



---

# **BACHELORARBEIT**

---

Herr  
Leon D. Wutke

**Serielle Auswertung von Datensätzen  
einer Wildtierkamera mit dem  
Schwerpunkt Kennzeichendetektion**

Mittweida, September 2022



Fakultät Angewandte Computer- und Biowissenschaften

---

# BACHELORARBEIT

---

## **Serielle Auswertung von Datensätzen einer Wildtierkamera mit dem Schwerpunkt Kennzeichendetektion**

Autor:

**Leon D. Wutke**

Studiengang:

Allgemeine und Digitale Forensik

Seminargruppe:

FO19w5

Erstprüfer:

Prof. Dr. rer. nat. Dirk Labudde

Zweitprüfer:

Alexander Kari, Dipl.-Verww. (FH)

Einreichung:

Mittweida, 12.09.2022

Verteidigung/Bewertung:

Mittweida, 2022



Faculty of **Applied Computer Sciences and Biosciences**

---

# **BACHELOR THESIS**

---

## **Serial evaluation of data sets from a wildlife camera with focus on license plate detection**

Author:

**Leon D. Wutke**

Course of Study:

General and digital forensics

Seminar Group:

FO19w5

First Examiner:

Prof. Dr. rer. nat. Dirk Labudde

Second Examiner:

Alexander Kari, Dipl.-Verww. (FH)

Submission:

Mittweida, 12.09.2022

Defense/Evaluation:

Mittweida, 2022



### **Bibliografische Beschreibung:**

Wutke, Leon D.:

Serielle Auswertung von Datensätzen einer Wildtierkamera mit dem Schwerpunkt Kennzeichendetektion. – 2022. – 47 S.

Mittweida, Hochschule Mittweida – University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften, Bachelorarbeit, 2022.

### **Referat:**

Nummernschild- und Objektdetektion sind wichtige Bestandteile der Verkehrsüberwachung. Jedoch befassen sich die meisten Forschungsarbeiten zu diesen Themen mit der Detektion auf gut beleuchteten Bildern und nur wenige mit der Detektion auf sehr schlecht beleuchteten Aufnahmen. Diese Arbeit versucht, verschiedene Objektdetektionsalgorithmen auf schlecht beleuchteten Bildern zu vergleichen, sowie den Einfluss verschiedener einfacher Bildverbesserungsmethoden auf das Detektionsergebnis zu bewerten. Auf Grundlage dieser Ergebnisse wird anschließend eine Anwendung entwickelt, welche automatisiert Datensätze mehrerer Wildtierkameras auswertet, um diese zur einfacheren Verwendung zur Verfügung zu stellen.





# Inhaltsverzeichnis

<b>Inhaltsverzeichnis</b>	<b>I</b>
<b>Abbildungsverzeichnis</b>	<b>III</b>
<b>Tabellenverzeichnis</b>	<b>V</b>
<b>Abkürzungsverzeichnis</b>	<b>VII</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Problemstellung . . . . .	1
1.2 Methodik . . . . .	1
1.3 Verwandte Werke . . . . .	2
<b>2 Methoden und Grundlagen</b>	<b>5</b>
2.1 Datensatz . . . . .	5
2.2 Neuronale Netze . . . . .	7
2.3 Objektdetektionsalgorithmen . . . . .	9
2.3.1 YOLOv5 . . . . .	9
2.3.2 Faster R-CNN . . . . .	11
2.3.3 Single Shot MultiBox Detektor . . . . .	13
2.4 Möglichkeiten der Bildverbesserung . . . . .	15
2.4.1 Bilden des Negativs eines Bildes . . . . .	15
2.4.2 Gamma Korrektur . . . . .	16
2.4.3 Histogramm Bearbeitung . . . . .	17
2.5 Weitere wichtige Berechnungen . . . . .	18
2.6 Grundlagen der Optical Character Recognition . . . . .	19
<b>3 Durchführung</b>	<b>23</b>
3.1 Training . . . . .	23
3.2 Evaluierung . . . . .	24
3.3 Bildverbesserungen . . . . .	26
<b>4 Ergebnisse</b>	<b>29</b>
4.1 YOLOv5 . . . . .	29
4.2 Faster R-CNN . . . . .	31
4.3 SSD . . . . .	32
4.4 Dauer der Bildverbesserungen . . . . .	34
4.5 Direkter Vergleich . . . . .	34
<b>5 Diskussion</b>	<b>35</b>
<b>6 Anwendung</b>	<b>41</b>
6.1 Theorie und Umsetzung . . . . .	41
6.2 Probleme . . . . .	42

<b>7</b>	<b>Zusammenfassung</b>	<b>45</b>
7.1	Fehlerdiskussion . . . . .	45
7.2	Zukünftige Arbeit . . . . .	45
7.3	Fazit . . . . .	46
	<b>Glossar</b>	<b>49</b>
	<b>Literaturverzeichnis</b>	<b>51</b>
	<b>Eidesstattliche Erklärung</b>	<b>55</b>

# Abbildungsverzeichnis

2.1	Aufbau Daten-DVD mit Bildern der Wildtierkameras	5
2.2	Screenshot aus der labeling-Anwendung	6
2.3	Aufbau Coco-Datensatz	6
2.4	Anzahl der Labels	7
2.5	Graphen der Stufen- und Sigmoidfunktion	7
2.6	Aufbau eines Neuronales Netzes	8
2.7	allgemeines YOLO-Model	9
2.8	Aufbau des FPN [25]	10
2.9	Ablauf des R-CNN	11
2.10	Formel zur Berechnung des Objectness-Score	13
2.11	Struktur des SSD	14
2.12	allgemeine Objektive Verlustfunktion von SSD	14
2.13	Formel für smooth $L_1$ Verlust	15
2.14	Formel für die Berechnung des Negativs eines Bildes	15
2.15	Negativ eines Bildes	15
2.16	Formel für die Power Law Transformation	16
2.17	Beispiel Gamma Korrektion	16
2.18	Piecewise linear transformation graph	16
2.19	Originalbild und zugehöriges Histogramm	17
2.20	Ergebnis der Histogram Equalization und zugehöriges Histogramm	17
2.21	Ergebnis der Lokalen Histogram Equalization	18
2.22	Ergebnis von CLAHE	18
2.23	Zeichensatz der FE-Schrift	20
3.1	Precision-Recall Kurve des YOLOv5 mit 25 Trainingsepochen	25
5.1	Durch CLAHE „zerstörtes“ Nummernschild	35
5.2	CLAHE mit „clipLimit“ von 20	38
5.3	Erkannte Besonderheiten durch die verschiedenen Algorithmen	39
5.4	Aufgenommenes Polizeiauto	39
5.5	Objekte auf einem LKW erkannt durch YOLOv5	40
6.1	Graphische Oberfläche der Anwendung	41
6.2	fälschlicherweise erkannte Nummernschilder	43



# Tabellenverzeichnis

2.1	Konfusionsmatrix	19
4.1	mAP des YOLOv5 auf dem Validierungs-Datensatz	29
4.2	mAP des YOLOv5 trainiert und validiert auf verbesserten Bildern	29
4.3	mAP@0.5:0.95 des YOLOv5 im Vergleich	30
4.4	mAP des YOLOv5 unter Verwendung unterschiedlicher YOLOv5 Modelle	30
4.5	mAP des Faster R-CNN auf dem Validierungs-Datensatz	31
4.6	mAP des Faster R-CNN trainiert und validiert auf verbesserten Bildern	31
4.7	mAP@0.5:0.95 des Faster R-CNN im Vergleich	32
4.8	mAP des SSD300 auf dem Validierungs-Datensatz	32
4.9	mAP des SSD300 trainiert und validiert auf verbesserten Bildern	33
4.10	mAP des SSD300 im Vergleich	33
4.11	Dauer der einzelnen Bildverarbeitungen pro Bild	34
4.12	mAP aller Algorithmen	34



# Abkürzungsverzeichnis

<b>AP</b> .....	Average Precision
<b>BBR</b> .....	Bounding Box Regression
<b>CNN</b> .....	Convolutional Neuronal Network
<b>CTC</b> .....	Connectionist Temporal Classification
<b>DPI</b> .....	Dots per Inch
<b>DVD</b> .....	Digital Video Disk
<b>FPN</b> .....	Feature Pyramid Network
<b>FPS</b> .....	Frames per second
<b>IoU</b> .....	Intersection over Union
<b>KPI</b> .....	Kriminalpolizeiinspektion
<b>LKW</b> .....	Lastkraftwagen
<b>LSTM</b> .....	Long Short-Term Memory
<b>mAP</b> .....	mean Average Precision
<b>NMS</b> .....	Non Maximum Supression
<b>OCR</b> .....	Optical Character Recognition
<b>PANet</b> .....	Path Aggregation Network
<b>R-CNN</b> .....	Region Convoluted Neuronal Network
<b>RoI</b> .....	Region of Interest
<b>RPN</b> .....	Region Proposal Network
<b>SGD</b> .....	Stochastic Gradient Descent
<b>SOKO</b> .....	Sonderkomission
<b>SPP</b> .....	Spatial Pyramid Pooling
<b>SSD</b> .....	Single Shot Multibox Detector
<b>SVM</b> .....	Support Vector Machine
<b>VGG</b> .....	Visual Geometry Group
<b>VOC</b> .....	Visual Object Classes
<b>YOLO</b> .....	You Only Look Once
<b>z.B.</b> .....	zum Beispiel





# 1 Einleitung

## 1.1 Problemstellung

Mit der steigenden Zahl versendeter Pakete steigt ein Problem: das Aufschlitzen der Planen über LKWs und Kleintransportern und das anschließende Stehlen der transportierten, wertvolleren und weiterverkaufbaren Fracht des Transporters. Da dies zumeist nachts auf Rastplätzen an der Autobahn geschieht, wo alle anderen anwesenden Menschen ebenfalls Fahrer solcher Transporter sind, die ihre Nachtruhe halten, ist es schwierig an Augenzeugen dieser Diebstähle zu kommen. Andere Beweise und Hinweise auf den Täter gibt es meist nicht. Um das zu umgehen, hat die [SOKO KFZ](#) der [KPI](#) Chemnitz an Eingang und Ausgang mehrerer Rastplätze Wildtierkameras aufgehängt, in der Hoffnung, dass diese das Fahrzeug sowie das Nummernschild des oder der Täter aufnehmen.

Auf diese Art und Weise kommen jedoch selbst bei weniger stark befahrenen Rastplätzen auf 3.500 oder mehr Bilder zusammen. Da die Auswertung dieser einen Sachbearbeiter, der sich auch mit sinnvollerer Tätigkeiten befassen könnte, über mehrere Tage beschäftigen kann, ist eine automatisierte Auswertung dieser Bilder notwendig. Dabei muss jedoch das System zur Erkennung der Nummernschilder eine größtmögliche Genauigkeit erreichen, damit die gewonnenen Daten sinnvoll verarbeitbar sind. Da dies mit aktuellen Mitteln nur schwer erreicht werden kann, befasst sich diese Arbeit mit dem Vergleich verschiedener Algorithmen für die Lokalisierung der Nummernschilder und der Identifizierung von Fahrzeugen, hinsichtlich Performance und Genauigkeit, sowie dem Versuch die Genauigkeit durch Verbesserungen des Bildes zu erhöhen. Auf Grundlage dieser Ergebnisse ist anschließend eine Applikation zu entwickeln, welche mit einer grafischen Oberfläche die Eingabe der Bilder ermöglicht, diese automatisch auswertet und die Ergebnisse anschließend in einem gängigen Format wie `xlsx` oder `csv` speichern kann.

Nummernschilder die dabei auftreten stammen hauptsächlich aus Deutschland, Polen und Tschechien. Andere Nummernschilder können auch vorkommen, sind jedoch wesentlich unwahrscheinlicher. Die Fahrzeuge sind zumeist LKWs; Kleintransporter sind die relevantesten Fahrzeuge, die aufgenommen werden können.

Nummernschilder auf Bildern, die in dieser Arbeit gezeigt werden, sind unkenntlich gemacht.

## 1.2 Methodik

Für diese Arbeit wurden zunächst verwandte Arbeiten zu dem Thema „Nummernschilderkennung“ studiert und Objektdetektionsalgorithmen mit sinnvoller Genauigkeit und Laufzeit untersucht. Drei der häufigsten und besten Algorithmen sind [YOLOv5](#), [SSD](#) und [Faster R-CNN](#). [Faster R-CNN](#) wird dabei nie für Nummernschilderkennung genutzt, da dieser Algorithmus nicht schnell genug ist, um in Echt-Zeit zu laufen. Da das Ziel dieser Arbeit allerdings keine in Echt-Zeit laufende Anwendung ist, wird [Faster R-CNN](#) in die Betrachtungen einbezogen.

Diese drei wurden mit einem speziell angelegten Datensatz von Bildern einer Tatnacht trainiert und evaluiert. Anschließend wurden diese hinsichtlich ihrer [mAP](#), sowie ihrer Trainings- und Detektionszeit verglichen. Im Anschluss daran wurden verschiedene Bildbearbeitungsmethoden angewendet und die drei Algorithmen auf den bearbeiteten Bildern trainiert und evaluiert. Der effektivste dieser Algorithmen wurde anschließend für die zu entwickelnde Anwendung verwendet.

Für die Anwendung wird anschließend ein System implementiert, welches mithilfe des besten Objektdetektionsalgorithmus alle Objekte im Bild erkennt und Nummernschilder an die [OCR](#)-Anwendung „Tesseract“ weitergibt. Die Anwendung speichert anschließend alle Ergebnisse in einer csv-Datei, um diese nachfolgend in das gewünschte Format einer Excel-Datei zu überführen.

### 1.3 Verwandte Werke

Für die Erkennung von Nummernschildern gibt es verschiedene Ansätze. Die einfachste Methode ist das Nutzen und Erkennen der rechteckigen Form von Nummernschildern. Besitzen die Nummernschilder allerdings ein anderes Seitenverhältnis oder sind überdeckt und damit nicht mehr rechteckig, werden diese nicht mehr erkannt. Befinden sich auf dem Bild andere Rechtecke mit passendem Seitenverhältnis, können diese fälschlicherweise als Nummernschild erkannt werden. Weitere Methoden basieren auf Farb- und Textureigenschaften, welche jedoch aufgrund dessen, dass alle gegebenen Bilder in Graustufen und bei Nacht aufgenommen sind, nicht anwendbar sind. Möglich wäre auch das Interpretieren aller Regionen mit Buchstaben und Zahlen als Nummernschild, jedoch werden damit auch Schriftzüge und Logos auf Fahrzeugen als Nummernschild interpretiert. Diese Methoden lassen sich auch kombinieren, jedoch werden dadurch nicht alle Schwächen eliminiert. [1] Zudem lassen sich mit diesen Verfahren keine Fahrzeugtypen unterscheiden.

Eine gut funktionierende Alternative zum klassischen Ansatz ist das Verwenden von Objektdetektionsalgorithmen zur Lokalisierung von Nummernschildern. Bereits vorhandene Forschungsarbeiten zu ähnlichen Themen befassen sich meist mit Verkehrskontrollen und der Analyse des Verkehrsaufkommens. Dabei werden verschiedene Algorithmen und Methoden angewendet. So gibt es Möglichkeiten, herannahende Fahrzeuge durch minimale Veränderungen in der Beleuchtung der Umgebung festzustellen [2]. Dafür genutzt werden keine von vornherein festgelegte Objekt-Boxen, sondern sogenannte „KeyPoints“. Dies dient allerdings nur dazu, sich frontal nähernde Fahrzeuge für Fahrassistenzsysteme zu erkennen. Nummernschilder und der Fahrzeugtyp werden dabei nicht unterschieden. Andere Arbeiten verwenden meist Farbbilder zur Erkennung von Nummernschildern. Dabei wird erst nachdem das Nummernschild erkannt wurde, der Ausschnitt mit dem Nummernschild in ein Graustufenbild umgewandelt.[3]

Insgesamt beschäftigen sich jedoch nur wenige Arbeiten mit dem Erkennen von Fahrzeugen oder gar Nummernschildern unter bescheidenen Beleuchtungszuständen. Und auch nur wenige Arbeiten implementieren Bildverbesserungsmethoden für diese Situationen. Eine Arbeit evaluiert verschiedene Bildverbesserungsmethoden für Navigation auf visueller Basis. In dieser werden gute Ergebnisse mit den Verbesserungsmöglichkeiten CLAHE und Gamma Korrektion erzielt. [4]

Für Objekt Detektion unter schlechter Beleuchtung haben Yuxuan Xiao *et al.* einen speziellen Objektdetektor für entsprechende Bilder vorgestellt. Dafür haben sie die Herausforderungen der Objektdetektion unter schlechter Beleuchtung ausführlich untersucht und den Einfluss von Datensets, die verschiedenen beleuchtete Bilder beinhalten, bewertet. Zudem bearbeiteten sie die Anwendbarkeit von Algorithmen, welche die Beleuchtung verbessern. [5]

Eine ähnliche Arbeit stammt von Yirui Wu *et al.* Diese beschäftigen sich mit der Möglichkeit, die Bildverbesserung schlecht beleuchteter Bilder auf einen Server auszulagern, da mobile Geräte oftmals nicht die Leistung dafür haben. Das von ihnen vorgestellte Netzwerk zur Beleuchtungsverbesserung führt zu wesentlich verbesserten Ergebnissen auf dem Datensatz mit schlecht beleuchteten Bildern. [6]

Dem finden der Nummernschilder auf den Bildern folgt die Erkennung des Textes. Diesem können diverse Vorbearbeitungsschritte zur Verbesserung der Bildqualität und damit der Erkennungsgenauigkeit vorgeschaltet werden. Mit dem in [7] vorgestelltem Algorithmus wurden sehr gute Ergebnisse, sowohl für das finden der Nummernschilder als auch für das Erkennen des Textes auf diesen, erzielt. Für die Identifizierung der einzelnen Zeichen werden dabei unterschiedliche Features genutzt, darunter die Anzahl an Löchern (bei einer 8 wären es z.B. 2), Endpunkten und Stellen, an denen mehrere Linien der Zeichen aufeinander treffen. Zusätzlich werden die semantische Struktur von Nummernschildern zur Unterscheidung von numerischen und alphabetischen Zeichen verwendet. [7]

Eine Arbeit implementiert die Erkennung von Nummernschildern unter Verwendung von YOLOv5. In dieser wird die Erkennung der Zeichen auf den Nummernschildern über die OCR-Bibliothek EasyOCR realisiert. [8] In anderen Arbeiten wird statt YOLOv5 der SSD Algorithmus mit einem MobileNet-Backbone zur Nummernschilderkennung genutzt. [9, 10] Die Arbeit [10] evaluiert und vergleicht dabei die Genauigkeit der OCR-Anwendungen Tesseract und EasyOCR. SSD eignet sich zusätzlich für die Segmentierung und Klassifizierung der Zeichen auf einem Nummernschild und erreicht dabei sehr gute Ergebnisse. [11].

Die Arbeiten zur Nummernschilderkennung nutzen dabei zumeist viele Bilder, die unter Optimalbedingungen aufgenommen wurden. Die Kennzeichen sind gut ausgeleuchtet und vollständig zu sehen. Unter reellen Bedingungen wären deshalb die beschriebenen Ergebnisse vermutlich etwas schlechter als die in den Arbeiten ermittelten Werte.

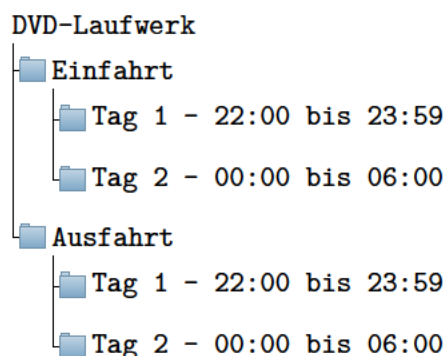


## 2 Methoden und Grundlagen

### 2.1 Datensatz

Der Datensatz, für sowohl Training als auch Validierung, stammt aus Aufnahmen zweier Wildtierkameras. Diese befanden sich an Ein- und Ausfahrt eines Rastplatzes an der Autobahn. Die Aufnahmen stammen dabei, aufgrund rechtlicher Beschränkungen, aus dem Zeitraum von 22:00 bis 06:00 Uhr am nächsten Morgen der Tatnacht.

Diese lagen, wie bei der Polizei üblich, auf einer Daten-DVD. Deren Aufbau kann man in [Abbildung 2.1](#) sehen. In dieser Form werden der entwickelten Applikation zukünftig auch die Bilder übergeben.



**Abbildung 2.1:** Aufbau Daten-DVD mit Bildern der Wildtierkameras

Vor der Weiterverarbeitung musste die Ordnerstruktur verändert werden. Sie wurden alle in den gleichen Ordner kopiert und aufgrund der Tatsache, dass es zwei Kameras gibt und so Bilder mit gleichen Namen auftreten können, dabei mit einer fortlaufenden Nummer umbenannt. Auf diese Weise kamen 3.468 Dateien zusammen, 1.704 für die Einfahrt und 1.764 für die Ausfahrt.

