spuren auf Kamerabildern erkannt werden, um daraus Daten

zur Gewinnung von High Definition Karten des Straßennetzes zu gewinnen. Die Spuren werden dabei ebenfalls anhand der Spurmarkierungen begrenzt. Ähnlich wie bei Hyperface basiert die Grundarchitektur auf einem bekannten CNN-Netz; in diesem Fall auf ResNet[45]. Der Layer für die Vorhersage der Spuren besteht wie auch bei Hyperface aus mehreren Ausgängen für die verschiedenen Aufgaben (siehe Abbildung 3.2). Bei LineNet handelt es sich dabei um eine Maske der eigenen beiden Spurmarkierungen, ihre Positionen, Richtung, Wahrscheinlichkeit, Abstand und Typ.

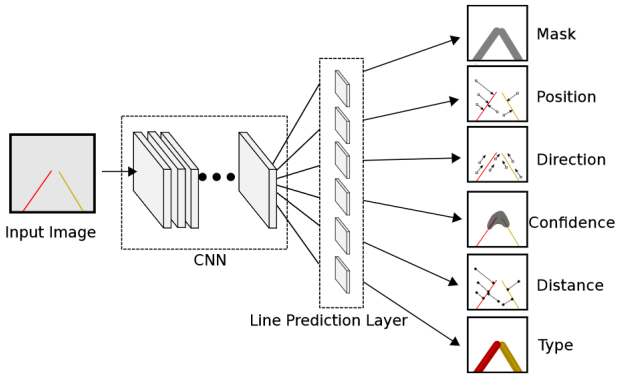


Abbildung 3.2: LineNet Architektur[43]

Bei diesem Architekturtyp besteht die größte notwendige Anpassung in der Wahl der richtigen Ausgänge. Zunächst sollen nur zwei Ausgänge für das Netz verwendet werden, einer für die Landmarks (Koordinaten der Spurmarkierungen) und einer für die Spurmarkierungstypen. Die Landmarks beschränken sich zunächst nur auf die Punkte, die direkt an den Spurmarkierungen liegen; Kontrollpunkte von Bézierkurven werden zunächst weggelassen. Für die Klassifizierung der verschiedenen Spurmarkierungstypen werden die Label One-Hot-encodet, um bessere Vorhersagen zu bestärken. Eine weitere Anpassung ist, dass nicht nur eine oder zwei Spurmarkierungen erkannt werden sollen sondern, bis zu 49.

3.4.3 Ergebnisse

Zunächst soll das Netz nur für die Punkte, die an den Spurmarkierungen liegen und ihren One-Hot-encodeten Klassen trainiert werden. Dafür werden zwei unterschiedliche Lossfunktionen für die zwei unterschiedlichen Aufgaben (Klassifizierung und Regression) verwendet. Für die Klassen wird der Crossentropy-Loss eingesetzt und für die Koordinaten der Euclidische Loss, der feststellen soll, wie weit die wirklichen Punkte von den vorhergesagten entfernt sind. Der Euclidische Loss (siehe Formel 3.1[42]) basiert auf der Euclidischen Distanz zwischen zwei Punkten und wurde im Hyperface Paper ebenfalls verwendet, wo er zusätzlich mit einem Sichtbarkeits-Faktor gewichtet wurde.[42]

$$\text{Euclidischer Loss} = \frac{1}{2N} \sum_{i=1}^{N} ((\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2) \tag{3.1}$$

Auf den ersten Blick schienen die Trainingsergebnisse für die Klassifizierung der Spurmarkierungstypen gut zu sein, da bei den Validierungsdaten eine Accuracy über 98% erreicht werden konnte. Nach genauerer Analyse der Vorhersagen fiel allerdings auf, dass das NN gelernt hat, alle Werte auf Null zu setzen, da es damit meistens richtig liegt. Auch bei den Vorhersagen zu den Koordinaten liegen fast alle Werte um die Null. Die Schlussfolgerung konnte nur sein: die vielen Nullen der Label stellen ein großes Problem dar. Die Datensätze, die wirklich über vierzig Spurmarkierungen enthalten, sind deutlich in der Unterzahl. Der Regelfall sind durchschnittlich sechs Spurmarkierungen pro Bild; dadurch sind viele Zeilen des Labels für die Spurmarkierungen nur Nullen. Die Überlegung, diese Problematik zu beheben, war, eine Lossfunktion zu verwenden, die nur die "richtig" und "falsch" klassifizierten Spurmarkierungen bewertet und die anderen Nullen, die korrekt als "keine" Spurmarkierung gelabelt wurden, nicht in den Loss miteinzubeziehen.

Mit boolschen True-False-Masken konnte erreicht werden, nur die "richtig" und "falsch" klassifizierten Spurmarkierungen zu berücksichtigen. Mit ihnen ließ sich leicht feststellen, wie viele Spurmarkierungen sich im Label und in der Vorhersage befinden. Für die weitere Lossberechnung wurden nur diese verwendet. Dadurch sollte erreicht werden, dass korrekt als "keine" Spurmarkierung eingeordnete Zeilen des Padding, keinen Einfluss auf die Performance haben.

Durch die Änderung der Lossfunktion konnte erreicht werden, dass die Accuracy- und Loss-Werte deutlich realistischer die Performance des NN widerspiegeln. Allerdings war es dem Netz nicht möglich, einen Loss unter hundert zu erreichen und auch die Accuracy konnte nur von 0.0059 auf 0.1981 verbessert werden. Bei genauerer Betrachtung der Vorhersagen war bei den Klassen keine wirkliche Verbesserung der Prognosen zu erkennen. Die Koordinaten entsprachen in keiner Hinsicht den erwarteten Werten, sodass anzunehmen war, dass ein wirklicher Lerneffekt nicht erzielt werden konnte. Als letzte Möglichkeit wurde in Erwägung gezogen, die Daten etwas zu vereinfachen und weiter einzugrenzen. Für den nächsten Versuch wurden nur noch die Start- und Endpunkte der Spurmarkierungen verwendet, was allerdings zu keiner Verbesserung des angestrebten Ergebnisses führte.

Nachdem mehrere Versuche zu keinen Verbesserungen in der Vorhersage geführt haben, war es naheliegend, erneut die Unterschiede zwischen der zu lösenden Aufgabe mit dem ursprünglichen Anwendungsfall zu betrachten. Bei der Landmark-Detection von Gesichtern gibt es den Vorteil, dass die Punkte auf einem Gesicht immer in derselben Reihenfolge und Anzahl auftreten. So hat jeder Punkt einen genauen Platz im Array der Label. Bei Spurmarkierungen gibt es in der Regel keine prägnanten Punkte und der Abstand der Punkte in den Labeln des Datensets war ebenfalls nicht gleichmäßig. Erschwerend und als zusätzliche Unregelmäßigkeit zu klassifizieren, kam hinzu, dass die Spurmarkierungen teilweise an beiden Rändern der Markierungen gelabelt wurden und teilweise nur durch eine Mittellinie. Auch lagen die Label der Markierungsränder nicht immer exakt auf den Spurmarkierungen, was es zusätzlich kompliziert macht, ein Muster zu erkennen.

Abschließend lässt sich zu diesem Lösungsanstz sagen, dass es mit den zur Verfügung stehenden Daten nicht möglich scheint, die Kombination von Klassifikation und Regression zielführend zu kombinieren. Mit Daten, die einheitlicher gelabelt und nach einem genauen Schema angeordnet sind, könnte dieser Ansatz möglicherweise trotzdem gute Ergebnisse liefern. Für diese Arbeit schien dieser Lösungsweg allerdings nicht erfolgversprechend, da kein passenderer Datensatz zur Verfügung stand.

3.5 End-to-End Spurerkennung

Für moderne Spurerkennungsverfahren werden oftmals CNN verwendet, mit denen sehr gute Ergebnisse in Echtzeit erzielt werden können. Bei diesen Verfahren wird die Klassifizierungs-Genauigkeit stark durch die Anzahl und Variabilität der Trainingsdaten beeinflusst.

In der Regel wird bei der End-to-End Spurerkennung der Zwischenschritt der Segmentierung vorgenommen; im weiteren Verlauf wird eine Kurve auf die segmentierten Spurmarkierungen appliziert. Diese Vorgehensweise wird in dem Paper „Towards End-to-End Lane Detection: an Instance Segmentation Approach“ von Davy Neven, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans und Luc Van Gool, dargelegt, wissenschaftlich untersucht und als Spurerkennungsarchitektur vorgeschlagen. Dafür haben die Autoren zwei verschiedene Netze genutzt, zum einen LaneNet, ein Netz zum Segmentieren der Spurmarkierungen und H-Net zum Anpassen der Verlaufskurven der Spurmarkierungen.[46]

Das LaneNet kombiniert die Vorteile der binären Spursegmentierung mit Clustering, um keine Beschränkungen in der Anzahl von Spurmarkierungen zu haben, die erkannt werden können. Im Schritt der Segmentierung wird zunächst für jedes Pixel festgelegt, ob es zu einer Spurmarkierung gehört oder nicht. Im Anschluss werden Cluster gebildet, um zusammengehörige Spurpixel mit derselben Lane-ID zu kennzeichnen. In der Ausgabe wird jedem Pixel eine Lane-ID zugewiesen, damit genau zugeordnet werden kann, zu welcher Spurmarkierung es gehört. Um das Clustering realisieren zu können, kommt Lane-Instance-Embedding zum Einsatz, da die meisten gängigen Segmentierungsmethoden für Spurmarkierungen nicht geeignet sind. Sie sind keine kompakten Objekte, die mit Boxkoordinaten repräsentiert werden können. Ziel ist es, durch die Clustering-Lossfunktion Ergebnisse von dem Instance-Embedding Branch zu erhalten. Gehören Pixel zur selben Spurmarkierung, ist der Abstand zwischen den Pixel-Embeddings gering, falls sie nicht zusammengehören, groß. Im Anschluss wird noch ein Polynom auf die unterschiedlichen Spurmarkierungen gelegt, was bedingt, dass zunächst eine Transformation zur Vogelperspektive vorgenommen wird, da dann Polynome niedrigerer Ordnung als in der Ego-Perspektive ausreichend sind. Durch dieses Verfahren gibt es keine Beschränkungen in der Anzahl der Spurmarkierungen, die erkannt werden können, da die Segmentierung hier keine Beschränkungen hat.[46]

Der zweite Lösungsansatz basiert auf solchen Methoden der End-2-End Lane detection. Diese Ansätze beruhen meist auf mehreren Schritten von der Segmentierung bis zur Erfassung des Verlaufs. Dafür ist es notwendig, zunächst eine geeignete Methode zum Segmentieren der Spurmarkierungen zu finden. Anders als bei dem zuvor beschriebenen Paper „Towards End-to-End Lane Detection: an Instance Segmentation Approach“[46] dargestellt, wird statt einer binären-, eine Segmentierung mit mehreren Klassen benötigt. Das ist unverzichtbar, damit weiterhin zwischen den verschiedenen Arten der Spurmarkierung unterschieden werden kann. Mit einer gut funktionierenden Segmentierung, die nur die Pixel der Spurmarkierungen klassifiziert, kann die bestehende Toolkette zum Generieren von Teststrecken aus Bilddaten bereits deutlich verbessert werden. Der Grund dafür ist, dass es von Vorteil ist, wenn die Klassen der Spurmarkierungen bereits feststehen und dass der Unterschied von Spurmarkierungen zu Straße und Hintergrund deutlich leichter erkennbar ist, wenn diese Schwarz sind. Als Datenset eignet sich für diesen Lösungsansatz am besten Lane Segmentation von ApolloScape[33]. Zunächst müssen die Daten allerdings für das Training vorbereitet werden.

