

# Machbarkeitsstudie: Smart Warehouse

**Echtzeit-Objektdetektoren im Vergleich**

## **Studienarbeit**

im Rahmen der Prüfung zum  
**Bachelor of Science (B.Sc.)**

des Studienganges Angewandte Informatik  
an der Dualen Hochschule Baden-Württemberg Karlsruhe

von

**Felix Hausberger und Robin Kuck**

Oktobe 2019 - Mai 2019

**-Sperrvermerk-**

Abgabedatum: 18. Mai 2020  
Bearbeitungszeitraum: 30.09.2019 - 18.05.2020  
Matrikelnummer, Kurs: 2773463, 4409176, TINF17B2  
Ausbildungsfirma: SAP SE  
Dietmar-Hopp-Allee 16  
69190 Walldorf, Deutschland  
Gutachter an der DHBW: PD Dr.-Ing. Markus Reischl

# Eidesstattliche Erklärung

Wir versichern hiermit, dass wir unsere Studienarbeit mit dem Thema:

*Machbarkeitsstudie: Smart Warehouse*

gemäß § 5 der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. September 2017 selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, den 8. März 2020

Gez. Felix Hausberger und Robin Kuck  
Hausberger, Felix und Kuck, Robin

## **Abstract**

- *English* -

In this thesis the object detectors *You Only Look Once* and *Single Shot MultiBox Detector* are compared for precision, reactivity, training and inference behaviour and examined for their potential for industrial use. The background scenario of the *Smart Warehouse* offers live video data of a drone with goods in a warehouse, which are to be classified and localized in real time. In the future, this should make it possible to carry out inventories and inventory analyses of a warehouse in a time- and cost-efficient manner conserving resources.

The goal of this feasibility study is to find out whether the *Smart Warehouse* scenario is technically feasible and economically reasonable. In addition, the focus is also on the object detectors themselves and their differences in architecture and behavior in the *Smart Warehouse* environment.

## **Abstract**

- *Deutsch* -

In dieser Arbeit werden die Objektdetektoren *You Only Look Once* und *Single Shot MultiBox Detector* nach Präzision, Reaktionsvermögen, Trainings- und Interaktionsverhalten miteinander verglichen und auf deren Potential zum industriellen Einsatz untersucht. Das Hintergrundszenario des *Smart Warehouses* bietet dabei Live-Video Daten einer Drohne mit Warengegenständen in einem Warenhaus, die in Echtzeit klassifiziert und lokalisiert werden sollen. Dadurch sollen in Zukunft in der Industrie Inventuren und Bestandsanalysen eines Warenhauses zeit- und kostengünstig sowie ressourcenschonend ermöglicht werden können.

Diese Machbarkeitsstudie hat zum Ziel herauszufinden, ob das Szenario des *Smart Warehouse* technisch umsetzbar sowie wirtschaftlich sinnvoll ist. Zusätzlich liegt der Fokus ebenso auf den Objektdetektoren selbst und deren Unterschiede hinsichtlich Architektur und Verhalten im *Smart Warehouse* Umfeld.

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>VI</b>
<b>Abbildungsverzeichnis</b>	<b>VIII</b>
<b>Formelverzeichnis</b>	<b>IX</b>
<b>Listenverzeichnis</b>	<b>X</b>
<b>0 Vorwort</b>	<b>1</b>
<b>1 Einführung</b>	<b>2</b>
1.1 Forschungsumfeld . . . . .	2
1.2 Problemstellung und Motivation . . . . .	3
1.3 Vorgehensweise und Zielsetzung . . . . .	3
<b>2 Grundlagen und Forschungsstand</b>	<b>5</b>
2.1 Deep Learning zur Bildverarbeitung . . . . .	5
2.2 Neuronale Netze . . . . .	7
2.3 Hyperparameter . . . . .	12
2.4 Datensatzlehre . . . . .	17
2.5 Objektdetektoren . . . . .	21
2.6 Cloud Infrastruktur . . . . .	35
<b>3 Konzeption</b>	<b>39</b>
3.1 Bewertungskriterien . . . . .	39
3.2 Vergleich der Objektdetektoren . . . . .	40
3.3 Echtzeitumgebung . . . . .	41
<b>4 Realisierung</b>	<b>43</b>
4.1 Trainieren der Objektdetektoren . . . . .	43
4.2 Drohnen Anbindung . . . . .	44
4.3 Dashboard Entwicklung . . . . .	44
<b>5 Ergebnisse</b>	<b>45</b>

<b>6</b>	<b>Bewertung</b>	<b>46</b>
<b>7</b>	<b>Zusammenfassung und Ausblick</b>	<b>47</b>
	<b>Literaturverzeichnis</b>	<b>XI</b>
<b>A</b>	<b>Anhang</b>	<b>XV</b>

# Abkürzungsverzeichnis

<b>ANN</b>	Artificial Neural Network
<b>CNN</b>	Convolutional Neural Network
<b>COCO</b>	Common Objects in Context
<b>CLI</b>	Command Line Interface
<b>CPU</b>	Central Processing Unit
<b>CUDA</b>	Compute Unified Device Architecture
<b>DBN</b>	Deep Belief Network
<b>ELU</b>	Exponential Linear Unit
<b>FCN</b>	Fully Convolutional Network
<b>FLOPS</b>	Floating Point Operations Per Second
<b>FPGA</b>	Field Programmable Gate Array
<b>FPS</b>	Frames Per Second
<b>GPU</b>	Graphics Processing Unit
<b>IoU</b>	Intersection over Union
<b>LReLU</b>	Leaky Rectified Linear Unit
<b>mAP</b>	mean Average Precision
<b>PReLU</b>	Parametric Rectified Linear Unit
<b>PascalVOC</b>	Pascal Visual Object Classes
<b>PaaS</b>	Platform-as-a-Service
<b>R-CNN</b>	Regional Convolutional Neural Network
<b>ReLU</b>	Rectified Linear Unit
<b>RoI</b>	Region of Interest
<b>RPN</b>	Region Proposal Network

<b>SaaS</b>	Software-as-a-Service
<b>SSD</b>	Single Shot MultiBox Detector
<b>TOPS</b>	Tera Operations Per Second
<b>TPU</b>	Tensor Processing Unit
<b>XML</b>	Extensible Markup Language
<b>YOLO</b>	You Only Look Once

# Abbildungsverzeichnis

2.1	Anwendungsgebiete von Deep Learning zur Bildverarbeitung im Überblick	6
2.2	Linear Threshold Unit . . . . .	7
2.3	Das einschichtige Perzeptron . . . . .	9
2.4	Gradientenverfahren . . . . .	15
2.5	ReLU-Aktivierungsfunktionen . . . . .	16
2.6	ELU . . . . .	17
2.7	Convolutional Layer . . . . .	22
2.8	Zero-Padding . . . . .	22
2.9	Feature Maps . . . . .	23
2.10	Pooling Layer . . . . .	24
2.11	Intersection over Union . . . . .	26
2.12	Berechnung mAP . . . . .	27
2.13	R-CNN . . . . .	28
2.14	Faster R-CNN . . . . .	29
2.15	SSD Bounding Box Proposals . . . . .	30
2.16	Bounding Boxes . . . . .	30
2.17	SSD Architektur . . . . .	31
2.18	Vergleich SSD . . . . .	32
2.19	Vereinfachte Darstellung der Objekterkennung mit dem YOLO Algorithmus [28] . . . . .	34
2.20	YOLO Architektur [28] . . . . .	35
2.21	Vergleich V100 - TPU Pod . . . . .	37
3.1	Die neun Kategorien . . . . .	41
3.2	Smart Warehouse Regal . . . . .	42

# Formelverzeichnis

2.1	Die Heaviside-Funktion . . . . .	8
2.2	Die Softmax-Funktion . . . . .	8
2.3	Die RMSE-Funktion . . . . .	10
2.4	Neuberechnung der Gewichtungsmatrix durch partielle Differentiation . .	11
2.5	Superpositionsprinzip anhand der Varianz . . . . .	13
2.6	Standardverteilung nach Xavier Initialisierung . . . . .	13
2.7	Momentum Optimierung . . . . .	14
2.8	Precision und Recall [1] . . . . .	25
2.9	Die Smooth L1 Funktion . . . . .	31

# Listenverzeichnis

2.1	PascalVOC Bildannotation . . . . .	17
2.2	Konfigurationsdatei zum Trainingsjob . . . . .	37

## **0. Vorwort**

Besonderen Dank ist an unseren Betreuer PD Dr. -Ing. Markus Reischl auszusprechen, ohne den die folgenden Forschungsergebnisse nicht zustande gekommen wären. Auch dem Informatik Labor unter Enrico Hühneborg der DHBW ist für die nötige finanzielle Unterstützung zum Erwerb der Drohne zu danken.

# 1. Einführung

## 1.1. Forschungsumfeld

Einen Teilbereich des maschinellen Lernens (engl.: machine learning) stellt das *Deep Learning* dar, welches auf künstlichen neuronalen Netzen (engl.: artificial neural networks) (ANNs) basiert [2]. Unter einer Vielzahl von Typen von ANNs wie Autoencodern, Deep Boltzmann Machines oder rekurrenten neuronale Netzen befindet sich ebenso die Klasse der *Convolutional Neural Networks* (CNNs), welche hauptsächlich zur Lösung von Klassifikationsproblemen in der Audio-, Text- und Bildverarbeitung genutzt werden [3].