Anschließend wurde das Pythonskript 'labelimg' [12] verwendet, um auf diesen Bildern Objekte manuell zu klassifizieren. Ein Screenshot aus dieser Anwendung ist in [Abbildung 2.2](#) zu sehen. Dabei wird eine rechteckige Box um alle zu erkennenden Objekte auf diesem Bild gezogen. In dem hier gezeigten Fall sind die zu erkennenden Objekte ein Nummernschild, ein PKW, ein LKW, eine Zugmaschine und der zugehörige Anhänger des LKWs. Die unterschiedlichen Klassen sind dabei verschiedenfarbig. Von den gezogenen Boxen können die Eckpunkte in verschiedenen Formaten exportiert werden. So gibt es XML-Dateien für Pascal-VOC, JSON-Dateien für Create-ML und, die im Rahmen dieser Arbeit genutzt, txt-Dateien für [You Only Look Once \(YOLO\)](#). Diese wurden genutzt, da diese den geringsten Speicherbedarf haben, mit ihnen der [YOLOv5](#) direkt trainiert werden kann und sie sehr einfach in andere Modelle implementiert werden können.

Die Textdateien enthalten für jeden im zugehörigen Bild erstellten Begrenzungsrahmen eine Zeile. Diese beginnt mit der Klassen-ID der entsprechenden Klasse, gefolgt von den X- und Y-Koordinaten des Mittelpunktes, sowie der Breite und Höhe der Box. Getrennt sind diese Einträge durch Leerzeichen. Diese Werte sind so umgerechnet, dass sie unabhängig vom Seitenverhältnisses des Bildes sind. Zusätzlich zu einer Textdatei für jedes Bild existiert eine Textdatei, welche die existierenden Klassen enthält; die Klassen-ID entspricht dabei der Zeilennummer.

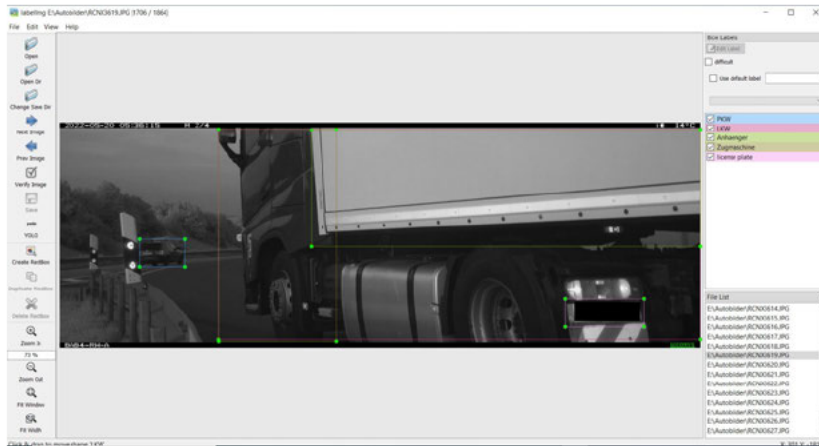


Abbildung 2.2: Screenshot aus der labeling-Anwendung

Nachdem alle Bilder auf diese Art und Weise vorbereitet wurden, wurde die Struktur des Datensatzes in Coco-Struktur gebracht. Allerdings wurden nicht alle Bilder übernommen, da auf einigen Bildern keine Objekte zu erkennen waren oder in mehreren Bildern keine Veränderungen feststellbar waren. Den Aufbau des Coco-Datensatzes kann man in Abbildung 2.3 sehen. Dabei wurden die Bilder bereits in Trainings- und Validierungsdaten geteilt, in einem Verhältnis 70 zu 30. Zu jeder verwendeten Bildverbesserung wurde aus diesem Datensatz ein Datensatz mit bearbeiteten Bildern erstellt. Die Datei *Autodaten.yaml* dient dabei der Konfiguration des YOLOv5-Algorithmus.

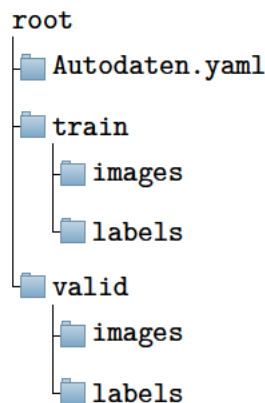


Abbildung 2.3: Aufbau Coco-Datensatz

So kamen im Trainingsdatensatz 1.969 Bilder und 844 Dateien im Validierungsdatensatz zusammen. Die gesamte Verteilung der Anzahl an Klassen ist in Abbildung 2.4 zu sehen. Die meisten Objekte existieren von der Klasse „LKW“, dicht gefolgt von der Klasse „license plate“. Die wenigsten Labels existieren für die Klassen „Wohnwagen“ und „PKW Anhaenger“. Die Klasse „Besonderheit“ stellt den Versuch dar, Besonderheiten auf Fahrzeugen zu erkennen, wie z.B. Logos oder Schriftzüge. Die Klasse „Anhaenger“ steht für Anhänger von LKWs, während „PKW Anhaenger“ für Anhänger an PKWs steht.

Sämtliche bei Nacht aufgenommenen Bilder sind grau, lediglich wenn es vor 06:00 Uhr hell genug war, liegen Bilder in Farbe vor. Aufgrund dieser Verteilung wurden alle Bilder in Graustufenbilder überführt. Die Bilder sind dabei immer gleich aufgebaut. Am oberen und unteren Rand befinden sich

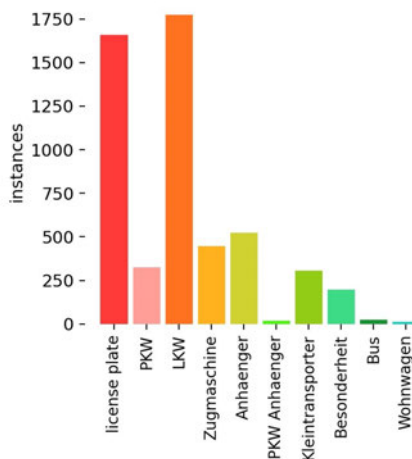


Abbildung 2.4: Anzahl der Labels

schwarze Streifen. Links oben steht der Zeitstempel, gefolgt von der Nummer der Aufnahme. Rechts oben steht die aktuelle Temperatur. Unten links steht der Name der Kamera. Die Größe der Bilder ist immer 2048x720 Pixel.

## 2.2 Neuronale Netze und CNN

Die nachfolgend beschriebenen Algorithmen beruhen alle auf neuronalen Netzen. Dieses Unterkapitel befasst sich näher mit diesen.

Ein neuronales Netz ist ein Zusammenschluss mehrerer Perceptronen. Ein Perceptron ist eine Art künstliches Neuron. Es hat  $n$  Eingänge und eine beliebige Zahl an Ausgängen. Die Ausgabe eines Perceptrons berechnet sich mit der Summe aller Eingänge in einer Schwellwertfunktion. Diese kann variabel festgelegt werden, zwei mögliche sind die Stufenfunktion und die Sigmoid-Funktion, deren Graphen in [Abbildung 2.5](#) gezeigt sind. [13]

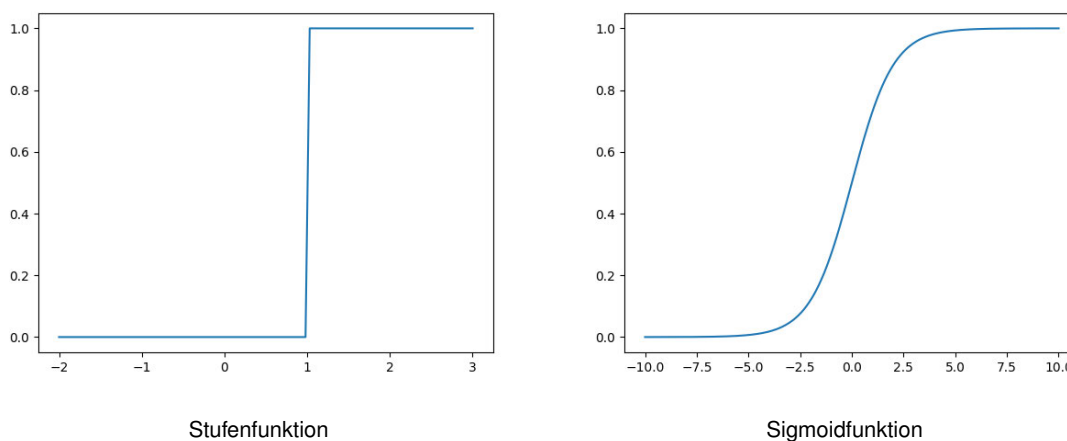
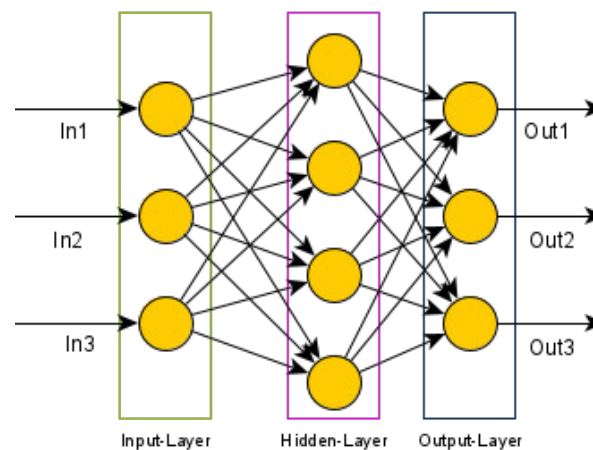


Abbildung 2.5: Graphen der Stufen- und Sigmoidfunktion



In einem neuronalen Netz sind mehrere Schichten von Perceptronen verbunden (Abbildung 2.6). Es hat dabei immer die Eingangs- und Ausgangsschicht; optional sind eine beliebige Anzahl dazwischen geschalteter versteckter Schichten. Die Eingangsschicht dient der Eingabe von Daten ins Netz, die Ausgangsschicht dient der Ausgabe des Ergebnisses. Die Verbindungen zwischen den Perceptronen sind mit Gewichten versehen. Während des Lernprozesses werden diese angepasst, wodurch der letztendliche Lerneffekt entsteht. Die Anpassung erfolgt auf Grundlage des berechneten Ausgabefelders für jeden Durchgang. Der Fehler wird entsprechend einer Fehlerfunktion berechnet; um ein Optimum des Netzes zu bekommen, ist die Bestimmung des Minimums der Fehlerfunktion notwendig. Da die Ermittlung des Fehlers für einzelne Knoten in einem komplexen Netzwerk sehr schwierig wird und die Funktion so komplex ist, dass ihr Minimum sich nicht durch einfache Algebra ermitteln lässt, wird das Minimum in einem Annäherungsverfahren bestimmt. Dieses Verfahren wird Gradientenverfahren genannt. Bei diesem wird die Position auf dem zur Fehlerfunktion gehörigen Graphen entsprechend des Anstieges angepasst. Da bei diesem Verfahren die Gefahr besteht, sich in einem lokalen Minimum zu verfangen, muss die Größe der Schritte, die auf der Kurve gemacht werden, richtig gewählt werden. Die Größe der Schritte wird auch als **Lernrate** bezeichnet. [13, 14]

Das einfache Gradientenverfahren berechnet dabei in jeder Trainingsepoche zunächst den Verlust über den kompletten Trainingsdatensatz und passt anschließend die Gewichte an. Dadurch wird ein Gerät benötigt, welches über ausreichend Speicher verfügt, um den gesamten Datensatz zu laden. Zudem ist es dadurch sehr langsam und ein fortgesetztes Training mit weiteren Trainingsdaten ist nicht möglich. Dafür ist garantiert, dass auf konvexen Oberflächen der Fehlerfunktion immer das globale Minimum erreicht wird. Alternativ dazu berechnet **SGD** jede Epoche den Fehler auf jedes Bild und passt somit die Gewichte nach jedem Bild an. Dadurch führt es keine redundanten Berechnungen durch und wird wesentlich schneller. Auch eine Fortsetzung des Trainings mit neuen Daten ist einfach möglich. Allerdings kompliziert die Schwankung des **SGD** die Konvergenz zum gewünschten Minimum. Umgangen werden kann das durch eine langsame Verringerung der **Lernrate**, wodurch ein ähnliches Konvergenzverhalten wie beim einfachen Gradientenverfahren erhalten werden kann. [15]



**Abbildung 2.6:** Aufbau eines Neuronalen Netzes

Bei einem **Convolutional Neuronal Network (CNN)** wird die Anzahl der zu speichernden Parameter verringert, indem ein Pixel/Perceptron des folgenden Layers lediglich einen Bereich der vorherigen Schicht als Input bekommt, im Gegensatz zu dem in Abbildung 2.6 gezeigten vollständig verbundenen neuronalem Netz. Erreicht wird das über die sogenannte „Convolution Operation“, bei der



ein Ausschnitt des Bildes mit einer Matrix lernbarer Parameter, auch Kernel oder Filter genannt, ein Skalarprodukt bildet. Der Kernel ist kleiner als das Eingabebild, besitzt jedoch die gleiche Tiefe. Anschließend wird der Filter um einen Wert, der „Stride“ genannt wird, auf dem Eingabebild verschoben, zur Ermittlung des nächsten Pixels. Das Ergebnis der „Convolutional Operation“ dient als Eingabe für die nächste Schicht im Netzwerk. Zusätzlich eignet sich ein CNN gut zur Featureextraktion, da die Faltung der „Convolutional-Layer“ dafür sorgt, dass die ersten Layer nur kleine Bereiche der zu detektierenden Objekte zeigen und um so tiefer die Schicht liegt, desto komplexer werden die Features, die der Layer erkennt. Dadurch entsteht in jeder Schicht eine Featuremap mit unterschiedlich skalierten Merkmalen der Objekte. [16, 17]

## 2.3 Objektdetektionsalgorithmen

Objektdetektionsalgorithmen sind Algorithmen, welche der Lokalisierung und Klassifizierung von Objekten auf Bildern oder in Videos dienen. Diese arbeiten meist auf Grundlage neuronaler Netze.

### 2.3.1 YOLOv5 - You only look once v5

YOLOv5 ist ein Algorithmus zur Objectdetektion auf Bildern. YOLO steht dabei für „You Only Look Once“ und v5 bezeichnet die Version 5 des Algorithmus.

Die erste Version von YOLO wurde von Joseph Redmon *et al.* 2016 als Alternative zu anderen Objektdetektionalgorithmen wie Faster-RCNN vorgestellt. Ziel dieser Arbeit war das Erstellen eines Algorithmus der zuverlässig Objekte in Real-Time detektieren kann. Dies wird erreicht, indem das Bild in ein  $S \times S$  Gitter geteilt wird, und jede Zelle dieses Gitters eine bestimmte Anzahl möglicher Boxen für das Objekt in dieser Zelle und einen Wahrscheinlichkeitswert für diese vorhersagt. Fällt der Mittelpunkt eines Objektes auf eine Zelle, ist diese verantwortlich für die Detektion dieses Objektes. Zusätzlich sagt jede Zelle noch die vermutliche Klasse dieser Zelle vorher, zusammen mit einer Wahrscheinlichkeit für diese. Dadurch entsteht eine Klassenwahrscheinlichkeitskarte.

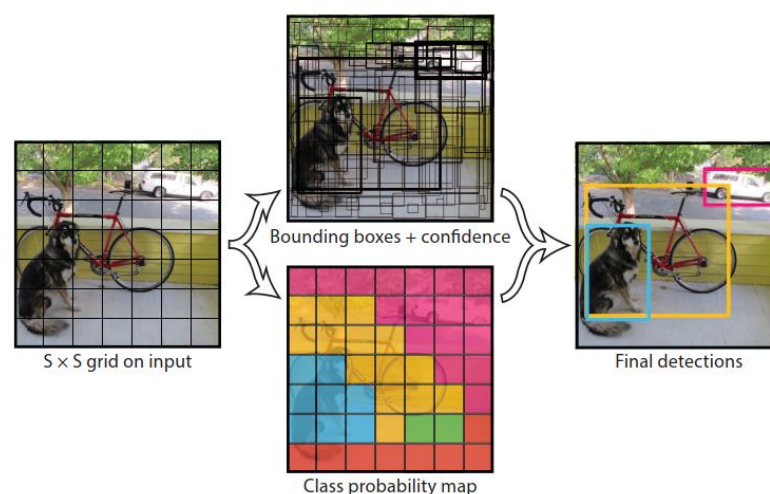


Abbildung 2.7: allgemeines YOLO-Modell [18]

Aus einer Kombination der möglichen Boxen und derer Wahrscheinlichkeiten, sowie der Klassenwahrscheinlichkeitskarte werden die endgültigen Boxen der Objekte berechnet, zu sehen in Abbildung 2.7. Bestandteil dieser Berechnung ist die **Non Maximum Supression**. Aufgrund dieses Aufbaus ist **YOLO** ein Single-Stage Objekt Detektor, was bedeutet, der Algorithmus berechnet in einem Schritt die Begrenzungsrahmen für Objekte und deren zugehörige Klassen. Der ursprüngliche Algorithmus nutzt ein **CNN** mit 24 „convolutional Layern“, gefolgt von zwei vollständig verbundenen Schichten. Dabei werden die Input-Bilder auf eine bestimmte Größe skaliert, für **YOLO** auf 448x448. **YOLO** übertrifft den, in der ursprünglichen Arbeit zum Vergleich herangezogenen, Algorithmus DPM 30Hz um 15 **FPS**, Fast **YOLO** übertrifft den 100HZ DPM um sogar 55 **FPS**. **YOLO** hat dabei auf dem VOC 2012 Leaderboard eine **mAP** von 57,9, wird dabei jedoch von anderen Algorithmen, wie dem **Faster R-CNN**, vorgestellt in Kapitel 2.3.2, übertroffen. Die größte Schwäche von **YOLO** ist das Erkennen kleiner Objekte. [18]

Über die Jahre danach wurden diverse Verbesserungen für **YOLO** veröffentlicht, die Versionen **YOLOv2** und **YOLOv3** ebenfalls von Joseph Redmon und **YOLOv4** von Alexey Bochkovskiy *et al.* [19]. Die Version **YOLOv5** wurde ohne Paper auf Github von ultralytics veröffentlicht. Die hauptsächliche Entwicklung ist die Verbesserung der Performance und die bessere Detektion kleiner Objekte. Dafür wurden in den Generationen die Strukturen der genutzten Netze geändert und diverse Zwischenschritte hinzugefügt. **YOLOv5** ist lediglich eine minimal veränderte Version des **YOLOv3** und weniger fortschrittlich als **YOLOv4**. [20]

Da zu **YOLOv5** kein wissenschaftliches Paper existiert, stammen alle nachfolgenden Informationen zum Aufbau des **YOLOv5** aus dem Artikel [21]: **YOLOv5** besitzt, wie fast jeder Objekt Detektor folgende drei Elemente [19]:

1. „Model Backbone“
2. „Model Neck“
3. „Model Head“

Der „Model Backbone“ dient der Featureextraktion, genutzt wird von **YOLOv5** dafür eine CSP-Darknet53-Struktur [22]. Diese übertrifft, aufgrund des CSPNet-Aufbaus, in ihrer Geschwindigkeit, die normale Darknet53-Struktur. Die CSPNet-Struktur sorgt dafür, dass die Eingabe in 2 Teile geteilt, nur eine Hälfte durch das Netzwerk geleitet wird und beide Teile anschließend wieder zusammengeführt werden. Dadurch wird die Anzahl benötigter Berechnungen stark verringert, bei gleichbleibender Genauigkeit.[23] Darknet53 wurde für **YOLOv3** eingeführt und besitzt 53 „Convolutional Layer“. Diese Struktur sorgt für bessere Ergebnisse als das für **YOLOv2** genutzte Darknet19 und ist dabei immer noch effizienter als verschiedene ResNet-Strukturen. [24] Über dem Darknet53 wird ein **SPP** Block eingefügt. Dieser separiert die stärksten Features ohne die Rechenzeit des Modells zu erhöhen. [19]

Der „Model Neck“ dient der Identifizierung verschieden groß skaliertes Objekte, z.B. durch Erstellung von Feature-Pyramiden. **YOLOv5** nutzt für diesen den **PANet**. Dieser übertrifft (stand 2018) fast alle anderen Algorithmen mit vergleichsweise wenigen Trainingsdaten. **PANet** startet mit einem **FPN**. Das **FPN** besteht aus einem „bottom-up“ und einem „top-down“ Pfad. Der „bottom-up“ Pfad ist ein einfaches **CNN**, welches verschieden große Featuremaps erstellt. Schichten in diesem, die dabei Featuremaps gleicher Größe erzeugen, werden in einer

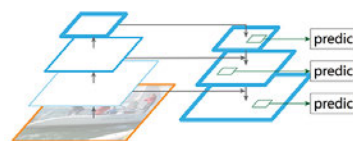


Abbildung 2.8: Aufbau des **FPN** [25]

„Network Stage“ oder einem „Pyramid Level“ zusammengefasst. Der „top-down“-Pfad vergrößert die Auflösung der kleinsten Featuremap, dabei werden die Featuremaps der letzten Schichten des entsprechenden „Pyramid Levels“ zur Anreicherung genutzt, wie in Abbildung 2.8 gezeigt. [25] Dem FPN folgt ein weiterer „bottom-up“ Pfad. Dieser ist wieder ein einfacher CNN. Aus diesem werden letztendlich die Features extrahiert. [26]

Der „Model Head“ dient der letztendlichen Vorhersage der vorhandenen Klassen, sowie deren zugehörigen Boxen. In YOLOv5 entspricht dieser dem von YOLOv3 und YOLOv4 genutzten [22, 19]. Der „Head“ des YOLOv3 besteht aus drei Detektoren, welche auf verschiedenen groß skalierten Featuremaps arbeiten. Dadurch wird die Detektion verschieden großer Objekte ermöglicht. Die Vergrößerung der Schichten zwischen den Detektoren erfolgt unter Anreicherungen durch vorherige Layer. [24]

Wie bei allen neuronalen Netzen ist auch bei YOLOv5 die Verwendung der richtigen Aktivierungsfunktion wichtig. Die Wahl der YOLOv5-Autoren fiel für die versteckten Schichten auf Leaky ReLu und für die finale Detektionsschicht auf die Sigmoid-Funktion. Bei Leaky ReLu wird der Ausgabewert für einen Eingabewert  $x$  kleiner Null über  $f(x) = \alpha * x$  berechnet, wobei  $\alpha$  ein festgelegter Wert ist. Für Eingabewerte größer oder gleich Null wird die Ausgabe über  $f(x) = x$  berechnet. [27] Den Graphen der Sigmoid-Funktion kann man in Abbildung 2.5 sehen.