3.5.1 Datenvorverarbeitung

Das Datenset Lane Segmentation von ApolloScape[33] besteht aus 79.864 Trainingsdaten, 33.790 Validierungsdaten und 885 Testdaten. Dargestellt werden diese nach dem Herunterladen in Form von Bildern (.jpg) und Masken (.png). Die verschiedenen Klassen der Masken sind durch unterschiedliche RGB-Farbtöne gekennzeichnet, wie in Tabelle3.4 zu sehen ist.

ID	Beschreibung	RGB	Farbe
0	Hintergrund / keine Spurmarkierung	(0,0,0)	
1	durchgängig, weiß	(70,130,180)	
2	durchgängig, gelb	(220,20,60)	
3	doppelt durchgängig, weiß	(128,0,128)	
4	doppelt durchgängig, gelb	(255,0,0)	
5	doppelt (durchgängig und unterbrochen), weiß	(0,0,60)	
6	doppelt (durchgängig und unterbrochen), gelb	(0,60,100)	
7	unterbrochen, weiß	(0,0,142)	
8	unterbrochen, gelb	(119,11,32)	
9	doppelt unterbrochen, weiß	(244,35,232)	
10	doppelt unterbrochen, gelb	(0,0,160)	
11	Haltelinie	(220,220,0)	
12	Haltelinie doppelt	(250,170,30)	
13	Haltelinie unterbrochen	(153,153,153)	
14	chevron / Sperrfläche, weiß	(102,102,156)	
15	chevron / Sperrfläche, gelbézier-Kurver	(128,0,0)	
16	Parkplätze, weiß	(128,64,128)	
17	Parkplätze, andere Farbe	(238, 232, 170)	
18	Zebrastreifen	(190,153,153)	
19	Wendepfeil	(0,0,230)	
20	Pfeil geradeaus, weiß	(128,128,0)	
21	Pfeil gerade und links	(128,78,160)	
22	Pfeil gerade und rechts	(150,100,100)	
23	Pfeil gerade, rechts und links	(255,165,0)	
24	Pfeil links	(180,165,180)	
25	Pfeil rechts	(107,142,35)	
26	Pfeil links und rechts	(201,255,229)	
27	Pfeil links und wenden	(0,191,255)	
28	Pfeil gerade und wenden	(51,255,51)	
29	Pfeil Zusammenführung	(250,128,114)	
30	Pfeil geradeaus, gelb	(127,255,0)	

Tabelle 3.4: Übersicht der Label - Teil 1

ID	Beschreibung	RGB	Farbe
31	Bremsschwelle	(255,128,0)	
32	Raute / diamond	(0,255,255)	
33	Parkverbot (Rechteck)	(178,132,190)	
34	sichtbare alte Markierungen	(128,128,64)	
35	andere Markierungen	(102,0,204)	

Tabelle 3.5: Übersicht der Labels - Teil 2

Zum Training müssen diese Masken zunächst in eine geeignete Form gebracht werden. Ziel ist es, zwei verschiedene Labelarten zu erhalten, zweidimensionale, binäre Darstellungen der Bilder mit Nullen für den Hintergrund und Einsen für Spurmarkierungen und mehrdimensionale Matritzen, die in jeder Dimension ein Label in binärer Form darstellen. Die erste Variante lässt sich einfach realisieren, indem man alle von schwarz abweichenden Farben der Label zu weiß ändert. Für die binäre Darstellung aller Klassen wird zunächst jeder Klasse eine Zahl zugewiesen (0 bis 27) und eine zweidimensionale Matrix des Bildes erstellt. Im Anschluss lässt sich mit dem Befehl $tf.oneHot(mask, depth = NumberOfClasses)$ der Tensorflow-Bibliothek problemlos die gewünschte Maske erstellen.

Da die Rechenleistung, die zur Verfügung steht, nicht unbegrenzt ist, müssen die Bilder noch verkleinert werden. Nach einigen Versuchen fiel die Wahl auf 564x451 Pixel für die Masken und Bilder. Die Verkleinerung wurde erst nach der Erstellung der Masken vorgenommen, da sonst bei den Labeln RGB-Werte aufgetreten wären, die nicht zu den Labelwerten gehören. Nach der Verkleinerung bestehen die Label-Masken aus Werten von Null (nicht diese Klasse) bis Eins (diese Klasse); an den Übergängen gibt es auch Werte dazwischen. Die Übergangswerte wurden auch für das Training beibehalten, um keine weiteren Informatinsverluste zuzulassen.

(a) Image (b) label (c) label binär

Abbildung 3.3: Lane Segmentation Datenset von ApolloScape[33]

In Abbildung 3.3 sind die verschiedenen Versionen des Bildes zu sehen. Links ist die Originalaufnahme, in der Mitte die Maske mit der Unterscheidung zwischen den Markierungsklassen und rechts die binäre Bildmaske.

3.5.2 Segmentierungs-Architekturen

Bevor sich eine genaue Wahl treffen lässt, muss eine geeignete Semantische Segmentierung gefunden werden, die mögichst gute Ergebnisse liefert. Zu gängigen und erfolgversprechenden Segmentierungs-Architekturen zählen unter anderem altbewährte wie U-Net und SegNet, aber auch neuere Ansätze wie zum Beispiel das DeepLab, dass regelmäßig verbessert und erweitert wird. Um eine fundierte Entscheidung treffen zu können, ist es sinnvoll, zunächst verschiedene Varianten mit dem Datenset zu trainieren, um die Ergebnisse auswerten zu können. So lässt sich am besten herausfinden, welche Segmentierung sich besonders für Spurmarkierungen eignet. Im Anschluss werden die Ergebnisse mehrerer Segmentierungsansätze an erster Stelle nur für die binäre Segmentierung verglichen.

U-Net

U-Net ist ein Segmentierungsnetz, das 2015 in dem Paper „U-Net: Convolutional Networks for Biomedical Image Segmentation“[47] von Olaf Ronneberger, Philipp Fischer und Thomas Brox veröffentlicht wurde. Ursprünglich wurde es speziell für die Segmentierung in der Medizin entwickelt, konnte aber aufgrund seiner guten Ergebnisse auch in anderen Bereichen der Segmenetierung eingesetzt werden. Es basiert auf einer Encoder-Decoder-Architektur (siehe Abbildung 3.4), bei der zunächst der Kontext erfasst wird und im Zuge der Vergrößerung eine präzise Lokalisierung ermöglicht wird. Um Objekte überall auf dem Input-Bild erkennen zu können, wurden Convolutions verwendet. Die ursprüngliche Implementierung wurde mithilfe des Caffe-deep-learning-Framework [48] vorgenommen. Zum Messen der Ergebnisse diente die Intersection over Union (IoU) Metrik; diese wird zum Messen von Ergebnissen in der Objekterkennung genutzt. Bei den ursprünglich verwendeten Datenset PhC-U373 konnte eine IoU von 0.9203 erreicht werden.[47]

Die linke Hälfte der Architektur (siehe Abbildung 3.4) ist der Encoder und wurde in der originalen Veröffentlichung als „contracting path“ bezeichnet. Dieser Teil des Netzes ist wie ein typisches CNN-Netz aufgebaut und hat 3x3 same-padded Convolutions gefolgt von jeweils einer ReLU Aktivierungsfunktion und einem maxpooling-Layer zum Downsampling. Die Decoder-Hälfte vom Netzwerk (auch „expansive path“), sichtbar auf der rechten Bildhälfte, vergrößert die Feature Map wieder mithilfe von Upsampling. Durch Verkettungen (concatenations) mit den entsprechenden, gleichgroßen vorigen Convolution-Stufen soll dem Informationsverlust aus vorigen Stufen entgegengewirkt werden. Der Output hat dieselben X/Y-Koordinaten wie das ursprüngliche Input-Bild und eine weitere Dimension für die Anzahl der Klassen.[47]

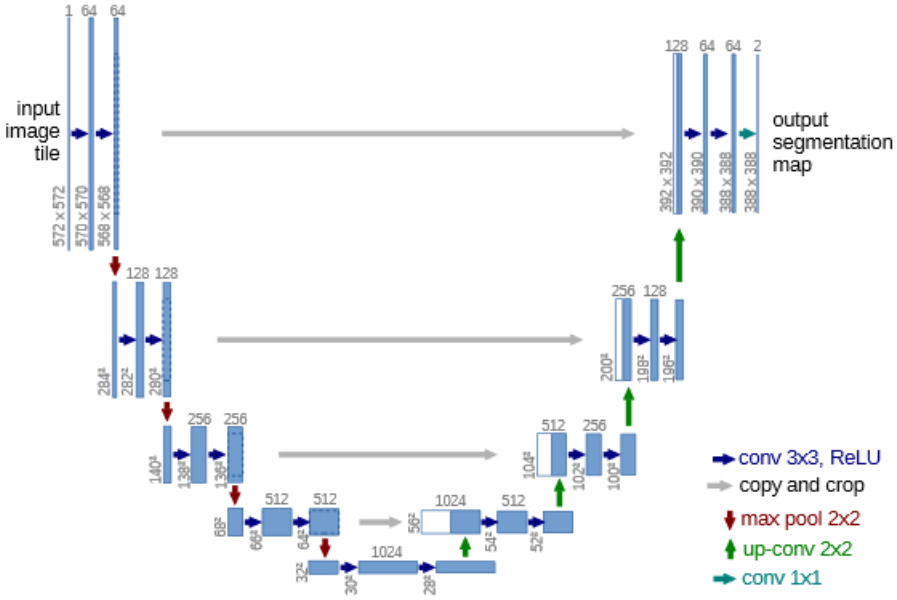


Abbildung 3.4: U-Net Architektur[47]

Trainiert wurde U-Net ursprünglich mit der Stochastic Gradient Descent Implementierung des Caffe[48] Frameworks. Der Anwendungsfall der Zellsegmentierung machte es zudem erforderlich, eine gewichtete Lossfunktion zu verwenden, um sich berührende Objekte voneinander klar trennen zu können.[47]

