Verwendung von Deep Learning für die Erkennung von Werbung am Straßenrand in Dashcam-Videos

Mario Voithofer

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2019

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 25. Juni 2019

Mario Voithofer

Inhaltsverzeichnis

Kurzfassung

Diese Arbeit befasst sich mit der Thematik der generischen Erkennung von Werbung mit Deep Learning. Dazu wird ein Überblick über die notwendigen Grundlagen in Bezug auf neuronale Netzwerke geschaffen. Dieser fokussiert sich auf Objekterkennungsmethoden und *Convolutional Neural Networks*, die zur Bildklassifizierung verwendet werden. Darüber hinaus werden häufig verwendete Deep Learning Frameworks kurz vorgestellt. Um gängige Ansätze in diesem Themengebiet besser verstehen zu können, wird auf ähnliche Arbeiten eingegangen. In der Folge wird das Thema der Arbeit „Die Erkennung von Werbung in Dashcam-Videos“ erläutert. Verschiedene Objekterkennungsmethoden und die Auswahl eines Deep Learning Frameworks werden näher beschrieben. Um den Vergleich der Ergebnisse dieser Arbeit mit anderen Lösungen zu ermöglichen, wird gezeigt, wie solche Objekterkennungsmethoden evaluiert werden können. Für die Umsetzung des Konzeptes wurden zwei verschiedene Objekterkennungsmethoden implementiert. Diese Methoden sind *Mask R-CNN* und *YOLOv3*. Der Vergleich der beiden bietet sich an, da deren Fokus auf verschiedenen Bereichen liegt. Mask R-CNN zielt darauf ab, bestmögliche Ergebnisse auf Kosten der Analysezeit zu erzielen. YOLO versucht, die Analysezeit von Bildern so gering wie möglich zu halten. Um den erforderlichen Trainings-Datensatz möglichst klein zu gestalten, wird *Transfer Learning* eingesetzt. Mit dieser Technik wird das neuronale Netzwerk mit einem anderen, größeren Datensatz antrainiert bzw. mit bereits trainierten Gewichtungen initialisiert. Erst anschließend wird der eigentliche Trainings-Datensatz verwendet. Um möglichst gute Ergebnisse zu erzielen, wurde Mask R-CNN mit verschiedenen Kombinationen von Netzwerkarchitekturen, Initialgewichtungen und Lernraten trainiert. Dieser Prozess wird in der Arbeit ausführlich beschrieben. Die Ergebnisse werden anhand von Testbildern aus einem Validierungs-Datensatz evaluiert. Dabei erreichte Mask R-CNN eine Durchschnittsgenauigkeit von 94.5%, YOLOv3 erlangte eine Genauigkeit von 60.0%. Des Weiteren werden die Ergebnisse von Mask R-CNN und YOLO anhand von Schnappschüssen von Dashcam-Videos gezeigt. Abschließend werden die Ergebnisse interpretiert.

Abstract

This thesis looks upon the problem of recognizing advertising generically using deep learning. For this purpose, an overview of the necessary basics concerning neural networks is created. The overview focuses on *Convolutional Neural Networks*, which are used for image classification and object recognition methods. Furthermore, commonly used deep learning frameworks are briefly introduced. In order to better understand conventional approaches in this area, similar projects will be reviewed. Afterward, the subject of this thesis – the recognition of advertising in dashcam videos – will be explained. The taken choices of object recognition methods and the selection of a deep learning framework will be discussed. To allow comparison of the results of this thesis with other solutions, a description of how the object recognition methods were evaluated is specified. To realize the proposed concept, two different methods were implemented. These methods are *Mask R-CNN* and *YOLOv3*. The comparison of these methods is valuable because their center is in different areas. Mask R-CNN aims to generate the best possible results at the expense of processing time, whereby YOLO tries to keep the processing time as short as possible. In order to keep the required training data set as small as possible, *Transfer Learning* is used. Using this technique the neural network is pre-trained with a different broader data set or is initialized with already trained weights before using the dedicated training dataset. To archive the best possible results for the Mask R-CNN approach, different compositions of network architectures, initial weightings, and learning rates have been tested. This process is described in detail throughout this thesis. The results are evaluated using test images from a validation data set. Thereby Mask R-CNN achieved an average accuracy of 94.5% and YOLOv3 achieved an accuracy of about 60.0%. Furthermore, the results of Mask R-CNN and YOLOv3 are presented using snapshots of dashcam videos. The observed results are then discussed in more detail and interpreted.

Kapitel 1

Einleitung

Im Bereich *Computer Vision* haben sich in letzter Zeit immer öfter künstliche neuronale Netzwerke zur Lösung von Aufgabenstellungen durchgesetzt. Vor allem im Bereich autonomes Fahren, wie zum Beispiel beim Erkennen von Straßenschildern, hat sich diese Technik stark etabliert. Abgesehen davon gibt es weitere Anwendungsgebiete. In dieser Arbeit wird darauf eingegangen, wie man Deep Learning zum Erkennen von Werbung am Straßenrand anhand von Dashcam-Videos verwenden kann.

1.1 Problemstellung

Es wird beschrieben, wie Werbung am Straßenrand innerhalb von Bildern und Videos generisch erkannt werden kann. Als direkter Anwendungsfall sollen in dieser Arbeit Dashcam-Videos fokussiert werden. Das kann heikel sein, da sich das Aussehen von Werbungen oft stark voneinander unterscheidet und so eine Generalisierung sehr schwierig ist. Dafür müssen eine passende Methode zum Erkennen von Objekten in einem Bild sowie eine passende Netzwerkarchitektur und akzeptable Hyperparameter gefunden werden. Um dieses Netzwerk trainieren zu können, muss manuell eine Sammlung von Bildern erstellt und annotiert werden, da es keine gelabelte Datenbank mit Werbungen gibt. Ein neuronales Netzwerk benötigt dafür im Normalfall eine große Menge, weshalb ein Weg gefunden werden muss, um diese Sammlung möglichst klein zu halten.

1.2 Zielsetzung

Ziel dieser Arbeit ist es, einen Überblick über künstliche neuronale Netzwerke und Methoden zur Objekterkennung innerhalb von Bildern oder Videos zu schaffen. Dafür sollen eine oder mehrere Methoden gefunden werden, um Werbung generisch erkennen zu können. Um den Umfang der Bilder, die zum Trainieren des Netzwerkes benötigt werden, möglichst gering zu halten, soll Transfer Learning eingesetzt werden. Die erhaltenen Resultate können so verglichen und diskutiert werden. Wie das Ergebnis aussehen soll, ist in Abbildung 1.1 dargestellt.

Abbildung 1.1: Videoschnappschuss eines Dashcam-Videos [68] mit erkannten Werbungen. Diese werden anhand eines Begrenzungsrahmens sowie der zugehörigen Kategorie und der Zuordnungswahrscheinlichkeit markiert.

1.3 Aufbau der Arbeit

Im Einleitungskapitel wird der Hintergrund sowie die Zielsetzung dieser Arbeit kurz erklärt. Die restliche Arbeit ist wie folgt aufgebaut.

- Kapitel 2 gibt einen Überblick über die Theorie hinter neuronalen Netzwerken und Objekterkennung. Weiters werden die bekanntesten Deep Learning Frameworks kurz beschrieben.
- In Kapitel 3 werden einige vergleichbare Arbeiten erläutert.
- In Kapitel 4 wird ein Konzept beschrieben, was in dieser Arbeit behandelt wird.
- Kapitel 5 beschreibt, wie der praktische Teil dieser Arbeit umgesetzt wurde.
- Kapitel 6 gibt einen Überblick über die erhaltenen Resultate.
- In Kapitel 7 werden die Umsetzung und die Ergebnisse interpretiert sowie die gesamte Arbeit abschließend zusammengefasst.

Kapitel 2

Grundlagen

In diesem Kapitel wird die grundlegende Theorie erläutert, auf der die Arbeit aufbaut. Diese umfasst eine allgemeine Einführung in künstliche neuronale Netze, was den Aufbau und das Prinzip des Trainierens beinhaltet. Weiters werden Performance-Probleme erläutert sowie das Anwenden von Methoden zur Regularisierung, um die Leistung des Netzes zu verbessern. Danach wird auf Convolutional Neural Networks (CNN) im Allgemeinen näher eingegangen und gängige CNNs und Methoden zum Erkennen von Objekten in einem Bild beschrieben. Am Ende des Kapitels werden die bekanntesten Deep Learning Frameworks kurz erläutert und es wird auf den GPU Support eingegangen.

2.1 Künstliche neuronale Netze

Ein künstliches neuronales Netz ist ein von einem Computer erstelltes Modell, das auf einem vereinfachten Prinzip des Gehirns basiert. Wie das menschliche Gehirn, verbessert sich ein künstliches neuronales Netzwerk über die Zeit. Für diesen Lernvorgang werden Trainingsdaten benötigt [97].

2.1.1 Neuronen und Netzwerk-Struktur

Ein künstliches neuronales Netzwerk besteht aus unzähligen Neuronen, die unterschiedlichen Layern innerhalb dieses Netzwerkes zugeordnet sind. Diese Layer sind grundsätzlich in drei Kategorien unterteilt: in einen *Input Layer*, mehrere sogenannte *Hidden Layer* und einen *Output Layer*. In Abbildung 2.1 ist ein simples künstliches neuronales Netzwerk dargestellt. Neuronen können als eine Art Knoten in diesem Netzwerk angesehen werden. Jeder dieser Knoten repräsentiert einen Wert. Die Werte der Knoten im Input Layer entsprechen den Informationen, die dem Netz zugeführt werden. In den Hidden Layern sind diese Knoten jeweils mit einem oder mehreren Knoten aus dem vorhergehenden und dem nachfolgenden Layer durch gewichtete Verbindungen miteinander verknüpft. Diese Gewichtungen verändern sich während des Lernvorganges des Netzwerkes. Der Wert, der ein Neuron von einer Verbindung zu einem Neuron im vorhergehenden Layer erhält, wird als Input bezeichnet. Ein Input ist das Produkt aus dem Wert des jeweiligen Knotens im vorhergehenden Layer und der Gewichtung der Verbindung zwischen den beiden Neuronen. Auf Grundlage aller Inputs eines Neurons wird der Wert dieses Neurons im Hidden Layer berechnet. Dieser errechnet sich für gewöhn-

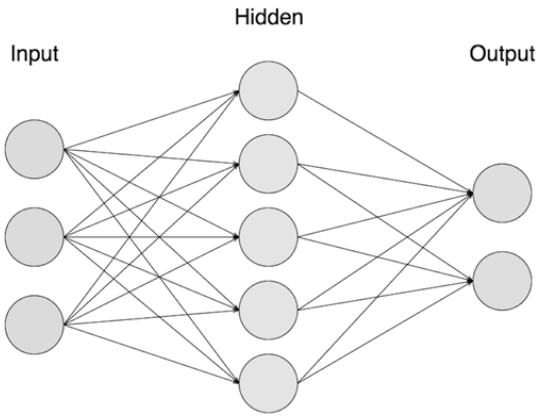


Abbildung 2.1: Grundlegende Architektur eines künstlichen neuronalen Netzes, die anhand eines Input Layers, eines Hidden Layers und eines Output Layers samt deren Verbindungen grafisch dargestellt ist. Grafik adaptiert aus [105].

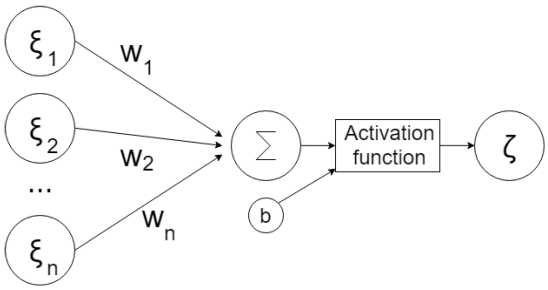


Abbildung 2.2: Berechnung des Outputs eines Neurons auf Basis der Neuronen-Werte im vorherigen Layer in Kombination mit den Gewichtungen der Verbindungen zwischen den Neuronen, des Bias und der Aktivierungsfunktion. Bildquelle [5].

lich aus der Summe aller Inputs. Der Wert kann noch durch einen sogenannten Bias (s. Abschnitt 2.1.3) ergänzt, bzw. durch eine Aktivierungsfunktion (s. Abschnitt 2.1.2) angepasst werden. Diese Berechnung ist in Abbildung 2.2 grafisch dargestellt. ξ sind die Werte der Neuronen im vorhergehenden Layer, w sind die Gewichtungen der Verbindungen, b ist der Bias (s. Abschnitt 2.1.3) und ζ ist der Output bzw. der errechnete Wert des Neurons. Je nach den erlernten Gewichtungen der Verbindungen breiten sich die Werte der Neuronen im Netzwerk unterschiedlich stark aus [33]. Der Output Layer ist der letzte Layer und liefert die Kategorisierung des Netzwerkes. Die Anzahl der Neuronen im Output Layer definiert, wie viele verschiedene Kategorien das Netzwerk erkennen kann. Jeder Kategorie wird genau ein Neuron des Output Layers zugewiesen. Soll ein Input zum Beispiel durch das Netzwerk in eine von zehn Kategorien eingeteilt werden, so besteht dieser Output Layer auch aus genau zehn Neuronen. Der Input wird der Kategorie zugewiesen, deren Neuron den höchsten Wert aufweist [111].

2.1.2 Aktivierungsfunktionen

Aktivierungsfunktionen sind Funktionen, um den Wert eines Neurons anzupassen, bzw. in einem gewissen Wertebereich zu halten. Diese werden verwendet, um zu bestimmen, wie groß der Einfluss des Inputs auf den Output ist. So kann dadurch zum Beispiel auch vermieden werden, dass gewisse Neuronen einen um Vieles höheren Wert erreichen als andere und somit einen zu hohen Einfluss auf die Berechnung von Werten in späteren Layern haben. Grundsätzlich lassen sich Aktivierungsfunktionen in lineare und nicht lineare Aktivierungsfunktionen unterteilen. Dafür können jeweils viele verschiedene Funktionen verwendet werden [111]. Die bekanntesten inklusive ihrer mathematischen Funktion werden in Abbildung 2.3 dargestellt. Die Grafiken wurden mithilfe der Webseite FOOPLOT[1] erstellt.

2.1.3 *Bias*

Als Bias wird ein optionaler Wert bezeichnet, der nach errechnetem Wert aus den Inputs bei der Anwendung der Aktivierungsfunktion addiert werden kann. Der Bias kann sowohl im negativen als auch im positiven Wertebereich liegen und somit dazu verwendet werden, die Aktivierungsfunktion nach rechts oder links zu verschieben. Das heißt, der Bias gibt einen Schwellenwert an, ab welchem Input ein Neuron je nach Aktivierungsfunkion abgefeuert wird. Auch dieser Wert kann sich während des Lernvorgangs verändern [80].

2.1.4 Initialisierung

Damit ein Netzwerk lernfähig ist, müssen die Werte der Neuronen initialisiert werden. Das wird im Normalfall mit einem zufälligen Wert realisiert. Es hat sich bewährt, diese Werte normalverteilt mit den Parametern

$$\mu = 0, \qquad \sigma = \sqrt{\frac{1}{n}} \tag{2.1}$$

zu wählen, wobei n die Anzahl der Verbindungen zum vorhergehenden Layer ist. Alternativ kann für die Standardabweichung auch

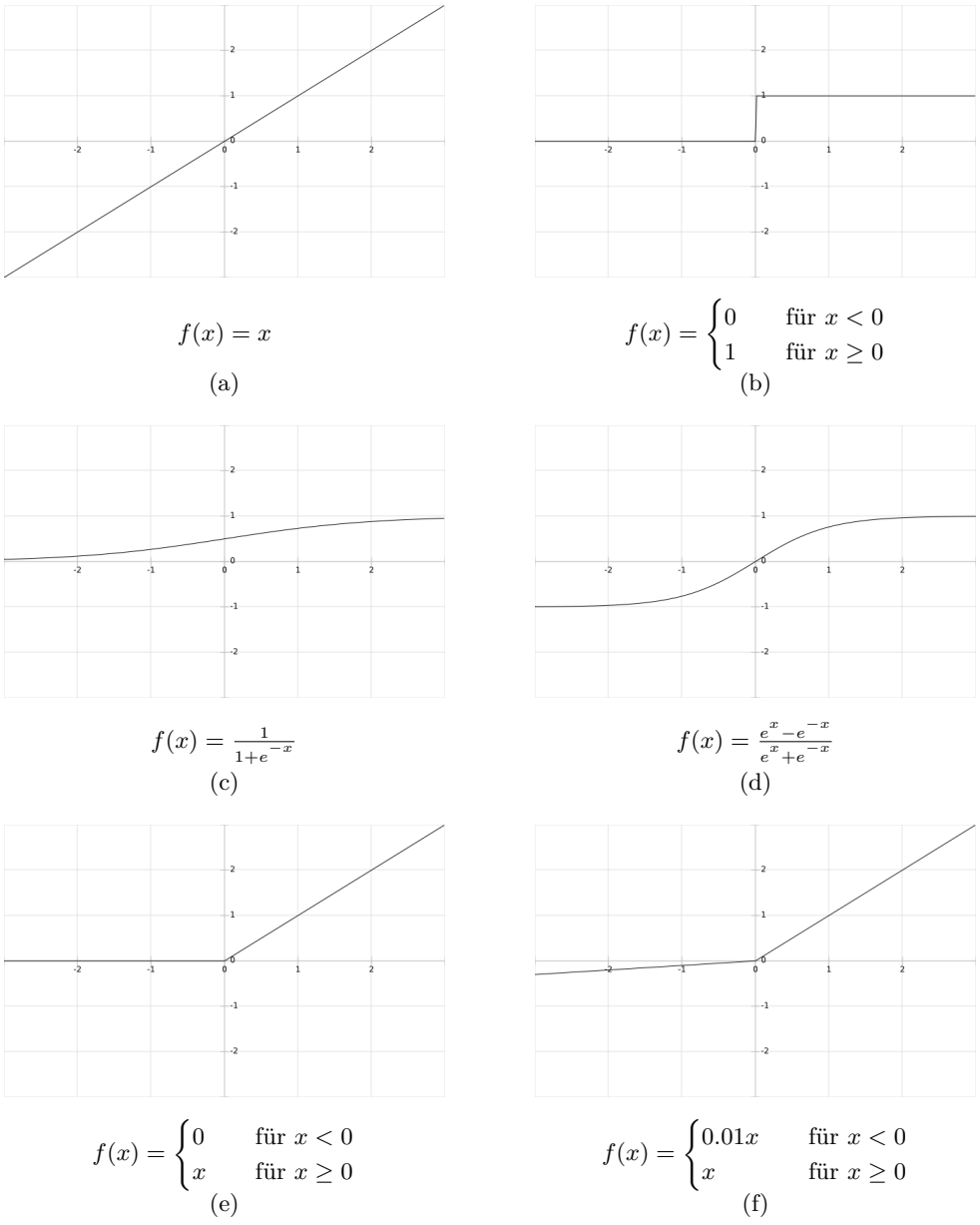
$$\sigma = \sqrt{\frac{2}{n_{\text{in}} + n_{\text{out}}}} \tag{2.2}$$

verwendet werden, mit n_{in} die Anzahl der Verbindungen zum vorhergehenden Layer und n_{out} die Anzahl der Verbindungen zum nachfolgenden Layer [76]. Diese Initialisierung wird von den meisten Deep Learning Frameworks automatisch umgesetzt.

2.2 Trainieren von neuronalen Netzen

Damit ein künstliches neuronales Netzwerk vernünftige Ergebnisse liefert, muss dieses nach der Initialisierung auf die gewünschte Art trainiert werden. Hierfür wird ein sogenannter Trainings-Datensatz verwendet. Das Trainieren von neuronalen Netzen ist

[1] http://fooplot.com



$f(x) = x$

(a)

$$f(x) = \begin{cases} 0 & \text{für } x < 0 \\ 1 & \text{für } x \geq 0 \end{cases}$$

(b)

$f(x) = \frac{1}{1+e^{-x}}$

(c)

$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

(d)

$$f(x) = \begin{cases} 0 & \text{für } x < 0 \\ x & \text{für } x \geq 0 \end{cases}$$

(e)

$$f(x) = \begin{cases} 0.01x & \text{für } x < 0 \\ x & \text{für } x \geq 0 \end{cases}$$

(f)

Abbildung 2.3: Grafische Darstellung diverser Aktivierungsfunktionen. Die X-Achse repräsentiert den Inputwert und die Y-Achse den Outputwert. *Identity* (a), *Binary Step* (b), *Sigmoid* (c), *Tanh* (d), *Rectified Linear Units* (ReLu) (e), *Leaky ReLu* (f).

ein Optimierungsproblem, bei dem der Fehler zwischen dem Output des Netzwerkes und dem richtigen Ergebnis minimiert werden muss. Zur Fehlerberechnung wird eine sogenannte Kostenfunktion (s. Abschnitt 2.2.3) verwendet. Für den Trainingsvorgang benötigt man einen umfangreichen Datensatz mit meist mehreren tausend Einträgen, damit das Netz möglichst exakte Ergebnisse liefern kann.

2.2.1 *Supervised* versus *Unsupervised Learning*

Grundsätzlich kann das Lernen von neuronalen Netzen in Supervised und Unsupervised Learning unterteilt werden. Beim Supervised Learning ist für jede Datei im Trainings-Datensatz die richtige Klassifizierung mittels eines Labels bereits bekannt, man spricht von einem gelabelten Datensatz. Der Input wird durch das neuronale Netzwerk gesendet, der Output mit dem bereits bekannten Label des Inputs verglichen und der Fehler berechnet. Dieser Vergleich kann durch die Kombination einer Softmax-Funktion im Output Layer und dem One-Hot Encoding Labeling-Prinzip (s. Abschnitt 2.2.2) durchgeführt werden. Die Fehlerberechnung geschieht aufgrund einer definierten Kostenfunkion (s. Abschnitt 2.1.2). Durch eine Optimierungsfunktion (s. Abschnitt 2.2.4) wird der auftretende Fehler der berechneten Wahrscheinlichkeit der Klassifizierung verkleinert [33]. Hierbei werden mithilfe von Backpropagation (s. Abschnitt 2.2.5) die Gewichtungen des gesamten Netzes aktualisiert. Während des Lernvorganges kann zusätzlich mit einem sogenannten Validierungs-Datensatz überprüft werden, ob das Netzwerk zu Overfitting (s. Abschnitt 2.3.1) oder Underfitting (s. Abschnitt 2.3.2) neigt [95].

Beim Unsupervised Learning ist der Trainings-Datensatz nicht gelabelt. Das bedeutet, das neuronale Netz muss selbst eine gewisse Struktur durch das Analysieren gewisser Features in den Daten finden. Dieses Prinzip wird zum Beispiel beim Clustern von Daten verwendet.

Weiters gibt es auch die Möglichkeit, die beiden Varianten zu kombinieren. Hier werden anfangs die gelabelten Daten verwendet, um das Netzwerk grundlegend zu trainieren. Den nicht gelabelten Daten werden anschließend durch das bereits grundlegend trainierte Netzwerk Labels zugewiesen. Mit diesen neuen „pseudo-gelabelten“ Daten wird das Netzwerk weiter trainiert.

2.2.2 *One-Hot Encoding* und *Softmax*

One-Hot Encoding ist ein Prinzip zum Labeln von Datensätzen. Dazu wird jeder Kategorie ein Vektor von Nullen und Einsen zugeordnet. Die Dimension der Vektoren entspricht der Anzahl der möglichen unterschiedlichen Kategorien. Jeder Kategorie wird ein bestimmter Index in diesem Vektor zugewiesen. Das bedeutet, der Vektor dieser Kategorie besteht aus lauter Nullen, nur an dem Index der jeweiligen Kategorie befindet sich im Vektor eine Eins [84]. Da mit diesem Prinzip die Summe der Werte des Vektors Eins ergibt, kann der Vektor auch als gegebene, klar definierte Wahrscheinlichkeitsverteilung der jeweiligen Kategorie gewertet werden. Sollte im Nachhinein eine Kategorie hinzukommen, hat es den Vorteil, dass bei den vorhanden Vektoren nur eine Dimension mit dem Wert Null hinzugefügt werden muss. Das Prinzip eines One-Hot Vektors ist in Abbildung 2.4 dargestellt.

Wird nun bei den Werten des Output Layers des Netzwerkes eine Softmax-Funktion angewendet, so wird die Summe der Werte der Neuronen im Output Layer auf Eins normalisiert [101], dies kann zum Beispiel so aussehen:

$$\begin{bmatrix} 1.68 \\ 0.36 \\ \vdots \\ 0.024 \end{bmatrix} \rightarrow \text{Softmax} \rightarrow \begin{bmatrix} 0.7 \\ 0.15 \\ \vdots \\ 0.01 \end{bmatrix}. \tag{2.3}$$

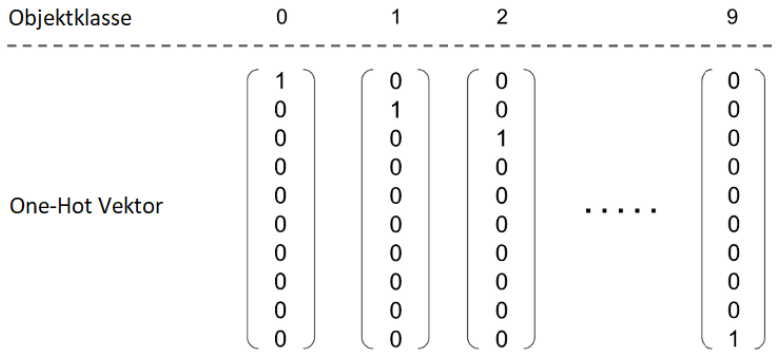


Abbildung 2.4: Grafische Darstellung der One-Hot Vektor Methode. Jeder Objektklasse wird ein One-Hot Vektor zugewiesen, die sich durch die Dimension, an der sich die 1 befindet, unterscheiden. Da die Summe der Elemente des Vektors 1 ergibt, kann dieser als Wahrscheinlichkeitsverteilung der Klassenzuweisung angesehen werden. Angepasst von Bildquelle [107].

Das bedeutet weiter, dass die normalisierten Werte des Output Layers auch als ein Vektor einer Wahrscheinlichkeitsverteilung angesehen werden kann. Die Dimension dieses Vektors entspricht der Summe der Neuronen im Output Layer.

2.2.3 Kostenfunktionen

Die Kostenfunktion definiert den Fehler zwischen dem Ergebnis des neuronalen Netzes und dem korrekten Ergebnis. Dafür werden der Output Vektor des Netzwerkes und der One-Hot Encoding Vektor miteinander verglichen. Dies ist möglich, da die Anzahl der Dimensionen beider Vektoren identisch ist. Am oben gezeigten Beispiel sieht dieser Vergleich so aus:

$$\begin{bmatrix} 0.7 \\ 0.15 \\ \vdots \\ 0.01 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{2.4}$$

Links ist der Vektor des Output Layers des Netzwerkes zu sehen. Der rechte Vektor entspricht dem One-Hot Vektor des Labels. Für die Art der Fehlerberechnung gibt es viele verschiedene Varianten. Eine der meistangewendeten Methoden ist die Verwendung der mittleren quadratischen Abweichung (L2-Distanz). Diese ist als

$$C = \frac{1}{2n} \sum_{\xi} \|\zeta - d(\xi)\|^2 \tag{2.5}$$

definiert, wobei n die Anzahl der Trainingsdaten, ζ der Ergebnisvektor des Inputs ξ und $d(\xi)$ der gewünschte Output für den Input ξ ist. Eine Alternative wäre auch die Verwendung der Manhattan-Fehlerabweichung (L1-Distanz) oder der Kreuzentropie [95].

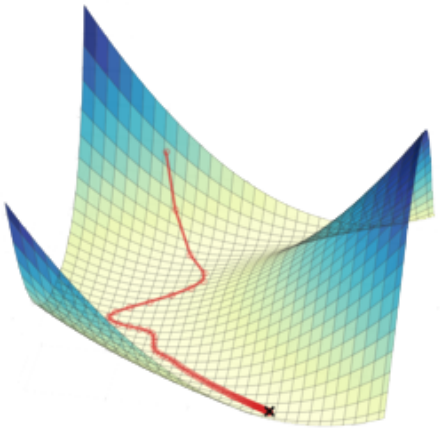


Abbildung 2.5: Stark vereinfachte, grafische Darstellung der Stochastic Gradient Descent Optimization im dreidimensionalen Raum. In Rot wird der Verlauf des Fehlers dargestellt. Ziel ist es, durch Anpassung der Gewichtungen der Fehler den Punkt mit dem niedrigsten Wert zu erreichen. Grafik adaptiert aus [87].

2.2.4 Optimierungsfunktion

Sobald die Kostenfunktion definiert ist, ist das Ziel des Lernprozesses, den berechneten Fehler zu minimieren. Für diesen Minimierungsvorgang werden sogenannte *Optimizers* oder Optimierungsfunktionen verwendet. Eine gängige Methode ist, hierfür *Gradient Descent*[2] Algorithmen zu verwenden. Durch diese Methode werden die Gewichtungen und Biases im Netzwerk schrittweise aktualisiert, um ein Fehlerminimum zu finden. Die Aktualisierung der Werte wird mittels *Backpropagation* (s. Abschnitt 2.2.5) realisiert. Eine stark vereinfachte Fehlerminimierung im dreidimensionalen Raum wird in Abbildung 2.5 gezeigt. Hier soll der Fehler, der in Rot dargestellt ist, den niedrigsten Punkt der Fläche finden. Der niedrigste Punkt stellt das Fehleroptimum dar. In Neuronalen Netzen wird diese Berechnung in einem wesentlich größeren Dimensionsausmaß durchgeführt. Für die Berechnung des Gradienten wird meist ein Verfahren verwendet, das als *Stochastic Gradient Descent* (SGD) bezeichnet wird. Anhand dieses Gradienten kann ausgesagt werden, auf welche Art sich die Werte verändern müssen, damit sich der Fehler minimiert. Weitere mögliche Optimierungsverfahren sind AdaGrad, AdaDelta, RMSProp und Adam [52].

2.2.5 *Backpropagation*

Um ein Netzwerk trainieren zu können, muss der Gradient der Kostenfunktion verarbeitet werden. Diese Verarbeitung muss möglichst schnell ablaufen, um in einem neuronalen Netz verwendbar zu sein. Ein solcher schneller Algorithmus ist Backpropagation. Dieser ist in modernen neuronalen Netzen die meistangewendete Methode. Im Grunde ist Backpropagation nur das Anwenden der Kettenregel auf die Ableitung der Kostenfunktion (s. Abschnitt 2.2.3). Der Kernpunkt dieses Algorithmus ist die Erkenntnis, dass der Gradient von einem Layer über Terme des Gradienten des nächsten Layers ausgedrückt

[2] Dt. *Gradientenverfahren*

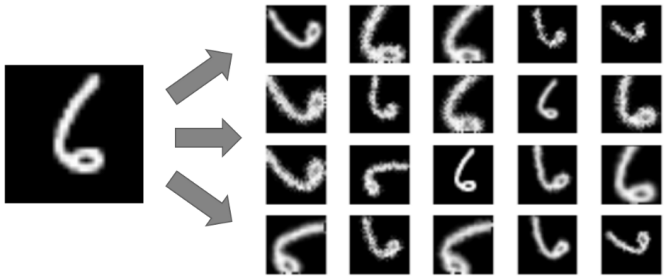


Abbildung 2.6: Visualisierung von Data Augmentation anhand eines Beispielbildes. Dieses wird durch verschiedenste Transformationen verändert, um so ein größeres Datenset zum Trainieren des Netzwerkes zu erhalten. Bild adaptiert aus [100].

werden kann. Das bedeutet, wenn der Gradient eines Layers bestimmt ist, können durch rekursive Ausbreitung die Gradienten im gesamten Netzwerk bestimmt werden. Auf die genauen mathematischen Hintergründe wird in dieser Arbeit nicht eingegangen [53].

2.2.6 *Data Augmentation*

Data Augmentation wird verwendet, um einen kleinen Trainings-Datensatz mit möglichst wenig Aufwand und ohne neue Daten zu benötigen, zu vergrößern. Dabei werden lediglich vorhandene Daten leicht verändert, um neue Daten zu erhalten. Ein Bild kann beispielsweise gespiegelt, gedreht oder verzerrt werden. Weiters kann auch nur ein zugeschnittener Teil des ursprünglichen Bildes als neue Datei verwendet werden. Wird dies bei jeder Datei des Datensatzes erledigt, so kann dessen Größe vervielfacht werden, ohne neue Daten zu benötigen [60]. Ein solches Beispiel wird in Abbildung 2.6 dargestellt. Deep Learning Frameworks stellen diese Funktionalität meist zur Verfügung.

2.2.7 Lernrate

Die Lernrate ist ein Hyperparameter, der beim Lernvorgang angegeben werden kann. Er sagt aus, wie schnell ein künstliches neuronales Netzwerk neue Informationen lernen soll. Das heißt, der Faktor bestimmt, wie stark bereits früher Erlerntes behalten wird und wie stark neue Erkenntnisse auf das Netzwerk Einfluss haben. Grundsätzlich liegt dieser Faktor in einem Wertebereich zwischen 0.01 und 0.0001 und kann auch optional während des Lernvorgangs angepasst werden [95].