YOLOv5 bietet zwei Varianten für die Optimierungsfunktion: SGD und Adam; SGD ist als Standard eingestellt. Die Verlustfunktion ist die von Pytorch implementierte „Binary Cross-Entropy with Logits Loss“-Funktion <sup>1</sup>.

### 2.3.2 Faster R-CNN

R-CNN steht für **Region Convoluted Neuronal Network**. Der Algorithmus wurde das erste mal 2014 von Ross Girshick *et al.* vorgestellt [28]. Es ist im Gegensatz zu YOLO ein mehrstufiger Ansatz, was bedeutet, Regionsvorschläge und zugehörige Klassen werden in zwei Schritten berechnet.

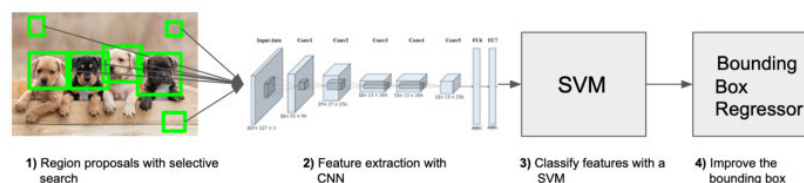


Abbildung 2.9: Ablauf des R-CNN [29]

In Abbildung 2.9 ist der Ablauf des R-CNN gezeigt. Er startet mit einem „Region Proposal“-Algorithmus. Dies ist ein Algorithmus der Regionen vorschlägt, die es lohnt näher anzuschauen, da sie möglicherweise die zu detektierenden Objekte beinhalten. Der Algorithmus dafür kann frei gewählt werden, dabei beeinflusst der gewählte Algorithmus das endgültige Detektionsergebnis. In Girshicks Paper wurde lediglich „selective search“ <sup>2</sup> genutzt. Anschließend werden aus den vorgeschlagenen Regionen, mithilfe eines CNN, die Features extrahiert. Dabei können bis zu 94% der Parameter des CNN entfernt werden, ohne große Verluste in der Detektionsgenauigkeit zu erhalten. Die extrahierten

<sup>1</sup><https://pytorch.org/docs/master/generated/torch.nn.BCEWithLogitsLoss.html>

<sup>2</sup><https://learnopencv.com/selective-search-for-object-detection-cpp-python/>

Features werden anschließend für jede vorgeschlagene Region von einer linearen SVM klassifiziert. Durch Verwendung einer anschließenden „Bounding Box Regression“, lässt sich die mAP um 3.5 verbessern. [28]

Die BBR wird genutzt, da R-CNN die vorgeschlagenen Regionen des „Region Proposal“-Algorithmus, insofern diese ein zu detektierendes Objekt beinhalten, als Begrenzungsrahmen dieses Objektes annimmt. Diese müssen jedoch nicht immer perfekt sein, so können diese auch nur einen Teil des Objektes beinhalten. Die BBR ist ein Algorithmus der diese Boxen optimiert. Eine Möglichkeit für einen solchen ist der von Seungkwon Lee *et al.* beschriebene „Universal BBR“, welcher, auf Basis einer CNN-Struktur, lernt wie die vorgeschlagenen Boxen verbessert werden können. [30]

2015 veröffentlicht Girshick eine Weiterentwicklung des R-CNN, und zwar den Fast R-CNN. Dabei ist das Hauptziel die Verbesserung der drei größten Schwächen des R-CNN:

1. R-CNN nutzt beim Training mehrere Etappen, was das Training verlangsamt
2. Das Training ist sehr Zeit- und Speicherintensiv
3. Die Detektion mit R-CNN ist nicht sonderlich schnell

Die hauptsächlichen Veränderungen, die dies verbessern, sind das Hinzufügen eines „RoI Pooling Layers“<sup>3</sup> sowie eines „Softmax-Layers“ für die Vorhersage eines Objektes und eines vollständig verbundenem Layers für die endgültige Vorhersage der entsprechenden Boxen des Objektes. Fast R-CNN ist dabei schneller als R-CNN, da er mehrere Regionsvorschläge gleichzeitig verarbeitet und nicht jede Region einzeln betrachtet. Zusätzlich baut Fast R-CNN ein Netzwerk und hat folglich nur noch eine Etappe beim Training. [31]

Der RoI Pooling Layer nutzt „Max Pooling“, um die Featuremap in jeder RoI auf eine festgelegte Größe zu konvertieren. [31] Bei „Max Pooling“ wird die Eingabe in ein Gitter bestimmter Größe geteilt und aus jeder Zelle lediglich das Element mit dem größten Wert behalten und als Eingabe für den nächsten Layer genutzt [32].

Im selben Jahr wurde auch die Verbesserung Faster R-CNN veröffentlicht. Diese stellt einen großen Fortschritt in Sachen Verarbeitungsgeschwindigkeit dar, da sie nicht mehr auf dem Zeitintensiven „selective search“ beruht. Stattdessen nutzt Faster R-CNN einen vollständig verbundenen R-CNN als RPN. Das Nutzen eines RPN bietet dabei eine Reihe an Vorteilen über „selective search“. Da es ein neuronales Netz als Grundlage nutzt, kann es auf spezifische Detektionsaufgaben trainiert werden, wodurch es bessere Ergebnisse liefert als generische Algorithmen wie „selective search“. Zudem nutzt es die „Convolutional Layer“ des Fast R-CNN, wodurch sowohl beim Trainieren als auch beim Generieren von Regionen Zeit gespart wird. Für die letztendliche Regionsvorhersage wird ein „sliding window“ Ansatz genutzt. Dabei gleitet ein  $n \times n$  großes Netzwerk über den letzten von RPN und Fast R-CNN geteilten Layer. Die Ausgabe dieses Netzwerkes ist für jede Position ein Vektor mit geringerer Dimensionalität, welcher als Eingabe für zwei vollständig verbundene Layer dient. Einer davon bestimmt den „Objectness Score“ der entsprechenden Region, der andere dient der Vorhersage der zugehörigen Boxen. Die vorhergesagten Boxen sind relativ zu sogenannten „Anchor Boxen“, das sind  $k$  Boxen (im Paper zu Faster R-CNN 9), welche auf den Mittelpunkt des „sliding windows“ zentriert sind und unterschiedliche Seitenverhältnisse oder Skalierungen aufweisen. [33]

<sup>3</sup><https://deepsense.ai/region-of-interest-pooling-explained>

Der *Objectness-Score* wird vereinfacht über die in Abbildung 2.10 gezeigte Formel berechnet. Dieser besagt, ob eine vorgeschlagene Region Hintergrund oder ein Objekt ist. [33, 34]

Für das Training des RPN wird jeder „Anchor“, der mit einer *Ground Truth Box* eine *IoU* über 0,7 hat, sowie der „Anchor“ mit der höchsten *IoU*, falls keiner eine *IoU* über 0,7 hat, als positiv angenommen. Als negative gelten alle nicht positiven „Anchor“ mit einer *IoU* unter 0,3. Alle anderen Boxen werden während des Trainings ignoriert. [33]

$$\text{Objectness}(IoU) = \begin{cases} \text{positiv} & \text{falls } IoU > 0.7 \\ \text{negativ} & \text{falls } IoU < 0.3 \end{cases}$$

**Abbildung 2.10:** vereinfachte Formel zur Berechnung des Objectness-Score [34]

Für das Training des gesamten Netzwerkes, gibt es mehrere Varianten. Die Erste wird „Alternating Training“ genannt. Bei dieser werden RPN und das Fast R-CNN Modul separat trainiert. Gestartet wird dabei mit dem RPN, dieser wird für die später geteilten „Convolutional Layer“ mit auf ImageNet vortrainierten Daten und für die RPN spezifischen Layer mit zufälligen Daten initialisiert und anschließend mit der beschriebenen Methode trainiert. Darauf folgend wird der Fast R-CNN initialisiert mit den geteilten Schichten des RPN und für die Fast R-CNN spezifischen Schichten mit zufälligen Daten. Trainiert wird dieser mit den Regionsvorschlägen des RPN. Der nächste Schritt ist das erneute Trainieren des RPN, allerdings werden bei diesem Training nur die RPN spezifischen Layer optimiert. Zuletzt muss der Fast R-CNN erneut trainiert werden und auch bei diesem werden nur noch die Schichten spezifisch für diesen angepasst. Damit teilen beide Netze die gleichen „Convolved Layer“. Die zweite Methode wird „Approximate Joint Training“ genannt. Bei dieser werden RPN und der Fast R-CNN als ein Netzwerk betrachtet. Während des Trainings bei diesem werden die Gewichte des RPN nicht angepasst, und die Gewichte der geteilten Schichten erst nach der Ausgabe des Fast R-CNN aktualisiert. Das verringert die Genauigkeit des Netzwerkes, reduziert aber auch die Trainingsdauer um 25 bis 50 Prozent. [33, 34]

Eine weitere Variante des R-CNN ist der Mask R-CNN. Dieser erweitert Faster R-CNN um die Fähigkeit eine Objekt-Maske für Objekte vorherzusagen. Dabei bleibt das RPN gleich, und zu den beiden Ausgaben „Klasse“ und „Boxoffset“ gibt der Algorithmus zusätzlich eine binäre Maske für jede *Roi* aus. Wirkliche Verbesserungen in der Performance bietet Mask R-CNN im Vergleich zu Faster R-CNN jedoch nicht, und auch wenn die erstellte Maske für Aufgaben wie das Vorhersagen der Stellung von Menschen sehr nützlich sein kann, ist es für das hier vorhandene Problem nicht zielführend. [35]

### 2.3.3 Single Shot MultiBox Detektor

Der *Single Shot Multibox Detector* wurde ebenfalls 2015 vorgestellt. Er ist dabei wie YOLO ein „Single Stage Detector“, der dabei in Real-Time, mit einer Genauigkeit, die die von nicht Real-Time Algorithmen wie Faster R-CNN teilweise übertrifft, Objekte detektieren kann. Als Backbone (siehe Kapitel 2.3.1) nutzt SSD eine veränderte Version der CNN-Struktur VGG-16. Allerdings ist diese austauschbar, und unter Nutzung einer schnelleren könnte auch die Variante SSD512 in Real-Time laufen. Bei der veränderten VGG-16 Variante wurden die finalen vollständig verbundenen Klassifizierungsschichten entfernt und durch zwei „Convolved Layer“ ersetzt. Anschließend werden

mehrere „Auxiliary Convolutional Layer“ („Extra Feature Layers“ in Abbildung 2.11) eingefügt, welche die Featuremap verkleinern sollen. Zusätzlich wird die Struktur des Pooling Layer „pool5“ angepasst und der „à trous“-Algorithmus [36] genutzt, um dabei entstehende Löcher zu füllen. [37]

Die Originale VGG Struktur hat den Nachteil, dass die finalen vollständig verbundenen Schichten für eine sehr große Menge Parameter sorgen [14]. Dadurch, dass diese bei SSD entfernt sind, existiert dieses Problem für diesen Algorithmus nicht. Resultierend wird sowohl der benötigte Speicher als auch die Zahl der benötigten Rechenoperationen, folglich auch die Rechenzeit, verringert.

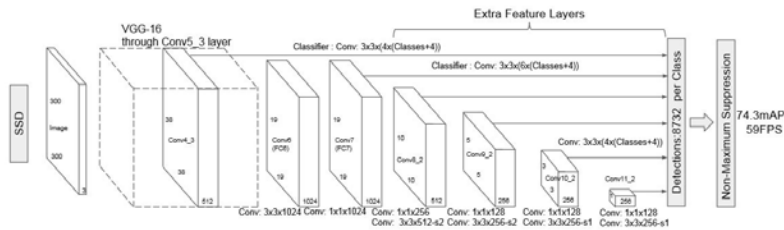


Abbildung 2.11: Struktur des SSD[37]

Für die abschließende Detektion werden, wie in Abbildung 2.11 gezeigt, die Featuremaps der Layer Conv4\_3, Conv7, sowie der zusätzlichen Auxiliary Layer genutzt. Diese können über eine Reihe von „Convolutional Filtern“ 8732 Detektionsvorhersagen treffen. Diese Filter sind Bestandteil des „Model Heads“ des SSD. Diese Filter oder Kernel haben eine Größe von 3x3 und werden auf den Featuremaps der verschiedenen „Convolutional Layer“ angewendet. Für jeden Detektionskopf wird, über zwei „Convolutional Layer“, der „Confidence Score“ und ein Offset für eine zugehörige Objektbox zurückgegeben. Der „Confidence Score“ ist ein Vektor, bei dem die Länge der Anzahl der zu detektierenden Klassen entspricht. Das Offset beschreibt, inwieweit die Position einer Anchor Box verändert werden muss, um besser die zugehörige Ground Truth Box zu treffen. [38]

SSD teilt, ähnlich dem YOLO, das Bild in ein Gitter. Dabei ist jede Zelle dieses Gitters für die Detektion der Objekte deren Mittelpunkt in dieser liegt verantwortlich. Für jede Zelle wird eine bestimmte Anzahl an Anchor Boxen verschiedener Skalierungen und Seitenverhältnisse generiert.

Während des Trainings werden alle Standard- oder auch Anchorboxen, die mit den Begrenzungsrahmen auf den annotierten Bildern einen IoU von über 0.5 haben, als Positive genommen. Negative sind jedoch nur die Standardboxen, die den höchsten Verlust erzeugen, sodass das Verhältnis von Negativen zu Positiven lediglich 3:1 ist. Abbildung 2.12 zeigt dabei die allgemeine Objektive Verlustfunktion. Diese setzt sich zusammen aus dem Lokalisierungsverlust  $L_{loc}$  und dem Vertrauensverlust  $L_{conf}$ .  $N$  stellt dabei die Anzahl an zugeordneten Standardboxen dar. [37] Der Lokalisierungsverlust

$$L(x, c, l, g) = \frac{1}{N} * (L_{conf}(x, c) + \alpha * L_{loc}(x, l, g))$$

Abbildung 2.12: allgemeine Objektive Verlustfunktion von SSD

ist ein smooth L1 Verlust (Formel in Abbildung 2.13) zwischen vorhergesagter Box und tatsächlicher Ground Truth Box, der angibt, wie genau die Ground Truth Box durch die Vorhersage getroffen wurde. Dabei gibt es für Negative keine Ground Truth Boxen, damit der Algorithmus nicht lernt, um leere Regionen Boxen zu generieren. Der Vertrauensverlust bezeichnet die Summe der Kreuzentropiever-



luste<sup>4</sup> der Positiven sowie der Negativen, gemittelt auf die Anzahl der Positiven. Er betrifft dabei die Klassenlabel der Detektionsergebnisse. Positive beinhalten ein Objekt, Negative entsprechen immer dem Hintergrund. [38]

$$\text{smooth}_{L1}(x) = \begin{cases} 0,5 * x^2 & \text{falls } |x| < 1 \\ |x| - 0,5 & \text{falls } |x| > 1 \end{cases}$$

**Abbildung 2.13:** smooth L<sub>1</sub> Verlust [31]

Die Erweiterung der Daten, durch z.B. Spiegelungen oder das zufällige Ausschneiden eines Teils des Bildes und das ledigliche Nutzen dieses Ausschnitts, ist für diesen Algorithmus essentiell, da auf diese Weise die mAP um bis zu 8.8% verbessert werden kann. Auch ist die Anzahl an Standardboxen relevant für die Detektionsgenauigkeit. Mehr Boxen erhöhen die Genauigkeit, jedoch auch die benötigte Zeit für Training und Detektion. Da bei SSD sehr viele Boxen generiert werden, muss zum Schluss eine Non Maximum Supression durchgeführt werden, um nur die besten Begrenzungsrahmen-Vorhersagen zu behalten.[37]

## 2.4 Möglichkeiten der Bildverbesserung

### 2.4.1 Bilden des Negativs eines Bildes

Das Bilden des Negativs eines Bildes ist vermutlich die einfachste Methode der Bildverbesserung. Diese ist nützlich für die Hervorhebung weißer oder grauer Details, welche umgeben sind von dunklen Regionen. Berechnet wird das Negativ eines Bildes der Maße RxC, bei dem jeder Pixel des Originalbildes durch I(r,c) und jeder Pixel des Negatives durch N(r,c) beschrieben wird mit der in Abbildung 2.14 gezeigten Formel. [39]

$$N(r, c) = 255 - I(r, c) \text{ mit } 0 \leq r \leq R \text{ und } 0 \leq c \leq C$$

**Abbildung 2.14:** Formel für die Berechnung des Negativs eines Bildes [39]



**Abbildung 2.15:** Negativ eines Bildes

Abbildung 2.15 zeigt das Negativbild zu dem in Kapitel 2.4.3, in Abbildung 2.19, gezeigten Bild. Dabei hebt diese Form der Bildverbesserung vielleicht das Fahrzeug nicht hervor, allerdings besteht die Möglichkeit, dass Nummernschilder besser erkannt werden. Das Originalbild zeigt einen an der Kamera vorbeifahrenden LKW.

<sup>4</sup><https://www.justintodata.com/logistic-regression-for-machine-learning-tutorial/>

## 2.4.2 Gamma Korrektur

**Power Law Transformationen.** Gamma Korrektur beschreibt den Prozess des Veränderns der Helligkeit eines Bildes unter Verwendung nicht-linearer Abbildung. Dies dient der Verbesserung des Kontrastes. Die Formel dafür ist in Abbildung 2.16 dargestellt.  $\gamma$  ist dabei der gewünschte Gammawert,  $c$  und  $\gamma$  müssen dabei positiv sein.  $s$  ist der Ausgabepixel,  $r$  der Eingabepixel. [39, 4]

$$s = c * r^{\frac{1}{\gamma}}$$

Abbildung 2.16: Formel für die Power Law Transformation [39, 4]

Abbildung 2.17 zeigt das bereits bekannte Bild eines vorbeifahrenden LKWs mit unterschiedlichen Gammawerten.

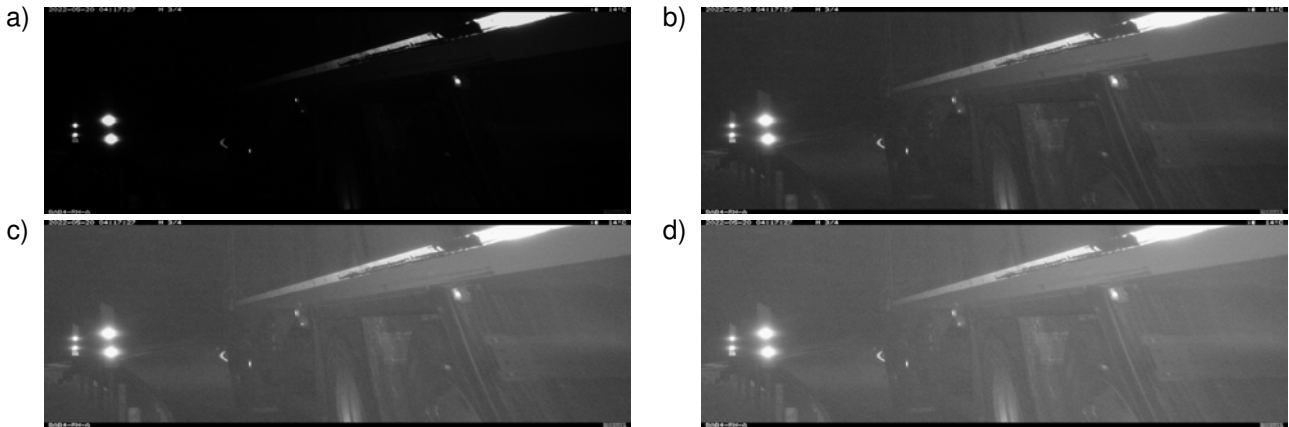


Abbildung 2.17: Das gleiche Bild mit verschiedenen Gammawerten: a) Gammawert von 0.5, b) Gammawert von 1.5, c) Gammawert von 2, d) Gammawert von 2.5

**Piecewise Linear Transformation.** Die Piecewise Linear Transformation ist ebenfalls eine Gamma-Korrektur, allerdings wird dabei keine wohldefinierte mathematische Funktion genutzt, sondern eine Nutzerdefinierte Funktion, wie in Abbildung 2.18 zu sehen.