SegNet

SegNet wurde 2016 in unter dem Titel „SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation“[49] von Vijay Badrinarayanan, Alex Kendall und Roberto Cipolla veröffentlicht. Ähnlich wie U-Net handelt es sich hier ebenfalls um eine Encoder-Decoder-Architektur (siehe Abbildung 3.5), basierend auf Convolutions. Das Decoder-Netz erfüllt die Aufgabe, die Featuremaps des Encoder-Netwerks auf die volle Bildgröße abzubilden.[49]

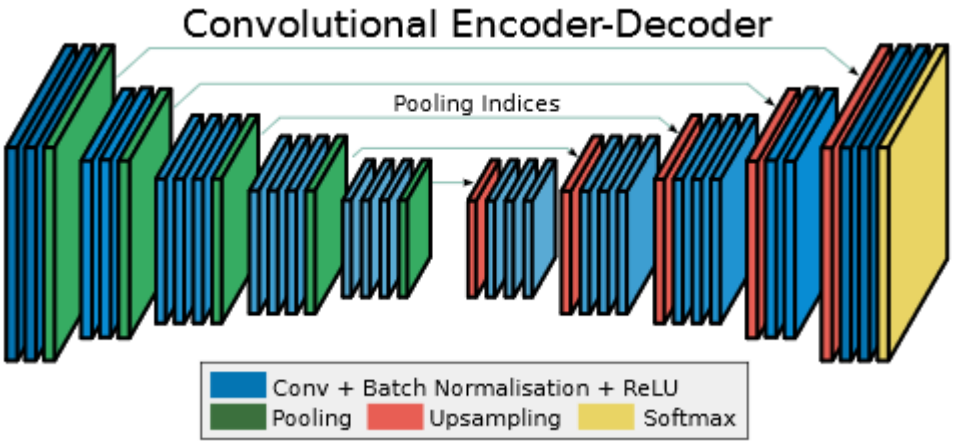


Abbildung 3.5: SegNet Architektur[49]

Das Encoder-Netzwerk von SegNet entspricht dem CNN VGG16[50] für Bilderkennung und Klassifizierung, lediglich die Fully-Connceted-Layer am Ende wurden nicht verwendet, da diese für die Segmentierung nicht benötigt werden. Auf jeden Convolutional Layer folgt jeweils eine Batch Normalisation und eine ReLU Aktivierungsfunktion. Die Output-Feature map wird am Ende mit der Softmax Aktivierungsfunktion in den Wertebereich zwischen Null und Eins transformiert. Das Ergebnis ist dann eine pixelweise Klassifizierung des Bildes, die Wahrscheinlichkeiten zur Klassenzugehörigkeit von jedem Pixel ausgibt. Somit ist der Output ein Bild mit K Kanälen, wobei K die Anzahl der verschiedenen Klassen ist. Für das Training vom SegNet wurden in der ursprünglichen Implementierung Stochastic Gradient Descent als Optimierungsfunktion und Crossentropy als Lossfunktion verwendet.[49]

Die Unterschiede zum U-Net liegen unter anderem in der Anzahl der verwendeten Layer. Zudem transferiert SegNet nicht die gesamte Feature map zu den entsprechenden Decodern, sondern die Pooling Ergebnisse und verknüpft diese mit den vergrößerten Feature maps. Dadurch wird weniger Spreicher benötigt.[49]

Deeplab

Im Jahr 2015 wurde vom Google Research Department eine alternative Encoder-Decoder-Architektur veröffentlicht, die Atrous Convolutions verwendet. In dem Paper „Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs“[51] von Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy und Alan L. Yuille werden diese erstmals vorgestellt. Sie bieten die Möglichkeit, größere Filter zu verwenden, ohne dass die Zahl der Parameter oder die Anzahl der Berechnungen steigt.

Durch die wiederholte Verwendung von Max-Pooling und Downsampling entstehen stark verkleinerte Feature-Maps mit deutlich reduzierter räumlicher Auflösung. Um dies zu umgehen, haben die Autoren den downsampling-Operator von den letzten Max-Pooling-Layern entfernt und stattdessen ein Upsampling vorgenommen um Feature-Maps mit höherer Abtastrate zu erhalten. Dafür wurde eine Methodik der Signalverarbeitung namens „algorithme à trous“ benutzt, bei der das Upsampling durch das Einfügen von Löchern zwischen den sogenannten Filter-Taps, die von Null abweichen, erfolgt. Bei Filter-Taps handelt es sich bei einem bespielsweise 2x2 großen Filter um die 4 “Taster” des Filters. Mithilfe der Atrous Convolutions lässt sich das Sichtfeld der Convolutions vergrößern, wofür sie einen zusätzlichen Parameter für die Rate besitzen. Dieser Parameter gibt an, wie viele Lücken in dem Sichtfenster verwendet werden. Bei einer Rate von eins, gibt es keine Lücken, womit es sich um die normalen Convolutions

handelt, die bereits bekannt sind. Wählt man allerdings eine größere Rate, zum Beispiel zwei, wird das Sichtfenster auseinandergezogen und nur jedes zweite Pixel wird von dem Fenster erfasst. Nun können also mit derselben Anzahl von Parametern größere Bereiche des Bildes erfasst werden. Mathematisch werden die Atrous Convolutions $y[i]$ für eindimensionale Signale $x[i]$ mit einem Filter $w[k]$, einer Länge K und einer Rate r wie folgt definiert[52]:

$$y[i] = \sum_{k=1}^{K} x[i + r * k]w[k] \tag{3.2}$$

Theoretisch wäre es mithilfe der Atrous Convolutions möglich, die Bildauflösung konstant zu halten und kein Downsampling zu betreiben. Da dies aber trotzdem zu rechenaufwändig wäre, haben die Autoren des Papers sich für eine hybride Version entschieden, die einen Kompromiss zwischen Effizienz und Genauigkeit darstellt.

Nachfolgend wird auf die verschiedenen Versionen von Deeplab eingegangen, die über einen mehrjährigen Zeitraum entstanden sind[53]:

Deeplabv1

Die ursprüngliche Intention von Deeplab war es, bekannte Probleme der Semantischen Segmentierung zu lösen. Dazu zählten die verringerte Feature-Auflösung und die verringerte Lokalisierungsgenauigkeit aufgrund der Invarianz von tiefen CNN.

Ersteres lässt sich durch die Benutzung von Atrous Convolutions lösen. Um die verringerte Lokalisierungsgenauigkeit aufgrund von Invarianz zu verhindern, nutzten sie ebenfalls eine alternative Methode, die die Fähigkeit, kleine Details zu erkennen, deutlich verbessert. Diese Methode beinhaltet den Einsatz von Conditional Random Field (CRF), die bereits zum Abschwächen von Segmentierungsmaps mit viel Noise (Rauschen) verwendet werden. Sie maximieren typischerweise die Label-Übereinstimmung von Pixeln, die sich ähnlich sind. Die Hauptfunktion von CRFs besteht darin, die falschen Vorhersagen von schwachen Klassifizierern zu bereinigen, wodurch sie die Lokalisierung potentiell verbessern könnten. Allerdings schaffen sie es nicht, notwendige feine Strukturen zu erkennen. Um diese Limitierungen zu durchbrechen, haben sie Fully-Connected-CRFs[54] verwendet, die Details mit scharfen Kanten erkennen können und trotzdem und zusätzlich langfristige Abhängigkeiten berücksichtigen.[52]

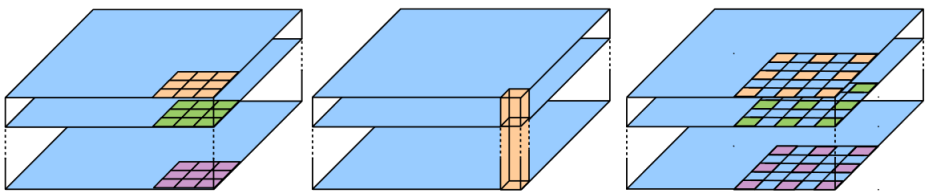
Die Deeplabv1 Architektur besteht aus einem klassischen CNN, gefolgt von ein bis zwei Atrous Schichten. Anschließend wird mittels bilinearer Interpolation die ursprüngliche Auflösung wiederhergestellt und die Ergebnisse werden mithilfe von Fully Connected CRF verbessert. Ab Deeplabv3 werden die CRF nicht mehr verwendet, da sie als Post-Processing zählen, wodurch Deeplabv1 und Deeplabv2 keine End-to-End Lösungen sind.[53]

Deeplabv2

Die Herausforderung, die mit dieser Erweiterung vom Deeplab behoben werden sollte, ist die Existenz von Objekten in mehreren Maßstäben. Dafür empfehlen die Autoren ein vom Spatial Pyramid Pooling inspiriertes Verfahren zum Resampling gegebener Feature-Layer vor den Convolutions. Indem das Bild mit Filtern, die unterschiedliche Sichtfelder haben, untersucht wird, können Objekte in mehreren Maßstäben erkannt werden. Anstatt die Features alle zu resamplen, wurden mehrere parallele Atrous Convolutional Layer mit unterschiedlichen Sampling Raten verwendet und fusioniert; diese Technik nennen sie Atrous Spatial Pyramid Pooling (ASPP).[52]

Deeplabv3

Die Entwicklung von Deeplabv3[55] basierte auf der Notwendigkeit, noch schärfere Objektkanten erkennen zu können. Die Encoder-Decoder-Architektur wurde um sogenannte „depth-wise separable convolutions“ erweitert, die die Rechenleistung erhöhen. Dafür wurden standardmäßige Convolutions zu Depthwise Convolutions faktorisiert, gefolgt von punktweisen Convolutions. Die Depthwise Convolutions führen eine Spacial (räumliche) Convolution für jeden Input Channel separat durch, die darauf folgende punktweise Convolution kombiniert den Output der Depthwise Convolution. Aus dieser Vorgehensweise kombiniert mit Atrous Convolutions resultieren „Atrous Separable Convolutions“(siehe Abbildung 3.6), die die Berechnungskomplexität verringern.[55]



(a) Depthwise conv. (b) Pointwise conv. (c) Atrous depthwise conv.

Abbildung 3.6: Atrous Separable Convolutions[55]

Deeplabv3+

Bei Deeplabv3+ handelt es sich um eine Erweiterung vom Deeplabv3, bei der ein weiteres Decoder-Modul eingefügt wird, um die Segmentierungsergebnisse vor allem an den Objektkanten weiter zu verfeinern. Für den Encoder wurde die aus ResNet[45] basierende Grundarchitektur gegen Aligned Xception[56] ausgetauscht. Diese haben sie allerdings modifiziert, indem die Max Pooling Operationen gegen depth-wise separable convolutions ausgetauscht wurden.[53]

Der Encoder gibt als Output eine um den Faktor 16 verkleinerte Feature-Map zurück. Anstatt diese direkt mit bilinearem Upsampling mit Faktor 16 zu vergrößern, werden die encodeten Features zunächst um den Faktor 4 erhöht und mit den entsprechenden low-level Features des Encoders verkettet. Vor der Verkettung wird auf die low-level features allerdings noch eine 1x1 Convolution angewendet, um die Anzahl der Channel zu verringern. Nach der Verkettung werden noch weitere 3x3 Convolutions zum Einsatz gebracht, und die Features werden erneut um den Faktor 4 erhöht, wodurch sie wieder das Ausgangsformat haben.[53]

Für die weitere Bearbeitung wurde nur mit Deeplabv3+ gearbeitet.