2.2.8 *Batch Size* und *Epochs*

Batch Size und Anzahl der Epochs sind Hyperparameter beim Trainieren eines neuronalen Netzes. Die Batch Size sagt aus, wie viele Daten gleichzeitig durch das Netzwerk gespielt werden, bevor dieses aktualisiert wird. Dieser Wert liegt im Normalfall bei zirka 100 und hängt von der Größe des Datensets ab [104].

Die Anzahl der Epochs sagt aus, wie oft der gesamte Datensatz für das Training des Netzwerkes verwendet wird. Der Wert variiert je nach Größe des Datensatzes. Für eine durchschnittliche Datensatzgröße liegt dieser Wert zwischen 3 und 30. Werden zu wenig

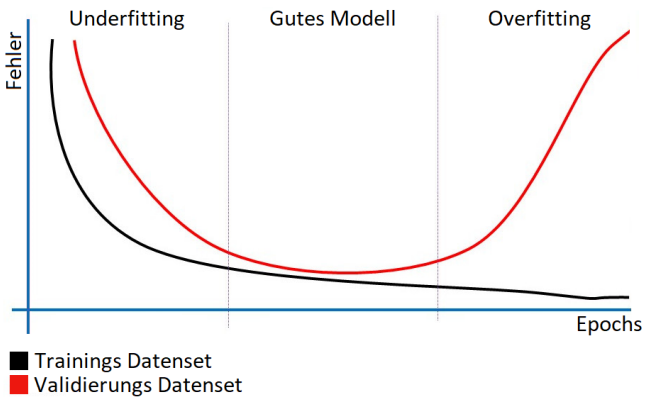


Abbildung 2.7: Aus der Fehlkategorisierung von Objekten im Trainings- bzw. im Validierungs-Datenset kann auf Underfitting und Overfitting geschlossen werden. Grund dafür kann eine zu niedrige oder zu hohe Anzahl der Epochs sein. Die Y-Achse repräsentiert die Fehlklassifizierungen, die X-Achse die Anzahl der gelernten Epochen. Grafik adaptiert aus [96].

Epochs durchgeführt, so kann *Underfitting* (s. Abschnitt 2.3.2) auftreten. Bei zu vielen Epochs kann das Netzwerk jedoch zu *Overfitting* (s. Abschnitt 2.3.1) neigen [104].

2.3 Performance-Probleme

Beim Trainieren von neuronalen Netzen können jedoch auch Probleme auftreten, die einem guten Ergebnis entgegenwirken. Die Probleme sind zum Beispiel *Overfitting* (s. Abschnitt 2.3.1) oder *Underfitting* (s. Abschnitt 2.3.2). Durch gewisse Aktionen bzw. Änderung gewisser Hyperparameter, wie zum Beispiel die Anzahl der Epochs (s. Abschnitt 2.2.8), kann dies reduziert werden. Abbildung 2.7 zeigt Over- bzw. Underfitting, abhängig von der Anzahl der Epoch-Durchläufe.

2.3.1 *Overfitting*

Wenn die Genauigkeit der Ergebnisse im Validierungs-Datenset geringer als die Genauigkeit der Ergebnisse im Trainings-Datenset ist, spricht man von Overfitting. Tritt Overfitting auf, entsteht ein Problem mit der Generalisierung des Netzwerkes. Das Netzwerk lernt zwar die Daten des Trainings-Datensets, kann das erlernte Wissen jedoch nicht richtig auf unbekannte Daten anwenden. Um diesem Phänomen entgegen zu wirken, kann zum einen der Trainings-Datensatz vergrößert werden, was sich zum Beispiel durch Data Augmentation realisieren lässt, zum anderen kann auch die Komplexität des Netzwerkes reduziert werden. Hierfür wird entweder die Anzahl der Layer oder die Anzahl der Neuronen verringert. Auch *Dropout* (s. Abschnitt 2.4.1) oder das Verringern der Epoch-Anzahl sind mögliche Lösungen [78].

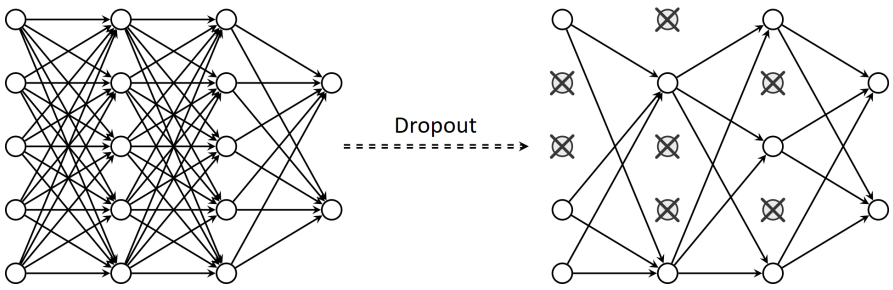


Abbildung 2.8: Visualisierung von Dropout durch Ausschluss von Neuronen in einem neuronalen Netzwerk, um so die Komplexität des Netzes zu verringern. Grafik adaptiert aus [75].

2.3.2 *Underfitting*

Wenn das neuronale Netzwerk es trotz Lernaufwand nicht bewerkstelligt, Daten richtig zu kategorisieren, so spricht man von Underfitting. Dieses Phänomen tritt bereits im Trainings-Datenset auf. Um diesem Problem entgegenzuwirken, muss die Komplexität des neuronalen Netzes erhöht werden. Dafür muss die Anzahl der Layer oder die Anzahl der Neuronen in den Layern vergrößert werden. Weitere Möglichkeiten sind, die Anzahl der Features der Daten im Input Layer zu erhöhen oder die Dropout-Rate (s. Abschnitt 2.4.1) zu verringern. Es kann schon ausreichen, die Anzahl der Epochs zu erhöhen [78].

2.4 *Regularisation*

Das Lernverhalten kann durch gewisse Hyperparameter angepasst werden. Auch potentielle Leistungsprobleme können so verringert werden [30].

2.4.1 *Dropout*

Als Dropout wird das zufällige Ausschließen von Neuronen im Netzwerk bezeichnet. Ein ausgeschlossenes Neuron kann keine Informationen weitergeben, was in Abbildung 2.8 dargestellt ist. Das hat den Effekt, dass das Netzwerk keinen zu hohen Einfluss von einzelnen Neuronen hat, und führt zu einer verbesserten Generalisierung des Netzwerkes, was Overfitting (s. Abschnitt 2.3.1) entgegenwirkt [56].

2.4.2 *Batch Normalisation*

Ein Problem beim Kategorisieren von Bildern ist die Tatsache, dass die Helligkeit von Bild zu Bild variieren kann. Das hätte Auswirkungen auf das gesamte Netzwerk. Eine Möglichkeit, dieses Problem zu lösen, nennt man Batch Normalisation. Da jeder sogenannte Mini-Batch unterschiedlich ist, ist auch die Input-Verteilung anders. Batch Normalisation verwendet den Input, der normalerweise durch eine gewisse Batch Size definiert ist und berechnet dessen Mittelwert und Varianz. Diese Werte werden nun normalisiert. Batch Normalisation kann auch zwischen Layer innerhalb des Netzwerkes verwendet werden [26].

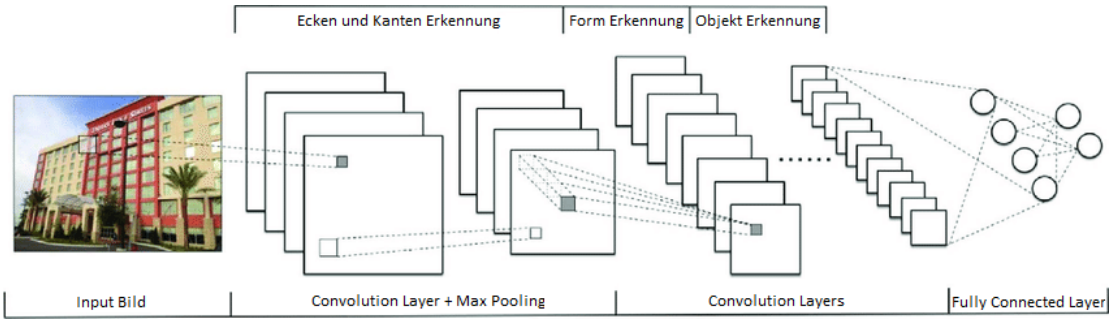


Abbildung 2.9: Allgemeine Darstellung eines Convolutional Neural Networks und dessen Funktionsweise. Die Convolution Layer am Anfang des Netzwerkes sind notwendig, um Low-Level Features, wie zum Beispiel Ecken und Kanten, zu erkennen. High-Level Features am Ende des Netzwerkes sind für ein genaues Bildverständnis zuständig. Die Fully Connected Layer am Ende sind für die Klassifizierung des Bildes verantwortlich. Grafik adaptiert aus [42].

2.5 Bildklassifizierung und *Convolutional Neural Networks*

Zur Klassifizierung von Bildern wird eine spezielle Art von künstlichem neuronalem Netzwerk verwendet bzw. benötigt. Diese Art von neuronalen Netzen wird als Convolutional Neural Network (CNN)[3] bezeichnet und kann Features in Bildern analysieren. Dieses Netzwerk unterscheidet sich hauptsächlich durch spezielle Layer von herkömmlichen neuronalen Netzwerken. Zu diesen Layern gehören *Convolutional Layer* (s. Abschnitt 2.5.1), *Fully Connected Layer* (s. Abschnitt 2.5.4) und *Pooling Layer* (s. Abschnitt 2.5.3). Der Aufbau von Convolutional Neuronal Networks besteht aus einem Input Layer, gefolgt von einer Kombination aus mehreren Convolutional und Pooling Layer, in weiterer Folge von ein oder mehreren Fully Connected Layern und schlussendlich einem Output Layer. Dieser Output Layer ist wieder für die Kategorisierung des Input-Bildes verantwortlich. In Abbildung 2.9 ist ein Aufbau eines solchen CNN veranschaulicht. Verglichen mit einem herkömmlichen neuronalen Netzwerk, kann eine Featuremap mit dem Output eines Neurons und ein Kernel eines Convolutional Layers mit einer gewichteten Verbindung gesehen werden. Diese Art von Netzwerken ist in Bezug auf Skalierung und Translation von Features bis zu einem gewissen Teil invariant [110].

2.5.1 *Convolutional Layer*

Ein Convolutional Layer besteht aus einer Menge von Filtern, die auch als Kernel bezeichnet werden. Ein solcher Filter ist im einfachsten Fall eine zweidimensionale Matrix aus Gewichtungen, sofern jede zweidimensionale Activation Map des Inputs als eigenständiger Input gesehen wird. Die Berechnungen in einem Convolutional Layer werden durchgeführt, indem für jeden Input ein Kernel über den gesamten zweidimensionalen Matrix verschoben und für jede Position ein Wert für die Output Activation Map berechnet wird. Die horizontale und vertikale Verschiebung des Filters wird als *Stri-*

[3] Dt. faltendes neuronales Netzwerk

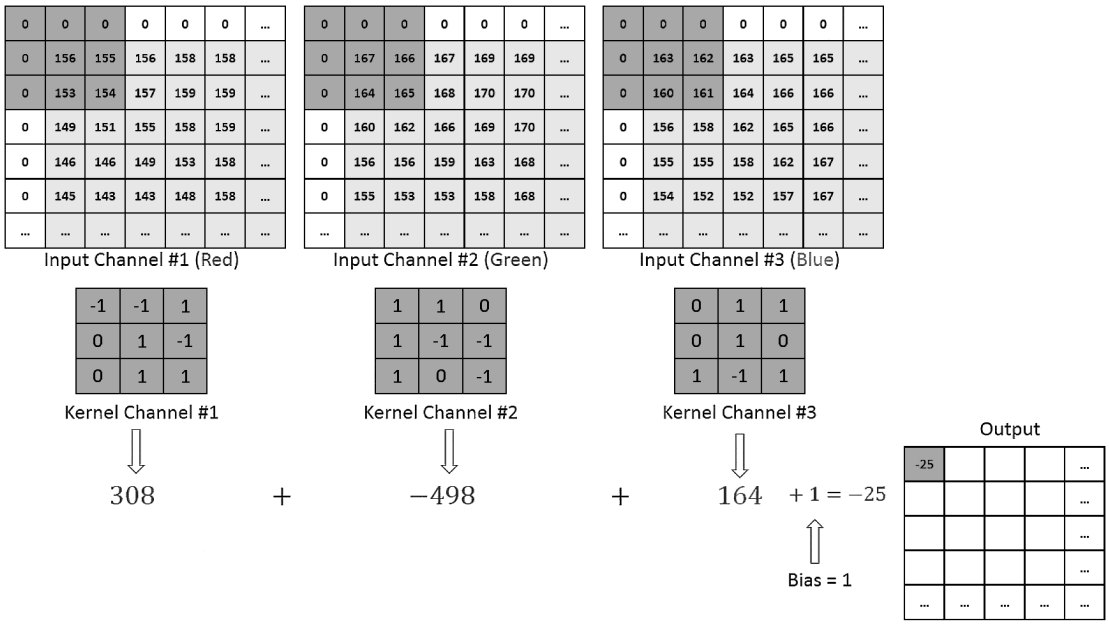


Abbildung 2.10: Grafische Darstellung des Funktionsprinzips von einem Convolutional Layer. Durch mehrmalige Summierung der Input-Bereiche nach Anwendung des Kernels (Filter) wird eine neue Feature Map erstellt. Grafik adaptiert aus [103].

de bezeichnet. Der Output wird im Grunde gleich wie bei einem herkömmlich Neuron berechnet. Hierfür werden die durch den Filter gewichteten Inputs summiert und gegebenenfalls ein Bias (s. Abschnitt 2.1.3) addiert. Diese Berechnung ist in Abbildung 2.10 grafisch dargestellt. Anschließend kann eine Aktivierungsfunktion (s. Abschnitt 2.1.2) angewendet werden. Durch Anwendung eines Kernels ergibt sich, dass, je nach Größe des Filters und der Verschiebung, Informationen am Rand des Inputs verloren gehen und die Output Matrix kleiner als die Input Matrix ist. Das kann unter gewissen Umständen durch Padding (s. Abschnitt 2.5.2) vermieden werden. Ein Convolutional Layer kann auch mehrere Activation Maps als Output haben [110].

2.5.2 *Padding*

Durch Padding soll Informationsverlust am Rand des Inputs vermieden werden. Zu diesem Zweck wird die Activation Map des Inputs vergrößert, indem der Rand um zusätzliche Informationen erweitert wird. Das hat zur Folge, dass ein Kernel nun auch auf den ursprünglich äußersten Positionen des Inputs zentral angewendet wird. Es gibt verschiedene Methoden, mit welchen Informationen der Input erweitert werden kann. Eine weitverbreitete Methode ist es, die Input Matrix überall mit Null zu erweitern. Dies wird als Zero-Padding bezeichnet und ist in Abbildung 2.11 zu sehen [98].

2.5.3 *Pooling Layer*

Pooling Layer werden verwendet, um die Activation Maps zu verkleinern. Das ist positiv zu sehen, da dadurch Rechenaufwand eingespart wird und so tiefere Netzwerke möglich

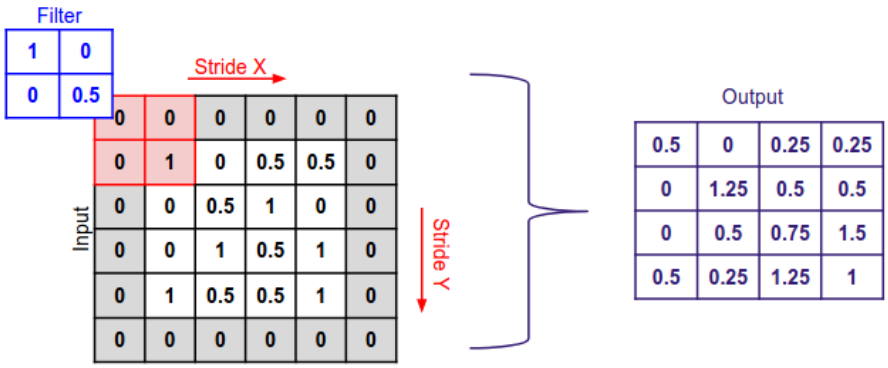


Abbildung 2.11: Grafische Darstellung des Funktionsprinzips von Zero-Padding. Durch Hinzufügen von Nullen am Rand der Ausgangs-Map behält die erstellte Featuremap dieselbe Größe nach Anwendung des Kernels. Bildquelle [98].

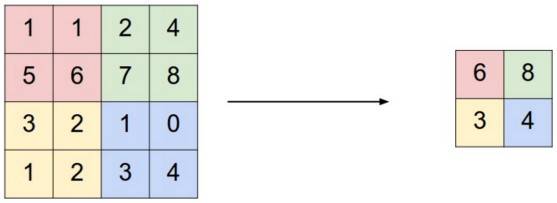


Abbildung 2.12: Grafische Darstellung des Funktionsprinzips von Max-Pooling. Aus jedem farblich markierten Bereich wird die höchste Zahl ausgewählt und in die nächste Featuremap übertragen. Grafik adaptiert aus [110].

sind. Dabei gehen verhältnismäßig wenig Informationen verloren und es hat den Vorteil, dass das Netzwerk weniger sensibel auf unterschiedliche Positionen von Features reagiert. Ein weiterer Vorteil ist, dass Kernel, die auf bereits verkleinerte Activation Maps angewendet werden, einen größeren Bereich des Originalbildes abdecken, ohne den Kernel und somit die Anzahl der Parameter zu vergrößern. Die meistverwendeten Pooling-Strategien sind Max-Pooling und Average-Pooling. Grundsätzlich funktioniert Pooling ähnlich wie ein Convolutional Filter. Hier wird jedoch keine Matrizen-Berechnung durchgeführt, sondern lediglich der größte bzw. der durchschnittliche Wert aus dem Input-Bereich, der vom Filter ausgewählt ist, als Output weitergegeben. In Abbildung 2.12 ist Pooling anhand von Max-Pooling veranschaulicht [110].

2.5.4 *Fully Connected Layer*

Ein Fully Connected Layer ist ein Layer in einem CNN, der nicht aus Filtern, sondern aus Neuronen besteht, die einen einzelnen Wert speichern. Diese Layer-Art ist am Ende eines CNN angesiedelt. Beim Übergang von einer Activation Map in einen Fully Connected Layer wird jeder Wert der Activation Maps mit einem Neuron im nachfolgenden Fully Connected Layer verbunden. Diese Neuronen sind über Gewichtungen mit Neuronen aus dem nachfolgenden Layer verbunden. Die Anzahl der Neuronen im Output Layer entspricht der Anzahl der möglichen Bildkategorien [110].

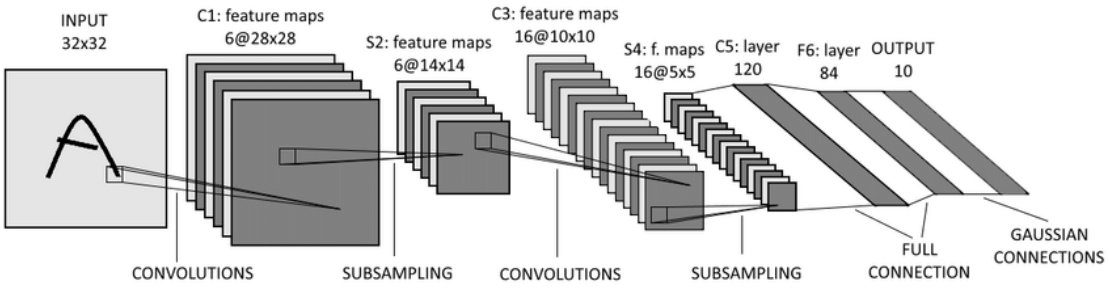


Abbildung 2.13: Grafische Darstellung der Architektur des LeNet-5 Modells. Bildquelle [34].

2.5.5 Bewährte CNN Modelle

Über die letzten Jahre haben sich verschiedenste Convolutional Neural Networks zum Klassifizieren von Bildern bewährt. Sie unterscheiden sich durch deren Anzahl und Anordnung von bestimmten Layern sowie deren Parameter voneinander. Die meisten dieser Netzwerke haben an der *ImageNet Large-Scale Visual Recognition Challenge*[4] (ILSVRC) teilgenommen und diese auch gewonnen. ILSVRC ist ein renommierter Wettbewerb im Bereich Computer Vision. Zum Trainieren verwendeten diese Netzwerke die Datenbank von ImageNet, die in etwa 14 Millionen Bilder enthält. Der Erfolg wird beim Wettbewerb mit der Top 5 Testfehlerrate angegeben, die den Rückschluss zulässt, wie hoch der relative Anteil der zu klassifizierenden Bilder ist, die vom Netzwerk nicht richtig unter den Top 5 wahrscheinlichsten Kategorien eingeordnet wurden. Die bekanntesten CNN werden im Folgenden kurz erläutert [110].

LeNet

LeNet ist ein von LeCun im Jahr 1998 entwickeltes Modell, um handgeschriebene Ziffern zu erkennen [34]. Es besteht aus insgesamt sieben Layern. Die sieben Layer setzen sich aus drei Convolutional Layern, zwei Average Pooling Layern und zwei Fully Connected Layern zusammen. Der Input war bei diesem Modell auf 32×32 Pixel große Graustufenbilder beschränkt. Diese Architektur ist in Abbildung 2.13 dargestellt.

AlexNet

AlexNet wurde von Krizhevsky, Sutskever und Hinton im Jahr 2012 entwickelt [29] und war der erste große Durchbruch für neuronale Netze im Bereich Computer Vision. Dieses Netzwerkmodell hat 2012 die ILSVRC mit einer Top 5 Testfehlerrate von 15.4% gewonnen. Das Netzwerk besteht, verglichen zu modernen Netzwerken, aus einer relativ simplen Architektur mit nur acht Layern. Es besteht in Summe aus fünf Convolutional Layern, Max Pooling Layern und Dropout Layern sowie drei Fully Connected Layern. Die Filter in den Convolutional Layern haben eine Größe von 11×11, 5×5 bzw. 3×3. Als Aktivierungsfunktion (s. Abschnitt 2.1.2) wurde ReLu verwendet. Das Netzwerk wurde dafür trainiert, zwischen 1000 unterschiedlichen Bildkategorien unterscheiden zu

[4] http://www.image-net.org/challenges/LSVRC/

können.

VGGNet

Die VGGNet Architektur wurde von Simonyan und Zisserman von der University of Oxford im Jahr 2014 entwickelt [55] und besteht aus 19 Layern. Es hat bei der ILSVRC eine Top 5 Testfehlerrate von 7.3% erreicht. Die Layer setzen sich aus Convolutional Layern, Max Pooling Layern und Fully Connected Layern zusammen. Die Filter in den Convolutional Layern haben eine einheitliche Größe von 3×3. Durch das Hintereinander-Anordnen von zwei oder mehreren Layern mit kleinen Filtern kann ein Layer mit einem größeren Filter nachgebildet werden. Das hat den Vorteil, dass die Anzahl der Parameter verringert wird. Als Ergebnis der Pooling Layer verringert sich die Größe der Output Activation Map in jeder Dimension um die Hälfte. Die Anzahl der Filter im darauffolgenden Convolutional Layer wird jedoch erhöht, was zu einer größeren Anzahl an Activation Maps führt.

GoogLeNet

GoogLeNet war eines der ersten Modelle, das die Idee einführte, CNN Layer nicht immer sequentiell hintereinander, sondern auch parallel anzuordnen [57]. Aus diesem Grund wird das Modell auch als Inception bezeichnet. Grundsätzlich besteht dieses Netzwerk aus 22 hintereinander ausgeführten Layern, wobei es insgesamt neun solche parallelen Module enthält. Bei dieser Architektur werden auch Convolutional Layer mit einer Filtergröße von 1×1 genutzt. Dies kann dazu verwendet werden, um die Anzahl der Activation Maps zu verändern. Statt der Fully Connected Layer wird die Größe der Activation Map durch Average Pooling bis auf die Größe von 1×1 verkleinert, was einem Neuron eines Fully Connected Layers entspricht und somit als Klassifikations-Layer verwendet werden kann. Durch das Verzichten von Fully Connected Layern reduziert sich die Anzahl der lernbaren Parameter enorm. Dieses Modell hat bewiesen, dass eine kreative Architektur sowohl zu Geschwindigkeits- als auch Leistungsvorteilen führen kann, und gewann die ILSVRC 2014 mit einer Top 5 Testfehlerrate von 6.7%. Im Vergleich zum AlexNet hat diese Architektur nur 1/12 der Parameter.

ResNet

ResNet wurde im Jahr 2015 von Microsoft Research Asia entwickelt [18] und besteht aus insgesamt 152 Layern. Die Besonderheit an dem Modell ist, dass Inputs aus früheren Layern über eine Art Bypass im Netzwerk weiter nach hinten geleitet und so mit dem Output eines späteren Layers kombiniert werden. Das wird in Abbildung 2.14 gezeigt. Diese Netzwerkarchitektur gewann bei der ILSVRC 2015 den ersten Platz mit einer Top 5 Testfehlerrate von 3.6%.

2.6 Transfer Learning

Die Daten zum Trainieren eines neuronalen Netzes sind ein sehr wichtiger Bestandteil, um ein brauchbares Ergebnis zu erhalten. Oftmals ist der Umfang der vorhandenen Daten, den das Netzwerk benötigt, nicht groß genug. In so einem Fall kommt Transfer

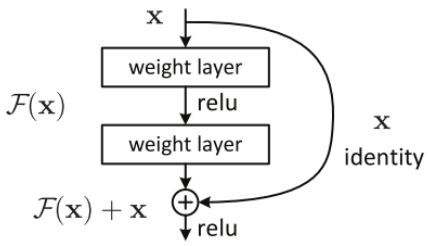


Abbildung 2.14: Grafische Darstellung des Prinzips des ResNets anhand einer Weiterleitung von Layer Outputs nach hinten im Netzwerk. Bildquelle [18].

Learning zum Einsatz. Beim Transfer Learning wird das Netzwerk mit einem anderen umfangreichen Datensatz angelernt. Dieses Modell inklusive der angelernten Parameter und Gewichtungen wird als Grundlage für den Lernvorgang mit den eigenen Daten verwendet. Dazu wird der letzte Layer im Netzwerk entfernt und durch einen eigenen Klassifikations-Layer ersetzt. Die Gewichtungen in den Convolutional Layern werden zum Teil eingefroren. Das bedeutet, dass sich deren Gewichtungen beim erneuten Lernvorgang nicht mehr verändern können. Dieser Vorgang wird auch als Fine-Tuning bezeichnet. Da die frühen Layer im Netztwerk dafür zuständig sind, allgemeine Features, wie zum Beispiel Kanten und Ecken, zu erkennen, können diese erlernten Features unverändert für das eigene Datenset genutzt und brauchen nicht erneut antrainiert werden. Je weiter hinten die Layer im Netzwerk angesiedelt sind, desto spezifischer werden die Features, die vom Layer extrahiert werden. Je nachdem, wie sehr sich das eigene Datenset von dem unterscheidet, was zum grundlegenden Training verwendet wurde, können mehr oder weniger Convolutional Layer eingefroren werden. Die Layer, die für die Klassifizierung zuständig sind, müssen neu trainiert werden [65].

2.7 Objekterkennung

Die Erkennung von Objekten in einem Bild entspricht nicht der Klassifizierung eines Bildes an sich. Der Unterschied zwischen Objekterkennung und Klassifizierung ist, dass bei Objekterkennungs-Algorithmen zusätzlich die Position des Objektes im Bild analysiert wird. Dies wird oft mittels eines Begrenzungsrahmens dargestellt und ist in Abbildung 2.15 ersichtlich. Weiters sollen mittels Objekterkennung auch mehrere Objekte innerhalb eines Bildes gefunden werden können. Das kann mit einem CNN nicht direkt gelöst werden, da die Anzahl der Objekte innerhalb eines Bildes im Vorhinein nicht bestimmt werden kann und der Output Vektor somit eine variable Länge aufweist [85]. Ein anfänglicher, gutgläubiger Versuch, dieses Problem zu lösen, war hier, verschiedene Interessensregionen im Bild zu definieren und auf jede dieser Regionen ein CNN zur Klassifikation anzuwenden. Das Problem dieses Versuches ist, dass Objekte an unterschiedlichen Orten im Bild sein können, sie eine unterschiedliche Aspekt Ratio haben oder unterschiedlich groß sein können. Daher musste eine große Menge an Regionen definiert werden, um alle Möglichkeiten abzudecken. Um dieses Problem zu lösen, wurden Methoden wie Region-based Convolutional Neural Networks (R-CNNs) oder You Only Look Once (YOLO) entwickelt [85].

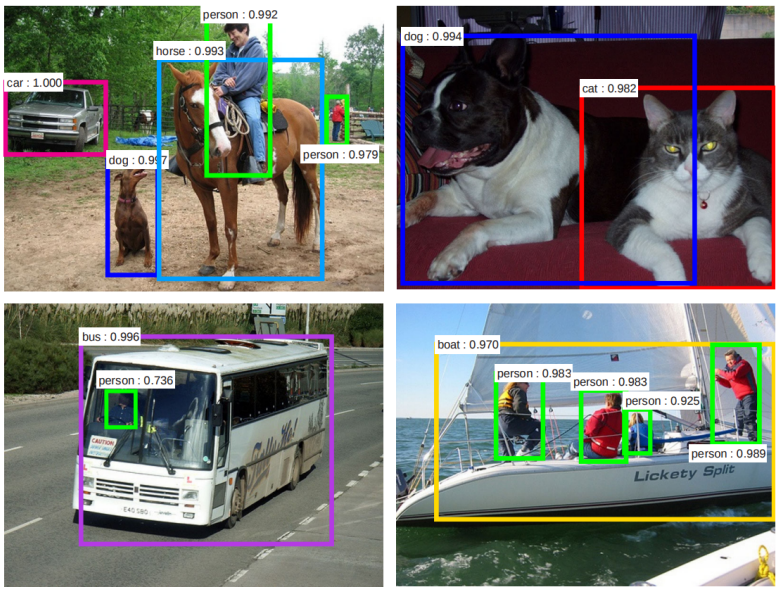


Abbildung 2.15: Erkennung und Lokalisierung von mehreren erkannten Objekten in einem Bild durch Begrenzungsrahmen und Klassifizierung. Bildquelle [51].

2.7.1 Region-based Convolutional Neural Network

Eine der bekanntesten Methoden zur Erkennung von Objekten innerhalb eines Bildes nennt sich Region-based Convolutional Neural Network. Über die Zeit haben sich hier unterschiedliche Lösungsansätze etabliert, die im Folgenden beschrieben werden.

R-CNN

Um das Problem der benötigten, großen Anzahl von Regionen zu lösen, beschreibt [16] eine Methode, die einen selektiven Suchalgorithmus verwendet, damit die Anzahl der Interessensregionen auf 2000 reduziert werden kann. Ein selektiver Suchalgorithmus funktioniert nach dem Prinzip, zu Beginn eine Untersegmentierung, die eine Vielzahl von potentiellen Regionskandidaten liefert, durchzuführen. Danach wird ein Greedy-Algorithmus angewendet, um ähnliche Regionen rekursiv zu größeren Bereichen zu vereinen. Die vereinten Regionen werden schlussendlich genutzt, um die finalen Interessensregionen zu bestimmen. Wie ein selektiver Suchalgorithmus für Objekterkennung funktioniert, kann unter [58] im Detail nachgelesen werden. Diese 2000 Regionen werden zu einer einheitlichen, quadratischen Größe transformiert und mit einem CNN analysiert, das einen 4096-dimensionalen Feature-Vektor als Ausgabe erzeugt. Die vom CNN extrahierten Features werden in eine *Support Vector Machine* (SVM) [86] eingespeist, um damit ein potentielles Objekt in einer dieser Regionen zu klassifizieren. Zusätzlich zur Klassifizierung des Objektes wird noch die Position und Größe des Objektes im Bild prognostiziert. Das wird in Abbildung 2.16 grafisch dargestellt. Die Lösungsmethode hat jedoch noch einige unausgereifte Ansätze. Es dauert sehr lange, das Netzwerk zu trainieren, da pro Bild 2000 unterschiedliche Regionen klassifiziert werden müssen. Infolgedessen ist auch die benötigte Zeit, ein Testbild zu analysieren, entsprechend hoch.

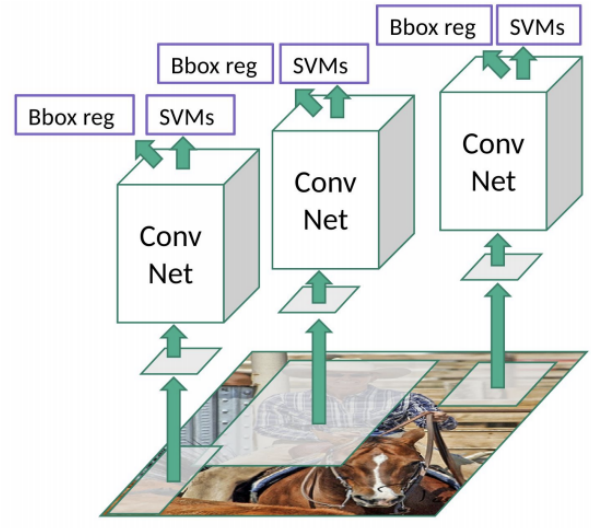


Abbildung 2.16: Darstellung der Architektur des R-CNN Modells. Jede durch einen selektiven Suchalgorithmus bestimmte Region wird mit einem CNN analysiert. Bildquelle [85].