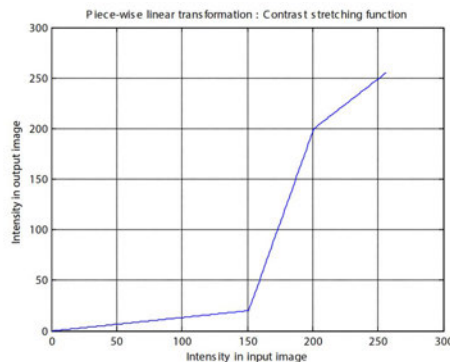


Abbildung 2.18: Piecewise linear transformation graph [40]



### 2.4.3 Histogramm Bearbeitung

**Histogram Equalization.** „Histogram Equalization“ ist eine Form von Histogrammbearbeitung bei der ein Histogramm mit einem Peak in einem bestimmten Bereich, auf einen weiteren Bereich gestreckt wird. Bei einem Bild mit einem Peak im dunklen Bereich, wie z.B. in Abbildung 2.19 zu sehen, wird dieser Peak in den helleren Bereich gestreckt, zu sehen in Abbildung 2.20. Die Funktion für diese Streckung wird dabei automatisch ermittelt. [39]

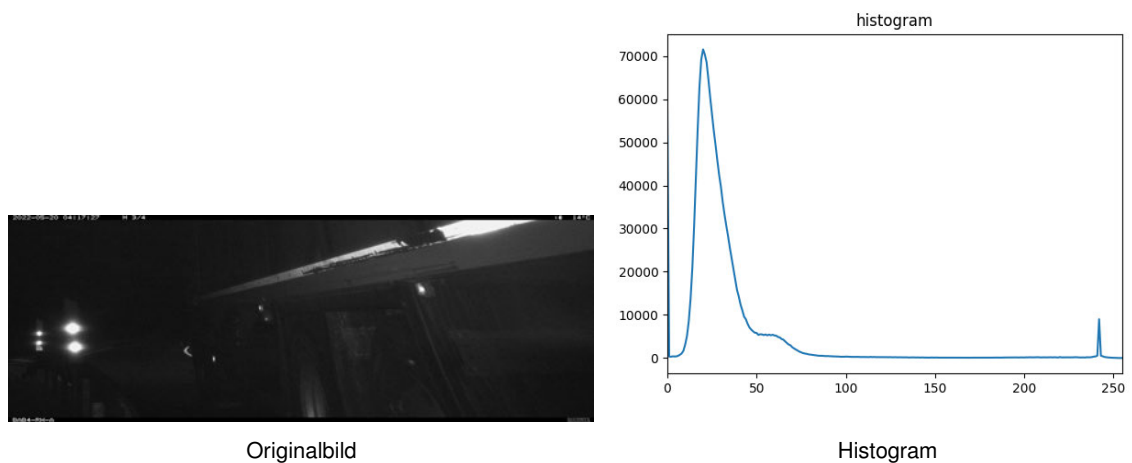


Abbildung 2.19: Originalbild und zugehöriges Histogramm

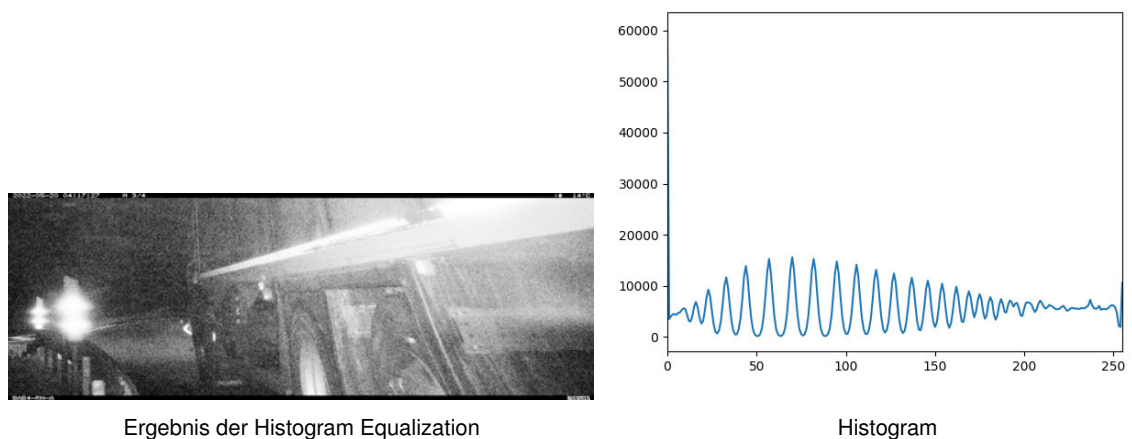


Abbildung 2.20: Ergebnis der Histogram Equalization und zugehöriges Histogramm

**Histogram Matching.** „Histogram Equalization“ ermittelt anhand des Histogramms automatisch eine passende Transformationsfunktion. Bei „Histogram Matching“ wird das Bild auf ein gegebenes Histogramm angepasst. Dafür wird das Histogramm des Eingabebildes bestimmt und eine passende Transformationsfunktion mit dem gegebenen Histogramm ermittelt.[39]

**Local Histogram Equalization.** „Local Histogram Equalization“ ist die Anwendung der „Histogram Equalization“ auf lokale Nachbarschaften. Dafür wird das Bild in ein Gitter geteilt und auf jede Zelle dieses Gitters die „Histogram Equalization“ angewendet.[39] Das Ergebnis ist in Abbildung 2.21 zu sehen.



Abbildung 2.21: Ergebnis der Lokalen Histogram Equalization

**CLAHE.** „Contrast Limited Adaptive Histogram Equalization“ ist ein vor allem in medizinischer Bildverarbeitung genutzter Algorithmus. Dabei wird das ursprüngliche Bild in mehrere kleinere Rechtecke unterteilt, mehrere Schwellwerte angewendet, gefolgt von einer „Histogram equalization“ für jedes kleinere Rechteck. [41] Das Ergebnis von CLAHE auf dem Bild des vorbeifahrenden LKWs ist in Abbildung 2.22 zu sehen.

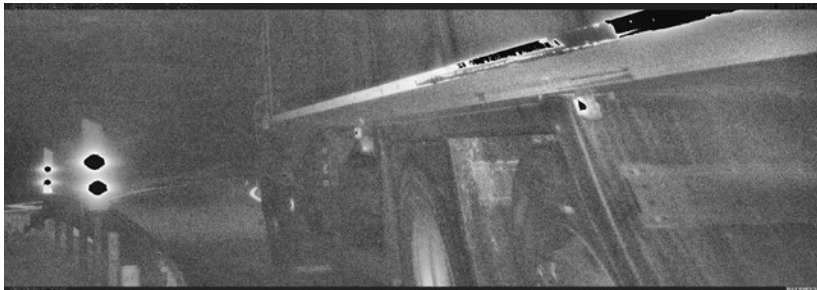


Abbildung 2.22: Ergebnis von CLAHE

## 2.5 Weitere wichtige Berechnungen

**Jaccard Index.** Der Jaccard Index, auch [Intersection over Union](#), gibt an, wie stark sich zwei Boxen A und B überschneiden.

$$\text{Berechnung: } IoU = \frac{A \cap B}{A \cup B}$$

**Softmax-Funktion.** Die Softmax-Funktion nimmt einen Vektor reeller Zahlen und normalisiert diese auf Werte zwischen Null und Eins, welche sich zusammen zu Eins summieren. Dadurch können diese anschließend als Wahrscheinlichkeitswerte interpretiert werden. [42, 14]

$$\text{Berechnung: } \sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

**Precision und Recall.** Precision und Recall sind Werte, welche zur Evaluierung herangezogen werden können. Precision beschreibt dabei den Anteil der richtig als Positiv klassifizierten; Recall beschreibt den Anteil der positiv klassifizierten von den eigentlich Positiven. Zur Berechnung dieser Werte wird die in Abbildung 2.1 dargestellte Tabelle genutzt. [43]

	Wahrheit		
		Positiv	Negativ
Vorhersage	Positiv	richtig Positiv (rp)	falsch Positiv (fp)
	Negativ	falsch Negativ (fn)	richtig Negativ (rn)

Tabelle 2.1: Konfusionsmatrix

Berechnung: Precision  $p = \frac{rp}{rp+fp}$ , Recall  $r = \frac{rp}{rp+fn}$

**F-Wert.** Der F-Wert beschreibt das Verhältnis zwischen Precision und Recall. Besonders häufig wird der  $F_1$ -Wert genutzt. [44]

Berechnung:  $F_\beta = (1 + \beta^2) * \frac{p*r}{r+\beta^2*p}$

**mean Average Precision.** Die **mAP** berechnet sich indem die **Average Precision** jeder Klasse gemittelt wird. Die **AP** einer Klasse stellt die Fläche unter der Precision-Recall Kurve der entsprechenden Klasse dar. [45] Bei der Notation **mAP@X**, ist die **mAP**, mit einem **IoU**-Schwellwert, mit einer **Ground Truth Box**, von X, ab dem ein Objekt als positiv angenommen wird, gemeint. Die **mAPX:Y** beschreibt die gemittelte **mAP** über **IoU**-Schwellwerte in einem Intervall von X bis Y.

Berechnung **AP**:  $AP = \sum_{k=0}^{k=n-1} [Recall(k) - Recall(k+1)] * Precision(k)$ , mit n als Anzahl an Schwellwerten

Berechnung **mAP**:  $mAP = \frac{1}{n} * \sum_{k=1}^{k=n} AP_k$ , mit  $AP_k$  als **AP** der Klasse k und n als Anzahl an Klassen

## 2.6 Grundlagen der Optical Character Recognition

**OCR** beschreibt den Prozess, Schriftzeichen auf Bildern zu erkennen. Die ersten Geräte, die dazu in der Lage waren, wurden vor dem ersten Computer entwickelt, das Erste von Charles R. Carey 1870 [46]. Trotz dessen wurde erst in den letzten 30 bis 40 Jahren viel Forschung in diesem Bereich betrieben. Hauptaugenmerk bei dieser liegt auf Geschwindigkeit und Genauigkeit, da die Fähigkeit von Maschinen, Text verlässlich zu lesen, die von Menschen weit untertrifft. Dabei ist die Erkennung von gedrucktem Text einfacher, da die Zeichen bei diesem zumeist von uniformer Größe sind. Schwieriger ist die Erkennung von Handschriften, da jeder Mensch unterschiedliche Schreibmuster hat und ein Mensch für denselben Buchstaben verschiedene Schreibstile aufweisen kann. [47]

Für deutsche Nummernschilder ist die Texterkennung einfacher, da diese, mit Ausnahme des Buchstabens für das Land, komplett in FE-Schrift beschriftet sind.<sup>5</sup> Diese wurde ab 1978 für den Einsatz auf deutschen Kennzeichen entwickelt. Bei der Entwicklung dieser waren das Erschweren von Fälschungen als auch die Maschinenlesbarkeit die zwei Hauptaspekte. [48]

Eine **OCR**-Anwendung besteht meist aus folgenden Schritten [47]:

1. Aufnahme des Bildes

<sup>5</sup>FZV Anlage 4 Nummer 2

2. Verbesserung des Bildes
3. Segmentierung des Bildes in die einzelnen Zeichen
4. Feature Extraktion der einzelnen Zeichen
5. Klassifizierung der Zeichen
6. Nachbearbeitungen

Verbesserungen und Nachbearbeitungen sind optional, die Schritte 3 bis 5 können in einen zusammengefasst werden, z.B. indem für diese ein Single-Stage-Detektor, wie SSD [11], eingesetzt wird. Ein wichtiger Bestandteil der Verbesserungen ist die Bestimmung der Schräglage im Dokument. Nachbearbeitungen können das Kombinieren verschiedener Klassifizierer oder das Anwenden kontextueller Analysen beinhalten. [47]

Eine beliebte und weit verbreitete Technik für das Klassifizieren von Zeichen in einer OCR-Anwendung ist „Template Matching“. Bei diesem werden die in Schritt 3 segmentierten Zeichen mit Zeichen eines Referenzbilddatensatzes verglichen und das Zeichen mit der größten Ähnlichkeit ist das letztendlich Detektierte. Einen möglichen Referenzdatensatz für die FE-Schrift kann man in Abbildung 2.23 sehen. Diese Methode ist schneller zu implementieren als ein Ansatz mit neuronalen Netzen, jedoch empfindlich gegenüber Verzerrungen und ähnlichen Störfaktoren, wie der Neigung des Nummernschildes auf dem Bild. [49] Auch beeinflusst die Qualität der Templates die Erkennungsgenauigkeit. So spielen die Schriftart des Templates, sowie die Auflösung der einzelnen Zeichen in diesem eine große Rolle. [50]

ABCDEFGHIJKLM  
NOPQRSTUVWXYZ  
0123456789 ÄÖÜ

Abbildung 2.23: Zeichensatz der FE-Schrift[48]

Ein Beispiel für einen Algorithmus der auf der Erkennung von Zeichenmuster beruht, ist der ursprüngliche Tesseract Ansatz. Dies ist eine ursprünglich bei Hewlett-Packard entwickelte und von 2006 bis 2018 bei Google weiterentwickelte OCR-Software.

Tesseract startet mit einem Page Layout Algorithmus, um Regionen mit Text zu finden. Dem folgt ein „Line-Finding“-Algorithmus. Dieser ist so konstruiert, dass eine Seite ohne Anpassung der Schräglage gelesen werden kann. Die zwei Hauptbestandteile des Prozesses sind ein „Blob Filter“ und eine „Linien Konstruktion“. Dabei werden durch einen Höhenfilter vertikal verbundene Zeichen getrennt und Flecken zu geringer Höher entfernt. Für übriggebliebene Zeichen werden Grundlinien generiert, welche anschließend mit „quadratischem Spline“ besser an die Schrift angepasst werden. Anhand dieser Linien wird der Abstand der Zeichen ermittelt. Für die Unterteilung wird zunächst an Stellen separiert, die einen festen Abstand aufweisen. Alle Worte die danach nicht unterteilt sind, werden an Verbundstellen einzelner Zeichen getrennt. Für die Klassifizierung werden ein statischer und ein adaptiver Klassifizierer implementiert. [51] Ab Tesseract 4 nutzt auch Tesseract einen Algorithmus beruhend auf neuronalen Netzen basierend auf einem Long Short-Term Memory Netzwerk. [52]

Alternativ dazu existiert „EasyOCR“<sup>6</sup>, eine auf PyTorch beruhende Python-Bibliothek. Diese nutzt für die Detektion den Craft-Algorithmus. Anschließend folgt eine Feature-Extraktion mithilfe eines ResNets, eine Markierung von Sequenzen mit einem Long Short-Term Memory-Algorithmus und

<sup>6</sup><https://github.com/JaidedAI/EasyOCR>

eine Decodierung basierend auf CTC. Sowohl der Craft-Algorithmus als auch die folgenden Schritte sind austauschbar, da die Autoren von EasyOCR das Ziel verfolgen, State of the Art-Modelle einfach in EasyOCR einbinden zu können.

Tesseract ist dabei 0,24 Sekunden schneller als EasyOCR, jedoch auch wesentlich ungenauer. Ob dies auch auf Tesseract 4 zutrifft ist fraglich, da der beschriebene Tesseract-Ansatz dem ursprünglichen Algorithmus gleicht. [10] Übertroffen wird die Genauigkeit von Tesseract und EasyOCR durch den in [11] vorgestellten Ansatz, die Segmentierung und Klassifizierung durch einen SSD-Algorithmus zu lösen. Auch ist der SSD-Ansatz, unter Verwendung von OpenVINO <sup>7</sup>, auf einem Raspberry Pi 3B um etwas mehr als das 9-fache schneller als Tesseract. [11]

---

<sup>7</sup><https://docs.openvino.ai/latest/index.html>



## 3 Durchführung

Sämtlicher Quellcode wurde in der Programmiersprache Python geschrieben. Als Bibliothek für den „Machine Learning“ Anteil des Codes wurde PyTorch verwendet. Diese ist einfach zu verwenden und wurde auf Programmierung in Python optimiert. Die Bibliothek wurde 2017 von Facebook veröffentlicht. Sie hat dabei viele Ideen und Konzepte von Torch, einer Lua-basierten Bibliothek für Neuronale Netze, und Chainer, einer aus Japan und dem Jahr 2015 stammenden Bibliothek für Neuronale Netze, übernommen. [14] Bildverarbeitungen und Bildverbesserungen erfolgten zu großen Teilen unter Verwendung der openCV-Bibliothek für Python.

### 3.1 Training

Das Training der Modelle erfolgt mit 25 und 50 Epochen. Verwendet wird dafür der Trainingsdatensatz mit sowohl Originalbildern als auch Datensätze mit verbesserten Bildern.

**YOLOv5.** Der Code von YOLOv5 wurde durch die Firma Ultralytics im Github Repository [53] veröffentlicht. Dieses beinhaltet bereits eine Datei „train.py“. Diese enthält sämtlichen Code der benötigt wird, um jedes YOLOv5 Modell zu trainieren und anschließend zu validieren. Beim Aufruf dieses Codes muss man verschiedene Parameter übergeben:

- „-data“ - Pfad zur YAML-Datei des Datensets (Autodaten.yaml in Abbildung 2.3)
- „-weights“ - welche Art von Modell trainiert werden soll (wo nicht anders angegeben erfolgte das Training von yolov5s.pt)
- „-img“ - auf welche Größe die Bilder zur Verarbeitung skaliert werden sollen, die genutzten Modelle haben eine Eingabegröße von 640x640 Pixel
- „-epochs“ - Anzahl an Epochen die das Modell trainiert werden soll
- „-batch-size“ - Größe der Batches während des Trainings. Diese ist abhängig vom verfügbaren Speicher des Gerätes auf dem trainiert wird; für das Training in dieser Arbeit wurde die Größe von 16 gewählt
- „-name“ - Pfad zum Ordner in dem die Ergebnisse gespeichert werden müssen

Bricht das Training zwischendurch ab, ist eine Fortsetzung mit dem Parameter „-resume“ vom letzten Checkpoint aus möglich.

Als **Optimierer** dient die **SGD**-Funktion, welche standardmäßig von YOLOv5 genutzt wird.

Die Gewichte des trainierten YOLOv5 werden als pt-Datei gespeichert. Zusätzlich dazu werden für die Auswertung relevante Graphiken, wie z.B. die zugehörige Precision-Recall-Kurve oder die in Kapitel 2.1 gezeigte Übersicht über die Anzahl aller Klassen, erstellt.

**Faster R-CNN.** Als Modell für den Faster R-CNN wird das von PyTorch implementierte „fasterrcnn\_resnet50\_fpn“ genutzt. Geladen wird es dabei mit bereits trainierten Gewichten. Als **Optimierer** dient die ebenfalls von PyTorch implementierte **SGD**-Funktion<sup>8</sup>. Initialisiert wird diese mit einer **Lernrate** von 0,001, einem **Momentum Faktor** von 0,9 und einer **Weight Decay/L2 Strafe** von 0,0005.

<sup>8</sup><https://pytorch.org/docs/stable/generated/torch.optim.SGD.html>

Während des Trainings wird für jedes Bild im Trainingsdatensatz der Verlust berechnet und anschließend führt der **Optimierer** mit der zugehörigen `step()`-Funktion einen einzelnen Optimierungsschritt durch. Das Modell wurde alle zwei Trainingsepochen gespeichert, damit im Falle eines Abbruchs das Training fortgesetzt werden kann.

Der Code dafür stammt zu großen Teilen von der Webseite [54], dieser ähnelt dem in [14] genutztem Code. Vor Nutzung musste der Code modifiziert werden, um auf dem erstellten Datensatz zu funktionieren. Die hauptsächlichste Änderung dabei ist das Anpassen des Datensatzes auf die Coco-Struktur.

**SSD.** Genutzt wird **SSD300**, was bedeutet, die Bilder müssen vor Übergabe an das Modell auf eine Größe von 300x300 Pixel skaliert werden. Das für den **SSD** genutzte Modell stammt aus dem Github-Repository [38]. Es beschreibt perfekt das im Paper zu **SSD** erläuterte Modell und beinhaltet bereits alle für das Modell notwendigen Funktionen. Bei der Initialisierung des Modells wird es mit einem vortrainierten VGG16 Modell aus der Bibliothek Torchvision geladen<sup>9</sup>. Als **Optimierer** dient auch hier die **SGD**-Funktion. Initialisiert wird diese mit den selben Werten wie die Optimierungsfunktion des Faster **R-CNN**.