Ein Forschungsfeld im *Deep Learning* stellen Objektdetektoren dar, welche basierend auf CNNs neben Bildklassifikationsproblemen ebenso in der Lage sind, Lokalisationsprobleme zu lösen. Solchen Objektdetektoren werden in der heutigen Zeit immer mehr Bedeutung zugesprochen angesichts neuer Herausforderungen wie autonomen Fahren, automatisierter industrieller Verarbeitung oder aber auch staatlicher Überwachung. Verschiedene Ansätze werden zur Realisierung von Objektdetektoren verwendet, unter anderem Netzarchitekturen wie *You Only Look Once* (YOLO) oder *Single Shot MultiBox Detector* (SSD).

Gerade in Zeiten des industriellen Wandels in Richtung *Industrie 4.0* können solche Objektdetektoren ein großes Optimierungspotential für bestehende Industrieszenarien bieten, beispielsweise in der Lagerhaltung und Logistik. Kombiniert mit einer autonomen Drohne können Objektdetektoren es ermöglichen, ohne menschliche Hilfe Inventuren und Bestandsprüfungen in einem Lager- oder Warenhaus durchzuführen. Start-up Unternehmen wie *doks. innovation* werben bereits mit ähnlichen Lösungen, die 80% Zeiteinsparung und 90% Kostensenkung versprechen [4]. Lösungen wie *inventAIRyX* beschränken sich allerdings speziell auf Lagerhäuser, in denen die verpackten Waren mittels Sensoren identifiziert werden, was Großhändler mit Warenhäusern wie *Baumarkt* oder *Selgros* ausschließt. Statt Waren mittels RFID Chips oder Barcodes zu identifizieren, soll in dieser Arbeit der Einsatz von Objektdetektoren für dieses Szenario evaluiert werden.

Wie sich die unterschiedlichen Objektdetektoren unter Echtzeitvoraussetzungen im Be-

trieb verhalten, soll anhand des Industriebeispiels *Smart Warehouse* innerhalb dieser Arbeit untersucht werden.

## 1.2. Problemstellung und Motivation

Das *Smart Warehouse* beschreibt ein Warenhaus, welches unter Einsatz einer Drohne in der Lage sei soll, Inventuren und Bestandsprüfungen weitgehend ohne menschliche Hilfe durchzuführen. Das Live-Bild der Drohne soll von den Objektdetektoren dazu genutzt werden, Warengegenstände zu lokalisieren und klassifizieren.

Neben der Frage, ob ein solches Industrieszenario überhaupt umsetzbar und wirtschaftlich sinnvoll ist, sollen die Objektdetektoren in diesem Anwendungsszenario nach verschiedenen Kriterien miteinander verglichen und beurteilt werden. Diese Kriterien lassen sich hauptsächlich in die Kategorien Präzision, Reaktionsvermögen, Trainings- und Interaktionsverhalten untergliedern und werden später genauer eingeführt. Dadurch lassen sich Aussagen darüber treffen, ob nach dem momentanen Forschungsstand um Objektdetektoren solche das Potential bieten, industriell eingesetzt zu werden.

Falls die Machbarkeitsstudie des *Smart Warehouse* glückt, so kann der Industrie ein kostengünstiges, zeitsparendes und ressourcenschonendes Modell zur Inventurverwaltung eines Warenhauses angeboten werden.

## 1.3. Vorgehensweise und Zielsetzung

Im Grundlagenkapitel II muss sich mit den theoretischen Grundlagen von CNNs und Objektdetektoren auseinander gesetzt werden. Hierzu ist zunächst eine Einführung in neuronale Netz erforderlich, darunter zu Perzepronen, dem Gradientenverfahren, dem Backpropagation Algorithmus und Hyperparametern zum Trainieren eines neuronalen Netzes (siehe Kapitel 2.1 und 2.2).

Nachdem kurz auf den Grundbaustein moderner Objektdetektoren eingegangen wird, den CNNs, können anschließend die Funktionsweisen und Architekturen der zwei miteinander verglichenen Objektdetektoren *YOLO* und *SSD* erläutert werden (siehe Kapitel 2.3). Bei *YOLO* ist zu bemerken, dass unterschiedliche Evolutionsstufen der drei Detek-

toren zu betrachten sind.

Um weitere Grundlagen zum Umfeld während des Trainierens von neuronalen Netzen einzuführen, wird anschließend über die Anforderungen eines Datensatzes gesprochen (siehe Kapitel 2.4), bevor Trainingsinfrastrukturen in der Cloud vorgestellt werden (siehe Kapitel 2.5).

In der Konzeptionsphase in Kapitel III sollen zunächst die Vergleichskriterien eingeführt werden und deren Metriken anschließend für initiale Benchmarkdatensätze für jeden Objektdetektor ermittelt werden. Hierzu wird auf die Datensätze *Pascal Visual Object Classes* (PascalVOC), *Common Objects in Context* (COCO) und ImageNet zurückgegriffen. Anschließend wird der Datensatz für das *Smart Warehouse* Szenario eingeführt.

In der Realisierung werden die Herausforderungen zur Steuerung und Anbindung der Drohne betrachtet und zudem die Objektdetektoren auf die realen Datensätze trainiert. Auch die Entwicklung der Webapplikation zur Visualisierung des Live-Bildes und der erkannten Objekte wird Bestandteil dieses Kapitels sein. Die Ergebnisse der Realisierungsphase werden im folgenden Kapitel dargestellt.

Ziel der Arbeit ist es Aussagen über die Fähigkeit von Objektdetektoren zum Einsatz in der Industrie zu treffen, indem eine Bewertung der Verhaltensweisen der Objektdetektoren nach den eingeführten Bewertungskriterien durchgeführt wird. Auch wirtschaftliche Gesichtspunkte werden in diesem Kapitel nicht außer Acht gelassen.

Zuletzt wird das Wesen der Arbeit nochmals kurz zusammengefasst und anschließend auf mögliche Verbesserungen und Ausblicke in die Zukunft aufmerksam gemacht.

## 2. Grundlagen und Forschungsstand

Neben einer Einführung in den Anwendungsbereich von *Deep Learning* zur Bildverarbeitung soll sich das folgende Kapitel speziell mit Architekturen unterschiedlicher Objektdetektoren auseinander setzen und herausstellen, wie sich diese voneinander abgrenzen. Davor wird allerdings zunächst grundlegendes Wissen über neuronale Netze und wie diese „lernen“ vermittelt sowie wie ein eigener Datensatz zu gestalten ist. Abschließend werden technische Grundlagen für die Programmierung des *Smart Warehouse* Szenarios aufgezeigt.

### 2.1. Deep Learning zur Bildverarbeitung

Ein klassisches Anwendungsgebiet von *Deep Learning* zu Bildverarbeitung oder auch allgemein von maschinellem Lernen beschreibt die *Klassifikation*. Hierbei werden bestimmte Kategorien, auch *Klassen* genannt, definiert, in die ein Bild eingeordnet werden soll. Die *Klassifikation* wird anhand von aus dem Bild extrahierten Merkmalen, auch *Features* genannt, getroffen. Die Merkmale werden zu einem *Merkmalsvektor* oder auch *Feature Map* zusammengefasst und von einem *neuronalen Netz* verarbeitet. Das Ergebnis der Verarbeitung durch das neuronale Netz ist die Einordnung in eine bestimmte Klasse.

Zusätzlich zur *Klassifikation* eines Bildes kann das auf dem Bild abgebildete Objekt ebenfalls lokalisiert werden. Es wird dann von sogenannter *Objektdetektion* gesprochen. Es können auch mehrere Objekte auf einem Bild detektiert werden. Ergebnis der Objektdetektion ist somit nicht nur eine Klasseneinordnung sondern ebenso eine klare Positionsangabe des Objektes auf dem Bild. Die Positionsangabe erfolgt durch Angabe sogenannter *Bounding Boxen*. Diese umrahmen das jeweils detektierte Objekt und werden durch ihren linken oberen Eckpunkt sowie ihre Höhe und Breite beschrieben.

Neben der klassischen *Klassifikation* und der *Objektdetektion* existiert ebenso ein drittes Anwendungsgebiet von *Deep Learning* zu Bildverarbeitung, die *Segmentierung*. Bei der *semantischen Segmentierung* wird versucht, jede einzelne Pixel eines Bildes einer Klasse zuzuordnen und dementsprechend farblich im Bild zu hinterlegen. *Instanzbasierte*

*Segmentierung* hingegen zielt darauf ab, nicht nur jeden Pixel zu einer Klasse zu klassifizieren, sondern ebenso eine Identität zu einem Objekt zuzuweisen [5]. Es setzt sich zusammen aus *semantischer Segmentierung* und paralleler *Objektdetektion*.

Einen Überblick über die vorgestellten Anwendungsgebiete ist in Abbildung 2.1 zu sehen.

