Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Burak Ulu

Semantische Kantendetektion mit Neuronalen Netzen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Burak Ulu

Semantische Kantendetektion mit Neuronalen Netzen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Andreas Meisel
Zweitgutachter : Prof. Dr. Bettina Buth

Abgegeben am 22.01.2021

Burak Ulu

Thema der Arbeit
Semantische Kantendetektion mit Neuronalen Netzen

Stichworte
Deep Learning, Bildverarbeitung, Semantische Kantendetektion, Neuronale Netze, Faltungsnetzwerke, Autoencoder

Kurzzusammenfassung
In dieser Arbeit werden Neuronale Netze zur semantischen Kantendetektion verwendet. Im Fokus steht dabei die grundlegende Erkennung von Autos in Bildern. Unter Verwendung unterschiedlicher Netzkonfigurationen soll durch Vergleiche untersucht werden, welche Einflüsse die einzelnen Hyperparameter des Neuronalen Netzes auf die Ergebnisse haben. Für die Gewinnung aussagekräftiger Erkenntnisse wird pro Szenario ein Hyperparameter des Netzes geändert. Diese Erkenntnisse sollen dann dazu genutzt werden, um die Ergebnisqualität zu steigern. Dabei hat sich gezeigt, dass die Größe der erfassten Merkmale einen wesentlichen Einfluss auf die Ergebnisse haben. Zudem sollte der Bilddatensatz möglichst alle unterschiedlichen Konditionen umfassen wie die Bereitstellung von genügend Beispielbildern ohne Autos.

Burak Ulu

Title of the paper
Semantic edge-detection with neural networks

Keywords
Deep learning, image processing, semantic edge-detection, neural networks, convolutional neural networks, autoencoder

Abstract
In this thesis, neural networks are used for semantic edge detection. The focus is on the basic detection of cars in images. Using different network configurations, comparisons are made to investigate the influence of the individual hyperparameters of the neural network on the results. To gain meaningful insights, one hyperparameter of the network will be changed per scenario. These findings will then be used to improve the quality of the results. It has been shown that the size of the features captured have a significant impact on the results. In addition, the image dataset should include all different conditions such as providing enough sample images without cars.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Künstliche Intelligenz gestaltet den Alltag der Menschen immer stärker mit. Angefangen bei Anfragen bei einer Suchmaschine über Kaufempfehlungen im Online-Shopping, der Gesichtserkennung auf dem Smartphone, die Kommunikation mit Sprachassistenten wie Siri oder Alexa oder die Umwandlung natürlicher Sprache in Text – all das sind Dinge, die Künstliche Intelligenz bereits heute kann (1). Doch die Künstliche Intelligenz beeinflusst nicht nur die Menschen, sondern treibt auch andere Technologien und Wissenschaften an. Dank Neuronaler Netze konnten insbesondere im Bereich der Computer Vision für bestehende Techniken neue Ansätze entwickelt werden (2). Eine dieser profitierenden Techniken ist die Kantendetektion, die Schwerpunkt dieser Bachelorarbeit ist. Sie bildet die Basis für zahlreiche andere Techniken wie die Objekterkennung.
Bei der Kantendetektion geht es im Allgemeinen darum, Kanten unterschiedlicher Objekte innerhalb eines Bildes zu finden. Dabei handelt es sich bei einer Kante vereinfacht gesagt um die sichtbare Umrisslinie eines Objektes. Mit Objekten im Bild sind z.B. Personen, Gegenstände oder Fahrzeuge gemeint. Objekte können einzeln auftreten oder sich auch gegenseitig überlagern. Zum Beispiel kann in einem Bild eine Person hinter einem Auto stehen, wodurch die Person nicht vollständig auf dem Bild enthalten wäre. Dasselbe wäre auch für die Kante der Fall. Gleiche zusammengehörige Objekte haben eine gemeinsame Kante. In der Vergangenheit wurden unterschiedliche Algorithmen zur Kantendetektion entworfen. Der Canny-Algorithmus ist hier als Beispiel zu nennen. Dieser findet erfolgreich die Kanten aller im Bild befindlichen Objekte, seien es Gebäude, Personen etc.

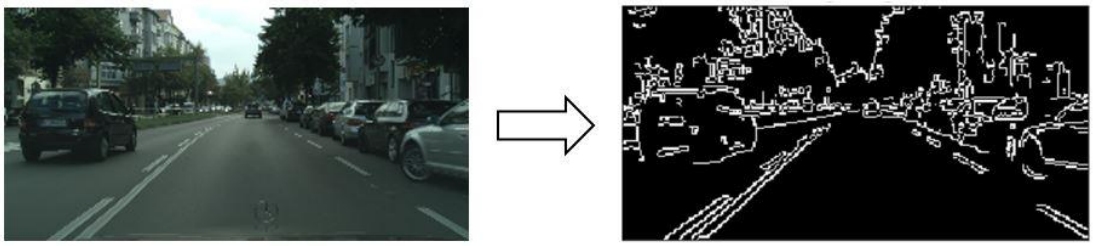


Abbildung 1-1 Klassische Kantendetektion: Anwendung des Canny-Algorithmus auf ein Eingabebild

Das Ganze hat aber auch einen entscheidenden Nachteil. Das Problem dieser klassischen Algorithmen zur Kantenerkennung ist, dass diese nicht die Fähigkeiten besitzen, die nur zu einem bestimmten Objekt zugehörigen Kanten zu finden. Sie arbeiten vollständig bildpunktorientiert und somit auf einer sogenannten Low-Level-Ebene. Hier werden die Kanten unabhängig von den Objekten im Bild betrachtet.
Eine Lösung auf dieses Problem bietet die sogenannte semantische Kantendetektion an. Hier geht es darum, nur diejenigen Kanten zu finden, die zu einem bestimmten Objekttyp angehören. Im Grunde sind das nur die für den Betrachter wesentlichen Informationen. Daher arbeiten semantische Kantendetektoren auf einer High-Level-Ebene, wo sie Kanten in Beziehung zu den Objekten im Bild setzen. Nachfolgende Abbildung zeigt das Ergebnis eines semantischen Kantendetektors, der darauf trainiert wurde, nur Autos innerhalb von Bildern zu detektieren. Es erfolgt ein Vergleich zu dem Ergebnis des oben beschriebenen Canny-Kantendetektors.

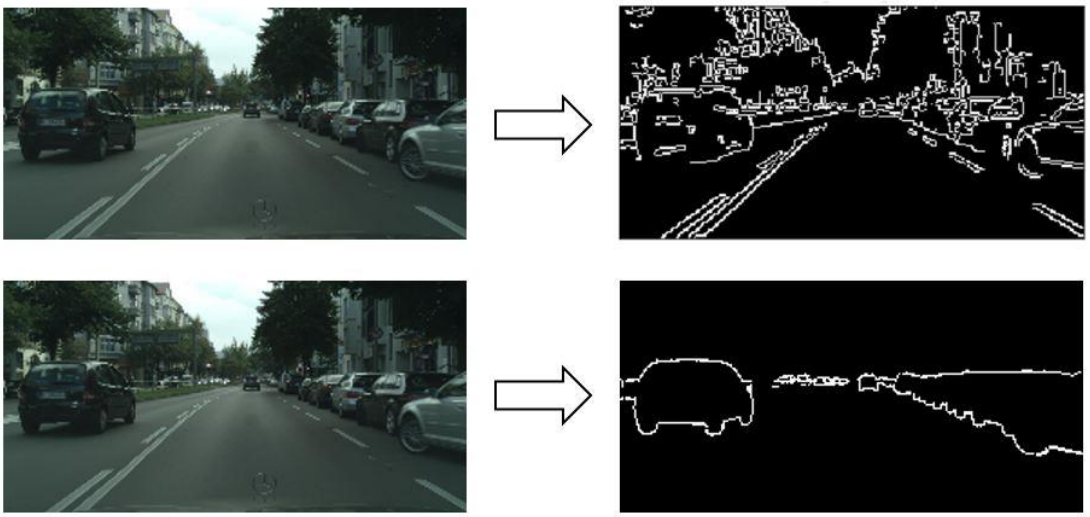


Abbildung 1-2 Vergleich: Semantische Kantendetektion vs. klassische Kantendetektion

Aus diesem Vergleich wird deutlich, dass eine starke Informationsreduktion erfolgt und der Betrachter sich auf die wesentlichen Informationen im Bild fokussieren kann.

1.1 Motivation

Die Kantendetektion ist ein fundamentales Problem im Bereich Computer Vision. Sie ist für eine Vielzahl von Aufgaben von großer Bedeutung und wird in zahlreichen Anwendungsgebieten der Computer Vision eingesetzt. Dazu gehören klassischen Aufgabenstellungen wie die Objekterkennung, Bildsegmentierung, Semantische Segmentierung, Bewegungsanalyse, Nachverfolgung, 3D-Rekonstruktion, 3D-Vision. Auch modernere Anwendungen profitieren von der Kantendetektion. Beispiele hierfür sind Mobile Computing, Bild-zu-Text-Analyse,

das Anwendungsfeld der Robotik und einige weitere, die im nachfolgenden detaillierter beschrieben werden.
Eine der weiteren interessanten Einsatzgebiete sind autonom fahrende Autos. Diese müssen in der Lage sein, andere Autos in ihrer Umgebung zu erkennen. Während sie Straßen befahren, müssen sie ständig auf andere Autos achten und immer im Blickfeld haben. Die Erkennung von anderen Autos ist eine grundlegende Eigenschaft für alle weiteren Manöver. Die von der Kamera erfassten Bilder enthalten zunächst alle Objekte in der Umgebung, wovon die meisten aber überhaupt nicht relevant sind. Hier könnte ein semantischer Kantendetektor zum Einsatz kommen, der darauf trainiert wurde, Autos in Bildern zu erkennen. Dieser filtert die in den Bildern nicht relevanten Informationen heraus und reduziert somit die Menge an zu verarbeitenden Daten gewaltig. Das Ergebnis ist ein Bild, das nur die Umrisslinien von Autos enthält. Auf Basis dieses Ergebnisses erhält das autonome Fahrzeug die Information, ob es andere Autos in der Nähe gibt. Ein darauf aufbauender Schritt könnte sein, dass das autonome Fahrzeug die Entfernung zu den detektierten Autos misst und überprüft, ob eine Reaktion notwendig ist. Die Echtzeitbedingung stellt hierbei deutlich höhere Anforderungen, unter anderem an die Kantendetektion.
Ein anderes potenzielles Einsatzgebiet ist im medizinischen Bereich, der ebenfalls von hoher Bedeutung ist. Die Erkennung von pathologischen Objekten in medizinischen Aufnahmen ist ein Beispiel für die medizinische Bildverarbeitung. Hier geht es darum, beispielsweise Tumore innerhalb der Bildaufnahmen zu erkennen. Ein Arzt wäre mit der Analyse von über 1000 Bildern einige Stunden beschäftigt. Ein Kantendetektor, der Tumore erkennt, würde den Arzt erheblich bei der Arbeit unterstützen. Insbesondere dann, wenn dieser Vorgang automatisiert würde. Im Anschluss könnte sich der Arzt direkt auf die relevanten Aufnahmen fokussieren und entsprechende Behandlungsmaßnahmen für die betroffenen Patienten in die Wege leiten.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es zu ermitteln, ob ein einfaches Neuronales Netz dazu geeignet ist, eine semantische Kantendetektion auf Autos durchzuführen. Im ersten Schritt soll zunächst eine passende Netzarchitektur ausgewählt werden. Anschließend sollen in unterschiedlichen Szenarien unterschiedliche Netzkonfigurationen trainiert, getestet und bewertet werden. Dabei wird ein Vergleich zwischen den Ergebnissen unterschiedlicher Szenarien gegeneinander erfolgen. Den Schwerpunkt der Untersuchungen bilden dabei die einzelnen Hyperparameter des Neuronalen Netzes. Pro Szenario soll jeweils ein einzelner Hyperparameter verändert werden, um deren Auswirkung auf die Ergebnisse besser nachvollziehen zu können und zugleich eine einfache Vergleichsbasis zu gewährleisten. Die auf diese Weise gewonnenen Erkenntnisse in den Szenarien sollen dann genutzt werden, um das Neuronale Netz schrittweise zu verfeinern. Primäres Ziel dabei ist die grundlegende Erkennung von Autos in Bildern.

1.3 Gliederung

Diese Arbeit ist in sechs Kapitel unterteilt, deren Inhalte im Folgenden zusammengefasst sind.

- **Kapitel 1** gibt eine Einführung in das Thema der semantischen Kantendetektion und erläutert die Zielsetzung dieser Arbeit.

- **Kapitel 2** gibt einen Überblick über die Grundlagen und vermittelt das notwendige Hintergrundwissen, die für das Verständnis dieser Arbeit notwendig sind.

- **Kapitel 3** setzt sich mit ähnlichen Arbeiten zu diesem Thema auseinander und stellt diese kurz vor.

- **Kapitel 4** gibt Informationen darüber, welche Arbeitsumgebung und softwareseitigen Mittel für diese Arbeit verwendet wurden.

- **Kapitel 5** gibt eine Einführung in die Vorgehensweise dieser Arbeit und erläutert die schrittweise Umsetzung. Daraufhin werden Untersuchungen in unterschiedlichen Szenarien durchgeführt, deren Ergebnisse im Anschluss bewertet und miteinander verglichen werden.

- **Kapitel 6** schließlich fasst die im vorherigen Kapitel erzielten Resultate zusammen und gibt einen Ausblick über die Möglichkeiten zur Weiterentwicklung.

2 Grundlagen und verwendete Techniken

2.1 Bilder und ihre Repräsentation innerhalb eines Computerprogramms

Dieser Abschnitt beschreibt den wesentlichen Aufbau von Bilddateien. In dieser Arbeit werden zum einen Farbbilder und zum anderen Schwarzweiß-Bilder verwendet.
Den Farbbildern liegt der RGB-Farbraum zugrunde. Bei diesem kann durch eine additive Mischung der Grundfarben rot, grün und blau jede andere Farbe erzeugt werden. Eine solche Grundfarbe wird als Farbkanal bezeichnet. Die Farbtiefe gibt an, wie viele verschiedene Abstufungen es von einer Grundfarbe gibt. Bei einer von Farbtiefe von 8 Bit beträgt die Anzahl der möglichen Abstufungen 256. Für die Darstellung eines ganzen Bildes wird eine Anzahl von Bildpunkten, sogenannten Pixeln, verwendet. Je mehr Pixel verwendet werden, desto feiner lassen sich kleinere Details im Bild abbilden und die auf dem Bild enthaltenen Objekte sind noch besser und klarer zu sehen. In diesem Zusammenhang wird auch von der Bildauflösung gesprochen. Die Bildauflösung gibt die Gesamtanzahl der Pixel eines Bildes an. Übliche Größen hierbei sind 1024x768 oder 1920x1080 für HD[1].
Bei einem Bild hat jeder Pixel eine bestimmte Farbe, der sich durch den Rot-, Grün- und Blauanteil an dieser Stelle ergibt. Dieser Anteil einer solchen Grundfarbe wird durch einen Zahlenwert vorgegeben, der bei einer Farbtiefe von 8 Bit immer zwischen 0 und 255 liegt. Die Datenstruktur für Farbbilder bei der Programmierung bilden dreidimensionale Arrays, wobei die Dimensionen durch Höhe, Breite und Anzahl der Farbkanäle des Bildes bestimmt werden. Nachfolgende Abbildung 2-1 Datenstruktur eines Teilausschnitts eines Farbbildes als dreidimensionales Arrayveranschaulicht beispielhaft, wie die Daten von einem Ausschnitt eines Farbbildes in Form eines dreidimensionalen Arrays abgespeichert sind.

[1] High Definition

Abbildung 2-1 Datenstruktur eines Teilausschnitts eines Farbbildes als dreidimensionales Array (3)

Bei Schwarzweiß-Bildern, die als Graustufenbilder abgespeichert sind, wird dasselbe Prinzip angewendet. Der einzige Unterschied hierbei ist lediglich, dass es nur noch einen Farbkanal gibt und die Farbe von jedem Pixel durch einen sogenannten Grauwert angegeben wird. Der Wertebereich eines Grauwerts ist ebenfalls von der Farbtiefe abhängig. Damit entfällt eine dritte Dimension bei der Abspeicherung eines Graustufenbilds in einem mehrdimensionalen Array. Abbildung 2-2 Datenstruktur eines Teilausschnitts eines Schwarzweißbildes als zweidimensionales Arraystellt beispielhaft die Speicherung der Daten eines Ausschnitts eines Graustufenbildes in einem zweidimensionalen Array dar.

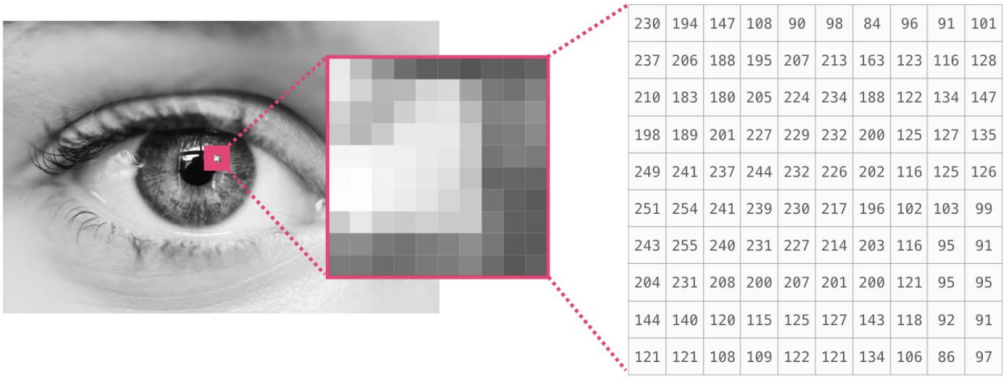


Abbildung 2-2 Datenstruktur eines Teilausschnitts eines Schwarzweißbildes als zweidimensionales Array (3)

Ein Grauwert von 0 entspricht hierbei einem schwarz. Der Grauwert 255 hingegen einem weiß.

2.2 Künstliche Intelligenz, Machine Learning und Deep Learning

In diesem Abschnitt werden grundlegende Begrifflichkeiten und deren Zusammenhänge erläutert. Daneben wird auch beschrieben, in welchen Bereich diese Arbeit einzuordnen ist.

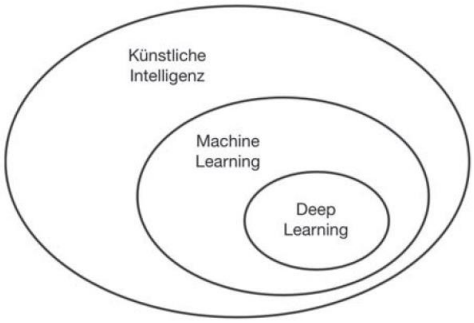


Abbildung 2-3 Zusammenhang zwischen Künstlicher Intelligenz, Machine Learning und Deep Learning (4)

Die Künstliche Intelligenz (KI) ist ein Teilgebiet der Informatik, der exponentiell gewachsen ist und in den letzten Jahren sehr große Erfolge verzeichnet hat und dadurch immer mehr Nachfrage gewann. Bei der KI geht es zunächst einmal nur darum, Computern das „Denken" beizubringen. Hierbei sollen von Menschen erledigte Arbeiten von Computern automatisiert gelöst werden. Nach außen hin sind damit Computer gemeint, die menschenähnliches Verhalten zeigen. In der Anfangszeit verwendeten KI-basierte Systeme lediglich einen fest vorgegebenen Regelsatz, der von Programmierern entwickelt wurde. Dieser Ansatz verzeichnete bei wohldefinierten Aufgaben wie dem Schachspielen Erfolge. Wurden jedoch die Aufgabenstellungen komplexer, wie es bei der Bildklassifizierung, Spracherkennung und Übersetzen von Fremdsprachen der Fall ist, stieß dieser Ansatz an seine Grenzen und wurde später durch einen neuen abgelöst, der als Machine Learning (ML) bekannt wurde. Dieser Ansatz ging der Frage nach, ob ein Computer von sich aus erlernen könnte, wie eine bestimmte Aufgabe erledigt wird. Hierzu werden beim ML dem Computer für eine konkrete Problemstellung Beispiele in Form von Datensätzen vorgegeben, aus dem es dann den Regelsatz eigenständig erlernt. Diesen Regelsatz wiederum kann der Computer dann auf vollkommen fremden Datensätzen anwenden und eigenständig Ergebnisse liefern.

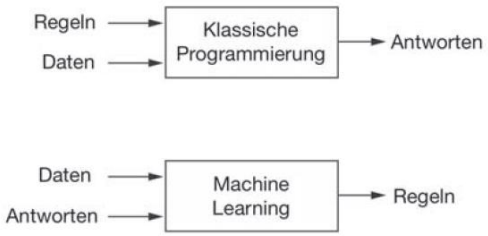


Abbildung 2-4 Unterschied zwischen klassischer Programmierung und Machine Learning (4)

Um den Unterschied von Deep Learning (DL) und ML besser erklären zu können, wird ML zunächst etwas detaillierter beschrieben. Beim ML müssen die Eingabedaten in Ausgabedaten umgewandelt werden. Anhand der vorgegebenen Beispiele sucht der Computer nach einer inneren statistischen Struktur, um einen Regelsatz für die Automatisierung dieser Aufgabe zu erlernen. Entscheidend hierbei ist, dass diese Umwandlung auf sinnvolle Weise erfolgen muss. Hierzu müssen sinnvolle und angemessene Repräsentationen der Eingabedaten erlernt werden, die die vorliegende Aufgabe vereinfachen. Eine Repräsentation ist lediglich eine andere Art, Daten zu betrachten. Aus technischer Sicht wird ML auch beschrieben als die Suche nach sinnvollen Repräsentationen der Eingabedaten in einer vorgegebenen Menge an Möglichkeiten (4). Das DL geht hier einen Schritt weiter und erweitert diesen Ansatz. Hier wird eine hierarchische Repräsentation der Eingabedaten erlernt und das vollständig automatisiert. Durch diese Vorgehensweise wurden beim DL große Erfolge in besonders komplexen Problemstellungen wie der Bildklassifizierung, Spracherkennung, Übersetzung von Fremdsprachen, Handschriftenerkennung, Gesichtserkennung, Verbesserungen der Suchergebnisse im Web u.v.m. erzielt.

Im Wesentlichen werden die Fortschritte beim ML von den nachfolgenden Faktoren beeinflusst (4):

- Verbesserte Hardware in Form von GPUs mit weitaus höheren Rechenleistungen
- Wachstum der bereitgestellten Datensätze und Benchmarks, die Forschergruppen zum Übertreffen motivieren
- Verbesserte Algorithmen, mit denen sich sehr große Tiefe Künstliche Neuronale Netze trainieren lassen

Die semantische Kantendetektion, um die es in dieser Arbeit geht, ist ein Teilbereich der Bildverarbeitung, für deren Problemstellungen sich DL in den letzten Jahren als sehr nützlich erwiesen hat. Genau dieser Ansatz wird auch in dieser Arbeit angewendet.

2.3 Künstliche Neuronale Netze

2.3.1 Das Neuron

Mithilfe künstlicher neuronaler Netze, im Folgenden vereinfacht Neuronale Netze (NN) genannt, wird versucht, die Arbeitsweise des menschlichen Gehirns nachzubilden. Bevor auf den Begriff des Neuronalen Netzes (NN) im Detail eingegangen wird, wird zunächst einmal erklärt, was ein einzelnes Neuron ist.
Der Aufbau eines einzelnen Neurons ist in Abbildung 2-5 Aufbau eines Neuron ersichtlich. Zu einem Neuron werden immer folgende Größen betrachtet: die Eingänge (x_i), die Gewichte (w_i), der Biaswert (b) und die Aktivierungsfunktion $\varphi(z)$.

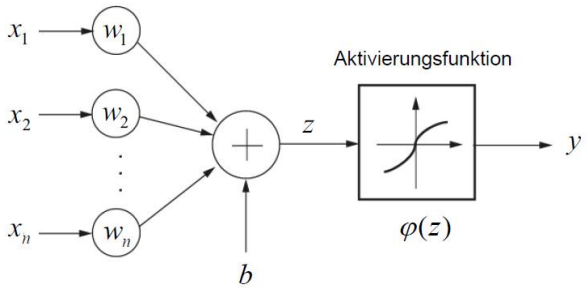


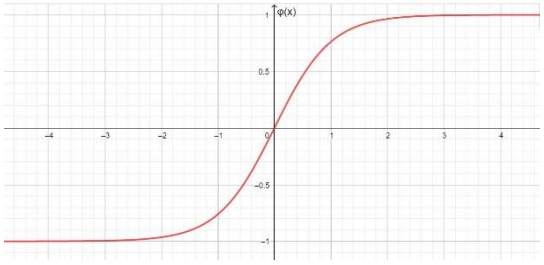
Abbildung 2-5 Aufbau eines Neurons (5)

Ein Neuron erhält von außen Eingangssignale, welche die Eingangswerte des Neurons repräsentieren. Diese Eingangswerte werden über Kanten zu einem Wert zusammengefasst. Jede Kante hat einen sogenannten Kostenfaktor, der als Gewicht bezeichnet wird. Hier wird zwischen positiven und negativen Gewichten unterschieden. Während positive Gewichte einen erregenden Einfluss auf das empfangende Neuron ausüben, üben negative Gewichte einen hemmenden Einfluss auf das empfangende Neuron. Neutrale Gewichte mit dem Wert 0 sagen aus, dass es gar keinen erregenden Einfluss auf das Neuron hat. Die Anzahl der Eingangswerte muss mit der Anzahl der Gewichte übereinstimmen, da jedem Eingangswert genau ein Gewicht zugeordnet ist. Der Biaswert b ist eine Konstante und hat entscheidenden Einfluss auf das Ausgabeverhalten eines Neurons. Die von einem Neuron repräsentierbaren Funktionen unterliegen der Einschränkung, das sie immer durch den Ursprung verlaufen müssen. Durch den Biaswert jedoch wird diese Einschränkung aufgehoben. Wie dem Bias und den einzelnen Gewichten ihre konkreten Werte jeweils zugewiesen werden, wird in dem Abschnitt 2.3.6 näher beschrieben.
Die in einem Neuron erfolgte Berechnung läuft auf diesem Wege ab: Die Eingangswerte werden mit ihren jeweiligen Gewichten multipliziert, der Biaswert dazu addiert und das Ergebnis z über eine hier nicht näher definierte Aktivierungsfunktion weitergeleitet. Das Ergebnis y wird dann weitergegeben. Diese Berechnung kann in zusammengefasster Form wie in der nachfolgenden Gleichung dargestellt werden.

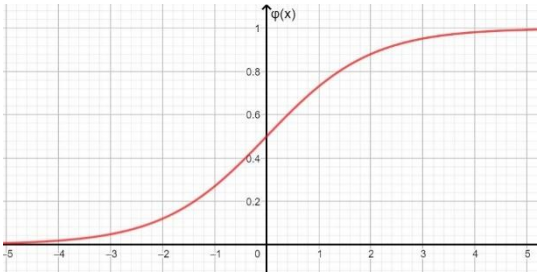
$$y = \varphi\left(\sum_{i=1}^{n} w_i \cdot x_i + b\right)$$

Aktivierungsfunktion

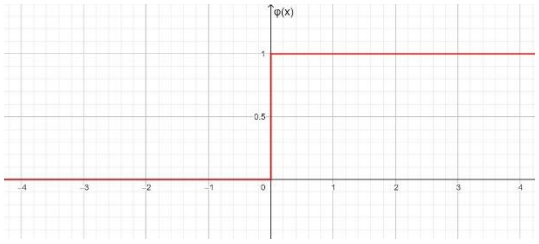
Es können verschiedene Funktionen als Aktivierungsfunktion verwendet werden. Eine Übersicht über mögliche Aktivierungsfunktionen ist in Abbildung 2-6 Verschiedene Aktivierungsfunktionen ersichtlich. Hier dran ist auch erkennbar, dass Aktivierungsfunktionen sehr unterschiedliche Eigenschaften aufweisen. Bei der Tangens Hyperbolicus-Aktivierungsfunktion $\varphi^{tanh}(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ sowie der logischen Aktivierungsfunktion, auch Sigmoid-Funktion genannt $\varphi^{sig}(z) = \sigma(z) = \frac{1}{1+e^{-z}}$ liegen alle Werte in einem bestimmten Bereich. Die binäre Schrittfunktion $\varphi(z) = \begin{cases} 0 : z < 0 \\ 1 : z \geq 0 \end{cases}$ weist ebenfalls diese Eigenschaft auf, hat aber den Nachteil der Nicht-Differenzierbarkeit. Andere gebräuchliche Namen hierfür sind binäre Treppenfunktion, Heaviside-Funktion oder auch Sprungfunktion. Die ReLu-Funktion $\varphi(z) = max(z, 0)$ liefert für negative Werte den Funktionswert 0 und ist ansonsten linear verlaufend. Hier entfällt die Einschränkung von Funktionswerten in einem bestimmten Wertebereich. Zudem bringt diese Funktion mit der Eigenschaft der Nichtlinearität einen entscheidenden Vorteil mit.



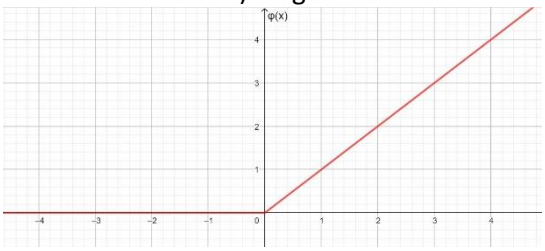
a) Tangens Hyperbolicus



b) Sigmoid



c) Binäre Schrittfunktion



d) Rectified linear unit (ReLU)

Abbildung 2-6 Verschiedene Aktivierungsfunktionen

Die Auswahl einer Aktivierungsfunktion hat einen entscheidenden Einfluss auf das Lernverhalten von NN (Vorgriff auf Abschnitt 2.3.6**Fehler! Verweisquelle konnte nicht gefunden werden.**). Die Eigenschaft der Differenzierbarkeit wird bei gradientenbasierten Optimierungsverfahren vorausgesetzt. Zudem gibt es einen weiteren entscheidenden Aspekt: wie

bereits beschrieben versuchen ML-Algorithmen nach Transformationen zu suchen, die für eine Aufgabe nützliche Repräsentationen der Daten liefern. Bei der Suche nach einer Transformation wird nur eine mögliche Menge an Operationen durchsucht, die Hypothesenraum genannt wird. Kann ein Layer (Vorgriff auf Abschnitt 2.3.2) nur lineare Operationen durchführen, ist der Hypothesenraum viel zu beschränkt. Auch über mehrere Layer hinweg bleibt die Operation weiterhin nur linear. Eine nichtlineare Aktivierungsfunktion verschafft dem Layer den Zugang zu einem viel umfassenderen Hypothesenraum, der für das Lösen von komplexen Aufgaben zwingend notwendig sind (4).
Aufgrund der oben beschriebenen Aspekte wird bei modernen NNs die ReLU-Aktivierungsfunktion, meist in Verbindung mit der Softmax- oder Sigmoid-Aktivierungsfunktion verwendet. Die Softmax-Funktion kommt dann zum Einsatz, wenn die Ausgabe des NNs als Wahrscheinlichkeit interpretiert wird, wird aber in dieser Arbeit nicht weiter betrachtet.

2.3.2 Netzarchitektur

Mit Netzarchitektur wird die strukturelle Anordnung eines NNs bezeichnet. Hierfür werden mehrere Neuronen zu einem NN zusammengeschlossen. In welcher Art und Weise das erfolgt, hängt davon ab, welche Aufgaben die Neuronen jeweils erfüllen. Es werden drei Arten von Neuronen unterschieden. Einmal gibt es Neuronen, die Eingaben von außen entgegennehmen. Daneben gibt es Neuronen, die Eingaben von anderen Neuronen entgegennehmen und diese an andere Neuronen weiterleiten. Die letzte Art von Neuronen ist lediglich für die Ausgabe zuständig. Diese Neuronen werden entsprechend ihrer Aufgaben in sogenannten Schichten (engl. layer), angeordnet. Folglich wird zwischen Eingabeschicht, versteckter Schicht und Ausgabeschicht unterschieden. Während die Anzahl der Ein- und Ausgabeschicht genau eins ist, ist die Anzahl der versteckten Schichten variabel. Im einfachen Fall ist gibt es nur Verbindungen von Neuronen zwischen benachbarten Schichten. Eine Beispielarchitektur mit einer Eingabeschicht (engl. input layer), versteckten Schicht (engl. hidden layer) und Ausgabeschicht (engl. output layer) ist in Abbildung 2-7 Beispielhafte Netzarchitektur eines NN mit je einer Eingabeschicht, versteckten Schicht und Ausgabeschicht zu sehen. Im Folgenden wird für den Begriff Schicht ausschließlich der englische Begriff Layer verwendet.