Weiters ist die Verwendung eines selektiven Suchalgorithmus zur Bestimmung der Interessensregionen keine optimale Lösung, da dieser ein fix definierter Algorithmus ist und sich daher nicht über die Zeit verbessern kann, was zu schlechten Regions-Kandidaten führt.

Fast R-CNN

Derselbe Autor wie bei R-CNN konnte einige dieser Problematiken lösen und war in der Lage, einen schnelleren Lösungsansatz zu finden. Diesen Lösungsansatz nannte er Fast R-CNN [15]. Grundsätzlich ist der Ansatz ziemlich ähnlich. Statt jedoch jede einzelne Region mittels eines CNN zu analysieren, wird bei dieser Methode das gesamte Bild mit einem CNN analysiert, was einen enormen Zeitvorteil bringt. Von den daraus erhaltenen Featuremaps werden die interessanten Regionen mit einem selektiven Suchalgorithmus identifiziert und in Quadrate mit fixer Größe transformiert. So können diese zu einem Fully Connected Layer umgewandelt werden. Zuletzt wird ein Softmax-Layer angewandt, um die Regionen zu klassifizieren. Weiters wird mit einem BoundingBox-Regressor die Position und Größe des Objektes im Bild bestimmt. Diese Architektur wird in Abbildung 2.17 dargestellt. In Abbildung 2.18 sieht man, dass Fast R-CNN signifikant schneller als R-CNN ist. Die Werte wurden hier von [91] übernommen und sollen auf den relativen Zeitunterschied hinweisen, da die absolute Zeit, die benötigt wird, immer von der Art der Aufgabe und des verwendeten Systems abhängig ist. Bei der benötigten Zeit zum Testen lässt sich erkennen, dass ein Großteil der Berechnungszeit des Algorithmus für die Regions-Vorhersage benötigt wird. Das bedeutet, dass die Vorhersage der Interessensregionen durch den selektiven Suchalgorithmus der Schwachpunkt des Fast R-CNN Algorithmus in Bezug auf die Performance ist und diese noch Verbesserungspotential aufweist.

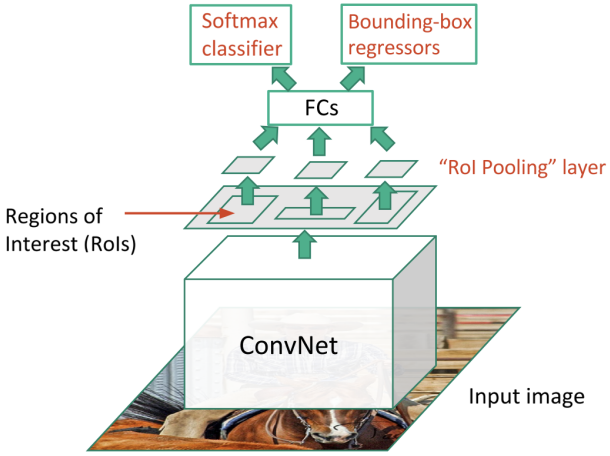
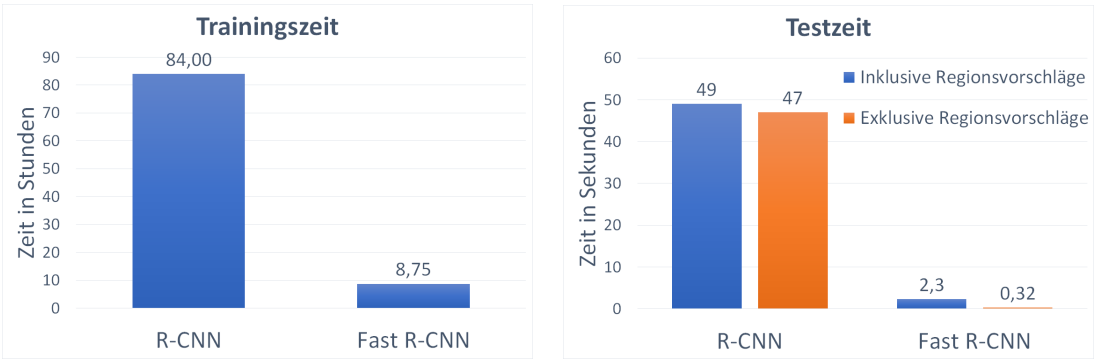


Abbildung 2.17: Darstellung der Architektur des Fast R-CNN Modells. Die Interessensregionen werden erst nach der Analyse mittels CNN bestimmt. Bildquelle [113].



(a) (b)

Abbildung 2.18: Zeitvergleich zwischen R-CNN und Fast R-CNN. Die benötigte Trainingszeit ist bei R-CNN um einiges größer als bei Fast R-CNN (a). Fast R-CNN ist auch bei der Analyse von Bildern schneller (b).

Faster R-CNN

Das Performance-Problem bei der Bestimmung der Interessensregionen wird bei Faster R-CNN [51] gelöst. Gleich wie bei Fast R-CNN wird das Bild als Ganzes mittels eines CNN analysiert. Um Regionsvorschläge zu treffen, wird nicht, wie in den zuvor beschriebenen Methoden, ein selektiver Suchalgorithmus verwendet. Stattdessen wird ein Objekterkennungs-Algorithmus eingeführt, bei dem das Netzwerk die Regionsvorschläge selbstständig durch Lernen identifiziert. Hierfür wird ein eigenes, sogenanntes Region Proposal Network eingesetzt. Das bringt einen enormen Zeitvorteil und hat zur Folge, dass der gesamte Ablauf in etwa gleich viel Zeit benötigt, wie Fast R-CNN ohne Bestimmung der Regionsvorschläge. Dies ist in Abbildung 2.20 zu sehen. Die verwendeten Werte wurden wieder von [91] übernommen. Die vorhergesagten Regionsvorschläge werden anschließend mit einer RoI-Pooling-Schicht umgestaltet, um das Bild innerhalb

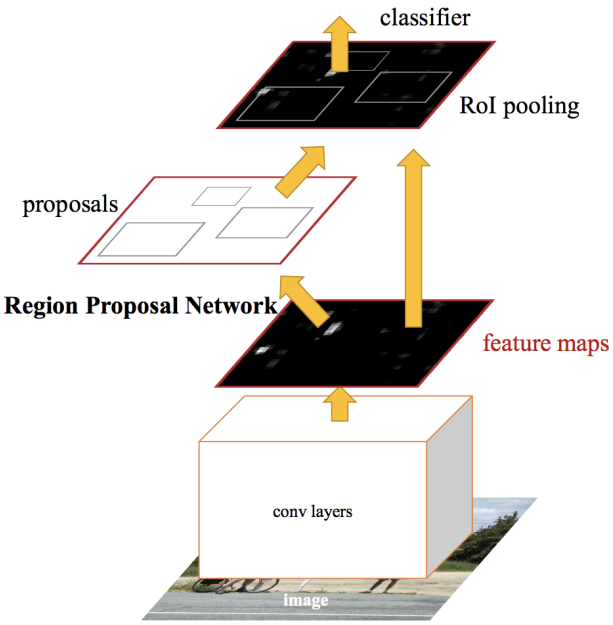


Abbildung 2.19: Darstellung der Architektur des Faster R-CNN Modells. Bildquelle [85].

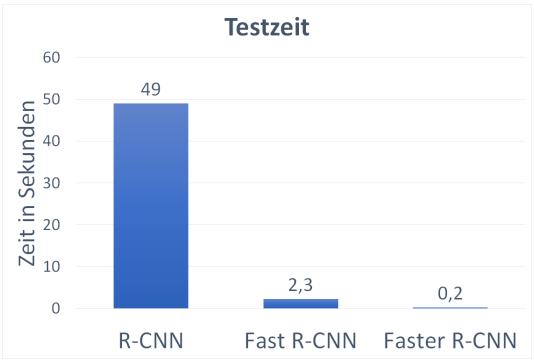


Abbildung 2.20: Testzeitvergleich zwischen R-CNN, Fast R-CNN und Faster R-CNN inklusive Regionsvorschläge. Faster R-CNN konnte die benötigte Zeit für Regionsvorschläge im Vergleich zu Fast R-CNN dramatisch reduzieren.

der vorgeschlagenen Region zu klassifizieren und die Offsetwerte für die Begrenzungsrahmen vorherzusagen. Die Architektur von Faster R-CNN wird in Abbildung 2.19 grafisch dargestellt.

Mask R-CNN

Mask R-CNN [19] beschreibt eine Methode, die Faster R-CNN funktionell erweitert. Hier wird zusätzlich zu den bestehenden Zweigen, die sowohl Klassifizierung als auch Position und Größe des Objektes bestimmen, ein weiterer Zweig hinzugefügt. Dieser Zweig ist für Vorhersagen von Segmentierungsmasken der einzelnen Interessensregionen zuständig. Dazu wird mittels eines Fully Connected Networks eine Segmentierungsmas-

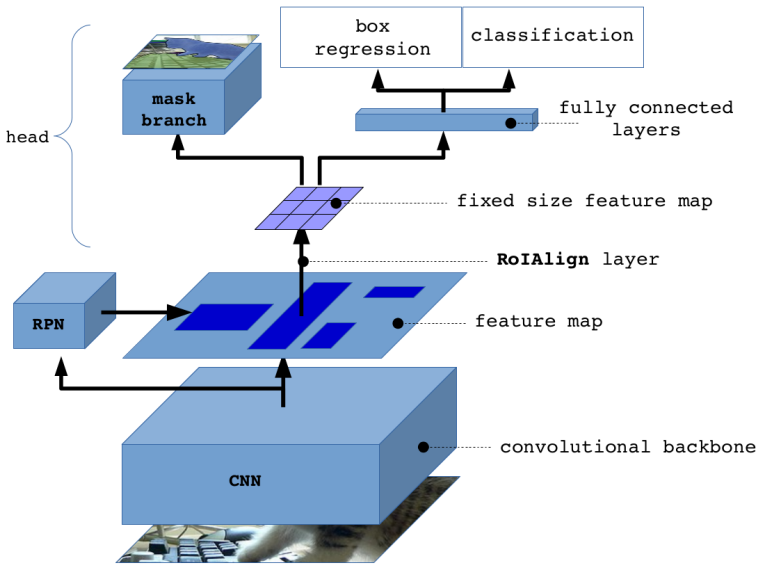


Abbildung 2.21: Darstellung der Architektur des Mask R-CNN Modells. Bildquelle [112].

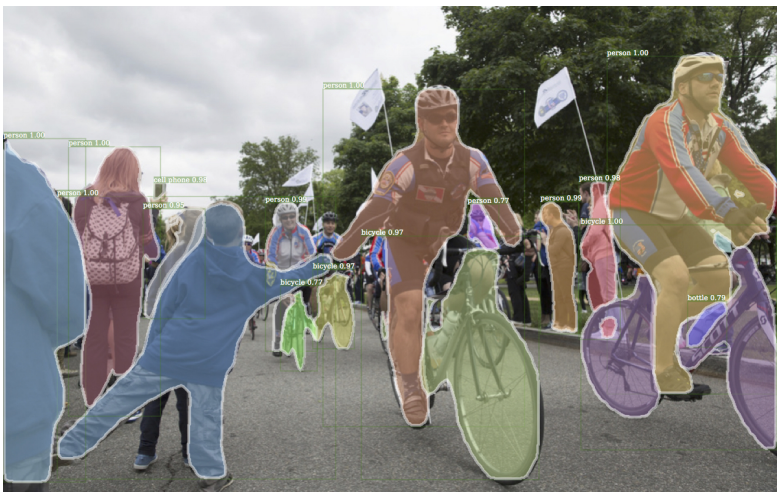


Abbildung 2.22: Erkannte Objekte sind mit einer Maske segmentiert. Bildquelle [70].

ke auf Pixelebene vorhergesagt. Die zusätzliche Berechnung führt nur zu einem minimal größeren Rechenaufwand. Da Faster R-CNN grundsätzlich nicht für Analysen auf Pixelebene ausgelegt ist, was sich am deutlichsten bei den Pooling-Operationen zeigt, wurde ein quantisierungsfreier Layer, der die genauen Positionsdaten persistiert, eingeführt. Die Systemarchitektur kann in Abbildung 2.21 gesehen werden. Abbildung 2.22 zeigt ein Beispielbild, wie eine Segmentierungsmaske mittels Mask R-CNN aussieht.

2.7.2 YOLO – You Only Look Once

You Only Look Once (YOLO) ist eine Objekterkennungsmethode, die den Fokus auf den Bereich Echtzeitanalyse legt. Der Name kommt daher, weil das Bild nur einmal vom Netzwerk betrachtet wird und nicht, wie bei anderen Objekterkennungsmethoden, für jede Interessensregion separat. Wie schon bei R-CNN, gibt es auch hier verschiedene Implementierungs-Iterationen. Diese werden im Folgenden kurz erläutert.

YOLOv1

Bei YOLO [49] wird das Bild in ein $S \times S$ großes Rasterfeld unterteilt. Die Zelle des Rasterfeldes, in die das Zentrum eines Objektes fällt, ist auch für die Erkennung dieses Objektes verantwortlich. Dafür werden für jede Zelle B Begrenzungsboxen vorhergesagt. Jede dieser Begrenzungsboxen ergibt eine Aussage über fünf Werte. Diese sind die X-Position, die Y-Position, die Breite und die Höhe der Box sowie ein Konfidenzwert, der angibt, wie hoch die Wahrscheinlichkeit ist, dass die Box ein Objekt, unabhängig davon, welches es ist, enthält. Die X- und Y-Position gibt hier das Zentrum der Box relativ zu der Rasterfeldbegrenzung an. Die Breite und die Höhe der Box werden relativ zum gesamten Bild angegeben. Zusätzlich werden für jedes Rasterfeld C Klassenwahrscheinlichkeiten vorhergesagt.

Im Fall der Evaluierung von YOLOv1 in Bezug auf den PASCAL VOC Datensatz wurde für $S = 7$, $B = 2$ gewählt. Weiters kann hier zwischen $C = 20$ verschiedenen Klassen unterschieden werden. Dies ergibt eine Output Tensor mit der Größe

$$S \times S \times (B \cdot 5 + C) = 7 \times 7 \times (2 \cdot 5 + 20) = 1470. \tag{2.6}$$

Eine Übersicht der zu bestimmenden Parameter ist in Abbildung 2.23 grafisch dargestellt. In Abbildung 2.24 wird im linken Bild die Aufteilung in ein Rasterfeld gezeigt. Im oberen Bild werden die vorhergesagten Begrenzungsboxen dargestellt, wobei hier die Linienstärke als Konfidenzwert zu interpretieren ist. Im unteren Bild wird jeweils die Klasse angezeigt, die für eine Zelle am wahrscheinlichsten befunden wurde. Im rechten Bild ist das Ergebnis ersichtlich. Hier werden nur die Begrenzungsboxen, bei denen der Konfidenzwert einen bestimmten Schwellenwert überschreitet, angezeigt und mit der Klassenprognose der zugehörigen Zelle kombiniert. Das bedeutet, dass pro Zelle maximal genau ein Objekt erkannt werden kann, egal, wie viele Begrenzungsboxen pro Zelle generiert werden. Das hat den Nachteil, dass ein Bild mit vielen kleinen Objekten nicht richtig erkannt wird. Hier werden die meisten ignoriert. Das Netzwerk Modell besteht aus 24 Convolutional Layern, gefolgt von zwei Fully Connected Layern. Zur Verringerung der Anzahl der Parameter werden 1×1 Filter verwendet. Die genaue Architektur ist in Abbildung 2.25 zu sehen. YOLO erreicht eine mittlere Durchschnittlichkeit von 63.4%, bei 45 fps. Alternativ gibt es noch eine weitere Umsetzung des neuronalen Netzwerkes. Dieses wird Fast YOLO genannt und besteht aus neun, statt 24 Convolutional Layern. Fast YOLO erreicht einen Wert von 52.7% bei 155 fps.

YOLOv2

YOLOv2 [47] ist eine überarbeitete Version von YOLOv1. So wird nun zum Beispiel Batch Normalization (s. Abschnitt 2.4.2) bei allen Convolutional Layern verwendet.

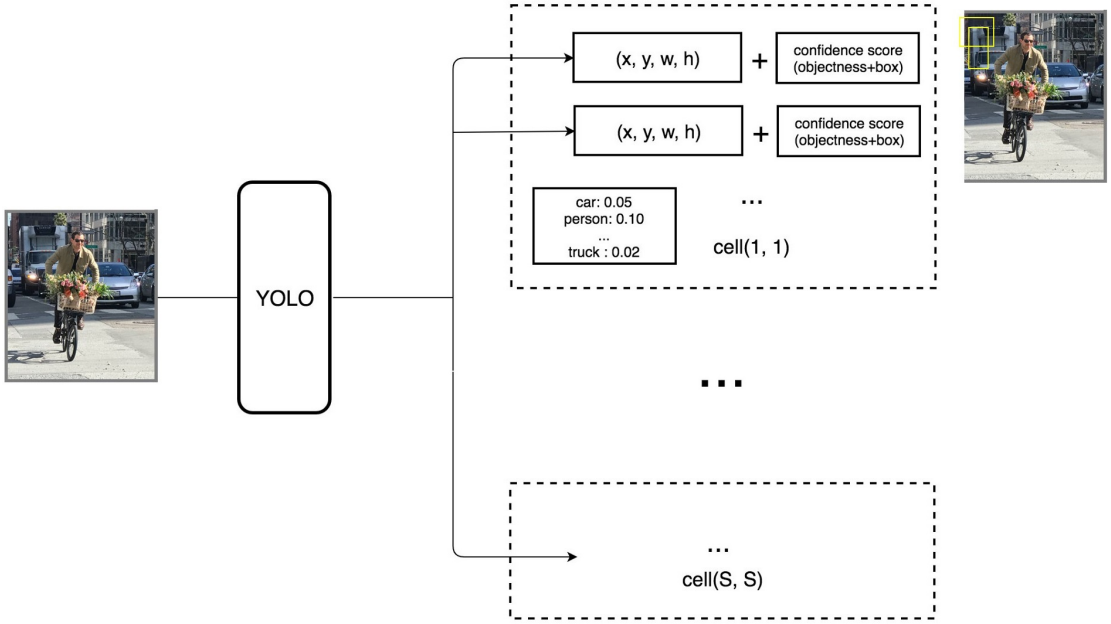


Abbildung 2.23: Grafische Darstellung des Netzwerk-Outputs von Mask R-CNN. Es werden Informationen zum Begrenzungsrahmen und ein Konfidenzwert berechnet sowie ein Wahrscheinlichkeitsvektor für die Klassifizierung. Bildquelle [88].

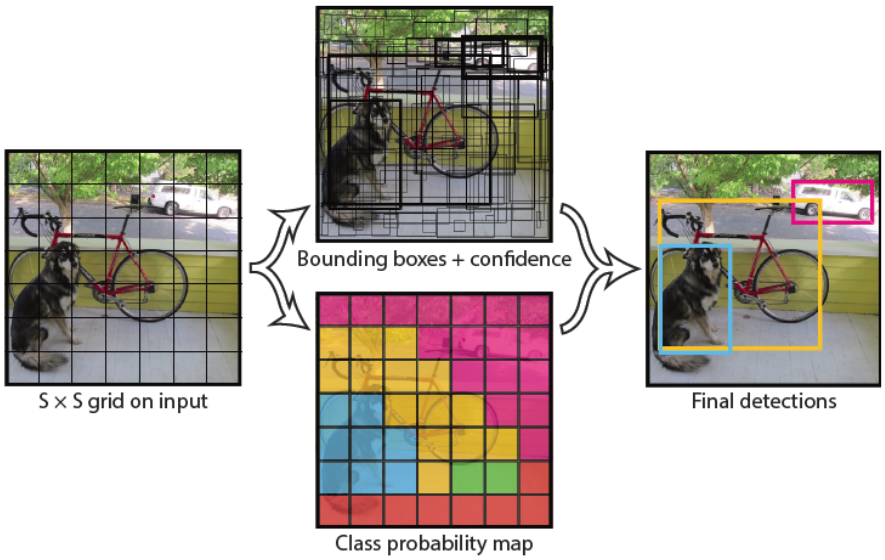


Abbildung 2.24: Beispielbild mit den unterschiedlichen Zwischenschritten von YOLO bis hin zum Ergebnisbild. Im linken Bild wird die Aufteilung in ein Rasterfeld gezeigt. Im oberen Bild werden die vorhergesagten Begrenzungsboxen dargestellt, wobei hier die Linienstärke als Konfidenzwert zu interpretieren ist. Im unteren Bild wird jeweils die Klasse angezeigt, die für eine Zelle am wahrscheinlichsten befunden wurde. Im rechten Bild wird das Ergebnis gezeigt. Bildquelle [49].

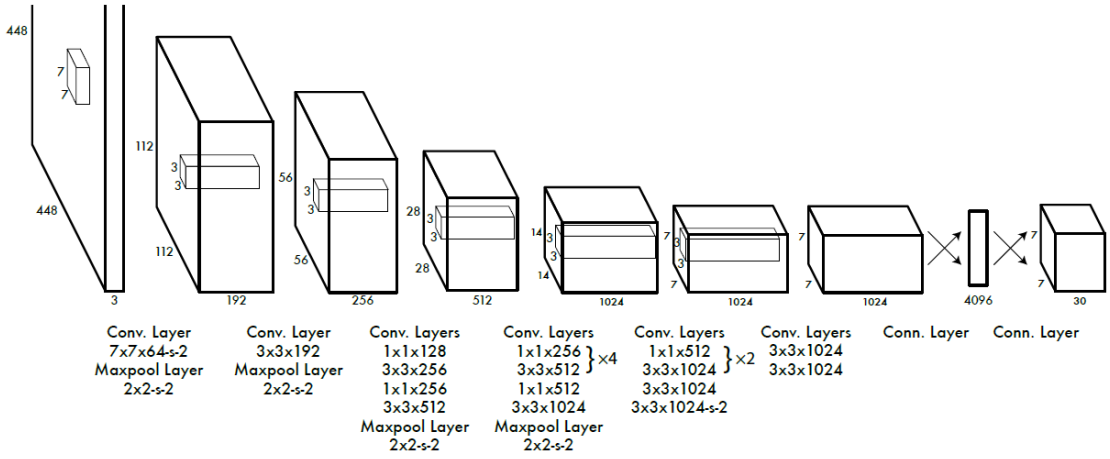


Abbildung 2.25: Grafische Darstellung der verwendeten Netzwerkarchitektur von YOLOv1. Bildquelle [49].

Weiters wird die Netzwerkarchitektur durch Darknet-19, die schneller ist, ersetzt. Sie wird in Abbildung 2.26 gezeigt. Dieses Netzwerk wird nun nach grundlegendem Training mit Bildern höherer Auflösung für 10 Epochs nachtrainiert. Welche weiteren Verbesserungen genau getroffen wurden, kann im Detail in Abbildung 2.27 gesehen werden. Mit all diesen Verbesserungen erreicht YOLOv2 einen mittleren Vorhersagewert von 76.8% bei 67 fps bzw. 78.6% bei 40 fps. Diese Werte werden in Abbildung 2.28 mit anderen Objekterkennungsmethoden verglichen.

YOLO9000

In [47] wird eine weitere Methode namens YOLO9000 beschrieben, die darauf eingeht, wie mehrere Datensets miteinander verknüpft werden können, selbst wenn sich deren Klassen nicht gegenseitig eindeutig ausschließen. So gibt es im COCO Datenset zum Beispiel die Kategorie *Hund* und im ImageNet Datenset eine Kategorie *Norfolk Terrier*. Somit ist keine eindeutige Zuordnung möglich. Um dieses Problem zu lösen, wurde ein Klassifikationsbaum, der in Abbildung 2.29 zu sehen ist, erstellt. Dieser Baum wird als *WordTree* bezeichnet. Hier gibt es verschiedene Detailstufen der Kategorisierung. Kann zum Beispiel ein Hund keiner Hunderassen-Kategorie eindeutig (mit hoher Wahrscheinlichkeit) zugewiesen werden, so wird dieser generell als *Hund* erkannt. Es ergeben sich 9418 unterschiedliche Klassen, woher auch der Name YOLO9000 stammt. Dieses Netzwerk wurde ausschließlich mit Bildern aus dem COCO Datenset trainiert. Evaluiert wurde es mit dem ImageNet Objekterkennungsdatenset. Dieses umfasst 200 unterschiedliche Kategorien, wobei nur 44 davon auch in den Trainingsdaten des COCO Datensets verfügbar sind. Für die restlichen 156 Kategorien hatte das Netzwerk keine Trainingsdaten. YOLO9000 erreichte eine allgemeine Vorhersagegenauigkeit von 19.7%, wobei für die 156 nicht trainierten Kategorien 16.0% erreicht wurden. Dies lässt sich auf die trainierten Elternklassen zurückführen. So wurden unterschiedliche Tierarten oftmals generell als Tier klassifiziert. Bei Kategorien, wie zum Beispiel Kleidung, hatte das Netzwerk jedoch Probleme.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Abbildung 2.26: Tabellarische Darstellung der bei YOLOv2 verwendeten Netzwerkarchitektur. Bildquelle [47].

YOLOv3

YOLOv3 [48] hat einige kleinere Verbesserungen im Vergleich zum Vorgänger. So wird nun als Netzwerk Darknet-53, das aus 53 Layern besteht, verwendet. Wie in der ResNet Architektur (s. Abschnitt 2.5.5), werden hier gewisse Layeroutputs im Netzwerk nach hinten weitergeleitet. Die Architektur des Netzwerkes kann in Abbildung 2.30 gesehen werden. Weiters wird bei diesem Netzwerk statt des mittleren quadratischen Fehlers binäre Kreuzentropie als Kostenfunktion verwendet. Die Implementierung ist online[5] frei zugänglich.

2.7.3 Alternative Objekterkennungsmethoden

Außer R-CNN und YOLO gibt es noch weitere Objekterkennungsmethoden. Die bekannteren hier sind Single Shot MultiBox Detector (SSD) [41], Region-based Fully Convolutional Network (R-FCN) [10] und RetinaNet [40]. Auf diese wird jedoch nicht genauer eingegangen. In [23] werden Faster R-CNN, R-FCN und SSD ausführlich miteinander verglichen.

[5] https://pjreddie.com/darknet/yolo/

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?						✓	✓	✓
location prediction?						✓	✓	✓
passthrough?							✓	✓
multi-scale?							✓	✓
hi-res detector?								✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Abbildung 2.27: Auflistung der Änderungen zwischen YOLOv1 und YOLOv2 wie auch deren Auswirkungen auf die Erkennung von Objekten. Bildquelle [47].

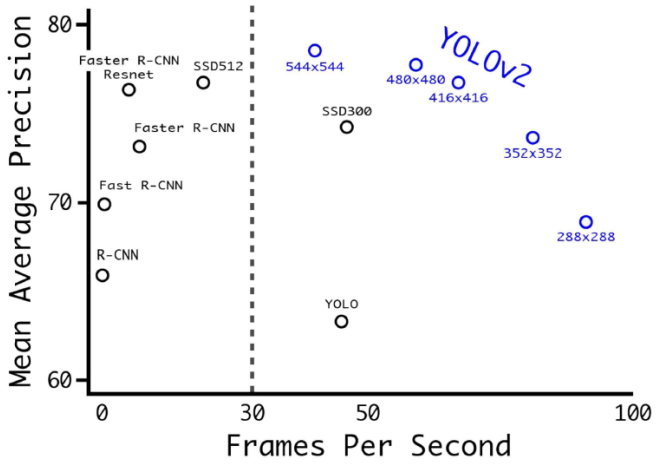


Abbildung 2.28: Vergleich verschiedener Objekterkennungsmethoden in Bezug auf Geschwindigkeit und Vorhersagegenauigkeit. Bildquelle [47].

2.8 Deep Learning Frameworks

Um ein neuronales Netzwerk zu realisieren, gibt es viele Möglichkeiten. Im Folgenden werden acht der bekanntesten Deep Learning Frameworks kurz erläutert.

2.8.1 TensorFlow

TensorFlow ist eine Open Source Plattform für maschinelles Lernen (ML), die verschiedene Abstraktionslevel bietet. Somit empfiehlt sich dieses Framework sowohl für Anfänger als auch für erfahrene Benutzer, die mehr Kontrolle über das Netzwerk haben wollen. Um ein neuronales Netzwerk möglichst simpel zu erstellen, bietet sich die offizielle high-level Keras API an (s. Abschnitt 2.8.2). Für größere ML Aufgaben lässt sich die Distribution Strategy API einsetzen, die dafür konzipiert ist, das Training auf verschiedene Hardwaregeräte aufzuteilen, ohne das Modell selbst verändern zu müssen. Weiters

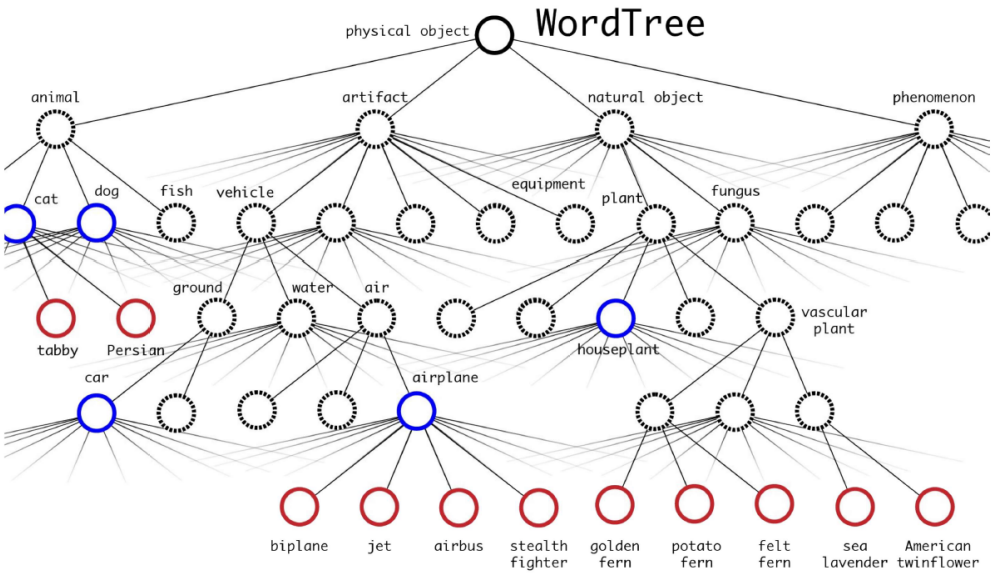


Abbildung 2.29: Darstellung eines Klassifikationsbaumes, bei dem die Kategorisierung über verschiedene Detailstufen möglich ist. Bildquelle [47].

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Abbildung 2.30: Tabellarische Darstellung der bei YOLOv3 verwendeten Netzwerkarchitektur. Bildquelle [48].

ist TensorFlow sprach- und plattformunabhängig. So ist es möglich, dieses für Desktop, Mobile, Web und Cloud Anwendungen zu verwenden. TensorFlow unterstützt Programmiersprachen, wie zum Beispiel Python, C++ und R. Weiters verfügt das Framework über ein Tool namens TesorBoard, das eine effektive Darstellung von der Netzwerkmodellierung und der Performance bietet. Es gibt eine umfangreiche Dokumentation und die größte Community im Bereich Deep Learning [106].

2.8.2 Keras

Keras ist eine high-level API für neuronale Netzwerke in Python, die in Kombination mit drei verschiedenen Backends verwendet werden kann. Dazu gehören TensorFlow (s. Abschnitt 2.8.1), CNTK (s. Abschnitt 2.8.6) und Theano (s. Abschnitt 2.8.7). Der Fokus der Entwicklung liegt bei diesem Framework auf einer einfachen Schnittstelle und auf der Ermöglichung von schnellen Versuchsdurchführungen. Keras unterstützt sowohl Convolutional Networks und Recurrent Networks als auch die Kombination der beiden. Weiters kann die CPU oder GPU zur Ausführung verwendet werden. Die größten Vorteile von Keras sind die leichte Verwendbarkeit, die Modularität und die einfache Erweiterbarkeit durch neue Module. Um den vollen Umfang von Keras nutzen zu können, müssen eine zusätzliche Software bzw. zusätzliche Python Module heruntergeladen und installiert werden. Dies wird in der Dokumentation ausführlich beschrieben [89].

2.8.3 PyTorch

PyTorch ist eine Open Source Deep Learning Plattform, die einen nahtlosen Übergang von einer prototypischen Entwicklung hin zur Produktentwicklung gewährleistet. Um die Performance zu optimieren, kann das Training auf mehrere Systeme aufgeteilt werden. PyTorch kann sowohl in Python als auch in einer C++ Laufzeitumgebung verwendet werden. Modelle können hier im Open Neural Network Exchange (ONNX) Format exportiert und somit von anderen Plattformen, die diesen Standard unterstützen, benützt werden. Weiters wird PyTorch von Cloud Plattformen unterstützt, was das Training mit einer großen Datenmenge ermöglicht [99].