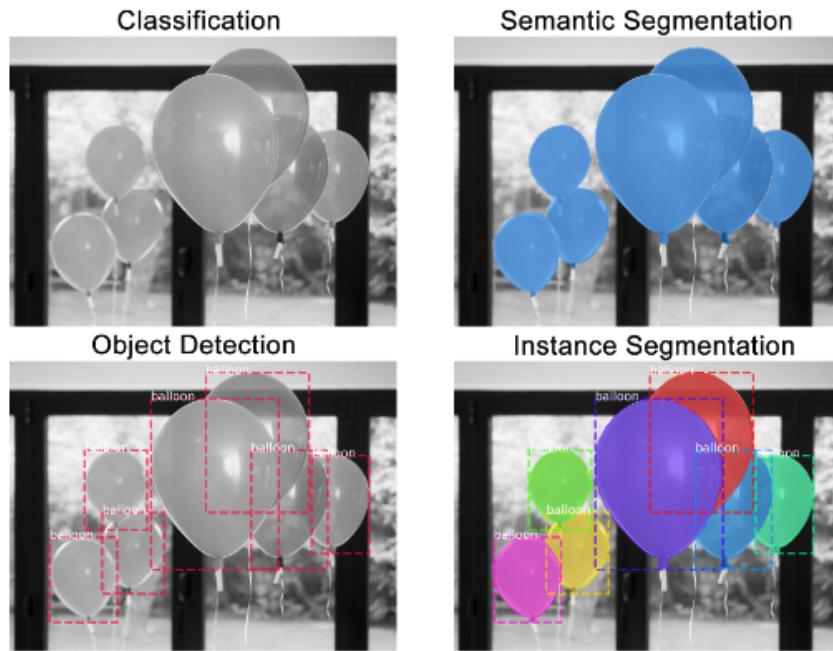


Abbildung 2.1.: Anwendungsgebiete von Deep Learning zur Bildverarbeitung im Überblick [6]

Für eine einfache *Klassifikation* eines Bildes können einfache sogenannte *Feed-Forward* Netze verwendet werden. Es kann hierbei aber auch auf konventionelle Methoden der Bildverarbeitung zurück gegriffen werden. Für die *Objektdetektion* stehen Architekturen wie *You Only Look Once* (YOLO), der *Single Shot MultiBox Detector* (SSD) oder neuronale Netze der *Regional Proposal Neural Networks* (R-CNN) bereit. Aus dieser Familie entstammt ebenso das *Mask R-CNN* Netz, das zur *Instanzbasierten Segmentierung* von Objekten verwendet wird.

## 2.2. Neuronale Netze

Ein neuronales Netz bildet die Grundlage des *Deep Learnings* [2]. Zunächst soll die einfachste Architektur eines neuronalen Netzes, das Perzeptron [2], exemplarisch erklärt werden als auch der Lernprozess eines maschinellen Lernmodells an sich, um darauf basierend die Auswirkungen von Hyperparametern auf den Lernprozess des Modells zu erklären.

### Das Perzeptron

Der Aufbau eines typischen Perzeptrons besteht aus einer oder mehreren Schichten so genannter *Linear Threshold Units* (LTU) wie in Abbildung 2.2 dargestellt.

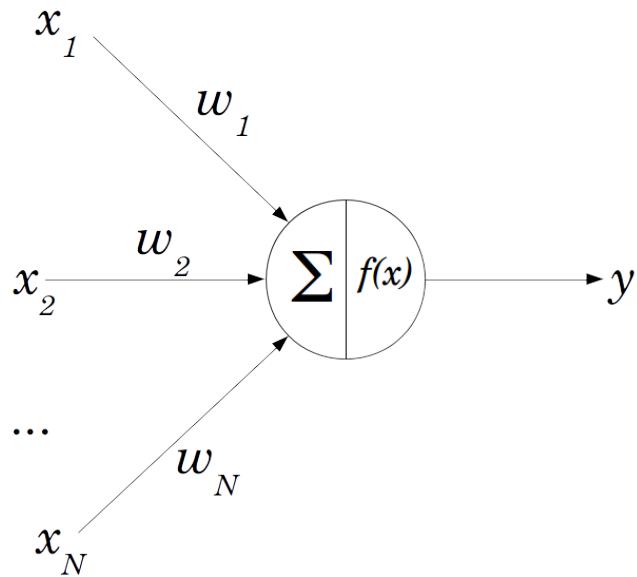


Abbildung 2.2.: Linear Threshold Unit [7]

Es besteht aus  $n$  Eingängen mit  $x_i \in \mathbb{Q}$ , die im Inputvektor  $\mathbf{x}$  zusammengefasst werden. Jeder Eingang wird mit einem Gewicht  $w_i$  aus dem Gewichtsvektor  $\mathbf{w}$  versehen [2]. Die LTU berechnet das Skalarprodukt  $\mathbf{w}^T \circ \mathbf{x}$  aller Eingänge  $\mathbf{x}$  mit ihren Gewichten  $\mathbf{w}$  und wendet anschließend auf das Ergebnis  $z$  eine Aktivierungsfunktion an [2]. Das Ergebnis

$h_w(x)$  kann anschließend als Eingabe für ein weiteres Perzeptron dienen. Die einfachste Aktivierungsfunktion für ANNs ist die *Heaviside-Funktion* [2]:

$$h_w(x) = s(\mathbf{w}^T \circ \mathbf{x}) = s(z) = \\ \left( w_1 \quad w_2 \quad \dots \quad w_n \right) \circ \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{cases} 1 & \text{wenn } z \geq 0 \\ 0 & \text{wenn } z < 0. \end{cases} \quad (2.1)$$

Falls eine Klassifizierung mit Wahrscheinlichkeiten vorliegen soll, so ist die letzte Schicht eines Perzeptrons meist mit der *Softmax-Funktion*

$$h_w(x) = \sigma(z)_j = \frac{e^{z_j}}{\sum_{i=0}^n e^{z_i}} \quad (2.2)$$

implementiert, die den Wert des  $j$ -ten LTUs einer Schicht mit allen anderen  $n$  Werten der LTUs derselben Schicht ins Verhältnis setzt [2]. Es gibt eine Vielzahl an möglichen Aktivierungsfunktionen, die im darauffolgenden Unterkapitel *Hyperparameter* betrachtet werden.

Die Aktivierung einer LTU hängt zusätzlich von einem Schwellwert  $\theta$  ab, der durch einen sogenannten *Bias* festgelegt wird. Dies ist die Gewichtung des letzten Eingangs, der standardmäßig den Wert 1 liefert. Wird die Gewichtung negativ gewählt, so ist es schwieriger die LTU zu aktivieren, während eine positive Gewichtung die Aktivierung vereinfacht [2].

Nun bilden ein oder mehrere Schichten solcher LTUs ein Perzeptron. Jede einzelne LTU ist dabei mit allen LTUs der vorherigen Schicht verbunden (siehe Abbildung 2.3) [2]. Hier wird auch von sogenannten vollständig verbundenen Schichten (engl.: *Fully-Connected Layer*) gesprochen. Die beiden LTUs zur Ausgabe können dabei Aussagen über eine Klassifikation von Daten anhand der Eingangsdaten treffen, während die LTUs im Input Layer wesentlich die Daten weiter reichen. Die Verbindungen zur ersten Schicht des Hidden Layer sind stets mit Eins belegt. Existiert keine verborgene Schicht, so wird das

ANN als einschichtiges Perzeptron bezeichnet, ab einer oder mehr verborgenen Schichten wird bereits von einem *Multi-Layer Perzeptron* (MLP), einem mehrschichtigen Perzeptron, gesprochen [2]. Ist das neuronale Netz optimal trainiert, so ist am Ende nur eines der LTUs zur Ausgabe aktiviert. Das folgende ANN ist zudem ein Beispiel für ein sogenanntes *Feed Forward Network*, bei dem die Auswertung der Daten von einer Schicht zur nächsten weitergereicht wird, ohne zu bereits besuchten Schichten zurückzukehren [2].

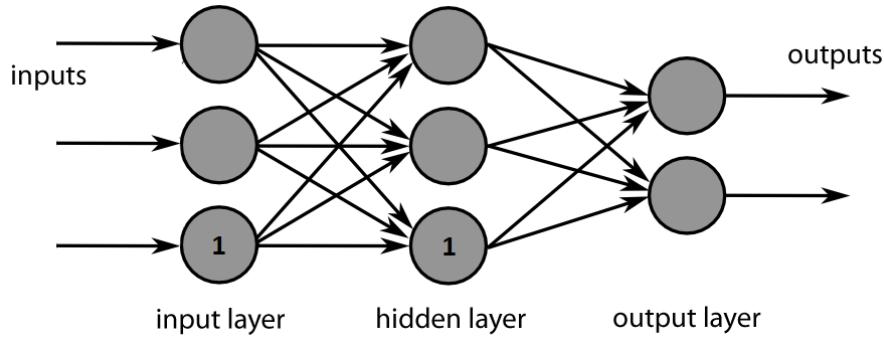


Abbildung 2.3.: Das einschichtige Perzeptron [8]

## Lernmethoden

Im Wesentlichen existieren vier Methoden, mit deren Hilfe neuronale Netze trainiert werden können. Beim *überwachten Lernen* werden den Trainingsdaten Lösungen, sogenannte *Labels*, hinzugefügt. Klassifikationsprobleme stellen eine typische Problemstellung für überwachte Lernverfahren dar. Auch die später eingeführten Objektdetektoren werden mittels überwachtem Lernen trainiert [2].

Beim *unüberwachten Lernen* werden den Trainingsdaten keinerlei Lösungen hinzugefügt, der Algorithmus muss selbstständig Klassifikationsaussagen treffen können. Ein Beispiel hierzu wäre *K-Means Clustering* [2].

*Deep Belief Networks* (DBNs) bestehend aus einzelnen *Restricted Boltzmann Machines* (RBMs) werden zunächst unüberwacht trainiert, bevor im *Fine Tuning* das Gesamtnetzwerk mit überwachten Lerntechniken fertiggestellt wird. Es wird hier von *halbüberwachtem Lernen* gesprochen.