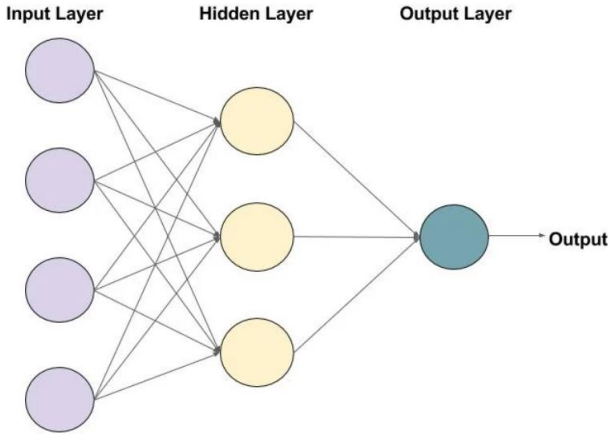


Abbildung 2-7 Beispielhafte Netzarchitektur eines NN mit je einer Eingabeschicht, versteckten Schicht und Ausgabeschicht (6)

Dabei führt jedes Neuron die in Abschnitt 2.3.1 beschriebene Berechnung durch und gibt das Ergebnis an die Neuronen weiter, mit denen es verbunden ist. Nach allen Berechnungen geben die Neuronen des Output-Layers einen Wert aus, der von den Entwicklern in Bezug auf die vorliegende Aufgabenstellung entsprechend interpretiert wird. Derartige Netze, bei der die Informationen in nur eine Richtung weitergegeben werden, werden vorwärtsgerichtete (engl. feedforward) Netze genannt. Daneben gibt es Netze, bei denen die interne Struktur etwas komplexer ist. Solche Netze können z.B. Rückkopplungen (direkt, indirekt oder lateral) oder shortcut Verbindungen enthalten.
Die Anzahl Parameter eines NN geben eine ungefähre Größenordnung an. Zum Beispiel enthält das VGG16-Netz (7), das im Rahmen einer Forschungsarbeit entwickelt wurde, 138 Millionen Parameter und ist damit schon ein recht großes NN. Eine Alternative besteht in der Betrachtung der Layeranzahl, was in manchen Fällen jedoch eine zu grobe Information liefern könnte, da die Anzahl Neuronen pro Layer stark variieren kann. Ein Beispiel für ein NN, das im Rahmen einer Forschungsarbeit von Microsoft entwickelt wurde, besteht aus bis zu 152 Schichten (8).
Noch einmal zurück zu den Aktivierungsfunktionen: In unterschiedlichen Layern werden üblicherweise unterschiedliche Aktivierungsfunktionen verwendet. In dieser Arbeit wird die ReLu-Aktivierungsfunktion (hidden layer) in Kombination mit der Sigmoid-Aktivierungsfunktion (output layer) verwendet.
Im Laufe der Zeit wurden speziellere Netzarchitekturen für verschiedenste Problemstellungen entwickelt, insbesondere für die Bildverarbeitung. Im Nachfolgenden werden einige vorgestellt.

2.3.3 Convolutional Neural Networks

Faltungsnetzwerke (engl. Convolutional Neural Networks (CNN)) sind eine spezielle Form von NNs. Initiiert von Fukushima (9) wurde sie später von LeCun und anderen Leuten in

einer Forschungsarbeit (10) verbessert. Ein Beispiel für ein solches CNN ist in Abbildung 2-8 CNN für Ziffernerkennung. Dieses CNN wurde mit Graustufenbildern der Größe 32x32 trainiert zu finden.

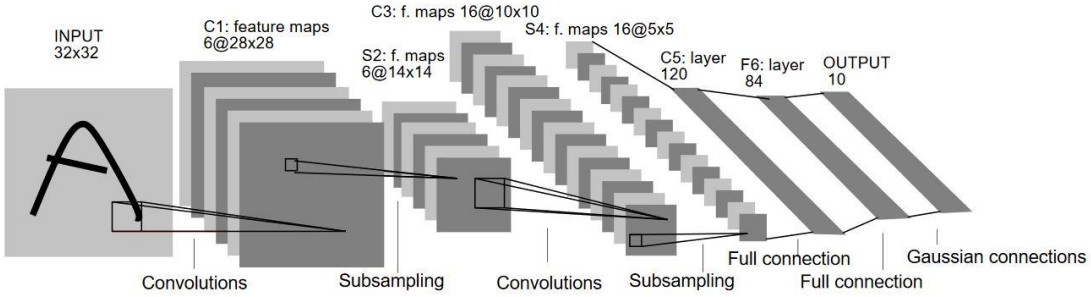


Abbildung 2-8 CNN für Ziffernerkennung. Dieses CNN wurde mit Graustufenbildern der Größe 32x32 trainiert (10)

Mit CNNs werden unter anderem neue Layertypen eingeführt. Ein typischer CNN beinhaltet Convolutional-, MaxPooling- sowie einen Fully-Connected-Layer. Diese sind auch in Abbildung 2-8 CNN für Ziffernerkennung. Dieses CNN wurde mit Graustufenbildern der Größe 32x32 trainiert enthalten, wobei mit Subsampling der MaxPooling-Layer gemeint ist.

Faltungsoperation

In CNNs kommt eine Technik zum Einsatz, die Faltung genannt wird. Die Faltung wird durch eine sogenannte Faltungsmaske durchgeführt. Die Faltungsmaske ist eine quadratische Zahlenmatrix mit einer ungeraden Dimension. Es gibt Faltungsmasken für unterschiedliche Zwecke, unter anderem für die Bildglättung, die Bildschärfung und die Kantenfilterung. Bei CNNs besteht die Hauptaufgabe der Faltung darin, wesentliche Informationen aus den Eingabedaten zu extrahieren. Alternative Bezeichnungen für diesen Begriff sind Faltungskern, Faltungskernel oder Filterkernel. Bei der Beschreibung der Faltungsoperation wird die Bezeichnung Faltungskern verwendet, während bei der Beschreibung des entsprechenden Layers der Begriff Filter verwendet wird. Diese unterschiedliche Begriffsverwendung wird mit einem besseren Verständnis in den jeweiligen Kontexten begründet.
Der Faltungskern h(m,n) wird zuerst über den oberen linken Quellbildabschnitt gelegt und angewendet . Die einzelnen Farbwerte des Quellbildabschnitts werden pixelweise mit den jeweiligen Werten der Faltungsmaske multipliziert und die einzelnen Ergebnisse anschließend aufsummiert. Das Endergebnis ist dann der Zielgrauwert des entsprechenden Zielbildpunktes g(x,y). Danach wird der Faltungskern um die Schrittweite, die mit dem gleichnamigen Hyperparameter (engl. stride) festgelegt wird, horizontal nach rechts bewegt, die Faltungsoperation ausgeführt, und zwar so oft, bis das Ende auf der rechten Seite erreicht wird. Danach wird der Faltungskern horizontal nach links bewegt, wandert die festgelegte Anzahl Stride nach unten und bewegt sich erneut horizontal nach rechts. Das wird solange durchgeführt, bis alle Pixel des Zielbildes berechnet wurden. Auf diese Weise ist das Ergebnisbild um je 2 Größeneinheiten in der Breite und der Höhe kleiner, weil die „Ränder feh-

len“. Durch das sogenannte Padding wird eingestellt, ob die ursprüngliche Breite und Höhe beibehalten werden soll. Ist die Beibehaltung gewünscht, wird der Rand des Eingabebildes künstlich mit Nullen aufgefüllt und so die Faltungsoperation angewendet.

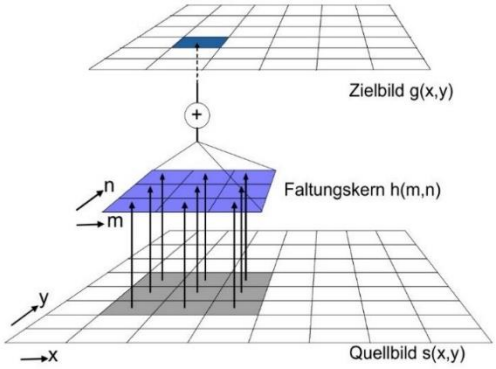


Abbildung 2-9 Beispielhafte Visualisierung einer Faltungsoperation im zweidimensionalen raum: Der Faltungskern wird über den grau markierten Quellbildausschnitt gelegt, die Faltungsoperation ausgeführt und der dunkelblaue Bildpunkt im Zielbild ausgerechnet (5)

Convolutional-Layer

Die Convolutional-Layer stellen im Wesentlichen den Kern von CNNs dar. Diese sind eine 3D-Anordnung von Feature-Maps. In den Feature-Maps sind die extrahierten Merkmale der Eingänge erfasst.

Die Parameter eines Layers innerhalb eines Feed-Forward-Netzes werden durch die Gewichte repräsentiert, die sich an den Kanten jeder Verbindung befinden. Die Parameter eines Convolutional-Layers werden hingegen durch einen Satz von Filtern gebildet. Ein Filter entspricht dem oben beschriebenen Faltungskern. Jeder einzelne Filter lernt, ein bestimmtes visuelles Merkmal im Bild zu erfassen. Ein Filter ist hinsichtlich der Breite und Höhe klein, erstreckt sich jedoch über die gesamte Tiefe der Eingangsdaten. Hat der Faltungskern die Höhe 3 und Breite 3 und beträgt die Anzahl der Eingabebilder 3, so ist die Größe des Faltungskerns 3x3x3. Im Vorwärtsdurchlauf wird der Filter über die Breite und Höhe der Eingangsdaten verschoben und die Faltungsoperation an jeder Position durchgeführt. Das Ergebnis dieser vollständigen Faltungsoperation ist dann eine Feature-Map, die die Antworten des Filters an jeder räumlichen Position wiedergibt. Die Faltungskerne werden trainiert. Dadurch entwickeln sich Filter, die „aktiviert“ werden, wenn es ein bestimmtes visuelles Merkmal sieht. Typischerweise hat ein Convolutional-Layer eine bestimmte Anzahl von Filtern, die der Anzahl Feature-Maps entsprechen, die erzeugt werden.

Nachfolgende Abbildung 2-10 Darstellung aller Operationen innerhalb eines Convolutional-Layers anhand von 3 Eingabebildern und der Erzeugung von 32 Feature-Maps fasst die eben beschriebenen Details zusammen und stellt im Wesentlichen dar, was innerhalb eines Convolutional-Layers passiert.

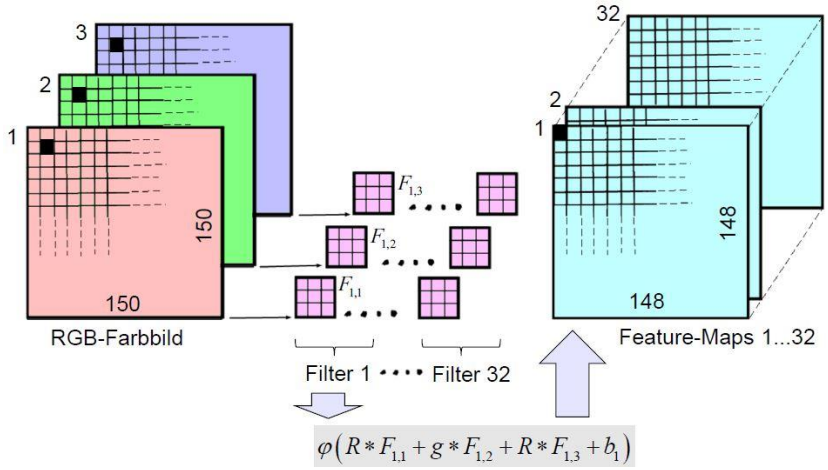


Abbildung 2-10 Darstellung aller Operationen innerhalb eines Convolutional-Layers anhand von 3 Eingabebildern und der Erzeugung von 32 Feature-Maps (5)

Hier wird Bild 1 mit dem Filter $F_{1,1}$ gefaltet, Bild 2 mit $F_{1,2}$ gefaltet, Bild 3 mit $F_{1,3}$ gefaltet. Die Einzelergebnisse werden aufsummiert und ein Biaswert dazu addiert und das Gesamtergebnis über die Aktivierungsfunktion weitergeleitet. Die Ausgabe der Aktivierungsfunktion entspricht dem Wert des Pixels an der entsprechenden Stelle in der Feature-Map 1. Nach allen Faltungsoperationen auf einem Convolutional-Layer werden aus M Eingabebildern N Feature-Maps erzeugt, wobei N der Anzahl der Filter entspricht. Anmerkung zur Darstellung: der Filter ist an sich dreidimensional, jedoch wurde er zur besseren Veranschaulichung an der dritten Dimension (Tiefe) „geteilt“. Dadurch wird besser ersichtlich, welcher Teil des Filters mit welchem Eingangsbild verbunden ist.
Jede Feature-Map wird durch eine Anzahl Neuronen berechnet, die dieselben Gewichtungen untereinander teilen (engl. shared weights). Dabei ist jedes Neuron eines Convolutional-Layers nur mit einer kleinen Region des Eingabebildes verbunden ist. Innerhalb dieser Region sind die Neuronen mit der gesamten Tiefe vernetzt. Diese räumlichen oder auch spatialen Regionen werden rezeptive Felder genannt. Das rezeptive Feld ist ein einstellbarer Hyperparameter, der der Größe des Faltungskerns entspricht.
Abbildung 2-11 Anordnung von Neuronen in einem 3D-Block innerhalb eines Convolutional-Layers zeigt dabei die Anordnung der Neuronen in einem sogenannten 3D-Block innerhalb eines Convolutional-Layers. Die in einem blauen Block befindlichen Neuronen sind jeweils mit demselben rezeptiven Feld der Eingabe verbunden. Dabei teilen sich bestimmte Neuronen die Gewichte, was die Parameteranzahl erheblich reduziert. Diese Unterteilung der Neuronen wird durch einen sogenannten Tiefenschnitt (engl. depth slice) umgesetzt. In Bezug auf Abbildung 2-11 Anordnung von Neuronen in einem 3D-Block innerhalb eines Convolutional-Layers entspräche dies 5 vertikalen Schnitten. Innerhalb eines solchen Tiefenschnitts (Höhe und Breite des 3D-Blocks mit der Tiefe 1) teilen sich die Neuronen die Gewichte. Die Aufteilung wird damit begründet, dass Bilder viele räumliche Beziehungen zwischen benachbarten Pixeln besitzen, die unabhängig von ihrer Position im Bild sind. Wenn die Erfassung eines Merkmals an einer bestimmten Position innerhalb eines Bildes

wichtig ist, sollte dessen Erfassung auch an einer anderen Position im Bild wichtig sein. Alle Neuronen innerhalb eines Tiefenschnitts erfassen immer dasselbe Merkmal im Bild, unabhängig von ihrer Position. Das führt wieder zum Vorteil, dass solche Merkmale immer erkannt werden. Dadurch entfällt auch die Notwendigkeit, dass dieselben Merkmale durch die Neuronen unterschiedlicher Tiefenschnitte erfasst werden müssen.

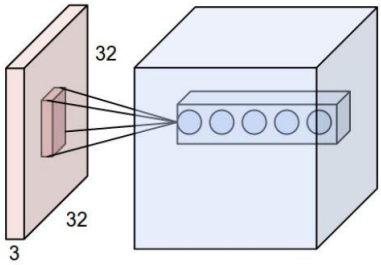


Abbildung 2-11 Anordnung von Neuronen in einem 3D-Block innerhalb eines Convolutional-Layers (11)

Pooling-Operation

Beim MaxPooling-Layer, auch Subsampling genannt, wird aus einem 3D-Block ein neuer kleiner 3D-Block berechnet. Bei der MaxPooling-Operation wird ähnlich wie bei der Faltungsoperation ein Filterkernel fester Größe über das Eingangsbild geschoben. Hier wird der maximale Wert vom bedeckten Ausschnitt des Eingangsbildes genommen, der zum entsprechenden Pixelwert im Ausgabebild wird. Abbildung 2-12 Beispielhafte Visualisierung einer MaxPooling-Operation zeigt die Arbeitsweise der MaxPooling-Operation bei einer Schrittweite von zwei und einem Filterkernel der Größe 2x2. Das Ziel dieser Operation ist die Datenkonzentration. Dabei wird die Größe der rezeptiven Felder verkleinert, während die Tiefe des 3D-Blocks beibehalten wird. Durch das Downsampling wird der Rechenaufwand in den Folgeschichten verringert, da die Parameteranzahl reduziert wird. Zusätzlich wird Überanpassung vermieden. Die Verwendung von MaxPooling-Layern zwischen Convolutional-Layern begünstigt zudem die Merkmalsabstraktion, da die Verkleinerung die Erkennung von Merkmalen höherer Ordnung (engl. high-level-features) ermöglicht.

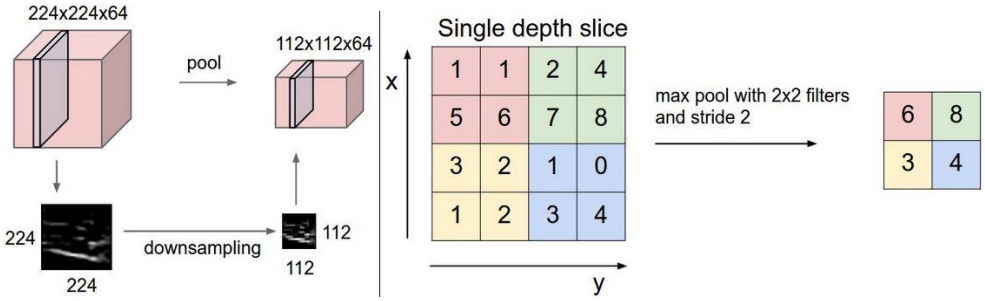


Abbildung 2-12 Beispielhafte Visualisierung einer MaxPooling-Operation (11)

Merkmalshierarchie

Werden mehrere Convolutional- und MaxPooling-Layer hintereinanderlegt, bilden die in den einzelnen Layern erlernten Merkmale eine Hierarchie. CNNs erlernen diese Merkmale

auf eine systematische Vorgehensweise. In den vorderen Layern werden ganz einfache Merkmale (engl. low-level-features), wie Ränder, horizontale oder vertikale Kanten erlernt. In den darauffolgenden Layern nimmt die Komplexität der erlernten Merkmale schrittweise zu. Diese setzen sich jeweils aus den vorherigen zusammen und werden dadurch zunehmend komplexer. Auf diese Weise erlernt das CNN komplexere und abstraktere Repräsentationen der Eingabedaten, die im Gesamtbild eine Hierarchie bilden (12). Daher ist die zugrundeliegende Idee bei CNNs das Erlernen einer Hierarchie von Merkmalsextraktionsstufen.
Bezogen auf einen beispielhaften Datensatz, der Tierbilder enthält, könnten die erlernten Merkmale wie folgt sein: In dem ersten Layer erlernt das CNN Merkmale wie ein Katzenauge, eine Hundenase oder ein Hundeohr. In dem darauffolgenden Layer werden Merkmale wie ein Hundekopf oder Katzenbeine extrahiert. Die erlernten Merkmale im letzten Layer könnten dann ein vollständiger Hund oder eine Katze sein.

Transposed Convolutional-Layer
Ein Transposed Convolutional-Layer setzt eine Faltungsoperation mit einem zusätzlichen Upsampling um. Es ist in der Art dem Gegenstück des MaxPooling-Layers recht ähnlich, unterscheidet sich aber in der grundlegenden Funktionsweise.

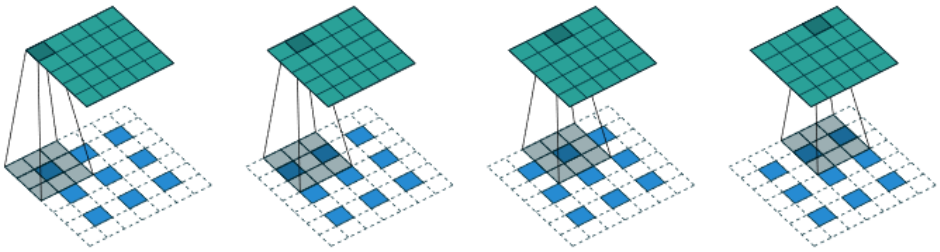


Abbildung 2-13 Funktionsweise eines Transpose-Layers: Upsampling eines 3x3 Bildes auf ein 6x6 Bild mit einem 3x3 Filterkernel und einem Stride von 2 (13)

Das Ziel ist die Vergrößerung der Dimensionen der Eingabe. Ein entscheidender Unterschied zum Convolutional-Layer in der Keras-Implementierung (Vorgriff auf Abschnitt 4.2.238) ist, dass der Stride-Hyperparameter nicht mehr die Schrittweite des Filterkernels angibt, sondern das Zeropadding zwischen den einzelnen Eingabebildpunkten. Dieser Filterkernel wird anders als beim MaxPooling-Layer zudem trainiert (mehr dazu in Abschnitt 2.3.6).

2.3.4 Autoencoder

Autoencoder (AE) sind eine Spezialform von NN. Die grundlegende Idee bei der Netzwerkarchitektur ist: Die Eingabedaten sind hochdimensioniert und werden an den Input-Layer übergeben. Diese werden durch das Netz durchgereicht, wobei die Anzahl der Neuronen mit zunehmenden Layern immer kleiner wird bis die Daten bei dem Engpass, auch Code

Layer oder latent space genannt, ankommen. Durch diesen Engpass werden die Informationen "durchgezwängt". Danach passieren die Daten das restliche Netz, wobei hier die Anzahl Neuronen mit jedem Layer immer größer wird. Am Ausgang ist die originale Eingangsgröße wiederhergestellt. Das Netz besteht grob aus zwei Teilen, nämlich dem Encoder
und dem Decoder.

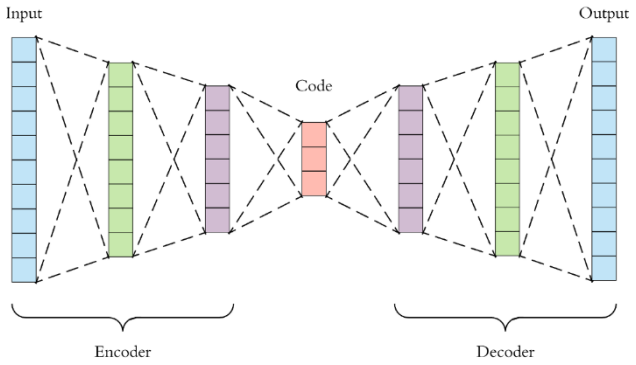


Abbildung 2-14 Netzarchitektur eines Autoencoders (14)

Die Besonderheit besteht darin, die Eingabedaten am Ausgang möglichst genau wieder so auszugeben. Diese Eigenschaft zeichnet einen klassischen AE aus.
Mit dem Engpass wird dafür gesorgt, eine niedrigdimensionale Repräsentation der Daten zu erlernen und somit nur die wesentlichen Informationen beizubehalten. Das muss auf eine Weise erfolgen, dass die Daten trotzdem am Ende rekonstruierbar sind. Daher wird der AE dazu gezwungen, abstrakte Zusammenhänge in den Daten zu finden, sie geeignet zu strukturieren und aufzubereiten.
Der Vorteil bei der Nutzung eines AE besteht darin, dass die Sammlung von gelabelten Trainingsdaten entfällt.

2.3.5 Convolutional Neural Network-Autoencoder

Ein Convolutional Neural Network-Autoencoder (CNN-AE) bildet die wesentlichen Aspekte eines normalen AE auf einen CNN mit einem Engpass ab. Der grundlegende Unterschied des CNN-AE gegenüber dem einfachen AE besteht folglich darin, dass die Eingangsdaten mehrdimensional sind. Auf der Encoder-Seite werden Convolutional- und MaxPooling-Layer verwendet, während die Decoder-Seite aus Transpose Convolutional-Layern besteht.
Alternativ kann die Decoder-Seite auch aus einer Anreihung von Deconvolutional-Layern und Unpooling-Layer bestehen. Bei dem Engpass handelt es sich typischerweise um einen Convolutional-Layer mit weniger Filterkerneln, die aus den bereits extrahierten Merkmalen die wesentlichen Merkmale erfassen muss.
Bei der Kantendetektion müssen ebenfalls die relevanten Informationen aus dem Eingabebild erfasst werden. Ein CNN bringt die Eigenschaft mit, nützliche Merkmale innerhalb von Bildern zu erfassen. Ein CNN-AE, der die beiden Aspekte Informationsreduktion und Merk-

malsgewinnung kombiniert, ist passend hinsichtlich der Aufgabenstellung der semantischen Kantendetektion und daher die in dieser Arbeit ausgewählte Netzarchitektur. Anzumerken ist dabei, dass der CNN-AE nicht im klassischen Sinne verwendet wird, da sich die Ein- und Ausgabebilder voneinander unterscheiden.
Im Nachfolgenden wird für einen CNN-AE vereinfacht der Begriff AE verwendet und für Engpass die Bezeichnung latent space.

2.3.6 Training Neuronaler Netze

Dieser Abschnitt beschreibt das Training von NNs und geht hierbei auf die notwendigen Begrifflichkeiten in Zusammenhang mit dem Training ein.
Ein NN sollte in der Lage sein, zu eingegebenen Daten die gewünschten Ausgaben zu liefern. Beim ML geht es darum, dass das NN in die Lage gebracht wird diese Aufgabe zu erfüllen, indem es aus Beispielen „lernt“. Mit Beispielen werden die Eingabedaten mit ihren dazugehörigen Ausgabedaten bezeichnet. Dieser Lernprozess mit all dem, was dazu gehört, wird als Training bezeichnet.

Klassifikation
Die Beziehung zwischen Ein- und Ausgabedaten wird in manchen Fällen durch eine Klassifikation beschrieben. Hier wird die Eingabe einer oder mehreren Klassen zugeordnet. Im Fall einer eindeutigen Beziehung zwischen Ein- und Ausgabedatum wird eine Einfachausgabeklassifizierung vor. Ein bekanntes Beispiel hierfür sind der MNIST-Datensatz, bei dem einem Bild eine Klasse von 0-9 zugeordnet wird. Kann ein Eingabedatum mehreren Klassen zugeordnet werden, so wird dies als Mehrfachausgabeklassifizierung bezeichnet.
In der Problemstellung, die in dieser Arbeit thematisiert wird, sind Ein- und Ausgabedaten jeweils Bilder. Die einzelnen Pixelpunkte, die entweder schwarz oder weiß sind, können vereinfacht als Klassen betrachtet werden.

Aufteilung des Datensatzes
Vor dem Training sollte der gesamte Datensatz aufgeteilt werden. Und zwar in Trainings-, Validierungs- und Testdaten. Die Trainingsdaten werden zum eigentlichen Training des NNs verwendet. Hier fließen die Informationen der Trainingsdaten direkt in das NN ein. Später wird das NN auf Basis der Validierungsdaten validiert. Hier wird auf Grundlage der Ergebnisse das NN entweder so genommen und ein Leistungstest durchgeführt oder verworfen und neu trainiert. Dabei fließen in das NN Informationen über die Validierungsdaten in indirekter Form ein. Für einen endgültigen Test wird der Testdatensatz herangezogen und das NN auf Grundlage dieser Daten bewertet. Es wird entschieden, ob das NN eine gute Generalisierungsfähigkeit besitzt.

Qualitätsmaße
Um die Leistungsfähigkeit eines NNs möglichst gut und präzise bewerten zu können, eignen sich abhängig von der vorliegenden Aufgabenstellung unterschiedliche Maße. Eines davon

ist die Genauigkeit (engl. accuracy), die die Anzahl der richtig klassifizierten Daten ins Verhältnis zu der Gesamtanzahl der Daten stellt. Daneben gibt es den Verlustwert (engl. loss), der die Summe der Fehler über den Trainingsdaten zeigt. Der Absolutwert des loss sagt nicht viel aus. Der einzige Zweck daran ist zu beobachten, dass dieser Wert während des Trainings fällt. Es dient als grobe Orientierung, um zu überprüfen, ob das Training in die richtige Richtung geht.
Bei einer ungleichmäßigen Verteilung der Klassen stellt die Accuracy jedoch keine gute zuverlässige Bewertungsgrundlage dar. Eine Alternative hierfür bildet die Precision und der Recall.

Trainingsablauf
Der Ablauf des Trainings ist wie folgt: Am Anfang werden alle Gewichte des NNs mit Zufallswerten initialisiert, um überhaupt einen Startpunkt festzulegen. Hier ist es gewöhnlich, wenn die Differenz zwischen Netzausgaben und den tatsächlichen Ausgaben am größten ist. Das Ziel des Trainings ist es, die bestmögliche Parametereinstellung (u.a. Gewichte und Biaswerte der Neuronen) zu finden, für die diese Differenz über allen Daten minimal wird.

Backpropagation-Algorithmus
Der Backpropagation-Algorithmus in Kombination mit dem Gradientenabstieg ist ein weit verbreitetes Lernverfahren für NN. Dieser lässt sich in die wesentlichen drei Schritte unterteilen:

1. Vorwärtsdurchlauf (eng. forward propagation)
2. Ermittlung des Fehlers
3. Rückwärtsdurchlauf (engl. backpropagation)

Beim Vorwärtsdurchlauf wird ein Eingabedatum dem NN übergeben, Layer für Layer durchgereicht und unter Verwendung der aktuellen Gewichte das dazugehörige Netzergebnis ermittelt. Anschließend wird die Abweichung des Netzergebnisses von der zu diesem Eingabedatum gewünschten Ausgabe berechnet. Diese Berechnung wird mittels einer Fehlerfunktion, auch Verlustfunktion genannt, durchgeführt. Die Fehlerfunktion dient als Maß dafür, wie genau das NN die Ausgaben vorhersagt. In diesem Zusammenhang bedeutet ein geringer Fehler hierbei, dass es zwischen der tatsächlichen und gewünschten Ausgabe einen geringen Unterschied gibt. Es gilt, das Ergebnis dieser Fehlerfunktion durch das Training zu minimieren. Eine Möglichkeit hierfür wäre, die eine analytische Suche nach der bestmöglichen Gewichtseinstellung (bezogen auf alle Gewichte des NN) für den kleinstmöglichen Fehler zu betreiben. Jedoch wäre diese Vorgehensweise in der Praxis sehr ungünstig, da die Komplexität dieses Verfahrens mit zunehmender Gewichtsanzahl exponentiell anwächst. Daher wurden im Laufe der Zeit unterschiedliche Optimierungsverfahren entwickelt, die anders an dieses Problem herangehen. Ein Beispiel dafür ist die gradientenbasierte Optimierung, der im nachfolgenden detaillierter beschrieben wird. Das Optimierungsverfahren führt basierend auf der Abweichung zwischen Netzergebnis und tatsächlichem Ergebnis Korrekturen an den Gewichtungen und dem Bias der einzelnen Neuronen durch, sodass der Fehler beim nächsten Durchlauf kleiner wird.

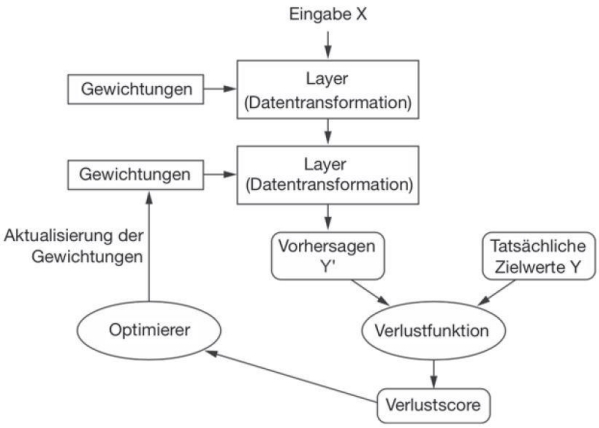


Abbildung 2-15 Lernprozess eines NNs (4)

Gradientenbasierte Optimierung

Das Ziel beim Gradientenabstieg ist, einen möglichst geringen Fehler innerhalb eines Fehlergebirges zu finden, indem Schritte in Richtung des steilsten Abstiegs durchgeführt werden. In der Abbildung 2-16 Visualisierung der einzelnen Schritte des Gradientenabstiegsverfahrens zum Finden von Minima ist der Prozess des Gradientenabstiegs beispielhaft dargestellt. Hier ist für die beiden Gewichte w_1 und w_2 das Fehlergebirge aufgetragen. Der rote Punkt kennzeichnet den Startpunkt, der sich aus der anfangs zufällig gewählten Gewichtskombination ergab. Der Startpunkt befindet sich an einer höheren Stelle auf dem Gebirge, da der Fehler am Anfang üblicherweise sehr groß ist. Im Verlauf dieses Verfahrens wird immer von der aktuellen Stelle heraus ein Schritt in Richtung des steilsten Abstiegs durchgeführt. Mit zunehmenden Schritten erfolgt eine Annäherung zum kleinstmöglichen Fehler, dem Tal des Fehlergebirges. Anzumerken ist hierbei, dass der Gradient nur die Richtung angibt, jedoch nicht die Schrittweite. Die Schrittweite ist mitunter ein entscheidender Faktor. Nachdem auf diese Weise der kleinstmögliche Fehler gefunden wurde, müssen die Anpassungen der Gewichte des NNs vorgenommen werden.

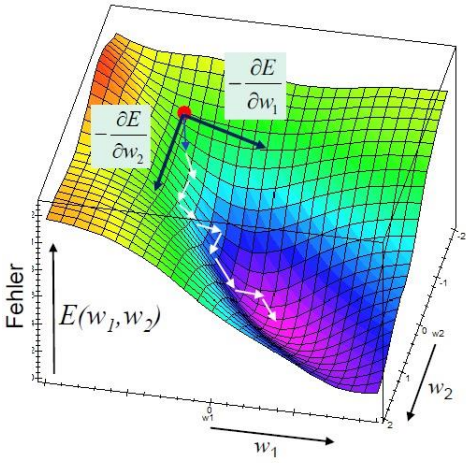


Abbildung 2-16 Visualisierung der einzelnen Schritte des Gradientenabstiegsverfahrens zum Finden von Minima (5)

Bei der Anwendung des Backpropagation-Verfahrens mittels gradientenbasierter Optimierung kann es auch zu Problemen kommen. Einer dieser Ursachen kann auf die Wahl der Schrittweite zurückzuführen sein, wenn diese vom Betrag des lokalen Gradienten abhängig gemacht wird. Ist diese zu klein, sind womöglich viele Iterationen zum Finden von Minima erforderlich, wodurch der Lernprozess erheblich verlangsamt wird. Zudem wird nicht sichergestellt, dass das globale Minimum gefunden wird. Ist sie zu groß, können steile Minima übersprungen werden. Ein anderes Problem ist, dass es in einem Tal mit zwei steilen Schluchten zur Oszillation kommen kann. Folglich kommt es zum Hin- und Herspringen, ohne dass das eine bessere Position zur Annäherung an das Minimum erreicht wird.