4 Ergebnisse

In diesem Kapitel sind die Ergebnisse der Abschlussarbeit dargelegt. Zunächst wird auf die spezifischen Trainingsergebnisse der verschiedenen Segmentierungsarchitekturen eingegangen und wie diese entstanden sind. Anschließend wird das Postprocessing beschrieben, das noch erforderlich ist, um die gewünschten Teststrecken zu erhalten.

4.1 Trainingsergebnisse

Das Training der unterschiedlichen Architekturen wurde zunächst nicht mit allen 36 Klassen, die ApolloScape[33] zur Verfügung stellt, durchgeführt. Es sollte zunächst getestet werden, wie gut sich die Methoden zum Klassifizieren von Spurmarkierungen allgemein eignen. Dafür wurden die Label-Bilder in zweidimensionale Masken umgewandelt, die nur zwischen “0 - keine Spurmarkierung” und “1 - Spurmarkierung” unterscheiden.

Die verschiedenen Segmentierungsarchitekturen wurden im Verlauf der Tests angepasst. Es wurden Dropout, Batch Normalization und Early Stopping zur Regularisierung integriert, sofern diese nicht Teil der ursprünglichen Implementierung waren. Zudem wurde die Filteranzahl der verschiedenen Convolutions variiert und so abgestimmt, dass sie sich von Ebene zu Ebene staffeln. In der ersten Ebene wird die initiale Filteranzahl direkt verwendet (Filteranzahl * 1), in den nächsten wird diese immer wieder verdoppelt bis zum Teil des Decoders, wo diese nach und nach wieder halbiert wird. Als Aktivierungsfunktionen wurden ReLUs verwendet, mit Ausnahme des Output Layers. Dort wurden Sigmoid und Softmax getestet; beide eignen sich für eine Klassifizierung zwischen Null und Eins, wie sie für die Aufgabenstellung benötigt wird. Eine weitere Möglichkeit, mit der die Ergebnisse verbessert werden sollten, waren die Loss- und Optimierungsfunktion. Auch bei der Lernrate und Batchgröße wurde getestet, mit welchen Werten die bestmöglichen Ergebnisse erzielt werden konnten.

Im Anschluss wurden alle Ergebnisse mit den folgenden Evaluationsmethoden, ermittelt mithilfe der Validierungsdaten, miteinander verglichen und bewertet. Nur die besten Kombinationen der drei Architekturen wurden anschließend für alle 36 Klassen trainiert.

Crossentropy Loss

Der Crossentropy Loss (siehe 2.4.1) wird zur Bewertung von Wahrscheinlichkeitswerten zwischen Null und Eins verwendet. Mit der Entfernung von der Prediction $\hat{y}$ zum Label y wird auch der Loss immer größer. In Keras existiert eine Implementierung dieser Lossfunktion, die verwendet werden kann. Er wird wie folgt berechnet:

Bei zwei Klassen: $loss = -(ylog(\hat{y}) + (1-y)log(1-\hat{y}))$

Bei mehr (N) Klassen: $loss = -\sum_{c=1}^{N} y_c log(\hat{y_c})$

Accuracy

Keras wendet je nach gewählter Lossfunktion unterschiedliche Methoden zur Berechnung der Accuracy an. Die Vorgehensweise richtet sich nach der Anzahl der Klassen: handelt es sich lediglich um zwei Klassen oder mehr. Im Fall von nur zwei Klassen wird davon ausgegangen, dass alle Prognosen über 0.5 als Spurmarkierung vorhergesagt werden und alle darunter als keine Spurmarkierung. In Keras wird die Accuracy für zwei Klassen wie folgt berechnet:

$$K.mean(K.equal(y, K.round(\hat{y})), axis = -1)$$

Sobald es mehrere Klassen gibt, kann diese Art der Accuracy allerdings nicht mehr verwendet werden, da so mehrere Klassen pro Pixel festgestellt werden könnten. Desswegen wird in diesem Fall die Klasse mit der höchsten Wahrscheinlichkeit als Vorhersage betrachtet. In Keras wird die Berechnung dann folgendermaßen durchgeführt:

$$K.cast(K.equal(K.argmax(y, axis = -1), K.argmax(\hat{y}, axis = -1)), K.floatx())$$

Das Ergebnis ist ein Wert, der aussagt, wie viele Pixel von dem Bild korrekt klassifiziert wurden. Dieser Wert allein reicht allerdings nicht aus, da sonst ein Bild, das alle Pixel als “keine Spurmarkierung“ labelt, bereits sehr hohe Accuracy-Werte erhält.[38]

Intersection over Union (IoU)

IoU oder Jaccard Index ist eine Evaluationsmethode, mit der ebenfalls die Genauigkeit der Segmentierung bestimmt werden kann. Die Zahl der Pixel, die bei der Prediction- und Labelmaske gleich sind, wird durch die Gesamtzahl an Pixeln, die auf beiden Masken sind, geteilt:

$$IoU = \frac{y \cap \hat{y}}{y \cup \hat{y}} = \frac{TruePositives}{TruePositives + FalsePositives + FalseNegatives} \tag{4.1}$$

Pixel, die als Hintergrund/Void klassifiziert wurden, haben keinen Einfluss auf den IoU-Wert. Der Wert wird für jede Klasse einzeln bestimmt; aus diesen Werten wird dann der Durchschnitt (mean IoU) zur Beurteilung der Netze verwendet.[57]

Diese Metrik ist noch nicht in der Keras-Bibliothek vorhanden und muss zunächst implementiert werden, um genutzt werden zu können. Anschließend kann sie während des Trainings verwendet werden.

Dice Koeffizient

Der Dice Koeffizient nutzt ähnlich wie der IoU die True Positive Pixel der Segmentierung:

$$Dice = \frac{2 * (y \cap \hat{y})}{y + \hat{y}} = \frac{2 * TruePositives}{Pixelzahl\ beider\ Bilder} \tag{4.2}$$

Durch die Ähnlichkeit der beiden Berechnungsverfahren kann es vorkommen, dass die Werte gleich sind. Beide geben Werte zwischen Null und Eins aus.[57]

Wie bereits beim IoU, muss auch der Dice Koeffizient zunächst hinzugefügt werden. Der Dice Koeffizient wird häufig auch als Lossfunktion in der semantischen Segmentierung verwendet. Dafür muss er aber angepasst werden, da er sonst nicht die Anforderungen einer Lossfunktion erfüllt. Das liegt daran, dass er wie IoU immer größer wird, je besser die Ergebnisse werden. Eine einfache Lösung ist es, bei der Verwendung als Lossfunktion den berechneten Dice Koeffizienten von 1 abzuziehen. Diese Art, den Dice Loss zu berechnen, wird auch als “Soft Dice Loss” bezeichnet.[58]

Mithilfe dieser Metriken ist es möglich, die verschiedenen Trainingsergebnisse zu bewerten und zu vergleichen. Vor allem der IoU und Dice Koeffizient werden dabei ausschlaggebend sein, da sie für die semantische Segmentierung besser unterscheidbare

Werte liefern. In Tabelle 4.1 sind ausgewählte Ergebnisse des Trainings aufgelistet; für eine umfangreichere Übersicht der durchgeführten Tests ist eine weitere Tabelle 5.1 im Anhang enthalten.

Arch.	**Opt.**	**loss**	**Act.**	**filter**	**Drop**	**lr**	**Bat**	**CE**	**acc**	**IoU**	**Dice**
U-Net	Adam	CE	Softmax	8	0.1	0.01	64	0.9913	0.0116	0.0124	0.0245
U-Net	Adam	DL	Sigmoid	8	0.3	0.1	16	0.1088	0.9909	0.5405	0.7001
U-Net	Adam	DL	Sigmoid	8	0.3	0.01	64	0.0944	0.9915	0.5590	0.7167
U-Net	Adagrad	DL	Sigmoid	8	0.3	0.01	64	0.0698	0.9890	0.4417	0.6123
U-Net	Adadelta	DL	Sigmoid	8	0.3	0.1	64	0.0947	0.9870	0.2474	0.3961
SegNet	Adam	DL	Softmax	8	0.3	0.01	64	0.1803	0.9870	0.0801	0.1482
SegNet	Adam	DL	Sigmoid	8	0.3	0.1	64	0.1147	0.9917	0.5548	0.7133
SegNet	Adam	DL	Sigmoid	8	0.3	0.03	64	0.0984	0.9926	0.5976	0.7478
SegNet	Adagrad	DL	Sigmoid	8	0.3	0.03	64	0.0971	0.9925	0.5788	0.7328
SegNet	Adadelta	DL	Sigmoid	8	0.3	0.01	64	0.1229	0.9582	0.0339	0.0655
Deeplab	Adam	DL	Sigmoid	16	0.1	0.01	8	0.1150	0.9905	0.4985	0.6619
Deeplab	Adam	DL	Sigmoid	8	0.2	0.1	4	1.2158	0.0121	0.0128	0.0252
Deeplab	Adam	DL	Sigmoid	16	0.3	0.01	8	0.1231	0.9901	0.4840	0.6480
Deeplab	Adagrad	DL	Sigmoid	16	0.2	0.01	8	0.0671	0.9902	0.4780	0.6441
Deeplab	Adadelta	DL	Sigmoid	16	0.2	0.01	8	0.0613	0.9889	0.4222	0.5909

Tabelle 4.1: Übersicht der Trainingsergebnisse für die binäre Segmentierung

Anhand der Trainingsergebnisse (siehe Tabelle 4.1) wird deutlich, dass die SegNet-Architektur für die binäre Segmentierung der Spurmarkierungen mit einem IoU von 0.5976 und einem Dice Koeffizienten von 0.7478 die besten Vorhersagen erzielen konnte. Die jeweils besten Resultate für jede der drei Grundarchitekturen sind farblich hinterlegt. Darüber hinaus können in der Tabelle die Eigenschaften der Trainingsarchitektur und die Ergebnisse der Metriken auf den Validierungsdaten abgelesen werden. Zu den Architektureigenschaften in der Tabelle zählen die Grundarchitektur (Arch.), die verwendete Optimierungsfunktion (Opt.), die Lossfunktion (loss), die Aktivierungsfunktion des Output-Layer (Act.), die Ausgangs-Filteranzahl der ersten Convolution (filter), die Dropoutrate (Drop), die initiale Lernrate (lr) und die Batchgröße (Bat). Zudem sind die Ergebnisse in den zuvor erwähnten Metriken in der Tabelle aufgeführt; unter CE findet sich der Crossentropy Loss, unter acc die Accuracy und zusätzlich der IoU und der Dice Koeffizient.