Als letztes ist das sogenannte *Reinforcement Learning* zu nennen. Hierbei wird ein neu-

ronales Netz durch das Erteilen von Belohnungen bzw. Bestrafungen so konditioniert, dass es zukünftig basierend auf der wahrgenommenen Umwelt selbstsicher die richtige Aktionen auswählen kann.

## Gradientenverfahren und Backpropagation

Um zu verstehen, wie ein neuronales Netz durch überwachtes Lernen „lernt“, muss zunächst der Begriff der Kostenfunktion (engl.: cost function) eingeführt werden. Die Kostenfunktion ist ein Qualitätsmaß dafür, wie weit die Ausgabe einer LTU vom erwarteten Wert abweicht [2]. Angenommen dem neuronalen Netz wird ein Datensatz zur Klassifikation übergeben, so ist am Ende meist nicht nur eine LTU zu Ausgabe aktiviert, was auf eine eindeutige Klassifikation schließen würde, sondern meist mehrere zu einem frühen Stadium des neuronalen Netzes.

Eine oft genutzte Kostenfunktion ist die *Root Mean Squared Error Funktion* (RMSE) [2]:

$$E(\mathbf{z}, \mathbf{o}) = \sqrt{\frac{1}{n} \sum_{k=0}^n \|z_k - o_k\|^2} = \sqrt{\frac{1}{n} \sum_{k=0}^n (z_k - o_k)^2}. \quad (2.3)$$

Hierbei ist  $z$  der erwartete Ausgabevektor des Perzeptrons beim überwachten Lernen, während  $o$  die momentane Ausgabe darstellt. Den Fehler der Abweichung dieser beiden Werte gilt es nun schrittweise zu minimieren. Um dies zu erreichen können die drei Parameter

1. Gewichtung der Verbindungen zum Perzeptron
2. Bias zur Aktivierung der LTUs des Perzeptrons und
3. Stärke der Aktivierung des vorherigen Perzeptrons

angepasst werden [2]. Hierbei wird das sogenannte *Gradientenverfahren* eingesetzt. Es berechnet in einem iterativen Prozess über mehrere Testdaten das globale Minimum der Kostenfunktion nach den Gewichtungen der Verbindungen und damit auch nach den Bias Werten, die natürlich ebenso Gewichtungen darstellen. Ergebnis eines Durchlaufs im Gradientenverfahren (2.4) ist die Gewichtungsmatrix, die die Änderung der Gewichtung

jeder einzelnen Verbindung eines Perzeptrons zu jeder LTU des Folgeperzeptrons angibt [2]:

$$w_{ijt} = w_{ijt-1} - \eta \frac{\partial E}{\partial w_{ij}}. \quad [7] \quad (2.4)$$

Das Gradientenverfahren eignet sich allerdings nur für stetig differenzierbare Funktionen ohne Plateaus. Somit können beispielsweise bei der Heaviside-Funktion als Aktivierungsfunktion Probleme auftreten, da eine Ableitung der Kostenfunktion stets Null betragen würde, wohingegen bei der später eingeführten Sigmoid-Funktion im gesamten Definitionsbereich immer kleine Änderungen der Gewichtungen zu verzeichnen wären [2].

Nun stellt sich auch der Vorteil von MSE als Kostenfunktion gegenüber anderen, durchaus komplexeren Kostenfunktionen heraus. Während MSE genau ein Minimum, das zugleich das globale Minimum der Funktion darstellt, besitzt, haben andere Kostenfunktionen im Gradientenverfahren das Problem, dass anstelle des globalen Minimums auch nur lokale Minima erreicht werden können [2]. Dies hat zur Folge, dass mehrere iterative Durchläufe mit mehreren Testdatensätzen nötig werden, um durch unterschiedliche Startkonfigurationen die unterschiedlichen Minima miteinander vergleichen zu können und damit das globale Minimum herauszustellen.

Durch das Gradientenverfahren werden somit nur diejenigen Verbindungen verstärkt, die zum richtigen Ergebnis führen.

Nun bleibt nur noch die dritte Möglichkeit zur Minimierung der Kostenfunktion übrig, die Anpassung der Stärke der Aktivierung des vorherigen Perzeptrons. Zu diesem Problem veröffentlichten David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams 1985 den sogenannten *Backpropagation-Algorithmus* [9]. Dieser berechnet mit Hilfe des Gradientenverfahrens welchen Anteil am Fehler der Ausgabe jede LTU des letzten Perzeptrons hat und anschließend welcher Anteil davon wiederum auf das vorherige Perzepron der vergorenen Schicht zurück zu führen ist. Das Gradientenverfahren wird solange wiederholt, bis die Eingangsschicht erreicht wurde, es berechnet also für jede LTU deren Anteil am Fehler des Ergebnisses [2].

Mit Hilfe des Gradientenverfahrens im Backpropagation Algorithmus wird nun also das neuronale Netz durch mehrere iterative Durchläufe trainiert, wobei das Training als

Anpassung der Gewichtungen einzelner Verbindungen zu verstehen ist.

## 2.3. Hyperparameter

Hyperparameter sind die Parameter, die zur anfänglichen Konfiguration des neuronalen Netzes als auch zur Konfiguration des Lernprozesses herangezogen werden. Um im Laufe der Arbeit verstehen zu können, wie die Objektdetektoren auf Seiten der Netzarchitektur und des Lernverhaltens optimiert wurden, ist demnach ein kurzer Einblick in den Themenbereich der Hyperparameter von Nöten.

### Anzahl der LTUs

Die Anzahl der LTUs im ANN ist dafür ausschlaggebend, wie hoch der Komplexitätsanspruch eines Klassifizierungsproblems sein darf, um noch vom ANN gelöst werden zu können. Die Anzahl der LTUs hängt hauptsächlich von den Eingangsdaten ab. Über die optimalste Anzahl an LTUs pro Schicht lässt sich allerdings nur schwer etwas vorhersagen. Generell gilt, dass bei gleicher Anzahl an LTUs tiefere Netze eine weitaus höheren Parametereffizienz aufweisen als breitere Netze, da diese schneller gegen den gewünschten Zustand konvergieren. Zudem lassen sie sich somit schneller und kostengünstiger trainieren. So müssten bei einem 2x32 Netz 1024 Gewichtungen angepasst werden, während es bei einem 32x2 Netz dies nur 128 sind [2].

### Initialisierung der Gewichtungen

Auch stellt die Initialisierung der Gewichte eines ANNs zu Beginn des Trainingsprozesses eine berechtigte Frage dar. Falls keine bereits trainierten ANNs für ein Klassifikationsproblem vorliegen, so werden die Gewichtungen meist zufällig nach einer Normalverteilung gewählt [2].

Dies hat allerdings zur Folge, dass nach der Berechnung der gewichteten Summen aller LTUs die Gewichtungswerte der folgenden Schicht nicht mehr normalverteilt sind, da für

die Varianz zweier unkorrelierter Zufallsvariablen das Superpositionsprinzip

$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) \quad (2.5)$$

gilt.

Durch die größer werdende Standardabweichung können demnach Gewichtungswerte entstehen, die weit vom Mittelwert Null abweichen. Dies kann wiederum dazu führen, dass der Gradientenabstieg während des Backpropagation-Verfahrens nur langsam vollzogen werden kann, da der Gradient bei bestimmten Aktivierungsfunktionen (siehe Abbildung ??) gegen Null konvergiert [2].

Eine *Xavier Initialisierung* umgeht das Problem der sogenannten *schwindenden Gradi- enten*, indem die Gewichte nach

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right] \quad (2.6)$$

gleichverteilt werden, wobei  $n_j$  die Anzahl an LTUs der  $j$ -ten Schicht sind [10].

## Anzahl an Epochen

Die Anzahl der Epochen beschreibt die Durchläufe durch einen bestimmten Trainingsdatensatz während der Trainingsphase. Ist die Anzahl zu hoch gewählt, wird Gefahr gelaufen, sogenanntes *Overfitting* des ANNs zu erreichen. Dies bedeutet ein fehlendes Abstraktionsvermögen des ANNs und damit alleinig eine richtige Erkennung der Trainingsdatensätze zu ermöglichen.

## Lernrate

Die Lernrate  $\eta$  (2.4) gibt an, wie groß die Sprünge sein sollen und damit indirekt wie viele Iterationen benötigt werden, um das Minimum der Kostenfunktion zu erreichen. Ziel der Anpassung einer Lernrate ist es, mit möglichst wenig Iterationen und Testdaten

die optimale Konstellation des neuronalen Netzes zu berechnen. Deshalb wird sie standardmäßig zu Beginn der Iterationen groß gewählt, um sich dem Minimum schnell zu nähern, während sie am Ende immer kleiner gewählt wird, um nicht über das globale Minimum hinaus zu gehen. Dieses Vorgehen wird als *Simulated Annealing* bezeichnet, während das Funktion zum Festlegen der Lernrate als *Learning Schedule* betitelt wird [2].

Die Anzahl der Durchläufe wird zu Beginn des Verfahrens zunächst hoch angesetzt, das Verfahren wird aber genau dann gestoppt, sobald der Gradientenvektor unter eine gewisse Abbruchgrenze fällt. Zwar ist das globale Minimum zu diesem Zeitpunkt noch nicht erreicht, allerdings kann es auch nie vollkommen erreicht werden, da die für das Gradientenverfahren genutzten Aktivierungsfunktionen nie einen partiellen Ableitungswert gleich Null zulassen [2]. In diesem Sinne wird auch von *Toleranz* gesprochen.