Für die Erweiterung der Daten während des Trainings wurden verschiedene photometrische Verzerrungen, diese beinhalten den Kontrast, die Helligkeit, die Sättigung und den Farbton, genutzt. Zudem das zufällige Verkleinern des Bildes und das Füllen der daraus entstehenden Lücken, sowie das horizontale Spiegeln des Bildes. Jede dieser Bearbeitungen hat eine 50%ige Wahrscheinlichkeit angewendet zu werden.

## 3.2 Evaluierung

Für die Evaluierung der Modelle werden verschiedene Maße genutzt. Die Evaluierung erfolgte für alle Modelle und Bildverbesserungen auf einer AMD Ryzen 7 3700 CPU.

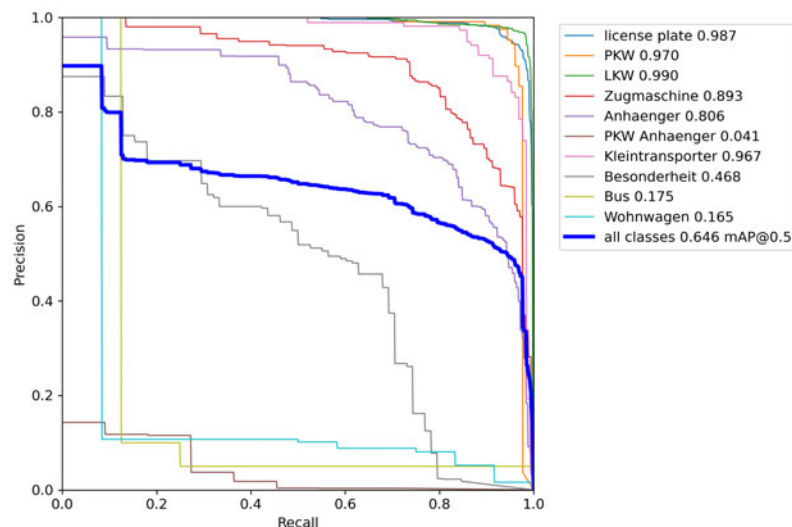
**Precision-Recall Kurven.** Precision-Recall Kurven dienen als Grundlage für die Berechnung der **mAP**. Außerdem kann an diesen ungefähr abgeschätzt werden, wie gut welche Klasse erkannt wird, abhängig vom Verlauf der Kurven. Je später die Precision fällt, desto besser. In Abbildung 3.1 wäre die Klasse „LKW“ am besten erkannt worden und „PKW Anhaenger“ am schlechtesten.

**mean Average Precision.** Die **mAP** berechnet sich entsprechend der Berechnungsvorschrift in Kapitel 2.5. Das ergibt einen Wert zwischen null und eins. Je größer dieser Wert ist, desto besser ist das trainierte Modell. Das Ziel ist also die Maximierung dieses Wertes. Die **mAP** wird dabei über alle Klassen berechnet, zusätzlich wird die **AP** jeder einzelnen Klasse angegeben. Bei diesen ist vor allem die Klasse „Nummernschild“ relevant.

Für die Bestimmung der **mAP** der unterschiedlichen Algorithmen werden sowohl die Originalbilder, als auch die bearbeiteten Bilder verwendet. Zusätzlich werden die Modelle auf Bildern evaluiert, welche anders bearbeitet wurden, als die Bilder auf denen trainiert wurde.

<sup>9</sup><https://pytorch.org/vision/stable/models/generated/torchvision.models.vgg16.html#torchvision.models.vgg16>





**Abbildung 3.1:** Precision-Recall Kurve des YOLOv5 mit 25 Trainingsepochen

Die Berechnung der **mAP** der YOLOv5 Modelle erfolgt unter der Verwendung des Evaluierungsskripts in dem Repository [53].

Das GitHub Repository für SSD [38] beinhaltet ein Evaluierungsskript, unter dessen Verwendung die **mAP@0.5** für diesen berechnet wurde. Die **mAP@0.5:0.95** konnte mit leichter Abänderung dieses Skriptes ebenfalls berechnet werden. Dafür werden für jeden **IoU**-Wert im Intervall zwischen 0,5 und 0,95, mit einer Schrittgröße von 0,05, die **APs** jeder Klasse, sowie die **mAP** gemittelt.

Mit weiteren Anpassungen konnte das Evaluierungsskript des SSD ebenfalls für den Faster R-CNN genutzt werden. Die hauptsächliche Veränderung dabei ist das Anpassen der Detektionsfunktion, da Faster R-CNN keine 8732 Boxen zurück gibt, sondern lediglich jene Boxen, die dem Neuronalen Netz zufolge ein Objekt beinhalten.

**Trainingszeit.** Da sämtliche Verarbeitungszeiten hardwareabhängig sind, wird bei der Trainingszeit das Verhältnis zum zeitintensivsten der verwendeten Algorithmen genutzt, in dieser Arbeit ist das Faster R-CNN. Damit ergibt sich eine Trainingszeit für den Faster R-CNN von 1.

Gemessen wird die Zeit unter Verwendung der Python-Bibliothek „time“.

**Detektionszeit.** Die Detektionszeit wird auf ähnlicher Hardware, wie jene auf der die Anwendung letztendlich laufen soll, gemessen. Gemessen wird über den kompletten Validierungsdatensatz und die Zeit anschließend auf die Dauer eines einzelnen Bildes heruntergerechnet. Da die Problemstellung fordert, dass viele Bilder schnell ausgewertet werden, ist dieser Wert relevanter als die Trainingszeit, jedoch ist eine größere **mAP** wichtiger als eine geringe Inferenzzeit.

**Dauer von Bildverbesserungen.** Entsprechend der Trainings- und Detektiondauer der einzelnen Detektionsalgorithmen wird die Dauer der einzelnen Bildverbesserungen auf ein einzelnes Bild berechnet. Diese Dauer ist jedoch im Vergleich zur Trainings- und Inferenzzeit so gering, dass diese nur marginal ist.

### 3.3 Bildverbesserungen

Nicht alle in Kapitel 2.4 gezeigten Bildverbesserungen werden genutzt. Die „Power Law Transformation“ und die „Piecewise Linear Transformation“ führen zu ähnlichen Bildern, weshalb lediglich die „Power Law Transformation“ genutzt wurde. Ähnlich ist es auch mit der „Histogram Equalization“ und der „lokalen Histogram Equalization“, allerdings übertrifft das Ergebnis von „CLAHE“ beide Algorithmen. Außerdem lieferte „CLAHE“ in einer Arbeit zur visuellen Navigation unter schlechten Lichtverhältnissen gute Ergebnisse [4], sodass dieser Algorithmus letztendlich genutzt wurde. Für „Histogram Matching“ wäre es erforderlich, ein Histogramm zu haben, von dem bekannt ist, dass es gute Ergebnisse liefert. Es wurde also ebenfalls nicht genutzt.

Genutzt wurden außerdem Negative, sowie eine Kombination aus Gamma Korrektion und CLAHE.

CLAHE erfolgte unter Verwendung der CLAHE-Funktion der Bibliothek cv2. Diese wird mit dem Parameter „clipLimit“ als fünf initialisiert. Dieser Wert wurde durch probieren verschiedener Werte bestimmt und gewählt, da dieser die Qualität des Bildes für menschliche Augen erhöht, ohne zu starkes Rauschen zu verursachen. Zusätzlich hat die Funktion den Parameter „tileGridSize“, welcher die Unterteilung des Bildes in kleinere Rechtecke angibt. Dieser wurde mit dem Standardwert von  $8 \times 8$  initialisiert.<sup>10</sup>

Den Code für das Erstellen der Negative ist in Listings 3.1 zu sehen. Dabei wird zunächst das Bild als Graustufenbild eingelesen, umgewandelt und anschließend am gewünschten Speicherort abgelegt.

**Quelltext 3.1:** Code für das Erstellen eines Negatives

```
import cv2

# Bild laden
img = cv2.imread(Pfad\zum\Original-Bild , 1)
# Negativ erstellen
img_neg = 1 - img
# Bild speichern
cv2.imwrite(Pfad\zum\Speicherort ,img_neg)
```

Das Listings 3.2 zeigt den Code für die Gamma Korrektion. Bei diesem wird das geladene Bild zusammen mit dem gewünschtem Gammawert als Parameter die Funktion `adjust_gamma` aufgerufen. In dieser wird zunächst eine Lookup Tabelle erstellt, in welcher die angepassten Gammawerte für jedes Eingabepixel gespeichert sind. Anschließend wird diese Tabelle auf das Bild mit der cv2 Funktion `LUT()` angewendet. Zum Schluss wird das Bild zurück gegeben und gespeichert.

<sup>10</sup>Code entspricht dem aus <https://www.geeksforgeeks.org/clahe-histogram-equalization-opencv/>

**Quelltext 3.2:** Code für das Anwenden der Gamma Korrektion

```
import cv2
import numpy as np

# Funktion der Gamma Korrektion
def adjust_gamma(image, gamma=1.0):
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255
                      for i in np.arange(0, 256)]).astype("uint8")
    return cv2.LUT(image, table)

# Bild laden
img = cv2.imread(Pfad\\zum\\Original-Bild)
# Gamma Korrektion anwenden
img_gamma = adjust_gamma(img, 2)
# Bild speichern
cv2.imwrite(Pfad\\zum\\Speicherort, img_gamma)
```

In dieser Arbeit wurde der Funktion für das Anwenden der Gamma Korrektion immer der Wert zwei übergeben. Dieser Wert hat von allen getesteten die besten Ergebnisse für das menschliche Auge erzielt.

Für die Kombination aus Gamma Korrektion und CLAHE wurde das geladene Bild zunächst der Gamma Korrektion mit einem Gamma-Wert von 2 unterzogen und anschließend die bereits beschriebene CLAHE Funktion genutzt. Das Ziel der Kombination war das Kombinieren der positiven Auswirkungen beider Algorithmen, beschrieben in [4].



## 4 Ergebnisse

In den nachfolgenden Ergebnistabellen finden sich die berechneten **mAP**-Werte der getesteten Modelle. Wie bereits in Kapitel 3.2 beschrieben, können diese einen Wert zwischen Null und Eins annehmen, wobei das Modell umso genauer ist, je größer diese Werte sind.

### 4.1 YOLOv5

Klasse	25 Epochen		50 Epochen	
	mAP@0.5	mAP@0.5:0.95	mAP@0.5	mAP@0.5:0.95
Alle	0,646	0,517	0,827	0,661
Nummernschild	0,987	0,729	0,98	0,75
PKW	0,97	0,777	0,962	0,76
LKW	0,99	0,927	0,986	0,929
Zugmaschine	0,893	0,707	0,892	0,715
Anhaenger	0,806	0,617	0,87	0,672
PKW Anhaenger	0,041	0,0209	0,864	0,526
Kleintransporter	0,967	0,824	0,94	0,796
Besonderheit	0,468	0,29	0,471	0,277
Bus	0,175	0,141	0,456	0,381
Wohnwagen	0,165	0,14	0,85	0,804

Tabelle 4.1: **mAP** des YOLOv5 auf dem Validierungs-Datensatz

Tabelle 4.1 zeigt den Vergleich zwischen 25 und 50 Trainingsepochen des YOLO auf den unbearbeiteten Bildern. Angegeben sind dabei die **mAP** für einen IoU-Schwellwert von 0,5 und der gemittelte Wert mehrerer Schwellwerte zwischen 0,5 und 0,95, ab denen ein detektiertes Objekt als Positiv erkannt wird.

Klasse	Negativ		Gamma Correction		CLAHE	
	mAP@.5	mAP@.5:.95	mAP@.5	mAP@.5:.95	mAP@.5	mAP@.5:.95
Alle	0,615	0,475	0,684	0,545	0,64	0,491
Nummernschild	0,98	0,723	0,979	0,738	0,977	0,699
PKW	0,955	0,746	0,965	0,739	0,951	0,697
LKW	0,987	0,909	0,987	0,919	0,988	0,906
Zugmaschine	0,889	0,687	0,9	0,713	0,882	0,691
Anhaenger	0,805	0,575	0,845	0,637	0,812	0,553
PKW Anhaenger	0,0414	0,00585	0,0108	0,00432	0,0137	0,00304
Kleintransporter	0,931	0,785	0,943	0,809	0,92	0,767
Besonderheit	0,427	0,201	0,474	0,259	0,372	0,174
Bus	0,0639	0,0559	0,314	0,256	0,129	0,0992
Wohnwagen	0,0674	0,0615	0,421	0,38	0,355	0,321

Tabelle 4.2: **mAP** des YOLOv5 trainiert und validiert auf verbesserten Bildern

Tabelle 4.2 zeigt, ähnlich wie Tabelle 4.1, die mAP-Werte mit verschiedenen IoU-Schwellwerten auf 25 Epochen, jedoch trainiert und evaluiert auf bearbeiteten Bildern.

Im Gegensatz zu den bisherigen Ergebnissen, bei denen auf gleich bearbeiteten Bildern trainiert und evaluiert wurde, zeigt Tabelle 4.3 die Ergebnisse der Evaluierung auf Bildern mit sich unterscheidenden Bearbeitungen. Allerdings nur für alle Klassen insgesamt und Nummernschilder, da dies die beiden relevantesten Kenngrößen sind.

		Training					
		Klasse	ohne Verbesserung	Negativ	Gamma Correction	CLAHE	Gamma Correction & CLAHE
Evaluierung	ohne Verbesserung	Alle	0,517	0,229	0,479	0,343	0,306
		Nummernschild	0,729	0,557	0,717	0,653	0,643
	Negativ	Alle	0,213	0,475	0,258	0,28	0,299
		Nummernschild	0,574	0,723	0,461	0,686	0,702
	Gamma Correction	Alle	0,501	0,364	0,545	0,443	0,456
		Nummernschild	0,737	0,604	0,738	0,667	0,671
	CLAHE	Alle	0,439	0,375	0,437	0,491	0,483
		Nummernschild	0,574	0,622	0,503	0,699	0,707
	Gamma Correction & CLAHE	Alle	0,427	0,377	0,411	0,481	0,483
		Nummernschild	0,529	0,659	0,419	0,687	0,708

**Tabelle 4.3:** mAP@0.5:0.95 des YOLOv5 im Vergleich

Tabelle 4.4 zeigt Ergebnisse der drei kleinsten YOLOv5 Modelle. Zusätzlich beinhaltet die Tabelle die Größen der Modelle, sowie die Trainings- und Detektionszeit.

	yolov5n		yolov5s		yolov5m	
Größe	3,721 MB		14,039 MB		41,196 MB	
Trainingsdauer	ca. 0,038		ca. 0,098		ca. 0,225	
Detektionsdauer	0,13921s		0,34973s		0,60915s	
Klasse	mAP@.5	mAP@.5:.95	mAP@.5	mAP@.5:.95	mAP@.5	mAP@.5:.95
Alle	0,597	0,454	0,646	0,517	0,806	0,651
Nummernschild	0,976	0,705	0,987	0,729	0,977	0,748
PKW	0,948	0,715	0,97	0,777	0,949	0,78
LKW	0,987	0,915	0,99	0,927	0,987	0,932
Zugmaschine	0,901	0,678	0,893	0,707	0,896	0,729
Anhaenger	0,783	0,539	0,806	0,617	0,812	0,654
PKW Anhaenger	0,00693	0,00142	0,041	0,0209	0,62	0,369
Kleintransporter	0,91	0,745	0,967	0,824	0,943	0,817
Besonderheit	0,377	0,169	0,468	0,29	0,525	0,334
Bus	0,0328	0,0294	0,175	0,141	0,427	0,387
Wohnwagen	0,0505	0,0433	0,165	0,14	0,873	0,758

**Tabelle 4.4:** mAP des YOLOv5 unter Verwendung unterschiedlicher YOLOv5 Modelle

## 4.2 Faster R-CNN

Tabelle 4.5 zeigt die mAP des Faster R-CNN auf den unverbesserten Bildern. Aufgrund der Trainingsdauer wurde dabei auf das Training unter 50 Epochen verzichtet.

Klasse	25 Epochen	
	mAP@.5	mAP@.5:.95
Alle	0,320	0,180
Nummernschild	0,845	0,547
PKW	0,409	0,248
LKW	0,841	0,373
Zugmaschine	0,250	0,159
Anhaenger	0,412	0,202
PKW Anhaenger	0,0	0,0
Kleintransporter	0,293	0,179
Besonderheit	0,155	0,093
Bus	0,0	0,0
Wohnwagen	0,0	0,0

Tabelle 4.5: mAP des Faster R-CNN auf dem Validierungs-Datensatz

Tabelle 4.6 beinhaltet die mAP für auf verbesserten Bildern trainierte und evaluierte Modellen.

Klasse	Negativ		Gamma Correction		CLAHE	
	mAP@.5	mAP@.5:.95	mAP@.5	mAP@.5:.95	mAP@.5	mAP@.5:.95
Alle	0,281	0,146	0,380	0,188	0,293	0,161
Nummernschild	0,795	0,505	0,841	0,492	0,734	0,450
PKW	0,143	0,069	0,418	0,213	0,326	0,178
LKW	0,811	0,328	0,828	0,352	0,818	0,362
Zugmaschine	0,210	0,111	0,174	0,086	0,190	0,111
Anhaenger	0,408	0,153	0,395	0,180	0,442	0,234
PKW Anhaenger	0,015	0,008	0,007	0,002	0,020	0,006
Kleintransporter	0,228	0,166	0,357	0,207	0,264	0,181
Besonderheit	0,198	0,119	0,172	0,094	0,138	0,091
Bus	0,0	0,0	0,0	0,0	0,0	0,0
Wohnwagen	0,0	0,0	0,608	0,256	0,0	0,0

Tabelle 4.6: mAP des Faster R-CNN trainiert und validiert auf verbesserten Bildern

Tabelle 4.7 zeigt die mAP@0.5:0.95 für Modelle evaluiert auf Bildern, welche anders bearbeitet wurden als die Bilder auf denen diese Modelle trainiert wurden.

		Training					
		Klasse	ohne Verbesserung	Negativ	Gamma Correction	CLAHE	Gamma Correction & CLAHE
Evaluierung	ohne Verbesserung	Alle	0,180	0,088	0,151	0,103	0,108
		Nummernschild	0,547	0,070	0,504	0,445	0,354
	Negativ	Alle	0,096	0,146	0,066	0,100	0,089
		Nummernschild	0,251	0,505	0,088	0,412	0,285
	Gamma Correction	Alle	0,157	0,081	0,188	0,120	0,114
		Nummernschild	0,516	0,042	0,492	0,454	0,321
	CLAHE	Alle	0,126	0,057	0,087	0,161	0,129
		Nummernschild	0,319	0,157	0,227	0,450	0,310
	Gamma Correction & CLAHE	Alle	0,116	0,053	0,081	0,149	0,127
		Nummernschild	0,336	0,235	0,186	0,436	0,310

**Tabelle 4.7:**  $mAP@0.5:0.95$  des Faster R-CNN im Vergleich

Die Detektionszeit des Faster R-CNN beträgt pro Bild 5,6s.

### 4.3 SSD

Tabelle 4.8 zeigt die  $mAP@0.5$  und die  $mAP@0.5:0.95$  des SSD-Algorithmus mit Training unter 25 und 50 Epochen.

Klasse	25 Epochen		50 Epochen	
	$mAP@.5$	$mAP@.5:.95$	$mAP@.5$	$mAP@.5:.95$
Alle	0,614	0,372	0,649	0,393
Nummernschild	0,806	0,421	0,861	0,528
PKW	0,902	0,537	0,890	0,549
LKW	0,905	0,747	0,902	0,741
Zugmaschine	0,736	0,396	0,676	0,356
Anhaenger	0,699	0,404	0,639	0,372
PKW Anhaenger	0,272	0,122	0,640	0,221
Kleintransporter	0,850	0,522	0,768	0,528
Besonderheit	0,249	0,106	0,291	0,101
Bus	0,361	0,199	0,477	0,301
Wohnwagen	0,363	0,266	0,349	0,223

**Tabelle 4.8:**  $mAP$  des SSD300 auf dem Validierungs-Datensatz



In Tabelle 4.9 ist die **mAP** des **SSD**, trainiert und evaluiert auf bearbeiteten Bildern, dargestellt.