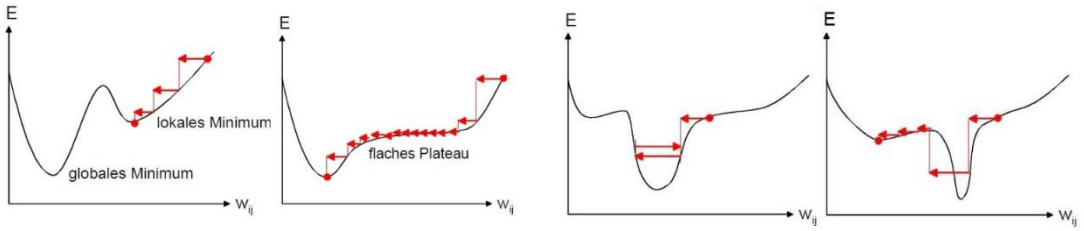


Abbildung 2-17 Potenzielle Probleme beim Gradientenabstiegsverfahren (5)

Eine Verbesserungsmöglichkeit in Bezug auf diese Probleme bietet der sogenannte Momentum-Term. Damit wird bei der Bestimmung der aktuellen Schrittweite die vorherige Schrittweite unter Verwendung eines Wichtungsfaktors mitberücksichtigt. Das hat einerseits den Effekt, dass bei Plateaus beschleunigt wird, indem zunehmend größere Schritte gemacht werden. Andererseits wird bei Richtungswechseln abgebremst und die Schrittweite verkleinert. Jedoch bleibt ein weiteres Problem bestehen: Schmale Minima werden bei einer ausreichend großen Schrittweite übersprungen, da hier kein Richtungswechsel er-

folgt. Um dieses Problem abzumildern, kann die Lernrate verringert werden, was mit einer Reduzierung der Lerngeschwindigkeit einhergehen würde.
Aus diesen Erkenntnissen zeigt sich, dass das Finden des globalen Minimums in der kürzest möglichen Zeit eine besondere Herausforderung stellt. Hierfür wurden unterschiedliche Varianten des Gradientenabstiegsverfahrens entwickelt, die neben dem Momentum-Term noch weitere Verbesserungen enthalten. Dazu zählen Adagrad, RMSProp und Adadelta und Adam, für deren detaillierte Beschreibung an dieser Stelle auf (15) verwiesen wird.

Einfluss der Aktivierungsfunktion auf den Trainingsprozess
Nun noch einmal zurück zu den Aktivierungsfunktionen. Ihr Steigungsverhalten hat eine wesentliche Bedeutung in Zusammenhang mit dem oben beschriebenen Gradientenabstiegsverfahren. Bei der Sigmoid-Funktion nähert sich die Ausgabe bei kleinen oder großen Eingaben an 0 bzw. 1 heran. An diesen Stellen ist die Steigung sehr gering. Derselbe Effekt tritt auch beim Tangens Hyperbolicus ein. Dies hat zur Folge, dass der Gradient gegen 0 geht. Somit ist das Neuron gesättigt und „lernt“ nicht mehr, denn die Fehlerrückführung mittels Backpropagation kommt genau in diesem Neuron „zum Stehen“. Folglich wird der Fehler an die davorliegenden Neuronen nicht mehr weitergeleitet und das „Lernen“ bleibt vereinfacht gesagt aus bzw. schreitet nur sehr langsam voran. Dieser Effekt kann insbesondere in den vorderen Layern des NNs auftreten.
Darüber hinaus gibt es ein weiteres Problem im Zusammenhang mit sehr tiefen Netzen. Besitzen diese mindestens vier Layer oder mehr, lassen sich diese mit Aktivierungsfunktionen wie der Sigmoid-Funktion nicht sinnvoll trainieren. Die Sigmoid-Funktion hat an der steilsten Stelle gerade einmal eine Steigung von 0,25. Das hat den Effekt, dass der mittlere Gradient (rückwärtsgehend von dem Output Layer) von Layer zu Layer kleiner wird. Dadurch nimmt die Lerngeschwindigkeit entsprechend von Layer zu Layer ab. Im schlimmsten Fall bleibt das Lernen in den vorderen Layern „aus“. Dies wird als „Vanishing Gradient Problem“ bezeichnet. Eine der möglichen Maßnahmen gegen dieses Problem ist die Verwendung der ReLU-Aktivierungsfunktion. Sie bringt die günstige Eigenschaft mit, dass sie im positiven Bereich immer eine Steigung von eins hat und dadurch der mittlere Gradient in den vorderen Layern nicht mehr so klein ist.

Konfigurierbare Parameter
Die Leistungsfähigkeit eines NNs hängt im Wesentlichen vom Finden der optimalen Hyperparameter ab. Die möglichen Parameter lassen sich in zwei Kategorien einteilen: die Hyper- und die Modellparameter. Es gibt noch zahlreiche andere Begriffe, die alle dasselbe meinen.

Hyperparameter
Die Hyperparameter werden vor dem Training von dem Entwickler festgelegt. Hierzu gehören die Wahl der Netzarchitektur, die Anzahl der hidden layer, die Anzahl Neuronen pro Layer, die Auswahl der Aktivierungsfunktion. Bei CNNs kommen folgende Parameter hinzu: Anzahl der Convolutional-Layer, Anzahl der Pooling-Layer, Größe der Faltungsmaske,

Schrittweite bei der Faltungsoperation, Anzahl der Filter. Daneben gibt es noch die Anzahl Epochen und die Chargengröße. Darüber hinaus gibt es noch weitere Parameter, die aber hier nicht weiter betrachtet werden.
Die Konfiguration der Hyperparameter des NNs ist entscheidend, da sie entscheidend für die Lernkapazität des Netzes sind. Die Anzahl der Input- und Output-Layer ist gewöhnlich eins. Die Anzahl Neuronen beeinflusst die Lernkapazität erheblich. Ein Neuron kann eine lineare Funktion realisieren. Mithilfe von mehreren Neuronen ist es möglich, komplexere Funktionen zu approximieren. Diese Eigenschaft ist für komplexe Aufgaben unerlässlich. Zu beachten ist, dass Netze mit einer zu großen Kapazität dazu neigen, sich überanzupassen. Die Auswahl der Aktivierungsfunktion nimmt wie bereits beschrieben ebenfalls eine bedeutsame Rolle ein. Sie hat unter anderem einen wesentlichen Einfluss auf das Training.
Den CNNs liegen die gleichen Merkmale zugrunde. Die Stride-Parameter sowie das Padding haben einen wesentlichen Einfluss auf die Größe der resultierenden Feature-Maps bei den Convolution-Operationen. Die Anzahl der Filter hat einen Einfluss auf Anzahl der Merkmale, die erfasst werden.
Mit Epoche ist ein ganzer Durchlauf gemeint, in dem alle Eingabedaten dem NN einmal übergeben wurden. Ein NN wird üblicherweise mehrere Epochen lang trainiert. Mit der Chargengröße, im Englischen batch size und im Nachfolgenden ausschließlich so genannt, wird festgelegt, wann die Aktualisierung der Gewichte erfolgen soll. Hat die batch size den Wert 16, wird damit ausgedrückt, dass nach 16 übergebenen Eingabedaten an das NN die Gewichte aktualisiert werden. Der Wert sollte auf jeden Fall größer als 1 und kleiner die Größe der Anzahl der Trainingsdaten sein. Als genauer, passender Wert wird ein Vielfaches von zwei empfohlen. Es ist üblich, dass die Gewichte innerhalb einer Epoche mehrmals aktualisiert werden. Diese Vorgehensweise wird als Minibatch-Verfahren bezeichnet.

Modellparameter
Die Modellparameter werden im Trainingsprozess schrittweise optimiert. Diese umfassen die Gewichte des NNs. Die Gewichte werden anfangs mit Zufallswerten belegt. Werden den Gewichten hohe Werte zugewiesen, kann der Wert, der innerhalb eines Neurons an die Aktivierungsfunktion übergeben wird, sehr groß werden. Ist die Steigung der gewählten Aktivierungsfunktion an dieser Stelle sehr klein, verlangsamt das den Lernprozess. Daher erfolgt die Initialisierung mit kleinen Werten. Die Gewichte werden dann vom Optimierer während des Trainings automatisch angepasst.
Bei den CNNs gehören die Faltungskerne mit zu den Modellparametern.

Überanpassung
Beim Training eines NNs gibt es zwei Größen, die im Konflikt zueinanderstehen: die Optimierung und die Generalisierungsfähigkeit des NNs. Die Optimierung verfolgt das Ziel, dass das NN bei den Trainingsdaten die bestmögliche Accuracy erzielt. Hingegen bezieht sich die Generalisierungsfähigkeit eines NNs darauf, dass das NN auch bei unbekannten Daten möglichst gute Resultate erzielt. Ein zu sehr an die Trainingsdaten angepasstes NN zeigt bei völlig fremden, ungesehenen Daten schlechte Ergebnisse. In diesem Fall hat das NN die Trai-

ningsdaten auswendig gelernt, aber besitzt eine sehr schlechte Generalisierungsfähigkeit. Dieser eintretende Effekt wird mit Überanpassung (engl. overfitting) bezeichnet. Überanpassung ist ein Problem, das bei NN am häufigsten eintritt. Es gilt, Überanpassung zu meiden.
Zur Vermeidung von Überanpassung müssen zunächst die Ursachen hierfür identifiziert werden. Überanpassung kann auf unterschiedliche Gründe zurückgeführt werden. Ist das NN sehr groß und besitzt sehr viele Parameter, hat es womöglich genug Kapazität sich die Trainingsdaten zu „merken" bzw. auswendig zu lernen und wird sich leicht an diese überanpassen. Der Überanpassungseffekt tritt auch dann ein, wenn das Netz zu viele Epochen lang trainiert wurde. Damit hatte das NN die Möglichkeit, sich Stück für Stück die Trainingsdaten zu merken und sich in Bezug auf diese zu optimieren.
Es wurden unterschiedliche Maßnahmen gegen die Überanpassung entwickelt. Im Folgenden werden nur die in dieser Arbeit verwendeten Maßnahmen kurz vorgestellt.

Normalisierung der Eingabedaten
Daten mit relativ großen Werten einem NN zu übergeben, kann Schwierigkeiten bereiten. Dadurch kann es zu umfangreichen Aktualisierungen des Gradienten kommen, die den Trainingsprozess erheblich verlangsamen. Es ist hilfreich, wenn die Daten stattdessen kleine Werte annehmen und in dieser Form an das NN übergeben werden. Hierzu ist eine Normalisierung der Eingabedaten erforderlich. Dieser Vorverarbeitungsschritt beschleunigt den Trainingsprozess und erleichtert dem NN das Lernen. Die Eingabedaten sind Farbbilder. die einen Rot-, Grün- und Blauanteil haben, deren Wertebereich von 0 bis 255 verläuft. Durch die Normalisierung werden alle Werte durch 255 dividiert und somit auf den Wertebereich von 0 bis 1 gebracht. Auf die gleiche Weise sollten alle Datensätze und unter anderem auch die Ausgabedaten normalisiert werden. Dies ist ohnehin sinnvoll, da es dann leichter ist, eine passende Aktivierungsfunktion zu finden. Befinden sich die Werte pro Pixel eines Ausgabebildes im Wertebereich von 0 bis 1, dann erweise sich die Verwendung der Sigmoid-Funktion als vorteilhaft. Diese gibt die Funktionswerte genau in diesem Bereich aus.

BatchNormalization
Die ReLU-Aktivierungsfunktion hat den Effekt, dass alle negativen Eingangswerte zu 0 werden und lässt die positiven Werte unverändert. Dies führt insgesamt zu einer ungleichmäßigen Werteverteilung. Der Mittelwert beträgt nicht mehr 0 und die Standardabweichung nicht mehr 1, sondern beide Werte werden verschoben. In den nachfolgenden Layern kann diese Veränderung zu Problemen führen. Diese ungleichmäßige Werteverteilung entsteht ebenfalls durch die Änderungen an den Parametern des letzten Layers beim Training. Dieses Problem wird als interne Kovarianzverschiebung (engl. internal covariate shift) bezeichnet (16). Eine Abhilfe hierbei schafft die BatchNormalization. Sie reduziert diesen Effekt. Durch sie wird die Ausgabe eines Layers, bevor sie an die nachfolgende Aktivierungsfunktion weitergereicht wird, normalisiert, sodass der Mittelwert nahe bei 0 und die Standardabweichung bei 1 liegt.

Die Normalisierung wird für die Daten zwischen zwei Layern jeweils pro Dimension durchgeführt sowie für jeden Minibatch beim Training. Später werden die optimalen Werte im Training durch die Selbstoptimierung der BatchNormalization eingestellt. Zudem wird dadurch der Trainingsprozess beschleunigt.
Die BatchNormalization ist in Form eines Layers umgesetzt.

2.4 Bewertungstechniken

Die Bewertung von Bildern stellt besondere Anforderungen, die es zu berücksichtigen gilt. Um einerseits einen Vergleich zwischen Ergebnissen unterschiedlicher Szenarien durchführen und zugleich die Ergebnisqualität möglichst aussagekräftig messen zu können, werden unterschiedliche Metriken verwendet. Die Anzahl der weißen und schwarzen Pixelpunkte in den Kantenbildern ist sehr ungleich verteilt. Eine Accuracy wird bei einem AE ohnehin nicht ausgegeben. Daher werden für alle Bilder die Precision- und Recall-Werte sowie der F1-Score berechnet, die den Schwerpunkt bei der Entscheidungsgrundlage bilden. Zum anderen wird eine subjektive Genauigkeit herangezogen. Durch eine geeignete Auswahl von Bildern werden Beispiele zu den Ergebnissen gezeigt. Diese sind bei der Bewertung von marginaler Bedeutung und spielen bei der Entscheidungsgrundlage eine untergeordnete Rolle.

2.4.1 Precision, Recall und F1-Score

Für jedes Bild werden die Precision und Recall-Werte ausgerechnet. Hierbei ist zu beachten, dass weiße Pixel Autos repräsentieren, während schwarze Pixel hingegen bedeuten, dass es an diesen Stellen keine Autos gibt.
Precision wird hier definiert als die Anzahl weißer Pixel, die auch als weiße Pixel detektiert werden (Positives) im Verhältnis zu allen detektierten weißen Pixeln (alle erkannten Positives). Die Precision wird dann insbesondere herangezogen, wenn falsche Positives gefährliche Zustände hervorrufen können. Die Precision kann als ein Maß für die Genauigkeit angesehen werden.

$$P = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

Recall wird definiert als die Anzahl weißer Pixel, die auch als weiße Pixel detektiert werden (Positives) im Verhältnis zu allen wirklichen weißen Pixeln (alle tatsächlichen Positives). Dieser kommt dann üblicherweise zum Einsatz, wenn keine Positives übersehen werden sollen, weil die Kosten beim Eintreten solcher Fälle hoch sein könnten. Der Recall ist so gesehen ein Maß für die Vollständigkeit.

$$R = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Sind beide Werte gleich eins, dann ist die Vorhersage des NNs perfekt. Jedoch ist das folgende Verhalten im Zusammenhang mit beiden Werten in der Praxis eher üblich: Steigt der eine Wert an, fällt der andere Wert.

Die Einordnung der Begriffe ist in der folgenden Abbildung zu finden.

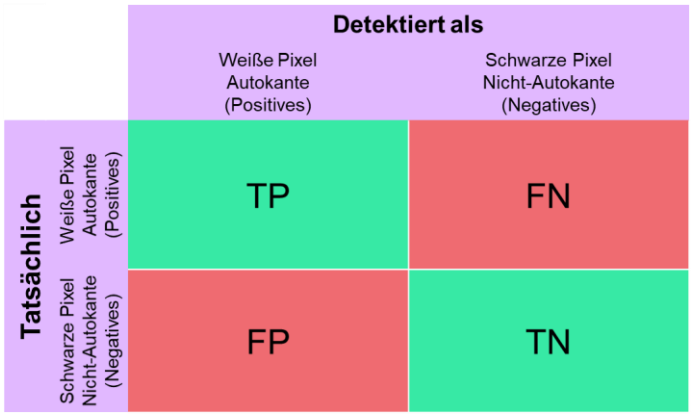


Abbildung 2-18 Zusammenhang zwischen true positives, false negatives, false positives und true negatives

Die Ergebnisinterpretationen beider Werte in Bezug auf diese Arbeit sind in folgender Tabelle zusammengefasst.

Maß	**Wert**	**Interpretation**
Recall	1	Alle Autos wurden erkannt.
Recall	0	Alle Autos wurden übersehen.
Precision	1	Überall wo kein Auto ist, wurden auch keine Autos erkannt.
Precision	0	Überall wo kein Auto ist, wurden Autos erkannt.

Tabelle 2-1 Interpretationen der Recall- und Precision-Werte im Zusammenhang mit dieser Arbeit

Neben diesen zwei Metriken wird noch eine weitere Metrik verwendet, nämlich der F1-Score. Der F1-Score ist die Kombination von Precision und Recall unter Verwendung des harmonischen Mittels. Der größtmögliche Wert für den F1-Score ist 1, das einen perfekten Recall- und Precision-Wert (je 1) impliziert. Der kleinstmögliche Wert von 0 ergibt sich, wenn entweder die Precision gleich 0 oder der Recall gleich 0 ist.

$$\mathrm{F} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

Die drei Werte werden für alle Bilder mit 100 multipliziert, um eine Prozentzahl zu erhalten. Für die Darstellung der Precision- und Recall-Werte wird normalerweise eine sogenannte Precision-Recall-Curve verwendet, aus der abgelesen werden kann, in welchem Verhältnis

Precision und Recall zueinanderstehen. Das Problem bei dieser Darstellung ist, dass aus der Kurve nicht hervorgeht, auf welche Anzahl von Bildern das an den jeweiligen Stellen in der Grafik zutrifft. Um eine bestmögliche und zugleich einfache Darstellung dieser drei Kennzahlen für alle Bilder zu ermöglichen, wird je ein Histogramm für die Precision, für den Recall und für den F1-Score verwendet.

Ausnahmen: Treten dann ein, wo in bestimmten Fällen beim Recall oder der Precision der Nenner in der Formel 0 wird. Bei Bildern, die weder true positives noch false negatives beinhalten, wird die Berechnung des Recalls nicht durchgeführt. Dies trifft bei GT-Kantenbildern ohne Autos zu, also komplett schwarzen Bildern. Ebenso entfällt die Berechnung der Precision, sollte es für Bilder weder true positives noch false positives geben. Dies ist dann der Fall, wenn das Ergebnisbild vom AE komplett schwarz ist. Ist die Berechnung der Precision oder des Recalls für ein Bild nicht möglich oder sind beide Werte gleichzeitig 0, entfällt auch die Berechnung des F1-Scores.

2.4.2 Subjektive Genauigkeit

Da es sich bei den Ergebnissen um Bilder handelt und diese auch einen optischen Eindruck beim Betrachter hinterlassen, wird zusätzlich eine sogenannte subjektive Genauigkeit verwendet. Diese soll lediglich durch eine Auswahl von geeigneten Einzelbildern Beispiele für die drei oben beschriebenen Metriken zeigen. Bei der eigentlichen Bewertungsgrundlage ist sie von marginaler Bedeutung.

Verwendete Bilder für die Ergebnisse
Um die Ergebnisqualität des NNs auf subjektive Weise vernünftig und weitestgehend präzise darstellen zu können, muss die Auswahl der Bilder nach ganz bestimmten Aspekten erfolgen. Hierzu werden die Bilder in drei Kategorien aufgeteilt:

- Bilder mit vielen Autos
- Bilder mit wenigen und teilweise einzelnen Autos
- Bilder ohne Autos

In jeder dieser Kategorien müssen zudem Bilder enthalten sein, die die unterschiedlichen Konditionen abdecken, unter denen die Aufnahmen entstanden sind. Dazu gehören unter anderem unterschiedliche Verkehrssituationen, unterschiedliche Gegenden, das Tageslicht und Schatten. Zudem sollten auf den Bildern weitere unterschiedliche Objekte enthalten sein wie Personen, Gebäude, Verkehrsschilder etc.
Für jedes untersuchte Szenario werden für jede der drei Kategorien jeweils sieben Bilder herausgesucht und diese zur Ergebnisdarstellung verwendet. Hierfür werden pro Reihe jeweils immer folgende drei Bilder gezeigt:

- Das Eingabebild in Farbe, zu dem die semantische Kantendetektion erfolgt
- Wunschergebnisbild, gegen den der Vergleich erfolgt (im Weiteren wird dies als Ground Truth Kantenbild, kurz GT-Kantenbild, bezeichnet
- die nachbereitete Netzausgabe bzw. das aktuelle Ergebnisbild vom AE, das nachbereitet wurde

Um eine möglichst gute und zugleich faire Vergleichsbasis zu gewährleisten, werden die ausgewählten Bilder für jedes untersuchte Szenario immer dieselben sein. Nur dadurch lassen sich Verbesserungen als auch Verschlechterungen in der Ergebnisqualität feststellen. Ebenfalls wird damit ein besserer Bezug zu den Precision-, Recall- und dem F-Werten hergestellt.

3 Stand der Technik

In diesem Kapitel werden kurz einige Arbeiten zur semantischen Kantendetektion vorgestellt, die in den letzten Jahren von unterschiedlichen Forschungsgruppen veröffentlicht wurden. Für detailliertere Informationen wird auf die entsprechende Quelle verwiesen.

3.1 VGG16

Einige der im nachfolgenden vorgestellten Arbeiten nutzen unter anderem das VGG16-Netz, das hier zum besseren Verständnis kurz vorgestellt wird.
VGG16, erstellt durch die **V**isual **G**eometry **G**roup des Departments of Engineering Science der University of Oxford im Rahmen des "ImageNet Large Scale Visual Recognition Challenge", ist ein aus 16 Layern bestehendes NN (7). Abbildung 3-1 Architektur des VGG16-Netzes veranschaulicht die Architektur des Netzes, das insgesamt über 134 Millionen Parameter besitzt.

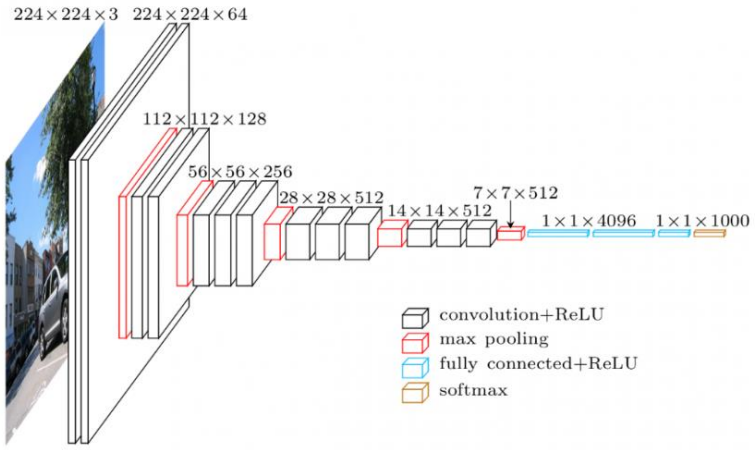


Abbildung 3-1 Architektur des VGG16-Netzes (17)

Dieses Netz ist anwendbar bei Problemstellungen der Objektklassifizierung. Der Teil mit den Convolutional Layern extrahiert die Merkmale der Eingabebilder. Diese werden dann an den Fully-Connected-Layer weitergereicht, der anhand dieser Informationen eine Klassifikation durchführt. Ein Anwendungsbeispiel hierfür ist die Bildklassifikation, für die unter anderem der ImageNet-Wettbewerb existiert. In dem ImageNet-Datensatz werden über 14

Millionen Bilder bereitgestellt, die zu über 20000 unterschiedlichen Klassen angehören. Die Herausforderung besteht darin, dass jedes Bild seiner Klasse korrekt zugeordnet werden muss. VGG16 erzielte bei diesem Wettbewerb gute Ergebnisse.
Inzwischen wurde das neuere VGG19-Netz entwickelt, das seinerseits wiederum aus 16 Convolutional sowie 3 Fully-Connected-Layern besteht.
Für detailliertere Informationen zu VGG16 wird an dieser Stelle auf (7) verwiesen.

3.2 TD-CEDN

Das TD-CEDN, die Kurzform für “Top-Down Fully Convolutional Encoder-Decoder Network”, ist ein sogenanntes Ende-zu-Ende Konturerkennungssystem. In dieser Arbeit wurde das Problem der Konturerkennung als binäres Klassifizierungsproblem pro Bildpunkt formuliert, wobei „0“ für Nicht-Kontur und „1“ für Kontur stehen. Dieses Netzwerk nutzt für die pixelweise Vorhersage des Ausgabebildes die Merkmale von allen Layern.
Folgende Punkte drücken die Besonderheiten dieses Netzes aus:

- Erlernen von Multi-Scale Features
- Erlernen von Multi-Level Features

Die Architektur ist in Abbildung 3-2 Netzarchitektur des TD-CEDN veranschaulicht.

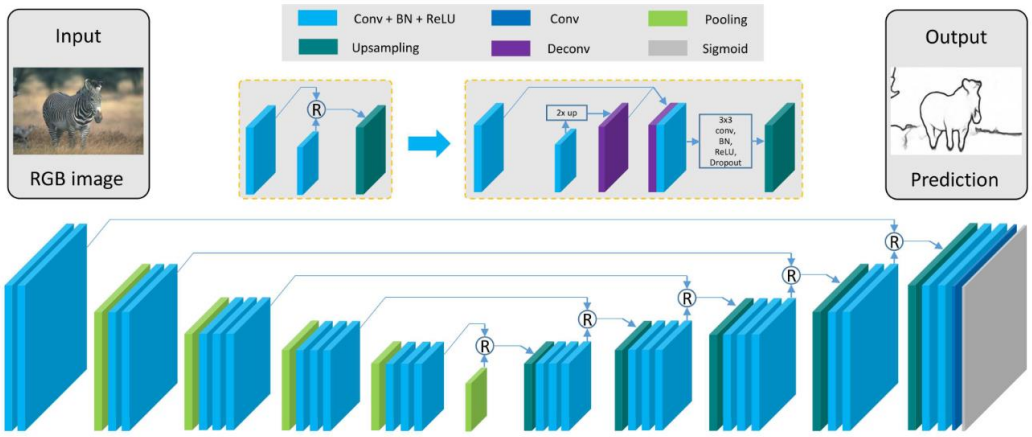


Abbildung 3-2 Netzarchitektur des TD-CEDN (18)

Das Netzwerk besitzt eine symmetrische Struktur und arbeitet vollständig faltungsbasiert. Der Encoder besteht vollständig aus dem VGG16-Netz bis einschließlich dem letzten MaxPooling-Layer. Die Gewichte werden mit denen vom VGG16 initialisiert. Zudem wird zwischen jedem Convolutional- und ReLU-Layer BatchNormalization angewendet, um die interne Kovarianzverschiebung zu reduzieren. Der Decoder ist vereinfacht gesehen eine Spiegelung des Encoders. Es besteht aus einer Abfolge von Deconvolutional- und BatchNormalization-Layern sowie der ReLU-Aktivierungsfunktion.

Mit diesem Netzwerk wird eine Top-Down-Lernstrategie verfolgt. Zunächst erlernt das Netz im Encoder einfache Merkmale, die mit zunehmenden Layern komplexer werden und am Ende des Encoders eine grobe Feature-Map ergeben. Diese wird dann im Decoder unter Zuhilfenahme der einfacheren Merkmale aus den niedrigeren Layern vom Encoder schrittweise verfeinert. Hierbei erfolgt im Decoder das Upsampling in der Weise, dass die Ausgabe des davor liegenden Convolutional-Layer mit der Ausgabe des entsprechenden Gegenstücks vom Encoder konkateniert wird. Der darauffolgende Convolutional-Layer erlernt durch das Training mithilfe der konkatenierten Feature-Map, noch feinere Ergebnisse zu erzeugen. Durch diese besondere Upsampling-Strategie erlernt und optimiert das Netzwerk die aussagekräftigen Merkmale der visuellen Wahrnehmung von der High-Level-Ebene hingehend zur Low-Level-Ebene top-down-artig.
Weitere Details zu diesem Netz können in (18) entnommen werden.

3.3 Efficient Boundary Detection from Deep Object Features and it's Applications to High-Level-Vision

Das in dieser Arbeit veröffentliche Netzwerk verfolgt einen High-For-Low Ansatz, wo Objektmerkmale höherer Ordnung für die Erkennung von Grenzmarkierungen auf Low-Level-Ebene verwendet werden, was mit „High-For-Low Boundaries" bezeichnet wird. Die hier verwendete Definition für das Problem der Erkennung von Grenzmarkierungen ist: der Anteil der von Menschen durchgeführten Annotationen, die sich darüber einig, dass ein bestimmter Pixel eine Grenze darstellt. Das Lernziel ist die Imitation der menschlichen Entscheidungsfähigkeit.

Folgende Punkte kennzeichnen die Besonderheiten dieses Netzwerks:

- Vorhersagen von Grenzmarkierungen niederer Ordnung unter Verwendung von Objektmerkmalen höherer Ordnung
- Die so erlernten Grenzmarkierungen können gezielt zur Steigerung der Ergebnisqualität von anderen Wahrnehmungsaufgaben auf höherer Ordnung (engl. high-level vision tasks) verwendet werden. Dazu zählen unter anderem die semantische Segmentierung und semantische Grenzmarkierung

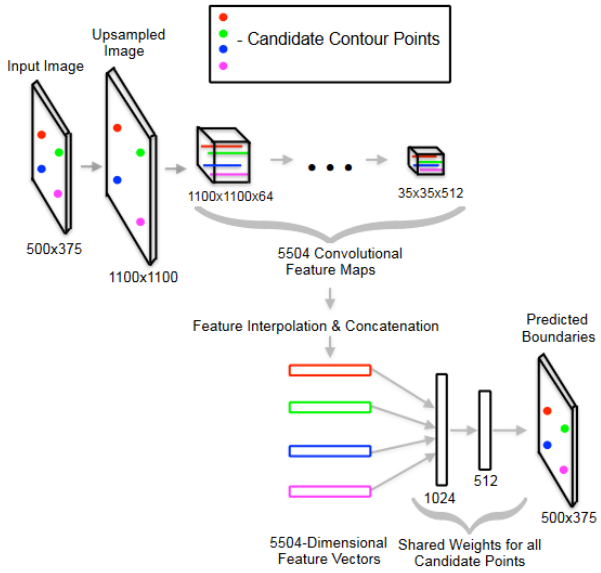


Abbildung 3-3 Netzarchitektur der Forschungsarbeit “Efficient Boundary Detection from Deep Object Feature” (19)

Die Idee für die Arbeitsweise des Netzes lässt sich daraus ableiten, dass Menschen bei der Entscheidung, ob ein Pixel zu einer Kontur gehört, Schlussfolgerungen auf einer sogenannten Objektebene ziehen. Hierzu wird das vortrainierte VGG-Netz verwendet, dass die Fähigkeit besitzt, Objektklassen zu erkennen (die über 1000 vom ImageNet-Datensatz). Das ist dadurch möglich, weil es Merkmale höherer Ordnung bzw. auf Objektebene extrahiert hat. Die Fully-Connected-Layer vom VGG-Netz wurden hingegen nicht übernommen, da sie keine spatialen Informationen beibehalten, die aber bei der Erkennung von Grenzmarkierungen eine entscheidende Rolle spielen. Stattdessen werden an das Ende zwei Fully-Connected-Layer getan, durch die eine Vorhersage anhand von Kriterien auf der Basis menschlicher Beurteilung durchgeführt wird.
Die Arbeitsweise des Netzes lässt sich in die folgenden Schritte unterteilen:

- Auswahl möglicher Kandidaten für mögliche Konturpixelpunkte
- Extraktion von Merkmalen höherer Ordnung auf Objektebene
- Anwendung von Feature Interpolation
- Vorhersage und Erzeugung einer Boundary-Probability-Map

Zu Beginn werden mithilfe des SE Edge-Detektors (20) ein Satz an möglichen Kandidaten für Konturpixelpunkte extrahiert. Bevor diese an die 16 Convolutional-Layer vom VGG-Netz übergeben werden, wird das Bild durch Upsampling höher dimensioniert, z.B. auf die Bildgröße 1100x1100. Der Grund hierfür besteht darin, dem Effekt entgegenzuwirken, dass bei den nachfolgenden MaxPooling-Operationen vom VGG-Netz wesentliche Informationen verloren gehen könnten. Nachdem die Daten das VGG-Netz durchlaufen haben, wird für

jeden Kandidaten der entsprechende Bildpunkt in jeweils allen 5504 Feature-Maps gesucht. Aufgrund der Dimensionsunterschiede der Feature-Maps sind die Entsprechungen folglich nicht exakt. Daher wird hier „Feature Interpolation“ angewendet. Dabei werden pro Feature-Map die nächstgelegenen vier Punkte ermittelt und die Aktivierungswerte gemittelt. Das Endergebnis dieser Operation ist ein 5504-dimensionaler Vektor pro Kandidatenpunkt. Jede dieser Vektoren wird dann an die zwei Fully-Connected-Layer weitergereicht und eine Vorhersage durchgeführt, wodurch eine sogenannte Boundary-Probability-Map erzeugt wird. Diese Werte repräsentieren den imitierten Anteil der Menschen, die pro Pixel jeweils entschieden haben, ob es sich dabei um eine Grenzmarkierung handelt oder nicht.
Das Vorgehen, dass von vornherein mögliche Konturpunkte ausgewählt werden, reduziert die Rechenzeit enorm. Durch die Anwendung der Interpolation auf alle Feature-Maps können die Merkmale für alle Kandidatenpunkte in einem einzigen Durchlauf berechnet werden, was zu einer effizienten Vorhersage führt. Insgesamt beträgt die Laufzeit auf einer K40 GPU 1,2 Sekunden.