## Moment

Das Gradientenverfahren kann beschleunigt werden, indem während des Gradientenabstiegs frühere Gradienten Einfluss auf den nächsten Gradientenschritt nehmen. Es wird ein „Momentum“ aufgebaut. Damit das Momentum

$$m_x = \beta \cdot m_{x-1} + \eta \frac{\partial E}{\partial w_{ij}} \quad (2.7)$$

$$w_{ijt} = w_{ijt-1} - m$$

allerdings nicht zu groß wird, beschränkt der Hyperparameter  $\beta \in [0, 1]$  die Größe des Momentums [2].

Die Momentum Optimierung kann dazu benutzt werden, das *stochastische Gradientenverfahren* bzw. *Mini-Batch* Verfahren zu beschleunigen und lokale Minima besser zu überwinden.

## Auswahl des Gradientenverfahrens

Generell wird zwischen drei verschiedenen Arten unterschieden, das Gradientenverfahren durchzuführen (siehe Abbildung 2.4):

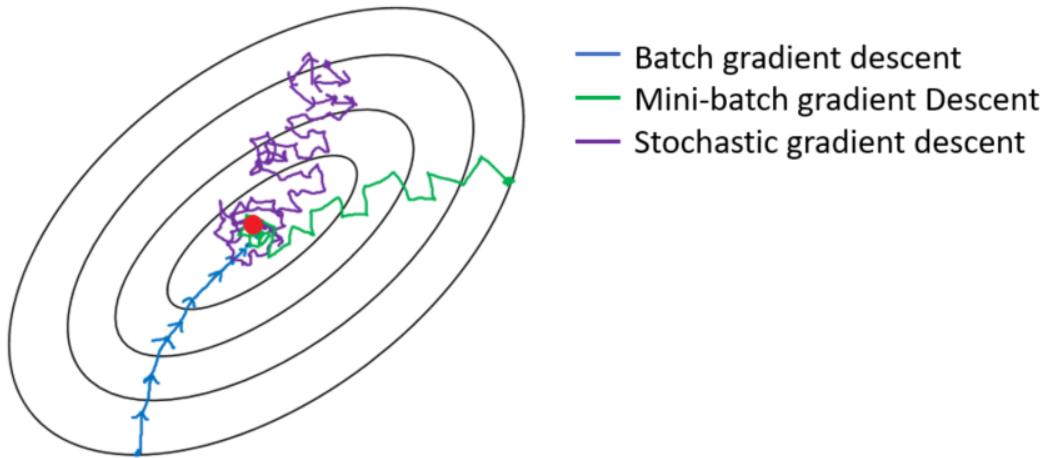


Abbildung 2.4.: Gradientenverfahren [11]

Beim *Batch* Verfahren werden in einem Trainingsdurchlauf, der *Epoche*, alle vorhandenen Daten des Trainingsdatensatzes herangezogen, um einen Gradientenabstieg zu vollziehen. Dies ist bei großen Trainingsdatensätzen auffällig langsam, dafür aber hinsichtlich der Erreichung des lokalen Minimums sehr zielstrebig [2].

Das *stochastische Gradientenverfahren* führt nach jedem einzelnen Dateneintrag im Trainingsdatensatz einen Gradientenabstieg durch. Da nur wenige Daten des ANNs verändert werden müssen, ist dieses Verfahren deutlich schneller, dafür aber unregelmäßiger hinsichtlich der Erreichung des Minimums. Oft wird das stochastische Gradientenverfahren verwendet, wenn nicht der komplette Trainingsdatensatz in den Hauptspeicher oder Grafikspeicher geladen werden kann. Diese Fähigkeit wird oft als *Out-of-Core* Fähigkeit bezeichnet. Es hat auch den Vorteil, besser das globale Minimum der Kostenfunktion aufzufinden, da bei lokalen Minima die Chance besteht, durch den unregelmäßigen Gradientenabstieg das lokale Minimum wieder zu überwinden [2].

Ein Kompromiss der beiden Verfahren bietet das *Mini-Batch* Verfahren, bei dem wie-

derholt Teilmengen des gesamten Datensatzes für einen Gradientenabstieg verwendet werden. Genauso wie das *Batch* Verfahren bietet das *Mini-Batch* Verfahren den Vorteil, die partiellen Ableitungen als Matrizenoperationen auf die Grafikkarten auszulagern, um die Performanz durch Parallelisierung zu steigern [2].

## Aktivierungsfunktionen

Zwei bekannte und ähnliche Aktivierungsfunktionen sind die *Sigmoid-Funktion* und die *Tangens Hyperbolicus* Funktion. Da diese allerdings durch ihr schnelles Konvergieren gegen den Grenzwert anfällig für das Problem *schwindender Gradienten* sind [2], wird die *Rectified Linear Unit* (ReLU) bzw. *Parametric/Leaky Rectified Linear Unit* (PReLU/LReLU) Aktivierungsfunktion bevorzugt (siehe Abbildung 2.5).

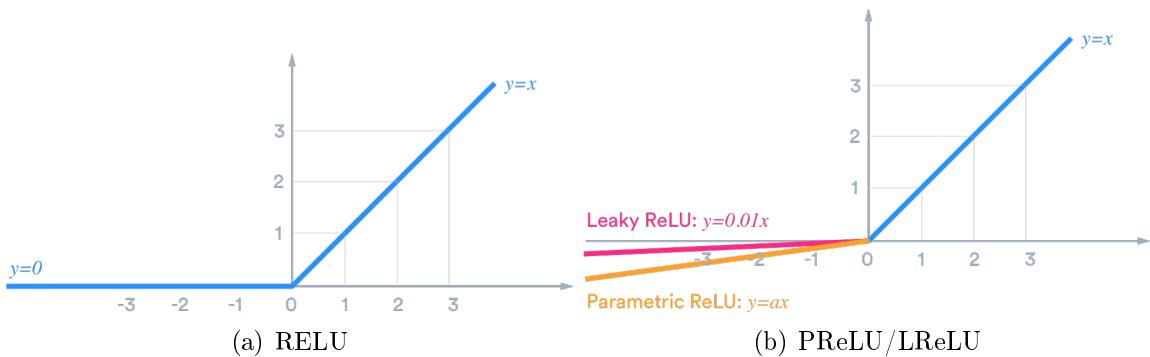


Abbildung 2.5.: ReLU-Aktivierungsfunktionen [12]

Bei ReLU kann es während des Trainingsprozesses dazu kommen, dass LTUs nach dem Gradientenabstieg einen negativen Wert aufweisen, weshalb sie nicht weiter aktiviert werden und für den Rest der Trainingsdauer „tot“ sind. Um dies zu verhindern, wurde *LReLU* dazu genutzt, um eine Reaktivierung zu ermöglichen, da auch für negative LTU Werte ein Gradient der Aktivierungsfunktion bestimmt werden kann. Bei *LReLU* ist die Steigung der Funktion im zweiten Quadranten statisch gewählt, während sie bei *PReLU* dynamisch von neuronalen Netz während des Trainingsprozesses selbst gelernt werden kann [2].

Eine letzte Variante der Aktivierungsfunktionen beschreibt die *ELU* Funktion (siehe Abbildung 2.6).

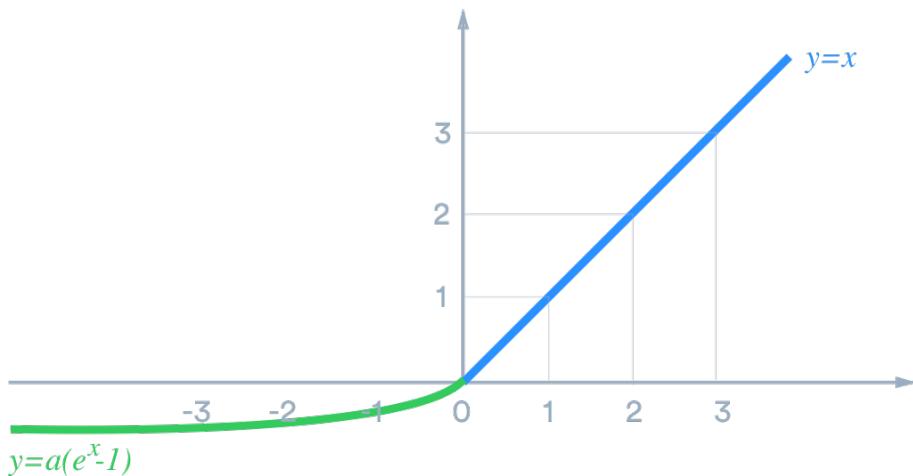


Abbildung 2.6.: ELU [12]

Sie besitzt nicht nur die Eigenschaft schwindende Gradienten und damit nicht anpassbare, sogenannte „tote“LTUs zu verhindern, sondern ist im gesamten Definitionsbereich ebenso eine stetig differenzierbare Funktion, was das Gradientenverfahren beschleunigt. Als Standardwert für den Streckungsfaktor  $\alpha$  der niederen Funktion wird oft Eins verwendet. Nachteil der *ELU* Funktion ist der erhöhte Rechenaufwand, was aber durch die schnellere Konvergenz kompensiert wird [2].

## 2.4. Datensatzlehre

### Datensatzformate

Basierend darauf, welcher Objektdetektor trainiert werden soll, muss der zum Training verwendete Datensatz in einem bestimmten Format vorliegen. Zum Trainieren des *SSDs* wird das sogenannte *Pascal Visual Object Classes* (*PascalVOC*) Format benötigt.