Klasse	Negativ		Gamma Correction		CLAHE	
	mAP@.5	mAP@.5:.95	mAP@.5	mAP@.5:.95	mAP@.5	mAP@.5:.95
Alle	0,628	0,374	0,614	0,376	0,610	0,388
Nummernschild	0,818	0,446	0,841	0,489	0,816	0,454
PKW	0,893	0,525	0,883	0,547	0,865	0,533
LKW	0,906	0,718	0,906	0,755	0,907	0,759
Zugmaschine	0,721	0,393	0,693	0,396	0,631	0,362
Anhaenger	0,732	0,401	0,654	0,378	0,745	0,477
PKW Anhaenger	0,567	0,275	0,527	0,177	0,249	0,108
Kleintransporter	0,803	0,535	0,826	0,561	0,825	0,546
Besonderheit	0,251	0,104	0,265	0,104	0,332	0,145
Bus	0,371	0,206	0,356	0,261	0,524	0,337
Wohnwagen	0,221	0,143	0,185	0,096	0,209	0,158

**Tabelle 4.9:** **mAP** des **SSD300** trainiert und validiert auf verbesserten Bildern

In Tabelle 4.10 sind die Ergebnisse der Evaluierung des **SSD**, wenn dieser auf anders bearbeiteten Bildern evaluiert als trainiert wird. Dabei beinhaltet diese nur die **mAP** für alle Klassen kombiniert, sowie für die Klasse „Nummernschild“.

		Klasse	Training				
			ohne Verbesserung	Negativ	Gamma Correction	CLAHE	Gamma Correction & CLAHE
Evaluierung	ohne Verbesserung	Alle	0,372	0,246	0,376	0,311	0,363
		Nummernschild	0,421	0,279	0,472	0,418	0,261
	Negativ	Alle	0,220	0,374	0,119	0,211	0,284
		Nummernschild	0,095	0,446	0,110	0,348	0,349
	Gamma Correction	Alle	0,374	0,223	0,376	0,327	0,378
		Nummernschild	0,414	0,180	0,489	0,419	0,265
	CLAHE	Alle	0,223	0,237	0,224	0,388	0,407
		Nummernschild	0,152	0,289	0,231	0,454	0,343
	Gamma Correction & CLAHE	Alle	0,223	0,261	0,214	0,378	0,415
		Nummernschild	0,125	0,342	0,165	0,439	0,361

**Tabelle 4.10:** **mAP** des **SSD300** im Vergleich

Die Trainingszeit des **SSD300** beträgt 0,245s, die Detektionszeit pro Bild beträgt 0,83798s.

## 4.4 Dauer der Bildverbesserungen

Tabelle 4.11 zeigt die Dauer der einzelnen Bildverbesserungen. Gemessen wurde über den Validierungsdatensatz und die gemessene Zeit anschließend auf ein Bild herunter gerechnet.

Bildbearbeitung	Dauer (pro Bild in Sekunden)
Negative	0,01689
Gamma Korrektion	0,01265
CLAHE	0,01339

**Tabelle 4.11:** Dauer der einzelnen Bildverarbeitungen pro Bild

## 4.5 Direkter Vergleich aller Algorithmen

Tabelle 4.12 zeigt die besten Ergebnisse für die  $mAP@0.5:0.95$  aller drei Algorithmen auf 25 Epochen. Für YOLOv5 wurde das Modell „yolov5s“ genutzt.

	YOLOv5 unter Gamma Korrektion		Faster R-CNN unter Gamma Korrektion		SSD unter Gamma Korrektion und CLAHE	
Klasse	$mAP@.5$	$mAP@.5:.95$	$mAP@.5$	$mAP@.5:.95$	$mAP@.5$	$mAP@.5:.95$
Alle	0,684	0,545	0,380	0,188	0,672	0,415
Nummernschild	0,979	0,738	0,841	0,492	0,800	0,361
PKW	0,965	0,739	0,418	0,213	0,903	0,577
LKW	0,987	0,919	0,828	0,352	0,902	0,736
Zugmaschine	0,9	0,713	0,174	0,086	0,761	0,422
Anhaenger	0,845	0,637	0,395	0,180	0,765	0,473
PKW Anhaenger	0,0108	0,00432	0,007	0,002	0,503	0,252
Kleintransporter	0,943	0,809	0,357	0,207	0,850	0,616
Besonderheit	0,474	0,259	0,172	0,094	0,311	0,135
Bus	0,314	0,256	0,0	0,0	0,473	0,266
Wohnwagen	0,421	0,38	0,608	0,256	0,455	0,313

**Tabelle 4.12:**  $mAP$  aller Algorithmen

## 5 Diskussion

**YOLOv5.** YOLOv5 ist der für den Einstieg in die Objektdetektion vermutlich am besten geeignete Algorithmus. Er lässt sich vergleichsweise schnell trainieren und eine Beschäftigung mit dem Quellcode ist nicht notwendig, da der Code vollständig von der Firma, die ihn entwickelt hat, veröffentlicht wurde. Das zugehörige GitHub-Repository beinhaltet dabei Code sowohl für Training als auch Evaluierung.

Bei einem Vergleich des Einflusses der Anzahl an Trainingsepochen des YOLOv5, fällt auf, dass vor allem Klassen mit sehr wenigen annotierten Objekten im Trainingsdatensatz, bei Training mit mehr Epochen, wesentlich besser erkannt werden. So unterscheidet sich die  $mAP@0.5:0.95$  der Klasse „PKW Anhaenger“ nach 25 weiteren Trainingsepochen um 0,5051. Klassen mit mehr Trainingsdaten unterscheiden sich weniger stark, und die Klasse „PKW“ hat nach 25 weiteren Trainingsepochen sogar eine kleinere  $mAP$ . Der Unterschied ist allerdings nur sehr gering und wird von den anderen, besseren Ergebnissen ausgeglichen.

Zwei der genutzten Bildverbesserungen führen zu keinem deutlichen Anstieg der Detektionsgenauigkeit; das Nutzen von Negativen und CLAHE führt bei fast allen Klassen zu einem starken Abstieg der  $mAP$ . Lediglich „Wohnwagen“ wurden unter Verwendung von CLAHE wesentlich besser erkannt. Das Anwenden einer Gamma Korrektion führt zu einer insgesamt höheren  $mAP$  im Vergleich zum Training auf unveränderten Bildern. Lediglich „PKW“, „LKW“ und insbesondere „PKW Anhaenger“ werden schlechter erkannt.



**Abbildung 5.1:** Durch CLAHE „zerstörtes“ Nummernschild

Eine Kombination aus Gamma Korrektion mit CLAHE führt ebenfalls nicht zu besseren Ergebnissen. Das Problem mit CLAHE könnte das Rauschen sein, das auf dem veränderten Bild erzeugt wird. Zusätzlich werden bei CLAHE vor allem Nummernschilder teilweise durch die Bearbeitung unkenntlich gemacht, wie in Abbildung 5.1 zu sehen.

Werden trainierte Modelle auf Bildern, die anders bearbeitet wurden, als die, auf denen trainiert wurde, evaluiert, sinkt die  $mAP$  nicht unwesentlich. Die einzige Kombination, bei der Nummernschilder besser erkannt werden als auf komplett unveränderten Bildern, ist das Training auf den Originalbildern und die Evaluierung auf Bildern, bei denen eine Gamma Korrektion durchgeführt wurde. Diese wird allerdings von dem auf Gamma-korrigierten Bildern trainierten und evaluierten Modell übertroffen.

Die verschiedenen Modelle des YOLOv5 führen auch zu unterschiedlichen Ergebnissen. „yolov5n“ ist dabei das kleinste. Die Trainingszeit, sowie die Detektionszeit pro Bild ist bei diesem am geringsten, dafür allerdings auch die Genauigkeit. Es eignet sich aufgrund der geringen Größe am besten für den Einsatz auf Mikrocontrollern oder die Verwendung in Real-Time Systemen. „yolov5s“ ist das nächstgrößere Modell des YOLOv5. Es ist ungefähr viermal größer als „yolov5n“, Trainings- und Detektionsdauer sind jedoch nur etwas weniger als dreimal größer. Der Unterschied in der  $mAP@0,5:0,95$  beträgt nur 0,063. Bei Nummernschildern, der wichtigsten der Klassen, ist der Unterschied geringer. Die besten Ergebnisse mit 25 Trainingsepochen wurden mit dem Modell „yolov5m“

erzielt. Den größten Unterschied machen dabei die Klassen mit den wenigsten Trainingsdaten. Der Unterschied in der  $mAP@0,5:0,95$  der Klasse „Wohnwagen“ beträgt 0,644, den geringsten Unterschied bietet die Klasse „PKW“ mit 0,003. Der Speicherbedarf und die Trainingszeit dieses Modells ist größer als bei den bisherigen Modellen, genauso wie die Detektionsdauer.

Alle drei Modelle übertreffen sowohl mit ihrer  $mAP0.5$  als auch mit ihrer  $mAP0.5:0.95$  die Ergebnisse der YOLOv5 Autoren auf dem „COCO val2017“ Datenset. Eine mögliche Ursache dafür wäre, dass in dem hier genutzten Datensatz insgesamt weniger Klassen vorhanden sind. Dadurch können weniger Klassen, dadurch dass sie schlecht erkannt werden, das Gesamtergebnis negativ beeinflussen. Außerdem nehmen die an den Rastplätzen aufgehängenen Kameras alle Bilder in ähnlicher Perspektive auf, wodurch die wichtigen Objekte immer ungefähr gleich auf den Bildern abgebildet sind und die Modelle lediglich diese Ansicht lernen müssen. Zudem ist die Beleuchtung auf allen hier genutzten Bildern sehr ähnlich, während auf dem „COCO val2017“ Datensatz sehr unterschiedliche Beleuchtungen vorliegen. Vor allem Probleme mit Schatten können auf dem hier verwendeten Datensatz umgangen werden, da bei keinem Bild ausreichend Beleuchtung vorhanden ist um Schatten zu erzeugen.

Kombiniert man dieses Wissen lässt sich vermuten, dass optimale Ergebnisse unter YOLOv5 erreicht werden können unter Verwendung von Gamma Korrektion, möglichst vielen Trainingsepochen sowie einem größtmöglichem Modell. Dies führt zwar zu einer wesentlichen Verlangsamung der Detektion, allerdings arbeitet YOLOv5 mit einer ausreichend hohen Geschwindigkeit, dass das nicht störend sein dürfte.

**Faster R-CNN.** Faster R-CNN hat aufgrund seines Aufbaus als zweistufiger Objektdetektor eine hohe Trainings- und Detektionsdauer. Diese hohe Trainingsdauer ist verantwortlich dafür, dass nur mit 25 Epochen trainiert wurde. Allerdings haben zweistufige Algorithmen oftmals eine höhere Detektionsgenauigkeit als einstufige Ansätze.

Dieser Algorithmus hat ein Problem mit dem Erkennen von Objekten, die nur wenige Trainingsdaten haben. Vor allem die Klassen „PKW Anhaenger“, „Bus“ und „Wohnwagen“ werden fast nie oder nur schlecht erkannt. Lediglich unter Verwendung von Gamma Korrektion werden Wohnwagen überhaupt festgestellt. Besonders mit der genauen Erkennung eines größeren Objektes, wie eines LKWs, hat Faster R-CNN Probleme. So ist die  $mAP@0.5:0.95$  um ungefähr 0,45 kleiner als die  $mAP@0.5$  der Klasse „LKW“. Die  $mAP$  der restlichen Klassen, die erkannt werden, unterscheiden sich dabei lediglich um 0,1 bis 0,2.

Jede Verwendung von Bildverbesserungen sorgt dafür, dass „PKW Anhaenger“ überhaupt erkannt werden. CLAHE sorgt dafür für die beste  $mAP@0.5$  und Negative für die höchste  $mAP@0.5:0.95$  dieser Klasse. „Busse“ werden in keinem Fall erkannt. Nur das Anwenden der Gamma Korrektion führt zu einer erhöhten  $mAP$  im Vergleich zu dem auf den Originalbildern trainierten und evaluierten Modell, allerdings nimmt unter allen Verbesserungen die Genauigkeit für die Detektion von Nummernschildern ab.

Unterscheiden sich die Bildverbesserungsmethoden des Trainings- und des Validierungsdatensatzes, sinkt die  $mAP@0.5:0.95$  für alle Klassen, sowie die  $AP$  der Klasse Nummernschild teilweise beträchtlich. Die schlechtesten Ergebnisse für die  $mAP@0.5:0.95$  ergaben sich für das Modell trainiert auf gamma-korrigierten Bildern und evaluiert auf Negativen.

Die Ergebnisse des Faster R-CNN in dieser Arbeit liegen weit unter den Ergebnissen, welche im Originalpaper erzielt wurden. [33] Für die drei Klassen, welche teilweise nicht erkannt werden, liegt das an den wenig vorhandenen Trainingsdaten und der Ähnlichkeit dieser mit anderen Klassen. Die schlechten Ergebnisse der Klassen „PKW“, „Zugmaschine“, „Anhänger“ und „Kleintransporter“ sind möglicherweise auf die geringe Anzahl von Trainingsepochen zurückzuführen, wobei diese kein Problem für die Detektion von Nummernschildern darstellen. Die geringe mAP der Klasse „Besonderheit“ war zu erwarten, da Besonderheiten auf Fahrzeugen sehr vielfältig und damit nicht einfach generalisierbar sind.

**SSD.** SSD300 übertrifft in dem zugehörigen Originalpaper die mAP des YOLO um 7,9% und ist dabei mehr als doppelt so schnell [37]. Durch die Kombination aus Geschwindigkeit und Genauigkeit wirkt dieser Algorithmus optimal für Objektdetektionsaufgaben. Außerdem lieferte er in ähnlichen Arbeiten sehr gute Ergebnisse [9, 10, 11].

Die Anzahl der Trainingsepochen hat nur einen kleinen Einfluss auf die mAP des SSD. Nach 50 Epochen haben, im Vergleich zur mAP@0.5 nach 25 Trainingsepochen, lediglich die Klassen „PKW Anhaenger“, „Bus“, „Nummernschild“ und „Besonderheit“ einen Anstieg erfahren, bei allen anderen ist die mAP sogar abgefallen. Besonders stark ist dieser Abfall bei der Klasse „Kleintransporter“. Für die mAP@0.5:0.95 ist diese Verteilung anders und der Abfall der mAP der Klassen geringer. Abgenommen haben lediglich die mAP der Klassen „LKW“, „Zugmaschine“, „Anhaenger“, „Besonderheit“ und „Wohnwagen“, den stärksten Anstieg hat die Klasse „Nummernschild“ erfahren, dicht gefolgt von „Bus“.

Das Training und die Evaluierung auf verbesserten Bildern führt dagegen zu meist besseren Ergebnissen; lediglich die mAP@0.5 sinkt unter CLAHE etwas ab, dafür übertrifft die mAP@0.5:0.95 des mit CLAHE trainierten und evaluierten Modells die der unter anderen Bildverbesserungen trainierten Modelle. Nummernschilder werden unter allen drei Bildverbesserungen besser und nur Zugmaschinen schlechter erkannt. Den größten Unterschied machen die Bildverbesserungen auf die mAP@0.5:0.95, diese übertrifft in den meisten Klassen die mAP auf unverbesserten Bildern.

Die Evaluierung auf Bildern die anders bearbeitet wurden, als das evaluierte Modell trainiert wurde, führt zumeist zu schlechteren Ergebnissen. Die einzigen Ausnahmen, die zu Verbesserungen der mAP der Klasse „Nummernschilder“ führt, sind das Training auf Gamma-korrigierten und die Evaluierung auf den Originalbildern, sowie das Training auf mit CLAHE verbesserten Bildern und die Evaluierung auf Bildern die sowohl mit Gamma Korrektion als auch CLAHE bearbeitet wurden.

Herausragende Ergebnisse wurden bei der Evaluierung auf durch CLAHE optimierten Bildern erzielt, mit einem Modell, das mit Bildern, welche mit einer Kombination aus Gamma Korrektion und CLAHE bearbeitet wurden, trainiert wurde. Es übertrifft in fast allen Klassen die meisten anderen Modelle, allerdings ist die mAP für Nummernschilder um 0,003 geringer als für ein auf vollständig unveränderten Bildern trainiertes und evaluiertes Modell. Bessere Ergebnisse wurden lediglich für Training und Evaluierung auf durch Gamma Korrektion und anschließender CLAHE erzielt. Diese übertrifft auch die mAP des für 50 Epochen trainierten Modells. Dafür werden dabei Nummernschilder wesentlich schlechter erkannt, als bei den anderen SSD Modellen.

Die Ergebnisse aller SSD300 Modelle sind in der  $mAP@0.5$  schlechter, als die im Paper zu SSD auf dem VOC2007 und VOC2012 Datensatz erzielten Ergebnisse. Lediglich die Ergebnisse auf dem COCO test-dev2015 Datensatz liegen unter den in dieser Arbeit erreichten Werten. [37] Mögliche Ursachen sind die wesentlich geringere Anzahl von Trainingsepochen und Trainingsdaten.

**Vergleich der Modelle** Bei einem direkten Vergleich der drei Modelle fällt auf, das YOLOv5 insgesamt die besten Ergebnisse liefert und das bei der geringsten Trainings- und Inferenzdauer. Dabei hat selbst das kleinste Modell des YOLOv5 eine höhere  $mAP@0.5:0.95$  als FASTER R-CNN und SSD. Vor allem FASTER R-CNN hat Probleme mit der Detektion von Objekten, welche nur schwach in den Trainingsdaten vertreten sind. Etwas schwächer ist dieses Problem in den Modellen YOLOv5n und YOLOv5s des YOLOv5 vertreten, allerdings behebt das längere Training dieses Problem. Auch das Nutzen eines größeren Modells verringert diese Problematik. Am effektivsten funktioniert SSD für die Detektion von Objekten geringer Trainingsdaten. YOLOv5 und FASTER R-CNN erkennen LKWs und Nummernschilder am besten, was den Erwartungen entspricht, da diese Klassen mit Abstand die meisten Trainingsdaten haben. SSD erkennt LKWs und PKWs am besten, Nummernschilder haben, bei diesem Algorithmus lediglich die vierthöchste  $mAP$ .

FASTER R-CNN ist mit Abstand der Langsamste der drei Algorithmen, sowohl was das Training als auch die Detektion betrifft. Er ist sechsmal langsamer im Finden aller Elemente auf einem Bild als SSD300 und untertrifft in der  $mAP$  sowohl SSD als auch YOLOv5. SSD ist ebenfalls sechsmal langsamer als das kleinste YOLOv5 Modell und wird von diesem ebenfalls in der  $mAP@0.5:0.95$  übertroffen. Der zeitliche Unterschied zwischen dem YOLOv5m-Modell und SSD ist geringer, jedoch wird SSD auch von diesem übertroffen.

Gamma Korrektur mit einem Gamma-Wert von 2 führt bei allen Algorithmen zu einem Anstieg der  $mAP$ . Den stärksten Anstieg erfährt dabei YOLOv5. Sie verringert dabei unter YOLOv5 die Detektionsprobleme der Klassen „Bus“ und „Wohnwagen“, verschlechtert jedoch die sowieso schon schlechte Erkennungsquote der Klasse „PKW Anhaenger“. FASTER R-CNN unter Gamma Korrektur führt zu den insgesamt besten Ergebnissen für die  $mAP@0.5$  der Klasse „Wohnwagen“, jedoch sinkt diese um 0,35 für die  $mAP@0.5:0.95$  und wird damit schlechter als die der anderen Algorithmen. Für SSD ist der Anstieg am geringsten. Die Gamma Korrektur ist diesen Ergebnissen zufolge, entsprechend der Ergebnisse in [4], eine sehr effektive Form der Bildverbesserung. Zusätzlich dazu ist Gamma Korrektur die schnellste der Bildverbesserungen. Ein Problem der Gamma Korrektur ist jedoch das automatische Finden eines optimalen Gamma Wertes, wie bereits in [5] beschrieben.



**Abbildung 5.2:** CLAHE mit „clipLimit“ von 20

CLAHE führt, entgegen der Beobachtungen aus [4], lediglich für den SSD zu einem geringen Anstieg der Detektionsgenauigkeit. Für den YOLO führt CLAHE zum stärkeren Abstieg, für den FASTER R-CNN sinkt die  $mAP$  unter CLAHE weniger stark. Ursache ist vermutlich das Rauschen, welches von CLAHE erzeugt wird. Andere Ergebnisse werden möglicherweise unter Verwendung anderer Werte für die CLAHE-Funktion erzielt, jedoch steigt mit der Erhöhung des „clipLimit“-Parameters auch das Rauschen (wie in Abbildung 5.2).

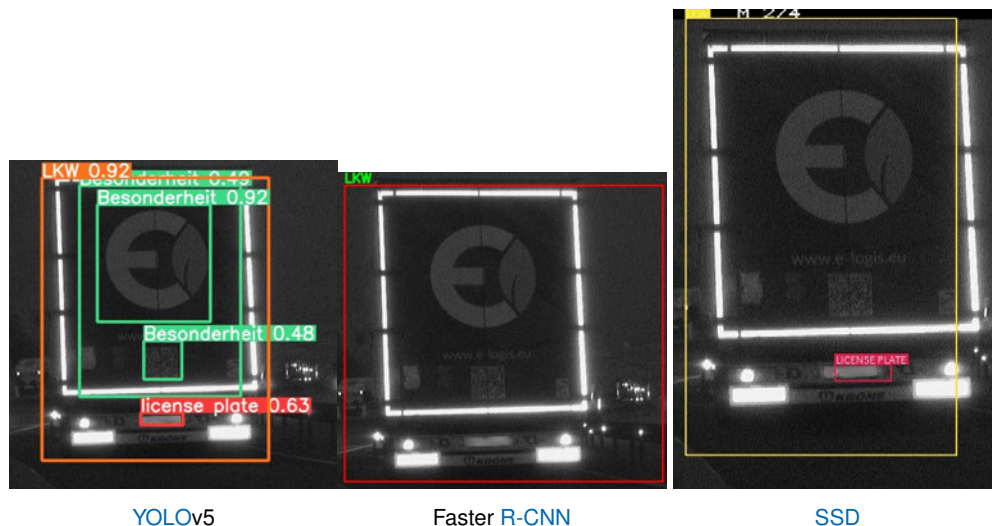
CLAHE ist nur minimal langsamer als die Gamma Korrektur, jedoch relativieren die schlechten Ergebnisse diese Zeit.