Als Optimierungsfunktion hat sich Adam angeboten und konnte mit Abstand bessere Resultate als Adadelta und Adagrad liefern. Bei den Lossfunktionen hat sich der Dice Loss durchgesetzt, was wahrscheinlich daran liegt, dass die Hintergrundpixel nicht mit beachtet wurden. Unter Berücksichtigung der Hintergrundpixel haben die Netze schnell gelernt, dass durch Klassifizierung von Null der Loss drastisch reduziert werden kann, und die Spurmarkierungen lagen somit nicht im Fokus des Trainings. Mit dem Dice-Loss ist das nicht möglich, da die falsche Klassifikation der Spurmarkierungen großen Ein-

fluss haben würde und es keine “Belohnung” für richtig zugeordnete Hintergrundpixel gibt. Bei allen Ergebnissen fällt auf, dass die Accuracy und der Crossentropy Loss immer sehr positiv ausfallen, was ebenfalls daran liegt, dass die Voidpixel miteinbezogen wurden. Der IoU und Dice Koeffizient zeigen offensichtlichere Veränderungen zwischen den verschiedenen Testläufen und dienen als besserer Indikator für die Performance der Architekturen.

Während des Trainings fiel auf, dass die Sigmoid-Aktivierungsfunktion bei der binären Segmentierung der Softmax-Aktivierung überlegen ist. Das könnte möglicherweise an einer der Eigenschaften der Softmax-Aktivierung liegen, dass die Ergebnisse für die verschiedenen Klassen gesamt Eins ergeben und so die Klasse mit der höchsten Wahrscheinlichkeit für jedes Pixel eindeutig bestimmt wird. Da es in der binären Segmentierung nur eine Klasse gibt, ist es Sinnvoll, die Sigmoid-Funktion zu verwenden. Bei der im Anschluss durchzuführenden Multiclass-Segmentierung kann es sich allerdings durchaus lohnen, Softmax erneut zu testen. Bei der initialen Filteranzahl wurden die Ergebnisse besser, je höher diese gesetzt wurde, was ebenfalls auf die Batchgröße zutrifft. Die Begrenzung lag allerdings in der zur Verfügung stehenden Rechenleistung. Dadurch lag das Maximum zunächst bei der Filteranzahl 8 und einer Batchgröße von 64. Die am besten getestete Dropoutrate liegt bei 30% Neuronenausfall; also 0.3 und die Lernraten konnten zwischen 0.01 und 0.03 die besten Ergebnisse erzielen.

Beim Betrachtung der drei Grundarchitekturen fällt auf, dass Deeplab im Vergleich zu U-Net und SegNet schlechtere Resultate liefert. Ein Grund dafür könnte sein, dass manche Vorteile dieser Architektur auf Spurmarkierungen nicht zutreffen. Dazu zählt zum Beispiel das Erkennen von Objekten in mehreren Maßstäben. In dem verwendeten Datensatz sind Bilder von zwei verschiedenen Kameras, die fest an dem Fahrzeug angebracht sind, enthalten. Somit gibt es keine großen Veränderungen im Maßstab der Spurmarkierungen, da die Kameras immer die gleiche Entfernung haben. Auch das größere Sichtfeld ist bei Spurmarkierungen nur bedingt hilfreich.

Zum Trainieren der Multiclass-Segmentierung für alle verschiedenen Klassen wurden für jede der drei Grundarchitekturen nur die besten Zusammenstellungen verwendet. Zusätzlich wurde die Softmax-Aktivierungsfunktion erneut getestet, ob sie bessere Ergebnisse bei mehreren Klassen erzielen kann. Die Netze wurden allerdings nicht komplett neu trainiert, die Gewichte die zuvor für den binären Anwendungsfall am besten waren, wurden wieder verwendet und mit einer geringeren Lernrate weiter optimiert. Nach der Erstellung der Multiclass Label musste noch beachtet werden, dass die erste

der 36 Masken nicht mittrainiert werden soll, da sie den Hintergrund klassifiziert. Wird diese mit in das Training einbezogen, entfällt der Vorteil von dem Diceloss und das Netz lernt, dass gute Ergebnisse (IoU über 97%) erreicht werden, wenn alle Pixel als Hintergrund klassifiziert werden. In der nachfolgenden Tabelle sind die Ergebnisse der semantischen Segmentierung für alle Klassen aufgeführt:

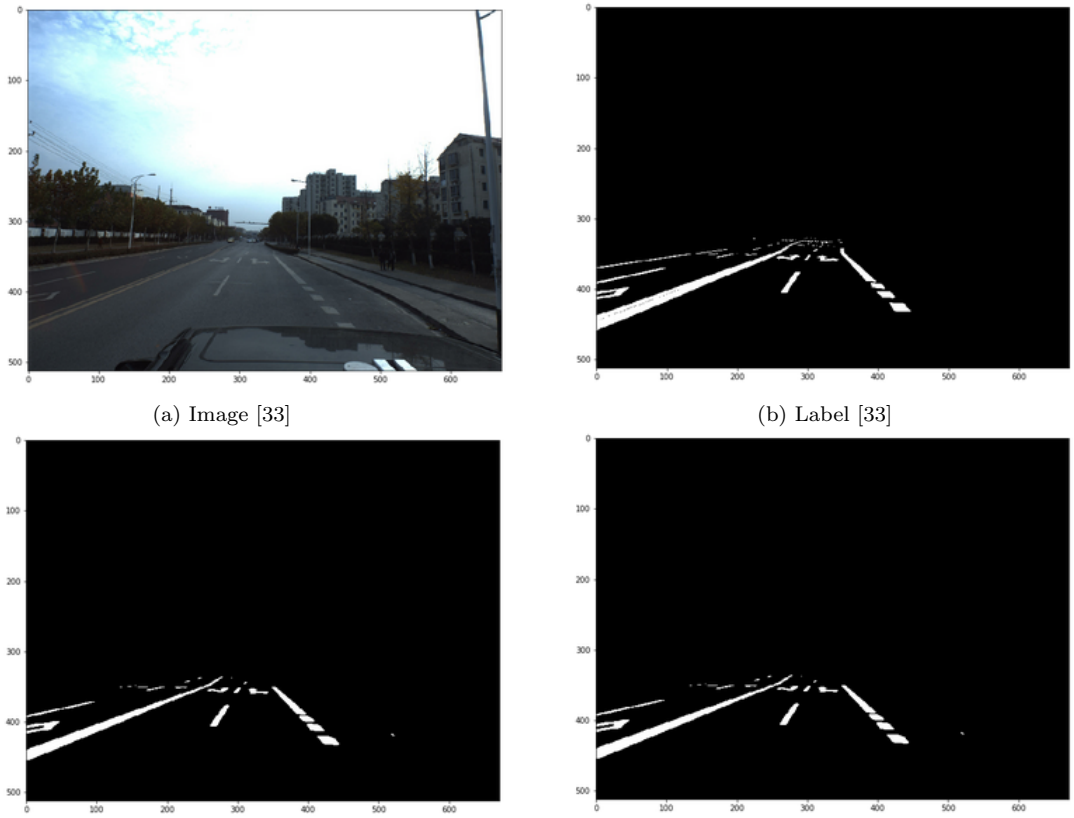
Arch.	Opt.	loss	Act.	filter	Drop	lr	Bat	CE	acc	IoU	Dice
U-Net	Adam	DL	Sigmoid	8	0.3	0.001	32	0.0228	0.9909	0.2842	0.4402
U-Net	Adam	DL	Softmax	8	0.3	0.001	16	0.1320	0.0243	0.0038	0.0075
SegNet	Adam	DL	Sigmoid	8	0.3	0.003	32	0.0131	0.9916	0.3169	0.4790
SegNet	Adam	DL	Softmax	8	0.3	0.03	16	0.0946	0.8095	0.0066	0.0132
Deeplab	Adam	DL	Sigmoid	16	0.2	0.003	6	0.0064	0.9562	0.3160	0.4696
Deeplab	Adam	DL	Softmax	16	0.2	0.003	4	0.1974	0.6105	0.0041	0.0081

Tabelle 4.2: Übersicht der Trainingsergebnisse für die Multiclass Segmentierung

Die besten Ergebnisse konnten für die Multiclass Segmentierung, ebenfalls mit der SegNet-Architektur (siehe Tabelle 4.2, farblich hinterlegt) erzielt werden. Als Aktivierungsfunktion für den Output-Layer konnte sich - auch bei mehreren Klassen - die Sigmoid-Funktion erneut durchsetzen. Die Batchgrößen mussten im Vergleich zu der binären Segmentierung allerdings verkleinert werden, da die vorhandene Rechenkapazität sonst nicht ausreichend gewesen wäre. Diese Vorgehensweise ist der Grund, dass nicht die kompletten Trainingsdaten - in jeder Epoche - zum Training verwendet werden konnten. Da die Daten zufällig gewählt werden, sollte trotzdem die Variabilität erhalten bleiben. Bei einem Testlauf mit allen Trainingsdaten in jeder Epoche, basierend auf der U-Net-Grundarchitektur, hat die Berechnung der ersten Epoche ungefähr acht Stunden gedauert. Vor dem Hintergrund möglicher Hardwareausfälle, erschien die Zeitspanne zu riskant, da ein Absturz vor Sicherung der Ergebnisse zu kostenintensiv wäre.

4.2 Bewertung

Anhand der Trainingsergebnisse war es möglich, zwischen den drei Architekturen die beste für den Anwendungsfall der Spurmarkierungserkennung zu identifizieren. Wie gut die Segmentierung die Bilder wirklich beurteilt, lässt sich auf der Visualisierung der Vorhersagen gut erkennen. Aus Abbildung 4.1 ist ersichtlich, dass die Vorhersage sich kaum von der vorgegebenen Maske unterscheidet. Wenn man nachträglich noch einen Threshold verwendet, zum Beispiel 0.5, können noch schärfere Kanten bei der Segmentierung erreicht werden, da Vorhersagen mit niedriger Wahrscheinlichkeit aussortiert werden. So werden vor allem in entfernteren Bereichen des Bildes verschwommene Bereiche vermieden.