Es definiert eine Unterteilung in *Annotations*, *ImageSets* und *JPEGImages*. Während in dem Ordner *JPEGImages* alle Bilder des Datensatzes vorhanden sind, befindet sich unter anderem die Information über die vorhandenen Objekte in dem Bild im Ordner *Annotations*. Für jedes Bild des Datensatzes werden die Informationen in einer gleichnamigen XML-Datei abgelegt.

```

1 <annotation>
2     <folder>JPEGImages</folder>
3     <filename>000001.jpg</filename>
4     <path>..\\JPEGImages\\000001.jpg</path>
5     <source>
6         <database>Unknown</database>
7     </source>
8     <size>
9         <width>4608</width>
10        <height>2112</height>
11        <depth>3</depth>
12    </size>
13    <segmented>0</segmented>
14    <object>
15        <name>Saskia Wasser Groß</name>
16        <pose>Unspecified</pose>
17        <truncated>0</truncated>
18        <difficult>0</difficult>
19        <bndbox>
20            <xmin>1575</xmin>
21            <ymin>95</ymin>
22            <xmax>3163</xmax>
23            <ymax>635</ymax>
24        </bndbox>
25    </object>
26 </annotation>
```

Listing 2.1: PascalVOC Bildannotation

Neben allgemeinen Metainformationen über das Bild befindet sich hier ebenso eine Liste aller markierten Objekte. Pro Objekt wird die Klassifikationskategorie, die Ausrichtung (z.B. „Frontal“), die Information über vollständiges Erscheinen im Bild, die Information über schwere Erkennbarkeit und die Bounding Box angegeben. Im Ordner *ImageSets/Main* wird eine Unterteilung in Trainings- und Testdatensatz durch zwei Textdateien realisiert, die die Dateinamen der Bilddateien als Auflistung enthalten [13].

Das *YOLO* Format für den *YOLO* Objektdetektor definiert in einer *.names*-Datei alle im

Datensatz vorhandenen Kategorien durch simple Auflistung der Bezeichner. Die Bilder werden zusammen mit ihren Annotationen in einem separaten Ordner abgelegt. Die Annotationen folgen hier dem Format:

< *Kategorie-ID* > < *Zentrum-X* > < *Zentrum-Y* > < *Breite* > < *Hoehe* >

Die Unterteilung in Trainings- und Testdatensatz erfolgt durch Referenzierung der Bildpfade in zwei getrennten Textdateien. Schließlich wird in einer *.data*-Datei der Pfad zu den beiden Textdateien und zur *.names*-Datei sowie die Anzahl an Kategorien gespeichert [14].

## Datensatzzusammensetzung

Zum Erstellen und Auswählen eines *Deep Learning* Modells wird der Datenbestand in der Regel in drei Kategorien unterteilt. Ein Datensatz wird für das Training des Modells verwendet. Durch das anschließende Anwenden des Modells auf zuvor ungesehene Daten, den Testdaten, wird der *Verallgemeinerungsfehler* gemessen, der möglichst niedrig ausfallen sollte. Fällt der allgemeine Fehler während des Trainings niedrig aus, der *Verallgemeinerungsfehler* während des Testdurchlaufs allerdings hoch, so liegt klassisches *Overfitting* vor, die Trainingsdaten wurden auswendig gelernt [2].

Anschließend werden in mehreren Durchläufen die Hyperparameter des Trainingsprozesses angepasst, sodass letztendlich der *Verallgemeinerungsfehler* für die Testdaten niedrig ausfällt. Kommt es anschließend zum Einsatz des Modells in der Produktivumgebung, so können trotz allem unerwartete Ergebnisse bezüglich des Abstraktionsvermögens des Modells auftreten. Dies liegt daran, dass das Modell allein auf die Testdaten hin optimiert wurde. Um dies zu vermeiden wird ein dritter Datensatz, der Validierungsdatensatz, eingeführt. Mehrere Modelle werden dabei durch den Validierungsdatensatz getestet und das dabei am besten abschneidende Modell mit dessen Hyperparametern ausgewählt. Der eigentliche Testdatensatz wird anschließend nur noch zur Abschätzung des *Verallgemeinerungsfehlers* verwendet [2].

Oft wird der Trainingsdatensatz mit dem Validierungsdatensatz zum sogenannten *Trainval* Datensatz zusammengeführt. Dies steht im Kontext des sogenannten *K-Kreuzvalidierungsverfahrens*. Dabei wird der *Trainval* Datensatz in K gleich große, komplementäre Untermengen unterteilt. Eine dieser Untermengen dient anschließend als Validierungs-

datensatz. Für jedes zu trainierende Modell mit unterschiedlichen Hyperparametern wird eine andere Untermenge als Validierungsdatensatz ausgewählt. Hierdurch steigt die Aussagekraft des Abstraktionsvermögens nach der Validierung und zudem müssen keine Trainingsdaten dauerhaft für die Validierung zurück gelegt werden. In der Regel werden 80% der Gesamtdaten als *Trainval* Datensatz verwendet [2].

## Qualität und Quantität der Daten

Um ein funktionsfähiges Modell zu trainieren, muss der Datensatz einem gewissen Standard nachkommen. Demnach müssen die zu klassifizierenden Objekte vollständig im Bild enthalten und gut erkennbar sein. Zwar gibt es gerade im *PascalVOC* Datensatzformat ebenso die Möglichkeit, Objekte als „schwierig erkennbar“ zu markieren, dennoch sollen solche Objekte nicht die Mehrheit im gesamten Datensatz ausmachen. Auch die Aufnahme von Objekten in unterschiedlichen Umgebungen, Entfernung und Blicklagen fördert langfristig das Abstraktionsvermögen des Modells.

Ebenso muss ein ausreichend großer Datensatz vorliegen, um das gewünschte Abstraktionsvermögen des Modells zu erreichen. Die Ergebnisse aus 2.18 wurden beispielsweise durch Kombination der *Trainval* Datensätze von PascalVOC 2007 und 2012 erzielt und umfasst 16.551 Bilder im Trainingsverfahren [15] [16] [17].

Unter Hinzunahme des COCO *trainval135k* Datensatzes erreicht der *SSD* sogar das beste Ergebnis aus der ursprünglichen Veröffentlichung mit einer durchschnittlichen Präzision von 81.6% [15].

## Techniken zum Trainieren bei geringen Datenmengen

Bei Betrachtung der obigen Ergebnisse wird schnell deutlich, dass für ein komplexeres *Deep Learning* Modell ein umfangreicher Datensatz von Nöten ist. Allerdings gibt es zwei bekannte Techniken, wie auch mit kleineren Datenbeständen ein sehenswertes Ergebnis erzielt werden kann.

Beim sogenannten *Transfer Learning* können von einem bereits für ein ähnliches Problem trainiertem Modell die ersten Schichten des neuronalen Netzes für das neue Modell wiederverwendet werden. Die übernommenen Gewichtungen werden nicht mit trainiert.

Neben einer kleineren Datenmenge zum Trainieren hat das *Transfer Learning* ebenso den Vorteil das Training selbst zu beschleunigen [2].

Eine weitere Technik beschreibt das künstliche Vergrößern des Datensatzes durch affine Transformationen wie Translation, Rotation oder Skalierung und wird *Data Augmentation* genannt [2].

## 2.5. Objektdetektoren

Ein Anwendungsgebiet des *Deep Learnings* beschreiben die Objektdetektoren, die im *Smart Warehouse* Szenario zur Lokalisierung und Klassifizierung von Bestandsobjekten genutzt werden. Im folgenden Kapitel soll demnach der Grundbaustein von Objektdetektoren, das *Convolutional Neural Network*, zunächst genauer betrachtet werden, bevor auf die gängigsten Objektdetektoren, die der *Regional Convolutional Neural Networks* (R-CNN), der *Single Shot MultiBox Detector* und der *You Only Look Once* Ansatz eingegangen wird. Auch wird die gängiste Metrik zum Vergleich von Objektdetektoren, die *mean Average Precision* eingeführt.

### Convolutional Neural Networks

Ein CNN besteht größtenteils aus drei grundlegenden Bausteinen, den sogenannten *Convolutional Layern*, *Pooling Layern* und den bereits bekannten *Fully-Connected Layern*.

Ein Convolutional Layer zeichnen sich unter anderem dadurch aus, dass jede LTU dieser Schicht nicht mit allen vorherigen LTUs der vorgegangenen Schicht verbunden ist, sondern nur mit einer festen, beschränkten Anzahl. Es ist also kein vollständig verbundenes neuronales Netz. Dieser „lokale Wahrnehmungsbereich“ macht es möglich, dass örtliche Informationen und Merkmale im Bild erhalten bleiben. Auch können große Bilder klassifiziert werden, ohne dass die Anzahl an nötigen Verbindungen im ANN unüberschaubar groß wächst. Die folgenden LTUs der zweiten Schicht sind ebenfalls wiederum nur mit einem Ausschnitt vorangegangener Neuronen verbunden und fassen die erkannten, kleinteiligen Merkmale der ersten Schicht zu übergeordneten, zusammengesetzten und komplexeren Merkmalen zusammen [2].