3.4 CASENet

CASENet steht für „Deep Category-Aware Semantic Edge-Detection” und ist eine Ende-zu-Ende-Netzwerkarchitektur zur kategorie-bewahrenden semantischen Kantenerkennung. Hier wird das Problem der Kantenerkennung als Multi-Label-Klassifikation formuliert, wo einem einzelnen Pixel eine oder mehrere Klassen zugeordnet werden können. Hierfür wird jeder Kantenpixel durch einen Vektor bezeichnet, dessen einzelne Elemente die Assoziationsstärke des Pixels mit den verschiedenen semantischen Klassen ausdrücken.
Die Netzarchitektur mit ihren einzelnen Bestandteilen ist in Abbildung 3-4 CASENet Netzarchitektur mit den einzelnen Bestandteilen . Durchgezogenes Rechteck: Aneinanderreihung von CNN-Layern. Verringerung der Breite entspricht Verkleinerung der Feature-Map um den Faktor 2. Pfeil: Anzahl Channels. Blaues, durchgezogenes Rechteck: Stapel von ResNet-Layern. Lila Rechteck: Klassifikationsmodul des CASENet-Netzwerks. Graues, durchgezogenes Rechteck: Side-Merkmalsextraktionsmodul. Roter, gepunkteter Umriss: Überwachung der Ausgabe durch die Multi-Label-Loss-Funktion. Dunkelgrünes, durchgezogenes Rechteck: Fusioniertes Klassifizierungsmodul, das eine 1x1 Convolution mit K Gruppen durchführt dargestellt.
Die Besonderheiten dieser Arbeit bestehen in den folgenden Punkten:

- Architektur ResNet-101 wurde adaptiert, für eine verbesserte Bereitstellung von Low-Level-Kanteninformationen (u.a. detaillierte Kantenlokalisierungs- und Strukturinformationen) an die höheren Layer
- eine Skip-Layer-Architektur, die es ermöglicht, dass die Klassifikation auf dem letzten Layer unter Zuhilfenahme von den erlernten Merkmalen aus den niederen Layern entscheidend erweitert wird
- Verwendung einer eigendefinierten Multi-Label-Loss-Funktion (Verlustfunktion)

Das Besondere an dieser Architektur ist, dass die Informationen eines Blocks an mehrere Layer weitergereicht werden. Zuerst durchlaufen die Eingabe einen Block von ResNet-Layern. Parallel dazu werden auf die Ausgaben von res1 bis res3 jeweils eine sogenannte „side feature extraction“ angewendet. Hier wird eine 1x1-Faltung und ein Upsampling durchgeführt. Pro Extraktion wird eine einzige Feature-Map erzeugt.
Die Ausgabe aus Block res5 wird an die „side 5 classification“ weitergereicht, wo diese zunächst einen 1x1-Convolutional-Layer durchläuft und daraufhin ein bilineares Upsampling durchgeführt wird. Dadurch werden sogenannte Aktivierungs-Maps für jede Klasse (Anzahl K Klassen) erzeugt, die dieselbe Größe haben wie das Eingabebild.
Die K Aktivierungs-Maps (High-Level-Features) werden zusammen mit den drei einzelnen Feature-Maps (Low-Level-Features) in die „shared concatenation“ weitergereicht, wo die Feature-Maps repliziert (K Mal) und mit jeweils einer Aktivierungs-Map konkateniert werden. Die daraus entstehende Aktivierungs-Map wird zum Schluss an den „fused classification-layer“ übergeben, die am Ende die Kantenbilder zu jeder Klasse (K Klassen) erzeugt.

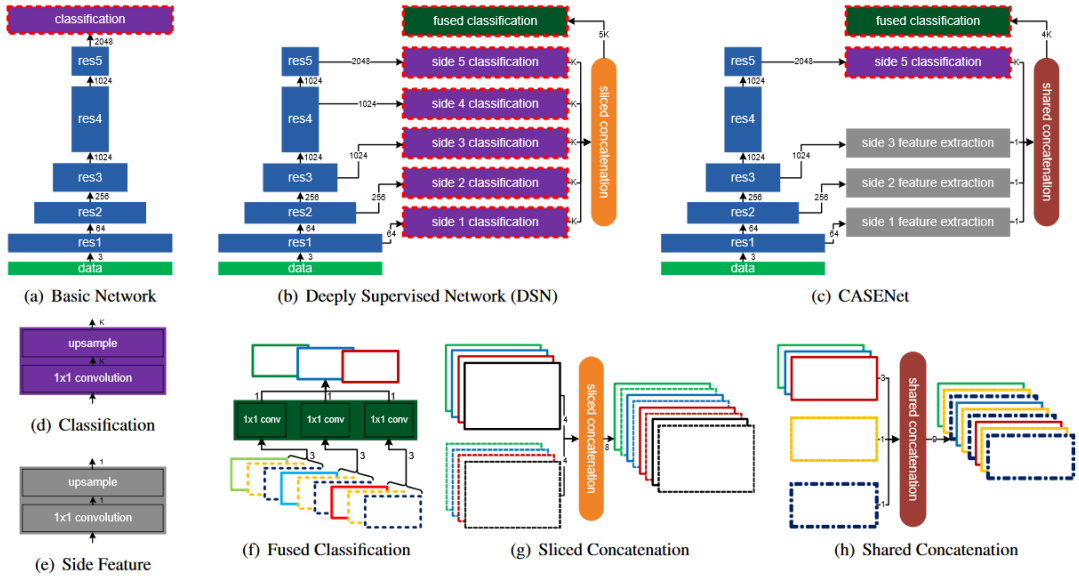


Abbildung 3-4 CASENet Netzarchitektur mit den einzelnen Bestandteilen (21). Durchgezogenes Rechteck: Aneinanderreihung von CNN-Layern. Verringerung der Breite entspricht Verkleinerung der Feature-Map um den Faktor 2. Pfeil: Anzahl Channels. Blaues, durchgezogenes Rechteck: Stapel von ResNet-Layern. Lila Rechteck: Klassifikationsmodul des CASENet-Netzwerks. Graues, durchgezogenes Rechteck: Side-Merkmalsextraktionsmodul. Roter, gepunkteter Umriss: Überwachung der Ausgabe durch die Multi-Label-Loss-Funktion. Dunkelgrünes, durchgezogenes Rechteck: Fusioniertes Klassifizierungsmodul, das eine 1x1 Convolution mit K Gruppen durchführt

In CASENet werden zu jedem Eingabebild Kantenbilder zu jeder definierten semantischen Klasse erzeugt. In einem Ergebnisnachbereitungsschritt werden diese Kantenbilder pro Klasse dann zu einem einzigen Bild zusammengefügt, wobei für jede Klasse ein entspre-

chender Farbcode verwendet wird. Dieser Prozess führt dann zu einer schönen Ergebnisdarstellung, wie sie in Abbildung 3-5 zu sehen ist.

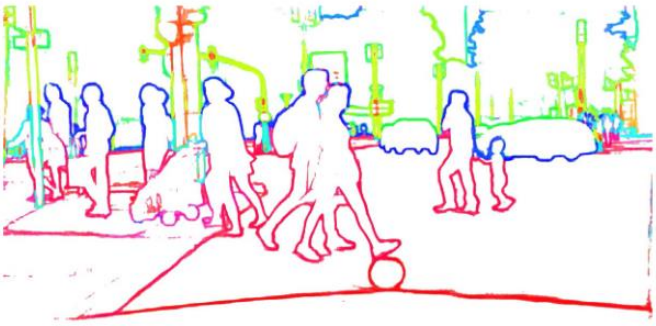


Abbildung 3-5 CASENet-Ausgabe für ein Bild nach der Ergebnisnachbereitung. Pro semantische Klasse wurde ein Farbcode verwendet

Für genauere Informationen zur Implementierung und Weiterem wird an dieser Stelle auf (21) verwiesen.

4 Arbeitsumgebung und verwendete Tools

Alle Experimente wurden auf dem JupyterHub der Hochschule für Angewandte Wissenschaften Hamburg durchgeführt (22), die Studierenden und Mitarbeitern zur Verfügung gestellt wird. Ein JupyterHub ist ein komfortabler Weg, um mehreren Leuten Jupyter Notebooks zur Verfügung zu stellen. Ein Jupyter-Notebook ist eine Webanwendung, die im Browser ausgeführt und eine Weboberfläche zur Verfügung stellt, in denen Python-Programme bequem ausgeführt werden können. Ein großer Vorteil im Hinblick zu üblichen integrierten Entwicklungsumgebungen besteht in der Datenvisualisierung zwischen reinen Codezeilen. Außerdem können einzelne zusammenhängende Codezeilen unabhängig voneinander ausgeführt werden, ohne dass das gesamte Programm immer wieder ausgeführt werden muss.

4.1 Hardware

Grafikkarte: Nvidia Tesla V100
Grafikkartenspeicher: 16GB HBM2

Prozessor: Intel® Xeon® Gold 5115 10-core
Geschwindigkeit: 2.40GHz
Cache: 13.75MB
RAM: 2666MHz DDR4 ECC Registered DIMM

Daneben persistenter Festplattenspeicher, der auf Nachfrage auf 100 GB erweitert wurde.

Alle Operationen, die das Training von NNs umfassen, werden auf der Grafikkarte ausgeführt, da sie auf einer CPU viel zu lange dauern würden. Ebenso erfolgt die Erzeugung der Ground-Truth-Kantenbilder mit der GPU.

4.2 Software

4.2.1 Programmiersprache Python

Als Grundlage dient die Programmiersprache Python in der Version 3 (23), die sich inzwischen weltweit unter den Entwicklern als Nummer 1 für ML-Projekte etabliert hat. Python ist eine moderne, universelle Programmiersprache und Open Source, hat eine sehr einfache Syntax und eine umfassende Standard-Bibliothek. Zudem unterstützt es dynamische Typisierung und mehrere Programmierparadigmen.

4.2.2 Programmbibliothek Keras

Keras (24) ist eine Open Source Bibliothek, die unter der MIT Lizenz vertrieben wird. Sie wurde in der Programmiersprache Python geschrieben. Die Verwendung der Programmbibliothek Keras in der Version 2 spielt bei dieser Arbeit eine wesentliche Rolle. Keras wurde für das Erstellen von Anwendungen für das DL entworfen. Die Besonderheit ist die angebotene High-Level-API. Diese setzt auf dem Framework Tensorflow auf und ermöglicht es Entwicklern so, mit nur wenigen Codezeilen schnell und einfach ganze NNs zu implementieren.
Keras bietet Entwicklern bereits eine Vielzahl von vordefinierten Layern, Aktivierungsfunktionen und sogar ganzen vortrainierten NNs an, die durch Wettbewerbe o.Ä. in den letzten Jahren bekannt geworden sind. Hier sei VGG16 als Beispiel zu nennen. Nachfolgende Abbildung veranschaulicht unter anderem die Verbreitung von Keras. Zu sehen sind die Software-Tools, die von den Top5-Kaggle Teams in jedem Wettbewerb verwendet wurden. Unter den Aufgaben, die das DL umfassen, ist Keras auf Platz 1.

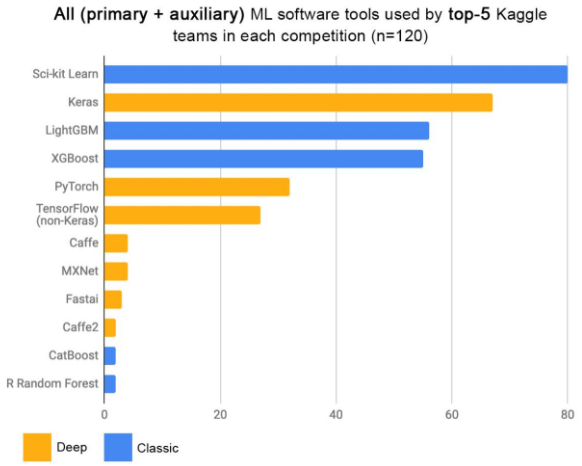


Abbildung 4-1 Genutzte Software für ML-Aufgaben der Top5-Teams in Kaggle (25)

Zum Aufbau der Architektur von NN werden zum einen die Sequential API und zum anderen die Functional API angeboten. Mithilfe der Sequential API werden einfache Netzarchitektur entworfen, indem mehrere Layer hintereinandergelegt werden. Jeder Layer ist dadurch mit dem jeweils nächsten verbunden.
Die Functional API hingegen ermöglicht den Entwurf weitaus komplexerer Netzarchitekturen. Hier können Eingangssignale anhand von Verzweigungen alternative Wege im Netz durchlaufen und nicht mehr einen bestimmten Weg wie bei der Sequential API. Ebenso sind Rückkopplungen möglich.
In dieser Arbeit wurde für die Erstellung der AE in den beschriebenen Szenarien in Kapitel 5 die Sequential API verwendet. Ebenso wurde das Training, die Validierung und auch das Testen mithilfe der von Keras angebotenen Methoden durchgeführt. Das Abspeichern von Netzen samt Gewichten und der späteren Wiederverwendung wurde ebenfalls über die Keras Bibliothek realisiert.

4.2.3 Weitere Programmbibliotheken

Alle weiteren Programmbibliotheken, mit denen insbesondere die Durchführung der Experimente in dieser Arbeit erfolgte, sind mit einer kurzen Beschreibung und des Verwendungszweckes in nachfolgender Tabelle ersichtlich.

Bibliothek	Beschreibung	Verwendung in dieser Arbeit
Numpy	Für die Unterstützung wissenschaftlicher Berechnungen entworfen. Bietet effiziente Implementierungen für numerische Berechnungen an.	Numpy-Arrays sind die Datenstruktur, in denen alle Bilder gespeichert werden Durchführung sämtlicher Operationen auf Numpy-Arrays mithilfe von Methoden der Numpy-Bibliothek
OpenCV	Bietet unter anderem Algorithmen und unterschiedliche Funktionalität für unterschiedliche Anwendungsfelder der Bildverarbeitung an	Bilder reinladen, Bilder speichern, Bildgröße verändern
Matplotlib	Unterschiedlichste Methoden zur Visualisierung von Daten aus Datenstrukturen	Ausgabe einzelner Bilder zur Kontrolle, ob sie fehlerfrei reingeladen wurden Ausgabe von mehreren Bildern nebeneinander zum einfachen subjektiven Vergleich der Ergebnisbilder
Tdqm	Bereitstellung einer Fortschrittsanzeige	Beobachten des Fortschritts bei sehr zeitaufwendigen Operationen
Keras2ascii	Ausgabe von Informationen eines NN	Ausgabe der Netzzusammenfassung

Tabelle 4-1 Verwendete Programmbibliotheken in dieser Arbeit

5 Implementierung und Bewertung

Dieses Kapitel beschreibt den Workflow, der in dieser Arbeit Anwendung findet und im nachfolgenden detaillierter erläutert wird.

5.1 Genereller Workflow

Nachfolgende Abbildung veranschaulicht den Workflow mit den wesentlichen Schritten und den dazugehörigen Teilschritten. Dieser entspricht der Vorgehensweise dieser Arbeit

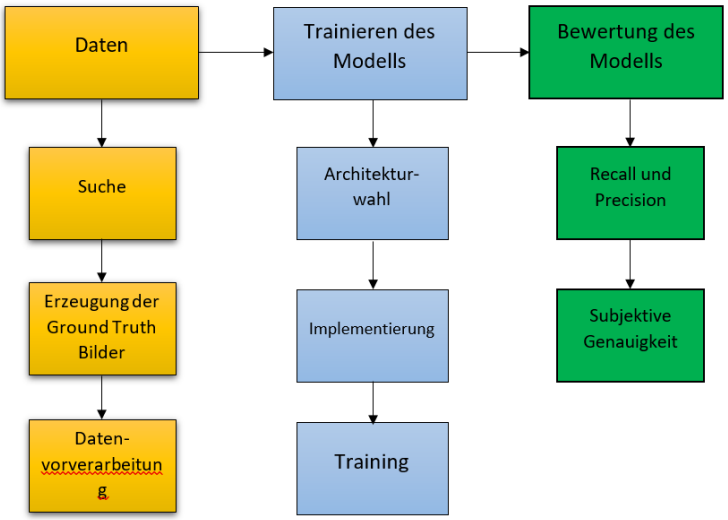


Abbildung 5-1 Workflow

Der erste Schritt umfasst alles, was die Daten im Allgemeinen betrifft. Dazu gehören unter anderem die Datensuche in unterschiedlichen Quellen, das Sammeln und Zusammentragen der Daten, die Erzeugung der Ground Truth-Kantenbilder, die Datenvorverarbeitung, die wiederum in sich aus Teilschritten besteht. Diese Teilschritte umfassen die Beseitigung fehlerhafter Daten, die Verkleinerung aller Bilder und optische Verfeinerung der Kantenbilder. Darüber hinaus werden die Kantenbilder als Schwarzweiß-Bilder reingeladen und die Bilder normalisiert. Das Ergebnis dieses ersten Schrittes sind die aufbereiteten Daten, die genau in dieser Form dem zweiten Schritt, nämlich dem Training übergeben werden.

Beim zweiten Schritt geht es zunächst darum, eine Entscheidung zu treffen, welche Netzarchitektur genommen werden soll. Dieses daraus entstandene Modell wird dann in Keras implementiert und mit den bereitgestellten Daten trainiert. Der Trainingsprozess selbst wurde im Detail bereits in Kapitel 2 erläutert.
Im letzten Schritt wird das Modell hinsichtlich seiner Leistungsfähigkeit auf ungesehenen, vollkommen fremden Daten bewertet. Liegt das Ergebnis über den Erwartungen und ist mehr als zufriedenstellend, könnte das Modell später in ernsthaften Anwendungen eingesetzt werden. Dies hängt jedoch unter anderem von viel mehr Faktoren ab. Ist das Ergebnis hingegen nicht zufriedenstellend, so müssen die Ursachen für die schlechten Ergebnisse ermittelt werden. Die im nachfolgenden Abschnitt erläuterte Vorgehensweise ist ein möglicher Weg dahin.
In der Abbildung ist der Workflow in vereinfachter Form sequenziell dargestellt. Jedoch stimmt dies nicht zwingend mit der Realität überein. Ein häufig auftretender Fall hierfür ist eine gewählte Netzarchitektur, die jedoch nicht die gewünschten Ergebnisse hervorbringt. Hier muss nach Ursachen gesucht werden. Es kann sein, dass die Netzarchitektur als solches ungeeignet ist und entsprechende Modifikationen daran vorgenommen werden müssen. In einem solchen Fall werden Wiederholungen innerhalb des zweiten Schritts durchgeführt.
Eine andere Ursache kann auf die eigentlichen Daten zurückzuführen sein. Die Probleme können in den Daten selbst liegen, weil sich später herausstellt, dass diese nicht geeignet sind oder nicht die erwünschte Qualität aufweisen. In dem Fall muss zurück zu Schritt 1 gesprungen werden.
In der konkreten Vorgehensweise in dieser Arbeit sind die Schritte zwei und drei aneinandergekoppelt. In dieser Arbeit erfolgt eine szenarienbasierte Untersuchung. Aufbauend auf dem Ergebnis eines Szenarios wird ein anderes untersucht. Die detailliertere Vorgehensweise zwischen Schritt zwei und drei ist in Abschnitt 5.4 zu finden.

5.2 Datensatz

5.2.1 Suche

Inzwischen existieren zahlreiche Datensätze mit Bildern für die unterschiedlichsten Zwecke. Um einige hiervon zu nennen: ImageNet, MSCoco, Pascal VOC, SBD, Cityscapes. Heute ist es dadurch relativ einfach, einen geeigneten Datensatz zu finden, der genügend Bilder bereitstellt. Die eigentliche Herausforderung bei der semantischen Kantendetektion jedoch besteht verstärkter darin, die zu diesen Bildern zugehörigen Kantenbilder herzubekommen.
Eine intensive Suche ergab, dass es unter anderem viele Forschungsprojekte gibt, die sich in der Vergangenheit intensiv mit dem Thema der semantischen Kantendetektion befasst haben, jedoch wurden die Kantenbilder bei keinem dieser Projekte öffentlich zur Verfügung gestellt. Daher wurde hier der Ansatz verfolgt, eines dieser Projekte auf dem eigenen Rechner aufzusetzen und damit die Kantenbilder zu erzeugen. Mithilfe der PyTorch-

Implementierung von CaseNet (26) wurden die zu dem Cityscapes-Datensatz zugehörigen Kantenbilder erzeugt, die dann für diese Arbeit als Grundlage verwendet wurden.

5.2.2 Cityscapes

In dieser Arbeit wird der Cityscapes-Datensatz (27) verwendet. Dieser wurde im Februar 2016 veröffentlicht und enthält Bilder zu städtischen Straßenszenen. Auf diesen Bildern sind unterschiedliche Objekte unterschiedlicher Klassen abgebildet. Beim Cityscapes-Datensatz werden die unterschiedlichen Klassen in Gruppen unterteilt, die der nachfolgenden Tabelle entnommen werden können. Die Tabelle beinhaltet alle Klassendefinitionen, die für die in dieser Arbeit verwendeten Bilder von Relevanz sind. Die Gruppen und Klassendefinitionen sind im Originalen englische Begriffe und werden mit zu der Übersetzung dazu genommen.

Gruppe	Zugehörige Klassen
Flach (flat)	Straße (road), Bürgersteig (sidewalk)
Mensch (human)	Person (person), Fahrer (rider)
Fahrzeug (vehicle)	Auto (car), Lastkraftwagen (truck), Bus (bus), Zug (vehicle on rails -> train), Motorrad (motorcycle), Fahrrad (bicycle)
Konstruktion (construction)	Gebäude (building), Mauer (wall), Zaun (fence)
Objekt (object)	Stange (pole), Ampel (traffic light), Verkehrsschild (traffic sign)
Natur (nature)	Vegetation (vegatation), Gelände (Gebiet)
Himmel (sky)	Himmel (sky)

Tabelle 5-1 Cityscapes-Datensatz: Klassendefinitionen unterteilt in Gruppen

Auf der Cityscapes-Webseite werden unterschiedliche Bildersammlungen, die als Packages angeboten werden, zum Download zur Verfügung gestellt. Registrierten Benutzern steht der Download kostenlos zur Verfügung. Für diese Arbeit werden die Originalbilder verwendet, die sich in der Datei *leftImg8bit_trainvaltest.zip* befinden. Diese Archivdatei umfasst insgesamt 5000 Bilder, in denen städtische Straßenszenen in 27 deutschen Städten aufgenommen wurden, zu unterschiedlichen Tageszeiten und unter unterschiedlichen Konditionen. Alle Bilder haben eine Auflösung von 1024x2048 Pixeln. Die Zip-Datei enthält darüber hinaus noch Ground-Truth-Dateien für die Semantische Bildsegmentierung sowie Semantische Instanzsegmentierung, die für diese Arbeit jedoch uninteressant sind und daher nicht verwendet werden. Die 5000 Bilder sind bereits unterteilt in 2975 Trainingsbilder, 500 Validierungsbilder und 1525 Testbilder.

Abbildung 5-2 Auszug aus dem Cityscapes-Datensatz: Aufnahmen unterschiedlicher städtischer Straßenszenen

Mithilfe von CASENet wurden die zugehörigen Kantenbilder erzeugt. CASENet erzeugt zu jedem Bild jeweils 19 Kantenbilder, die den nachfolgenden Klassen aus der obigen Tabelle entsprechen. Abbildung 5-3 visualisiert diesen Prozess.
Diese Arbeit konzentriert sich auf die semantische Kantendetektion von Autos. Die Entscheidung fiel insbesondere auf diese Klasse, da sie die folgenden drei Eigenschaften erfüllt:

- Sie sind in genügend Bildern vertreten (ca. 95% der Bilder)
- Sie sind groß genug. Dies ist insbesondere ein entscheidender Aspekt beim Herunterskalieren der Bildgröße, der ein Teilschritt der Datenvorverarbeitung ist
- Es gibt Bilder mit Gegenbeispielen, also Bilder ohne Autos (ca. 5% der Bilder)

Alle anderen Klassen schieden aufgrund des Nichterfüllens von mindestens einem dieser Punkte aus.

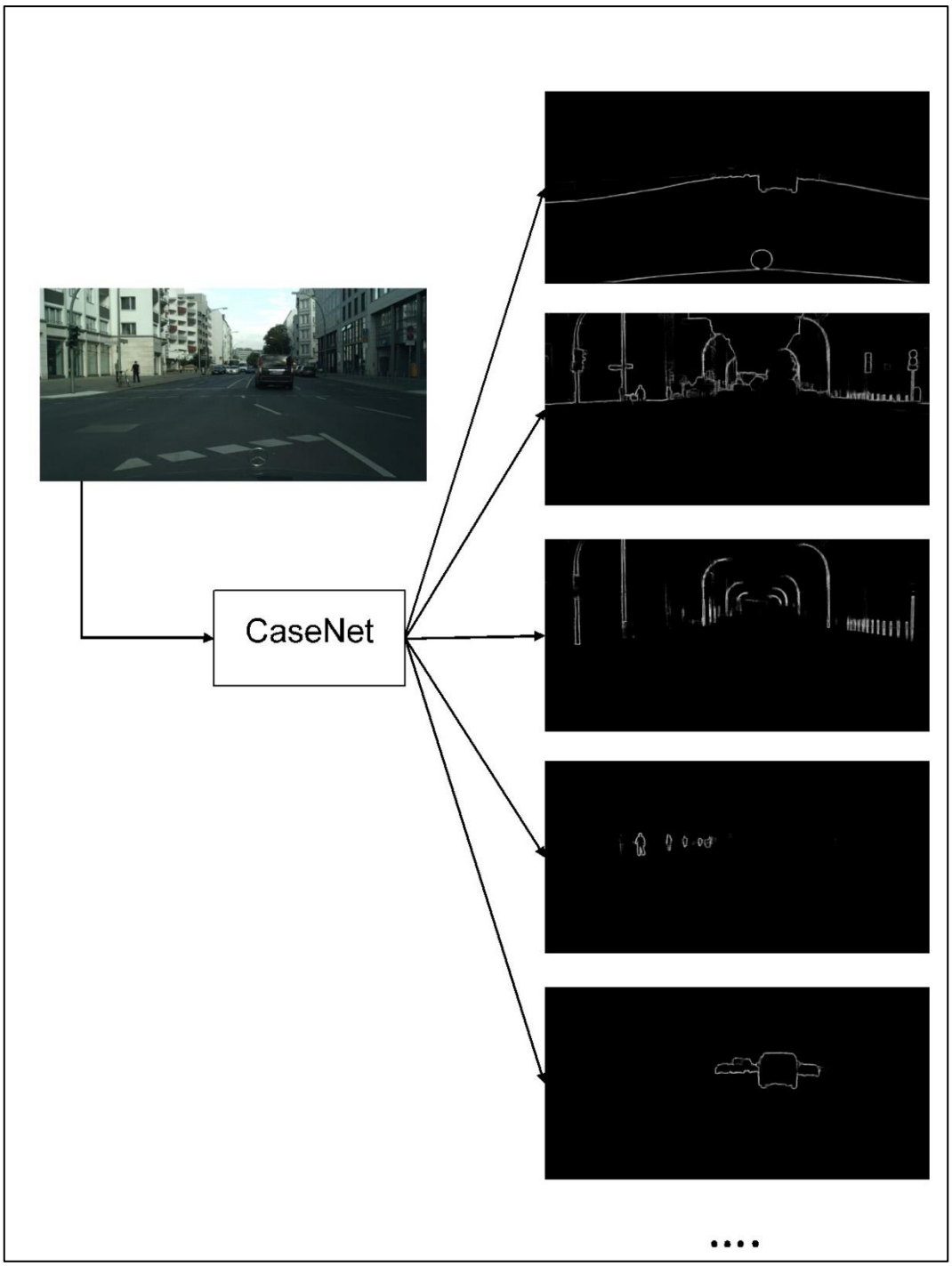


Abbildung 5-3 CaseNet erzeugt Kantenbilder unterschiedlicher Klassen zu einem Eingabebild

5.3 Datenvorverarbeitung

In diesem Abschnitt werden alle Schritte der Datenvorverarbeitung beschrieben. Im ersten Schritt werden fehlerhafte Kantenbilder mit ihrem zusammengehörigen Eingangsbild herausgenommen. Daraufhin werden die Bilder verkleinert. Der nächste Schritt umfasst die optische Verfeinerung der Kantenbilder. Zusätzlich werden die Kantenbilder dann im Jupyter Notebook mit nur einem Channel reingeladen, um Speicherplatz zu sparen. Bevor die Daten dem AE übergeben werden, werden in einem letzten Schritt die Farb- und Grauwerte der Bilder herunterskaliert.

5.3.1 Beseitigung fehlerhafter Bilder

Die mithilfe von CASENet erzeugten Kantenbilder zu Autos wurden in diesem Schritt einer genauen manuellen Kontrolle auf Richtigkeit unterzogen. Fehlerhafte Kantenbilder, in denen Autos nicht detektiert wurden, wo aber welche sind sowie Bilder, wo eine größere Anzahl von Objekten fälschlicherweise als Autos detektiert wurden, wurden herausgenommen. Dies war jedoch nur bei einer geringen Anzahl von Bildern der Fall. Ansonsten wurden einige wenige Bilder manuell nachbearbeitet und verbessert.

5.3.2 Verkleinerung aller Bilder

Die Originalbilder und die Kantenbilder sind in der Größe 1024x2048 Pixel relativ groß. Hierbei benötigen manche Bilder mehr als 1 Megabyte Speicherplatz. Einerseits um den zur Verfügung stehenden Speicher auf der Grafikkarte optimal auszunutzen, werden diese um den Faktor 16 auf die Größe 128x256 Pixel herunterskaliert. Dies führt unter anderem dazu, dass Feinheiten, insbesondere bei den Kantenbildern verloren gehen, jedoch wird das in Kauf genommen. Hierdurch wird das problemlose Reinladen von knapp 5000 Bildern in ein Jupyter-Notebook ermöglicht. Andererseits vermeidet dieser Schritt zudem unnötigen, zusätzlichen Aufwand beim Experimentieren.

5.3.3 Verfeinerung der Kantenbilder

In diesem Schritt werden nur die Kantenbilder bearbeitet. Hier werden die abgeschwächten Kanten in den Bildern deutlicher dargestellt und ein klares Unterscheidungsmerkmal zwischen Kante und Nicht-Kante festgelegt. Dazu werden alle Grauwerte ab einem bestimmten Wert auf 255 und alle darunterliegenden Grauwerte auf 0 gesetzt. Versuche haben gezeigt, dass der Grauwert 40 sehr gut als Schwellenwert geeignet ist. Bei niedrigeren Grauwerten zeigte sich, dass weiße Punkte sichtbar wurden, die aber für die eigentlichen Autoobjekte im Bild nicht relevant sind. Ein höherer Schwellenwert hat dafür gesorgt, dass zum Teil wesentliche Kanten verschwinden und somit wichtige Informationen verloren gehen.

Genauso wie optisch eindeutig zwischen Kante und Nicht-Kante unterschieden wird, spiegelt sich das nach diesem Schritt ebenfalls in den Grauwerten der Kantenbilder wider. Hier nach gibt es in allen Kantenbildern nur noch die Grauwerte 0 und 255. Der Grauwert 0 steht dabei für Nicht-Kante und ist auf dem Kantenbild ein schwarzer Punkt. Auf der anderen Seite steht der Grauwert 255 dafür, dass es sich hierbei um eine Kante handelt und sich auf dem Kantenbild an dieser Stelle ein weißer Punkt befindet. Nachfolgende Abbildung visualisiert diesen Prozess. Hier wird das originale Kantenbild dem verfeinerten gegenübergestellt.

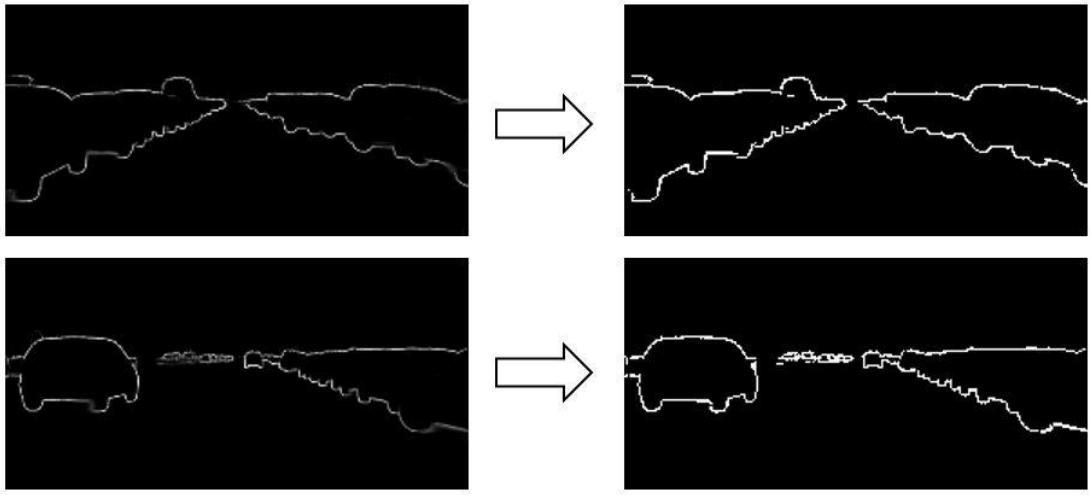


Abbildung 5-4 Verfeinerung der Kantenbilder: Kantenbild vor der Verfeinerung (l.), Kantenbild nach der Verfeinerung (r.)