2.8.4 Caffe

Caffe ist ein Framework für Deep Learning, das auf Geschwindigkeit und Modularität ausgerichtet ist. Es wird von Berkeley AI Research (BAIR) und von Community-Mitarbeitern entwickelt. Caffe kann in C, C++, Pyton und MATLAB verwendet werden. Modelle und Optimierungen können durch einfache Konfiguration (ohne Hartcodierung) definiert werden. Durch die Erweiterbarkeit des Codes und der Community-Mitarbeiter wird dieser immer bestmöglich auf dem neuesten Stand der Technik gehalten. Durch die hohe Geschwindigkeit bietet sich dieses Framework vor allem sehr gut für Convolutional Neural Networks zur Bildklassifizierung an [79].

2.8.5 MXNet

MXNet bietet eine Kombination von hoher Performance, sauberem Code sowie high-level APIs und low-level Kontrolle. MXNet automatisiert gängige Arbeitsabläufe, sodass

simple neuronale Netze in wenigen Codezeilen dargestellt werden können. Weiters bietet MXNet APIs unter anderem in den Sprachen C++, Java, Perl, Python, R und Scala [94].

2.8.6 Microsoft Cognitive Toolkit – CNTK

Das Microsoft Cognitive Toolkit (früher bekannt als CNTK) ist ein Open Source Deep Learning Framework von Microsoft, das für große Datenmengen optimiert und in kommerziell verwendbarer Qualität ausgelegt ist. Modelle können mit einer high-level oder low-level API in den Sprachen Python, C++ oder BrainScript umgesetzt werden. Zusätzlich kann das Modell in C# evaluiert werden. Die Berechnung der Modelle kann entweder auf der CPU, der GPU oder mithilfe von verteilten Systemen durchgeführt werden. Zusätzlich kann ein Modell auch über Azure trainiert und gehosted werden [92]. Des Weiteren unterstützt CNTK das Open Neural Network Exchange (ONNX) Format und kann somit Modelle in anderen Frameworks, wie zum Beispiel Caffe2, MXNet oder PyTorch, weiter verwenden. Da jedoch ARM Prozessoren im Moment nicht unterstützt werden, ist dies auf mobile Geräte nur bedingt anwendbar [93].

2.8.7 Theano

Theano ist ein Open-Source-Projekt und war eines der ersten Deep Learning Frameworks, das in erster Linie vom Montreal Institute for Learning Algorithms (MILA) an der Université de Montréal entwickelt wurde. Grundsätzlich ist Theano eine Python-Bibliothek und ein optimierter Compiler zur Manipulation und Auswertung mathematischer Ausdrücke, die insbesondere für Matrizen-Berechnungen ausgelegt ist. In Theano werden Berechnungen mit einer NumPy-ähnlichen Syntax ausgedrückt und so kompiliert, dass sie effizient entweder auf CPU- oder GPU-Architekturen ausgeführt werden können. Mittlerweile wurde jedoch bekannt gegeben, dass die Weiterentwicklung von Theano eingestellt wird [108].

2.8.8 Deeplearning 4 Java – DL4J

Deeplearning4j ist in Java geschrieben und mit allen JVM-Sprachen wie Scala, Clojure oder Kotlin kompatibel. Die zugrundeliegenden Berechnungen sind in C, C++ und Cuda umgesetzt. Laut der offiziellen DL4J Webseite dient Keras als Python-API, was von der offiziellen Keras Dokumentation [89] jedoch nicht bestätigt wird. Die Bibliotheken sind vollständig Open-Source und werden von der Entwicklergemeinschaft und dem Skymind-Team auf dem neuesten Stand gehalten. DL4J nutzt die Vorteile der neuesten verteilten Computer-Frameworks, wie Apache Spark und Hadoop, um das Trainieren der Netzwerke zu beschleunigen. Bei Multi-GPUs entspricht dies in etwa der Leistung von Caffe [83].

2.9 GPU-Support

Keras mit TensoFlow als Backend ermöglicht beim Trainieren von neuronalen Netzwerken entweder die Verwendung der CPU oder der GPU. Soll die GPU verwendet werden, so muss auch das GPU unterstützende Modul von TensorFlow in Python installiert werden. Das Training mithilfe der GPU hat den Vorteil, dass das neurona-

le Netzwerk beim Lernen zirka um den Faktor 100 schneller ist. Dies führt jedoch zur Einschränkung, dass es nur auf Systemen mit einer Grafikkarte der Firma NVIDIA und zusätzlicher Software möglich ist. Als zusätzliche Software muss das *CUDA Toolkit* in der Version 9.0 (Base Installer & alle Patches) von NVIDIA installiert werden[6]. Dabei muss darauf geachtet werden, dass der installierte Treiber der Grafikkarte mit der installierten Version des CUDA Toolkits kompatibel ist und bestenfalls auf die neueste Version aktualisiert wird. Die Kompatibilität kann in den *CUDA Toolkit* Release Notes[7] überprüft werden. Weiters muss die zu CUDA 9.0 passende Version der NVIDIA CUDA Deep Neural Network library (cuDNN)[8] heruntergeladen werden. Dafür ist eine kostenlose Mitgliedschaft bei *NVIDIA Developer* einzugehen. Der entzippte Ordner muss unter `C:\cudnn-9.0-windows10-x64-v7` abgelegt und der Ordner `C:\cudnn-9.0-windows10-x64-v7\cuda\bin` zu den System-Umgebungsvariablen hinzugefügt werden. Wurde all das umgesetzt, so kann TensorFlow in der GPU-Version verwendet werden.

[6] https://developer.nvidia.com/cuda-90-download-archive
[7] https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html
[8] https://developer.nvidia.com/cudnn

Kapitel 3

Stand der Technik – vergleichbare Arbeiten

Im Bereich der Objekterkennung innerhalb von Bildern gibt es Umsetzungen sowohl mit neuronalen Netzen als auch mit herkömmlicher Feature-Erkennung. Eine Arbeit, die sich mit Deep Learning Methoden zur Objekterkennung im Allgemeinen beschäftigt, ist jene von Agarwal et al. [2]. Sie befasst sich unter anderem mit Faster R-CNN, YOLO und SSD. Weitere vergleichbare Arbeiten gibt es im Bereich der Bilderkennung mit einem geringen Umfang von Trainingsdaten in Kombination mit Transfer Learning. Eine Arbeit, die sich mit Transfer Learning befasst, ist [65]. Vergleichbare Arbeiten lassen sich in diesem Fall in verschiedene Kategorien unterteilen. Dazu zählen unter anderem die Erkennung von Werbung in diversen Bereichen, wie zum Beispiel in Sportübertragungen, Firmenlogo-Erkennung im Allgemeinen sowie auch Objekt- und Personenerkennung im Straßenverkehr.

3.1 Erkennung von Werbung

Arbeiten im wohl ähnlichsten Anwendungsgebiet sind jene, die sich auch mit der Erkennung von Werbung beschäftigten. Hier veröffentlichten Hossari et al. in [21] eine Methode namens ADNet, die es ermöglicht, Werbung in Videos zu erkennen. ADNet setzt als Basis auf ein mit Bildern aus dem ImageNet Datenset antrainierten VGG19 Netzwerkmodell, das mittels Transfer Learning auf die Erkennung von Werbung neu trainiert wurde. Eine weitere interessante Arbeit von Intasuwan et al. wird in [25] beschrieben. Diese fokussiert sich auf die Erkennung von Werbetafeln sowie die Analyse von Text und Objekten auf der Tafel. Anhand dieser Informationen werden die relevantesten Webseiten zur Werbung als Resultat geliefert.

Ein Bereich, dem sich diverse Arbeiten widmen, ist die Erkennung von Werbung in Fernsehübertragungen. In [37] werden Werbesendungen mithilfe von CNNs erfasst und dokumentiert. Ein Fokus dieses Themengebietes liegt hier auf der Analyse von Sportübertragungen. Mit diesem Thema beschäftigten sich Cui et al. [6]. Die Technik, die dabei zur Identifikation von Werbung verwendet wird, beruht auf der Fast Hough Transformation und Text-Geometrie-Features. Chattopadhyay et al. [9] beschäftigen sich mit der Erkennung von Markenzeichen in Videos.

Im Gegensatz zu der Identifikation von Werbung im Allgemeinen versuchen folgende Arbeiten speziell Firmenlogos innerhalb von Bildern zu finden. Bao et al. [4] beschreiben eine Methode, die ein R-CNN zur Lösung des Problems verwendet. Eine weitere Arbeit, die sich mit der Erkennung von Firmenlogos beschäftigt, ist [11].

3.2 Erkennung von Schildern

Ein weiteres, ähnliches Anwendungsgebiet ist die Identifikation von Schildern im Allgemeinen. Genau mit diesem Thema beschäftigt sich die Arbeit [54] von Shen et al., die dieses Problem anhand von Kantenerkennung gelöst hat. Etwas tiefer auf die Erkennung von Schildern geht [62] von Yang et al. ein. Hier wird zusätzlich der Text der Anzeigetafel analysiert und übersetzt. Auch [61] beschäftigt sich mit der Übersetzung von Texten auf Schildern.

3.3 Erkennung von Objekten und Personen im Straßenverkehr

Ein weiterer Ansatzpunkt für vergleichbare Arbeiten ist das Einsatzgebiet. Es gibt viele Arbeiten, die sich mit Objekt- bzw. Personenerkennung im Straßenverkehr in Videos beschäftigen. So zeigt [36] von Li et al. das Erkennen von Fahrzeugen in Verkehrskameraaufzeichnungen. Dazu wird die Methode Faster R-CNN verwendet. In [22] wird ein System beschrieben, das eine Vielzahl von Objektklassen erkennt, die typisch für den Straßenverkehr sind. Dazu gehören unter anderem Fahrzeuge, Verkehrszeichen, Radfahrer und Fußgänger. Li et al. [38] beschreiben ein System, das auf stereoskopischen Bildern beruht. Hier analysiert ein R-CNN beide Bilder und sucht nach Fahrzeugen in diesen. Anhand dieser Informationen kann ein dreidimensionaler Begrenzungsrahmen um die Autos berechnet werden. In [12] wird eine Methode beschrieben, die ein R-CNN verwendet, um Motorräder zu erkennen. Dafür werden Kamerabilder von Drohnen verwendet. Cai et al. [7] benützen zum Erkennen von typischen Objekten im Straßenverkehr ebenfalls ein System, das auf einem R-CNN basiert. Das neuronale Netzwerk wurde mit Bildern aus dem COCO-Datensatz trainiert und mittels Transfer Learning auf das spezielle Anwendungsgebiet angepasst. Ein weiteres System, das auf Transfer Learning basiert, wird in [64] beschrieben.

Zahlreiche Arbeiten beschäftigen sich speziell mit dem Erkennen von Verkehrszeichen in Videostreams von On-Board Kameras in Fahrzeugen. So wird zum Beispiel in [44] genau auf diese Problemstellung eingegangen. Moutarde et al. [43] nehmen speziell auf die Erkennung von Geschwindigkeitsbeschränkungen Rücksicht. Zur Identifikation dieser Verkehrszeichen werden sowohl die Umrisse der Tafeln als auch eine Ziffernerkennung für deren Inhalte verwendet. Ozcan et al. [45] benützen Aggregate Channel Features und Chain Code Histogramme, um Verkehrszeichen erkennen zu können. Das soll den Vorteil haben, dass es nicht so rechenintensiv ist wie beispielsweise Systeme, die auf einem R-CNN beruhen. Jeon et al. verwenden in ihrem Ansatz Haar-like Features, um Verkehrszeichen zu erkennen. Dies wird in [27] genau beschrieben. In [67] wird ein Faster R-CNN eingesetzt, um Verkehrszeichen zu identifizieren. Reinders et al. [50] beschäftigen sich mit der Thematik, Verkehrszeichen speziell für Radfahrer zu kartografieren. Dazu werden mithilfe von Deep Learning die Straßenschilder erkannt

und zusätzlich die GPS Positionen gespeichert. Weitere Arbeiten, die sich mit dem Erkennen von Verkehrszeichen beschäftigen, sind [1, 8, 24, 28, 31, 35, 59, 63, 66].

Alternativ zu Verkehrszeichen geht es in [3] darum, Straßenschäden, wie zum Beispiel Schlaglöcher, anhand von On-Board Videos in Fahrzeugen zu erkennen. Weiters gibt es Arbeiten, die sich speziell mit dem Erkennen von Passanten im Straßenverkehr befassen. Eine solche Arbeit ist zum Beispiel [14], die ein Fast R-CNN zur Detektion verwendet. In [17] wird eine Methode beschrieben, die die Informationen aus erkannten Personen am Straßenrand verwendet, um deren Bewegungsflüsse zu analysieren. Es wird unterschieden, ob Personen von vorne oder von hinten erkannt wurden. Qian et al. [46] befassen sich damit, Verkehrshinweise, die direkt auf die Straße gezeichnet sind, zu erkennen.

Eine spezielle Anwendung von Schilder-Erkennung im Straßenverkehr ist die Identifikation von Kennzeichen. So beschäftigen sich Laroca et al. genau mit diesem Thema in [32]. Zur Detection wird hier die YOLO Methode verwendet. Auch [20] präsentiert eine Methode zur Nummerntafel-Erkennung, die auf dem YOLO Prinzip basiert.

Kapitel 4

Konzept

Im Folgenden ist das Konzept dieser Arbeit beschrieben. Dazu gehören die Methodik anhand eines Anwendungsfalles, die Definition von Werbung, die Auswahl der verwendeten Objekterkennungsmethoden und die Kriterien für den Erfolg dieser Erkennungsmethoden.

4.1 Methodik anhand eines Anwendungsfalles

Aufgabenstellung ist es, herauszufinden, ob Werbung im Generellen von neuronalen Netzen erkannt werden kann, da sich deren Aussehen, Inhalte und Kontext bzw. Hintergrund oft sehr stark voneinander unterscheiden. Weiters stellt sich die Frage, ob dies bereits mit einem kleinen Trainings-Datensatz (weniger als 1000 Bilder) möglich ist. Der Fokus der Arbeit wurde auf die Analyse von Dashcam-Videos gelegt, da das ein Anwendungsfall ist, bei dem Werbung sehr präsent ist. Grundsätzlich gibt es für die Umsetzung von Objekterkennung mit Deep Learning viele verschiedene Methoden. Welche für diese Arbeit gewählt wurden, ist in Abschnitt 4.3 beschrieben. Dazu muss dem neuronalen Netzwerk erst beigebracht werden, was Werbung eigentlich ist (s. Abschnitt 4.2). Für die Realisierung müssen ein Datensatz von Bildern erstellt und die Bereiche mit Werbung markiert werden. Dieser Vorgang wird *Labeling* genannt. Um den Datensatz für das Training des Netzwerkes möglichst klein halten zu können, wird Transfer Learning (s. Abschnitt 2.6) angewendet. Dieser Trainingsvorgang soll mit unterschiedlichen Einstellungen durchführt werden, damit möglichst gute Ergebnisse erzielt werden können. Das umfasst verschiedene Netzwerkarchitekturen, Initialgewichtungen und Lernraten. Sind die Trainingsvorgänge durchgeführt, werden die Ergebnisse mittels Validierungs-Datenset miteinander verglichen und die besten Einstellungen für weitere Bild- und Videoanalysen verwendet. Die Evaluierung dieser Vorhersagen ist in Abschnitt 4.4 ersichtlich. Die Erkennung von Werbung in Bildern oder Videos kann in weiterführenden Arbeiten dazu genutzt werden, um die Anzahl von Werbungen bzw. deren Position entlang der Straße zu erfassen. Es kann auch analysiert werden, wie lange sich diese Werbung im Blickfeld des Autofahrers befindet. Weiters kann das System genutzt werden, um Werbung gegebenfalls ausblenden zu können.

Abbildung 4.1: Beispielbild, bei dem Bildflächen, die als Werbung definiert sind, in Rot hervorgehoben sind. Diese Bildflächen wurden händisch annotiert. Bild adaptiert aus [68].

4.2 Definition von Werbung

Werbung ist ein Begriff, der sehr unterschiedlich definiert werden kann. Im Verlauf dieser Arbeit wird der Begriff für alle Bildbereiche verwendet, die mithilfe von Schrift, Logos oder Bildern auf etwas hinweisen. Diese können sowohl zweidimensional als auch dreidimensional bzw. entweder freistehend oder auf einem Hintergrund positioniert sein. Dazu gehören unter anderem:

- freistehende Werbetafeln,
- Plakate,
- dreidimensionale Schrift oder Logos auf Häuserwänden,
- auf Häuserwänden aufgemalte, zweidimensionale Schrift, Logos oder Bilder,
- Werbung auf Fahrzeugen, wie zum Beispiel LKW-Planen,
- Kundenstopper und
- Leuchtreklame.

Die Definition ist jedoch nicht auf diese Aufzählungspunkte beschränkt. Beispiele, bei denen Bildbereiche als Werbung definiert sind, werden in Rot hervorgehoben. Diese sind in den Abbildungen 4.1 und 4.2 zu sehen.

Abbildung 4.2: Beispielbild, bei dem Bildflächen, die als Werbung definiert sind, in Rot hervorgehoben sind. Diese Bildflächen wurden händisch annotiert. Bild adaptiert aus [68].

4.3 Auswahl der verwendeten Objekterkennungsmethoden und Deep Learning Frameworks

Für die praktische Umsetzung wurde entschieden, sich auf zwei Objekterkennungsmethoden zu fokussieren. Die Wahl fällt hier mit Hauptfokus auf Mask R-CNN und YOLOv3 für Vergleichswerte, die in Abschnitt 2.7 näher beschrieben werden. Mask R-CNN wurde gewählt, weil dieser Ansatz sehr gute Ergebnisse im Verglich zu anderen Methoden liefert. Weiters ist es möglich, pixelweise Segmentierungsmasken zu erstellen und so noch genauere Informationen zu erhalten, welche Bereiche des Bildes als Objekt kategorisiert werden. Dieser Ansatz benötigt verhältnismäßig lange Bildanalysezeiten. Im Gegensatz dazu setzt YOLOv3 auf eine möglichst schnelle Analyse eines Bildes und gehört in diesem Bereich zu den führenden Erkennungsmethoden. Beide gewählten Ansätze sind im Bereich der Objekterkennung sehr anerkannt und werden viel verwendet. Da der Fokus der beiden Methoden relativ unterschiedlich ist, hat es sich angeboten, diese miteinander zu vergleichen.

Es wurde entschieden, für YOLO eine in sich abgeschlossene Implementierung zu verwenden, die auf der ursprünglichen Realisierung des Erfinders basiert. Alternativ würde es auch hier Implementierungen in diversen Frameworks geben. Somit bezieht sich die Frage, welches Deep Learning Framework für die Umsetzung verwendet wird, nur auf Mask R-CNN. Grundsätzlich gibt es für Faster R-CNN bzw. Mask R-CNN in fast allen beschriebenen Deep Learning Frameworks (s. Abschnitt 2.8) eine Implementierung. So gibt es zum Beispiel Implementierungen für Keras [69] mit TensorFlow als Backend, Caffe2 [70], PyTorch [71], MXNet [72] und CNTK [73]. Die Implementierungen sind jedoch nicht nur auf diese beschränkt.

Weiters finden sich online viele Quellen, die auf diese Deep Learning Frameworks genauer eingehen und sie miteinander vergleichen. So hat zum Beispiel Hale [82] im September 2018 unter anderem die acht beschriebenen Deep Learning Frameworks (s. Abschnitt 2.8) auf ihre Verbreitung in Bezug auf sieben verschiedene Kategorien miteinander verglichen. Zu diesen Kategorien zählten:

- online Stellenangebote,
- KDnuggets-Nutzungsumfragen,
- Google-Suchanfragen,
- Medium-Artikel,
- Amazon-Bücher,
- ArXiv-Artikel und
- GitHub-Aktivitäten.

Dabei stellte sich TensorFlow in fast allen Kategorien als eindeutiger Sieger heraus. Keras wurde im Gesamtranking Zweiter. Andere Webseiten vergleichen diese Frameworks in Bezug auf ihre Funktionalität [81, 109]. Welches Deep Learning Framework für eine bestimmte Anforderung am besten geeignet ist, hängt jedoch von einer Reihe von Faktoren ab. Ist man ein Anfänger im Bereich Deep Learning, so ist ein Python-basiertes Framework wie Keras oft die beste Wahl. Ist man jedoch erfahren, so sollte bei der Auswahl auf die unterstützten Programmiersprachen, die Geschwindigkeit, den Ressourcenbedarf und -verbrauch zusammen mit der Kohärenz des trainierten Modells geachtet werden. Im weiteren Verlauf dieser Arbeit wird aufgrund der Verbreitung und der Einfachheit sowie der Tatsache, dass eine Implementierung von Mask R-CNN verfügbar ist, Keras in Kombination mit TensorFlow verwendet. Geschwindigkeit und die Verwendung von großen Datenmengen sind für diese Arbeit nicht relevant.

4.4 Kriterien für den Erfolg

Ziel der Arbeit ist es, sowohl mit Mask R-CNN als auch mit YOLOv3 zufriedenstellende Ergebnisse zu erreichen. Das bedeutet, dass Bildbereiche, die von einem Menschen zweifelsfrei als Werbung definiert werden (s. Abschnitt 4.2), zum größten Teil als solche erkannt werden kann. Dieser Wert wird auch als *Recall* bezeichnet und ist als

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{\text{korrekte Vorhersagen}}{\text{alle Grundwahrheiten}} \tag{4.1}$$

definiert. Weiters soll das Netzwerk keine anderen Bildbereiche fälschlicherweise als Werbung kategorisieren. Eine Aussage dazu kann mittels dem *Precision* Wert gegeben werden. Dieser ist als

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{\text{korrekte Vorhersagen}}{\text{alle Vorhersagen}} \tag{4.2}$$

definiert. *TP* steht in diesem Zusammenhang für *True Positive*, also alle korrekten Vorhersagen. *FP* steht für *False Positive*, also alle falschen Vorhersagen und *FN* für *False Negative*, also alle nicht erkannten Werbungen. Die Bewertung kann mittels Validierungs-Datenset durchgeführt und für verschiedene Vorhersagegenauigkeiten bestimmt werden. Die Vorhersagegenauigkeit wird über den sogenannten *Intersection over Union* (IoU)

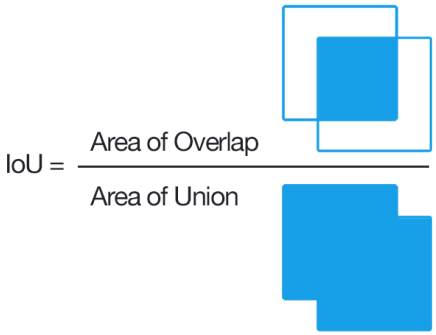


Abbildung 4.3: Grafische Darstellung der Intersection Over Union. Bildquelle [102]

Wert angegeben und lässt den Rückschluss zu, wie gut der Begrenzungsrahmen der Annotation mit dem Begrenzungsrahmen der Vorhersage übereinstimmt (s. Abbildung 4.3). Für die Definition einer Vorhersagegenauigkeit muss für die IoU ein Grenzwert angegeben werden. Dieser Schwellenwert sagt aus, ob eine Vorhersage als *TP* oder als *FP* gewertet wird. Meist wird für diesen Wert 0.5, 0.75 oder 0.95 gewählt. Die Exaktheit eines Netzwerkes wird meistens mit der Durchschnittsgenauigkeit angegeben. Sie wird mittels *Average Precision*[1] (AP) Wert angegeben. Dieser Wert ist als die vereinfachte Fläche unterhalb der Precision-Recall-Kurve, die sich aus den oben erwähnten Werten erstellen lässt, definiert. Kann das Netzwerk mehrere unterschiedliche Klassen kategorisieren, so wird dieser Wert für jede Klasse einzeln berechnet und für die Bewertung des gesamten Netzwerkes der Mittelwert daraus gebildet. Der Wert wird dann als *Mean Average Precision* (mAP) bezeichnet. In dieser Arbeit sind AP und mAP als identisch zu sehen, da das Netzwerk nur eine Klasse kategorisieren kann. Wie der Wert für die Durchschnittsgenauigkeit im Detail berechnet wird, kann in [13] nachgelesen werden. Eine weitere genaue Beschreibung ist unter [74] zu finden. Bei bekannten Wettbewerben schneidet Mask R-CNN zirka mit einem mAP-Wert von 60% ab [19]. YOLOv3 liefert hier Werte von in etwa $55\% - 60\%$ [48]. Dabei ist jedoch zu beachten, dass die Netzwerke bei solchen Wettbewerben weit mehr als nur eine Objektkategorie identifizieren können müssen. Deshalb liegen auch die Ziele dieser Arbeit ungefähr in diesem Bereich.

Dass diese Art der Bewertung jedoch in so mancher Hinsicht nicht wirklich sinnvoll ist, wird von Redmon und Farhadi [48] gezeigt. Weiters werden noch mögliche bessere Alternativen zur Bewertung eines Netzwerkes erwähnt.

[1] Dt. *Durchschnittsgenauigkeit*

Kapitel 5

Umsetzung

Das Konzept wurde mit Mask R-CNN (s. Abschnitt 2.7.1) und YOLOv3 (s. Abschnitt 2.7.2) praktisch umgesetzt. Zum Trainieren dieser Methoden wurden etwa 1000 Bilder, welche aus dem Internet bezogen wurden, gelabelt. Die Umsetzung dieser beiden Methoden wird nachfolgend erläutert.

5.1 Beschaffung der Bilder zum Trainieren der Objekterkennungsmethoden

Um ein neuronales Netzwerk auf die Erkennung von bestimmten Objekten zu trainieren, muss dieses mit annotierten Trainingsdaten versorgt werden. Die für diese Arbeit verwendeten Bilder zum Trainieren des neuronalen Netzwerkmodells wurden von der Webseite ImageNet[1] bezogen, die unter anderem von der Universität Stanford und der Universität Princeton betrieben wird. ImageNet verfügt über mehr als 14 Millionen referenzierte Bilder, die öffentlich zur Verfügung stehen. Für den Suchbegriff „Billboard“ wurden etwa 1000 Bilder auf Flickr[2] gefunden. Diese wurden mittels Massen-Downloads heruntergeladen und per Hand weiter selektiert, da manche der Bilder nicht für das Labeling-Verfahren (s. Abschnitt 5.2.1) verwendbar waren.

5.2 Umsetzung des Mask R-CNN-Modells

Im Folgenden wird die Umsetzung von Mask R-CNN im Detail beschrieben. Dazu gehören die Beschreibung des Labeling-Verfahrens, die verwendete Software und deren Setup sowie die genaue Umsetzung des Modells in Python inklusive dem Trainingsprozess.

5.2.1 Labeln der Bilder für Mask R-CNN

Es gibt viele verschiedene Applikationen, mit denen Bilder online annotiert werden, um von neuronalen Netzen zum Trainieren benützt werden zu können. Diese sind zum Beispiel:

[1] http://www.image-net.org/
[2] https://www.flickr.com/

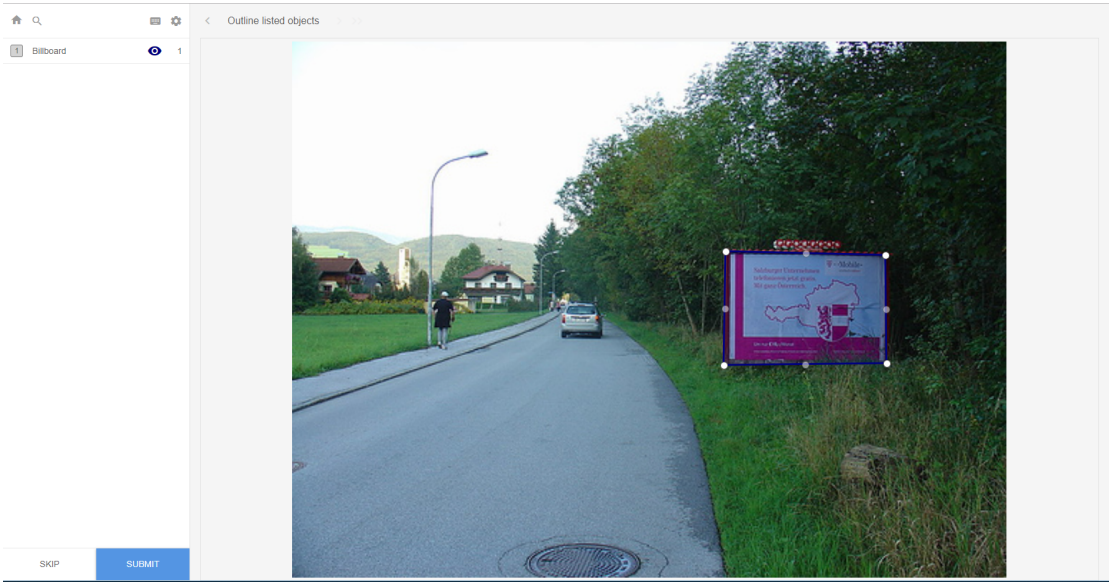


Abbildung 5.1: Darstellung des verwendenten Labeling-Werkzeuges LabelBox. Die Plakatwand im Bild wurde mittels Polygone genau markiert.

- Labelbox[3],
- LabelMe[4],
- DataTurks[5] oder
- RectLabel[6] (nur MacOS).

Bei dieser Arbeit wurde Labelbox verwendet, da dieses Tool kostenlos online verfügbar ist und alle grundlegenden Funktionen unterstützt. Dazu müssen alle Bilder, die annotiert werden sollen, auf die Webseite hochgeladen werden. Der Nachteil bei der Gratisversion ist, dass diese nicht verschlüsselt sind und somit jeder, der über den zugehörigen Link zum Bild verfügt, Zugriff auf die Bilddatei hat. Deshalb kann sie nicht für heikle Daten verwendet werden. Das Annotieren der Bilder kann mittels Begrenzungsrahmen, Polygonen oder pixelweise durchgeführt werden. Außerdem besteht die Möglichkeit, gesamte Bilder zu klassifizieren [90]. Bei der Verwendung von Mask R-CNN muss die Methode mittels Polygonen zum Labeln der Bilder verwendet werden. Ein Beispielbild des Label-Verfahrens ist in Abbildung 5.1 dargestellt. Die Annotationen der gelabelten Bilder können in verschiedenen Varianten als `JSON` oder `CSV`-Datei heruntergeladen werden. Insgesamt konnten 960 Bilder verwendet werden. Davon wurden 727 Bilder für einen Trainings-Datensatz und 233 Bilder für einen Validierungs-Datensatz eingesetzt.

[3] https://labelbox.com/
[4] http://labelme.csail.mit.edu/Release3.0/
[5] https://dataturks.com/
[6] https://rectlabel.com/

5.2.2 Software und Setup

Zur Umsetzung dieses Projektes wurde Keras (s. Abschnitt 2.8.2) mit TensorFlow (s. Abschnitt 2.8.1) als Backend verwendet. Um diese Frameworks verwenden zu können, muss Python in der Version 3.6 oder einer neueren Version installiert werden. Dazu wird idealerweise *Anaconda* installiert [77]. Soll die GPU zum Trainieren der neuronalen Netze verwendet werden, muss noch zusätzliche Software von NVIDIA heruntergeladen werden (s. Abschnitt 2.9). Ist Anaconda auf dem System installiert, kann mithilfe der Anaconda Prompt (ähnlich der Kommandozeile von Windows) eine Anaconda-Umgebung erstellt und aktiviert werden. Erstellt werden muss eine Umgebung nur einmal, aktiviert werden jedoch bei jeder Verwendung. Die Aktivierung kann auch innerhalb des Anaconda GUI (Anaconda Navigator) durchgeführt werden. In dieser Anaconda-Umgebung können die benötigten Frameworks und Python-Module, wie zum Beispiel TensorFlow, geladen und installiert werden:

```
conda create -n NameDerEntwicklungsumgebung pip python=3.6
activate NameDerEntwicklungsumgebung
pip install tensorflow-gpu
pip install keras
```

Bei der Nutzung von Anaconda können Jupyter Notebooks als Entwicklungsumgebung verwendet werden. Es handelt sich hier um eine Python-Programmierumgebung, die im Browser eingesetzt wird. In einem Jupyter Notebook hat man Zugriff auf lokale Python-Dateien sowie auf installierte Module innerhalb der Anaconda-Umgebung. Ein Vorteil von Jupyter Notebooks ist, dass Codeabschnitte für sich ausgeführt werden können, wobei jener Code, der bereits zuvor ausgeführt wurde, verwendbar ist. Weiters können Dokumentationsabschnitte zwischen Codeabschnitten eingefügt werden.