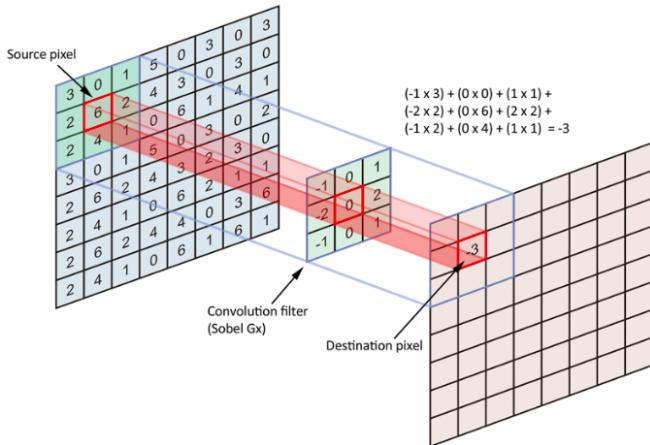


Abbildung 2.7.: Convolutional Layer [18]

0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

Abbildung 2.8.: Zero-Padding [19]

Um allerdings Convolutional Layer genauer zu verstehen, ist anstelle einer eindimensionalen Darstellung eines Layers eine dreidimensionale Darstellung besser geeignet.

Zunächst kann ein zweidimensionale Bild als Matrix dargestellt werden, bei der jedes Element der Matrix den Grauwert eines Pixels zwischen 0 und 255 trägt. Die dadurch entstandene zweidimensionale Schicht bildet den Input-Layer mit einer LTU pro Pixel. Anschließend werden auf diese Schicht nacheinander mehrere Filter angewandt, die die Gewichte des CNNs tragen und Muster aus dem Bild extrahieren. Die Stellen, die dem Muster ähnlich sind sollen verstärkt werden, während Stellen, die nicht dem Muster entsprechen durch eine Nullgewichtung ausgelöscht werden sollen. Der Lernprozess bei der Bilderkennung beruht also darauf, die bestmöglichen Filter für die gegebene Aufgabe zu finden und diese für eine Erkennung komplexer Mustern zusammen setzen zu können. In Abbildung 2.7 ist ein 3x3 Pixel Filter mit dessen Anwendung dargestellt. Ein Filter besitzt die Größe des künstlichen Wahrnehmungsbereiches einer LTU [2].

Ein Filter wird dazu verwendet, jeden Pixel der Eingabeschicht auf die folgende Schicht abzubilden. Um keine Informationen zu verlieren und den Filter ebenso auf Randbereich anwendbar zu machen, wird oft ein sogenanntes *Zero-Padding* auf eine Schicht angewandt, bei dem die Randbereiche mit LTUs des Wertes 0 aufgefüllt werden (siehe Abbildung 2.8) [2].

Falls eine gleich große folgende Schicht gewünscht ist, wird eine Schrittweite (engl.: *stride*) von 1 gewählt. Dies dient vor allem dazu kleinere Strukturen noch zu erkennen. Der Filter wird von einem Pixel zum direkt benachbarten Pixel bewegt und angewandt. In tieferen,

fortgeschritteneren Schichten kann die Schrittweite auch größer als 1 gewählt werden, da hier bereits nach dem Anwenden mehrerer Filter feinere Muster erkannt wurden und diese nun zu größeren zusammengesetzt werden. Dabei verkleinert sich die resultierende Schicht [2].

Das Ergebnis der Anwendung eines Filters wird als *Feature-Map* bezeichnet. Da mehrere Filter auf die gleiche Schicht angewandt werden, entstehen ebenso mehrere Feature Maps der Schicht. Werden diese Feature Maps übereinander gelagert vorgestellt, so entsteht der dreidimensionale, „faltungsbedingte“ (engl.: convolutional) Charakter eines Convolutional Layers. Eine Schicht eines Convolutional Layers ist mit den entsprechenden Wahrnehmungsbereichen aller vorhergehenden Feature Maps des vorhergehenden Convolutional Layers verbunden (siehe Abbildung 2.9) [2].

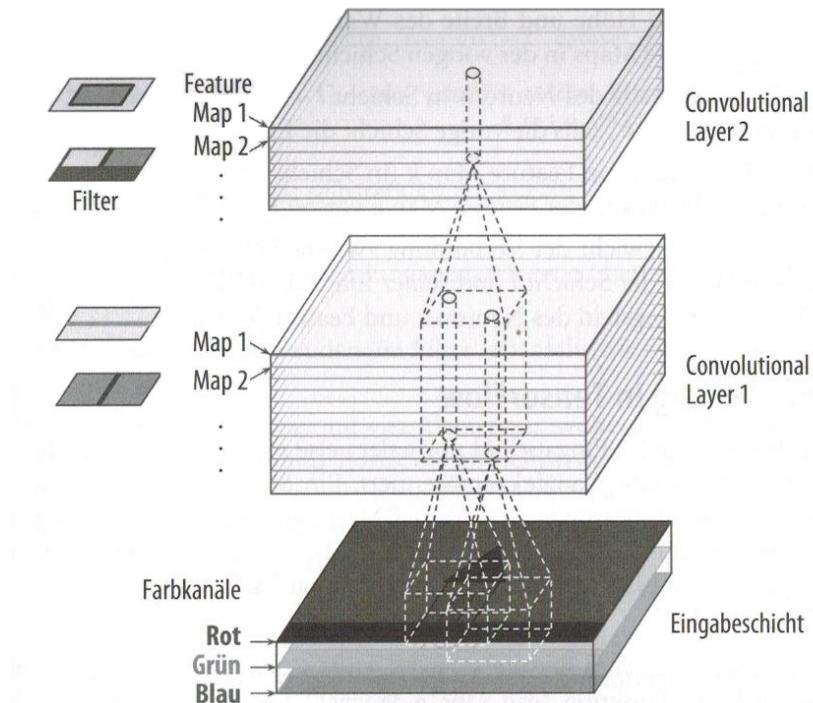


Abbildung 2.9.: Feature Maps [2]

Falls zusätzlich eine Farberkennung gewünscht ist, besitzt der Input Layer für jeden der drei Farbkanäle des RGB-Schemas eine Schicht, die Werte zwischen 0 und 255 in ihren LTUs tragen und den Stärken des Rot-, Grün- und Blaukanals entsprechen [2].

Der zweite Grundbaustein eines CNN sind Pooling Layer. Ähnlich zu den Convolutional

Layern ist auch hier jede LTU nur mit einer begrenzten Anzahl an LTUs des vorhergegangenen Layers verbunden, also nur mit dem lokalen Wahrnehmungsbereich. Der Hauptunterschied liegt aber darin, das keine Filter existieren, die die Werte vorhergehender LTUs unterschiedlich gewichten und dabei Muster erkennen. Statt den Filtern werden Aggregatfunktionen wie *MAX()* oder *MEAN()* dazu verwendet, um die Eingaben in nachfolgende Schichten zu verkleinern. So wird beispielsweise bei einem MAX-Pooling Layer mit Schrittweite größer als 1 der jeweils größte Wert des lokalen Wahrnehmungsbereiches weitergereicht und damit die Eingabe in nachfolgende Schichten verkleinert (siehe Abbildung 2.10), was mit einem Informationsverlust verbunden ist. Diese Verkleinerung des Bildes ist ein wesentlicher Schritt, um bei der Mustererkennung weiter Informationen und Merkmale abstrahieren zu können [2].

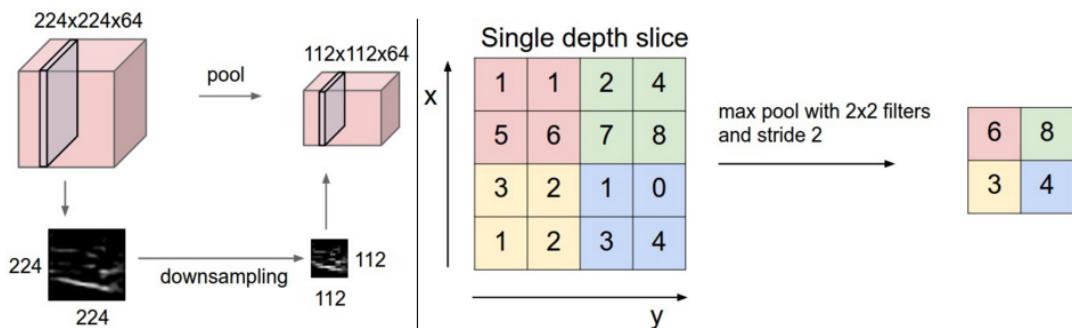


Abbildung 2.10.: Pooling Layer [20]

Daneben ist ein Pooling über die Tiefe der Feature Maps möglich. Hier bleibt die Größe der resultierenden Feature Maps gleich, die Anzahl verringert sich allerdings. Die unterschiedlichen Farbkanäle werden somit nach und nach abstrahiert [2].

Nachdem nun beide Grundbausteine eines CNNs genauer erläutert wurden, lassen sich diese nun kombinieren um ein vollständiges CNN zu bauen. Hierbei gibt es unterschiedlichste Architekturen, größtenteils äußerst komplexe. Im Rahmen dieser Arbeit genügt es allerdings, die grundlegende Architektur zu erläutern.

Diese beginnt mit einigen Convolutional Layern, die aufeinander folgen und am Ende durch eine ReLU-Funktion nochmals gefiltert und durch ein Pooling Layer abgeschlossen werden. Dies wird je nach Komplexität der zu erkennenden Muster und der Größe der Bilder einige Male wiederholt. Das ursprüngliche Bild wird durch die Pooling Layer zwar immer kleiner, allerdings auch durch die Convolutional Layer immer tiefer. Das

CNN schließt mit einem normalen *Feed-Forward ANN* mit *Fully-Connected Layern* ab, generiert dabei einen *Feature Vektor* und trifft durch eine Softmax-Funktion eine Klassifikationsaussage des Bildes [2].