Die Kombination von Gamma Korrektur und CLAHE führt für YOLOv5 und FASTER R-CNN zu wesentlich schlechteren Ergebnissen für die mAP. Im Gegensatz dazu werden die besten Ergebnisse des SSD erzielt unter Verwendung der Verbindung beider Bearbeitungen.

Negative führen für YOLOv5 und FASTER R-CNN ebenfalls zu nicht unerheblich schlechteren Ergebnissen, unter SSD sorgen Negative für die beste mAP@0.5 bei einem Training mit 25 Epochen. Die Auswirkungen sind unerwartet, da sich die Ausprägungen der Objekte nicht verändern sollten bei Anwendung von Negativen. Für FASTER R-CNN ist der Unterschied in der mAP@0.5 am höchsten, für die mAP@0.5:0.95 ist der Unterschied zwischen Negativen und den unveränderten Bildern für FASTER R-CNN und YOLOv5 ungefähr gleich. Das Erstellen der Negative mit der in dieser Arbeit beschriebenen Funktion ist mit Abstand die langsamste der Bildverbesserungen.

Entgegen den Erwartungen sind auch die guten Ergebnisse des YOLOv5 für die Detektion von Besonderheiten auf Fahrzeugen. In Abbildung 5.3 ist die Rückseite des selben LKWs gezeigt, zusammen mit allen Klassen und zugehörigen Boxen, welche von den drei Algorithmen gefunden wurden. YOLO hat als einziger Algorithmus das Logo und den QR-Code gefunden. FASTER R-CNN hat lediglich den LKW erkannt. Trotz der guten Ergebnisse des YOLOv5 wird der Algorithmus im



**Abbildung 5.3:** Erkante Besonderheiten durch die verschiedenen Algorithmen

Einsatz wesentlich weniger Besonderheiten feststellen. Dass zumindest das Logo gefunden wurde liegt daran, dass sich ebenfalls Bilder mit diesem Logo im Trainingsdatensatz befinden. Die Erkennung unbekannter Logos wird schwer, wenn nicht sogar unmöglich. QR-Codes können vermutlich auf jedem Bild, auf dem sich einer befindet, festgestellt werden, da QR-Codes einander sehr ähnlich sind. Allerdings könnten sich für eine zuverlässige Detektion dieser Codes zu wenige Trainingsdaten mit QR-Codes im Datensatz befinden.

Eine Besonderheit die von allen Algorithmen erkannt wird, ist der „Polizei“-Schriftzug auf der Seite eines Polizeiautos, beispielhaft durch YOLOv5 gezeigt in Abbildung 5.4.



**Abbildung 5.4:** Aufgenommenes Polizeiauto

Eine weitere Besonderheit, deren richtige Detektion essentiell ist, wird in Abbildung 5.5 gezeigt. Diese Besonderheit ist ein Schild auf der Rückseite eines LKWs, welches große Ähnlichkeit mit einem

Nummernschild hat. Wird dieses fälschlicherweise als Kennzeichen erkannt, steigt die Zeit, welche investiert werden muss, um die automatische Auswertung nutzen zu können. YOLOv5 und SSD haben dieses Schild richtig als Besonderheit klassifiziert, lediglich Faster R-CNN interpretiert es als Nummernschild.

Trotz dessen sind in Abbildung 5.5 zwei Nummernschilder durch YOLOv5 erkannt wurden. Von diesen beiden ist lediglich das Nummernschild mit der Wahrscheinlichkeit 0,89 richtig, das Andere ist lediglich eine reflektierende Fläche auf der Rückseite des LKWs. Diese wird auch von SSD und Faster R-CNN als Nummernschild erkannt. Für diese Regionen gibt die OCR-Anwendung einen leeren String zurück. Detektierte Nummernschilder, die keinen Text enthalten, sind nicht notwendigerweise falsch erkannt. Aufgrund ungünstiger Beleuchtung sind Nummernschilder teilweise vollkommen unkenntlich.



Abbildung 5.5: Objekte auf einem LKW erkannt durch YOLOv5

Ein weiteres Problem für die, der Detektion folgende, Auswertung der Nummernschilder stellt das ungenaue Erkennen von Nummernschildern dar. Ein Beispiel für dieses Problem ist in Abbildung 5.3, in dem zu SSD gehörigem Bild, gezeigt. In diesem Bild ist die Ungenauigkeit so groß, dass ein Zeichen des Nummernschildes abgeschnitten wird. Dieses Problem ist bei SSD am stärksten ausgeprägt, die Differenz zwischen der  $mAP@0.5$  und der  $mAP@0.5:0.95$  ist bei diesem Algorithmus am größten. Unter YOLOv5 konnte diese Form der Ungenauigkeit, bei manueller Durchsicht verschiedener Detektionsergebnisse, lediglich festgestellt werden, wenn das Nummernschild zum Teil überdeckt war.



## 6 Anwendung

Auf Grundlage der Ergebnisse wurde eine Anwendung entwickelt, welche die Auswertung gegebener Datensätze automatisiert. Der Code dieser Anwendung ist unter <https://github.com/1d438ef6/Wildtierkameraauswertungsprogramm> zu finden. <sup>11</sup>

### 6.1 Theorie und Umsetzung

Die graphische Oberfläche der Anwendung wurde unter Verwendung von Qt5 für Python entwickelt. Die Oberfläche ist dabei sehr einfach gehalten. Sie besteht aus einer Ordnerauswahl um das Quellverzeichnis der Daten-DVD anzugeben, einer Dateiauswahl um den Speicherort für das Ergebnis der Auswertung festzulegen, einem Knopf, der die Einstellungsmöglichkeiten öffnet und einem Start-Knopf.

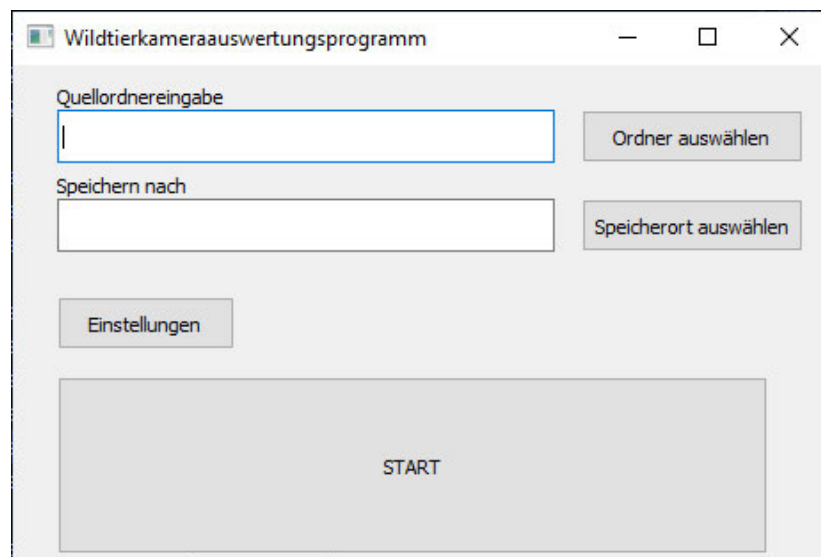


Abbildung 6.1: Graphische Oberfläche der Anwendung

Die Einstellungsmöglichkeiten beinhalten das genutzte Modell, diverse Bildverbesserungsmöglichkeiten, sowie den Pfad zu einer Texterkennungssoftware. Gespeichert werden diese in Form einer json-Datei. Der Code ist dabei darauf ausgelegt, möglichst viele verschiedene torchvision Modelle ausführen zu können, damit bei einer Verbesserung eines Modells nicht die gesamte Anwendung aktualisiert werden muss, sondern lediglich das genutzte Modell. Jedoch liegt bei diesen das Hauptaugenmerk auf YOLOv5 Modellen, da YOLOv5 in dieser Arbeit die besten Ergebnisse erzielt hat.

Als Texterkennungssoftware dient Tesseract. Tesseract wurde aufgrund der einfachen Implementierung gewählt und aufgrund der Tatsache, dass es schneller läuft als EasyOCR [10]. Auf die Implementation einer OCR-Anwendung auf Basis eines Objektdetektors wie SSD wurde aufgrund des Zeitaufwandes verzichtet.

<sup>11</sup>zum Zeitpunkt der Abgabe dieser Arbeit ist die Anwendung nicht vollständig entwickelt

Um mit Tesseract optimale Ergebnisse zu erzielen, muss der Ausschnitt, welcher der Software übergeben wird, gewissen Bearbeitungsschritten unterzogen werden. Die besten Ergebnisse werden erzielt mit Bildern von einer Auflösung von wenigstens 300 DPI. Es kann die Genauigkeit erhöhen das Bild zu vergrößern. Auch sollte das Bild in ein schwarz-weiß Bild überführt werden. Tesseract berechnet dies intern, liefert dabei aber nicht immer die optimalen Ergebnisse. Buchstaben die fettgeschrieben oder zu schmal sind, vermindern ebenfalls die Genauigkeit von Tesseract und müssen verbreitert oder verschmälert werden. Zu stark geneigter Text wird ebenfalls nicht besonders gut erkannt, eine Rotation behebt dieses Problem. Da die Nummernschilder von YOLOv5 sehr genau ausgeschnitten werden, kann das Hinzufügen eines weißen Rahmens die Genauigkeit erhöhen.<sup>12</sup>

Das Betätigen des Start-Knopfes führt zum Aufruf einer Funktion, welche jedes Bild im Datensatz auswertet und dabei alle auf dem aktuellen Bild befindlichen und relevanten Objekte detektiert. Ist ein detektiertes Objekt ein Nummernschild, wird der entsprechende Bildausschnitt an die OCR-Software übergeben. Die detektierten Klassen, die Koordinaten der zugehörige Boxen, der Zeitstempel des Bildes, die aufnehmende Kamera sowie der Dateiname des Bildes und falls möglich/notwendig der Text des Nummernschildes werden in einem „Pandas-Dataframe“ gespeichert und anschließend als csv-Datei abgelegt. Diese kann für weitere Analysen der Daten genutzt werden.

Diese weiteren Analysen beinhalten das Erstellen einer Excel-Datei mit drei Tabellen, einer für Einfahrt, einer für Ausfahrt und die dritte beinhaltet die Aufenthaltsdauer der einzelnen Fahrzeuge auf dem Rastplatz. Dabei müssen für das Erstellen die Bilderreihen, welche das gleiche Fahrzeug beinhalten, zu einer Zeile zusammengefasst werden; ein Vorgang, bei dem unvollständige Nummernschilder vervollständigt werden können. Bilder auf denen nichts erkannt wurde, werden in einen Extraordner kopiert, für die manuelle Auswertung durch einen Sachbearbeiter. Zusätzlich wird jedes Bild in der entsprechenden Zeile der Excel-Datei verlinkt, zur Vereinfachung manuell notwendiger Fehlerausbesserungen.

## 6.2 Probleme

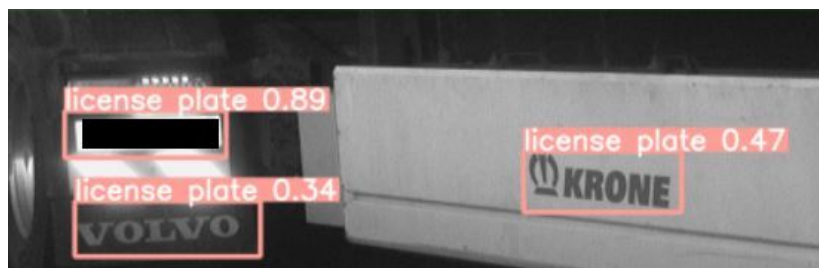
Für die Funktion der Anwendung sind gewisse Probleme aufgetreten. So ist der vom Objektdetektionsalgorithmus vorgeschlagene Bereich für ein Nummernschild nicht immer exakt genug. Resultierend werden teilweise Zeichen des Nummernschildes abgeschnitten.

Es gibt im Datensatz Nummernschilder, welche sich lediglich durch Betrachten einer Bilderreihe zusammensetzen lassen. Teilweise werden einzelne Zeichen des Nummernschildes überdeckt, sodass sie entweder gar nicht oder falsch erkannt werden.

Zusätzlich gibt es Bilder, auf denen Objekte fälschlicherweise als Nummernschild erkannt werden. In Abbildung 6.2 ist ein Beispiel dieses Problems zu sehen. Das Bild beinhaltet neben dem richtigen Nummernschild noch zwei falsche. Dabei ist es nicht möglich, die Nummernschilder mit den geringsten Wahrscheinlichkeiten zu entfernen, da es auch Bilder mit mehreren Nummernschildern geben kann, z.B. wenn zwei Fahrzeuge direkt hintereinander fahren oder wenn ein PKW auf der Ladefläche eines Transporters steht. Auch das Einführen eines Schwellwerts, ab dem ein erkanntes Nummernschild auch als richtiges Nummernschild angenommen wird, führt nicht zu dem gewünschten Ergebnis, da es auch richtig erkannte Nummernschilder mit einer niedrigen Wahrscheinlichkeit

<sup>12</sup><https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html>

gibt. Ein möglicher Ansatz für die Lösung dieses Problems ist ein Ausschluss von Nummernschildern



**Abbildung 6.2:** fälschlicherweise erkannte Nummernschilder

anhand ihrer Positionen und der Neigungsrichtung des Textes auf diesen. Allerdings würde dies vermutlich auch richtige Nummernschilder entfernen.

Einige Fahrzeuge können von den Objektdetektoren nicht eindeutig einer Klasse zugeordnet werden und bekommen mehrere mögliche Klassen zugeordnet. Aufgefallen ist das mit „PKW“ und „Kleintransporter“ sowie „Kleintransporter“ und „LKW“. Diese bekommen in der Ausgabe die Klasse „unsicher“.

Schwierigkeiten bereiten auch Fahrzeuge, welche von der Kamera an der Ausfahrt erfasst werden, den Rastplatz allerdings nicht verlassen, sondern lediglich Rückwärts einparken. Diese werden als „Parkplatz verlassen“ interpretiert und verfälschen so die Zeiten, in denen dieses Fahrzeug tatsächlich auf dem Parkplatz stand.

Eine wirkliche Lösung dieser Probleme existiert noch nicht. Aktuell muss jedes auftretende Problem durch das manuelle Sichten der Daten durch einen Sachbearbeiter gelöst werden.

Ein Problem für die Entwicklung der Anwendung stellt das Bekommen des Aufnahmedatums und der Aufnahmezeit dar. So sind die Informationen in den zur Verfügung gestellten Bildern zu Erstellungszeitpunkt sowie der Änderungszeitpunkt abweichend von dem im Bild angegebenen Aufnahmezeitpunkt. Gelöst wird dieses Problem durch das Ausschneiden des Bereiches mit dem Zeitpunkt und der Übergabe des Ausschnitts an die verwendete [OCR-Software](#).



## 7 Zusammenfassung

### 7.1 Fehlerdiskussion

Die vermutlich größte Fehlerquelle stellen falsch annotierte Klassen auf Bildern aufgrund besonders schlechter Beleuchtung dar. So existieren im Datensatz Bilder mit einem Fahrzeug, das ein Bus oder ein LKW sein kann. Diese können auch nicht von einem Menschen komplett richtig annotiert werden. Eine Umgehung dieses Problems ist möglich, indem alle zu schlecht beleuchteten Bilder aus dem Datensatz entfernt werden. Allerdings könnte es passieren, dass dadurch die Detektionsergebnisse auf dunkleren Bildern verschlechtert werden.

Aufgrund der geringen Anzahl einiger Klassen im Datensatz, wie in Abbildung 2.4 bereits zu sehen, können diese Klassen nicht besonders gut erkannt werden. Um das Problem zu beheben, werden weitere Bilder benötigt. Diese werden allerdings erst nach erneutem Einsatz der Kameras verfügbar sein. Aktuell werden deshalb z.B. keine Motorräder erkannt.

Eine weitere mögliche Fehlerquelle sind die sich im Laufe der Nacht verändernden Beleuchtungsverhältnisse. So sind die Aufnahmen zwischen 05:30 und 06:00 Uhr sehr gut ausgeleuchtet, während alle anderen Bilder sehr dunkel sind. Das verändert die Sichtbarkeit oder Ausprägtheit möglicherweise relevanter Merkmale. Die Bildverbesserungsmethoden adressieren dieses Problem zum Teil, weitere Lösungsmöglichkeiten können Bestandteil zukünftiger Arbeiten sein.

Eine in zukünftigen Arbeiten zu beachtende Fehlerquelle ist die Benennung einiger Elemente im gespeicherten Faster R-CNN Modell. In verschiedenen Versionen von PyTorch weichen die Namen einzelner Layer ab, wodurch diese vor Verwendung des Modells erst modifiziert werden müssen. Auch das Verwenden von Cuda und einer Grafikkarte kann dazu führen, dass diese Benennung abweicht.

### 7.2 Zukünftige Arbeit

Für zukünftige Arbeiten könnte der Datensatz an Bildern vergrößert werden. Somit könnten geringe Erkennungsgenauigkeiten von Klassen wie Wohnwagen oder Bussen verbessert werden. Außerdem können weitere Fahrzeugtypen inkludiert werden, wie z.B. Motorräder.

Ein weiterer Ansatz wären das Implementieren und Testen anderer Bildverbesserungsmöglichkeiten. So ließen sich Deep-Learning Methoden nutzen, wie etwa Super Resolution<sup>13</sup> oder ein Modell um Graustufenbilder einzufärben<sup>14</sup>. Eine weitere Möglichkeit wäre das Nutzen von Denoising-Techniken, vor allem für durch Histogramm-Bearbeitungen, wie CLAHE, erstellte Bilder. Außerdem wäre eine Implementation verschiedener Gamma Werte für die Gamma Korrektion möglich, in welcher getestet wird, welche Auswirkungen diese Veränderung auf das endgültige Detektionsergebnis hat. Es ließe sich auch ein Algorithmus implementieren, der die Helligkeit eines Bildes ermittelt und anschließend einen darauf angepassten Gamma Wert vorschlägt.

<sup>13</sup><https://deepai.org/machine-learning-model/torch-srgan>

<sup>14</sup><https://deepai.org/machine-learning-model/colorizer>

Statt des Nutzens von SSD300, könnte der SSD512 verwendet werden. Dieser ist langsamer und größer als der SSD300, besitzt jedoch auch eine etwas größere mAP. Zusätzlich könnte die veränderte VGG-16 Struktur als Backbone gegen eine Schnellere oder eine mit größerer Detektionsgenauigkeit ausgetauscht werden. Andere Arbeiten [9, 10] implementieren dafür eine MobileNet-Struktur und erzielen damit gute Ergebnisse.

Eine für die entwickelte Anwendung sinnvolle Weiterentwicklung ist das Schreiben einer eigenen OCR-Anwendung oder das Trainieren eines eigenen Modells für eine bestehende OCR-Anwendung, speziell für die Detektion von Buchstaben auf Nummernschildern. Auch ließen sich speziell auf den Bildausschnitt mit dem Nummernschild weitere Bildverbesserungen anwenden, welche darauf ausgelegt sind, die Zeichen auf dem Nummernschild hervorzuheben, damit diese anschließend von der OCR-Software besser erkannt werden. Vor der Übergabe an die OCR-Anwendung könnte eine Verbesserung der Boxen für detektierte Nummernschilder erfolgen, um das Problem abgeschnittener Zeichen zu beheben. Dafür müsste der groben Detektion ein Algorithmus folgen, der die Eckpunkte der Box des Nummernschildes an die grob detektierte Fläche anpasst, ähnlich dem in [9] vorgestellten Algorithmus.

Die Detektion von Logos, QR-Codes und weiteren auf den Fahrzeugen befindlichen Informationen kann in zukünftigen Arbeiten ebenfalls behandelt werden. Die Detektion von QR-Codes lässt sich dabei über einen einfachen Objektdetektor realisieren. Die Detektion von Logos theoretisch auch, allerdings wird der Detektor lediglich in den Trainingsdaten bereits bekannte Logos erkennen. Sinnvoller wäre ein Ansatz, bei dem auf erkannten Fahrzeugen nach Abweichungen gesucht wird. Die Erkennung weiterer Informationen wie z.B. Schriftzügen mit Firmennamen oder Telefonnummern lassen sich finden, indem aus dem entsprechenden Bild alle Nummernschilder entfernt werden und es anschließend einer OCR-Anwendung übergeben wird.