(a) Image [33]

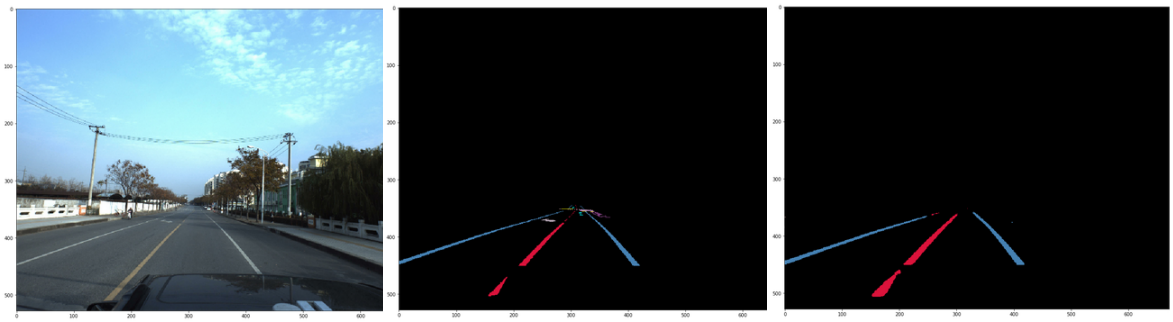
(b) Label [33]

(c) Segmentierungsergebnis

(d) Ergebnis mit Threshold 0.5

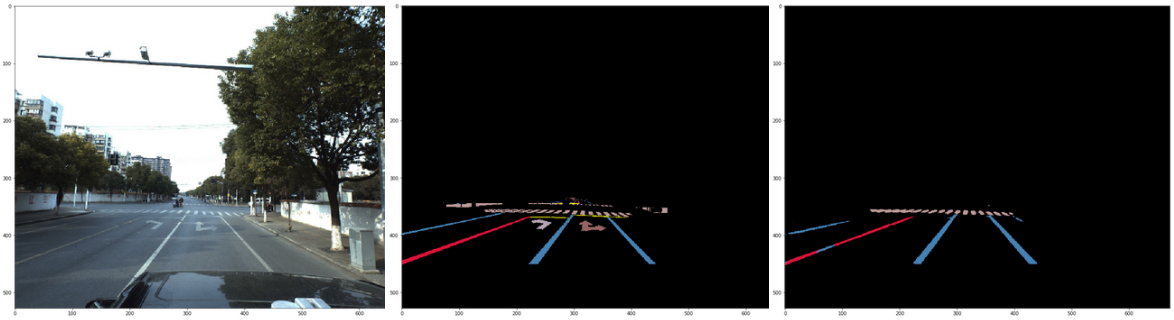
Abbildung 4.1: Ergebnis der binären Segmentierung mit SegNet

Bereits die Ergebnisse der binären semantischen Segmentierung bieten eine Verbesserung der aktuellen Toolkette zum Generieren von Teststrecken aus Bilddaten. Das ist damit zu begründen, dass die Störfaktoren der Bilder, die zuvor fälschlicherweise als Spurmarkierungen erkannt wurden, bereits ausgefiltert wurden und nur die Spurmarkierungen verblieben sind. Als besonders positiv ist anhand der Ergebnisse zu bewerten, dass die Semantische Segmentierung auch bei Spurmarkierungen gute Vorhersagen liefert. Im nächsten Arbeitsschritt, der Multiclass Segmentierung, sollen die Informationen weiter verfeinert werden. In den Bildern 4.2 und 4.3 sind die Ergebnisse der Multiclass Segmentierung visuell dargestellt. Auf dem ersten Bild (4.2) ist erkennbar, dass häufig auftretende Spurmarkierungen bereits sehr gut erkannt und klassifiziert werden. Spurmarkierungstypen, die seltener in den Trainingsdaten vorhanden sind, wie beispielsweise Pfeilmarkierungen, werden allerdings noch nicht gut erkannt (siehe Abbildung 4.3). Der Grund dafür könnte an dem, durch die gegebene Rechenkapazität, eingeschränkten Training liegen. Trotzdem ist es möglich, die Bildinformationen auf die Spurmarkierungen zu reduzieren, sofern diese erkannt wurden.



(a) Image [33] (b) Maske [33] (c) Segmentierungsergebnis

Abbildung 4.2: Prediction mit SegNet



(a) Image [33] (b) Maske [33] (c) Segmentierungsergebnis

Abbildung 4.3: Vorhersage von SegNet; seltene Klassen werden noch nicht erkannt

Mithilfe der Semantischen Segmentierung ist es möglich, nur relevante Daten auf den Bildern zu behalten und die restlichen Informationen auszusortieren. Dabei sind auch die Informationen zur Farbe und Art der einzelnen Spurmarkierungen immer noch für jedes Pixel erhalten. Im nächsten Abschnitt wird erörtert, in welcher Form diese Informationen am besten zu Teststrecken konvertiert werden können.

4.3 Generieren der Teststrecken

In diesem Abschnitt soll diskutiert werden, welche Bearbeitungsschritte erforderlich wären, um aus den Segmentierungsergebnissen der Spurmarkierungsserkennung virtuelle Teststrecken zu generieren. Um das zu erreichen, gibt es zwei verschiedene Herangehensweisen. Als eine Möglichkeit kann das trainierte Modell mit in die vorhandene Toolkette integriert werden; durch Abwesenheit von unnötigen Informationen ist eine verbesserte Performance zu erwarten. Dadurch würde sich die Toolkette allerdings um einen weiteren Arbeitsschritt verlängern.

In Hinblick auf eine zukünftige voll automatisierte Teststreckengenerierung aus Bilddaten wäre es zunächst notwendig, eine geeignete Bibliothek für die Beschreibung von Teststrecken zu wählen, die mit der vorhandenen Testumgebung kompatibel ist. Hierfür bietet sich das OpenDRIVE®-Format an, das die Beschreibung von Straßensystemen in Extensible Markup Language (XML)-Syntax ermöglicht. Um genauer definieren zu können, welche Bearbeitungsschritte in einem Postprocessing der Daten notwendig sind, muss zunächst das gewählte Teststreckenformat genauer betrachtet werden.

4.3.1 OpenDRIVE® XML-Syntax

OpenDRIVE® ist ein Format, das Basisfunktionen zur Beschreibung von streckenbasierten Straßennetzwerken mit der XML-Syntax zur Verfügung stellt. Alle notwendigen Schlüsseleigenschaften von echten Straßennetzwerken können nachgestellt werden. Das Format wurde von der VIRES Simulationstechnologie GmbH in Zusammenarbeit mit Daimler AG Driving Simulator entwickelt und kam 2006 auf den Markt. Ziel war es, ein globales, lieferantenunabhängiges Format zu schaffen, um die vielen verschiedenen Formate zur Straßenbeschreibung zu vereinheitlichen.[59] Um Aktualität zu bieten, muss erwähnt werden, dass der OpenDRIVE® Standard am 12. September 2018 in die Verantwortung von ASAM e.V. übergegangen ist.

Die Daten in den OpenDRIVE®-Dateien beschreiben die Geometrie der Straße und Eigenschaften, die die Logik beeinflussen (zum Beispiel Ampeln und Straßenschilder). Das Format ist in sogenannten „nodes“ organisiert, die vom Nutzer auch erweitert werden können. Dadurch ist die Spezialisierung für individuelle Applikationen möglich.[60]

OpenDRIVE® richtet sich nach der XML-Definition 1.0.[61] Mit der Version 1.4 wurde ein individueller Namensraum eingeführt, der verschiedene XML-Namensräume kombiniert. Die mit OpenDRIVE® erstellten XML-Dokumente werden mit der Endung .xodr gespeichert. Die genaue Verwendung der XML-Befehle wird in der OpenDRIVE®-Formatspezifizierung erläutert. [60, Seite 7]

Im OpenDRIVE®-Format werden Teststrecken in der Regel anhand von Referenzlinien beschrieben. Dafür ist es notwendig, Punkte vom Verlauf der spureingrenzenden Markierungen anzugeben. Anschließend orientieren sich auch die weiteren hinzugefügten Spurmarkierungen und Spuren an diesem Verlauf. Des Weiteren ist es möglich, die genauen Breiten der Spurmarkierungen und Spuren festzulegen, es können aber auch zunächst default-Werte verwendet werden. Für jede Spurmarkierung muss zudem ex-

plizit die Farbe und Art der Markierung angegeben werden; diese sind bereits aus der Segmentierung abzulesen.[60]

Für Sonderformen von Spurmarkierungen wie beispielsweise Haltelinien und Fußgängerüberwege können nur die Breite und betroffene Spuren, die gekreuzt werden, angegeben werden. Bei den Pfeilmarkierungen sind wieder weniger Informationen notwendig, nur die Richtung und Position werden vorgegeben, ansonsten sind sie standardisiert.[60]

4.3.2 Data Postprocessing

Um die für das OpenDRIVE®-Format benötigeten Informationen aus den Segmentierungen auslesen zu können, wäre zunächst eine Transformation zu Top-View-Bildern von Vorteil. Damit die dafür notwendige Transformationsmatrix erstellt werden kann, werden allerdings die genauen intrinsischen- und extrinsischen Daten der Kamerapositionen benötigt. Bei dem Lane Segmentation Datenset von ApolloScape[33] sind jedoch nur die intrinsischen Kamerawerte gegeben. Es gibt zwar einen weiteren ApolloScape[33] Datensatz, der auch extrinsische Daten enthält, allerdings sind diese für andere Testfahrten definiert worden.

Sollte das Projekt zur automatischen Teststreckengenerierung aus Bilddaten weitergeführt werden, wären Daten notwendig, die genaue Informationen zu den Kameras enthalten, damit die weitere Extraktion von Informationen möglich ist. Auch für die Ermittlung der Markierungsbreiten und Längen werden weitere Informationen zu den Kamerapositionen und Entfernungen benötigt. Zum momentanen Zeitpunkt und mithilfe der zur Verfügung stehenden Daten, könnten nur die Pixelbreiten ermittelt werden. Dadurch kann zwar ein relativ genaues Größenverhältnis zwischen den einzelnen Markierungselementen ermittelt werden; allerdings muss bei mindestens einem ein Standardwert verwendet werden.

Um mit dem OpenDRIVE®-Format die Teststrecken generieren zu können, werden von unterschiedlichen Spurmarkierungsarten unterschiedliche Informationen benötigt. Bei den durchgängigen- und unterbrochenen Spurmarkierungen werden die Markierungsart, die Breite und Punkte auf der Markierung, die in regelmäßigen Abständen den Verlauf beschreiben, benötigt. Die Markierungsart ist bereits durch die Segmentierung gegeben. Da kein Objekt mit bekannter Größe auf den segmentierten Bildern vorhanden ist, bietet sich nur die Möglichkeit, die Breite der Spurmarkierungen zu ermitteln, die intrinsischen Daten der Kamera zu benutzen.

Ein weiterer Punkt, der in der Arbeit bisher noch nicht thematisiert wurde, ist die Kombination mehrerer Bilder zu einer einheitlichen Teststrecke. Dabei wäre vor allem überlegenswert, wie die Übergänge zwischen den Bildern am besten realisiert werden. Sollten die genauen extrinsischen Kameradaten zur Verfügung stehen, ist es naheliegend, diese zu verwenden und von jedem Bild jeweils nur den Ausschnitt zu verwenden, der sich nicht mit denen der anderen Bilder überschneidet und nah genug ist, um eindeutige Vorhersagen zu treffen.