5.2.3 Implementierung des Modells in Python mittels Keras

Zur Objekterkennung in einem Bild wurde das Mask Region-Based CNN-Verfahren gewählt. Als Grundlage wurde hier die Implementierung, die unter [69] verfügbar ist, verwendet. Als Convolutional Neural Network kann zwischen ResNet50 und ResNet101 gewählt werden. Für beide Netzwerkarchitekturen sind bereits antrainierte Gewichtungen des COCO-Datensets[7] sowie des ImageNet-Datensets vorhanden. So haben die Gewichtungen bereits viele Features erlernt. Um die Erkennung von Objekten unterschiedlicher Größe zu verbessern, wurde ein *Feature Pyramid Network* (FPN) [39] verwendet. FPN erweitert die *Standard Feature Extraction Pyramid* um eine zweite Pyramide, die High-Level-Features von der ersten Pyramide in all ihre Ebenen aufnimmt. Auf diese Weise haben alle Ebenen der Pyramide auf Lower-Level Features und auf Higher-Level Features Zugriff. Im Umfang dieser Arbeit wurden verschiedenste Kombinationen von Hyperparametern, Netzwerkarchitektur und Initial-Gewichtungen kombiniert, um das Modell zu trainieren. Die Ergebnisse können in Kapitel 6 nachgelesen werden. Der Output Layer des Modells wurde ersetzt, um ausschließlich Plakatwände von Hintergrundbereichen unterscheiden zu können. Mittels Transfer Learning wurde anschließend die eigens erstellte Datenbank zum Trainieren von Plakatwänden verwendet.

[7] http://cocodataset.org/

Abbildung 5.2: Darstellung eines Beispielbildes aus dem Trainings-Datenset und dessen durch die Implementierung von Mask R-CNN erstellte Masken.

Verwendung eines eigenen Datensets

Grundsätzlich gibt es keine einheitliche Formatierung, wie die Annotationen gespeichert werden. Die in dieser Arbeit verwendeten Segmentierungsmasken können zum Beispiel durch eine `PNG`-Datei oder durch Polygon-Punkte, die in einer `JSON` oder `CSV`-Datei eingetragen sind, umgesetzt werden. Um hier möglichst flexibel zu sein, bietet die Implementierung von Mask R-CNN eine Datenset-Klasse an, von der abgeleitet werden kann und bestimmte Funktionen überschrieben werden können, um diese auf das eigene Datenset anzupassen. So wird in einer Funktion `load_dataset()` die `JSON`-Datei gelesen, die Annotation extrahiert und für die weitere Verarbeitung aufbereitet. Dabei kann zwischen Trainings- und Validierungs-Datensatz unterschieden werden. In einer Funktion `load_mask()` werden anhand der Polygone Bitmap-Masken für jedes Objekt im Bild erstellt. In Abbildung 5.2 wird ein Bild und dessen erstellte Masken angezeigt. Die Implementierung generiert aus diesen Masken eigenständig die zugehörigen Begrenzungsrahmen, selbst wenn diese in der Annotations-Datei enthalten sind. Das wird so gehandhabt, weil man möglichst unabhängig vom Datenset sein will und diese Information nicht in jeder Datenset-Annotation vorhanden ist. Auch das Laden der Bilder selbst wird von der Basisklasse übernommen. Zum Trainieren des Modells wird auf Data Augmentation verzichtet, da bereits mit dem vorhandenen Datenset akzeptable Ergebnisse erzielt werden.

Verwendung von Mini-Masken

Wird für jedes Objekt im Bild eine eigene Maske vom Netzwerk erstellt, kann es schnell zu Speicherproblemen kommen, wenn sie in Originalgröße gespeichert werden. Weil Numpy[8] ein Byte Speicher pro Booleschen Wert, also pro Pixel, benötigt, hätte jede Maske bei einer Auflösung von beispielsweise 1024×1024 Pixel eine Speichergröße von

[8] https://www.numpy.org/

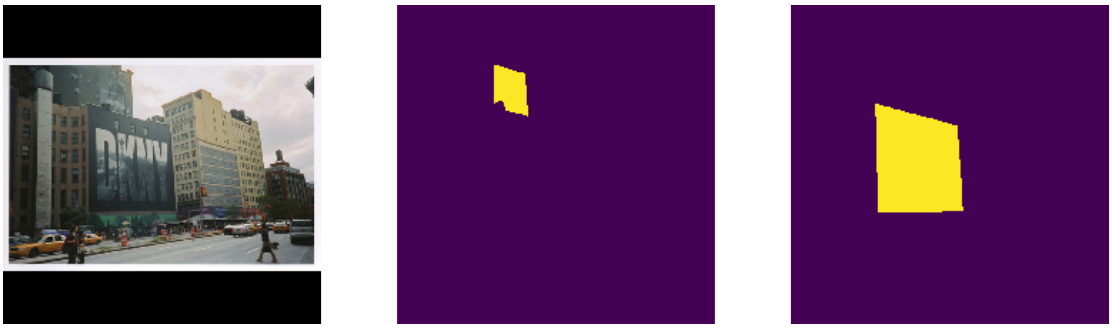


Abbildung 5.3: Darstellung eines Beispielbildes und dessen erstellte Masken in Originalgröße.

Abbildung 5.4: Qualität der Maske in Originalgröße anhand eines Beispielbildes.

einem MB. Wären 100 Objekte in einem Bild, so würden allein die Masken für ein einziges Bild 100 MB Speicher benötigen. Das sollte jedoch vermieden werden. Um Speicher zu sparen, ist es möglich, die Masken auf das Objekt zugeschnitten und in niedriger Auflösung zu erstellen. Für diese Arbeit wurde die vordefinierte Größe von 56×56 Pixel beibehalten. Die verkleinerten Masken werden als Mini-Masken bezeichnet. Um hier weniger Informationen zu verlieren, werden sie nicht binär abgebildet, sondern mittels Gleitkommazahlen repräsentiert. Der Unterschied zwischen Masken in Originalgröße und skalierten Minimasken kann in Abbildung 5.3 und in Abbildung 5.5 verglichen werden. Der Qualitätsunterschied bei der Überlagerung von Originalbild und Maske ist in Abbildung 5.4 und Abbildung 5.6 zu sehen.

Abbildung 5.5: Darstellung eines Beispielbildes und dessen erstellte Mini-Masken.

Abbildung 5.6: Qualität der skalierten Mini-Maske in Originalgröße anhand eines Beispielbildes.

Konfiguration

Auch für die Konfiguration des Netzwerkes ist eine Basisklasse in der verwendeten Implementierung verfügbar, die die meisten Hyperparameter des Netzwerkes enthält. Diese Parameter können mit einer abgeleiteten Klasse für das eigene Datenset angepasst werden, bzw. kann diese Datei mit zusätzlichen Parametern erweitert werden. Für das Training wird das Bild immer auf eine vorgegebene Größe, die in der Konfigurationskasse definiert ist, transformiert. In diesem Fall wird jedes Bild auf 1024×1024 Pixel skaliert. Ist ein Bild nicht quadratisch, so wird die Aspekt Ratio des eigentlichen Bildes beibehalten und das neue Bild durch *Zero Padding* auf 1024×1024 erweitert. Das ist in Abbildung 5.7 zu sehen.

Trainieren von Mask R-CNN mittels Transfer Learning

Beim Trainieren des Netzwerkes wurden acht verschiedene Kombinationen von Netzwerkarchitektur, Initialgewichtungen und Lernrate gewählt. Bei der Netzwerkarchitek-

Abbildung 5.7: Darstellung eines Beispielbildes mit Zero Padding. Das Bild wurde mit Nullwerten so erweitert, sodass das Bild quadratisch ist.

tur wurde ResNet50 bzw. ResNet101 getestet. Bei den Initialgewichtungen wurden Gewichtungen verwendet, die zum einen mit dem COCO-Datenset und zum anderen mit dem ImageNet-Datenset antrainiert wurden. Für die Lernrate wurden die Werte 0.001 und 0.005 gewählt. Diese Einstellungen können in der Netzwerk-Konfiguration definiert werden.

Grundsätzlich kann die Exaktheit eines Netzwerks zum einen über den Fehler (*Loss*) und zum anderen über die Durchschnittsgenauigkeit angegeben werden. Der Fehler ist ein Wert, der während des Trainings Aufschluss über den Lernprozess des Netzwerkes gibt, und wird in absoluten Zahlen angegeben. Hier kann zwischen Fehlern im Trainings-Datenset und Fehlern im Validierungs-Datenset unterschieden bzw. verglichen werden. Ziel des Trainings ist es, den Fehler, soweit es geht, zu verringern (s. Abschnitt 2.2.4). Bei zu vielen Trainingsepochen kann jedoch Overfitting (s. Abschnitt 2.3.1) auftreten. Das Training sollte vor diesem Phänomen beendet werden. Die genauen Fehler nach jeder Epoche der Trainingsdurchläufe für das Trainings- und Validierungs-Datenset sind in Tabelle 5.1 ersichtlich. `T_Loss` steht hier für den Fehler im Trainings-Datenset und `V_Loss` für den Fehler im Validierungs-Datenset. Diese Werte errechnen sich jeweils aus der Summe von `RPN_Class_Loss`, `RPN_BBox_Loss`, `MRCNN_Class_Loss`, `MRCNN_BBox_Loss` und `MRCNN_Mask_Loss`. Auf die Berechnung der einzelnen Summanden wird in dieser Arbeit nicht näher eingegangen. Bei den Trainingsdurchläufen wurde das Netzwerk jeweils über 5 Epochen trainiert, da ab diesem Zeitpunkt Anzeichen von

Tabelle 5.1: Lernerfolge nach jeder Epoche aller Trainingsdurchläufe anhand des Test- und Validierungsfehlers. Die Trainingsdurchläufe sind nach Architektur, Initialgewichtungen und Lernrate aufgeteilt. Der errechnete Fehler soll sowohl für das Test- als auch für das Validierungs-Datenset möglichst gering sein. Das beste Ergebnis ist in Rot hervorgehoben.

	ResNet101								ResNet50							
	COCO-Datenset				ImageNet-Datenset				COCO-Datenset				ImageNet-Datenset			
	LR = 0.001		LR = 0.005		LR = 0.001		LR = 0.005		LR = 0.001		LR = 0.005		LR = 0.001		LR = 0.005	
	T_Loss	V_Loss	T_Loss	V_Loss	T_Loss	V_Loss	T_Loss	V_Loss	T_Loss	V_Loss	T_Loss	V_Loss	T_Loss	V_Loss	T_Loss	V_Loss
Epoch 1	0.9975	0.7154	1.2728	0.8916	2.3219	1.5663	2.4635	1.7395	1.1766	0.8319	1.3250	1.0202	2.1327	1.5587	1.8565	1.0846
Epoch 2	0.7203	0.6224	1.2097	0.8638	1.5770	1.3586	1.8568	1.5440	0.8882	0.8515	1.1578	0.9573	1.3332	1.2428	1.2752	1.0565
Epoch 3	0.6132	0.5888	1.0823	0.9643	1.3251	1.2709	1.6397	1.2112	0.8028	0.7634	1.1051	1.0581	1.0452	1.1270	1.1405	1.0094
Epoch 4	**0.5455**	**0.5376**	1.0852	0.7583	1.1814	1.1305	1.4815	1.2096	0.7262	0.7309	1.1578	0.9147	0.8692	1.0797	1.1262	1.0014
Epoch 5	0.5053	0.5758	1.0952	2.1120	1.0579	1.1990	1.5988	1.0984	0.6605	0.8069	1.1299	2.4816	0.7880	1.0798	1.0628	1.1375

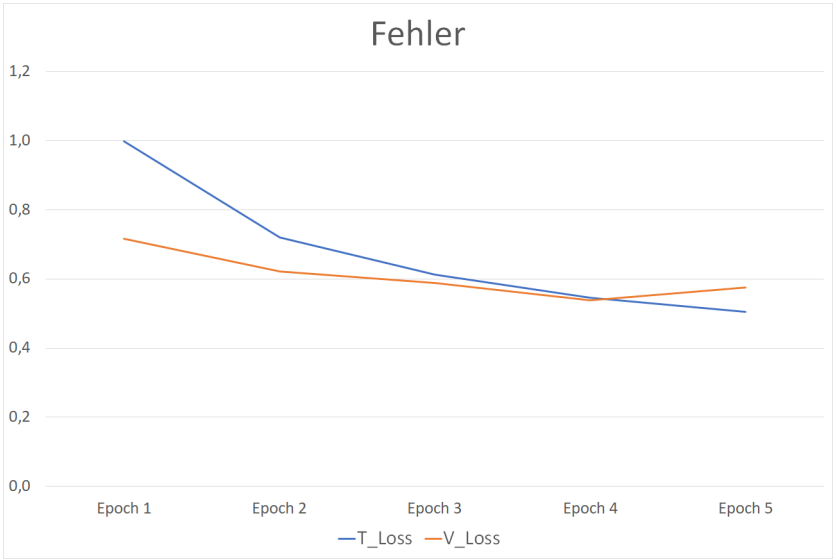


Abbildung 5.8: Grafische Darstellung des Lernerfolges des Trainingsdurchlaufes mit dem besten Ergebnis (ResNet101, COCO-Datenset, LR = 0.001).

Overfitting auftraten. In Abbildung 5.8 wird der Verlauf des Trainingserfolges mit der besten Kombination (ResNet101, COCO-Datenset, LR = 0.001) grafisch dargestellt. Für die Berechnung der Durchschnittsgenauigkeit (s. Abschnitt 4.4) wird im Umfang dieser Arbeit ein Bildbereich als Vorhersage gewertet, wenn der Konfidenzwert dieses Bereiches 0.8 überschreitet. Für alle Trainingskonfigurationen und nach Abschluss der vierten Epoche wurde der mAP-Wert mit einem IoU-Grenzwert von 0.5 berechnet. Diese Werte werden in Tabelle 5.2 gezeigt. Für die Kalkulation des Wertes wurden alle Bilder des Validierungs-Datensatzes verwendet.

Die Interpretation der unterschiedlichen Kombination ist in Abschnitt 7.1.1 im Detail beschrieben. Im weiteren Verlauf dieser Arbeit werden die Umsetzung und die Resultate von Mask R-CNN, die mit dem besten Trainingsergebnis erzielt wurden, beschrieben. Dieses beste Ergebnis ist in Tabelle 5.2 hervorgehoben.

Tabelle 5.2: Durchschnittsgenauigkeit (mAP) aller Trainingsdurchläufe zum Zeitpunkt nach Abschluss der vierten Epoche bei einem IoU-Wert von 0.5. Die Trainingsdurchläufe sind nach Architektur, Initialgewichtungen und Lernrate aufgeteilt. Diese Werte sollen möglichst hoch sein. Das beste Ergebnis ist in Rot hervorgehoben.

	ResNet101				ResNet50			
	COCO-Datenset		ImageNet-Datenset		COCO-Datenset		ImageNet-Datenset	
	LR = 0.001	LR = 0.005	LR = 0.001	LR = 0.005	LR = 0.001	LR = 0.005	LR = 0.001	LR = 0.005
mAP @ IoU = 0.5 (E = 4)	**0.945**	0.935	0.808	0.736	0.894	0.867	0.845	0.907

Funktionalität des Netzwerks im Detail

Im Folgenden werden die einzelnen Zwischenschritte von Mask R-CNN bei der Analyse eines Bildes im Detail erläutert.

Region Proposal Network (RPN): Auch Mask R-CNN nutzt wie Faster R-CNN zur Identifikation der Interessensregionen ein Region Proposal Network (RPN). Dieses scannt ein Bild nach dem *Sliding Window Prinzip*, in dem das Bild in Zellen aufgeteilt wird und nach Arealen im Bild sucht, die ein Objekt enthalten könnten. Die Regionen, die vom RPN analysiert werden, werden Anker genannt. Durch die Verschiebung zwischen den Zellen entstehen Überlappungen, die eine bestmögliche Abdeckung des Bildes garantieren. Diese Areale können unterschiedliche Größe und Aspekt-Ratio aufweisen. Die Werte werden bereits im Vorhinein in der Netzwerkkonfiguration definiert und sind für jede Zelle identisch. Bei dieser Arbeit wurde mit Anker-Größen von 32, 64, ... bzw. 512 Pixeln gearbeitet. Die Aspekt-Ratio der Anker wurden auf 0.5, 1 und 2 festgelegt. Dies wird in Abbildung 5.9 (a) für eine Zelle gezeigt. Genauer gesagt, wird nicht das Bild selbst, sondern werden dessen Featuremaps des CNNs gescannt. Ein Anker wird als Vordergrund (Objekt), Hintergrund oder Neutral kategorisiert. Dafür werden bestimmte Schwellenwerte der Wahrscheinlichkeit verwendet. In Abbildung 5.9 (b) werden die 50 Anker mit der höchsten Wahrscheinlichkeit dargestellt. Ein Anker, der als Vordergrund kategorisiert wurde, wird auch als positiver Anker bezeichnet. Diese Anker werden höchstwahrscheinlich ein Objekt nicht perfekt umschließen. Daher berechnet das RPN Delta-Werte für die X- und Y-Position sowie für die Höhe und die Breite des Ankers, um diese besser an das Objekt anzupassen. In Abbildung 5.9 (c) sieht man die 50 besten ursprünglichen Anker inklusive deren angepassten Regionen. Abbildung 5.9 (d) zeigt ausschließlich die angepassten Regionen der 50 vielversprechendsten Anker. In Abbildung 5.9 (e) werden sie am Rand des Bildes begrenzt. Überschneiden sich mehrere Positive Anker in einem zu großen Bereich, so werden all diese, bis auf den mit dem höchsten Wert, verworfen. Dieses Verfahren wird *Non-Max Suppression* genannt. In Abbildung 5.9 (f) werden die 50 besten verbleibenden Regionen nach Anwendung des Non-Max Suppression-Verfahrens gezeigt. Hier werden zusätzlich neue Regionen angeführt, die in den vorherigen Abbildungen nicht dargestellt werden. Von den vorher dargestellten Regionen wurden einige verworfen und somit sind andere Regionen nachgerückt. Die vielversprechendsten Anker werden als Interessensregionen (Region of Interest – RoI) weiter verwertet. Die Anzahl der weitergegebenen Regionen kann in der Konfigurationsklasse eingestellt werden. Im Fall dieser Arbeit liegt der Wert bei 1000.

(a) (b) (c) (d) (e) (f)

Abbildung 5.9: Grafische Darstellung aller wichtigen Zwischenschritte des RPN. Alle erstellten Anker einer Zelle des Bildes (a), die 50 wahrscheinlichsten Anker, die ein Objekt enthalten (b), die 50 wahrscheinlichsten Anker inklusive deren angepassten Regionen (c), die angepassten Regionen der 50 wahrscheinlichsten Anker (d), die an den Bildrand angepassten Regionen der 50 wahrscheinlichsten Anker (e), die 50 wahrscheinlichsten Regionen nach Anwendung des Non-Max Suppression-Verfahrens (f).

ROI Klassifikator & Bounding Box Regressor: In diesem Bereich des Netzwerkes werden die vom RPN kommenden Interessensregionen weiterverarbeitet. Hier werden all die Regionen gleichwertig behandelt. Diese werden zum einen zur Bestimmung der Objektklasse verwendet, zum anderen zur genauen Bestimmung des Begrenzungsrahmen. Im Unterschied zum RPN kann das Netzwerk nun nicht mehr nur zwischen Vordergrund und Hintergrund unterscheiden, sondern eine Region einer bestimmten Objektklasse zuordnen. Es besteht auch die Möglichkeit, eine Region erneut als Hintergrund einzuordnen. In Abbildung 5.10 (a) können 50 zufällige Anker, die vom RPN erhalten wurden, gesehen werden. Auch hier werden im nächsten Schritt diese Interessensregionen erneut an die Größe und Position der Objekte adaptiert. Diese Anpassung des Begrenzungsrahmens ist der Methode des RPN sehr ähnlich. Wurden die Regionen vom Netzwerk angeglichen, so werden all jene verworfen, die als Hintergrund bzw. als objektlos kategorisiert wurden. Fünf dieser angepassten und gefilterten Regionen werden in Abbildung 5.10 (b) gezeigt. In einem weiteren Selektionsschritt werden alle Regionen verworfen, deren Konfidenzwert unterhalb eines gewissen Schwellenwertes liegt. Fünf zufällig ausgewählte, übrig gebliebene Regionen werden in Abbildung 5.10 (c) gezeigt. Dieser Schwellenwert kann in der Konfigurationsklasse definiert werden. Schlussendlich wird das Non-Max Suppression-Verfahren erneut angewandt, um Regionen, die sich zum Großteil überschneiden, auszusortieren. In Abbildung 5.10 (d) sind die übrig gebliebenen Regionen und deren angepasste Varianten dargestellt. In Abbildung 5.10 (e) sieht man das Ergebnis nach angewendetem ROI Klassifikator und Bounding Box Regressor.

Klassifikatoren können mit unterschiedlich großen Interessensregionen schlecht umgehen, da sie im Normalfall eine fixe Inputgröße benötigen. Die Regionen, die man vom RPN erhält, unterscheiden sich jedoch in Bezug auf die Größe voneinander. Somit wird *RoI Pooling* benötigt. Beim RoI Pooling wird ein Teil der Featuremap ausgeschnitten und auf eine bestimmte Größe skaliert. Im ursprünglichen Beitrag [19] wird das als ROIAlign bezeichnet. Hier wird die Featuremap an verschiedenen Stellen abgetastet und eine bilineare Interpolation durchgeführt. Aufgrund der Einfachheit wird in dieser Umsetzung jedoch die `crop_and_resize()`-Funktion[9] von TensorFlow verwendet.

Masken-Segmentierung: Der Maskenzweig ist ein Convolutional Netzwerk, das die positiven Regionen des RoI Klassifikators verwendet, um Masken der Objekte zu generieren. Die generierten Mini-Masken werden später wieder auf die Größe der Interessensregionen skaliert, damit diese zum Originalbild passen. In Abbildung 5.11 kann zum Vergleich die manuell segmentierte Maske aus dem Validierungs-Datenset mit der vom Netzwerk erstellten Mini-Maske und der erstellten Maske in Originalgröße verglichen werden. In Abbildung 5.12 wird das finale Ergebnis des Mask R-CNNs inklusive der Maske gezeigt.

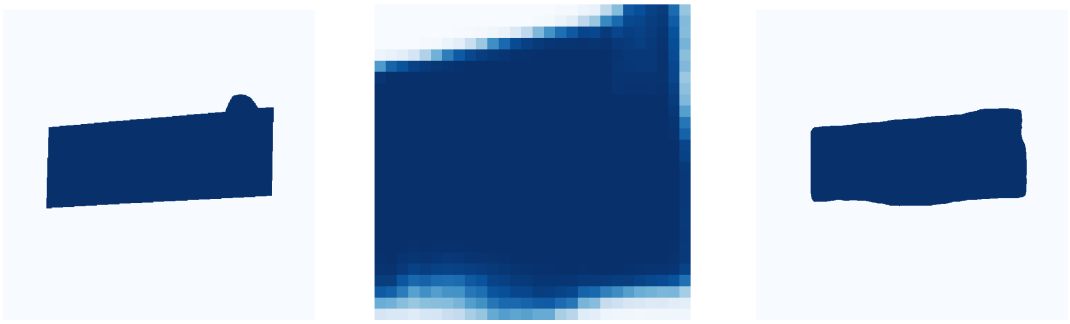
5.3 Umsetzung mit YOLOv3

Die vom Erfinder von YOLO veröffentlichte Implementierung von YOLOv3[10] ist nur bei der Nutzung eines Linux-Systems verwendbar. Jedoch gibt es davon abgespaltene

[9] https://www.tensorflow.org/api_docs/python/tf/image/crop_and_resize
[10] https://github.com/pjreddie/darknet

(a) (b) (c) (d) (e)

Abbildung 5.10: Grafische Darstellung aller wichtigen Zwischenschritte des RPN. 50 zufällige Interessensregionen, die vom RPN übernommen wurden (a), fünf zufällige Interessensregionen und deren angepasste Bereiche nach Filterung der als Hintergrund klassifizierten Regionen (b), fünf zufällige Interessensregionen und deren angepasste Bereiche nach Filterung von Regionen mit geringer Wahrscheinlichkeit (c), letzte übrig gebliebene Region und deren angepasster Bereich nach Anwendung des Non-Max Suppression-Verfahrens (d), finale Darstellung der erkannten Werbetafel mit Begrenzungsrahmen (e).



(a) (b) (c)

Abbildung 5.11: Grafische Darstellung diverser Maskenversionen. Maske aus dem Validierungs-Datenset (a), erstellte Mini-Maske (b), erstellte Maske in Originalgröße (c).

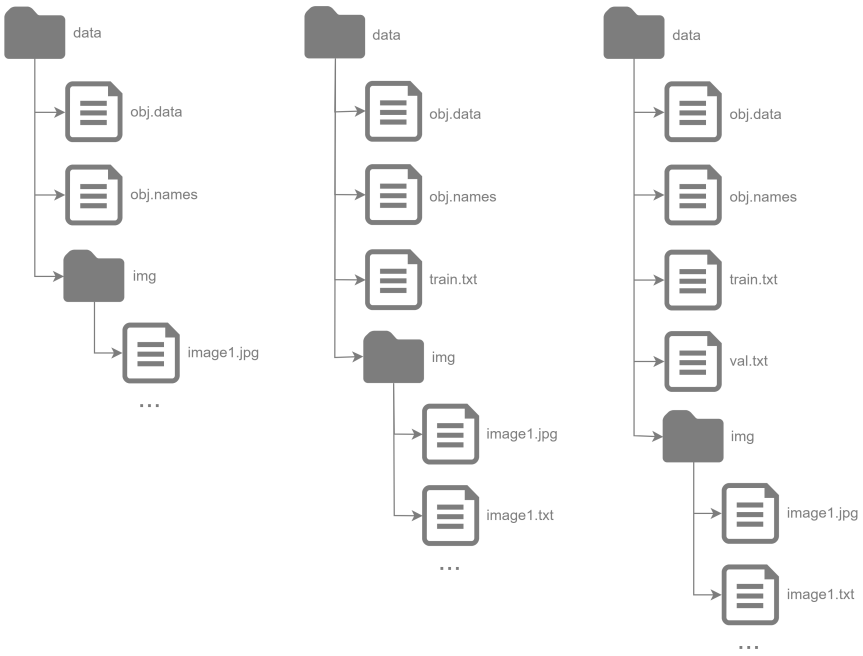
Abbildung 5.12: Darstellung des finalen Ergebnisses der Mask R-CNN-Methode inklusive Segmentierungsmaske.

Versionen, die eine Nutzung unter Windows ermöglichen. Die verwendete Implementierung, die dieses Kriterium erfüllt, ist auf GitHub[11] verfügbar. Um YOLOv3 mit einem eigenen Datenset nutzen zu können, müssen die Bilder mit Begrenzungsrahmen gelabelt werden. Weiters muss für die Verwendung von YOLOv3 zusätzlich OpenCV[12] am System verfügbar sein. Im Zuge des Labeling-, Trainings- und Analysevorganges werden Ordnerstrukturen vorgeschlagen, damit diese auch mit den gezeigten Kommandozeilenbefehlen übereinstimmen. Diese Struktur kann theoretisch geändert werden, infolgedessen müssen jedoch auch die Kommandozeilenbefehle bzw. Dateiinhalte dementsprechend abgeändert werden.

5.3.1 Labeln der Bilder für YOLOv3

Zum Annotieren der Bilder für YOLOv3 wurde ein Programm namens Yolo_Mask[13] verwendet, das vom selben Entwickler stammt wie die verwendete Implementierung von YOLOv3. Für die Verwendung des Programmes muss das GitHub-Repository geklont und der `Include`- bzw. der `Libary`-Ordner von OpenCV mit der Visual Studio Solution verknüpft werden. Die genaue Anleitung dazu findet sich in der `ReadMe`-Datei des Projektes. Sind diese Schritte erledigt, so kann das Projekt erstellt werden. Um eigene Bilder labeln zu können, muss der Ordner `data`, wie in Abbildung 5.13 (a) dargestellt, angepasst werden. In der Datei `obj.data` muss die Anzahl der Klassen auf die eigenen Anforderungen angepasst werden. Der Hintergrund zählt nicht als eigene Klasse. Der restliche Dateiinhalt kann beibehalten werden. In die Datei `obj.names` müssen alle gewünschten Klassennamen eingetragen werden. Jede Kategorie ist in eine separate Zeile zu schreiben. Im Unterordner `img` müssen alle zu labelnden Bilder abgespeichert werden. Um das Labeling-Programm schließlich auszuführen, muss die Datei `x64/Release/yolo_mark.cmd` ausgeführt werden. Dabei wird mit der Maus ein Begrenzungsrahmen über das zu labelnde Objekt gezeichnet. Das ist in Abbildung 5.14 dargestellt. Wechselt man zum nächsten Bild, ist der Annotationsvorgang des aktuellen Bildes abgeschlossen und es wird eine `.txt`-Datei mit demselben Namen des Bildes im Unterordner `img` erstellt, die die Annotationsinformationen enthält. Diese sind folgendermaßen strukturiert: `<Objektklasse> <X_Zentrum> <Y_Zentrum> <Breite> <Höhe>`. Die Objektklasse ist eine Zahl im Bereich von 0 bis (Klassenanzahl − 1). Die Werte für X_Zentrum, Y_Zentrum, Breite und Höhe sind relative Werte zur Höhe bzw. Breite des gesamten Bildes. Daher liegen diese immer im Bereich zwischen 0 und 1. Weiters wird eine Datei namens `train.txt` erstellt, die den relativen Pfad zu allen gelabelten Bildern enthält. Die Ordnerstruktur nach Beendigung des Labelingprozesses ist in Abbildung 5.13 (b) zu sehen. Die Datei kann manuell in ein Traingings- und Validierungs-Datenset aufgeteilt werden. Dafür wird ein Teil des Inhalts der Datei `train.txt` in eine Datei `val.txt` verschoben. Diese Ordnerstruktur ist in Abbildung 5.13 (c) dargestellt und kann zum Trainieren von YOLOv3 verwendet werden.

[11] https://github.com/AlexeyAB/darknet
[12] https://opencv.org/releases/
[13] https://github.com/AlexeyAB/Yolo_mark



(a) (b) (c)

Abbildung 5.13: Darstellung der Ordner- und Dateihierarchie. Diese ist vor dem Labeling (a), nach abgeschlossenem Label-Prozess (b) und nach manueller Aufteilung in einem Trainings- und Validierungs-Datensatz (c) zu sehen.

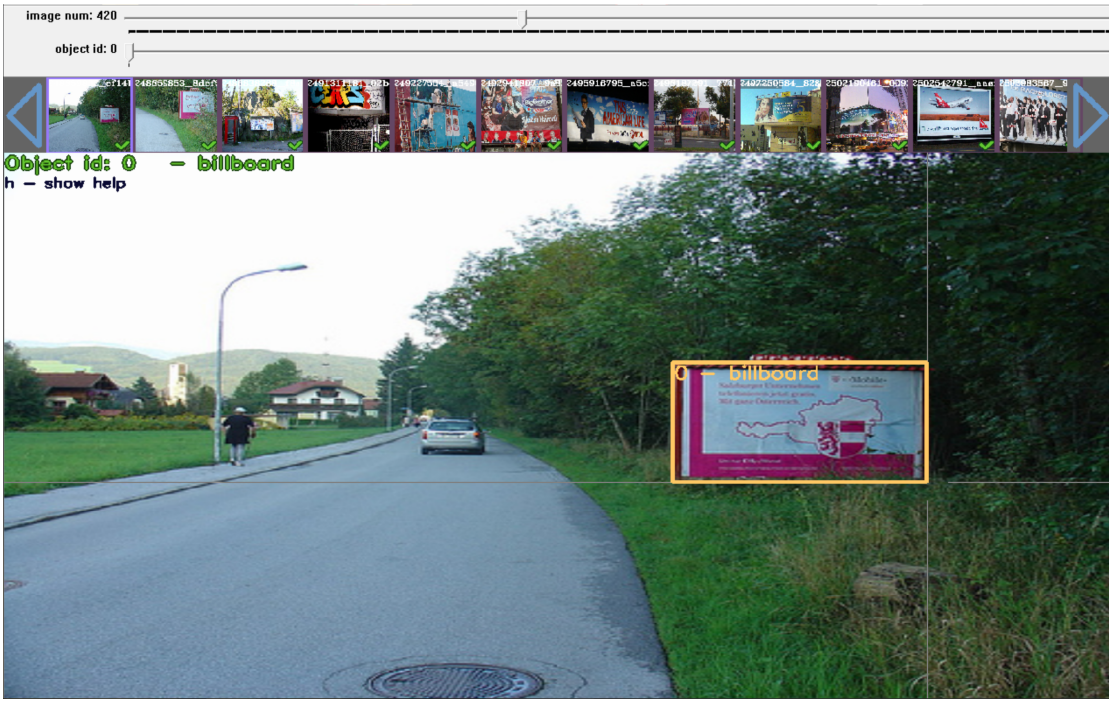


Abbildung 5.14: Darstellung des verwendeten Labeling-Werkzeuges YOLO_Mark. Die Plakatwand im Bild wurde mittels eines Begrenzungsrahmens markiert.

5.3.2 Software und Setup

Für die Verwendung von YOLOv3 muss das GitHub-Repository[14] geklont und der `Include`- bzw. der `Libary`-Ordner von OpenCV mit der Visual Studio Solution verknüpft werden. Ist die Verwendung der Grafikkarte gewünscht, so muss auch CUDA bzw. cuDNN in das Projekt mit einbezogen werden. Hier ist auf die Version von CUDA zu achten. Im Umfang dieser Arbeit wird CUDA 9.0 verwendet. Wie diese zusätzliche Software installiert werden kann, ist in Abschnitt 2.9 beschrieben. Die genaue Anleitung, wie dies in das Projekt inkludiert werden kann, findet sich in der `ReadMe`-Datei im Abschnitt *How to compile on Windows (legacy way)* des Projektes. Sind diese Schritte erledigt, so kann das Projekt erstellt und YOLOv3 unter Windows verwendet werden.