Durch die Verwendung von OpenVINO kann die Größe der Modelle verringert und damit auch die Rechenzeit für die Detektion verkleinert werden. OpenVINO nimmt dabei Modelle verschiedener Bibliotheken und konvertiert diese in das OpenVINO-Format.

### 7.3 Fazit

Objektdetektion, vor allem unter schlechten Aufnahmebedingungen, stellt ein wichtiges Forschungsfeld im Bereich des Machine Learnings dar. So haben aktuelle Algorithmen wie YOLOv5 eine hohe Detektionsgenauigkeit, jedoch erreicht bisher kein Algorithmus die Fähigkeit eines Menschen Objekte zu erkennen. Auch Algorithmen zur Verbesserung von Bildern, zum Erhöhen des Kontrastes oder der Helligkeit stellen ausreichend Material für zukünftige Arbeiten.

Algorithmen für automatische Nummernschilderkennung bieten ein breites Forschungsspektrum. So haben viele bisherige Arbeiten die Erkennung von Nummernschildern mit Eigenschaften wie der rechteckigen Form gestützt. Jedoch bietet diese Technik viele Schwachstellen und ist damit für den breiten Einsatz nicht besonders gut geeignet. Kennzeichen werden nicht erkannt wenn sie nicht vollständig zu sehen sind oder eine andere Form haben, z.B. weil das Fahrzeug aus einem anderen Land stammt. Aktuelle Objektdetektionsalgorithmen haben eine Genauigkeit, die hoch genug ist, um sie sinnvoll in Nummernschilderkennungssystemen zu verwenden. Außerdem ist die Geschwindigkeit dieser auch hoch genug, um die Erkennung in Real-Time durchzuführen.

Diese Arbeit hat drei Objektdetektionsalgorithmen für die Nummernschilderkennung in einem System zur automatischen Auswertung von Bildern mit Fahrzeugen, welche auf eine Autobahnraststätte ein- und ausfahren, verglichen. Dabei sind die meisten Bilder während der Nacht aufgenommen, weshalb zusätzlich verschiedene Bildverbesserungen getestet wurden.

Es erwies sich die Verbesserung durch Gamma Korrektur am Nützlichsten für die Erhöhung der Genauigkeit. Die Verbesserung durch CLAHE, welche in anderen Arbeiten sehr gute Ergebnisse gebracht hat, hat in dieser Arbeit lediglich für [SSD](#) bessere Ergebnisse erzielt. Die anderen Algorithmen haben dadurch die meisten Klassen schlechter erkannt. Die Verwendung von Negativen führt ebenfalls nicht zu einer signifikanten Verbesserung der Detektionsgenauigkeit von [YOLOv5](#) und [Faster R-CNN](#), lediglich [SSD](#) erzielte unter Negativen bessere Ergebnisse.

[YOLOv5](#) übertrifft in seiner Genauigkeit alle anderen verwendeten Algorithmen, [SSD](#) ist jedoch besser in der Erkennung von Objekten mit geringer Anzahl an Trainingsdaten. [Faster R-CNN](#) untertrifft mit Abstand beide Single-Stage Detektoren in seiner Genauigkeit.

Trotz der bereits sehr guten Ergebnisse der in dieser Arbeit genutzten [YOLOv5](#) Modelle und wegen der weniger guten Ergebnisse der anderen Algorithmen, müssen vor allem für Objekte, bei welchen eine genaue Erkennung essentiell ist, wie [z.B.](#) Nummernschildern, nach einer groben Detektion, Verbesserungen dieser implementiert werden.





# Glossar

**Ground Truth Box** Eine Ground Truth Box ist eine Box, von welcher bekannt ist, was sie beinhaltet. Sie ist das optimale gewünschte Ergebnis einer Problemstellung und wurde (in dieser Arbeit) durch die Annotierung der Bilder manuell festgelegt. [56].

**Lernrate** Rate mit der die Gewichte angepasst werden; je größer dieser Wert, desto stärker werden die Gewichte angepasst [13].

**Long Short-Term Memory** Ein LSTM-Netzwerk ist eine Struktur für ein Neuronales Netz, bei dem vorherige Eingaben das Ausgabenergebnis des Netzes beeinflussen. Die Stärke des Einflusses nimmt mit der Zunahme des Abstandes der Eingaben ab, wobei die Abnahme ein während des Trainings lernbarer Parameter ist. [14].

**Momentum Faktor** Momentum ist eine Methode, die hilft, SGD in die richtige Richtung zu beschleunigen. Dieser Wert wird standardmäßig auf 0,9 gesetzt. [15].

**Non Maximum Supression NMS** ist ein für die meisten Objektdetektoren relevanter Nachbearbeitungsschritt. Diese Detektoren liefern für ein auf einem Bild vorhandenem Objekt meist mehrere Boxen zurück. Allerdings ist dies so gut wie immer unerwünscht, weshalb nur die besten Boxen für ein Objekt ausgewählt werden müssen. Das erfolgt über Non Maximum Supression.

Erreicht wird das, indem der Algorithmus über alle gefundenen Objekte, geordnet nach Klassen, läuft. Für alle Objekte einer Klasse mit einem IoU über einem bestimmten Schwellenwert, wird dabei lediglich das Objekt mit der höchsten Wahrscheinlichkeit behalten. [57].

**Optimierer** Der Optimierer dient der Aktualisierung der Gewichte des neuronalen Netzes. [14].

**Pooling Layer** Pooling Layer verringern in Netzwerken die Auflösung der Eingabe der vorherigen Schicht. Dadurch wird eine schnellere Berechnung erreicht, zusätzlich dient es der Vorbeugung einer Überanpassung des Netzwerkes. [14] Zudem verringern Pooling Layer die Abhängigkeit der Features von ihren Positionen. [32].

**RoI** Eine „Region of Interest“, ist ein Bereich auf einem Bild der ein Objekt beinhalten könnte..

**SVM** Eine SVM ist ein binärer Klassifizierer, welcher in einem n-Dimensionalen Raum, durch Erzeugen einer „Hyperplane“ Punkte/Objekte zweier Klassen separiert. [55].

**Weight Decay/L2 Strafe** Das Nutzen eines „Weight Decays“ verringert das Problem des Overfitting eines Modells und reguliert dessen Gewichte. Das erfolgt unter Anwendung der L2 Norm. [58].



## Literaturverzeichnis

- [1] Shan Du u. a. „Automatic License Plate Recognition (ALPR): A State-of-the-Art Review“. In: *IEEE Transactions on Circuits and Systems for Video Technology* 23.2 (2013), S. 311–325. DOI: [10.1109/TCSVT.2012.2203741](https://doi.org/10.1109/TCSVT.2012.2203741).
- [2] Lukas Ewecker u. a. „Provident Vehicle Detection at Night for Advanced Driver Assistance Systems“. In: *CoRR* abs/2107.11302 (2021). arXiv: [2107.11302](https://arxiv.org/abs/2107.11302). URL: <https://arxiv.org/abs/2107.11302>.
- [3] Muh Ismail. „License plate Recognition for moving vehicles case : At night and under rain condition“. In: *2017 Second International Conference on Informatics and Computing (ICIC)*. 2017, S. 1–4. DOI: [10.1109/IAC.2017.8280649](https://doi.org/10.1109/IAC.2017.8280649).
- [4] Mohamed Aladem, Stanley Baek und Samir A. Rawashdeh. „Evaluation of Image Enhancement Techniques for Vision-Based Navigation under Low Illumination“. In: *Journal of Robotics* 2019 (2018). URL: <https://www.hindawi.com/journals/jr/2019/5015741/>.
- [5] Yuxuan Xiao u. a. „Making of Night Vision: Object Detection Under Low-Illumination“. In: *IEEE Access* 8 (2020), S. 123075–123086. DOI: [10.1109/ACCESS.2020.3007610](https://doi.org/10.1109/ACCESS.2020.3007610).
- [6] Yirui Wu u. a. „Edge Computing Driven Low-Light Image Dynamic Enhancement for Object Detection“. In: *IEEE Transactions on Network Science and Engineering* (2022), S. 1–1. DOI: [10.1109/TNSE.2022.3151502](https://doi.org/10.1109/TNSE.2022.3151502).
- [7] Shyang-Lih Chang u. a. „Automatic license plate recognition“. In: *IEEE Transactions on Intelligent Transportation Systems* 5.1 (2004), S. 42–53. DOI: [10.1109/TITS.2004.825086](https://doi.org/10.1109/TITS.2004.825086).
- [8] Piyush Batra u. a. „A Novel Memory and Time-Efficient ALPR System Based on YOLOv5“. In: *Sensors* 22.14 (2022). ISSN: 1424-8220. DOI: [10.3390/s22145283](https://doi.org/10.3390/s22145283). URL: <https://www.mdpi.com/1424-8220/22/14/5283>.
- [9] Yiwen Luo u. a. „Multiple Chinese Vehicle License Plate Localization in Complex Scenes“. In: *2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC)*. 2018, S. 745–749. DOI: [10.1109/ICIVC.2018.8492857](https://doi.org/10.1109/ICIVC.2018.8492857).
- [10] Ninad Awalgaoonkar, Prashant Bartakke und Ravindra Chaugule. „Automatic License Plate Recognition System Using SSD“. In: *2021 International Symposium of Asian Control Association on Intelligent Robotics and Industrial Automation (IRIA)*. 2021, S. 394–399. DOI: [10.1109/IRIA53009.2021.9588707](https://doi.org/10.1109/IRIA53009.2021.9588707).
- [11] Riel D. Castro-Zunti, Juan Yépez und Seok-Bum Ko. „License plate segmentation and recognition system using deep learning and OpenVINO“. In: *IET Intelligent Transport Systems* 14.2 (2020), S. 119–126. DOI: <https://doi.org/10.1049/iet-its.2019.0481>. eprint: <https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-its.2019.0481>. URL: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-its.2019.0481>.
- [12] tzutalin. *labelimg*. URL: <https://github.com/tzutalin/labelImg>. (zugegriffen: 30.06.2022).
- [13] Tariq Rashid. *Neuronale Netze selbst programmieren*. Übers. von Frank Langenau. 1. Aufl. Heidelberg: dpunkt.verlag GmbH, 2017. ISBN: 978-3-96009-043-4.

- [14] Ian Pointer. *PyTorch für Deep Learning. Anwendungen für Bild-, Ton- und Textdaten entwickeln und deployen*. Übers. von Marcus Fraaß. 1. Aufl. Heidelberg: dpunkt.verlag GmbH, 2021. Kap. 2, 3 und 5, sowie Vorwort. ISBN: 978-3-96009-134-9.
- [15] Sebastian Ruder. „An overview of gradient descent optimization algorithms“. In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: <http://arxiv.org/abs/1609.04747>.
- [16] Saad Albawi, Tareq Abed Mohammed und Saad Al-Zawi. „Understanding of a convolutional neural network“. In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, S. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.
- [17] Mayank Mishra. *Convolutional Neural Networks, Explained*. URL: <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939>. (zugegriffen: 23.07.2022).
- [18] Joseph Redmon u. a. „You Only Look Once: Unified, Real-Time Object Detection“. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, S. 779–788. DOI: 10.1109/CVPR.2016.91.
- [19] Alexey Bochkovskiy, Chien-Yao Wang und Hong-Yuan Mark Liao. „YOLOv4: Optimal Speed and Accuracy of Object Detection“. In: *CoRR* abs/2004.10934 (2020). arXiv: 2004.10934. URL: <https://arxiv.org/abs/2004.10934>.
- [20] Peiyuan Jiang u. a. „A Review of Yolo Algorithm Developments“. In: *Procedia Computer Science* 199 (2022). The 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 & 2021): Developing Global Digital Economy after COVID-19, S. 1066–1073. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2022.01.135>. URL: <https://www.sciencedirect.com/science/article/pii/S1877050922001363>.
- [21] Mihir Rajput. *YOLO V5 — Explained and Demystified*. URL: <https://pub.towardsai.net/yolo-v5-explained-and-demystified-4e4719891d69>. (zugegriffen: 01.07.2022).
- [22] WZMIAOMIAO. *YOLOv5 (6.0/6.1) brief summary*. URL: <https://github.com/ultralytics/yolov5/issues/6998>. (zugegriffen: 15.08.2022).
- [23] Chien-Yao Wang u. a. „CSPNet: A New Backbone that can Enhance Learning Capability of CNN“. In: *CoRR* abs/1911.11929 (2019). arXiv: 1911.11929. URL: <http://arxiv.org/abs/1911.11929>.
- [24] Joseph Redmon und Ali Farhadi. „YOLOv3: An Incremental Improvement“. In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: <http://arxiv.org/abs/1804.02767>.
- [25] Tsung-Yi Lin u. a. „Feature Pyramid Networks for Object Detection“. In: *CoRR* abs/1612.03144 (2016). arXiv: 1612.03144. URL: <http://arxiv.org/abs/1612.03144>.
- [26] Shu Liu u. a. „Path Aggregation Network for Instance Segmentation“. In: *CoRR* abs/1803.01534 (2018). arXiv: 1803.01534. URL: <http://arxiv.org/abs/1803.01534>.
- [27] Bing Xu u. a. „Empirical Evaluation of Rectified Activations in Convolutional Network“. In: *CoRR* abs/1505.00853 (2015). arXiv: 1505.00853. URL: <http://arxiv.org/abs/1505.00853>.
- [28] Ross Girshick u. a. „Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Juni 2014.
- [29] Sharif Elfouly. *R-CNN System*. URL: [https://miro.medium.com/max/700/1\\*Vc3wR0tbND2gYD-pi85cPw.png](https://miro.medium.com/max/700/1*Vc3wR0tbND2gYD-pi85cPw.png). (zugegriffen: 04.07.2022).

- [30] Seungkwan Lee, Suha Kwak und Minsu Cho. „Universal Bounding Box Regression and Its Applications“. In: *Computer Vision - ACCV 2018*. Springer International Publishing, 2019. ISBN: 978-3-030-20876-9. DOI: [https://doi.org/10.1007/978-3-030-20876-9\\_24](https://doi.org/10.1007/978-3-030-20876-9_24).
- [31] Ross Girshick. „Fast R-CNN“. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Dez. 2015.
- [32] Priyanshi Sharma. *Everything about Pooling layers and different types of Pooling*. URL: <https://iq.opengenus.org/pooling-layers/>. (zugegriffen: 16.08.2022).
- [33] Shaoqing Ren u. a. „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks“. In: *Advances in Neural Information Processing Systems*. Hrsg. von C. Cortes u. a. Bd. 28. Curran Associates, Inc., 2015. URL: <https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf>.
- [34] Ahmed Fawzy Gad. *Faster R-CNN Explained for Object Detection Tasks*. URL: <https://blog.paperspace.com/faster-r-cnn-explained-object-detection/>. (zugegriffen: 04.07.2022).
- [35] Kaiming He u. a. „Mask R-CNN“. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Okt. 2017.
- [36] M. Holschneider u. a. „A Real-Time Algorithm for Signal Analysis with the Help of the Wavelet Transform“. In: *Wavelets*. Hrsg. von Jean-Michel Combes, Alexander Grossmann und Philippe Tchamitchian. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, S. 286–297. ISBN: 978-3-642-75988-8.
- [37] Wei Liu u. a. „SSD: Single Shot MultiBox Detector“. In: *CoRR* abs/1512.02325 (2015). arXiv: [1512.02325](http://arxiv.org/abs/1512.02325). URL: <http://arxiv.org/abs/1512.02325>.
- [38] sgrvinod. *a PyTorch Tutorial to Object-Detection*. URL: <https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection>. (zugegriffen: 05.07.2022).
- [39] Raman Maini und Himanshu Aggarwal. „A Comprehensive Review of Image Enhancement Techniques“. In: *CoRR* abs/1003.4053 (2010). arXiv: [1003.4053](http://arxiv.org/abs/1003.4053). URL: <http://arxiv.org/abs/1003.4053>.
- [40] P. Janani, J. Premaladha und K. S. Ravichandran. „Image Enhancement Techniques: A Study“. In: *Indian Journal of Science and Technology* vol 8(22) (2015). URL: <https://sciresol.s3.us-east-2.amazonaws.com/IJST/Articles/2015/Issue-22/Article18.pdf>.
- [41] DEVI WILLIEAM ANGGARA u. a. „GRAYSCALE IMAGE ENHANCEMENT FOR ENHANCING FEATURES DETECTION IN MARKER-LESS AUGMENTED REALITY TECHNOLOGY“. In: *Journal of Theoretical and Applied Information Technology* Vol.98. No 13 (2020). URL: [http://eprints.utm.my/id/eprint/90015/1/MohdShafryMohdRahim2020\\_GrayscaleImageEnhancementforEnhancingFeaturesDetection.pdf](http://eprints.utm.my/id/eprint/90015/1/MohdShafryMohdRahim2020_GrayscaleImageEnhancementforEnhancingFeaturesDetection.pdf).
- [42] Thomas Wood. *What is the Softmax Function?* URL: <https://deepai.org/machine-learning-glossary-and-terms/softmax-layer>. (zugegriffen: 09.08.2022).
- [43] Ahmed Fawzy Gad. *Evaluating Deep Learning Models: The Confusion Matrix, Accuracy, Precision, and Recall*. URL: <https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/>. (zugegriffen: 17.07.2022).
- [44] Ahmed Gad. *Evaluating Object Detection Models Using Mean Average Precision*. URL: <https://www.kdnuggets.com/2021/03/evaluating-object-detection-models-using-mean-average-precision.html>. (zugegriffen: 17.07.2022).

- [45] Ahmed Fawzy Gad. *Evaluating Object Detection Models Using Mean Average Precision (mAP)*. URL: <https://blog.paperspace.com/mean-average-precision/>. (zugegriffen: 31.07.2022).
- [46] *Timeline of optical character recognition*. URL: [https://en.wikipedia.org/wiki/Timeline\\_of\\_optical\\_character\\_recognition](https://en.wikipedia.org/wiki/Timeline_of_optical_character_recognition). (zugegriffen: 26.08.2022).
- [47] Noman Islam, Zeeshan Islam und Nazia Noor. „A Survey on Optical Character Recognition System“. In: *CoRR* abs/1710.05703 (2017). arXiv: 1710.05703. URL: <http://arxiv.org/abs/1710.05703>.
- [48] *FE-Schrift*. URL: <https://de.wikipedia.org/wiki/FE-Schrift>. (zugegriffen: 26.08.2022).
- [49] A. Ahmed Biyabani, Salman A. Al-Salman und Khalaf S. Alkhalaf. „Embedded real-time bilingual ALPR“. In: *2015 International Conference on Communications, Signal Processing, and their Applications (ICCSPA'15)*. 2015, S. 1–6. DOI: [10.1109/ICCSPA.2015.7081311](https://doi.org/10.1109/ICCSPA.2015.7081311).
- [50] Priyanto Hidayatullah u. a. „Optical Character Recognition Improvement for License Plate Recognition in Indonesia“. In: *2012 Sixth UKSim/AMSS European Symposium on Computer Modeling and Simulation*. 2012, S. 249–254. DOI: [10.1109/EMS.2012.95](https://doi.org/10.1109/EMS.2012.95).
- [51] R. Smith. „An Overview of the Tesseract OCR Engine“. In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Bd. 2. 2007, S. 629–633. DOI: [10.1109/ICDAR.2007.4376991](https://doi.org/10.1109/ICDAR.2007.4376991).
- [52] *Tesseract*. URL: <https://github.com/tesseract-ocr/tesseract>. (zugegriffen: 31.07.2022).
- [53] ultralytics. *Yolov5*. URL: <https://github.com/ultralytics/yolov5>. (zugegriffen: 01.07.2022).
- [54] Sovit Ranjan Rath. *Custom Object Detection using PyTorch Faster RCNN*. URL: <https://debuggercafe.com/custom-object-detection-using-pytorch-faster-rcnn/>. (zugegriffen: 04.08.2022).
- [55] William S Noble. „What is a support vector machine?“ In: *Nature Biotechnology* 24.12 (Dez. 2006), S. 1565–1567.
- [56] *Ground truth*. URL: [https://en.wikipedia.org/wiki/Ground\\_truth](https://en.wikipedia.org/wiki/Ground_truth). (zugegriffen: 19.08.2022).
- [57] Jatin Prakash. *Non Maximum Suppression: Theory and Implementation in PyTorch*. URL: <https://learnopencv.com/non-maximum-suppression-theory-and-implementation-in-pytorch/>. (zugegriffen: 02.08.2022).
- [58] Ajitesh Kumar. *Weight Decay in Machine Learning: Concepts*. URL: <https://vitalflux.com/weight-decay-in-machine-learning-concepts/>. (zugegriffen: 22.08.2022).

## Eidesstattliche Erklärung

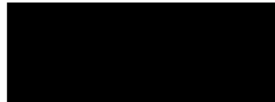
Hiermit versichere ich – Leon D. Wutke – an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

Mittweida, 12. September 2022

Ort, Datum

A solid black rectangular box used to redact the signature of Leon D. Wutke.

Leon D. Wutke