5 Zusammenfassung und Ausblick

Das Ziel dieser wissenschaftlichen Arbeit war es, eine zuverlässige, detailgenaue Spurmarkierungserkennung auf der Grundlage von Bilddaten zu erarbeiten. Diese wird als Vorraussetzung für die Erstellung virtueller Teststrecken benötigt, die als Eränzung zu real stattfindenden Testfahrten zunehmend an Bedeutung gewinnen. Die Anforderung auch an virtuelle Teststrecken steigt mit zunehmender Komplexität automatisierter Fahrfunktionen. Sie müssen das wirkliche Straßenbild immer realistischer nachstellen. In dieser Arbeit wurde untersucht, welche Ansätze erfolgverspechend sind und welche Methoden eingesetzt werden können, um einen ersten Ansatz einer praktikablen Spurmarkierungserkennung vorlegen zu können.

Bei der klassischen Spurerkennung wird die Art der Spurmarkierung häufig nicht ausreichend berücksichtigt. Für virtuelle Teststrecken ist das Erkennen unterschiedlicher Spurmarkierungen allerdings essenziell. Die Publikationen (betreffend Spurerkennung) der letzten Jahre belegen, dass der Trend immer mehr zu maschinellen Lernverfahren tendiert. Um die Aktualität der Abschlussarbeit zu gewährleisten, wurde für die Spurmarkierungserkennung eine solche Architektur verwendet. Zunächst musste ein geeignetes Datenset für das Training gefunden werden; ein eigenes anhand des Anforderungsprofils zu erstellen, war aus zeitlichen Gründen nicht möglich. Es wurden zwei durchaus geeignete Datensets gefunden: Lane Segmentation von ApolloScape[33] und BDD100k[28].

Der erste Lösungsansatz, der getestet wurde, war eine Variante der Landmark detection und basierte auf den Daten von BDD100k[28]. Nach mehreren Fehlversuchen musste allerdings festgestellt werden, dass keine verwertbaren Ergebnisse mit dem verfügbaren Datenset erzielt werden konnten. Die Suche nach einem erfolgversprechenderen Ansatz wurde intensiviert. Es war naheliegend, erneut die klassische Spurmarkierung zu konsultieren. Bei einigen Arbeiten wurde eine Art Segmentierung verwendet, um den Verlauf der Spurmarkierungen zu ermitteln. Dabei wird meist ein grober Verlauf von bis zu maximal vier der nächsten Markierungen ohne weitere Klassifizierung erfasst. Als nächster Schritt wurde eine detailliertere semantische Segmentierung durchgeführt,

was es möglich macht, alle Informationen, die für Teststrecken benötigt werden, auf den Bildern zu extrahieren und überflüssige Informationen zu eliminieren.

Da das Erkennen von Spurmarkierungen nicht typisch für die Semantische Segmentierung ist, wurden drei verschiedene Grundarchitekturen getestet: Unet, Segnet und Deeplabv3+. Sie wurden im Verlauf der Tests mit unterschiedlichen Layern, Hyperparametern und Methoden angepasst. Zunächst wurden die Architekturen nur mit binären Labeln trainiert und getestet, um schneller an die Ergebnisse hinsichtlich ihrer Eignung für die Aufgabenstellung zu gelangen. In dieser Phase der Arbeit sind mehrfach Hardwareprobleme und -ausfälle aufgetreten, die in einem Komplettausfall der Festplatte endeten. Nachdem diese ausgetauscht wurde, gab es erneut Probleme, was den Komplettaustausch des Testrechners nach sich zog. Die besten Ergebnisse wurden mit SegNet erzielt, wo ein IoU von 0.3169 und ein Dice Koeffizient von 0.4790 erreicht werden konnten. Mithilfe von diesen Ergebnissen ist es bereits möglich, die aktuelle Toolkette zu verbessern. Allerdings wäre zunächst ein ausführlicheres erneutes Training ratsam, da einzelne Spurmarkierungsarten noch nicht optimal erkannt werden. Da es sich dabei größtenteils um seltener vorkommende Markierungstypen handelt, würde sich auch eine “Data Augmentation” anbieten, um diese zu vervielfältigen.

Für das Ziel der vollständigen Automatisierung vom Foto zur Teststrecke müssen noch weitere Arbeitsschritte vorgenommen werden. Da sowohl für die Transformation zur Top-View, als auch für die genaue Ermittlung der Breiten und Längen von Markierungen, detaillierte intrinsische- und extrinsische Kamerakoordinaten notwendig sind, können diese mit den Daten des Lane Segmentation Datensets von ApolloScape[33] nicht durchgeführt werden. Dieser Punkt sollte bei der Generierung von den Daten, die später zum Erstellen der Teststrecken verwendet werden sollen, dringlich beachtet werden, da sonst mindestens eine der Längen auf dem Bild mithilfe von Standardwerten festgelegt werden muss. Das ist zwar auch möglich, dadurch geht allerdings wieder eine Information der realen Straße verloren. Eine weitere denkbare Variante könnte ein Training direkt auf gelabelten Top-View-Bildern sein. Dafür würde ein Datensatz benötigt werden, der die entsprechenden Masken zur Verfügung stellt. Diese Methode würde auch die Vereinigung von mehreren Bildern zu einer Teststrecke vereinfachen und das Postprocessing der segmentierten Bilder würde deutlich kürzer ausfallen.

Zusammenfassend lässt sich sagen, dass das Ziel, eine geeignete Spurmarkierungserkennung zu ermitteln, gelungen ist. In dieser Arbeit wurde nachgewiesen, dass sich dafür die Semantische Segmentierung anbietet, da alle erforderlichen Informationen

von den Bildern extrahiert werden. Mithilfe der SegNet-Architektur konnten die besten Ergebnisse erzielt werden, die auch die unterschiedlichen Spurmarkierungsarten und Farben unterscheiden können. Eine Frage, die in zukünftigen Untersuchungen noch geklärt werden könnte, ist, ob weitere Segmentierungsarchitekturen noch ausichtsreichere Ergebnisse erzielen können. Wünschenswert wäre auch, ein auf die Aufgabenstellung zugeschnittenes Datenset, basierend auf Top-View-Bildern, mit dem die Gewichte des CNN verfeinert werden könnten. Dadurch wäre die Spurmarkierungserkennung optimal auf den Anwendungsfall der Teststreckengenerierung abgestimmt.

Abbildungsverzeichnis

Bibliographie

Literatur

(7) *Bayer, Johannes; Collisi, Thomas und Wenzel, Sigrid:* Simulation in der Automobilproduktion, VDI-Buch, Springer-Verlag Berlin Heidelberg, 1. Auflage, 2003.

(8) *Schäuffele, Jörg und Zurawka, Thomas:* Automotive Software Engineering. Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen, ATZ/MTZ-Fachbuch, Springer Vieweg, Wiesbaden, 6. Auflage, 2016.

(11) *Szeliski, Richard:* Computer Vision. Algorithms and Applications, Springer Vieweg, Springer-Verlag London, 1. Auflage, 2011.

(16) *Aghdam, Hamed Habibi und Heravi, Elnaz Jahani:* Guide to Convolutional Neural Networks. A Practical Application to Traffic-Sign Detection and Classification, Springer International Publishing, Gewerbestrasse 11, 6330 Cham, Switzerland, 1. Auflage, 2017.

(17) *Hastie, Trevor; Tibshirani, Robert und Friedman, Jerome:* The Elements of Statistical Learning. Data Mining, Inference, and Prediction, Springer-Verlag New York, 2. Auflage, 2009.

(18) *Bishop, Christopher M.:* Pattern Recognition and Machine Learning, Springer-Verlag New York, 1. Auflage, 2006.

(19) *S. Kevin Zhou; Hayit Greenspan und Dinggang Shen:* Deep Learning for Medical Image Analysis, Academic Press, 1. Auflage, 2017.

(22) *Le Lu; Yefeng Zheng; Gustavo Carneiro und Lin Yang:* Deep Learning and Convolutional Neural Networks for Medical Image Computing. Precision Medicine, High Performance and Large-Scale Datasets, Springer International Publishing, Switzerland, 1. Auflage, 2017.

Dokumentationen

(28) *Fisher Yu:* Berkeley DeepDrive, Sep. 2019, `https://bdd-data.berkeley.edu/`.

(29) *Bosch:* Unsupervised Llamas. The unsupervised labeled lane markers dataset, Sep. 2019, `https://unsupervised-llamas.com/llamas/`.

(30) *Karlsruhe Institute of Technology:* Welcome to the KITTI Vision Benchmark Suite!, Sep. 2019, `http://www.cvlibs.net/datasets/kitti/`.

(31) *TuSimple:* tusimple benchmark ground truth, Sep. 2019, `https://github.com/TuSimple/tusimple-benchmark/issues/3`.

(32) *The Chinese University of Hong Kong:* CULane Dataset, Sep. 2019, `https://xingangpan.github.io/projects/CULane.html`.

(33) *ApolloScape:* ApolloScape. Advanced Open Tools and Datasets for Autonomous Driving, Sep. 2019, `http://apolloscape.auto/index.html`.

(34) *Project Jupyter:* Jupyter, Sep. 2019, `https://jupyter.org/`.

(35) Python, Sep. 2019, `https://www.python.org/`.

(36) GitLab, Sep. 2019, `https://about.gitlab.com/`.

(37) *initiiert durch Google Brain Team Community-Projekt:* An end-to-end open source machine learning platform, Sep. 2019, `https://www.tensorflow.org/`.

(38) *initiiert durch François Chollet Community-Projekt:* Keras: The Python Deep Learning library, Sep. 2019, `https://keras.io/`.

(39) *initiiert durch Google Brain Team Community-Projekt:* Keras, Sep. 2019, `https://www.tensorflow.org/guide/keras`.

(40) *initiiert durch Google Brain Team Community-Projekt:* Using TFRecords and tf.Example, Sep. 2019, `https://www.tensorflow.org/tutorials/load_data/tf_records`.

(41) Keras Documentation. Usage of callbacks, Sep. 2019, `https://keras.io/callbacks/`.

(48) *Yangqing Jia:* Caffe, Sep. 2019, `https://caffe.berkeleyvision.org/`.

(60) *Dupuis, Marius e.a:* OpenDRIVE®. Format Specification, Rev. 1.4, VIRES Simulationstechnologie GmbH, Nov. 2015.

Wissenschaftliche Paper

(2) *Xingang Pan; Jianping Shi; Ping Luo; Xiaogang Wang und Xiaoou Tang:* Spatial As Deep: Spatial CNN for Traffic Scene Understanding, ***abs/1712.06080***. *„CoRR“*, 2017, **abs/1712.06080**.

(3) *Mohsen Ghafoorian; Cedric Nugteren; Nóra Baka; Olaf Booij und Michael Hofmann:* EL-GAN: Embedding Loss Driven Generative Adversarial Networks for Lane Detection. *„CoRR abs/1806.05525“*, 2018.

(10) *T. S. Huang:* Computer Vision: Evolution and Promise, University of Illinois at Urbana-Champaign, 1996.