5.3.3 Trainieren von YOLOv3 mittels Transfer Learning

Um YOLOv3 mit einem eigenen Datensatz trainieren zu können, muss eine neue, angepasste Konfigurationsdatei angelegt werden. Deren Inhalt kann zum Großteil von der Datei `yolov3.cfg` übernommen werden. Was genau zu ändern ist, kann in der `ReadMe`-Datei des Projektes im Abschnitt *How to train (to detect your custom objects)* nachgelesen werden. Im Umfang dieser Arbeit wurde YOLOv3 nur mit den Standardeinstellungen trainiert. Die Lernrate beträgt 0.001. Die zugrundeliegende Netzwerkarchitektur ist Darknet-53. Die heruntergeladenen Initialgewichtungen[15] sind im selben Ordner wie die Datei `darknet.exe` abzulegen. Weiters muss der Ordner `build/darknet/x64/data` durch den im Labeling-Prozess erstellten Ordner ersetzt werden. Dieser sollte eine Ordnerstruktur wie in Abbildung 5.13 (c) aufweisen. Die genaue Struktur der wichtigsten Dateien und Ordner vor dem Training ist in Abbildung 5.16 (a) zu sehen. Der Trainingsvorgang kann inklusive der Berechnung des mAP Wertes durchgeführt und über die Kommandozeile gestartet werden:

```
darknet.exe detector train data/obj.data yolo-obj.cfg darknet53.conv.74 -map
```

Hier ist `yolo-obj.cfg` die erstellte Konfigurationsdatei und `darknet53.conv.74` sind die heruntergeladenen Initialgewichtungen. Pro Klasse sollten 2000 Iterationen durchgeführt werden. Während des Vorganges, der je nach System mehrere Stunden andauern kann, wird dynamisch ein Graph erstellt, der den Optimierungsfehler und die Durchschnittsgenauigkeit zeigt. Dieser Graph ist in Abbildung 5.15 dargestellt. Ist das Training abgeschlossen, wurde eine Datei mit den neuen, finalen Gewichtungen erstellt. Dies ist in Abbildung 5.16 (b) zu sehen. Beim Trainingsvorgang wurde ein Optimierungsfehler von 0.4042 erreicht. Für die Berechnung der Durchschnittsgenauigkeit anhand des Validierungs-Datensets wurde ein IoU-Schwellenwert von 0.5 gewählt. Diese Evaluierung lässt sich mit dem Kommandozeilenbefehl

```
darknet.exe detector map data/obj.data yolo-obj.cfg backup\yolo-obj_final.weights
  -thresh 0.25
```

berechnen. Die Ergebnisse für unterschiedliche Schwellenwerte des Konfidenzwertes sind in Tabelle 5.3 angeführt. *TP* steht hier für korrekte Vorhersagen, *FP* für falsche Vorhersagen und *FN* für nicht erkannte Werbungen. Der *Precision*-Wert gib an, wie viel

[14] https://github.com/AlexeyAB/darknet
[15] https://pjreddie.com/media/files/darknet53.conv.74

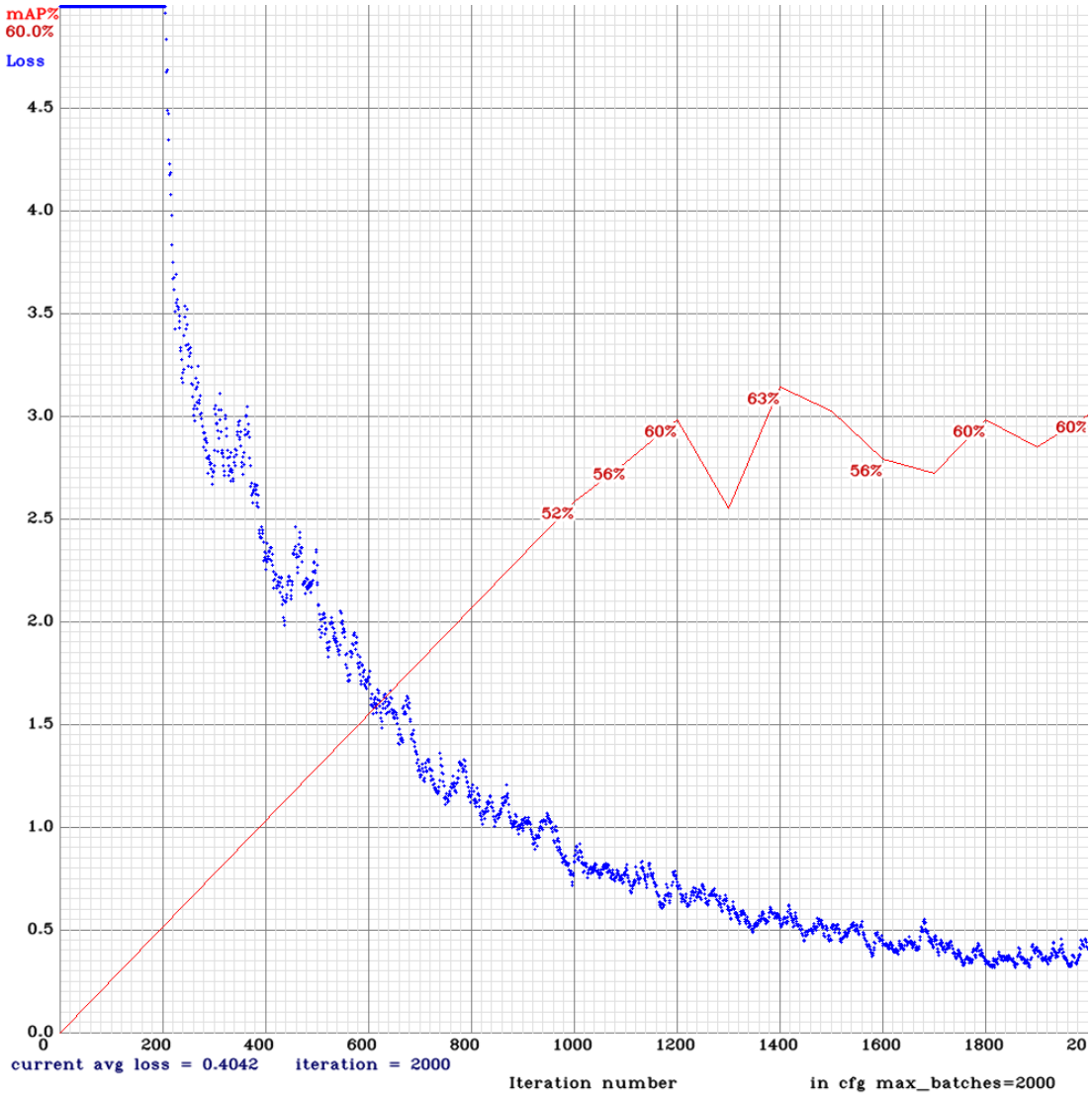
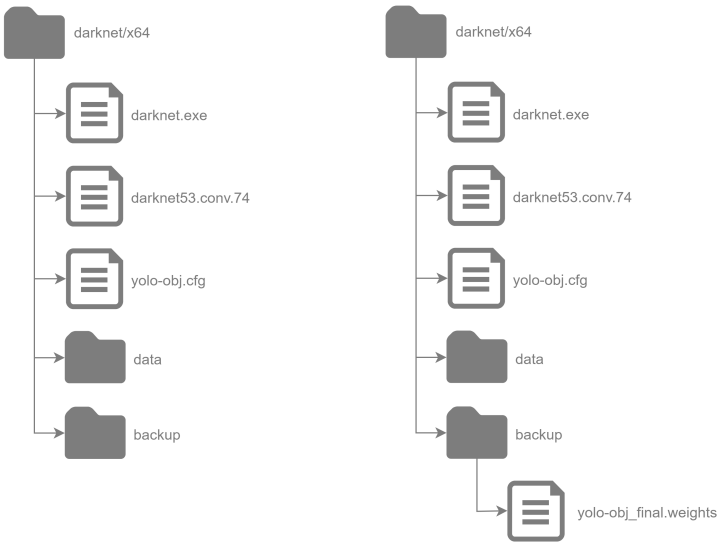


Abbildung 5.15: Aufzeichnung der Trainingserfolge bei YOLOv3. In Blau ist der Fehler während des Lernprozesses zu sehen. In Rot ist der mAP-Wert dargestellt, der mit einem IoU-Schwellenwert von 0.5 und einem Konfidenzwert von 0.25 berechnet wurde.

Prozent der getätigten Vorhersagen korrekt sind. Der *Recall*-Wert gibt Aufschluss, wie viele Grundwahrheiten identifiziert werden konnten. Grundsätzlich lässt sich hier kein wirklich bestes Ergebnis bestimmen, da eine Verbesserung in einem Bereich eine Verschlechterung in einem anderen zur Folge hat. Aus diesem Grund wurde im weiteren Verlauf der Arbeit mit dem Schwellenwert von 0.25, der gute Durchschnittswerte liefert, gearbeitet. Eine genauere Interpretation ist in Abschnitt 7.2.1 zu finden.



(a) (b)

Abbildung 5.16: Darstellung der wichtigen Ordner- und Dateihierarchie. Ordnerstruktur vor dem Training (a). Ordnerstruktur nach dem Training (b).

Tabelle 5.3: Evaluierungsergebnisse für unterschiedliche Schwellenwerte des Konfidenzwertes. *TP* steht hier für korrekte Vorhersagen, *FP* für falsche Vorhersagen und *FN* für nicht erkannte Werbungen. Der Precision-Wert gib an, wie viel Prozent der Vorhersagen korrekt sind. Der Recall-Wert lässt den Rückschluss zu, wie viele Grundwahrheiten identifiziert werden konnten. Die Ergebnisse mit dem im weiteren Verlauf verwendeten Schwellenwert sind hervorgehoben.

Schwellwert	Vorhersagen	*TP*	*FP*	*FN*	Precision	Recall	durchschn. IoU	mAP@0.50
0.02	426	272	154	169	0.64	0.62	49.97	60.01
0.05	324	255	69	186	0.79	0.58	62.18	60.01
0.10	291	241	50	200	0.83	0.55	66.00	60.01
0.15	267	231	36	210	0.87	0.52	69.38	60.01
0.20	254	223	31	218	0.88	0.51	70.63	60.01
0.25	**240**	**215**	**25**	**226**	**0.90**	**0.49**	**72.32**	**60.01**
0.30	227	204	23	237	0.90	0.46	72.97	60.01
0.35	217	198	19	243	0.91	0.45	74.16	60.01
0.40	212	194	18	247	0.92	0.44	74.59	60.01

5.3.4 Analyse von Bildern und Videos mit YOLOv3

Ist YOLOv3 auf den eigenen Datensatz trainiert, so können Bilder und Videos über die Kommandozeile analysiert werden. Die notwendige Ordnerstruktur vor dem Analyseprozess ist in Abbildung 5.17 (a) zu sehen. Das zu analysierende Bild ist hier als `test.jpg` und das zu analysierende Video als `test.mp4` benannt. Der Befehl zur Analyse eines

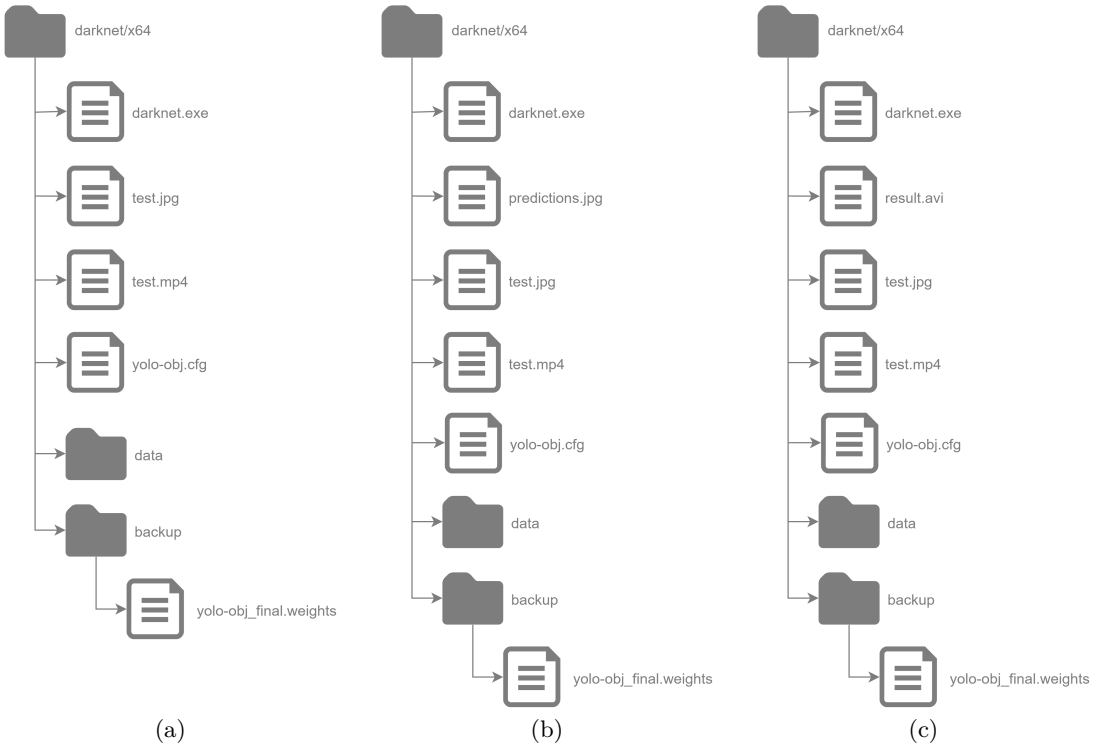


Abbildung 5.17: Darstellung der wichtigen Ordner- und Dateihierarchie. Ordnerstruktur vor der Analyse eines Bildes oder eines Videos (a). Nach Analyse eines Bildes (b). Nach Analyse eines Videos (c).

Bildes ist:

```
darknet.exe detector test data/obj.data yolo-obj.cfg backup/yolo-obj_final.weights
   -ext_output test.jpg
```

Dabei wird automatisch eine Datei namens `predictions.jpg` erstellt. Das zeigt die Abbildung 5.17 (b). Bei einer erneuten Analyse wird diese Datei automatisch überschrieben. Zum Analysieren eines Videos muss der Befehl

```
darknet.exe detector demo data/obj.data yolo-obj.cfg backup/yolo-obj_final.weights
   -ext_output test.mp4 -out_filename result.avi
```

verwendet werden. Hier ist der Name des Ergebnisvideos manuell zu vergeben. Wie die Ordnerstruktur nach der Analyse eines Videos aussieht, zeigt Abbildung 5.17 (c).

Kapitel 6

Ergebnisse

In diesem Kapitel werden die Ergebnisse von Mask R-CNN und YOLOv3 dargestellt. Die erzielten Resultate werden zum einen anhand von Testbildern und zum anderen mit der Analyse von Dashcam-Videos gezeigt. Bildbereiche, die als Werbung erkannt wurden, werden hier mit dem Begriff *Billboard* markiert. Er wird als Überbegriff für jegliche Art von Werbung verwendet, nicht nur für Plakatwände.

6.1 Ergebnisse mit Mask R-CNN

Für die Analyse der Mask R-CNN Methode wurde für das Trainieren des Netzwerkes eine Lernrate von 0.001 für einen Trainingszeitraum von 4 Epochen gewählt. Als CNN wurde ResNet101 mit den Initialgewichtungen des COCO-Datensets verwendet. Diese Einstellungen lieferten die besten Ergebnisse unter allen getesteten Kombinationen (s. Abschnitt 5.2.3). Der Trainings-Datensatz hatte einen Umfang von 727 Bildern. Aufgrund guter Ergebnisse wurde auf Data Augmentation verzichtet. Als Schwellenwert für positive Erkennung wurde bei der Bildanalyse ein Wert von 0.8 verwendet, da dieser die besten Ergebnisse (s. Abschnitt 4.4) bei der Analyse von Dashcam-Videos lieferte.

6.1.1 Ergebnisse anhand von Testbildern

Die erzielten Resultate der Testbilder wurden in vier verschiedene Kategorien eingeteilt. Diese Kategorien sind:

- gute Ergebnisse,
- Ergebnisse, bei denen Werbung nicht erkannt wurde,
- Ergebnisse, bei denen Bildbereiche fälschlicherweise als Werbung identifiziert wurden und
- mehrfach markierte Werbungen.

Bei der Einteilung der Testbilder in eine der vier Kategorien wurden kleinere Analysefehler nicht berücksichtigt, da die Zuteilung ansonsten nicht immer eindeutig möglich gewesen wäre. In den folgenden Bildern sind die Testbilder jeweils in drei Spalten dargestellt. Die linke Spalte zeigt das Ausgangsbild, das analysiert wird. Die mittlere Spalte zeigt die erkannten Werbungen und gibt Informationen, mit welcher Wahrscheinlichkeit

das identifizierte Objekt als solches klassifiziert wurde. In der rechten Spalte wird zusätzlich die Segmentierungsmaske dargestellt. Sie gibt auf Pixelebene an, welche Bereiche des Bildes als Werbung erkannt wurden. In Abbildung 6.1 ist eine Auswahl der Testbilder zu sehen, welche fehlerlos kategorisiert wurden. Abbildung 6.2 zeigt Testbilder, in denen Werbung fälschlicherweise nicht als solche erkannt wurde. Abbildung 6.3 zeigt Bilder, bei denen Mask R-CNN andere Objekte wie zum Beispiel Häuser oder Personen fälschlicherweise als Werbung kategorisiert hat. In Abbildung 6.4 werden Ergebnisse gezeigt, bei denen Werbung mehrfach als solche markiert wurde.

6.1.2 Ergebnisse bei der Analyse eines Dashcam-Videos

Für die Darstellung der Resultate anhand eines Dashcam-Videos wurde ein YouTube-Video [68], das eine Autofahrt in West Hollywood zeigt, gewählt, da dieses eine Vielzahl an Werbungen enthält und sich somit gut für die Analyse eignet. Bei der Erkennung von Werbung im Straßenverkehr werden gute Ergebnisse erzielt. Jedoch tritt auch hier der Fall auf, dass manche Werbungen nicht als solche erkannt wurden oder dass andere Objekte wie zum Beispiel Häuser oder Autos fälschlicherweise als solche kategorisiert werden. Die Abbildungen 6.5 bis 6.8 zeigen Videoschnappschüsse des analysierten Videos. Hier wurde auf Grund der Übersichtlichkeit auf die Segmentierungsmasken verzichtet.

6.2 Ergebnisse mit YOLOv3

Beim Training von YOLOv3 wurde ebenfalls eine Lernrate von 0.001 über 2000 Iterationen verwendet. Der Trainings-Datensatz umfasst 844 Bilder. Für die positive Kategorisierung von Werbung wurde ein Schwellenwert von 0.25 gewählt. Dieser Wert ist um vieles niedriger als bei Mask R-CNN. Dennoch wurden beinahe keine Objekte fälschlicherweise als Werbung kategorisiert.

6.2.1 Ergebnisse anhand von Testbildern

Die erzielten Resultate der Testbilder wurden in zwei verschiedene Kategorien eingeteilt. Diese Kategorien sind:

- gute Ergebnisse und
- Ergebnisse, bei denen Werbung nicht erkannt wurde.

In den folgenden Bildern sind die Testbilder jeweils in zwei Spalten dargestellt. Die linke Spalte zeigt das Ausgangsbild, das analysiert wird. Die rechte Spalte zeigt die erkannten Werbungen. In Abbildung 6.9 ist eine Auswahl der Testbilder zu sehen, die fehlerlos kategorisiert wurden. Abbildung 6.10 zeigt Testbilder, in denen Werbung fälschlicherweise nicht als solche erkannt wurde.

6.2.2 Ergebnisse bei der Analyse eines Dashcam-Videos

Für die Darstellung der Resultate anhand eines Dashcam-Videos wurde wiederum dasselbe YouTube-Video [68], das eine Autofahrt in West Hollywood zeigt, verwendet. Bei der Erkennung von Werbung im Straßenverkehr werden auch mit dieser Methode gute Ergebnisse erzielt. Jedoch tritt vermehrt der Fall auf, dass Werbung nicht als solche

Abbildung 6.1: Ausgewählte Testbilder, die gute Ergebnisse durch Analyse mit Mask R-CNN zeigen. Die linke Spalte beinhaltet das Ausgangsbild, das analysiert wird. Die mittlere Spalte zeigt die erkannten Werbungen und gibt Informationen, mit welcher Wahrscheinlichkeit das identifizierte Objekt als solches klassifiziert wurde. In der rechten Spalte wird zusätzlich die Segmentierungsmaske dargestellt.

Abbildung 6.2: Ausgewählte Testbilder, die Ergebnisse durch Analyse mit Mask R-CNN zeigen, bei denen Werbung nicht als solche erkannt wurde. Die linke Spalte stellt das Ausgangsbild dar, das analysiert wird. Die mittlere Spalte zeigt die erkannten Werbungen und gibt Informationen, mit welcher Wahrscheinlichkeit das identifizierte Objekt als solches klassifiziert wurde. In der rechten Spalte wird zusätzlich die Segmentierungsmaske dargestellt.

erkannt wird. Dies betrifft vor allem kleine Werbungen bzw. Werbungen, die weiter entfernt sind. Abbildungen 6.11 bis 6.14 zeigen dieselben Videoschnappschüsse, die hier jedoch mit YOLOv3 analysiert wurden.

Abbildung 6.3: Ausgewählte Testbilder, die Ergebnisse durch Analyse mit Mask R-CNN zeigen, bei denen andere Objekte fälschlicherweise als Werbung identifiziert wurden. Die linke Spalte führt das Ausgangsbild an, das analysiert wird. Die mittlere Spalte zeigt die erkannten Werbungen und gibt Informationen, mit welcher Wahrscheinlichkeit das identifizierte Objekt als solches klassifiziert wurde. In der rechten Spalte wird zusätzlich die Segmentierungsmaske dargestellt.

Abbildung 6.4: Ausgewählte Testbilder, die Ergebnisse durch Analyse mit Mask R-CNN zeigen, bei denen Werbung mehrfach als solche markiert ist. Die linke Spalte zeigt das Ausgangsbild, das analysiert wird. Die mittlere Spalte beinhaltet die erkannten Werbungen und gibt Informationen, mit welcher Wahrscheinlichkeit das identifizierte Objekt als solches klassifiziert wurde. In der rechten Spalte wird zusätzlich die Segmentierungsmaske dargestellt.

Abbildung 6.5: Videoschnappschuss des mit Mask R-CNN analysierten YouTube-Videos [68].

Abbildung 6.6: Videoschnappschuss des mit Mask R-CNN analysierten YouTube-Videos [68].

Abbildung 6.7: Videoschnappschuss des mit Mask R-CNN analysierten YouTube-Videos [68].

Abbildung 6.8: Videoschnappschuss des mit Mask R-CNN analysierten YouTube-Videos [68].

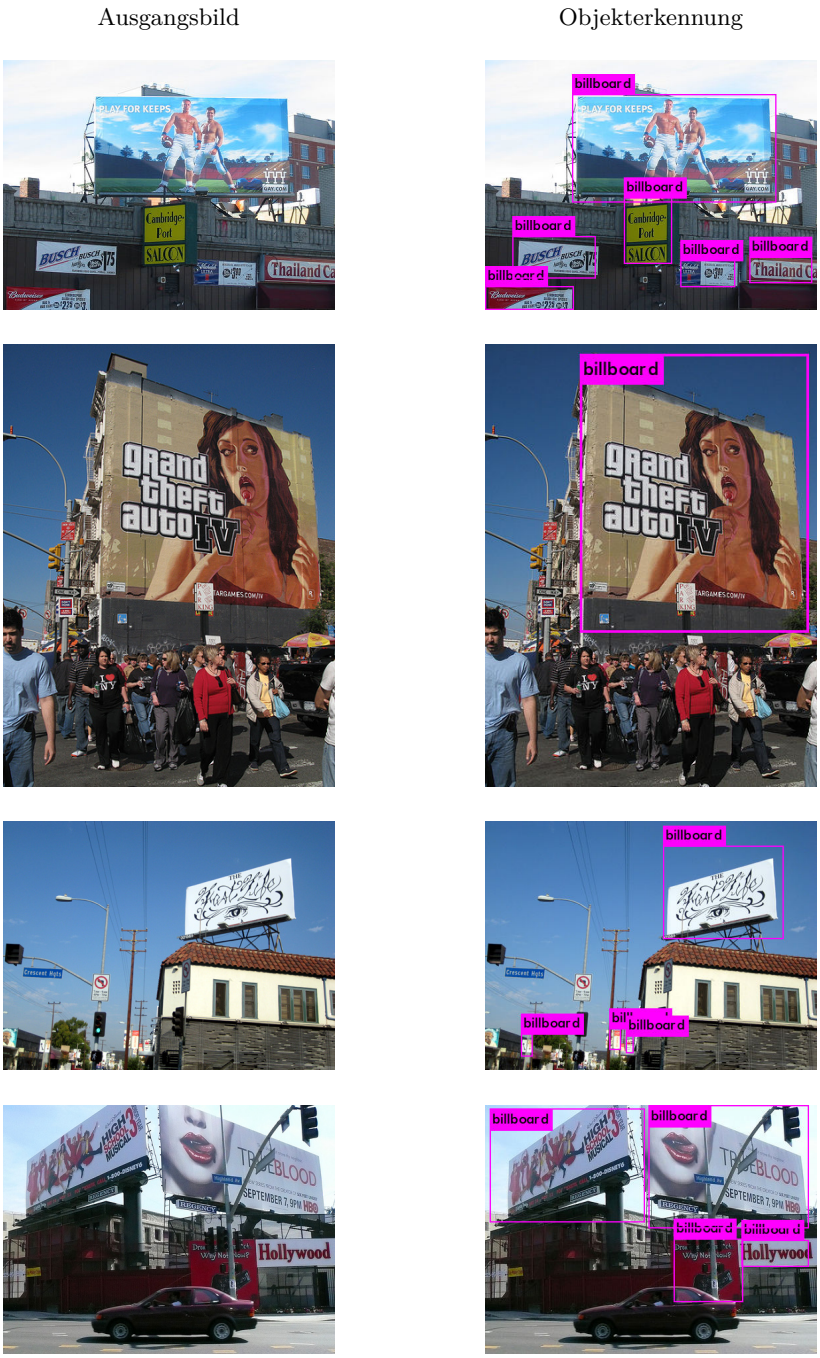


Abbildung 6.9: Ausgewählte Testbilder, die gute Ergebnisse durch Analyse mit YOLOv3 zeigen. Die linke Spalte führt das Ausgangsbild an, das analysiert wird. Die rechte Spalte zeigt die erkannten Werbungen.

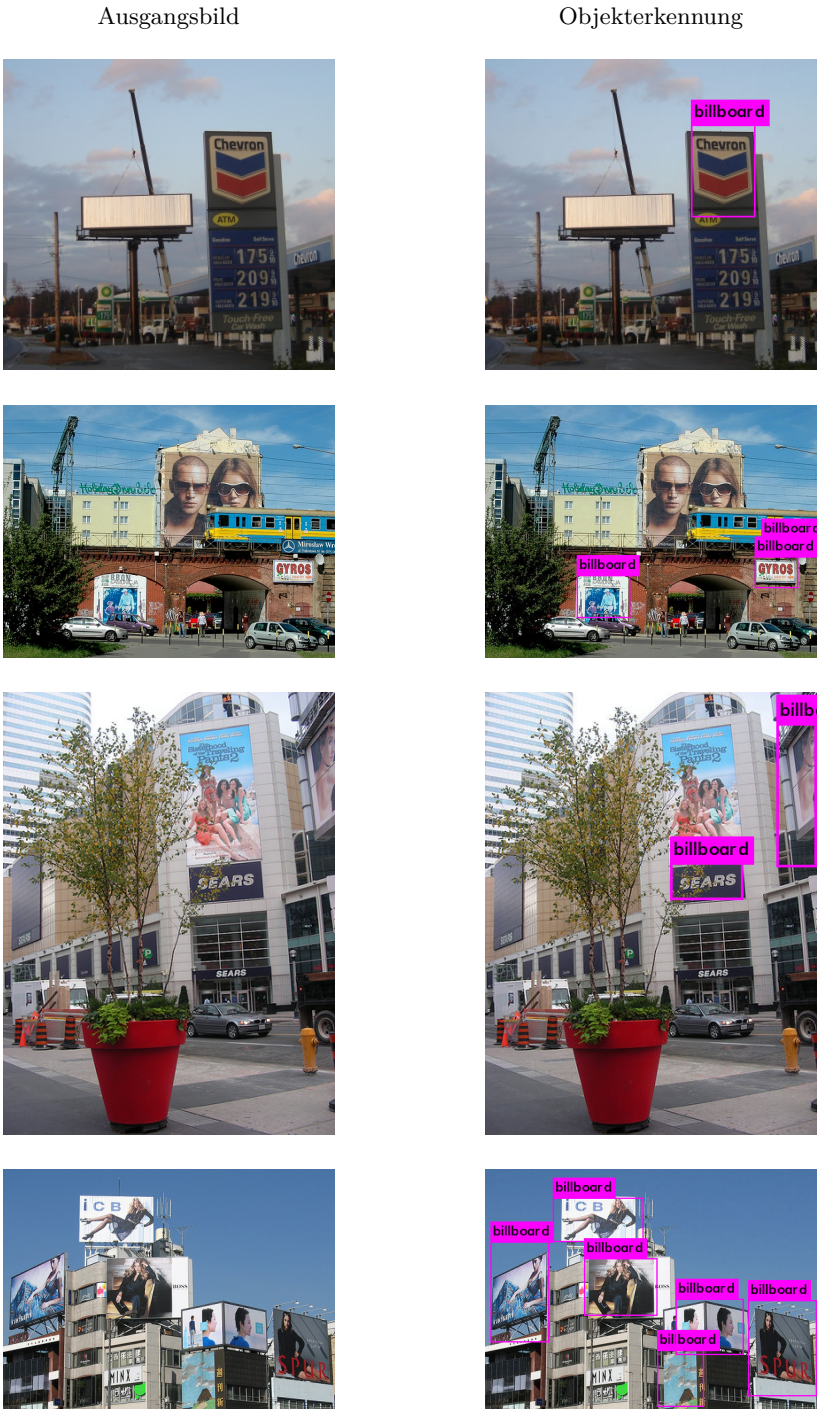


Abbildung 6.10: Ausgewählte Testbilder mit Ergebnissen durch Analyse mit YOLOv3, bei denen Werbung nicht erkannt wurde. Die linke Spalte beinhaltet das Ausgangsbild, das analysiert wird. Die rechte Spalte zeigt die erkannten Werbungen.

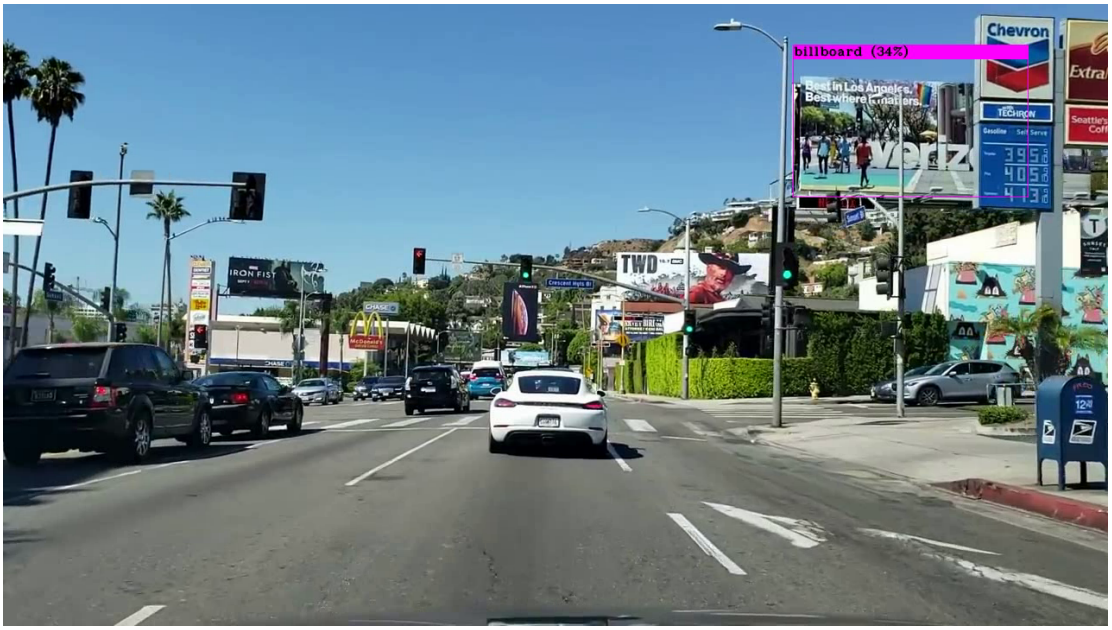


Abbildung 6.11: Videoschnappschuss des mit YOLOv3 analysierten YouTube-Videos [68].