Dieser Schritt hat keinen technisch notwendigen Hintergrund und wird aus rein optischen Gründen ausgeführt, um die Kantenbilder besser aussehen zu lassen. Dadurch stechen die weißen Kanten in den Bildern besser hervor.
Ebenso wird dieser Schritt auf die gleiche Weise bei der Ergebnisnachbereitung angewendet, um die vom AE gelieferten Ergebnisbilder zu verfeinern. Dies ist auch notwendig, um eine faire Vergleichsbasis mit den GT-Kantenbildern zu gewährleisten.

5.3.4 Reinladen der Kantenbilder als Graustufenbilder

Diesen Schritt eingeschlossen werden alle nachfolgenden Schritte der Datenverarbeitung innerhalb eines Jupyter-Notebooks durchgeführt. Die Kantenbilder werden als Graustufenbilder mit einem Channel reingeladen. Der Vorteil darin besteht, Speicherplatz zu sparen und somit eine bessere Ausnutzung des Grafikkartenspeichers zu ermöglichen. Die Eingabebilder werden unverändert mit drei Channels in ein Notebook geladen.

5.3.5 Normalisierung der Grauwerte

Die Bilder werden zunächst in Numpy-Arrays reingeladen. Die die jeweiligen Rot-, Grün- und Blauwerte liegen bei den Farbbildern im Bereich 0 bis 255. Dies gilt ebenso für die Grauwerte der Kantenbilder. In diesem Schritt werden diese Werte jeweils bei allen Bildern auf den Wertebereich von 0 bis 1 herunter skaliert. Danach werden die Daten in dieser Form dem Trainingsprozess übergeben. Die Notwendigkeit dieses Schritts wurde bereits im Abschnitt Überanpassung erläutert.

5.4 Szenarienbasierte Untersuchungen

Diese Arbeit beschäftigt sich mit der semantischen Kantendetektion auf Autoobjekten mithilfe eines AEs. Das primäre Ziel dabei ist, dass der AE die grundlegenden Strukturen der Autos innerhalb der Bilddaten erkennen soll. Für die Erreichung dieses Ziels werden Untersuchungen durchgeführt, die jeweils auf unterschiedlichen Szenarien basieren. Dabei wird systematisch vorgegangen und jeweils nach einem Szenario gezeigt, ob der AE nützliche Ergebnisse liefern kann. Im Mittelpunkt steht hierbei der Einfluss unterschiedlicher Hyperparameter des Netzes auf das Ergebnis.

Zunächst wird im ersten Szenario ein einfacher Autoencoder entworfen, trainiert und bewertet. Die Bewertung erfolgt dabei auf einem Testdatensatz. Daraufhin folgt die Entscheidung, ob die Ergebnisse zufriedenstellend sind. Dieser AE dient als Referenzmodell in Bezug auf die weiteren Szenarien. Anschließend werden in den weiteren Szenarien jeweils ein Hyperparameter des Netzes geändert, diese Netzkonfiguration getestet und bewertet. Diese Ergebnisse werden mit den Ergebnissen des aktuellen Referenzmodells oder eines passenderen Vergleichsmodells verglichen. Auf Grundlage der Metriken wird entschieden, ob sich wesentliche Verbesserungen in den Ergebnissen ergeben haben. Wurden in den Ergebnissen Verbesserungen festgestellt, werden diese gewonnenen Erkenntnisse für spätere Szenarien gemerkt. Durch die Veränderung eines einzelnen Hyperparameters wird eine sehr einfache Vergleichsbasis zum gewählten Referenzmodell oder Vergleichsmodell hergestellt. Die Veränderungen sind dadurch nachvollziehbarer. In einem späteren Szenario wird ein größerer AE verwendet und die gewonnenen Erkenntnisse im Aufbau des AE mitberücksichtigt. Dieses dient dann als Referenzmodell für die weiteren Szenarien, in denen wieder jeweils ein Hyperparameter verändert wird. Durch diese Vorgehensweise wird versucht, den AE schrittweise zu verfeinern, um dadurch bessere Ergebnisse erhalten zu können. Nachfolgende Abbildung veranschaulicht die wesentliche Vorgehensweise. Einige speziellere Details wurden dabei weggelassen, da es die Grafik sonst unnötig verkomplizieren würde. Dazu gehört, dass die Ergebnisse eines neuen Referenzmodells gegen die bisherigen Ergebnisse verglichen wird. Das tritt dann ein, wenn ein größeres Netz verwendet wird.

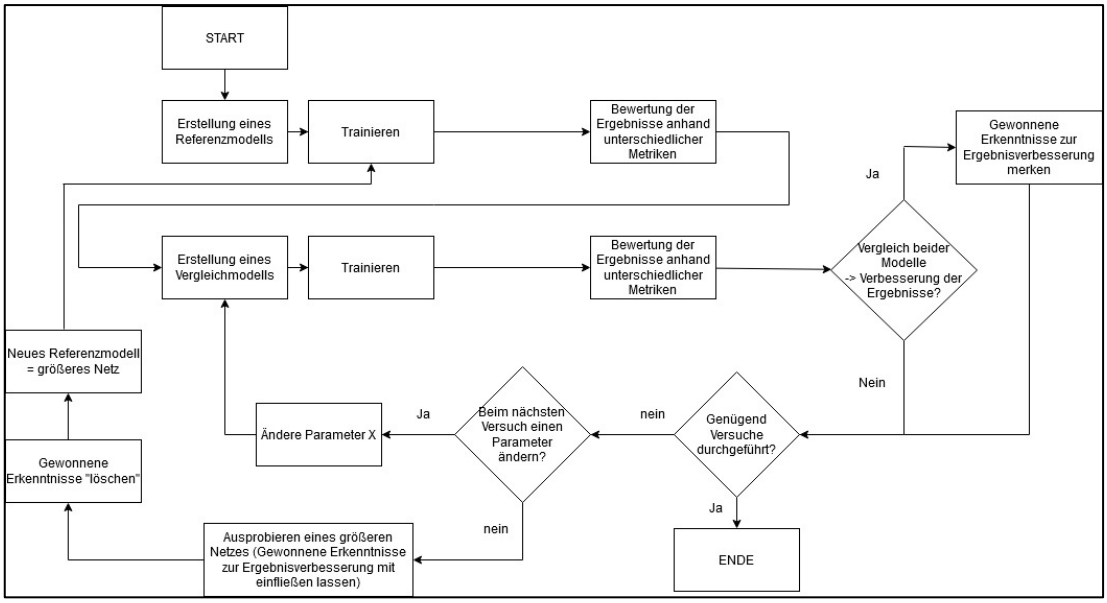


Abbildung 5-5 Detaillierterer Workflow – Abläufe in den Schritten 2 und 3 des generellen Workflow

Folgende Szenarien werden im weiteren Verlauf dieses Kapitels im Detail beschrieben:

- Untersuchtes Szenario 1: Referenzmodell
- Untersuchtes Szenario 2: Vergrößerung latent space
- Untersuchtes Szenario 3: Vergrößerung der Faltungsmaske auf 5x5
- Untersuchtes Szenario 4: Vergrößerung der Faltungsmaske auf 7x7
- Untersuchtes Szenario 5: Verwendung eines größeren Netzes
- Untersuchtes Szenario 6: Verdopplung der Filteranzahl

5.4.1 Struktur eines vorgestellten Szenarios

Ein Szenario wird eingeleitet durch eine kurze Beschreibung gefolgt vom Aufbau des dabei verwendeten Autoencoders mit zusätzlichen Informationen. Die Ergebnisdarstellung erfolgt in zwei Teilen. Zuerst werden die berechneten Recall-, Precision- und F1-Score-Werte in einem Histogramm dargestellt. Danach wird durch eine geeignete Auswahl von Bildern aus drei unterschiedlichen Kategorien die Kantenergebnisbilder des Netzes den GT-Kantenbildern gegenübergestellt. Anschließend erfolgt in einer kurzen Diskussion, ob der in dem jeweiligen Szenario vorgestellte Autoencoder bessere Ergebnisse geliefert hat als das Szenario, mit dem verglichen wird. Zudem wird entschieden, ob der AE in diesem Szenario die Aufgabe der semantischen Kantendetektion auf Autoobjekten zufriedenstellend gelöst hat. Die Basis, auf dem diese Entscheidung getroffen wird sind die Erfolgskriterien.

5.4.2 Architekturentscheidungen

Es gibt viele vortrainierte und verwendbare Netze, die in den letzten Jahren zunehmend an Popularität gewannen. VGG16 oder VGG19 seien hier als Beispiele zu nennen. Diese Deep Learning Modelle werden durch die Keras Applications (28) zur Verfügung gestellt und können auf eine einfache Weise verwendet werden. Diese Modelle sind insbesondere nützlich bei der Merkmalsextraktion oder bei Vorhersagen. Ein weiterer Vorteil besteht zudem darin, dass die Gewichtseinstellungen dieser vortrainierten Modelle mit übernommen werden können. So entfällt der Aufwand für das Training und die Modelle können gleich als „fertiges“ Netz für eigene Experimente verwendet werden.
Jedoch wird hier die Annahme getroffen, dass solche riesigen und komplexen Netze für die Aufgabe der semantischen Kantendetektion auf Autoobjekten nicht notwendig sein werden.
Die hier vorliegenden Ein- und Ausgabedaten sind jeweils Bilder. Wie in Abschnitt 2.3.5 bereits erläutert, eignet sich ein AE sehr gut für die vorliegende Aufgabe, da es die Aspekte Informationsreduktion und Merkmalsextraktion aus Bildern vereint und ist somit die für die untersuchten Szenarien gewählte Netzarchitektur.

5.4.3 Strategie

Generelle Einstellungen
Alle AE wurden mithilfe der Sequential API von Keras gebaut. Die Eingabe sind Bilder der Größe 128x256 mit drei Channels, die zunächst in ein Numpy-Array eingelesen und daraufhin auf den Wertebereich [0,1] normalisiert wurden. Die Ausgabe ist ein Kantenbild der Größe 128x256 mit einem Channel. Alle Layer verwenden ReLu als Aktivierungsfunktion. Der Output-Layer hingegen nutzt eine Sigmoid-Aktivierungsfunktion. Als Optimizer wird Adam verwendet. Die binary crossentropy dient als Verlustfunktion. Beim Training werden die vom Cityscapes-Datensatz zur Verfügung gestellten Trainings- und Validierungsbilder verwendet. Für die Bewertung werden die Testbilder aus dem Cityscapes-Datensatz herangezogen.

Zielsetzung
Es soll untersucht werden, ob der AE in der Lage ist, möglichst alle in Bildern auftauchende Autos zu detektieren. Primär geht es dabei um die grundlegende Erkennung von Autoobjekten. Es wird akzeptiert, wenn an einigen Stellen Autos detektiert werden, wo keine sind. Der Hauptfokus liegt darin, dass möglichst keine Autos übersehen werden sollen. Hierbei ist zum einen der Recall-Wert ausschlaggebend. Aber nur diesen Wert allein zu betrachten ist jedoch nicht ausreichend und kann keine zuverlässige Bewertungsgrundlage sicherstellen. Der Grund: Wären die Netzausgaben theoretisch vereinfacht rein weiße Bilder, würden immer alle Autos detektiert werden. Der Recall wäre für jedes Bild immer 100%. In diesem Fall wäre der Precision-Wert sehr nahe bei 0, da damit sehr viele nicht relevante Pixel als

Autokanten detektiert würden. Das Gesamtergebnis wäre in diesem Fall keinesfalls zufriedenstellend.

Die Kriterien, um Erfolg zu definieren sind wie folgt:

- Der AE erzielt mindestens bei der Hälfte der Testbilder einen Recall-Wert von 0,5 bzw. 50%.
- Zusätzlich soll der F1-Score bei mindestens der Hälfte der Testbilder 35% betragen. Damit wird die Precision automatisch mit einbezogen.

Mit den ausgewählten Bildern bei der subjektiven Genauigkeit sollen zum einen Beispiele für die erzielten Werte beim Recall, bei der Precision und bei dem F1-Score gezeigt werden. Zum anderen wird überprüft, ob der Betrachter dabei das „Gefühl kriegt", dass der AE grundlegend in der Lage ist, Autos detektieren zu können. Dabei muss beachtet werden, dass durch ein paar gewählte Einzelbilder keine allgemeingültigen Aussagen für alle Bilder innerhalb des Testdatensatzes getroffen werden können. Daher wird diese Überprüfung mehr nebensächlich durchgeführt und ist von marginaler Bedeutung bei der Entscheidungsgrundlage.

Informationen zum Trainingsprozess

Es muss ermittelt werden, nach welcher Epochenzahl der AE die besten Ergebnisse liefert. Anders gesagt: die Stelle ermitteln, bevor eine Überanpassung eintritt. Dieser AE in diesem Zustand samt aller Gewichtswerte wird dann als Grundlage für die Bewertung herangezogen. Die Bewertung erfolgt dann an den Ergebnissen des AE auf dem bereinigten Testdatensatz.

Das übliche Vorgehen hierbei ist, den Verlauf des Verlustwerts für Trainings- und Validierungsdaten zu betrachten und die Stelle zu ermitteln, ab da der Verlustwert für die Trainingsdaten weiterhin sinkt und der Verlustwert für die Validierungsdaten jedoch wieder anfängt, zu steigen. Beim Training und bei der Validierung wurde bei jedem der Szenarien beobachtet, wie rapide der Verlustwert für Training und Validierung sank. In einem kleinen Wertebereich (Nachkommastellen) sprang der Verlustwert nach Erreichen des Minimums zwar hin und her, jedoch ist das kein Indiz für eine Überanpassung.

Zudem kann der Verlustwert bei dem hier vorliegenden Fall ohnehin nicht sehr hoch sein. Dies hat folgende Gründe:

- Bei allen Bildern gibt es einen sehr hohen Anteil an schwarzen Bildpunkten. Abbildung 5-6 zeigt in vereinfachter Form, wie viele weiße Pixel in wie vielen Testbildern enthalten sind. Der AE wird erlernen, dass die meisten Bildpunkte schwarz sein müssen. Eine Analyse ergab, dass pro Testbild durchschnittlich 421 weiße Pixel enthalten sind. Wird das in Verhältnis zu der Gesamtanzahl der Pixel eines Bildes mit der Größe 128x256 gesetzt (insgesamt 32768), entspricht das 1,28% der Pixel im ganzen Bild. Selbst wenn der AE bei den Testdatenbildern nur komplett schwarze

Bilder ausgibt, würden die vorhergesagten Pixel in der Gesamtanzahl dennoch zu fast 99% übereinstimmen. Die Ergebnisse wären jedoch vollständig unbrauchbar.

- Die verwendeten AE in den Szenarien sind mit einer fünfstelligen Parameteranzahl ohnehin nicht sehr groß. Daher kann hier der Verlustwert nicht besonders groß werden. Zwar ändert sich das bei der Verwendung größerer AE (sechsstellige Parameteranzahl), aber dennoch wird auf die genauere Beobachtung des Verlustwerts verzichtet, da das Vorgehen sonst nicht einheitlich wäre.

Aufgrund der oben beschriebenen Gründe muss hier anders vorgegangen werden. Es wird die Stelle ermittelt, ab der sich in den Ergebnissen keine wesentlichen Verbesserungen mehr zeigen, und zwar von der Wahrnehmung des Betrachters her. Hierfür wurden die Ergebnisse eines Netzes immer nach jeweils 10 Epochen betrachtet und mit den GT-Kantenbildern des Validierungsdatensatzes abgeglichen. Insgesamt aufsteigend bis hin zu 200 Epochen. Dabei wurde der jeweilige AE samt Gewichten gespeichert, um eine spätere Rekonstruktion problemlos zu ermöglichen. Nach einer umfassenden Kontrolle wurde entschieden, nach welcher Epochenanzahl der AE die besten Ergebnisse liefert und dieses mithilfe des Testdatensatzes bewertet.
Um eine übermäßige Darstellung von Bildern zu meiden, wurde darauf verzichtet, dieses Vorgehen auf den nachfolgenden Seiten zu den Szenarien darzustellen. Stattdessen wurde direkt die Bewertung auf Basis des bereinigten Testdatensatzes mit 1422 Bildern durchgeführt.

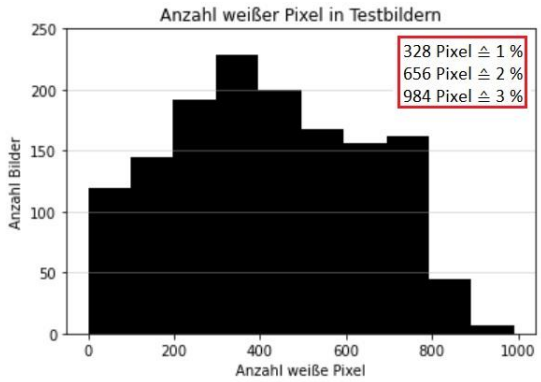


Abbildung 5-6 Vereinfachte Darstellung, welche Anzahl von Testbildern wie viele weiße Pixel beinhalten - Prozentangaben beziehen sich auf die Gesamtanzahl der Pixel pro Kantenbild

5.4.4 Untersuchtes Szenario 1: Referenzmodell

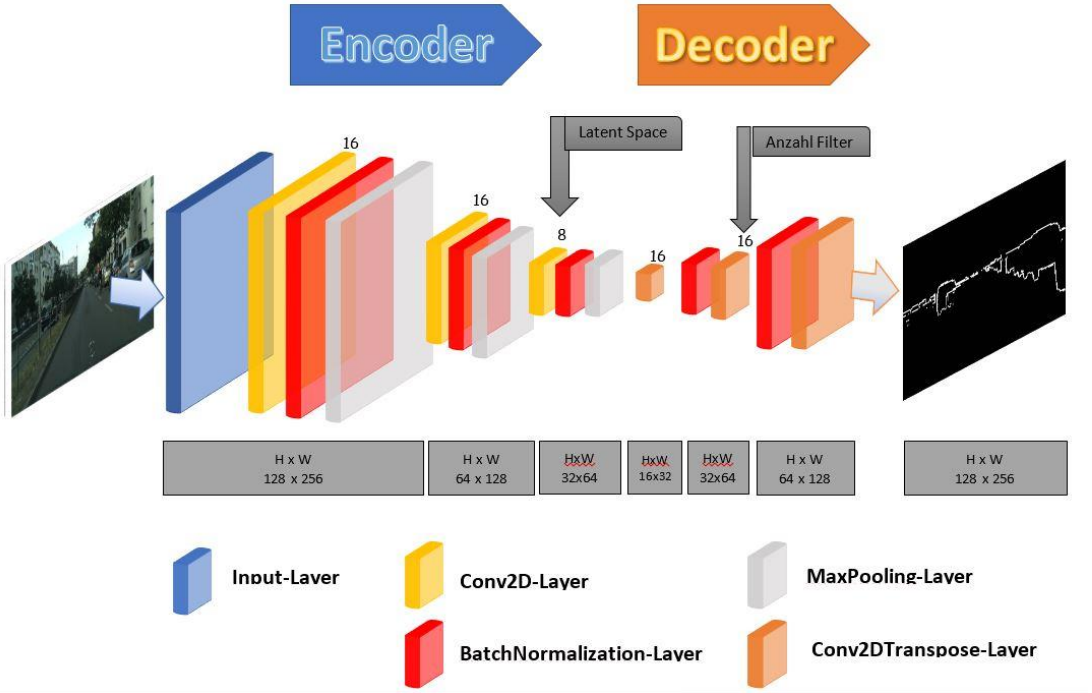


Abbildung 5-7 Netzarchitektur des Referenzmodells

Abbildung 5-7 zeigt die Netzarchitektur des hier gewählten Referenzmodells. Am Anfang befindet sich der Input-Layer, der die Bildeingabe entgegennimmt und diese an den Encoder-Teil im NN weitergibt. Auf der Encoder-Seite befinden sich drei Conv2D-Layer, auf die jeweils immer ein BatchNormalization- und ein MaxPooling-Layer folgen. Die Decoder-Seite besteht aus einer Folge von Conv2D-Transpose- und BatchNormalization-Layern, die das verarbeitete Bild wieder auf die Originalgröße von 128x256 Pixel bringen und dieses dann die Ausgabe vom AE bildet. Ein zusätzlicher Unpooling-Layer auf der Decoder-Seite, der die Bildgröße wieder verdoppelt und somit das Gegenstück zum MaxPooling-Layer bildet, ist hierbei nicht mehr notwendig, da es in Keras in der Version 2 möglich ist, ein stride-Argument an den Conv2D-Transpose-Layer mitzugeben, der genau diesen Vergrößerungseffekt erzielt.
Nachfolgend ist die keras2Ascii-Ausgabe (29) dieses Referenzmodells. Sie enthält eine Auflistung über die einzelnen Layer, den Output-Dimensionen sowie die Anzahl der Gewichte und dem prozentualen Anteil der Gewichte pro Layer im Verhältnis zur Gesamtanzahl an Gewichten.

OPERATION		DATA DIMENSIONS			WEIGHTS(N)	WEIGHTS(%)
Input	#####	128	256	3		
InputLayer	|	----------			0	0.0%
	#####	128	256	3		
Conv2D	|/	----------			448	5.7%
relu	#####	128	256	16		
BatchNormalization	μ|σ	----------			64	0.8%
	#####	128	256	16		
MaxPooling2D	Y max	----------			0	0.0%
	#####	64	128	16		
Conv2D	|/	----------			2320	29.6%
relu	#####	64	128	16		
BatchNormalization	μ|σ	----------			64	0.8%
	#####	64	128	16		
MaxPooling2D	Y max	----------			0	0.0%
	#####	32	64	16		
Conv2D	|/	----------			1160	14.8%
relu	#####	32	64	8		
BatchNormalization	μ|σ	----------			32	0.4%
	#####	32	64	8		
MaxPooling2D	Y max	----------			0	0.0%
	#####	16	32	8		
Conv2DTranspose	/| \	----------			1168	14.9%
relu	#####	32	64	16		
BatchNormalization	μ|σ	----------			64	0.8%
	#####	32	64	16		
Conv2DTranspose	/| \	----------			2320	29.6%
relu	#####	64	128	16		
BatchNormalization	μ|σ	----------			64	0.8%
	#####	64	128	16		
Conv2DTranspose	/| \	----------			145	1.8%
sigmoid	#####	128	256	1		

Abbildung 5-8 Szenario 1: Netzzusammenfassung

Parameter-Einstellungen:
Filteranzahl = 16
Faltungskern = 3x3
Verlustfunktion = binary crossentropy
Optimierer = Adam
Batch Size =16
Epochen = 90

Weitere Informationen:
Gesamtanzahl Parameter = 7849

Ergebnisse

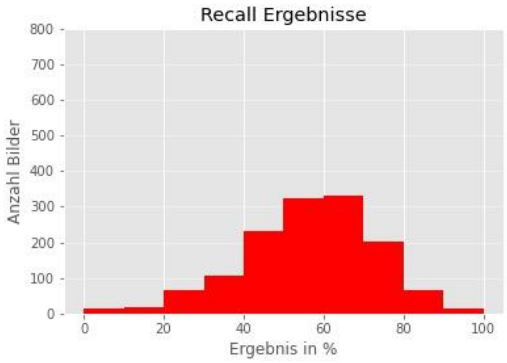


Abbildung 5-9 Szenario 1: Recall Ergebnisse

Maximum: 97.3% Minimum: 0.0% Durchschnitt: 56.6%
Ausnahmen: 47 Bilder

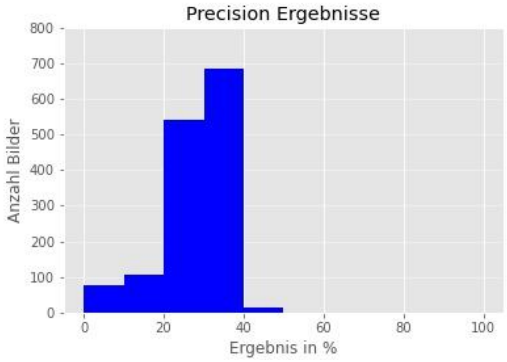


Abbildung 5-10 Szenario 1: Precision Ergebnisse

Maximum: 48.8% Minimum: 0.0% Durchschnitt: 27.7%
Ausnahmen: 3 Bilder

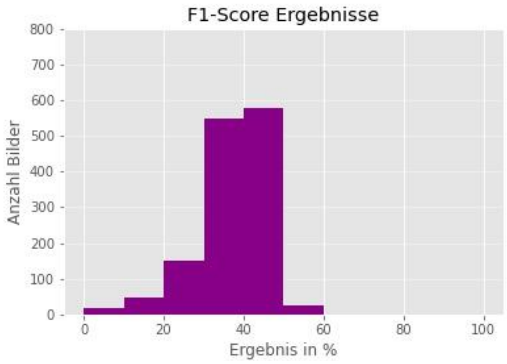


Abbildung 5-11 Szenario 1: F1-Score Ergebnisse

Maximum: 57.7% Minimum: 1.0% Durchschnitt: 37.3%
Ausnahmen: 51 Bilder

Kategorie 1: Bilder mit vielen Autos

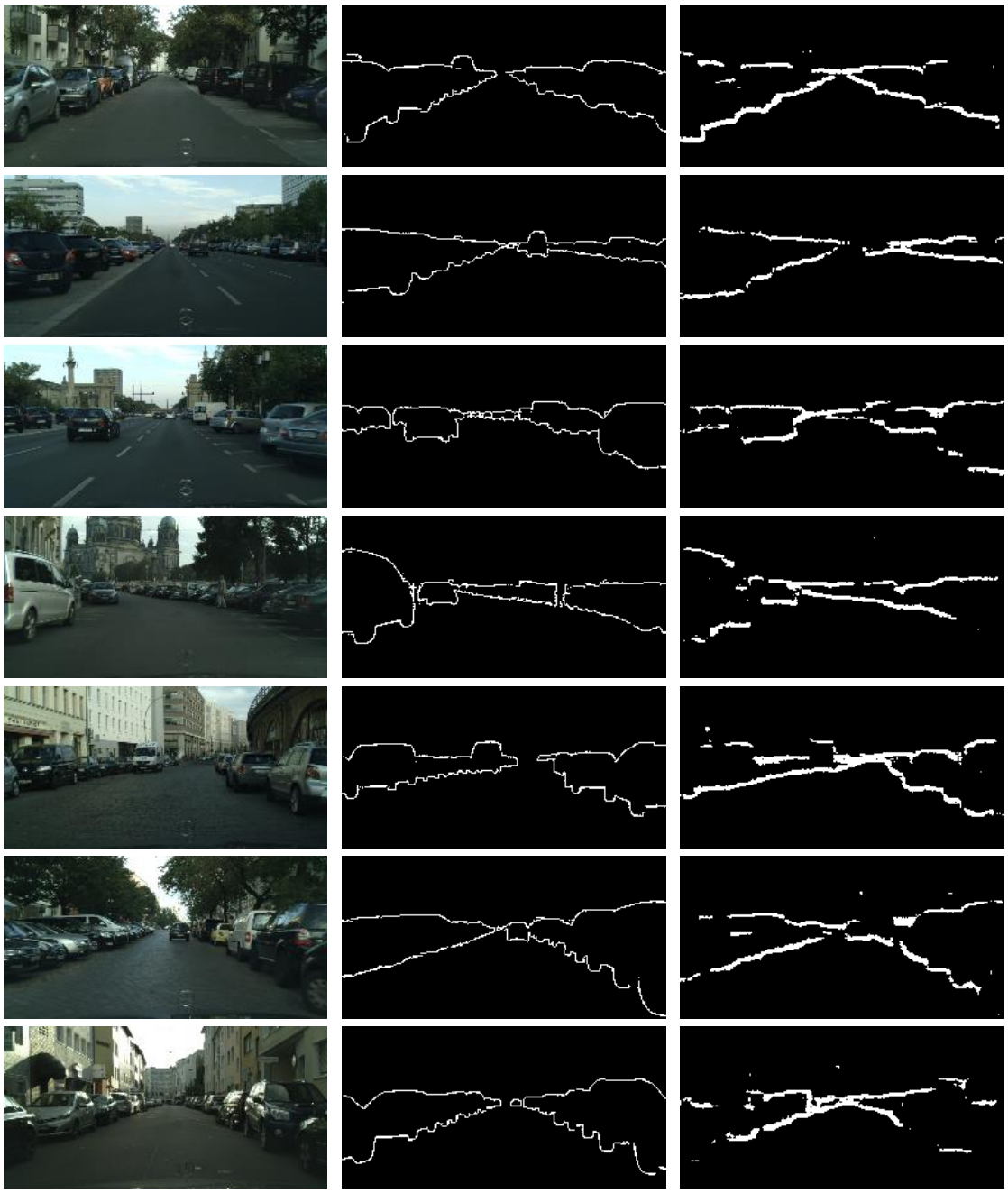


Abbildung 5-12 Szenario 1, Kategorie 1: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Kategorie 2: Bilder mit wenigen oder einzelnen Autos

Abbildung 5-13 Szenario 1, Kategorie 2: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Kategorie 3: Bilder ohne Autos

Abbildung 5-14 Szenario 1, Kategorie 3: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Bewertung

Recall

Aus den Ergebnissen wird ersichtlich, dass bei mehr als der Hälfte der Bilder ein Recall-Wert von mindestens 50% oder höher erzielt wurde. Insbesondere liegt der Recall-Wert für mehr als 300 Bilder jeweils einmal im Bereich von 50% bis 60% sowie 60% bis 70%. Mit einem erzielten Durchschnitt von 54,6% ist anzunehmen, dass bei mindestens der Hälfte der Bilder alle Autos erfolgreich detektiert wurden.

Precision

Alle Ergebnisse liegen unter dem Bereich von 50%. Knapp 700 Bilder und damit knapp die Hälfte der Testbilder haben eine Precision im Bereich 30% bis 40%. Etwa 550 Bilder liegen im Bereich von 20% bis 30%. Diese zwei Bereiche schließen insgesamt über 80% der Testbilder ein. Daraus lässt sich schließen, dass die in den Bildern detektierten Pixel zu einem höheren Anteil nicht für die eigentlichen Kanten relevant sind. Bei einem kleineren Anteil von knapp 200 Bildern liegen die Werte im Bereich von 0% bis 20%. Bei diesen Bildern ist ein sehr großer Anteil der detektierten Kanten für das Ergebnis nicht von Relevanz.

F1-Score

Die F1-Score-Werte liegen für über knapp 1100 Bilder im Bereich von 30% bis 50%. Durch eine einfache Beobachtung lässt sich sagen, dass die Säulen näher an den Säulen in dem Histogramm für die Precision-Werte liegen. Zudem haben sie eine ähnliche Höhe. Die Precision-Werte liegen alle im linken Teilbereich von 0% bis 50%, während die Recall-Werte über dem gesamten Bereich vertreten sind, an den zwei Enden jeweils weniger. Insgesamt ziehen die Precision-Werte das Gesamtergebnis etwas herunter. Dennoch wird bei den F1-Score-Werten ein Durchschnitt von 37,3% erzielt.

Subjektive Genauigkeit

Bei den Bildern mit vielen Autos erkennt der AE die grundlegenden Strukturen, wobei es einen Mangel bei den kleineren Feinheiten gibt. Bei der Kategorie 2 hingegen erkennt es die auf dem Bild befindlichen Autos in grundlegender Hinsicht, jedoch detektiert der AE ebenfalls Autos an vielen Stellen, wo jedoch keine sind. Kategorie 3 zeigt die Schwächen des AEs sehr deutlich auf. Obwohl es auf keinem einzigen Bild ein Auto gibt, detektiert der AE Autos bei einigen Bildern an vielen Stellen und bei einigen Bildern an weniger Stellen. Übergreifend ist zu erkennen, dass die Autokanten in sehr vielen Fällen nicht gut geschlossen werden.

Zusammengefasst wird ersichtlich, dass der AE grundlegend in der Lage ist, Autos zu erkennen, insbesondere, wenn es viele zusammengehörige sind. Hingegen weist es Schwächen darin auf, indem es Autos an vielen Stellen detektiert, wo jedoch keine sind.

Dieses Szenario dient zunächst als Start-Referenz für die weiteren Szenarien. Die Erfolgskriterien wurden in diesem Szenario erfüllt.

5.4.5 Untersuchtes Szenario 2: Vergrößerung des latent space

Es wird vermutet, dass der AE im ersten Szenario einen zu niedrigdimensionierten Engpass hat und dadurch wichtige und entscheidende Informationen verloren gehen. In diesem Szenario soll derselbe AE mit einem größeren Engpass, nämlich mit 16 Dimensionen, eingesetzt und geprüft werden, ob sich dies positiv auf die Ergebnisse auswirkt. Hier erfolgt ein Vergleich zu Szenario 1.
Aufgrund dieses minimalen Unterschieds wird auf die zusätzliche Darstellung des Netzes verzichtet.