(12) *Warren McCulloch und Walter Pitts:* A Logical Caluculus of the Ideas Immanent in Nervous Activity. *„Bulletin of Mathematical Biology“*, 1990.

(15) *P. Werbos: „Beyond regression: New tools for prediction and analysis in the behavorial sciences“*, Ph.D. dissertation, Haivard Univ., Cambridge, 1974.

(23) *Nadra Ben Romdhane; Mohamed Hammami und Hanene Ben-Abdallah:* A Lane Detection and Tracking Method for Driver Assistance System. *„Knowledge-Based and Intelligent Information and Engineering Systems“*, Hrsg. Andreas König; Andreas Dengel; Knut Hinkelmann; Koichi Kise; Robert J. Howlett und Lakhmi C. Jain, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, 407–417.

(24) *Yue Wang; Eam Khwang Teoh und Dinggang Shen:* Lane detection and tracking using B-Snake. *„Image and Vision Computing“*, Elsevier, 2004.

(25) *Jianzhuang Wang; Youping Chen; Jingming Xie und Haiping Lin:* Model-based Lane Detection and Lane Following for Intelligent Vehicles. *„2010 Second International Conference on Intelligent Human-Machine Systems and Cybernetics“*, IEEE, 2010.

(26) *Stefan Vacek; Stephan Bergmann; Ulrich Mohr und Rüdiger Dillmann:* 2006 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems. *„Rule-based tracking of multiple lanes using particle filters“*, 2006, 203–208.

(27) *Luo-Wei Tsai; Jun-Wei Hsieh; Chi-Hung Chuang und Kuo-Chin Fan:* Lane Detection Using Directional Random Walks. *„2008 IEEE Intelligent Vehicles Symposium“*, IEEE, 2008.

(42) *R. Ranjan; V. M. Patel und R. Chellappa:* HyperFace: A Deep Multi-Task Learning Framework for Face Detection, Landmark Localization, Pose Estimation, and Gender Recognition. *„IEEE Transactions on Pattern Analysis and Machine Intelligence“*, 2019.

(43) *Dun Liang; Yuanchen Guo; Shaokui Zhang; Song-Hai Zhang; Peter Hall; Min Zhang und Shimin Hu: „LineNet: a Zoomable CNN for Crowdsourced High Definition Maps Modeling in Urban Environments“*, 2018, **abs/1807.05696**.

(44) *A. Krizhevsky; I. Sutskever und G. E. Hinton:* mageNet classification with deep convolutional neural networks. *„Proc. Int. Conf. Adv. Neural Inform. Process. Syst.“*, 2012.

(45) *Kaiming He; Xiangyu Zhang; Shaoqing Ren und Jian Sun: „Deep Residual Learning for Image Recognition“*, 2015, **abs/1512.03385**.

(46) *Davy Neven; Bert De Brabandere; Stamatios Georgoulis; Marc Proesmans und Luc Van Gool:* Towards End-to-End Lane Detection: an Instance Segmentation Approach. *„2018 IEEE Intelligent Vehicles Symposium“*, 2018.

(47) *Olaf Ronneberger; Philipp Fischer und Thomas Brox:* U-Net: Convolutional Networks for Biomedical Image Segmentation. *„CoRR abs/1505.04597“*, 2015.

(49) *Vijay Badrinarayanan; Alex Kendall und Roberto Cipolla:* SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *„CoRR abs/1511.00561“*, 2016.

(50) *K. Simonyan und A. Zisserman:* Very deep convolutional networks for large-scale image recognition, 2014.

(51) *Liang-Chieh Chen; George Papandreou; Iasonas Kokkinos; Kevin Murphy und Alan L. Yuille: „Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs“*, ICLR 2015, 2015, **abs/1412.7062**.

(52) *Liang-Chieh Chen; George Papandreou; Iasonas Kokkinos; Kevin Murphy und Alan L. Yuille:* DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *„CoRR abs/1606.00915“*, 2017.

(54) *Philipp Krähenbühl und Vladlen Koltun: „Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials“*, 2012, **abs/1210.5644**.

(55) *Liang-Chieh Chen; Yukun Zhu; George Papandreou; Florian Schroff und Hartwig Adam: „Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation“*, 2018, **abs/1802.02611**.

(56) *Jifeng Dai; Haozhi Qi; Yuwen Xiong; Yi Li; Guodong Zhang; Han Hu und Yichen Wei: „Deformable Convolutional Networks“*, 2017, **abs/1703.06211**.

Artikel

(1) *Carmeq GmbH:* Das sind wir, Juni 2019, `https://www.carmeq.com/de/index.html`.

(5) *Deutscher Verkehrssicherheitsrat:* Leitfaden Fahrbahnmarkierung. Schriftenreihe Verkehrssicherheit 17, Nov. 2019, https://www.dvr.de/publikationen/schriftenreihe/17-leitfaden-fahrbahnmarkierung/.

(9) Machine Learning – Online Kurs der Stanford Universität, Aug. 2019, https://online-bildung.org/kurs/machine-learning-online-kurs-der-stanford-universitaet/.

(13) *Stanford University:* Neural Networks History: The 1940's to the 1970's, Nov. 2019, https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html.

(14) *Spektrum Lexikon der Mathematik:* Minsky-Papert-Kritik, Springer Verlag GmbH Deutschland, Nov. 2019, https://www.spektrum.de/lexikon/mathematik/minsky-papert-kritik/6426.

(20) *Sebastian Ruder:* An overview of gradient descent optimization algorithms, Nov. 2019, https://ruder.io/optimizing-gradient-descent/.

(21) *Jason Brownlee:* A Gentle Introduction to Padding and Stride for Convolutional Neural Networks, Nov. 2019, https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks/.

(53) *Beeren Sahu:* The Evolution of Deeplab for Semantic Segmentation. From classical image segmentation methods throuhgh Deep learning based semantic segmentation to the Deeplab and its variant. Nov. 2019, https://towardsdatascience.com/the-evolution-of-deeplab-for-semantic-segmentation-95082b025571.

(57) *Ekin Tiu:* Metrics to Evaluate your Semantic Segmentation Model. How do you know your segmentation model performs well? Find out here. Nov. 2019, https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2.

(58) *Jeremy Jordan:* An overview of semantic image segmentation. Nov. 2019, https://www.jeremyjordan.me/semantic-segmentation/.

(59) *OpenDRIVE®:* Welcome to the World of OpenDRIVE®!, Juni 2019, http://www.opendrive.org/.

Standards und Normen

(4) Deutsche und Englische Fassung prEN 1436:2016, 2016-07.

(6) Verein Deutscher Ingenieure, 2013.

(61) ersetzt RFC4646 und RFC4647, Feb. 2008.

Anhang

Übersicht der Trainingsergebnisse zur binären Semantischen Segmentierung

Arch.	Opt.	loss	Act.	filter	Drop	lr	Bat	CE	acc	IoU	Dice
U-Net	Adam	CE	Sigmoid	8	0.5	0.01	64	0.0265	0.9912	0.4069	0.5780
U-Net	Adam	CE	Sigmoid	2	0.5	0.01	32	0.0438	0.9684	0.0068	0.0144
U-Net	Adam	CE	Sigmoid	8	0.4	0.01	64	0.0244	0.9917	0.4489	0.6193
U-Net	Adam	CE	Sigmoid	8	0.3	0.01	64	0.0233	0.9921	0.4624	0.6320
U-Net	Adam	CE	Sigmoid	8	0.2	0.01	64	0.0219	0.9923	0.4605	0.6302
U-Net	Adam	CE	Sigmoid	2	0.2	0.01	32	0.0925	0.9814	0.0337	0.0668
U-Net	Adam	CE	Sigmoid	2	0.1	0.01	32	0.0923	0.9824	0.0210	0.0864
U-Net	Adam	CE	Sigmoid	2	0.1	0.01	16	0.1353	0.9815	0.0478	0.0612
U-Net	Adam	DL	Sigmoid	8	0.4	0.01	64	0.1048	0.9911	0.5357	0.6973
U-Net	Adam	CE	Sigmoid	8	0.1	0.01	64	0.0204	0.9923	0.4407	0.6115
U-Net	Adam	DL	Sigmoid	8	0.3	0.1	16	0.1088	0.9909	0.5405	0.7001
U-Net	Adam	CE	Softmax	8	0.1	0.01	64	0.9913	0.0116	0.0124	0.0245
U-Net	Adam	DL	Softmax	8	0.1	0.01	64	0.9904	0.0117	0.0126	0.0248
U-Net	Adam	DL	Sigmoid	8	0.3	0.01	64	0.0944	0.9915	0.5590	0.7167
U-Net	Adam	DL	Sigmoid	8	0.3	0.001	64	0.0788	0.9913	0.5384	0.6996
U-Net	Adagrad	DL	Sigmoid	8	0.3	0.01	64	0.0698	0.9890	0.4417	0.6123
U-Net	Adadelta	DL	Sigmoid	8	0.3	0.01	64	0.7011	0.5767	0.0214	0.0419
U-Net	Adadelta	DL	Sigmoid	8	0.3	0.1	64	0.0947	0.9870	0.2474	0.3961
SegNet	Adam	DL	Softmax	8	0.3	0.01	64	0.1803	0.9870	0.0801	0.1482
SegNet	Adam	DL	Sigmoid	8	0.3	0.01	64	0.1041	0.9922	0.5874	0.7397
SegNet	Adam	DL	Sigmoid	8	0.3	0.1	64	0.1147	0.9917	0.5548	0.7133
SegNet	Adam	DL	Sigmoid	8	0.3	0.03	64	0.0984	0.9926	0.5976	0.7478
SegNet	Adagrad	DL	Sigmoid	8	0.3	0.03	64	0.0971	0.9925	0.5788	0.7328
SegNet	Adadelta	DL	Sigmoid	8	0.3	0.01	64	0.1229	0.9582	0.0339	0.0655
Deeplab	Adam	DL	Sigmoid	16	0.1	0.01	8	0.1150	0.9905	0.4985	0.6619
Deeplab	Adam	DL	Sigmoid	8	0.2	0.1	4	1.2158	0.0121	0.0128	0.0252
Deeplab	Adam	DL	Sigmoid	16	0.3	0.01	8	0.1231	0.9901	0.4840	0.6480
Deeplab	Adam	DL	Softmax	16	0.2	0.01	8	15.1311	0.0123	0.0132	0.0260
Deeplab	Adam	DL	Sigmoid	16	0.3	0.03	8	0.1419	0.9885	0.3821	0.5485
Deeplab	Adagrad	DL	Sigmoid	16	0.2	0.01	8	0.0671	0.9902	0.4780	0.6441
Deeplab	Adadelta	DL	Sigmoid	16	0.2	0.01	8	0.0613	0.9889	0.4222	0.5909

Tabelle 5.1: Übersicht der Trainingsergebnisse für die binäre Segmentierung