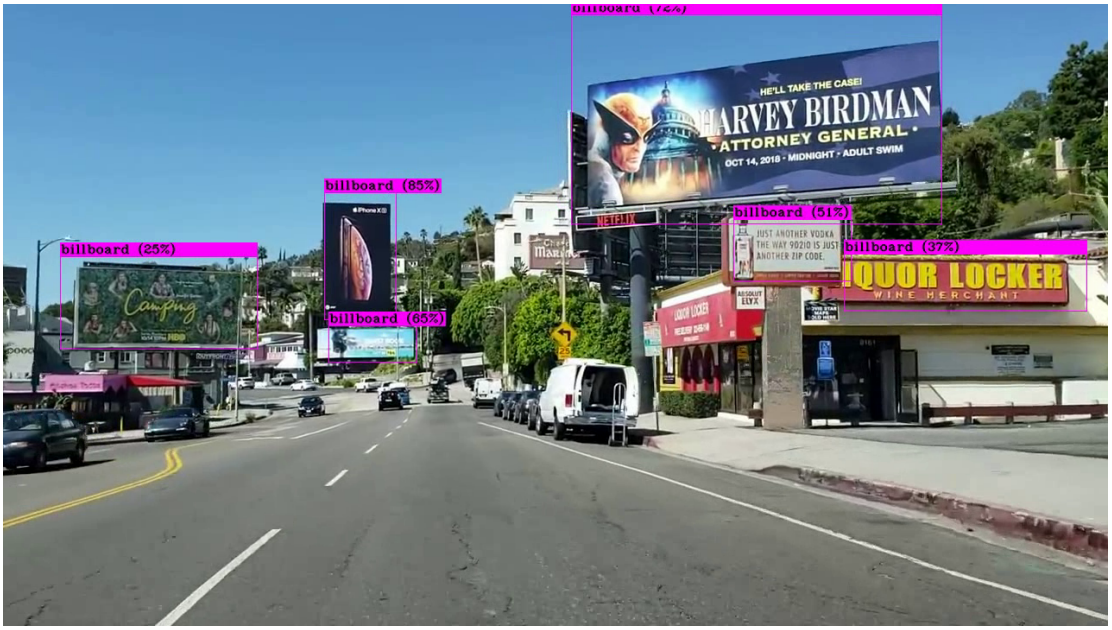


Abbildung 6.12: Videoschnappschuss des mit YOLOv3 analysierten YouTube-Videos [68].

Abbildung 6.13: Videoschnappschuss des mit YOLOv3 analysierten YouTube-Videos [68].

Abbildung 6.14: Videoschnappschuss des mit YOLOv3 analysierten YouTube-Videos [68].

Kapitel 7

Interpretation und Zusammenfassung

Für das Erkennen von Werbungen in einem Bild oder Video wurden zwei Deep Learning Methoden praktisch umgesetzt. Diese beiden Methoden sind Mask R-CNN (s. Abschnitt 5.2) und YOLOv3 (s. Abschnitt 5.3). Beide können sowohl mit der CPU als auch mit der GPU ausgeführt werden. Um die GPU nutzen zu können, muss zusätzliche Software installiert werden (s. Abschnitt 2.9). Die Nutzung der GPU hat jedoch den großen Vorteil, dass sowohl Training als auch Analyse von Bildern um ein Vielfaches schneller sind. Die für die Trainingsvorgänge der beiden Methoden benötigten Bilder wurden aus dem Internet bezogen und manuell gelabelt. Im Umfang dieser Arbeit wurden Mask R-CNN und YOLOv3 nur auf die Kategorie *Werbung* trainiert. Im Folgenden wird auf den Trainingsprozess, die Bildanalyse und auf die Grenzen und potentielle Verbesserungen der Methoden eingegangen.

7.1 Interpretation von Mask R-CNN

Mask R-CNN ist eine Objekterkennungsmethode, die den Fokus auf gute Analyseresultate legt. Darunter leidet jedoch die Geschwindigkeit, mit der die Bilder analysiert werden. Im Verlauf dieser Arbeit wurde Mask R-CNN mit Keras und TensorFlow als Backend in der Programmiersprache Python umgesetzt.

7.1.1 Trainingsprozess

Für Mask R-CNN müssen die Bildbereiche mit Werbung für den Trainingsprozess exakt markiert werden. Das bedeutet, dass die Umrisse dieser Bereiche mit Polygonen gelabelt werden müssen. Das ist sehr aufwendig und benötigt eine Menge Zeit. Insgesamt konnten 960 Bilder verwendet werden. Davon wurden 727 Bilder für einen Trainings-Datensatz und 233 Bilder für einen Validierungs-Datensatz eingesetzt. Das Labeling-Verfahren dauerte im Durchschnitt 24 Sekunden pro markierten Bildbereich. In Summe sind das etwas über sieben Stunden Arbeit. Auf Data Augmentation (s. Abschnitt 2.2.6) wurde im Umfang dieser Arbeit verzichtet. Das Modell wurde mehrfach mit verschiedenen Kombinationen aus Hyperparameter-Einstellungen, unterschiedlichen Netzwerkarchitekturen und Initialgewichtungen trainiert, um Vergleichswerte zu erhalten. Bei der Netzwerkarchitektur wurde ResNet50 bzw. ResNet101 getestet. Bei den Initialgewichtungen wurden Gewichtungen verwendet, die zum einen mit dem COCO-Datenset und

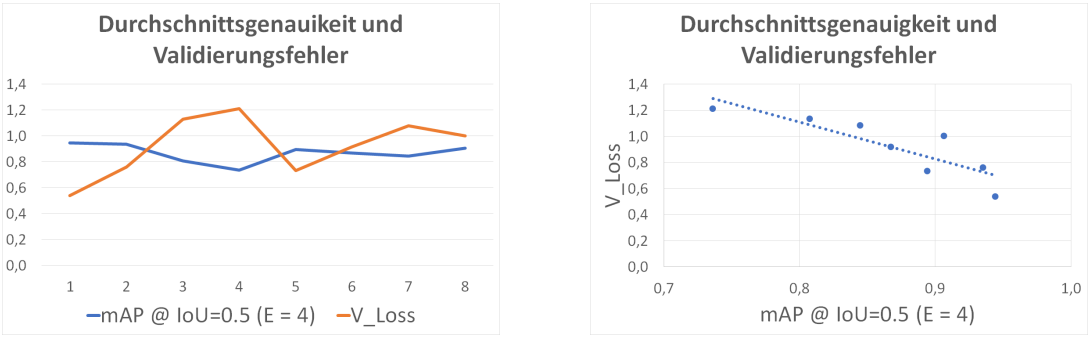


Abbildung 7.1: Im linken Diagramm sind der Validierungsfehler und die Durchschnittsgenauigkeit aller Testdurchläufe nach der vierten Epoche zu sehen. Im rechten Bild wird gezeigt, wie sich die beiden Werte zueinander verhalten.

zum anderen mit dem ImageNet-Datenset antrainiert wurden. Für die Lernrate wurden die Werte 0.001 und 0.005 gewählt. Diese Einstellungen können in der Netzwerkkonfiguration definiert werden. Beim Trainingsvorgang ist bereits bei manchen Kombinationen ab der fünften Epoche Overfitting (s. Abschnitt 2.3.1) aufgetreten, was auf den geringen Umfang des Trainings-Datensets zurückzuführen ist. Je höher die Lernrate war, desto dramatischer hat sich dieser Effekt gezeigt. Der Trainingsprozess mit ResNet101 als Netzwerkarchitektur dauerte im Durchschnitt 68 Minuten pro Epoche. Mit ResNet50 waren es durchschnittlich 66 Minuten. Wie sich der Optimierungsfehler während der einzelnen Trainingsvorgänge entwickelt hat, ist in Tabelle 5.1 zu sehen. Diese Werte können sich selbst bei gleichen Einstellungen von Trainingsprozess zu Trainingsprozess verändern, da die Reihenfolge der Bilder während des Prozesses zufällig erfolgt. Die Auswertung der Durchschnittsgenauigkeit ist in Tabelle 5.2 zu sehen. Diese zeigt, dass das beste Ergebnis mit der Kombination von ResNet101, Initialgewichtungen vom COCO-Datenset und einer Lernrate von 0.001 erzielt wurden. Der Zusammenhang zwischen dem Validierungsfehler und der Durchschnittsgenauigkeit ist in Abbildung 7.1 dargestellt. Der große Unterschied zu den Ergebnissen mit Initialgewichtungen vom ImageNet-Datenset bei gleicher Netzwerkarchitektur ist darauf zurückzuführen, dass diese Gewichtungen für eine ResNet50-Architektur bestimmt sind. Das bedeutet, dass die Layer, die bei ResNet101 zusätzlich vorhanden sind, mit zufälligen Werten initialisiert werden. Die Resultate mit ResNet50 liegen im Bereich dazwischen. Es lässt sich keine eindeutige Aussage treffen, welche Initialgewichtungen bei passender Netzwerkarchitektur beziehungsweise welche Lernrate die bessere Wahl sind.

7.1.2 Bildanalyse

Die Bild- und Videoanalyse wurde mit den Konfigurationseinstellungen, die die besten Ergebnisse erzielten, durchgeführt. Mit diesen Einstellungen war eine Durchschnittsgenauigkeit von 94.5% möglich. Dieser Wert ist im Vergleich zu anderen Arbeiten durchaus akzeptabel. In Abschnitt 6.1 sind die erzielten Ergebnisse zu sehen. In den Abbildungen 7.3 bis 7.6 sind Featuremaps von unterschiedlichen Layern zu erkennen. Abbildung 7.3 zeigt Featuremaps eines frühen Layers im Netzwerk. In diesem Stadium des

Abbildung 7.2: Ausgangsbild, anhand dessen die Featuremaps unterschiedlicher Layer gezeigt werden.

Netzwerkes werden allgemeine Features, wie zum Beispiel Kanten, erkannt. Hier ist auch die Auflösung mit 246×256 Pixel noch relativ groß. In diesem Layer sind 256 Featuremaps vorhanden. Die Features werden bis hin zu den in Abbildung 7.6 gezeigten Bildern immer spezieller. In diesem Bereich des Netzwerkes haben die Featuremaps noch eine Auflösung von 32×32. Ein Layer enthält 2048 Featuremaps. Bei diesen Maps kann das Ausgangsbild, das in Abbildung 7.2 zu sehen ist, nicht mehr erkannt werden. Anhand der Featuremaps kann auch keine Aussage darüber getroffen werden, welche Anhaltspunkte vom Netzwerk genau als Werbung definiert werden.

Ein weiterer wichtiger Analysepunkt sind die Position und die Größe der Regionsvorschläge, die vom RPN erhalten werden. In Abbildung 7.7 ist die Position der linken unteren als auch der rechten oberen Ecke der 1000 erhaltenen Regionen zu sehen. In Abbildung 7.8 ist die Verteilung der zentralen Position der Regionen demonstriert. Abbildung 7.9 zeigt die Größenverteilung dieser Regionen. Um die Verteilung auch in absoluten Zahlen zu erkennen, sind sie in Abbildung 7.10 als Histogramme dargestellt. Die Verteilungen zeigen, dass sich die meisten Regionsvorschläge in der Bildmitte des oberen Bereiches befinden. Der durchschnittliche Wert für die X-Position ist 0.487. Für die Y-Position beträgt dieser Wert 0.569. Weiters sind die meisten Regionen relativ klein, wobei die Breite meist größer als die Höhe ist. Für die Breite liegt der Durchschnittswert bei 0.212, für die Höhe liegt der Wert bei 0.125. Diese Werte sind relativ zur Bildgröße zu sehen. Wie die Regionen anhand des Bildes aussehen, ist in Abschnitt 5.2.3 genauer erklärt.

7.1.3 Detektionsgrenzen der Objekterkennungsmethode

Ein kritischer Umstand bei nur einer zu kategorisierenden Objektklasse ist, dass jede Interessensregion, die vom Netzwerk nicht als Hintergrund identifiziert wurde, als Werbung kategorisiert wird. Das hat zur Folge, dass eine erhöhte Anzahl von Bildbereichen, die fälschlicherweise als Werbung kategorisiert werden, auftreten. Dem kann nur mit dem Schwellenwert für positive Identifizierungen entgegengewirkt werden. In dieser Arbeit hat sich dafür ein Wert von 0.8 als sinnvoll erwiesen. Der Evaluierung des Trainingsprozesses kann entnommen werden, dass Bildbereiche die fälschlicherwei-

Abbildung 7.3: Featuremaps eines Layer am Anfang des Netzwerkes.

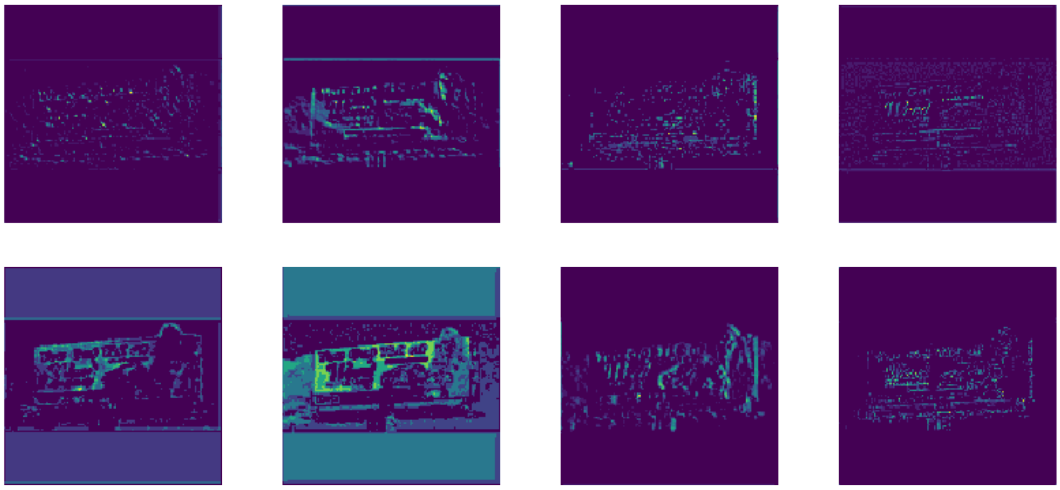


Abbildung 7.4: Featuremaps in einem der früheren Layer des Netzwerkes.

se als Werbung kategorisiert werden, häufiger auftreten als nicht erkannte Werbungen. Dies lässt sich jedoch nicht durch eine Änderung des Schwellenwertes des Konfidenzwertes ausgleichen, da der Konfidenzbereich der richtigen Kategorisierungen bereits ab 0.8 beginnt. Weiters sind auch die falsch kategorisierten Bildbereiche, die noch angezeigt werden, über den gesamten Wahrscheinlichkeitsbereich von $0.8-1.0$ verteilt. Somit würde eine Erhöhung dieses Schwellenwertes eher negative Auswirkungen haben.

Bei einem Vergleich von Testbildern vor und nach einer Rotation um 180° ist zu sehen, dass Mask R-CNN nicht völlig rotationsinvariant ist, sofern das Netzwerk nicht explizit darauf trainiert ist. Dies ist in Abbildung 7.11 zu erkennen. Weiters zeigt sich hier, dass bei den rotierten Testbildern hauptsächlich Bildbereiche im oberen Teil des

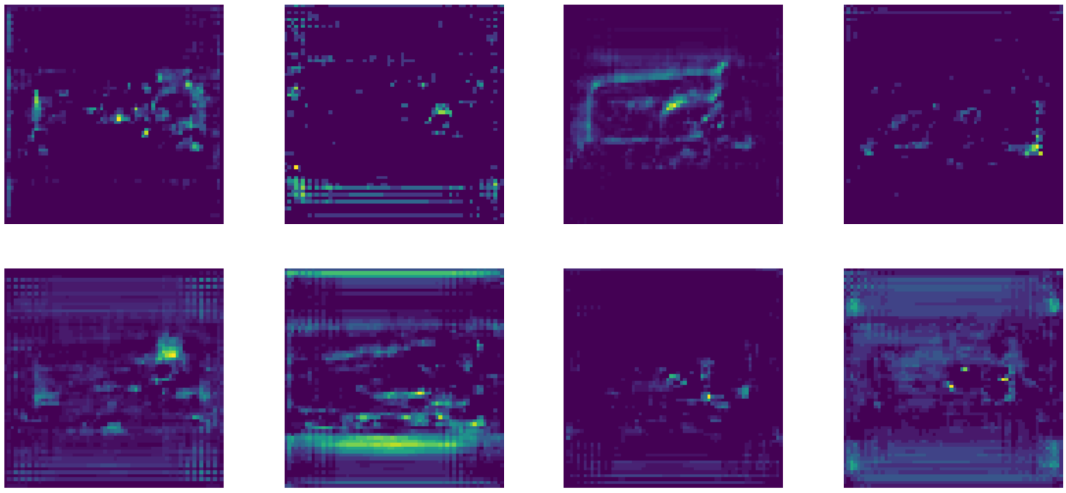


Abbildung 7.5: Featuremaps in einem der späteren Layer des Netzwerkes.

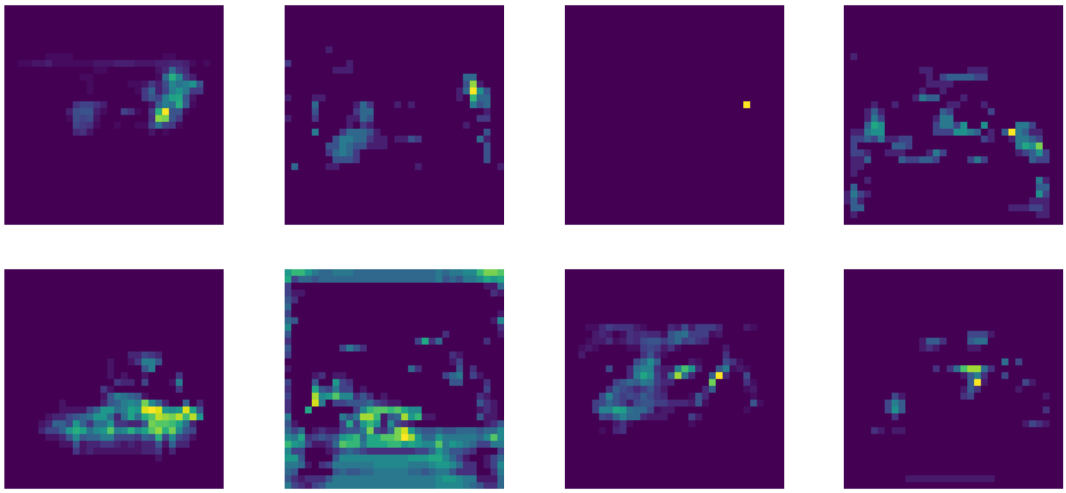


Abbildung 7.6: Featuremaps eines Layer am Ende des Netzwerkes.

Bildes fälschlicherweise als Werbung klassifiziert werden. Wie bereits in Abschnitt 7.1.2 angemerkt, lässt auch dies darauf schließen, dass das Netzwerk gelernt hat, dass sich Werbung eher im oberen Bildbereich befindet. Die Analyse eines Bildes dauert mit Mask R-CNN etwa zwei Sekunden. Somit ist diese Methode nicht echtzeitfähig.

7.1.4 Verbesserungen

Die erzielten Ergebnisse sind sehr gut, aber nicht perfekt. Durch ein paar Änderungen im Trainingsprozess können diese noch verbessert werden. So könnten die verwendeten, annotierten Bildern des Trainings-Datensatzes durch Bilder von lokalen Werbungen

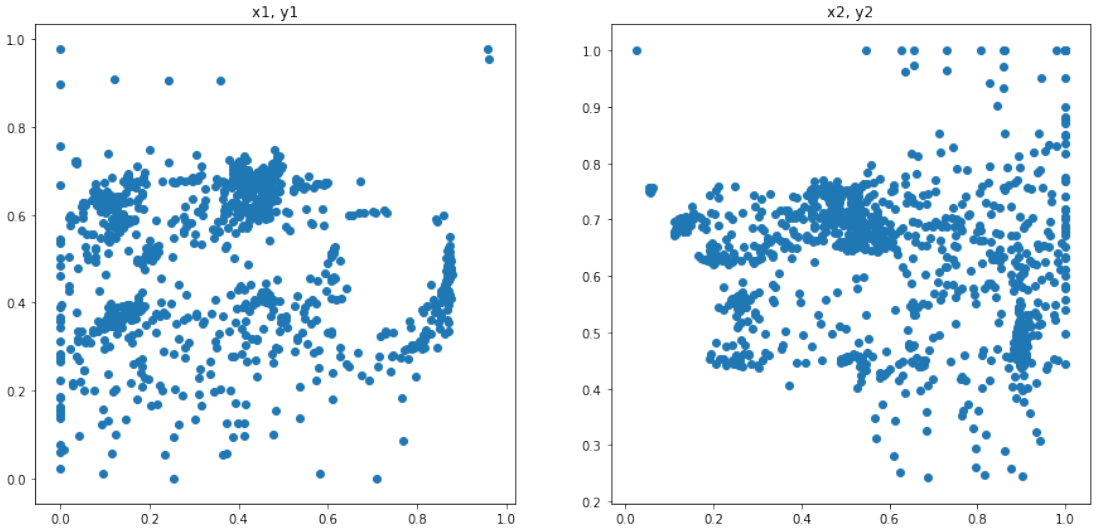


Abbildung 7.7: Im linken Diagramm sind die Verteilung der linken, unteren Positionen aller vom RPN erhaltenen Regionsvorschläge zu sehen. Rechts sind die rechten, oberen Positionen zu sehen.

am Straßenrand erweitert bzw. ersetzt werden. Das würde das Ergebnis von Dashcam-Videos vermutlich verbessern, da die aktuell verwendeten Bilder des Trainings-Datensets relativ wenig mit dem Kontext *Werbung im Straßenverkehr* zu tun haben. Da Werbung eher am Straßenrand zu finden ist, würde dies vermutlich zu einer zum seitlichen Bildrand verschobenen Positionsverteilung der Regionsvorschläge führen. Das konnte aus zeitlichen Gründen im Umfang dieser Arbeit nicht evaluiert werden. Weiters könnte durch eine Vergrößerung des Trainings-Datensatzes die Generalisierung von Werbung verbessert werden. Das kann zum Beispiel durch Data Augmentation (s. Abschnitt 2.2.6) umgesetzt werden. Somit könnten mehr Trainingsepochen ohne Auftreten von Overfitting durchgeführt werden. Diese verbesserte Generalisierung würde dazu führen, dass die Anzahl der nicht erkannten Werbungen verringert werden könnte. Um falsch positiven Ergebnissen entgegenwirken zu können, könnte das jeweilige Modell insofern angepasst werden, dass nicht nur Werbungen, sondern auch andere Objekte, die entlang von Straßen häufig vorkommen, klassifiziert werden. Im Anwendungsfall von Dashcam-Videos würden sich zusätzlich zum Beispiel Autos, Fußgänger, Zäune oder auch Bäume zur Klassifizierung anbieten, um so die Anzahl der falsch positiven Resultate zu reduzieren. Eine weitere potentielle Verbesserung lässt sich vom Optimierungsfehler während des Trainingsprozesses ableiten. Dieser setzt sich aus allen Komponenten des Netzwerkes zusammen. Analysiert man den Fehler der einzelnen Komponenten im Detail, sieht man, dass noch am meisten Verbesserungspotential bei dem Training des RPN vorhanden ist. Das könnte mit weiteren Parametereinstellungen, die diesen Bereich betreffen, verbessert werden. Im Umfang der Arbeit werden diese Komponenten jedoch nicht näher analysiert.

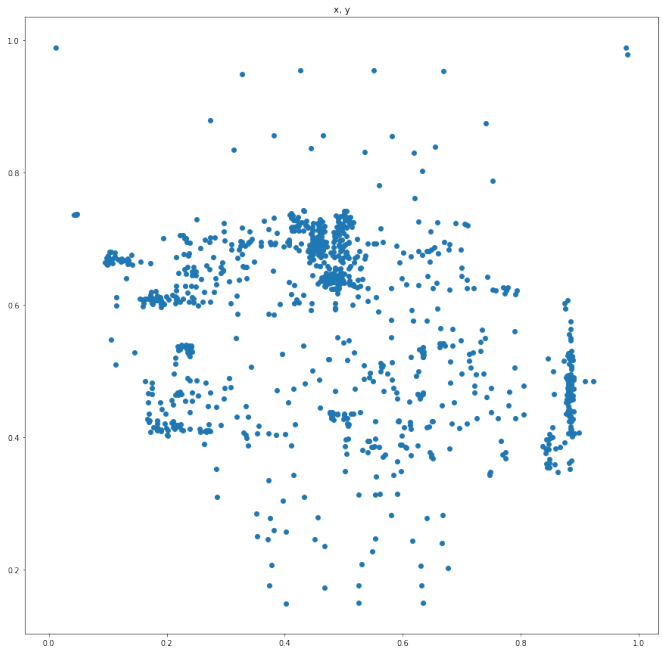


Abbildung 7.8: Grafische Darstellung der Verteilung der Mittelpunkte aller vom RPN erhaltenen Regionsvorschläge.

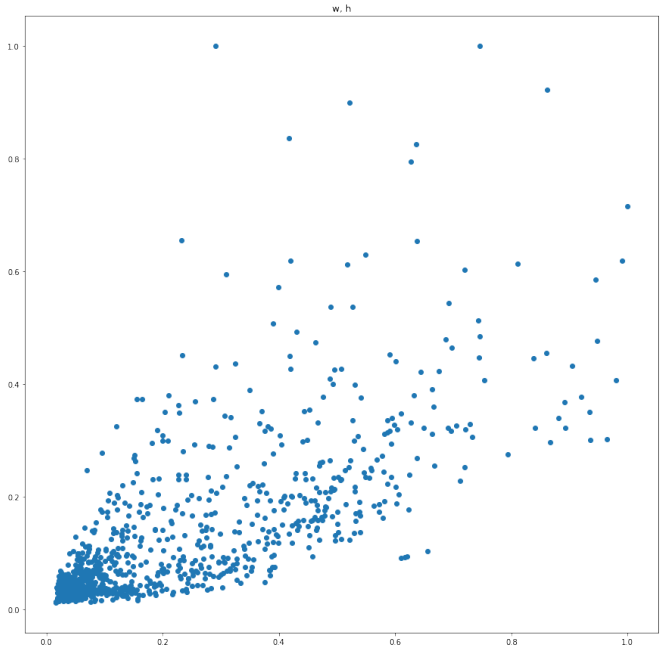


Abbildung 7.9: Grafische Darstellung der Verteilung der Größen aller vom RPN erhaltenen Regionsvorschläge.

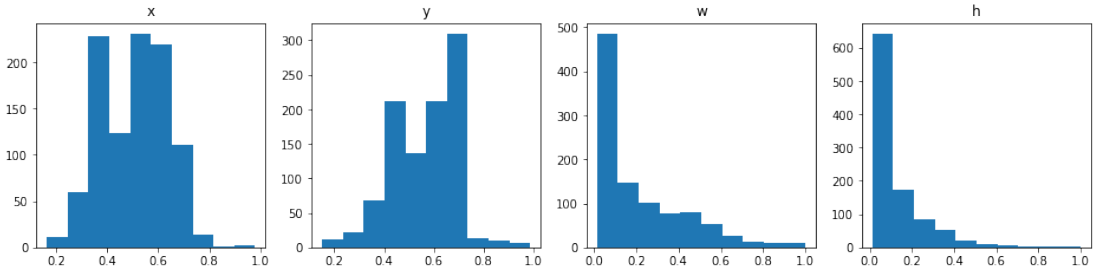


Abbildung 7.10: Histogramm-Verteilungen von Position und Größe aller vom RPN erhaltenen Regionsvorschläge. Im ersten Diagramm ist die Verteilung der horizontalen Position des Zentrums der Regionsvorschläge zu sehen. Im zweiten Diagramm ist die Verteilung der vertikalen Position gezeigt. Das dritte Diagramm zeigt die Verteilung der Breite der Regionen. Das vierte Diagramm stellt die Verteilung der Höhe der Regionen dar.

7.2 Interpretation von YOLOv3

YOLOv3 ist eine Objekterkennungsmethode, die den Fokus auf Geschwindigkeit legt, dafür werden jedoch Kompromisse bei der Vorhersagegenauigkeit eingegangen. Es wurde entschieden, für YOLO eine in sich abgeschlossene Implementierung für Windows zu verwenden, die auf der ursprünglichen Realisierung des Erfinders basiert.

7.2.1 Trainingsprozess

Für das Training von YOLOv3 müssen die Bildbereiche, die ein Objekt enthalten, mit einem Begrenzungsrahmen markiert werden. Dies spart im Vergleich zu der Markierung mit Polygonen enorm viel Zeit. Pro Label wurden nur etwa fünf Sekunden benötigt. Dies entspricht für den gesamten Datensatz, der 1050 Bilder enthält, ungefähr 90 Minuten. Die Anzahl der Bilder unterscheidet sich zu der von Mask R-RCNN, da auch jene Bilder zählen, die keine Werbung enthalten. Diese wurden bei Mask R-CNN ignoriert. Da die Ergebnisse von YOLOv3 nur als Vergleichswerte zu Mask R-CNN dienen sollen, wurde hier nur ein Trainingsvorgang mit den Standardeinstellungen durchgeführt. Die zugrundeliegende Netzwerkarchitektur ist Darknet-53. Die verwendeten Initialgewichtungen für diese Architektur sind jene, die vom Erfinder von YOLOv3 zur Verfügung gestellt werden. Die Lernrate beträgt 0.001. Das Netzwerk wurde mit einem Trainings-Datensatz, der 844 Bilder umfasst, über 2000 Iterationen trainiert, da dies die minimale empfohlene Anzahl pro Klasse ist. Der Trainingsvorgang über die gesamten 2000 Iterationen dauert ungefähr 25 Stunden. Deshalb ist die Trainingsdauer hier um einiges höher als bei Mask R-CNN. Während dieses Trainingsvorganges wird dynamisch ein Graph erstellt, der Informationen über den Optimierungsfehler und die Durchschnittsgenauigkeit angibt. Dieser ist in Abbildung 5.15 zu sehen. Der Optimierungsfehler kann jedoch nicht mit dem von Mask R-CNN verglichen werden, da sich die Komponenten, aus denen sich der Fehler zusammensetzt, komplett unterschiedlich sind.

Abbildung 7.11: Ausgewählte Testbilder, die die Ergebnisse vor und nach Rotation durch Analyse mit Mask R-CNN zeigen. Die linke Spalte stellt die Analyse des Bildes mit Segmentierungsmaske vor der Rotation des Bildes dar. Die mittlere Spalte zeigt die erkannten Werbungen nach Rotation des Bildes. In der rechten Spalte wird zusätzlich die Segmentierungsmaske dargestellt.

7.2.2 Bildanalyse

Mit YOLOv3 wurde eine Durchschnittsgenauigkeit von 60.01% erreicht. Dieser Wert liegt in einem zu erwartenden Bereich. Es wurden mehrere Schwellenwerte für den Konfidenzwert getestet, um die besten Ergebnisse zu erhalten. Die Auswertung ist in Tabelle 5.3 zu sehen. Hier lässt sich jedoch kein eindeutiges, bestes Ergebnis bestimmen, da die Verbesserung eines Wertes nur auf Kosten eines andern möglich ist. Je niedriger der Schwellenwert ist, desto mehr Vorhersagen werden getroffen. Das hat jedoch zur Folge, dass auch mehr falsche Vorhersagen entstehen. Dadurch sinkt zudem der Wert der übereinstimmenden Flächen. Ein Phänomen, das hier zu sehen ist, ist dass bei allen Einstellungen der mAP-Wert gleich ist. Das bedeutet, dass laut Definition jede Einstellung genau gleich gut ist. Wenn man sich die Vorhersagen im Einzelnen ansieht, sind diese Werte jedoch sehr unterschiedlich. Dies ist ein gutes Beispiel, dass, wie bereits in Abschnitt 4.4 erwähnt, die Bewertung eines Netzwerkes mittels mAP-Wertes nicht immer sinnvoll ist. Da hier kein eindeutig bestes Ergebnis identifiziert werden kann, wurde im weiteren Verlauf der Arbeit ein Schwellenwert von 0.25 gewählt, da dieser gute Durchschnittswerte liefert. Der Wert ist um vieles niedriger als bei Mask R-CNN, dennoch wurden andere Objekte selten fälschlicherweise als Werbung kategorisiert. Viel öfter wurden Werbungen nicht als solche erkannt, was darauf schließen lässt, dass die Generalisierung nicht gut funktioniert und noch Verbesserungspotential vorliegt. Der große Vorteil von YOLOv3 gegenüber Mask R-CNN ist, dass die Analyse nahezu in Echtzeit (ca. 11 fps) bereits mit einem Laptop funktioniert.

7.2.3 Detektionsgrenzen der Objekterkennungsmethode

YOLO hat vor allem das Problem, dass Werbungen oft nicht als solche identifiziert werden können. Dies spiegelte sich großteils bei kleinen bzw. weit entfernten Werbungen wider. Das ist jedoch eine bekannte Einschränkung von YOLO. Weiters ist das Verfahren nicht restlos rotationsinvariant, sofern das System nicht speziell für diesen Fall trainiert wurde. In Abbildung 7.12 sind die Analyseergebnisse von Testbildern vor und nach einer Rotation um 180° dargestellt. Hier ist zu sehen, dass das Ergebnis maximal gleich gut oder schlechter geworden ist. Jedoch lässt sich hier keine eindeutige Tendenz erkennen, aus welchem Grund sich die Ergebnisse verschlechtern.