OPERATION		DATA DIMENSIONS			WEIGHTS(N)	WEIGHTS(%)
Input	#####	128	256	3		
InputLayer	|	----------			0	0.0%
	#####	128	256	3		
Conv2D	|/	----------			448	4.4%
relu	#####	128	256	16		
BatchNormalization	μ|σ	----------			64	0.6%
	#####	128	256	16		
MaxPooling2D	Y max	----------			0	0.0%
	#####	64	128	16		
Conv2D	|/	----------			2320	22.8%
relu	#####	64	128	16		
BatchNormalization	μ|σ	----------			64	0.6%
	#####	64	128	16		
MaxPooling2D	Y max	----------			0	0.0%
	#####	32	64	16		
Conv2D	|/	----------			2320	22.8%
relu	#####	32	64	16		
BatchNormalization	μ|σ	----------			64	0.6%
	#####	32	64	16		
MaxPooling2D	Y max	----------			0	0.0%
	#####	16	32	16		
Conv2DTranspose	/|\	----------			2320	22.8%
relu	#####	32	64	16		
BatchNormalization	μ|σ	----------			64	0.6%
	#####	32	64	16		
Conv2DTranspose	/|\	----------			2320	22.8%
relu	#####	64	128	16		
BatchNormalization	μ|σ	----------			64	0.6%
	#####	64	128	16		
Conv2DTranspose	/|\	----------			145	1.4%
sigmoid	#####	128	256	1		

Abbildung 5-15 Szenario 2: Netzzusammenfassung

Parameter-Einstellungen:
Filteranzahl = 16
Faltungskern = 3x3
Verlustfunktion = binary crossentropy
Optimierer = Adam
Batch Size = 16
Epochen = 110

Weitere Informationen:
Gesamtanzahl Parameter = 10193

Ergebnisse

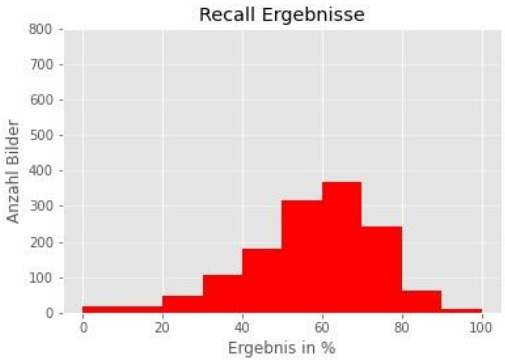


Abbildung 5-16 Szenario 2: Recall Ergebnisse

Maximum: 98.5% Minimum: 0.0% Durchschnitt: 57.7%
Ausnahmen: 47 Bilder

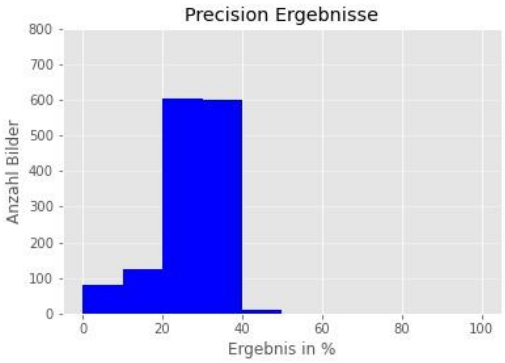


Abbildung 5-17 Szenario 2: Precision Ergebnisse

Maximum: 43.6% Minimum: 0.0% Durchschnitt: 27.0%
Ausnahmen: 3 Bilder

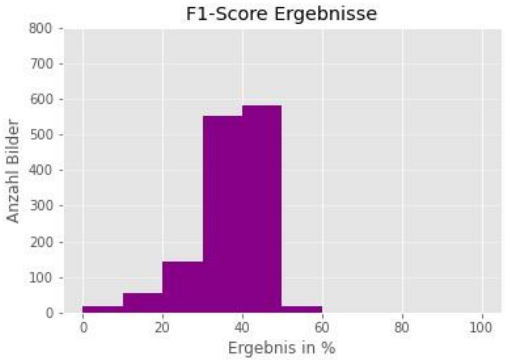


Abbildung 5-18 Szenario 2: F1-Score Ergebnisse

Maximum: 54.7% Minimum: 1.3% Durchschnitt: 37.2%
Ausnahmen: 54 Bilder

Kategorie 1: Bilder mit vielen Autos

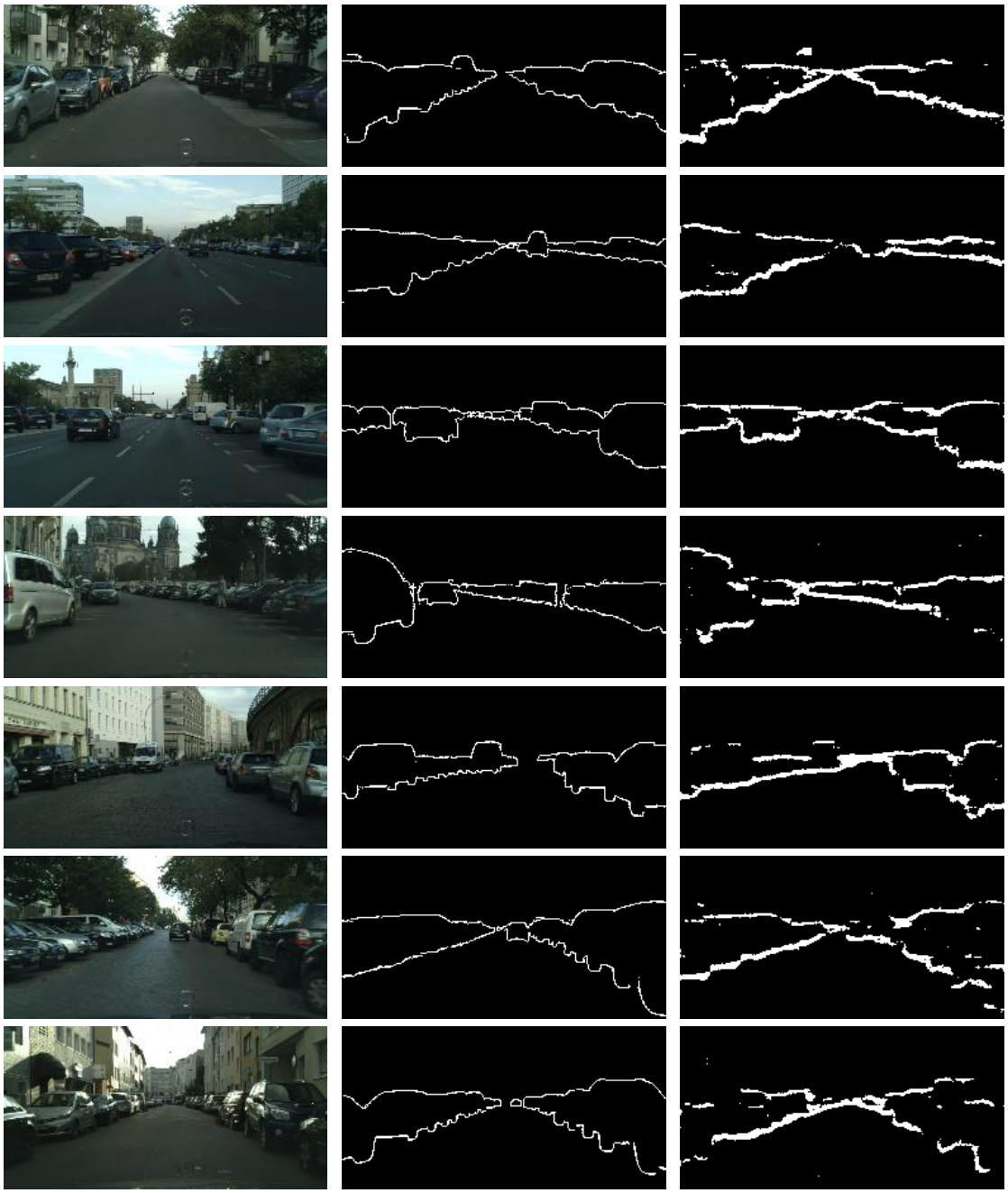


Abbildung 5-19 Szenario 2, Kategorie 1: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Kategorie 2: Bilder mit wenigen oder einzelnen Autos

Abbildung 5-20 Szenario 2, Kategorie 2: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Kategorie 3: Bilder ohne Autos

Abbildung 5-21 Szenario 2, Kategorie 3: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Bewertung

Vergleich Szenario 1 mit Szenario 2

Recall
Die Recall-Werte haben im Bereich von 60% bis 80% leicht zugenommen. Dementsprechend fällt der Durchschnitt um 1,1% höher aus und beträgt nun 57,7%. Hier sind minimale Verbesserungen ersichtlich.

Precision
Die Säulen im Bereich von 20% bis 30% sowie 30% bis 40% sind nun fast gleich hoch. Das kommt daher, weil ein Anteil der Werte im Bereich 30% bis 40% in den niedrigeren Bereich rüber gewandert ist. Insgesamt sind die Precision-Werte etwas schlechter als im letzten Szenario. Der Durchschnitt, der leicht um 0,7% gefallen ist und nun bei 27,0% liegt, bestätigt diese Aussage.

F1-Score
Das Histogramm ist nahe identisch mit dem aus Szenario 1. Eine ganz geringe Abnahme der Werte im Bereich 50% bis 60% ist zu erkennen. Insgesamt fällt der Durchschnitt um den Wert von 0,1% auf 37,3%.

Subjektive Genauigkeit
Auch hier findet der AE die grundlegenden Autokanten in den Bildern wieder. Bei der Kategorie 2 und 3 findet es wieder mehr Autos als auf den Bildern überhaupt vorhanden ist. Darüber hinaus werden die Autokanten ebenfalls nicht so gut geschlossen. Die Ergebnisse zeigen die gleichen Schwächen wie im ersten Szenario auf.

Entscheidung
Eine kleine Zunahme der Recall-Werte geht mit einer kleinen Abnahme der Precision-Werte einher. Die F1-Score-Werte sind nahezu identisch. Die Beispiele für die subjektive Genauigkeit zeigen in etwa dieselben Ergebnisse wie beim ersten Szenario. Schlussendlich lässt sich im Vergleich zu Szenario 1 sagen, dass dieses Szenario keine wesentlichen Verbesserungen in den Ergebnissen gezeigt hat. Die Erfolgskriterien werden hier erfüllt. Weil die Ergebnisse nicht besser geworden sind, wird die Vergrößerung des latent space in den weiteren Szenarien nicht weiter berücksichtigt.

5.4.6 Untersuchtes Szenario 3: Vergrößerung des Faltungskerns auf 5x5

In diesem Szenario soll der Einfluss eines anderen Parameters auf das Ergebnis untersucht werden. In den vorangegangenen Szenarien wurde ein Faltungskern der Größe 3x3 verwendet. Die Größenangabe des Faltungskerns erfolgt immer mit ungeraden Zahlen, da bei einer solchen Größe alle Pixel symmetrisch um das Ausgabepixel herum angeordnet werden. Werden für die Größe eine Faltungskerns jedoch gerade Zahlen verwendet, müssten Verzerrungen über die Ebenen hinweg berücksichtigt werden, was die Implementierung erschwert. Dieses Szenario soll folgende Frage beantworten: Führt eine Vergrößerung des Faltungskerns zu besseren Ergebnissen? Hierbei soll ein Faltungskern der Größe 5x5 verwendet werden. Die Ergebnisse dieses Szenarios werden mit den Ergebnissen aus Szenario 1 verglichen.
Auf die Darstellung der Netzarchitektur wird verzichtet, da der Faltungskern ohnehin nicht in der Abbildung zu sehen ist.

OPERATION		DATA DIMENSIONS			WEIGHTS(N)	WEIGHTS(%)
Input	#####	128	256	3		
InputLayer	|	----------			0	0.0%
	#####	128	256	3		
Conv2D	|/	----------			1216	5.7%
relu	#####	128	256	16		
BatchNormalization	μ|σ	----------			64	0.3%
	#####	128	256	16		
MaxPooling2D	Y max	----------			0	0.0%
	#####	64	128	16		
Conv2D	|/	----------			6416	30.3%
relu	#####	64	128	16		
BatchNormalization	μ|σ	----------			64	0.3%
	#####	64	128	16		
MaxPooling2D	Y max	----------			0	0.0%
	#####	32	64	16		
Conv2D	|/	----------			3208	15.2%
relu	#####	32	64	8		
BatchNormalization	μ|σ	----------			32	0.2%
	#####	32	64	8		
MaxPooling2D	Y max	----------			0	0.0%
	#####	16	32	8		
Conv2DTranspose	/|\	----------			3216	15.2%
relu	#####	32	64	16		
BatchNormalization	μ|σ	----------			64	0.3%
	#####	32	64	16		
Conv2DTranspose	/|\	----------			6416	30.3%
relu	#####	64	128	16		
BatchNormalization	μ|σ	----------			64	0.3%
	#####	64	128	16		
Conv2DTranspose	/|\	----------			401	1.9%
sigmoid	#####	128	256	1		

Abbildung 5-22 Szenario 3: Netzzusammenfassung

Parameter-Einstellungen:
Filteranzahl = 16
Faltungskern = 5x5
Verlustfunktion = binary crossentropy
Optimierer = Adam
Batch Size = 16
Epochen = 50

Weitere Informationen:
Gesamtanzahl Parameter = 21161

Ergebnisse

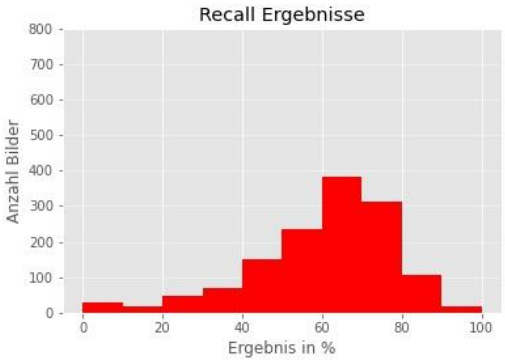


Abbildung 5-23 Szenario 3: Recall Ergebnisse

Maximum: 100.0% Minimum: 0.0% Durchschnitt: 60.2%
Ausnahmen: 47 Bilder

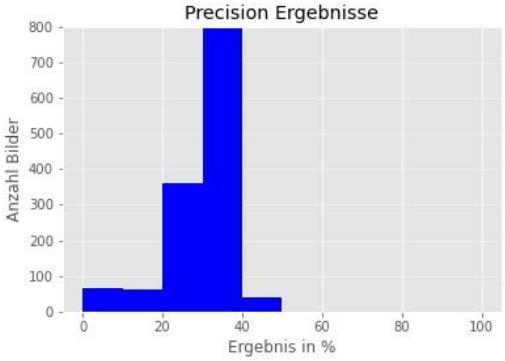


Abbildung 5-24 Szenario 3: Precision Ergebnisse

Maximum: 50.0% Minimum: 0.0% Durchschnitt: 29.9%
Ausnahmen: 7 Bilder

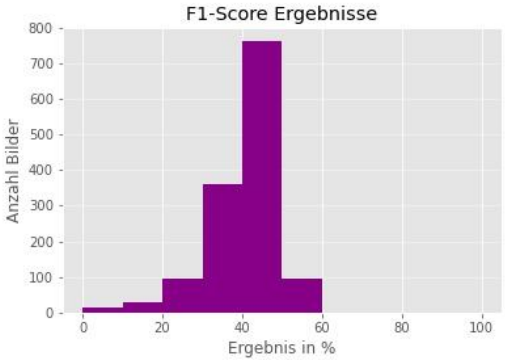


Abbildung 5-25 Szenario 3: F1-Score Ergebnisse

Maximum: 62.7% Minimum: 0.3% Durchschnitt: 40.5%
Ausnahmen: 62 Bilder

Kategorie 1: Bilder mit vielen Autos

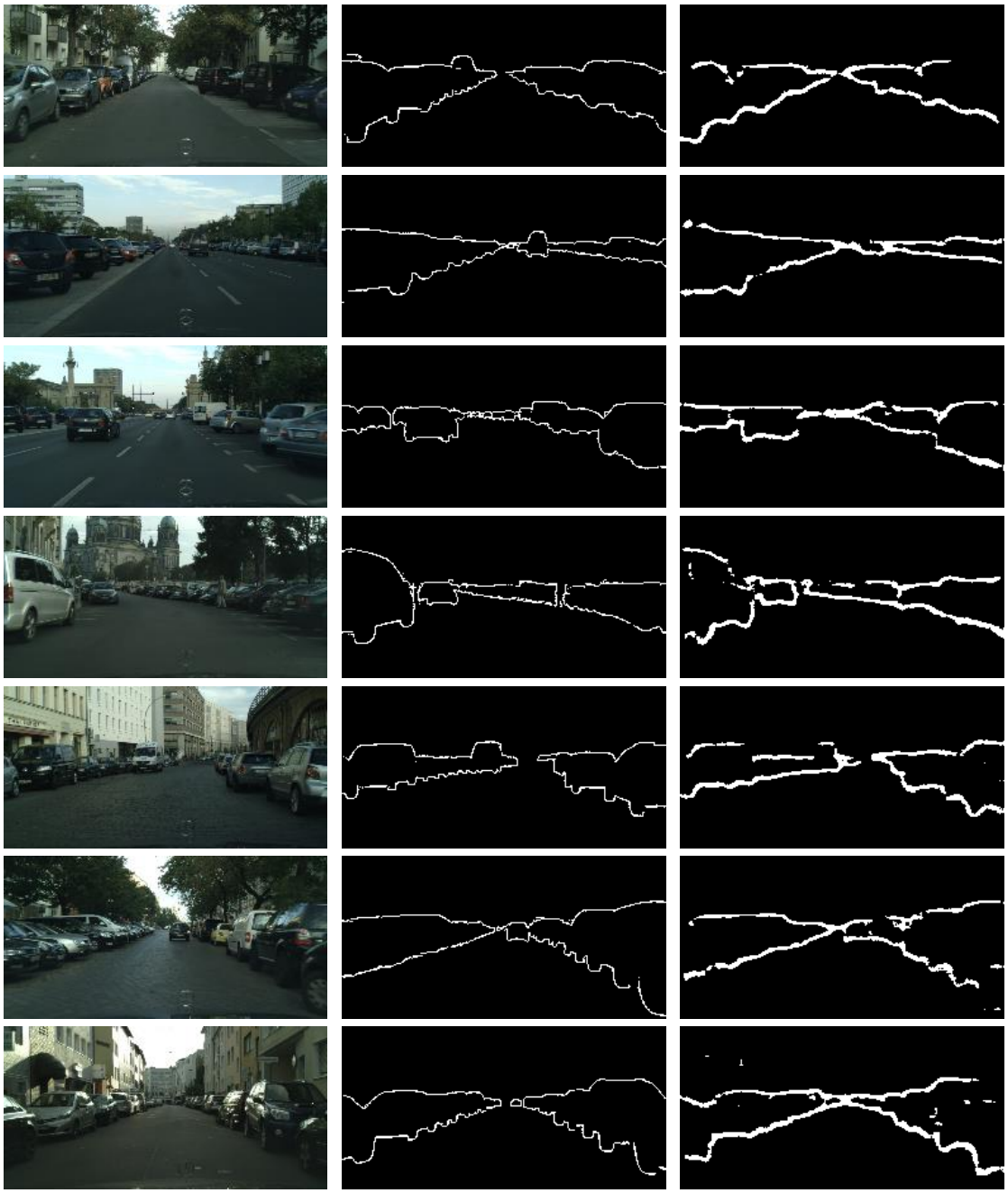


Abbildung 5-26 Szenario 3, Kategorie 1: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Kategorie 2: Bilder mit wenigen oder einzelnen Autos

Abbildung 5-27 Szenario 3, Kategorie 2: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Kategorie 3: Bilder ohne Autos

Abbildung 5-28 Szenario 3, Kategorie 3: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Bewertung

Vergleich Szenario 1 mit Szenario 3

Recall
Es ist eine Zunahme der Recall-Werte im Bereich 60% bis 80% erkennbar. Der Durchschnittswert ist um 3,6% angestiegen und bestätigt diese Verbesserung.

Precision
Durch eine einfache Beobachtung wird festgestellt, dass die Werte im Bereich von 0% bis 30% insgesamt abgenommen haben. Dafür sind die Werte im Bereich von 30% bis 40% sehr stark angestiegen. In diesem Bereich befinden sich nun knapp 800 Bilder, was eine deutliche Verbesserung in den Precision-Wert zeigt. Auch der Anteil im Bereich 40% bis 50% ist leicht angestiegen. Der Durchschnitt hat sich um 2,2% erhöht und liegt nun bei 29,9%.

F1-Score
Dieselbe Feststellung wie in den Precision-Werten wird auch gemacht. Der einzige Unterschied ist, dass insbesondere die Werte im Bereich von 40% bis 50% stark angestiegen sind. Zudem lässt sich ein geringer Anstieg der Werte im Bereich von 50% bis 60% beobachten. Die F1-Score-Werte sind insgesamt besser. Das zeigt sich auch im Durchschnittswert, der um 3,2% zugenommen hat und damit über 40% liegt.

Subjektive Genauigkeit
Die grundlegenden Kanten werden auch hier vom AE erkannt. Zudem ist folgendes sehr auffällig: Die Kantenverläufe in den Bildern sind flüssiger als im Vergleich zu Szenario 1. Feinheiten hingegen werden weiterhin nicht erkannt. Darüber hinaus erkennt der AE nicht mehr übermäßig viele Autos in der Kategorie 2. Das Vergrößern der Faltungsmaske hat dazu geführt, dass dieser Effekt eingedämpft wird. In der Kategorie 3 lassen sich keine deutlichen Verbesserungen feststellen. Auch hier werden bei einigen Bildern mehr Autos und bei einigen anderen Bildern weniger Autos detektiert. Darüber hinaus wird folgendes festgestellt: die Autokanten werden insgesamt besser geschlossen.

Entscheidung
Die Precision-, Recall- und F1-Score-Werte sind insgesamt besser. Die subjektive Genauigkeit zeigt das anhand der einzelnen Bilder sehr gut. Aus diesen Erkenntnissen und Beobachtungen lässt sich sagen, dass das Vergrößern der Faltungsmaske zu deutlichen Verbesserungen in den Ergebnissen geführt hat. Dieses Szenario erfüllt ebenfalls die Erfolgskriterien.

5.4.7 Untersuchtes Szenario 4: Vergrößerung des Faltungskerns auf 7x7

Das letzte Szenario mit der Vergrößerung des Faltungskerns hat zu wesentlichen Verbesserung geführt. In diesem Szenario soll diese Untersuchung der Faltungsmaske weitergeführt werden. Hierzu wird ein Faltungskern der Größe 7x7 verwendet und untersucht, welchen Einfluss diese Vergrößerung auf die Ergebnisse hat. Der Vergleich erfolgt zu den Ergebnissen des dritten Szenarios.
Auch hier wird auf die Darstellung der Netzarchitektur verzichtet, da der Faltungskern ohnehin nicht in der Abbildung zu sehen ist.

OPERATION		DATA DIMENSIONS			WEIGHTS(N)	WEIGHTS(%)
Input	#####	128	256	3		
InputLayer	|	----------			0	0.0%
	#####	128	256	3		
Conv2D	|/	----------			2368	5.8%
relu	#####	128	256	16		
BatchNormalization	μ|σ	----------			64	0.2%
	#####	128	256	16		
MaxPooling2D	Y max	----------			0	0.0%
	#####	64	128	16		
Conv2D	|/	----------			12560	30.5%
relu	#####	64	128	16		
BatchNormalization	μ|σ	----------			64	0.2%
	#####	64	128	16		
MaxPooling2D	Y max	----------			0	0.0%
	#####	32	64	16		
Conv2D	|/	----------			6280	15.3%
relu	#####	32	64	8		
BatchNormalization	μ|σ	----------			32	0.1%
	#####	32	64	8		
MaxPooling2D	Y max	----------			0	0.0%
	#####	16	32	8		
Conv2DTranspose	/|\	----------			6288	15.3%
relu	#####	32	64	16		
BatchNormalization	μ|σ	----------			64	0.2%
	#####	32	64	16		
Conv2DTranspose	/|\	----------			12560	30.5%
relu	#####	64	128	16		
BatchNormalization	μ|σ	----------			64	0.2%
	#####	64	128	16		
Conv2DTranspose	/|\	----------			785	1.9%
sigmoid	#####	128	256	1		

Abbildung 5-29 Szenario 4: Netzzusammenfassung

Parameter-Einstellungen:
Filteranzahl = 16
Faltungskern = 7x7
Verlustfunktion = binary crossentropy
Optimierer = Adam
Batch Size = 16
Epochen = 30

Weitere Informationen:
Gesamtanzahl Parameter = 41129

Ergebnisse

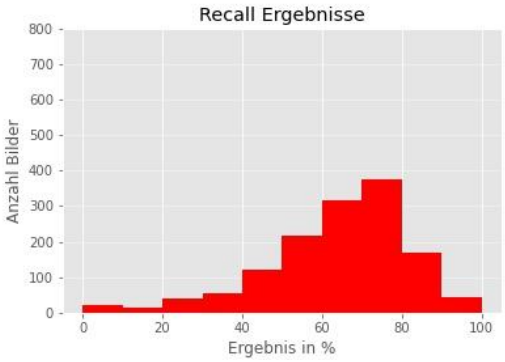


Abbildung 5-30 Szenario 4: Recall Ergebnisse

Maximum: 100.0% Minimum: 0.0% Durchschnitt: 63.8%
Ausnahmen: 47 Bilder

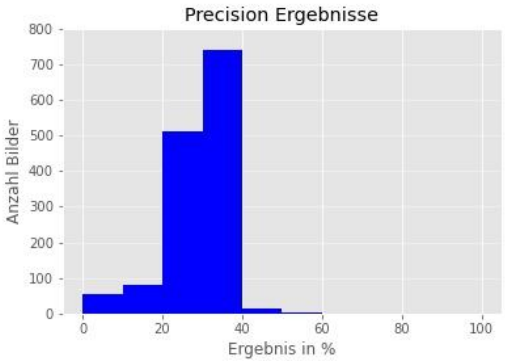


Abbildung 5-31 Szenario 4: Precision Ergebnisse

Maximum: 70.8% Minimum: 0.0% Durchschnitt: 28.6%
Ausnahmen: 20 Bilder

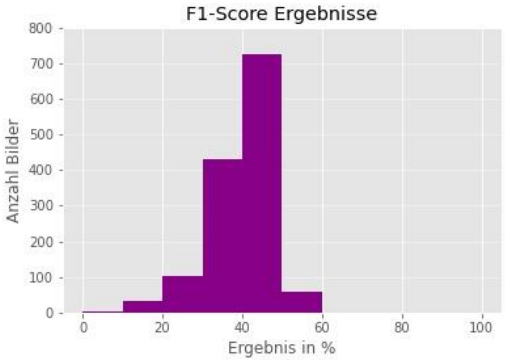


Abbildung 5-32 Szenario 4: F1-Score Ergebnisse

Maximum: 59.5% Minimum: 2.2% Durchschnitt: 39.8%
Ausnahmen: 62 Bilder

Kategorie 1: Bilder mit vielen Autos

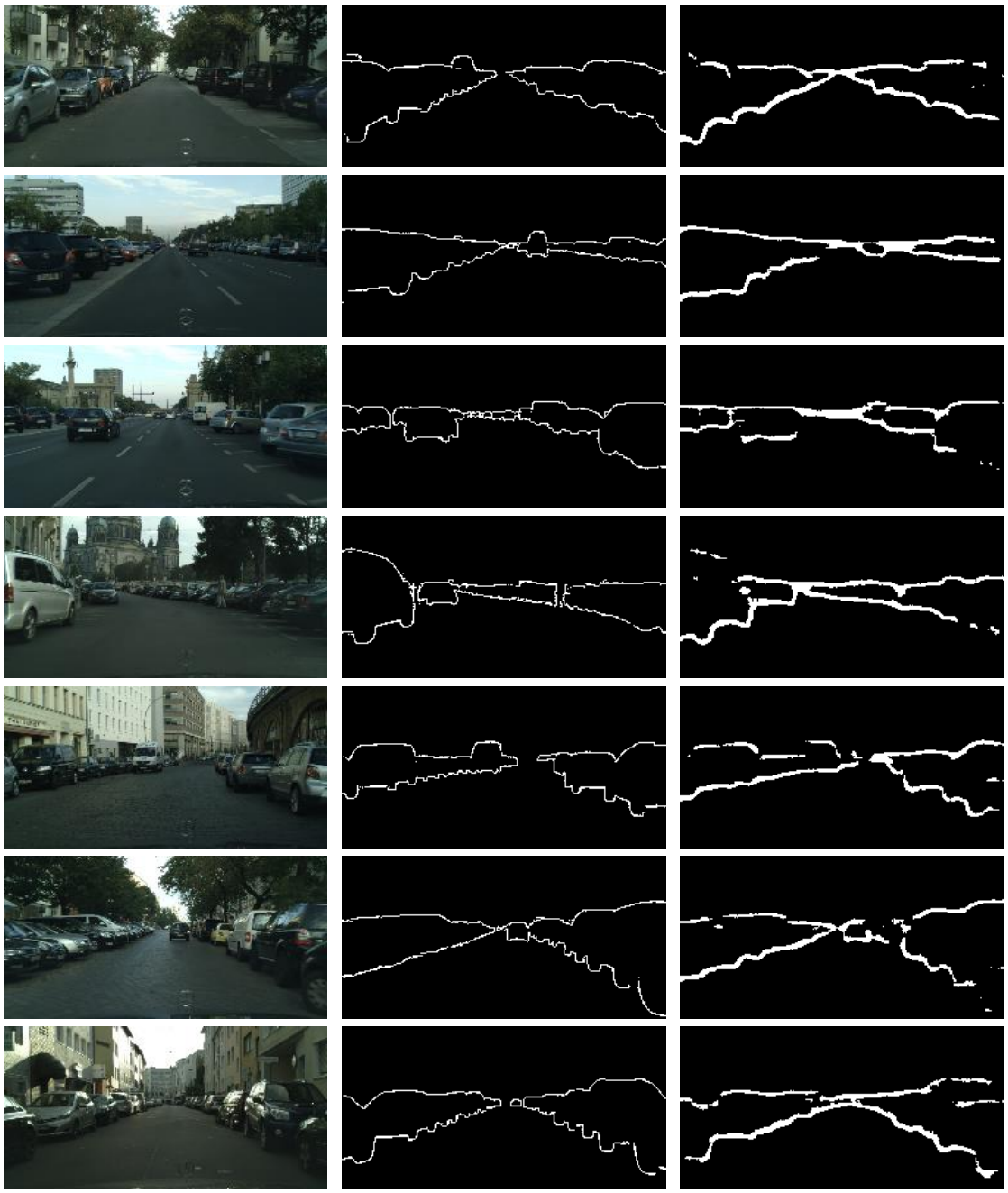


Abbildung 5-33 Szenario 4, Kategorie 1: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Kategorie 2: Bilder mit wenigen oder einzelnen Autos

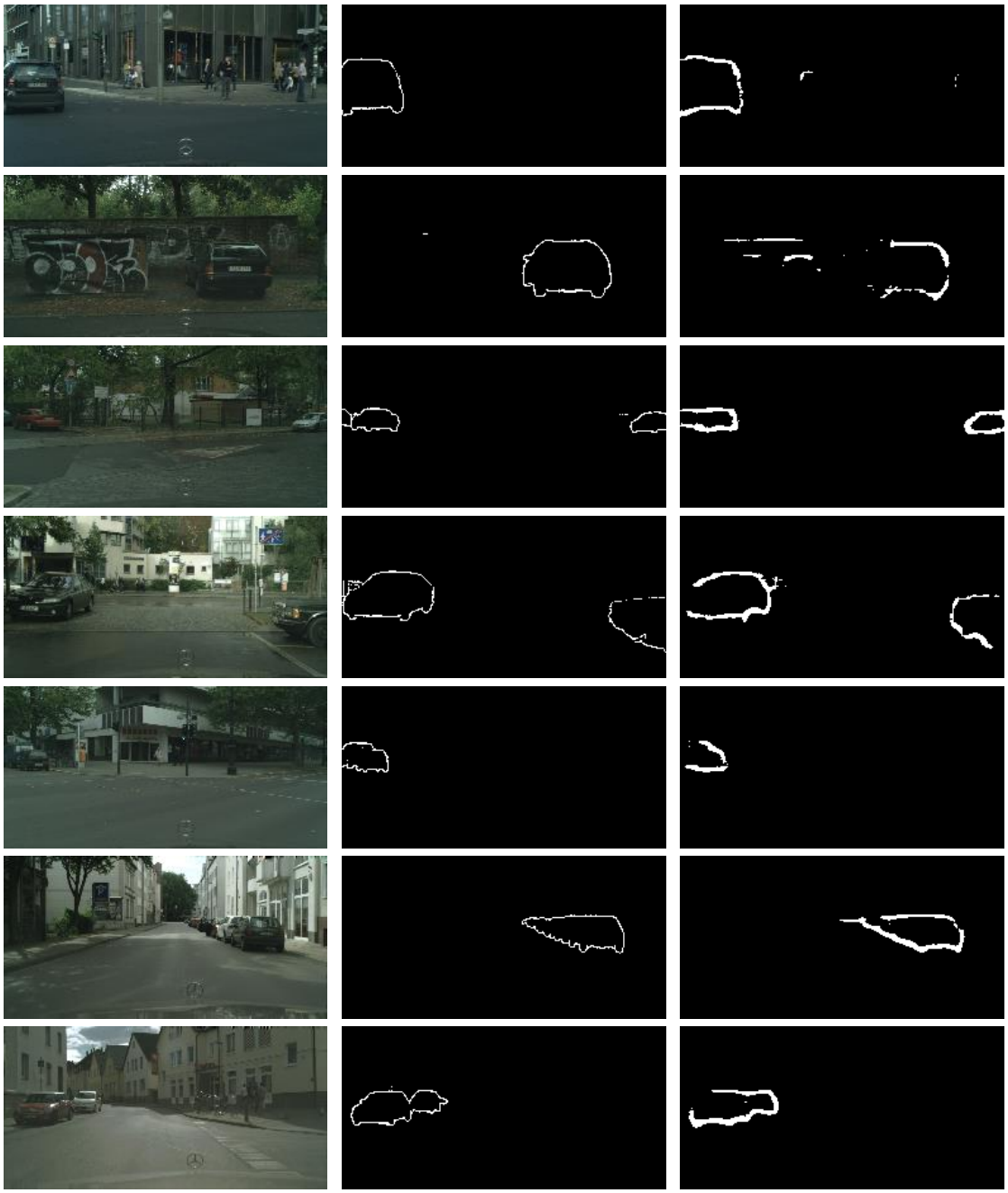


Abbildung 5-34 Szenario 4, Kategorie 2: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Kategorie 3: Bilder ohne Autos

Abbildung 5-35 Szenario 4, Kategorie 3: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Bewertung

Vergleich Szenario 3 mit Szenario 4

Recall
Der Anteil der Bilder, der insgesamt unter 50% liegt, ist nahezu identisch. Insbesondere lässt sich eine deutliche Zunahme der Werte im Bereich 70% bis 100% feststellen. Der Durchschnitt, der um 3,6% zugenommen hat, bestätigt diese Beobachtung.