7.2.4 Verbesserungen

Wie bereits in Abschnitt 7.2.2 erwähnt, gibt es Probleme bei der Generalisierung von Werbung. Dies ist auf den kleinen Trainings-Datensatz, der nur 844 Bilder umfasst, zurückzuführen. Für YOLOv3 wird empfohlen, pro Klasse etwa 2000 Bilder zu labeln, bzw. Data Augmentation zu verwenden. Im besten Fall wären diese aus dem Kontext vom Straßenverkehr. Weiters sollen im gleichen Umfang auch negative Beispielbilder, also Bilder ohne Werbung, für das Training verwendet werden. Statt der minimalen Iterationsanzahl sollten bei nur einer zu detektierenden Klasse 4000 Iterationen durchgeführt werden. Um das Ergebnis weiter zu verbessern, könnten mehrere Trainingsdurchläufe mit verschiedenen Konfigurationseinstellungen der Hyperparameter durchgeführt werden. Eine weitere Verbesserung wäre die Vergrößerung der Bilder während des Trainingsvorganges. Im Umfang dieser Arbeit wurden die Bilder für den Trainingsprozess

Abbildung 7.12: Ausgewählte Testbilder, die die Ergebnisse vor und nach Rotation durch Analyse mit YOLOv3 zeigen. Die linke Spalte stellt die erkannte Werbung vor der Rotation des Bildes dar. Die rechte Spalte stellt die erkannten Werbungen im rotierten Bild dar.

auf eine Auflösung von 416×416 skaliert. Auch wenn fälschlicherweise als Werbung erkannte Objekte keinen großen Einfluss haben, so würde das Training von YOLOv3 auf mehrere kontextbezogene Klassen sinnvoll sein.

7.3 Zusammenfassung

Deep Learning eignet sich sehr gut für Objekterkennungsprobleme. Es werden keine komplizierten, spezifischen Algorithmen benötigt, die für jedes Objekt unterschiedlich wären. Stattdessen kann ständig dieselbe Projektstruktur verwendet werden. Lediglich die Datensätze zum Trainieren des Netzwerkes und einige Konfigurationsparameter sind individuell zu wählen. Für die Umsetzung von Deep Learning Ansätzen gibt es eine Vielzahl von Frameworks, von denen TensorFlow das am weitesten verbreitete ist. Als Netzwerkarchitektur sollte, wenn möglich, auf ein bewährtes Modell zurückgegriffen werden. Steht nur eine geringe Menge an Daten zur Verfügung, ist Transfer Learning eine ausgezeichnete Methode, um ein neuronales Netzwerk zu trainieren. Dabei wird das Netzwerk mit einem anderen, größeren Datensatz antrainiert bzw. mit bereits trainierten Gewichtungen initialisiert. Für die Erkennung von Werbung wurden etwa 1000 Bilder gelabelt, die in einen Trainings-Datensatz und einen Validierungs-Datensatz im Verhältnis von ungefähr 80 zu 20 aufgeteilt wurden. Für das Erkennen von Werbung in einem Bild oder Video wurden zwei Deep Learning Methoden praktisch umgesetzt und am Anwendungsfall von Dashcam-Videos getestet. Die beiden Methoden sind Mask R-CNN (s. Abschnitt 2.7.1) und YOLOv3 (s. Abschnitt 2.7.2). Diese haben sich angeboten, da sie den Fokus jeweils auf einen anderen Bereich legen. Mask R-CNN fokussiert sich auf möglichst gute Vorhersagen, dagegen setzt YOLOv3 auf die Analysegeschwindigkeit. Für Mask R-CNN müssen die Objekte im Trainings-Datensatz exakt entlang der Umrisse mit Polygonen markiert werden. Bei YOLO werden die Objekte mit Begrenzungsrahmen gelabelt. Transfer Learning kann sowohl mit Mask R-CNN als auch mit YOLOv3 eingesetzt werden. Mit beiden Techniken wurden gute Ergebnisse (s. Kapitel 6) erzielt. Mask R-CNN erreichte eine Durchschnittsgenauigkeit von 94.5%, YOLO erlangte einen Wert von 60.0%. Beide Werte liegen in einem Bereich, der auch von vergleichbaren Arbeiten erzielt wird. Sowohl Mask R-CNN als auch YOLOv3 haben Grenzen, die durch Optimierungen ausgereizt werden können. Mask R-CNN hat den Nachteil, dass andere Bildbereiche zum Teil fälschlicherweise als Werbung kategorisiert werden. YOLOv3 hat die Einschränkung, dass die Generalisierung von Werbung mit dieser Trainings-Datensatz-Größe nicht fehlerfrei möglich ist. Weiters hat YOLO Schwierigkeiten bei der Erkennung von kleinen Objekten. YOLOv3 hat gegenüber Mask R-CNN den Vorteil, dass die Bildanalyse je nach System und Implementierung etwa um den Faktor 20 schneller ist.

Als weiterführende Arbeiten könnten die erkannten Werbungen zusätzlich auf deren Auffälligkeit für Autofahrer analysiert werden. Dazu könnte die Position und deren Größe der erkannten Werbeflächen in Betracht gezogen werden. Weiters kann die Dauer, wie lange sich diese im Blickfeld des Autofahrers befinden, berechnet werden. Eine weitere Möglichkeit wäre, diese Dashcam-Videos mit GPS Positionen abzugleichen, um auch den Standort von Werbungen speichern und analysieren zu können. Dafür könnte die Werbung zusätzlich einer jeweiligen Firma zugeordnet werden. Die Arbeit kann als Basis verwendet werden, um Werbung in Videos gegebenfalls ausblenden zu können.

Anhang A

Inhalt der DVD

Im Folgenden wird der Inhalt der DVD beschrieben.

A.1 Masterarbeit

Pfad: /Masterarbeit

Thesis_2019.pdf	Masterarbeit mit Instruktionen (Gesamtdokument)
/Bilder	Unterordner mit den in der Arbeit verwendeten Bildern
/Quellen	Unterordner mit einer Sammlung von den in der Arbeit verwendeten Quellen

A.2 Projekt

Pfad: /Projekt

/Ergebnisse	Unterordner mit Testbildern und Testvideos (Mask R-CNN & YOLOv3)
/Sonstiges	Unterordner mit allgemeinen Dateien wie zum Beispiel statistische Auswertungen
/Umsetzung	Unterordner mit den Quelldateien des Projektes (Mask R-CNN & YOLOv3)

Quellenverzeichnis

Literatur

[1] Lotfi Abdi und Aref Meddeb. „Deep Learning Traffic Sign Detection, Recognition and Augmentation“. In: *Proceedings of the 32nd Symposium on Applied Computing*. SAC'17. Marrakech, Morocco: ACM, Apr. 2017, S. 131–136 (siehe S. 35).

[2] Shivang Agarwal, Jean Ogier Du Terrail und Frédéric Jurie. *Recent Advances in Object Detection in the Age of Deep Convolutional Neural Networks*. Techn. Ber. Caen, France: Greyc, Sep. 2018. URL: https://hal.archives-ouvertes.fr/hal-01869779. Vorveröffentlichung (siehe S. 33).

[3] Sukhad Anand u. a. „Crack-pot: Autonomous Road Crack and Pothole Detection“. In: *Proceedings of the 20th International Conference on Digital Image Computing: Techniques and Applications*. DICTA'18. Canberra, Australia: IEEE, Dez. 2018, S. 1–6 (siehe S. 35).

[4] Yu Bao u. a. „Region-based CNN for Logo Detection“. In: *Proceedings of the 8th International Conference on Internet Multimedia Computing and Service*. ICIMCS'16. Xi'an, China: ACM, Aug. 2016, S. 319–322 (siehe S. 34).

[5] Josef Bengtson u. a. „Deep learning methods for recognizing signs/objects in road traffic“. Bachelorarbeit. Göteborg, Sweden, Mai 2018. URL: http://studentarbeten.chalmers.se/publication/255863-deep-learning-methods-for-recognizing-signsobjects-in-road-traffic (siehe S. 4).

[6] G. Cai, L. Chen und Junchang Li. „Billboard advertising detection in sport TV“. In: *Proceedings of the 7th International Symposium on Signal Processing and Its Applications*. ISSPA'03. Paris, France: IEEE, Juli 2003, S. 537–540 (siehe S. 33).

[7] Wendi Cai u. a. „Street Object Detection Based on Faster R-CNN“. In: *Proceedings of the 37th Chinese Control Conference*. CCC'18. Wuhan, China: IEEE, Juli 2018, S. 9500–9503 (siehe S. 34).

[8] Xiong Changzhen u. a. „A traffic sign detection algorithm based on deep convolutional neural network“. In: *Proceedings of the 1st IEEE International Conference on Signal and Image Processing*. ICSIP'16. Beijing, China: IEEE, Aug. 2016, S. 676–679 (siehe S. 35).

[9] Tanushyam Chattopadhyay und Ayan Chaki. „Identification of Trademarks Painted on Ground and Billboards Using H.264 Compressed Domain Features from Sports Videos“. In: *Proceedings of the 2nd International Conference on Computational Intelligence, Modelling and Simulation*. CSSIM'10. Tuban, Indonesia: IEEE, Sep. 2010, S. 219–223 (siehe S. 33).

[10] Jifeng Dai u. a. „R-FCN: Object Detection via Region-based Fully Convolutional Networks“. In: *Proceedings of the 29th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., Dez. 2016, S. 379–387 (siehe S. 27).

[11] Christian Eggert u. a. „Improving Small Object Proposals for Company Logo Detection“. In: *Proceedings of the 7th ACM on International Conference on Multimedia Retrieval*. ICMR'17. Bucharest, Romania: ACM, 2017, S. 167–174 (siehe S. 34).

[12] Jorge E Espinosa, Sergio A Velastin und John W Branch. „Motorcycle detection and classification in urban Scenarios using a model based on Faster R-CNN“. In: *Proceedings of the 9th International Conference on Pattern Recognition Systems*. ICPRS'18. Valparaíso, Chile: Institution of Engineering und Technology, Mai 2018, 16 (6 pp.)–16 (6 pp.) (Siehe S. 34).

[13] Mark Everingham u. a. „The Pascal Visual Object Classes (VOC) Challenge“. *International Journal of Computer Vision* 88 (Juni 2010), S. 303–338 (siehe S. 40).

[14] M. Farrajota, J. M.F. Rodrigues und J. M.H. du Buf. „Using Multi-Stage Features in Fast R-CNN for Pedestrian Detection“. In: *Proceedings of the 7th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion*. DSAI'16. Vila Real, Portugal: ACM, Dez. 2016, S. 400–407 (siehe S. 35).

[15] Ross B. Girshick. „Fast R-CNN“. In: *Proceedings of the 15th IEEE International Conference on Computer Vision*. ICCV'15. Santiago, Chile: IEEE, Dez. 2015, S. 1440–1448 (siehe S. 20).

[16] Ross Girshick u. a. „Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation“. In: *Proceedings of the 27th IEEE Conference on Computer Vision and Pattern Recognition*. CVPR'14. Columbus, OH, USA: IEEE, Juni 2014, S. 580–587 (siehe S. 19).

[17] Yusuke Hara u. a. „Sidewalk-level People Flow Estimation Using Dashboard Cameras Based on Deep Learning“. In: *Proceedings of the 11th International Conference on Mobile Computing and Ubiquitous Network*. ICMU'18. Auckland, New Zealand: IEEE, Okt. 2018, S. 1–6 (siehe S. 35).

[18] Kaiming He u. a. „Deep Residual Learning for Image Recognition“. In: *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition*. CVPR'16. Las Vegas, NV, USA: IEEE, Juni 2016, S. 770–778 (siehe S. 17, 18).

[19] Kaiming He u. a. „Mask R-CNN“. In: *Proceedings of the 16th IEEE International Conference on Computer Vision*. ICCV'17. Venice, Italy: IEEE, Okt. 2017, S. 2980–2988 (siehe S. 22, 40, 51).

[20] Hendry und Rung-Ching Chen. „A New Method for License Plate Character Detection and Recognition“. In: *Proceedings of the 6th International Conference on Information Technology: IoT and Smart City*. ICIT'18. Hong Kong, Hong Kong: ACM, Dez. 2018, S. 204–208 (siehe S. 35).

[21] Murhaf Hossari u. a. „ADNet: A Deep Network for Detecting Adverts“. In: *Proceedings of the 26th AIAI Irish Conference on Artificial Intelligence and Cognitive Science*. AICS'18. Dublin, Ireland, Dez. 2018, S. 1–9 (siehe S. 33).

[22] Qichang Hu u. a. „Fast Detection of Multiple Objects in Traffic Scenes With a Common Detection Framework“. *IEEE Transactions on Intelligent Transportation Systems* 17 (Apr. 2016), S. 1002–1014 (siehe S. 34).

[23] Jonathan Huang u. a. „Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors“. In: *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition*. CVPR'17. Honolulu, HI, USA: IEEE, Juli 2017, S. 3296–3297 (siehe S. 27).

[24] Syed Hussain, Munther Abualkibash und Samir Tout. „A Survey of Traffic Sign Recognition Systems Based on Convolutional Neural Networks“. In: *Proceedings of the 18th IEEE International Conference on Electro/Information Technology*. EIT'18. Rochester, MI, USA: IEEE, Mai 2018, S. 570–573 (siehe S. 35).

[25] Tripidok Intasuwan, Jakkraphatara Kaewthong und Sirion Vittayakorn. „Text and Object Detection on Billboards“. In: *Proceedings of the 10th International Conference on Information Technology and Electrical Engineering*. ICITEE'18. Kuta, Indonesia: IEEE, Juli 2018, S. 6–11 (siehe S. 33).

[26] Sergey Ioffe und Christian Szegedy. „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift“. In: *Proceedings of the 32nd International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, Juli 2015, S. 448–456 (siehe S. 12).

[27] Won J. Jeon u. a. „Real-time Detection of Speed-limit Traffic Signs on the Real Road Using Haar-like Features and Boosted Cascade“. In: *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*. ICUIMC'14. Siem Reap, Cambodia: ACM, Jan. 2014, 93:1–93:5 (siehe S. 34).

[28] Jihie Kim u. a. „Real-time Traffic Sign Detection with Vehicle Camera Images“. In: *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication*. ICUIMC'13. Kota Kinabalu, Malaysia: ACM, Jan. 2013, 20:1–20:4 (siehe S. 35).

[29] Alex Krizhevsky, Ilya Sutskever und Geoffrey E. Hinton. „ImageNet Classification with Deep Convolutional Neural Networks“. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, NV, USA: Curran Associates Inc., Dez. 2012, S. 1097–1105 (siehe S. 16).

[30] Jan Kukacka, Vladimir Golkov und Daniel Cremers. „Regularization for Deep Learning: A Taxonomy“. 2017. URL: http://arxiv.org/abs/1710.10686. Vorveröffentlichung (siehe S. 12).

[31] Meeta Kumar u. a. „Neural Network Based Smart Vision System for Driver Assistance in Extracting Traffic Signposts“. In: *Proceedings of the CUBE International Information Technology Conference*. CUBE'12. Pune, India: ACM, Sep. 2012, S. 246–251 (siehe S. 35).

[32] Rayson Laroca u. a. „A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector“. In: *Proceedings of the 27th International Joint Conference on Neural Networks*. IJCNN'18. Rio de Janeiro, Brazil: IEEE, Juli 2018, S. 1–10 (siehe S. 35).

[33] Yann LeCun, Yoshua Bengio und Geoffrey Hinton. „Deep Learning“. *Nature* 521.7553 (Mai 2015), S. 436–444 (siehe S. 4, 7).

[34] Yann LeCun u. a. „Gradient-Based Learning Applied to Document Recognition“. *Proceedings of the IEEE* 86 (Nov. 1998), S. 2278–2324 (siehe S. 16).

[35] Hee Seok Lee und Kang Kim. „Simultaneous Traffic Sign Detection and Boundary Estimation Using Convolutional Neural Network“. *IEEE Transactions on Intelligent Transportation Systems* 19 (Mai 2018), S. 1652–1663 (siehe S. 35).

[36] Chenge Li u. a. „TrackNet: Simultaneous Object Detection and Tracking and Its Application in Traffic Video Analysis“. Feb. 2019. URL: http://arxiv.org/abs/1902.01466. Vorveröffentlichung (siehe S. 34).

[37] Mengyue Li, Yuchun Guo und Yishuai Chen. „CNN-based Commercial Detection in TV Broadcasting“. In: *Proceedings of the 6th International Conference on Network, Communication and Computing*. ICNCC'17. Kunming, China: ACM, Dez. 2017, S. 48–53 (siehe S. 33).

[38] Peiliang Li, Xiaozhi Chen und Shaojie Shen. „Stereo R-CNN based 3D Object Detection for Autonomous Driving“. In: *Proceedings of the 32nd IEEE Conference on Computer Vision and Pattern Recognition*. CVPR'19. IEEE, Feb. 2019, S. 1–9 (siehe S. 34).

[39] Tsung-Yi Lin u. a. „Feature Pyramid Networks for Object Detection“. In: *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition*. CVPR'17. Honolulu, HI, USA: IEEE, Juli 2017, S. 936–944 (siehe S. 43).

[40] Tsung-Yi Lin u. a. „Focal Loss for Dense Object Detection“. In: *Proceedings of the 16th IEEE International Conference on Computer Vision*. ICCV'17. Venice, Italy: IEEE, Okt. 2017, S. 2999–3007 (siehe S. 27).

[41] Wei Liu u. a. „SSD: Single Shot MultiBox Detector“. In: *Proceedings of the 14th European Conference on Computer Vision*. ECCV'16. Cham, Germany: Springer International Publishing, 2016, S. 21–37 (siehe S. 27).

[42] Yufeng Ma u. a. „Effects of user-provided photos on hotel review helpfulness: An analytical approach with deep leaning“. *International Journal of Hospitality Management* 71 (Apr. 2018), S. 120–131 (siehe S. 13).

[43] Fabien Moutarde u. a. „Modular Traffic Sign Recognition applied to on-vehicle real-time visual detection of American and European speed limit signs“. In: *Proceedings of the 14th World congress on Intelligent Transportation Systems*. ITS'07. Beijing, China, Okt. 2007, S. 1043–1050 (siehe S. 34).

[44] Rinat Mukhometzianov und Ying Wang. „Review. Machine learning techniques for traffic sign detection“. Dez. 2017. URL: http://arxiv.org/abs/1712.04391. Vorveröffentlichung (siehe S. 34).

[45] Koray Ozcan, Senem Velipasalar und Anuj Sharma. „Traffic Sign Detection from Lower-quality and Noisy Mobile Videos“. In: *Proceedings of the 11th International Conference on Distributed Smart Cameras*. ICDSC'17. Stanford, CA, USA: ACM, Okt. 2017, S. 15–20 (siehe S. 34).

[46] Rongqiang Qian u. a. „Road surface traffic sign detection with hybrid region proposal and fast R-CNN“. In: *Proceedings of the 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery*. ICNC-FSKD'16. Changsha, China: IEEE, Aug. 2016, S. 555–559 (siehe S. 35).

[47] Joseph Redmon und Ali Farhadi. „YOLO9000: Better, Faster, Stronger“. In: *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition*. CVPR'17. Honolulu, HI, USA: IEEE, Juli 2017, S. 6517–6525 (siehe S. 24, 26–29).

[48] Joseph Redmon und Ali Farhadi. *YOLOv3: An Incremental Improvement*. Techn. Ber. Apr. 2018, S. 6. URL: https://pjreddie.com/media/files/papers/YOLOv3.pdf. Vorveröffentlichung (siehe S. 27, 29, 40).

[49] Joseph Redmon u. a. „You Only Look Once: Unified, Real-Time Object Detection“. In: *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition*. CVPR'16. Las Vegas, NV, USA: IEEE, Juni 2016, S. 779–788 (siehe S. 24–26).

[50] Christoph Reinders u. a. „Object Recognition from very few Training Examples for Enhancing Bicycle Maps“. In: *Proceedings of the 29th IEEE Intelligent Vehicles Symposium*. IV'18. Changshu, Suzhou, China: IEEE, Juni 2018, S. 1–8 (siehe S. 34).

[51] Shaoqing Ren u. a. „Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks“. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'15. Montreal, Canada: Curran Associates, Inc., Jan. 2015, S. 91–99 (siehe S. 19, 21).

[52] Sebastian Ruder. „An overview of gradient descent optimization algorithms“. 2016. URL: http://arxiv.org/abs/1609.04747. Vorveröffentlichung (siehe S. 9).

[53] David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams. „Learning representations by back-propagating errors“. *Nature* 323 (Okt. 1986), S. 533–536 (siehe S. 10).

[54] Hua Shen und Xiaoou Tang. „Generic Sign Board Detection in Images“. In: *Proceedings of the 5th ACM SIGMM International Workshop on Multimedia Information Retrieval*. MIR'03. Berkeley, CA, USA: ACM, Jan. 2003, S. 144–149 (siehe S. 34).

[55] Karen Simonyan und Andrew Zisserman. „Very Deep Convolutional Networks for Large-Scale Image Recognition“. In: *Proceedings of the 3rd International Conference on Learning Representations*. ICLR'15. San Diego, CA, USA, Mai 2015, S. 1–14 (siehe S. 17).

[56] Nitish Srivastava u. a. „Dropout: A Simple Way to Prevent Neural Networks from Overfitting“. *Journal of Machine Learning Research* 15 (Jan. 2014), S. 1929–1958 (siehe S. 12).

[57] Christian Szegedy u. a. „Going deeper with convolutions“. In: *Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition*. CVPR'15. Boston, MA, USA: IEEE, Juni 2015, S. 1–9 (siehe S. 17).

[58] J. R. R. Uijlings u. a. „Selective Search for Object Recognition“. *International Journal of Computer Vision* 104 (Sep. 2013), S. 154–171 (siehe S. 19).

[59] Liu Wei, Lu Runge und Liu Xiaolei. „Traffic sign detection and recognition via transfer learning“. In: *Proceedings of the 30th Chinese Control And Decision Conference*. CCDC'18. Shenyang, China: IEEE, Juni 2018, S. 5884–5887 (siehe S. 35).

[60] Sebastien C. Wong u. a. „Understanding Data Augmentation for Classification: When to Warp?“ In: *Proceedings of the 18th International Conference on Digital Image Computing: Techniques and Applications*. DICTA'16. Gold Coast, Australia: IEEE, Nov. 2016, S. 1–6 (siehe S. 10).

[61] Jie Yang u. a. „An Automatic Sign Recognition and Translation System“. In: *Proceedings of the 2001 Workshop on Perceptive User Interfaces*. PUI'01. Orlando, FL, USA: ACM, Nov. 2001, S. 1–8 (siehe S. 34).

[62] Jie Yang u. a. „Towards Automatic Sign Translation“. In: *Proceedings of the 1st International Conference on Human Language Technology Research*. HLT'01. San Diego, CA, USA: Association for Computational Linguistics, März 2001, S. 1–6 (siehe S. 34).

[63] Yuchen Yang u. a. „Efficient Traffic-Sign Recognition with Scale-aware CNN“. In: *Proceedings of the 28th British Machine Vision Conference*. BMVC'17. London, UK: BMVA Press, Sep. 2017, 168:1–168:13 (siehe S. 35).

[64] So Yeon Jo u. a. „Transfer Learning-based Vehicle Classification“. In: *Proceedings of the 15th International SoC Design Conference*. ISOCC'18. Daegu, South Korea: IEEE, Nov. 2018, S. 127–128 (siehe S. 34).

[65] Jason Yosinski u. a. „How Transferable Are Features in Deep Neural Networks?“ In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: Curran Associates, Inc., Dez. 2014, S. 3320–3328 (siehe S. 18, 33).

[66] J. D. Zhao, Z. M. Bai und H. B. Chen. „Research on Road Traffic Sign Recognition Based on Video Image“. In: *Proceedings of the 10th International Conference on Intelligent Computation Technology and Automation*. ICICTA'17. Changsha, China: IEEE, Okt. 2017, S. 110–113 (siehe S. 35).

[67] Zhongrong Zuo u. a. „Traffic Signs Detection Based on Faster R-CNN“. In: *Proceedings of the 37th International Conference on Distributed Computing Systems Workshops*. ICDCSW'17. Atlanta, GA, USA: IEEE, Juni 2017, S. 286–288 (siehe S. 34).

Audiovisuelle Medien

[68] *West Hollywood Billboard Row Monthly Update October 2018*. Okt. 2018. URL: https://www.youtube.com/watch?v=yM8qIX9wxWU (besucht am 12. 05. 2019) (siehe S. 2, 37, 38, 61, 66, 67, 70, 71).

Software

[69] Waleed Abdulla. *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. 2017. URL: https://github.com/matterport/Mask_RCNN (besucht am 18. 04. 2019) (siehe S. 38, 43).

[70] *Detectron*. 2018. URL: https://github.com/facebookresearch/Detectron (besucht am 14. 04. 2019) (siehe S. 23, 38).

[71] *Detectron.pytorch*. URL: https://github.com/roytseng-tw/Detectron.pytorch (besucht am 25. 05. 2019) (siehe S. 38).

[72] Jian Guo. *Parallel Faster R-CNN implementation with MXNet*. URL: https://github.com/ijkguo/mx-rcnn (besucht am 25. 05. 2019) (siehe S. 38).

[73] *Microsoft Cognitive Toolkit (CNTK)*. URL: https://github.com/microsoft/CNTK (besucht am 25. 05. 2019) (siehe S. 38).

[74] Rafael Padilla. *Metrics for object detection*. URL: https://github.com/rafaelpadilla/Object-Detection-Metrics (besucht am 15. 05. 2019) (siehe S. 40).

[75] Petar Veličković. *TikZ*. Jan. 2019. URL: https://github.com/PetarV-/TikZ (besucht am 30. 01. 2019) (siehe S. 12).

Online-Quellen

[76] *AI Notes: Initializing neural networks*. URL: https://www.deeplearning.ai/ai-notes/initialization/ (besucht am 01. 06. 2019) (siehe S. 5).

[77] *Anaconda*. URL: https://www.anaconda.com/ (besucht am 30. 01. 2019) (siehe S. 43).

[78] Anup Bhande. *What is underfitting and overfitting in machine learning and how to deal with it*. Medium. März 2018. URL: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76 (besucht am 01. 06. 2019) (siehe S. 11, 12).

[79] *Caffe*. URL: http://caffe.berkeleyvision.org/ (besucht am 30. 01. 2019) (siehe S. 30).

[80] Jaron Collis. *Glossary of Deep Learning: Bias*. Medium. Apr. 2017. URL: https://medium.com/deeper-learning/glossary-of-deep-learning-bias-cf49d9c895e2 (besucht am 01. 06. 2019) (siehe S. 5).

[81] *Comparison of AI Frameworks*. Skymind. URL: http://skymind.ai/wiki/comparison-frameworks-dl4j-tensorflow-pytorch (besucht am 12. 05. 2019) (siehe S. 39).

[82] *Deep Learning Framework Power Scores 2018*. Towards Data Science. URL: https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a (besucht am 12. 05. 2019) (siehe S. 39).

[83] *Deeplearning4j*. URL: https://deeplearning4j.org/ (besucht am 30. 01. 2019) (siehe S. 31).

[84] Michael DelSole. *What is One Hot Encoding and How to Do It*. Medium. Apr. 2018. URL: https://medium.com/@michaeldelsole/what-is-one-hot-encoding-and-how-to-do-it-f0ae272f1179 (besucht am 01. 06. 2019) (siehe S. 7).

[85] Rohith Gandhi. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. Towards Data Science. Juli 2018. URL: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e (besucht am 14. 04. 2019) (siehe S. 18, 20, 22).

[86] Rohith Gandhi. *Support Vector Machine — Introduction to Machine Learning Algorithms*. Towards Data Science. 7. Juni 2018. URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (siehe S. 19).

[87] Victor Genin. *Optimizations of Gradient Descent*. März 2016. URL: http://dsdeepdive.blogspot.com/2016/03/optimizations-of-gradient-descent.html (besucht am 30. 01. 2019) (siehe S. 9).

[88] Jonathan Hui. *Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3*. Medium. März 2018. URL: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088 (besucht am 18. 04. 2019) (siehe S. 25).

[89] *Keras*. URL: https://keras.io/ (besucht am 30. 01. 2019) (siehe S. 30, 31).

[90] *Labelbox - Documentation*. URL: https://support.labelbox.com/docs (besucht am 30. 01. 2019) (siehe S. 42).

[91] Fei-Fei Li, Justin Johnson und Serena Yeung. *Detection and Segmentation*. Mai 2017. URL: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf (besucht am 02. 06. 2019) (siehe S. 20, 21).

[92] *Microsoft Cognitive Toolkit*. URL: https://www.microsoft.com/en-us/cognitive-toolkit/ (besucht am 30. 01. 2019) (siehe S. 31).

[93] *Microsoft Cognitive Toolkit (CNTK) - Documentation*. URL: https://docs.microsoft.com/en-us/cognitive-toolkit/ (besucht am 30. 01. 2019) (siehe S. 31).

[94] *MXNet*. URL: https://mxnet.apache.org/ (besucht am 30. 01. 2019) (siehe S. 31).

[95] Michael A. Nielsen. *Neural Networks and Deep Learning*. 2015. URL: http://neuralnetworksanddeeplearning.com (besucht am 01. 06. 2019) (siehe S. 7, 8, 10).

[96] *Overfitting / Underfitting – How Well Does Your Model Fit?* Mai 2017. URL: https://meditationsonbianddatascience.com/2017/05/11/overfitting-underfitting-how-well-does-your-model-fit/ (besucht am 30. 01. 2019) (siehe S. 11).

[97] Shubham Panchal. *Artificial Neural Networks — Mapping the Human Brain*. Medium. März 2018. URL: https://medium.com/predict/artificial-neural-networks-mapping-the-human-brain-2e0bd4a93160 (besucht am 01. 06. 2019) (siehe S. 3).

[98] Ayeshmantha Perera. *What is Padding in CNN's*. Medium. Sep. 2018. URL: https://medium.com/@ayeshmanthaperera/what-is-padding-in-cnns-71b21fb0dd7 (besucht am 30. 01. 2019) (siehe S. 14, 15).

[99] *PyTorch*. URL: https://www.pytorch.org (besucht am 30. 01. 2019) (siehe S. 30).

[100] Bharath Raj. *Data Augmentation - How to use Deep Learning when you have Limited Data*. Medium. Apr. 2018. URL: https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced (besucht am 30. 01. 2019) (siehe S. 10).

[101] Sebastian Raschka. *What is Softmax regression and how is it related to Logistic regression?* Mai 2019. URL: https://sebastianraschka.com/faq/docs/softmax_regression.html (besucht am 01. 06. 2019) (siehe S. 7).

[102] Adrian Rosebrock. *Intersection over Union (IoU) for object detection*. Nov. 2016. URL: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/ (besucht am 15. 05. 2019) (siehe S. 40).

[103] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Towards Data Science. Dez. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (besucht am 01. 06. 2019) (siehe S. 14).

[104] Sagar Sharma. *Epoch vs Batch Size vs Iterations*. Towards Data Science. Sep. 2017. URL: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9 (besucht am 01. 06. 2019) (siehe S. 10, 11).

[105] TechnoReview. *Artificial Neural Network : Beginning of the AI revolution*. Hacker Noon. Juni 2018. URL: https://hackernoon.com/artificial-neural-network-a843ff870338 (besucht am 30. 01. 2019) (siehe S. 4).

[106] *TensorFlow*. URL: https://www.tensorflow.org/ (besucht am 30. 01. 2019) (siehe S. 30).

[107] *The Gentlest Introduction to Tensorflow*. KDnuggets. Feb. 2017. URL: https://www.kdnuggets.com/2017/02/gentlest-introduction-tensorflow-part-4.html (besucht am 30. 01. 2019) (siehe S. 8).

[108] *Theano*. URL: http://deeplearning.net/software/theano/ (besucht am 30. 01. 2019) (siehe S. 31).

[109] *Top 5 Deep Learning Frameworks, their Applications, and Comparisons!* Analytics Vidhya. März 2019. URL: https://www.analyticsvidhya.com/blog/2019/03/deep-learning-frameworks-comparison/ (besucht am 12. 05. 2019) (siehe S. 39).

[110] Stanford University. *CS231n Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/convolutional-networks/ (besucht am 30. 01. 2019) (siehe S. 13–16).

[111] *Unsupervised Feature Learning and Deep Learning Tutorial*. URL: http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/ (besucht am 01. 06. 2019) (siehe S. 4, 5).

[112] Bar Vinograd. *Instance Embedding: Segmentation Without Proposals*. Towards Data Science. Apr. 2018. URL: https://towardsdatascience.com/instance-embedding-instance-segmentation-without-proposals-31946a7c53e1 (besucht am 14. 04. 2019) (siehe S. 23).

[113] Joyce Xu. *Deep Learning for Object Detection: A Comprehensive Review*. Towards Data Science. Sep. 2017. URL: https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9 (besucht am 14. 04. 2019) (siehe S. 21).