Precision
Mehr als 700 und damit die Hälfte der Bilder liegt im Bereich von 30% bis 40%. Ein geringer Anteil der Werte ist vom Bereich 30% bis 40% auf den Bereich 20% bis 30% rüber gewandert. Der Durchschnitt ist folglich um den Wert 1,3% gefallen und beträgt nun 28,6%.

F1-Score
Grob betrachtet ist das Histogramm das Ebenbild des Histogramms der Precision. Der Durchschnittswert ist um insgesamt 0,7% gefallen. Diese minimale Verringerung in den F1-Score Werten ergibt sich dadurch, weil der geringe Anstieg innerhalb der Recall-Werte mit einer geringfügig stärkeren Abnahme der Precision-Werte einhergeht.

Subjektive Genauigkeit
Bei der Kategorie 1 wird durch eine genaue Beobachtung festgestellt, dass die Kantenverläufe in den Ergebnissen minimal flüssiger sind. Zudem werden die Kanten an einigen wenigen Stellen besser geschlossen. Wiederum gibt es jedoch Stellen, wo die Kanten nicht geschlossen werden. In der Kategorie 2 werden noch weniger Autos an den Stellen detektiert, wo keine sind. Auch hier sind einige Autokanten besser geschlossen, einige andere hingegen nicht. Bei den Ergebnissen zur Kategorie 3 werden ebenfalls minimal weniger Autos detektiert. Die Veränderungen sind hier im Wesentlichen marginal.

Entscheidung
Es lässt sich nicht sicher sagen, dass die Ergebnisse wirklich besser sind. Andererseits kann auch nicht gesagt werden, dass die Ergebnisse schlechter geworden sind. In den Recall-Werten werden zwar Verbesserungen festgestellt, die jedoch mit einer Abnahme in den Precision-Werten verbunden sind. Die F1-Score-Werte sind leicht gefallen. Es können keine wesentlichen Verbesserungen, aber auch keine wirklichen Verschlechterungen festgestellt werden. Bei der subjektiven Genauigkeit sind an den ausgewählten Beispielen marginale Verbesserungen ersichtlich, jedoch lässt sich nicht sagen, ob dies ebenfalls bei einem Großteil der restlichen Bilder zutrifft. Hier liegt die Vermutung nahe, dass die Ergebnisse bei den restlichen Bildern geringfügig schlechter geworden sind. Die subjektive Genauigkeit bestätigt die Ergebnisse der drei Metriken daher zu einem Teil. Der positive Effekt aus dem Vergrößern der Faltungsmaske wurde wohl ausgeschöpft. Insgesamt ist das Ergebnis nahezu gleichgeblieben. Auch hier werden die Erfolgskriterien erfüllt.

5.4.8 Untersuchtes Szenario 5: Verwendung eines größeren Netzes

Die bisher betrachteten AE hatten eine Parameteranzahl in der Größenordnung um die 1000 bis ca. 40000. Diese Netze waren relativ klein. Die Veränderungen an der Anzahl Dimensionen im latent space als auch der Faltungskerne haben lediglich zu verhältnismäßig kleinen Änderungen in der Gesamtparameterzahl des Netzes geführt. Der AE wurde hierbei hinsichtlich seiner Struktur nicht verändert, da die Anzahl der Layer gleichgeblieben ist. Wird der Fokus nun auf die Struktur des AE gelegt, ist die nachfolgende Fragestellung interessant: Ist ein AE mit einer Parameteranzahl im fünfstelligen Bereich überhaupt ausreichend groß genug, um die hier vorliegende Aufgabe der semantischen Kantendetektion auf Autoobjekten zufriedenstellend genug lösen zu können?
Ausgehend von dieser Frage wird nun ein etwas anderer Weg hinsichtlich der Modifizierung des Netzes eingeschlagen. Statt einzelne Parameter innerhalb des Netzes zu ändern, wird nun der ganze AE etwas größer gemacht. Auf der Encoder- sowie Decoder-Seite kommen jeweils ein zusätzlicher Layer dazu. Außerdem sollen die aus den vorangegangenen Szenarien gewonnenen Erkenntnisse mit in den Aufbau des Netzes einfließen. Dementsprechend wird die Encoder-Seite so aufgebaut, dass beginnend mit einer größeren Faltungsmaske diese bei jedem nachfolgenden Layer schrittweise verkleinert wird. Auf der Decoder-Seite wird das gleiche Prinzip genau in umgekehrter Weise angewendet. Zunächst wird ein kleiner Faltungskern verwendet, die über den Decoder hin weg schrittweise vergrößert wird, parallel zur Vergrößerung der Bildgröße.
Die Vergrößerung berücksichtigt zudem einen weiteren, entscheidenden Aspekt. Es wurde bereits erläutert, dass bei CNN die Idee zugrunde liegt, eine Hierarchie von Merkmalsextraktionsschritten aufzubauen. In den vorderen Layern erlernt das Faltungsnetzwerk einfache Merkmale, für deren Erfassung ein paar Feature-Maps bereits ausreichen sind. Je mehr die Eingabedaten in die hinteren Layer eines CNNs gelangen, desto zunehmend komplexer werden die Merkmale. Um diese Komplexität erfassen zu können, werden wiederum mehr Feature-Maps notwendig. Es hat sich in der Praxis bewährt, die Bildgröße immer kleiner werden und gleichzeitig die Anzahl der Feature-Maps größer werden zu lassen. Für die Anwendung dieses Prinzips wird die Filteranzahl auf der Encoder-Seite, beginnend bei 16, bei jedem nachfolgenden Conv2D-Layer jeweils verdoppelt. Auf der Decoder-Seite findet dieses Prinzip in der umgekehrten Weise Anwendung. Angefangen bei 64 wird die Filteranzahl bei jedem Conv2DTranspose-Layer halbiert. Durch diese Modifizierungen des AE in diesem Szenario wird genau diese zugrunde liegende Idee von Faltungsnetzwerken miteingeschlossen.
Außerdem wird mit dieser Vergrößerung des gesamten Netzes auch indirekt ein Vergleich zwischen kleiner Netzstruktur (vorangegangene Szenarien) mit größerer Netzstruktur durchgeführt (aktuelles Szenario).
Nachfolgende Tabelle gibt Informationen zu der Netzkonstellation in diesem Szenario.

Bereich des AE	Layer	Filteranzahl	Faltungskern
Encoder	1. Conv2D-Layer	16	7x7
Encoder	2. Conv2D-Layer	32	5x5
Encoder	3. Conv2D-Layer	64	3x3
Decoder	1. Conv2DTranspose-Layer	64	3x3
Decoder	2. Conv2DTranspose-Layer	32	5x5
Decoder	3. Conv2DTranspose-Layer	16	7x7

Tabelle 5-2 Szenario 5: Informationen zur Netzarchitektur

Nachfolgend ist die Netzarchitektur visualisiert.

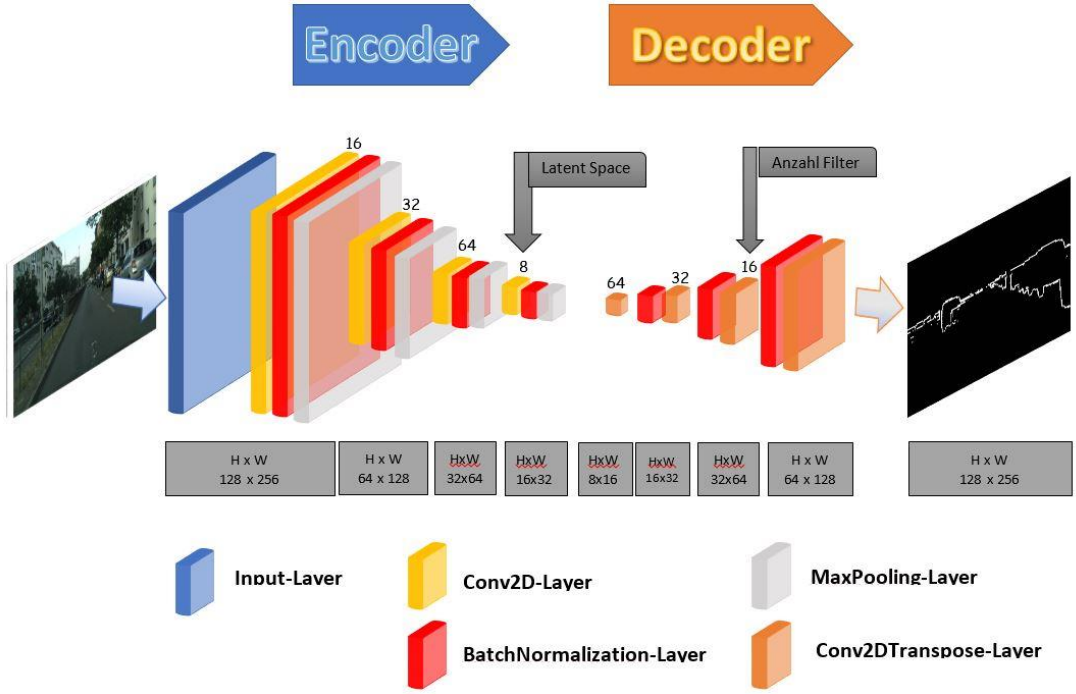


Abbildung 5-36 Netzarchitektur des vergrößerten CNN-Autoencoders

OPERATION		DATA DIMENSIONS			WEIGHTS(N)	WEIGHTS(%)
Input	#####	128	256	3		
InputLayer	|				0	0.0%
	#####	128	256	3		
Conv2D	|/				2368	2.0%
relu	#####	128	256	16		
BatchNormalization	μ|σ				64	0.1%
	#####	128	256	16		
MaxPooling2D	Y max				0	0.0%
	#####	64	128	16		
Conv2D	|/				12832	10.7%
relu	#####	64	128	32		
BatchNormalization	μ|σ				128	0.1%
	#####	64	128	32		
MaxPooling2D	Y max				0	0.0%
	#####	32	64	32		
Conv2D	|/				18496	15.4%
relu	#####	32	64	64		
BatchNormalization	μ|σ				256	0.2%
	#####	32	64	64		
MaxPooling2D	Y max				0	0.0%
	#####	16	32	64		
Conv2D	|/				4616	3.8%
relu	#####	16	32	8		
BatchNormalization	μ|σ				32	0.0%
	#####	16	32	8		
MaxPooling2D	Y max				0	0.0%
	#####	8	16	8		
Conv2DTranspose	/|\				4672	3.9%
relu	#####	16	32	64		
BatchNormalization	μ|σ				256	0.2%
	#####	16	32	64		
Conv2DTranspose	/|\				51232	42.6%
relu	#####	32	64	32		
BatchNormalization	μ|σ				128	0.1%
	#####	32	64	32		
Conv2DTranspose	/|\				25104	20.9%
relu	#####	64	128	16		
BatchNormalization	μ|σ				64	0.1%
	#####	64	128	16		
Conv2DTranspose	/|\				145	0.1%
sigmoid	#####	128	256	1		

Abbildung 5-37 Szenario 5: Netzzusammenfassung

Weitere Parameter-Einstellungen:
Verlustfunktion = binary crossentropy
Optimierer = Adam
Batch Size = 16
Epochen = 30

Weitere Informationen:
Anzahl Parameter = 120393

Ergebnisse

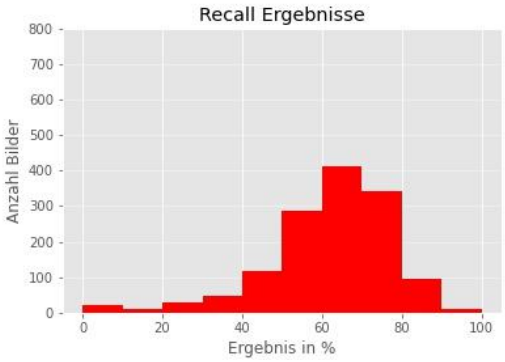


Abbildung 5-38 Szenario 5: Recall Ergebnisse

Maximum: 98.6% Minimum: 0.0% Durchschnitt: 61.9%
Ausnahmen: 47 Bilder

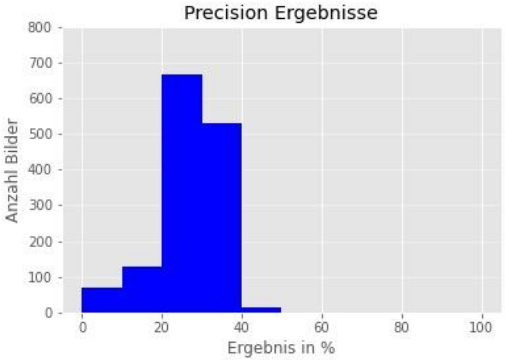


Abbildung 5-39 Szenario 5: Precision Ergebnisse

Maximum: 46.2% Minimum: 0.0% Durchschnitt: 26.8%
Ausnahmen: 13 Bilder

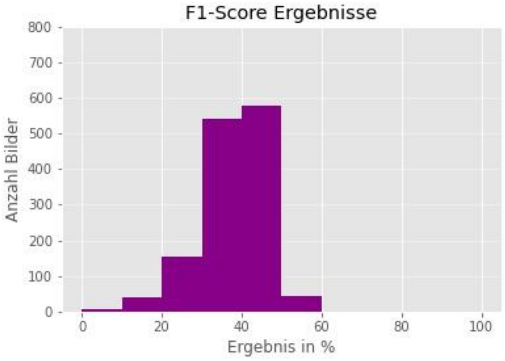


Abbildung 5-40 Szenario 5: F1-Score Ergebnisse

Maximum: 58.5% Minimum: 1.0% Durchschnitt: 38.0%
Ausnahmen: 62 Bilder

Kategorie 1: Bilder mit vielen Autos

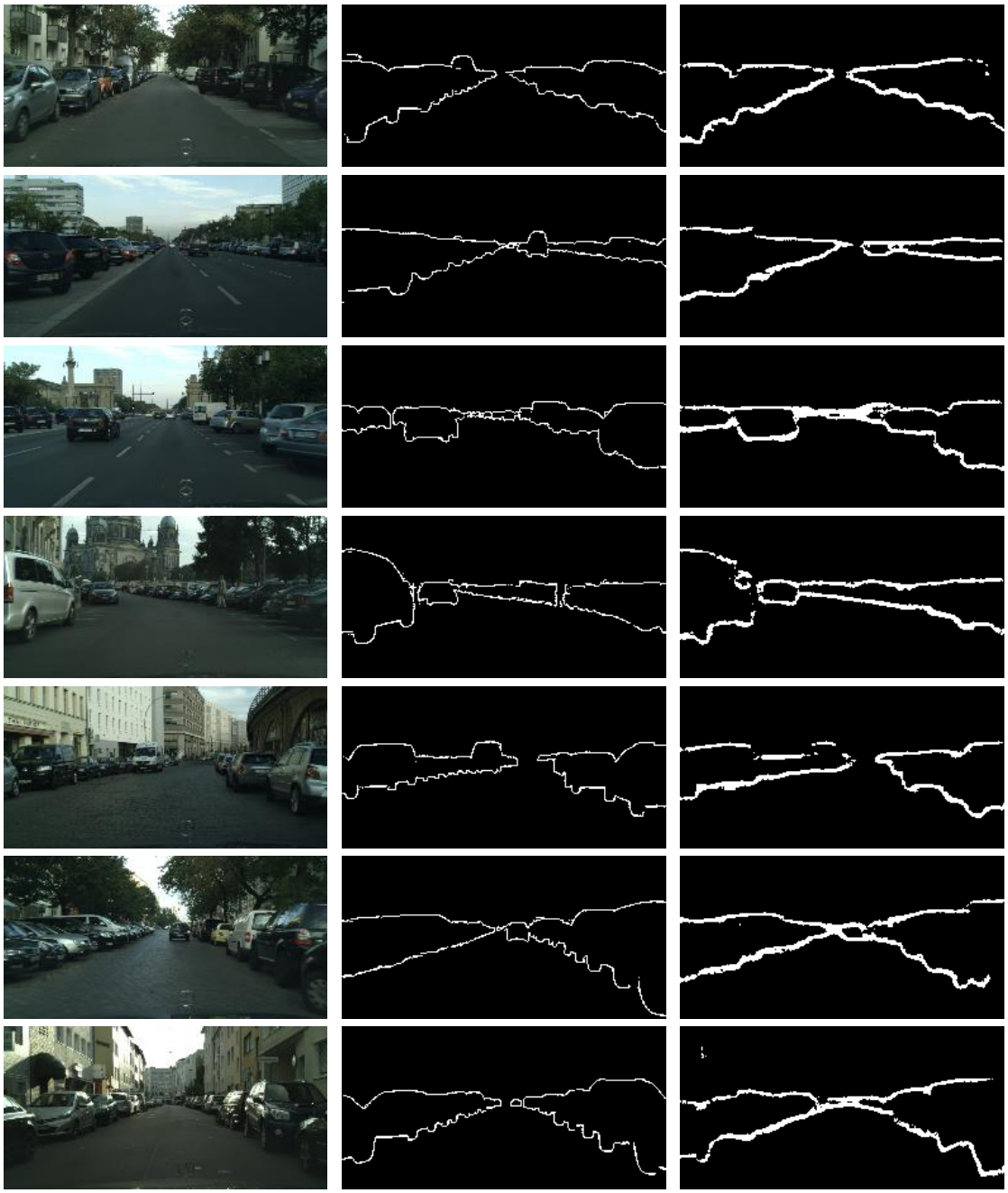


Abbildung 5-41 Szenario 5, Kategorie 1: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Kategorie 2: Bilder mit wenigen oder einzelnen Autos

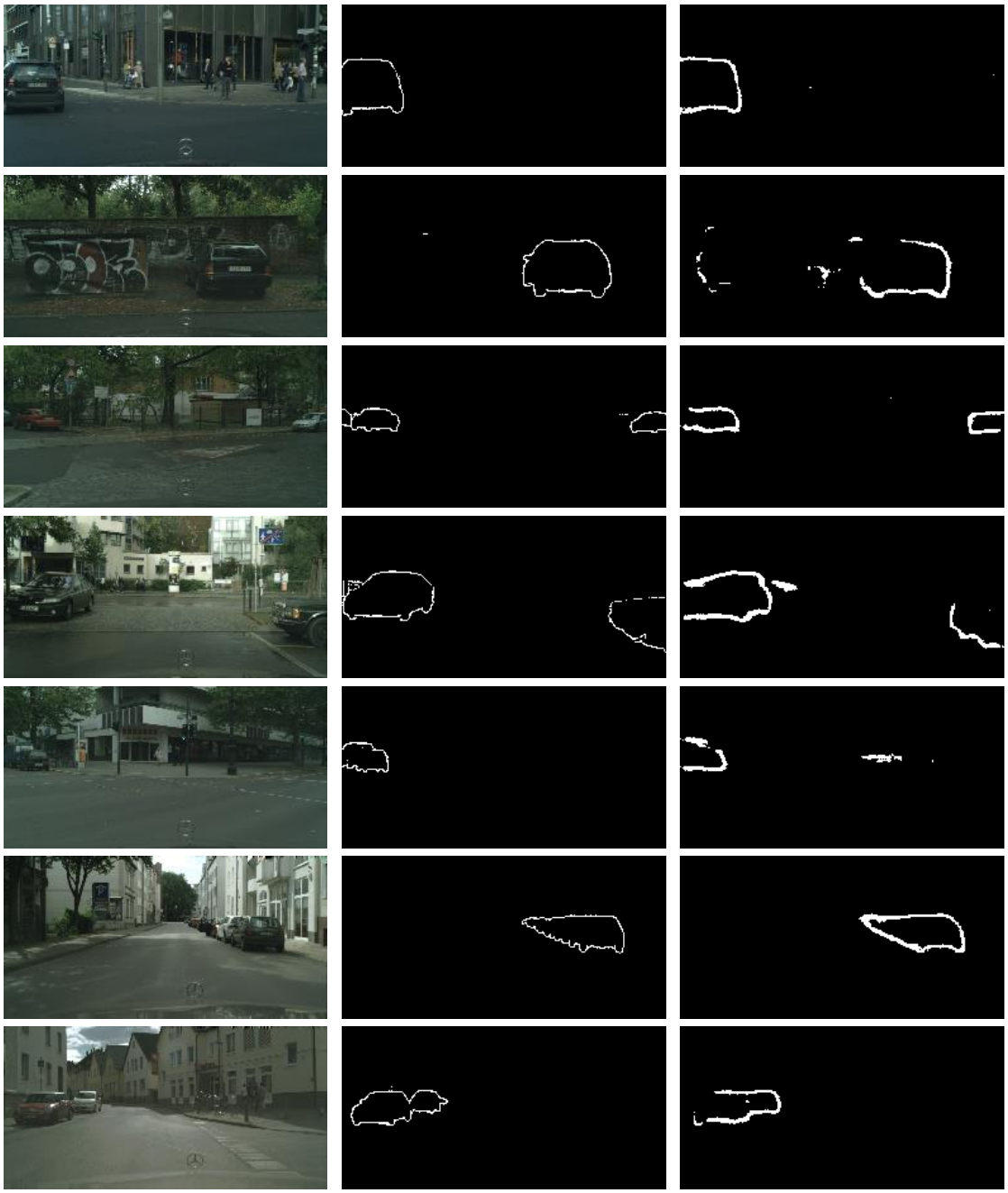


Abbildung 5-42 Szenario 5, Kategorie 2: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Kategorie 3: Bilder ohne Autos

Abbildung 5-43 Szenario 5, Kategorie 3: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Bewertung

Vergleich Szenario 3 mit Szenario 5

Hier erfolgt der Vergleich eines größeren Netzes mit einem kleineren Netz. Als Vergleichsmodell dient hierbei das Modell, das in den bisher untersuchten Szenarien die besten Ergebnisse geliefert hat. Auf Basis der bisherigen Erkenntnisse wird das Modell aus Szenario 3 zum Vergleich gewählt.

Recall
Die Anzahl der Bilder in den Bereichen 50% bis 80% sind leicht angestiegen. Entsprechend hat sich der Durchschnitt um 1,7% erhöht und ist nun bei 61,9%.

Precision
In den Werten lässt sich eine Verschlechterung beobachten. Die Anzahl der Bilder im Bereich 30% bis 40% ist von knapp 800 Bildern auf knapp über 500 Bilder gefallen. Der Bereich 20% bis 30% hingegen ist um eine Anzahl von knapp 300 Bildern angestiegen. Die restlichen Veränderungen sind eher marginal. Diese Verschlechterung zeigt sich deutlich im Durchschnitt, der 3,1% gefallen ist und 26,8% beträgt.

F1-Score
Ein ähnliches Verhalten wie bei den Precision-Werten ist auch hier zu erkennen. Im Ergebnis enthalten die Bereiche von 30% bis 40% sowie 40% bis 50% nun fast die gleiche Anzahl an Bildern. Insgesamt fällt der Durchschnitt um 2,5% auf 38,0%. Daraus lässt sich ableiten, dass infolge der Erhöhung der Recall-Werte die Precision-Werte etwas stärker abgenommen haben.

Subjektive Genauigkeit
Es können keine relevanten Veränderungen festgestellt werden. Die Ergebnisse sind im Grunde nahe identisch.

Entscheidung
Die Recall-Werte sind zwar höher, aber dementsprechend sind die Precision-Werte etwas stärker heruntergefallen, was durch die Abnahme in den F1-Score-Werten ersichtlich wird. Bei der subjektiven Genauigkeit sind keine wirklichen Veränderungen festzustellen. Die Vergrößerung des AE führt zu minimalen Verschlechterungen in den Ergebnissen. Die Erfolgskriterien erfüllt die Netzarchitektur in diesem Szenario trotzdem.

In diesem Szenario wurde der AE vergrößert und dient damit als neues Referenzmodell für die nächsten Szenarien. In diesen wird die grundlegende Netzstruktur beibehalten. Der Fokus liegt nun auf einzelnen Parametern innerhalb dieses größeren Netzes.

5.4.9 Untersuchtes Szenario 6: Verdopplung der Filteranzahl

In diesem Szenario wird die Netzarchitektur aus Szenario 5 wiederverwendet. Wie in dem Szenario bereits erläutert wurde, ist die Erhöhung der Filteranzahl eine möglicherweise gute Vorgehensweise, um die in den hinteren Layern entstehende Komplexität in den Merkmalen erfassen zu können. Hier soll untersucht werden, welchen Einfluss die gewählte Anzahl der Filter auf das Ergebnis haben. Eine interessante Frage ist hierbei: Welche Filteranzahl ist geeignet und notwendig, um die jeweils unterschiedlichen Merkmale von Kanten innerhalb der Bilder zu erfassen? Und zwar mit dem primären Zweck, die Ergebnisqualität zu steigern.
Ausgehend von dieser Fragestellung wird in diesem Szenario die Filteranzahl an den entsprechenden Layern jeweils verdoppelt. Dadurch ergibt sich ein sehr viel größeres Netz, das insgesamt über 445785 Parameter verfügt und im Vergleich zu den Netzen aus den vorangegangenen Szenarien um ein Vielfaches größer ist. In diesem Szenario erfolgt ein Vergleich mit den in Szenario 5 erzielten Ergebnissen.
Auch hier wird auf die Darstellung des Netzes verzichtet, da sich nur die Filteranzahl geändert hat, aber die eigentliche Netzstruktur hinsichtlich der Anzahl Layer gleichgeblieben ist.
Nachfolgende Tabelle gibt eine Übersicht über die Netzkonstellation mit Informationen zu den einzelnen Layern.

Bereich des AE	Layer	Filteranzahl	Faltungskern
Encoder	1. Conv2D-Layer	32	7x7
Encoder	2. Conv2D-Layer	64	5x5
Encoder	3. Conv2D-Layer	128	3x3
Decoder	1. Conv2DTranspose-Layer	128	3x3
Decoder	2. Conv2DTranspose-Layer	64	5x5
Decoder	3. Conv2DTranspose-Layer	32	7x7

Tabelle 5-3 Szenario 6: Informationen zur Netzarchitektur

Durch die nachfolgende Keras2Ascii-Ausgabe wird die Vergrößerung der Anzahl Parameter nochmal deutlich.

OPERATION		DATA DIMENSIONS	WEIGHTS(N)	WEIGHTS(%)
Input	#####	128 256 3		
InputLayer	|	----------	0	0.0%
	#####	128 256 3		
Conv2D	|/	----------	4736	1.0%
relu	#####	128 256 32		
BatchNormalization	μ|σ	----------	128	0.0%
	#####	128 256 32		
MaxPooling2D	Y max	----------	0	0.0%
	#####	64 128 32		
Conv2D	|/	----------	51264	11.2%
relu	#####	64 128 64		
BatchNormalization	μ|σ	----------	256	0.1%
	#####	64 128 64		
MaxPooling2D	Y max	----------	0	0.0%
	#####	32 64 64		
Conv2D	|/	----------	73856	16.2%
relu	#####	32 64 128		
BatchNormalization	μ|σ	----------	512	0.1%
	#####	32 64 128		
MaxPooling2D	Y max	----------	0	0.0%
	#####	16 32 128		
Conv2D	|/	----------	9224	2.0%
relu	#####	16 32 8		
BatchNormalization	μ|σ	----------	32	0.0%
	#####	16 32 8		
MaxPooling2D	Y max	----------	0	0.0%
	#####	8 16 8		
Conv2DTranspose	/|\	----------	9344	2.1%
relu	#####	16 32 128		
BatchNormalization	μ|σ	----------	512	0.1%
	#####	16 32 128		
Conv2DTranspose	/|\	----------	204864	44.9%
relu	#####	32 64 64		
BatchNormalization	μ|σ	----------	256	0.1%
	#####	32 64 64		
Conv2DTranspose	/|\	----------	100384	22.0%
relu	#####	64 128 32		
BatchNormalization	μ|σ	----------	128	0.0%
	#####	64 128 32		
Conv2DTranspose	/|\	----------	289	0.1%
sigmoid	#####	128 256 1		

Abbildung 5-44 Szenario 6: Netzzusammenfassung

Weitere Parameter-Einstellungen:

Verlustfunktion = binary crossentropy
Optimierer = Adam
Batch Size = 16
Epochen = 30

Weitere Informationen:
Anzahl Parameter = 455785

Ergebnisse

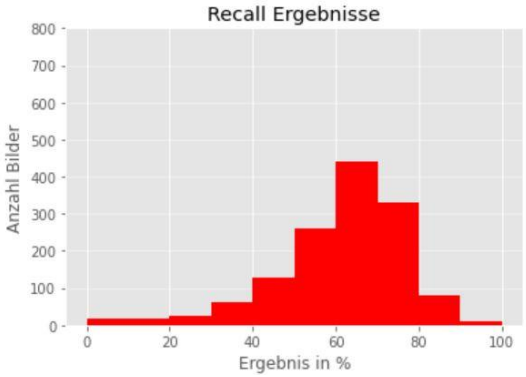


Abbildung 5-45 Szenario 6: Recall Ergebnisse

Maximum: 97.2% Minimum: 0.0% Durchschnitt: 61.6%
Ausnahmen: 47 Bilder

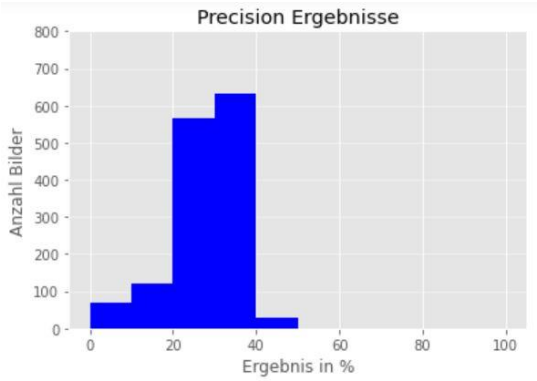


Abbildung 5-46 Szenario 6: Precision Ergebnisse

Maximum: 54.9% Minimum: 0.0% Durchschnitt: 27.8%
Ausnahmen: 10 Bilder

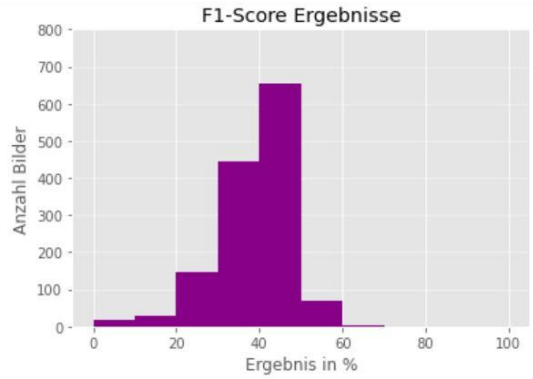


Abbildung 5-47 Szenario 6: F1-Score Ergebnisse

Maximum: 69.1% Minimum: 0.6% Durchschnitt: 38.8%
Ausnahmen: 54 Bilder

Kategorie 1: Bilder mit vielen Autos

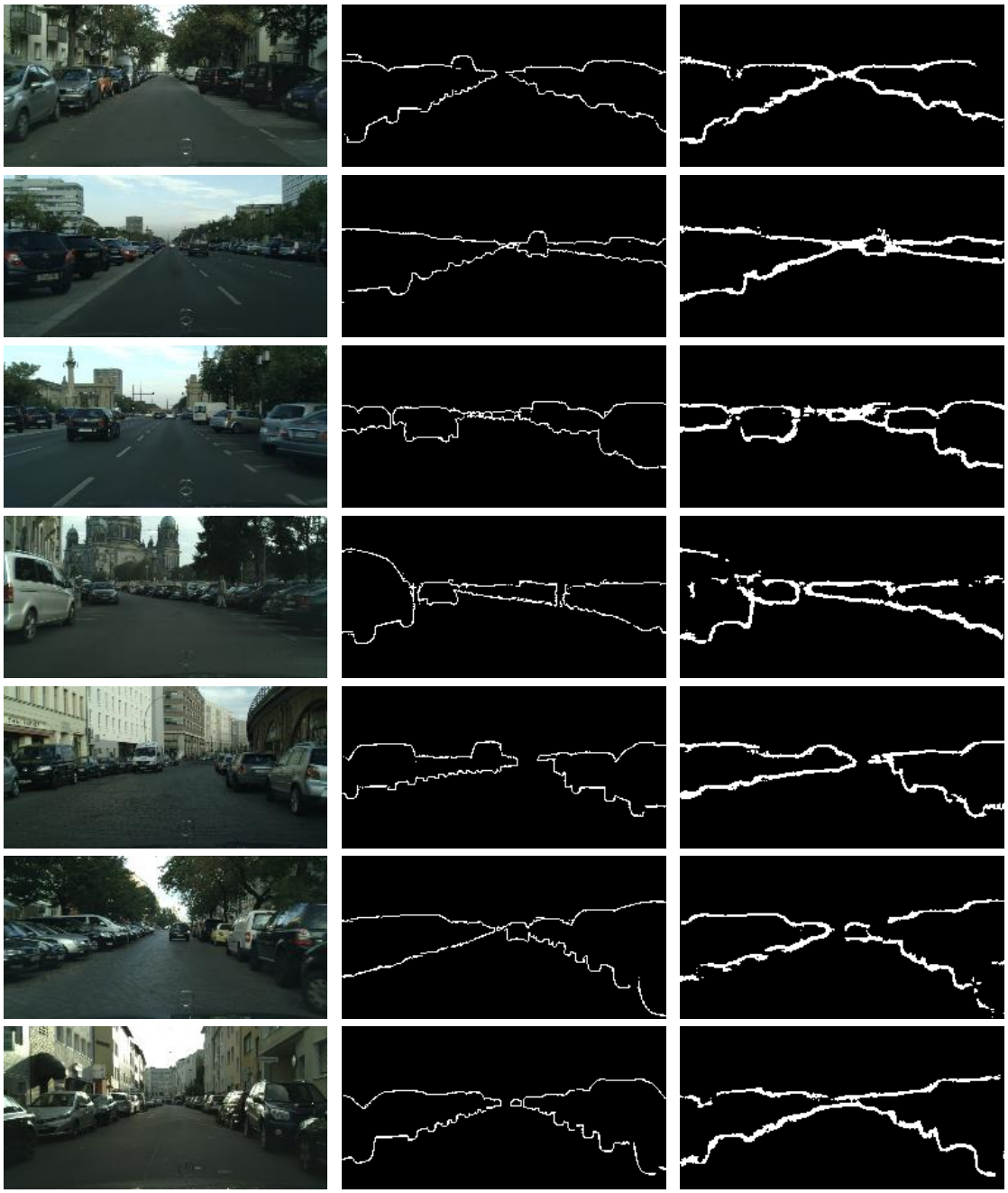


Abbildung 5-48 Szenario 6, Kategorie 1: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Kategorie 2: Bilder mit wenigen oder einzelnen Autos

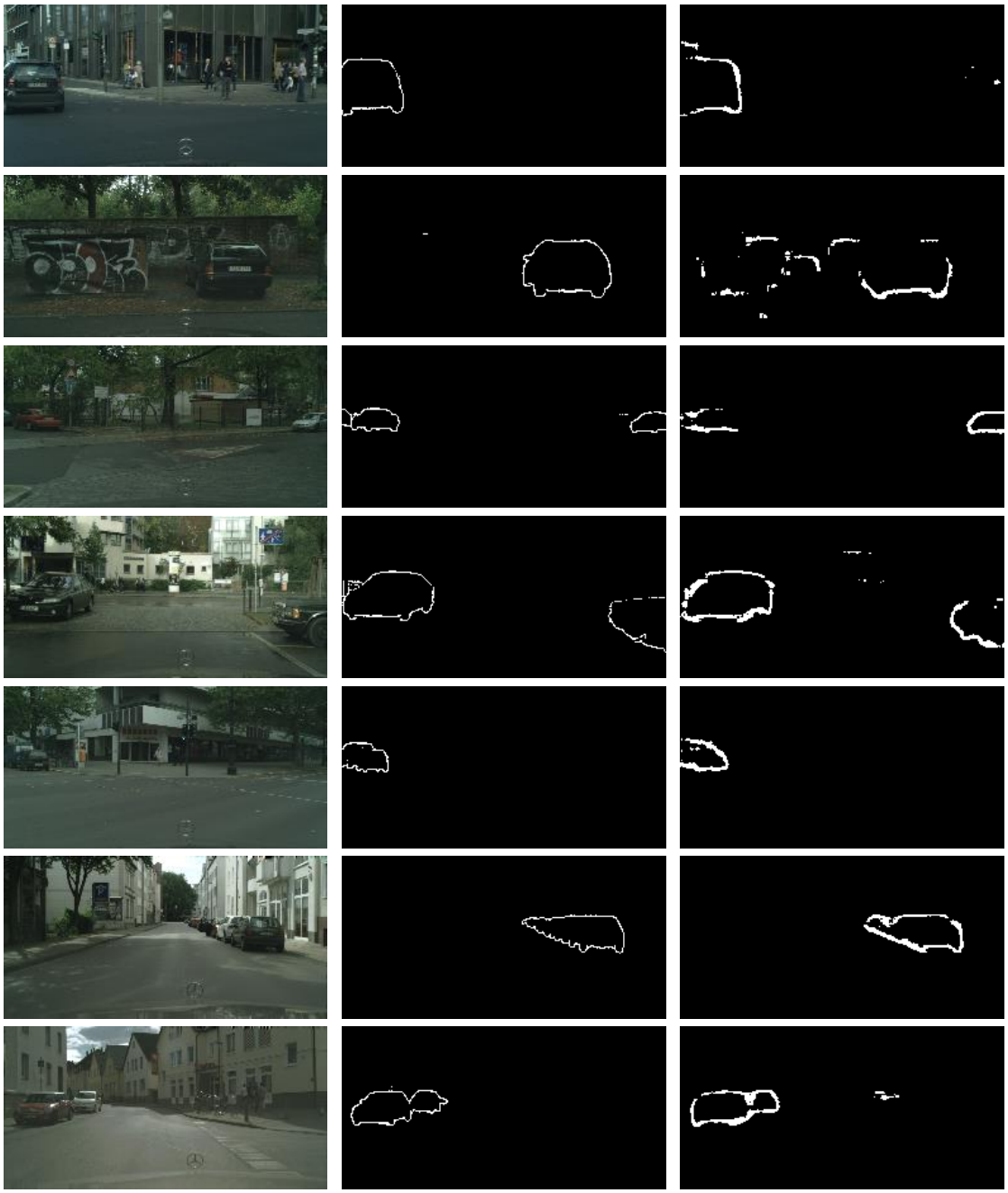


Abbildung 5-49 Szenario 6, Kategorie 2: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Kategorie 3: Bilder ohne Autos

Abbildung 5-50 Szenario 6, Kategorie 3: Eingabebild (l.), Ground Truth Kantenbild (m.), Nachbereitete Netzausgabe (r.)

Bewertung

Vergleich Szenario 5 mit Szenario 6

Recall
Die Veränderungen innerhalb der Recall-Werte sind sehr gering. Die Säulen in den unterschiedlichen Bereichen im Histogramm sind nahezu gleichgeblieben. Insgesamt fällt der Durchschnitt um 0,3% auf 61,6%. Das drückt ebenfalls aus, dass die Veränderungen minimal sind.

Precision
Hier haben die Anzahl der Bilder im Bereich 30% bis 40% um knapp 100 Bilder zugenommen. Der Bereich 20% bis 30% enthält nun entsprechend weniger Bilder. Zudem ist ein leichter Anstieg im Bereich von 40% bis 50% zu sehen. Insgesamt nimmt der Durchschnitt um 1,2% zu und beträgt nun 27,8%.

F1-Score
Beim F1-Score ist der Bereich von 40% bis 60% leicht angestiegen, während der Bereich 30% bis 40% nun in etwa 100 Bilder weniger enthält. Als Resultat hiervon ist der Durchschnitt um 0,8% auf 38,8% angestiegen.

Subjektive Genauigkeit
Auch hier lassen sich keine wesentlichen Veränderungen feststellen. Die Ergebnisse sind im Grunde nahe identisch.

Entscheidung
Die geringe Abnahme in den Recall-Werten hat diesmal eine stärkere Zunahme der Precision-Werte zufolge. Der F1-Score bestätigt dies mit einem leichten Anstieg. Jedoch sind die Veränderungen sehr gering. Dass keine wesentlichen Veränderungen zu beobachten sind, i ist auch bei den Bildern bei der subjektiven Genauigkeit zu erkennen. Die Vergrößerung der Filteranzahl hat zu keinen wesentlichen Verbesserungen in den Ergebnissen geführt. Daraus lässt sich schlussfolgern, dass die wesentlichen Merkmale bereits durch die Feature Maps erfasst wurden. Die Erfolgskriterien werden auch in diesem Szenario erfüllt.

5.4.10 Zusammenfassung der Ergebnisse

Nachfolgend sind alle Histogramme zusammengefasst dargestellt. Gezeigt werden nur die relevanten Informationen. Es wird nur der Ausschnitt vom Bereich 0%-100% übernommen.

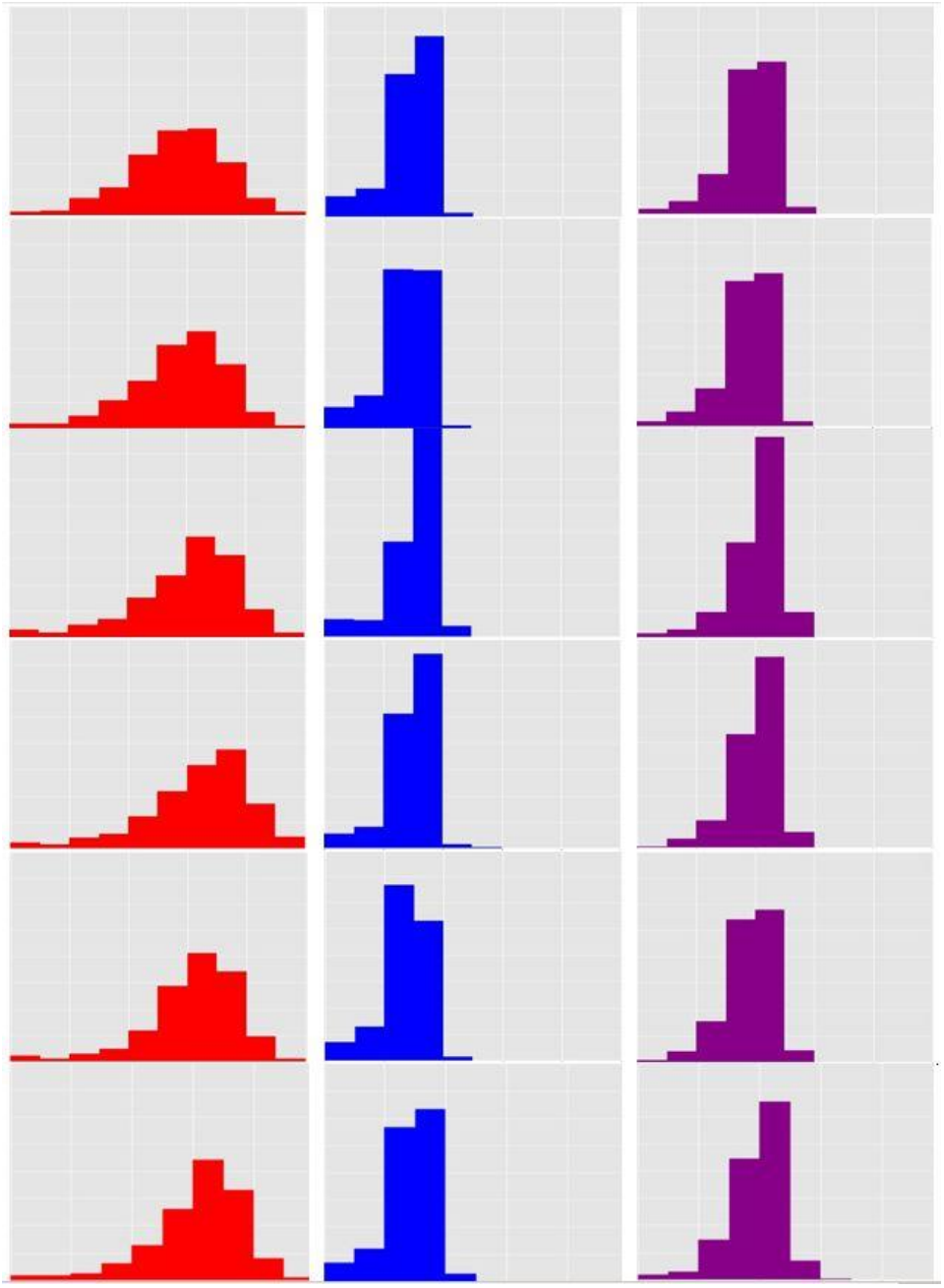


Abbildung 5-51 Zusammenfassende Übersicht der Histogramme aller untersuchten Szenarien - beginnend von Szenario 1 fortlaufend bis Szenario 6 - pro Zeile ein Szenario

6 Zusammenfassung und Ausblick

In diesem Kapitel werden die Resultate dieser Arbeit zusammengefasst und anschließend erläutert, welche Weiterentwicklungsmöglichkeiten die in dieser Arbeit gewonnenen Erkenntnisse bieten.

6.1 Fazit

In dieser Arbeit wurde das Themengebiet der semantischen Kantendetektion betrachtet und die Eignungsfähigkeit eines NNs in Bezug auf diese Aufgabe anhand des Cityscapes-Datensatzes analysiert. Bei der Kantenerkennung geht es hauptsächlich um die Informationsreduktion, indem die wesentlichen Merkmale innerhalb eines Bildes erfasst werden müssen. CNNs eignen sich generell gut für die Erkennung von Mustern in Bildern. Ein CNN-AE, der beide Aspekte kombiniert, wurde als Netzarchitektur für diese Problemstellung ausgewählt. In dieser Arbeit wurde zunächst ein solcher AE entworfen und von Grund auf an mit Daten trainiert. In Untersuchungen, die auf Basis unterschiedlicher Szenarien durchgeführt wurden, wurden Änderungen an der Netzarchitektur vorgenommen und die jeweiligen Ergebnisse anschließend miteinander verglichen.
Die unterschiedlichen Szenarien haben gezeigt, dass der AE die grundlegenden Autokanten in sehr vielen Fällen fand, insbesondere gut bei den zusammengehörigen Autos. Schwächen zeigte es darin auf, dass es an sehr vielen Stellen Autos detektiert hat, wo es keine gab. Beim genauen Hinsehen bei einigen Bildern fällt auf, dass es dort unterschiedlich viele Objekte gibt, die auf den ersten Blick wie ein Auto aussehen und auch eine Struktur besitzen, die an ein Auto erinnern könnte. Das Verhalten des AE lässt sich daraus wie folgt beschreiben: es versucht so gut es geht, nach autoähnlichen Strukturen in Bildern zu suchen und erkennt diese als Auto. Und das unabhängig davon, ob es sich bei dem Objekt um eine Haustür, eine Kiste oder einen Müllcontainer handelt. Sobald das Objekt in einem Bild auch nur annähernd eine Form wie ein Auto hat, wird es vom AE als solches identifiziert. Die Vergrößerung des latent space änderte an diesem Ergebnis nichts. Die anschließende Vergrößerung der Faltungsmaske dämpfte diesen Effekt zu einem Teil ein. Der AE erkannte weiterhin die grundlegenden Strukturen im Bild, schloss zudem die Kanten besser und detektierte insbesondere nicht mehr irrsinnig viele Autoobjekte. Die Vergrößerung des Netzes in den anschließenden Szenarien behielt die Leistung des Netzes bei, zeigte jedoch keine wesentlichen Verbesserungen oder Verschlechterungen mehr.

Bei den Histogrammen der Ergebnisse fielen gewisse Details besonders auf. Die Recall-Werte waren immer über den gesamten Bereich verteilt, wobei die Säulen an den beiden Enden sehr klein waren. Der Großteil der Werte erstreckte sich über den Bereich von 50% bis 90%. Insbesondere enthielt der Bereich von 60% bis 70% in vielen Fällen die meisten Werte. Bei der Precision erstreckten sich die Werte nur über die linke Teilhälfte. Insbesondere im Bereich 20% bis 40% waren die Säulen relativ groß. Beim F1-Score wurde eine Sache besonders bemerkbar: In einigen Szenarien war das Histogramm des F1-Scores nahezu das Ebenbild des Histogramms der Precision. Der einzige Unterschied bestand darin, dass die Säulen beim F1-Score etwas weiter nach rechts verschoben waren. Ebenso unterlag der Bereich, den die F1-Score-Werte bedeckten, einer Einschränkung. Die Werte befanden sich hauptsächlich im Bereich von 0% bis 60%. Hier lässt sich zusammenfassend sagen, dass die hohen Recall-Werte niedrigen Precision-Werten gegenüberstanden. Die Precision-Werte drückten das Gesamtergebnis ein wenig herunter.
Eine weitere Erkenntnis in dieser Arbeit ist, dass eine bessere Verteilung der Daten von enormer Bedeutung ist. Bilder mit vielen sowie wenigen Autos waren im Übermaß im Datensatz enthalten. Lediglich ein kleiner Anteil von knapp 5% waren Bilder, in denen es keine Autos gab. Das machte sich im Verhalten des AE besonders bemerkbar. Eine Erhöhung der Anzahl von Bildern ohne Autos würde dieses Verhalten entsprechend beeinflussen. Dabei sollte darauf geachtet werden, dies auf realistischer Basis umzusetzen. Eine Verteilung von 50:50 wäre hier unangemessen, da es sehr üblich ist, Autos auf Straßen zu sehen.
Insgesamt hat sich gezeigt, dass AE durchaus eine Möglichkeit sind, eine semantische Kantendetektion durchzuführen.

6.2 Ausblick

Die hier untersuchten Szenarien haben wertvolle Erkenntnisse geliefert. Diese Arbeit bietet viele Möglichkeiten für weitere Untersuchungen und soll als Grundlage für weitere Arbeiten bezüglich der semantischen Kantendetektion dienen. Im Nachfolgenden werden kurz einige Ideen für Weiterentwicklungsmöglichkeiten vorgestellt.

Erweiterung des Datensatzes
Diese Arbeit hat gezeigt, dass die semantische Kantendetektion auf Autoobjekten in Straßenszenen grundlegend funktioniert hat. Für weitere tiefergehende Untersuchungen wäre eine Erweiterung des Datensatzes angemessen und sinnvoll. Welche Bilddaten mit welchen Informationen hinzugenommen werden sollten, kann einerseits davon abhängen, in welcher Umgebung ein solches potenzielles System eingesetzt werden könnte. In dieser Arbeit wurden Bilddaten verwendet, die Aufnahmen von städtische Straßenszenen an hellen Tagen umfassen. In einigen Bildern schien unter anderem die Sonne mehr, mal weniger, unterschiedliche Objekte waren in den Bildern abgebildet und die Gegenden wechselten. Im Hinblick auf die Erweiterung der Datenbasis könnten folgende Fragen gestellt werden:

- Wie ist es mit gänzlich anderen Landschaften?
  Wälder, Berge sind hier als Beispiele zu nennen.

- Wie ist es mit anderen Orten?
  Autobahnen sind hier als Beispiel zu nennen.
- Wie sieht es bei unterschiedlichen Wetterverhältnissen aus?
  Regen und Nebel ist hier eine der möglichen Fälle. Wie gut sind Autos bei starkem Regen erkennbar? Anders verhält es sich hingegen dichtem Nebel, wo ein Auto, das gerade einmal paar Meter vor einem steht, trotzdem kaum zu erkennen ist.
- Wie ist es zu Nachtzeiten?
  Hier sind Autos auf den Straßen noch schlechter erkennbar. Im Gegensatz zur Tageszeit sind die Abblendlichter der Autos eingeschaltet sowie die Straßenlichter. Potenzielle Merkmale, an denen sich das NN bei der Detektion orientieren könnte. Parkende Autos ohne eingeschaltete Lichter jedoch müssten auf anderem Wege erkannt werden.
- Wie sieht es aus mit nicht echten Autos?
  Autos, die zwar auf riesigen Plakaten in der Hinsicht auf Werbung abgebildet sind, aber dennoch keine echten Autos sind. Oder Spiegelungen von Autos an Fensterscheiben von Läden. Andere Möglichkeiten wären Straßenschilder oder sonstige Objekte, auf denen Autos oder autoähnliche Strukturen abgebildet sind. Solche Bilder sind eine Möglichkeit zu überprüfen, wie robust die Erkennungsfähigkeit des NNs ist.

Eine übergreifende Frage, wäre in welchem Verhältnis die einzelnen Bilddaten in den Datensatz aufgenommen werden sollen. Hierbei ist höchste Vorsicht geboten. Diese Arbeit hatte gezeigt, dass ein Datensatz mit einem Übermaß an Bildern mit Autos das Verhalten des NN entsprechend beeinflusst. Es ist nicht in der Lage festzustellen, wenn in dem Bild wenige oder keine Autos enthalten sind. Hier könnten abhängig vom Einsatzort des potenziellen Systems Analysen betrieben werden, um zu ermitteln, ob an diesem Ort mit wenigen oder vielen Autos zu rechnen ist und versuchen, ein angemessenes Verhältnis innerhalb des Datensatzes zu schaffen. In einer Weiterführung der Analyse könnte die Uhrzeit mit hinzugenommen werden, zu welchen jeweils mit vielen Autos und wenigen Autos zu rechnen wäre. Das NN könnte hinsichtlich seines Verhaltens dazu gebracht werden, diese Informationen mit zu berücksichtigen.

Alternative Arbeitsumgebung

Die in dieser Arbeit verwendete GPU war sehr leistungsstark. Jedoch unterlag die hier verwendete Arbeitsumgebung einer technischen Restriktion. Die Ausführung von Jupyter-Notebooks auf dem Jupyter-Hub der HAW erfolgt immer komplett im Grafikspeicher, der auf 16 GB begrenzt ist. Durch eine andere Arbeitsumgebung mit einer GPU, wo die Daten nicht zwingend in den Grafikspeicher geladen werden müssen, könnten sehr viel mehr Bilddaten zum Training hinzugenommen werden.

Alternative Bewertungsgrundlage

Eine Verbesserungsmöglichkeit sollte auch in Bezug auf die Bewertung umgesetzt werden. Die Precision, der Recall und der F1-Score wurden in dieser Arbeit auf bildpunktorientierter

Basis berechnet. Eine mögliche Vorgehensweise wäre, indem von diesen bildpunktorientierten Maßen abstrahiert und Entscheidungen auf einem höheren Level durchgeführt werden, wie sie von Menschen getroffen werden. Eine damit verbundene Herausforderung wäre die automatisierte Durchführung dieses Vorgehens. Besteht der Datensatz aus beispielweise 50000 Bildern, dann ist die Automatisierung dieses Vorgangs auf jeden Fall sinnvoll und unumgänglich, aber stellt zugleich eine besondere Herausforderung dar. Zudem sollte dies auf eine Weise erfolgen, dass die Informationen zu den Ergebnissen in kompakter Form darstellt werden wie es mit den Histogrammen in dieser Arbeit umgesetzt wurden. Somit wäre es leichter den Überblick zu behalten und ist für Vergleiche mit anderen Ergebnissen eine gute Basis.

Durchführung weiterer Experimente und Ausprobieren alternativer Netzarchitekturen
Ansonsten könnten die hier durchgeführten Experimente in den unterschiedlichen Szenarien fortgeführt werden. Zum einen können weitere Hyperparameteranpassungen vorgenommen werden. Ein anderes Vorgehen wäre, einen noch größeren AE auszuprobieren, indem zusätzliche Layer auf der Encoder- und Decoder-Seite genommen werden. Die Netzarchitektur könnte dahingehend modifiziert werden durch den Einbau von Skip-Layer-Verbindungen. Low-Level-Informationen zusammen mit High-Level-Informationen an den hinteren Schichten des AE könnten feinere Feature-Maps erzeugen, wodurch die Qualität der Netzergebnisse gesteigert werden könnte. Alternativ könnte untersucht werden, was für Ergebnisse komplexere Modelle wie ResNet bei der semantischen Kantendetektion liefern.

Anhang

Nachfolgende Abbildung zeigt die Verzeichnisstruktur des USB-Sticks. Im Wurzelverzeichnis befinden sich mehrere Verzeichnisse und einzelne Dateien.
Der Ordner „images“ enthält alle Bilder, die in dieser Thesis verwendet wurden. Diese sind unterteilt in Ein- und Ausgabebilder, die wiederum in Trainings-, Validierungs- und Testbilder unterteilt sind. Die Ausgabebilder enthalten neben den Ground Truth-Bildern noch die Netzausgaben der unterschiedlichen Szenarien (lediglich bezogen auf den Testdatensatz). Die Ordnerbezeichnung scaled_0_125 drückt einen relativen Bezug zur Originalgröße der Bilder aus. Während mit scaled_1 die Bilder in der Originalgröße 1024x2048 gemeint sind, bezieht sich scaled_0_125 auf die Bilder, die um den Faktor 8 auf die Bildgröße 128x256 verkleinert wurden. Class_14 bezieht sich auf die verwendeten Klassenbezeichnungsnummern des Cityscapes-Datensatzes im STEAL-Projekt auf Github, der unter dem Link https://github.com/nv-tlabs/STEAL abrufbar ist. Die Zahl 14 steht dabei für die semantische Klasse Auto. Insgesamt gibt es 19 semantische Klassen.
Daneben ist der Ordner „all_notebooks“ enthalten, die alle Jupyter-Notebooks für die Experimente mit den Bildern enthält. Dazu gehören der Aufbau und das Training eines Modells. Daneben die Ergebnisberechnung für die Bewertung, das Herunterskalieren der Bilder und das Abspeichern der Bilder von Netzausgaben. Der Ordner „filenames“ enthält einzelne Textdateien, in denen die Namen der Trainings-, Validierungs- und Testbilder enthalten ist. Diese werden in jedem Jupyter-Notebook verwendet. Über die Pfadangaben und den Dateinamen werden immer auf die Bilder zugegriffen. Der Ordner „all_models“ enthält die in den Szenarien verwendeten Netze, die zusammen mit den Gewichten gespeichert wurden, sodass eine spätere Rekonstruktion möglich ist.
Im Wurzelverzeichnis befindet sich noch die Thesis als PDF-Datei. Darüber hinaus ist eine README.txt enthalten, die eine detaillierte Beschreibung über die auf dem USB-Stick befindlichen Verzeichnisse und Dateien enthält. Zudem enthält diese eine kurze Anleitung, wie für weiterführende Arbeiten vorzugehen ist. Hier sind bspw. Pfadanpassungen innerhalb der Jupyter-Notebooks für Experimente auf dem heimischen Rechner zwingend notwendig.

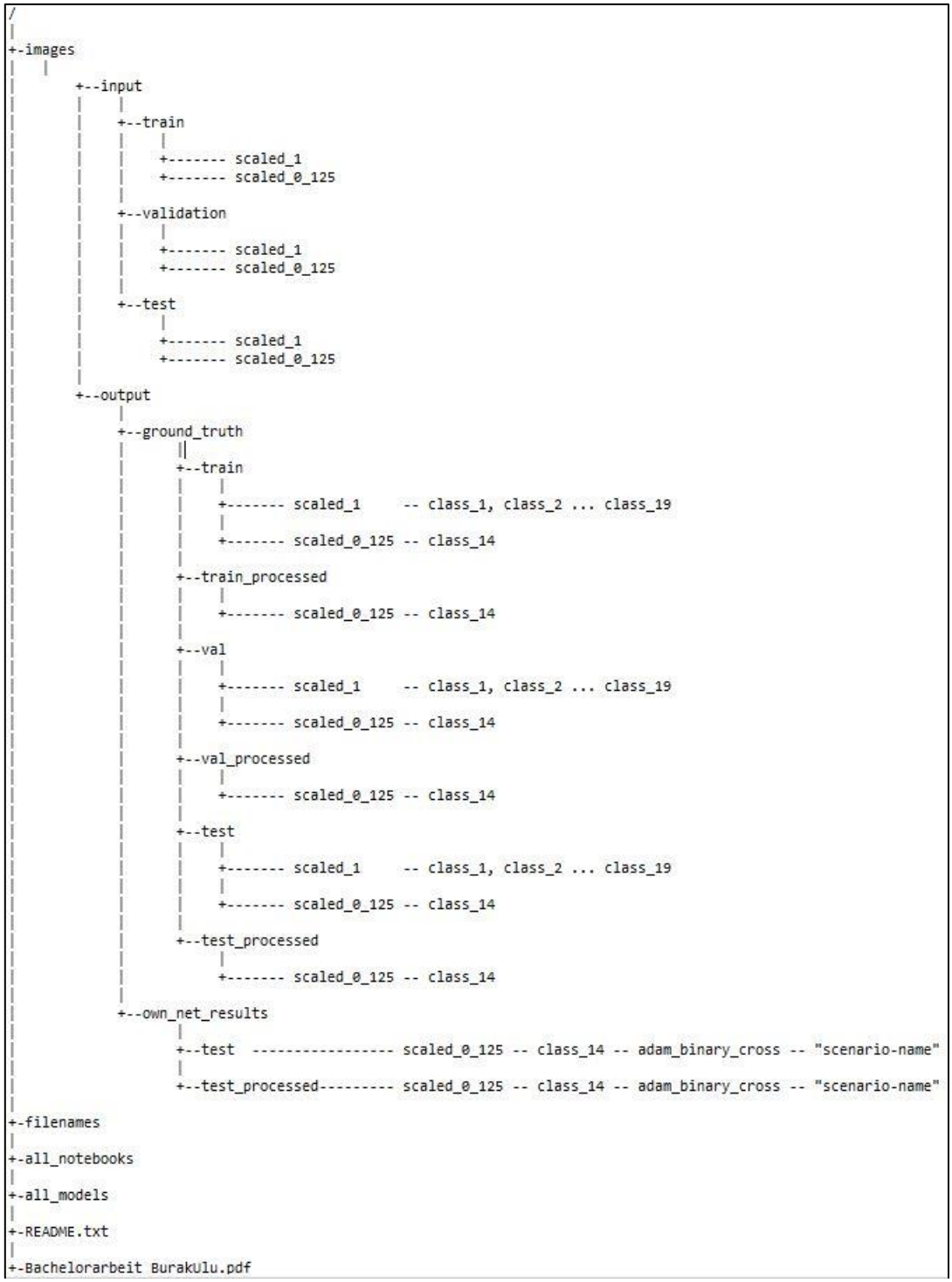


Abbildung A.1 Verzeichnisstruktur der DVD

Literaturverzeichnis

1. **Dohm, M.** digitaleweltmagazin. [Online] [Zitat vom: 10. 12 2019.] https://digitaleweltmagazin.de/2019/06/21/ki-gehoert-die-zukunft-sie-braucht-regeln/.
2. **Gong, Xin-Yi, et al.** An overview of contour detection approaches. [Hrsg.] Springer. *International Journal of Automation and Computing.* 29. 6 2018, Bd. 15, S. 656–672.
3. **Alammar, J.** *http://jalammar.github.io.* [Online] [Zitat vom: 28. 10 2020.] http://jalammar.github.io/visual-numpy/.
4. **Chollet, Francois.** *Deep Learning mit Python und Keras.* s.l. : mitp Verlags GmbH & Co. KG, 2018.
5. **Prof. Dr.-Ing. Meisel, Andreas.** *HAW Hamburg, Mustererkennung und Machine Learning: Kursmaterialien.* 2020.
6. **Gupta, Vikas.** *https://www.learnopencv.com.* [Online] [Zitat vom: 12. 12 2020.] https://www.learnopencv.com/understanding-feedforward-neural-networks/.
7. **Simonyan, Karen und Zisserman, Andrew.** Very Deep Convolutional Networks for Large-Scale Image Recognition. 2015, Bd. abs/1512.03385.
8. **He, Kaiming He, et al.** Deep Residual Learning for Image Recognition. 2015, Bd. abs/1512.03385.
9. **Fukushima, Kunihiko.** A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected By Shift in Position. *Biological Cybernetics.* 1980, 36, S. 193-202.
10. **Lecun, Y., et al.** Gradient-based learning applied to document recognition. 1998, Bd. 86, S. 2278-2324.
11. **Li, Fei F. und Karpathy, Andrej.** *http://cs231n.stanford.edu/.* [Online] [Zitat vom: 10. 12 2020.] https://cs231n.github.io/convolutional-networks/.
12. **Zeiler, Matthew D. und Fergus, Rob.** Visualizing and Understanding Convolutional Networks. 2013, Bd. abs/1311.2901.
13. **Dumoulin, Vincent und Visin, Francesco.** A guide to convolution arithmetic for deep learning. 2018, Bd. abs/1603.07285.

14. **Dertat, Arden.** *https://towardsdatascience.com.* [Online] [Zitat vom: 12. 12 2020.] https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798.
15. **Ruder, Sebastian.** An overview of gradient descent optimization algorithms. Bd. abs/1609.04747.
16. **Ioffe, Sergey und Szegedy, Christian.** *https://arxiv.org.* [Online] Batch Normalization: Accelerating Deep Network Training byReducing Internal Covariate Shift.
17. **Loukadakis, Manolis, Cano, José und O'Boyle, Michael.** Accelerating Deep Neural Networks on LowPower Heterogeneous Architectures. 2018.
18. **Liu, Yahui, et al.** Learning to Refine Object Contours with a Top-Down Fully Convolutional Encoder-Decoder Network. 2017, Bd. abs/1705.04456.
19. **Bertasius, Gedas, Shi, Jianbo und Torresani, Lorenzo.** High-for-Low and Low-for-High: Efficient Boundary Detection from Deep Object Features and its Applications to High-Level Vision. 2015.
20. **Dollár, P. und Zitnick, C. L.** Fast edge detection using structured forests. *PAMI.* 2015.
21. **Yu, Zhiding, et al.** CASENet: Deep Category-Aware Semantic Edge Detection. *CVPR.* 2017.
22. **HAW Hamburg, AI-Labor Informatik, HAW Hamburg AI-Labor.** [Online] https://jupyter.icc.informatik.haw-hamburg.de.
23. **Foundation, Python Software.** *https://www.python.org/.* [Online] https://www.python.org/.
24. **Chollet, Francois.** *https://keras.io/.* [Online] https://keras.io/.
25. ***https://twitter.com.*** **[Online] https://twitter.com/fchollet/status/1113476428249464833.**
26. Acuna, David, Kar, Amlan und Fidler, Sanja. Devil is in the Edges: Learning Semantic Boundaries from Noisy Annotations. *CVPR.* 2019.
27. Max-Planck-Gesellschaft zur Förderung der Wissenschaften e.V. *https://www.cityscapes-dataset.com/.* [Online] [Zitat vom: 05. 06 2020.] https://www.cityscapes-dataset.com.
28. *https://keras.io.* [Online] [Zitat vom: 10. 7 2020.] https://keras.io/api/applications.
29. Migdal, Piotr. *https://github.com/.* [Online] https://github.com/stared/keras-sequential-ascii.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____