Diese Architektur ermöglicht ebenso die Wiederverwendbarkeit einzelner Schichten und Gewichtungen für ähnliche Klassifikationsprobleme, bei denen gleiche Muster vorzufinden sind [2].

## Mean Average Precision

Um die Genauigkeit von Objektdetektoren zu messen, wird oft die Metrik *mean Average Precision* (mAP) gewählt. Diese setzt sich aus zwei grundlegenden Größen zusammen (siehe Formel 2.8).

$$\text{Precision} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalsePositive}} \quad (2.8)$$

$$\text{Recall} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalseNegative}}$$

*Precision* sagt also etwas über die Verlässlichkeit einer Klassifikation aus während *Recall* Aussagen über die Erkennungsfähigkeit eines Objektdetektors trifft. Wichtig ist es hierbei anzumerken, dass mehrfach detektierte Objekte nur einmal als positiver Befund aufgefasst werden, die restlichen Detektionen gehen als *False Positives* [21].

Die Klassifikation, ob eine Bounding Box das gewünschte Objekt enthält und demnach ein positiver Fall vorliegt, wird anhand eines sogenannten *Intersection over Union* (IoU) Schwellwertes bestimmt (siehe Abbildung 2.11). Er beschreibt ein Maß der Überdeckung der detektierten Bounding Box zur wahren Bounding Box und wird auch als *confidence score* bezeichnet. Für den kompletten Datensatz werden nun für unterschiedliche *confidence scores* jeweils *Precision* und *Recall* bestimmt und anschließend in einem Graphen aufgetragen. Meistens werden die *confidence scores* so gewählt, sodass sich eine äquidistante Abstufung in den *Recall* Werten ergibt [21].

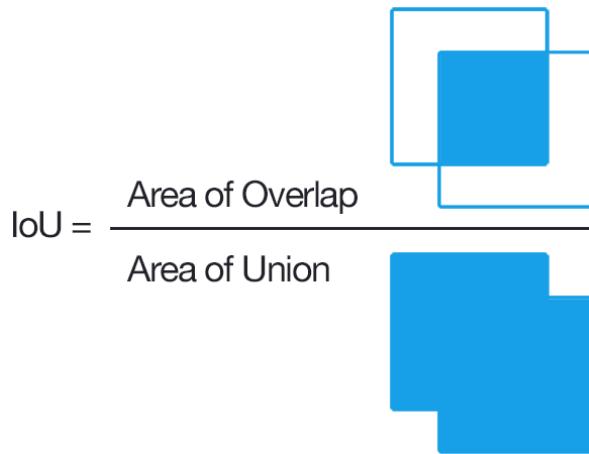


Abbildung 2.11.: Intersection over Union [22]

Im Graphen ist meist ein klassisches „Zick-Zack“ Muster zu erkennen (siehe Abbildung 2.12). Dieses Muster wird geglättet, indem nach jedem Einbruch für jeden *Recall* Wert der maximale *Precision* Wert rechts des aktuellen *Recalls* übernommen wird. Wird anschließend das diskrete Integral über alle *Recall* Werte gebildet, so ergibt sich der *Average Precision* Wert für eine zu klassifizierende Kategorie. Der Mittelwert der *Average Precisions* über alle Klassifikationskategorien hinweg ergibt letztendlich den *mAP* Wert [1].

## Regional Convolutional Neural Networks

Um eine *Objektdetektion* zu ermöglichen, reichen normale CNNs allerdings nicht aus. *Regional Convolutional Neural Networks* (R-CNNs) vertreten den Ansatz, für ein Bild Locationsvorschläge für mögliche Objekte zu liefern, sogenannte *Regions of Interest* (RoIs), und diese anschließend zu klassifizieren.

Bei dem klassischen *R-CNN* Detektor werden durch den *Selective Search* Algorithmus 2000 solcher *RoIs* vorgeschlagen. Auf den *Selective Search* Algorithmus soll im Rahmen dieser Arbeit nicht weiter eingegangen werden. Zur Merkmalsextraktion wird für jede *RoI* anschließend ein CNN eingesetzt. Der resultierende *Feature Vektor* wird zur Klassifikation eines Objektes einer *Support Vector Machine* unterzogen. Um zusätzlich die Bounding Boxen akkurat zu bestimmen, wird der *Feature Vektor* zudem einem Bounding Box Regressor unterzogen (siehe Abbildung 2.13) [23].

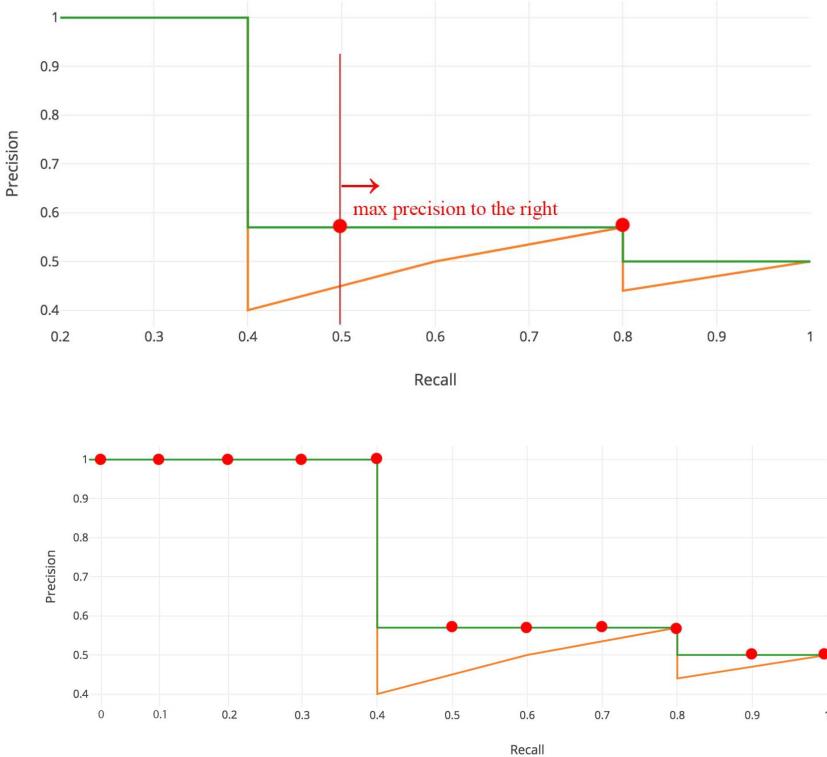


Abbildung 2.12.: Berechnung mAP [1]

Da der sogenannte *Region Proposal* Schritt durch den *Selective Search* Algorithmus allerdings viel Zeit in Anspruch nimmt, entstand eine Weiterentwicklung des *R-CNN* Netzes, das *Fast R-CNN* Netz. Dieses tauscht den Schritt des *Selective Search* Algorithmus mit dem Einsatz des CNNs. Außerdem wird das klassische CNN leicht angepasst. Bei *Fast R-CNN* wird ein Bild zunächst einem CNN unterworfen. Bevor eine *Feature Map* durch *Fully-Connected Layer* zu einem einzigen *Feature Vektor* vereinfacht wird, werden aus der *Feature Map* die verschiedenen *RoIs* extrahiert. Dies geschieht wiederum mit dem *Selective Search* Algorithmus, mit dem Unterschied, dass dieser nun nur auf der *Feature Map* operiert und nicht auf dem gesamten Bild. Durch *RoI Pooling Layer* werden die einzelnen entstandenen Regionen in eine feste Größe transformiert und einzeln einer Klassifikation durch *Fully-Connected Layer* und einer Softmax-Funktion unterworfen. Der Schritt mit dem *Bounding Box Regressor* bleibt gleich. Durch den Tausch des CNNs mit dem *Selective Search* Algorithmus werden die mathematischen Faltungsoperationen nur einmal statt 2000 Mal pro Bild ausgeführt, was die Performanz des Detektors gegenüber eines klassischen *R-CNNs* enorm steigert [23].

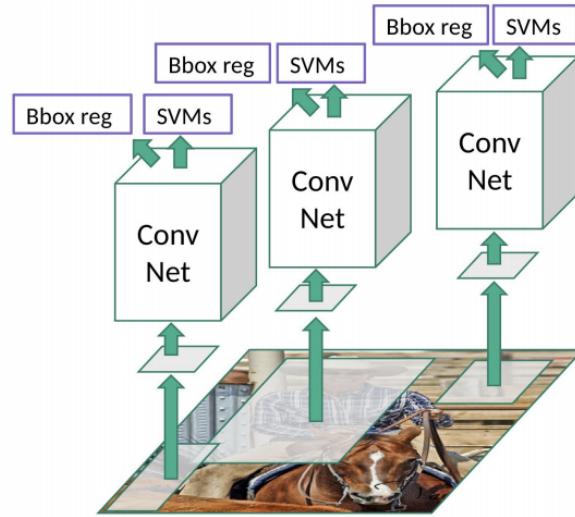


Abbildung 2.13.: R-CNN [23]

Die letzte Optimierung der R-CNN Familie entstand durch das *Faster R-CNN* Netz. Dieses ersetzt den statischen *Selective Search* Algorithmus des *Fast R-CNN* Detektors durch ein eigenes lernfähiges, sogenanntes *Region Proposal Network* (RPN) (siehe Abbildung 2.14) [23].

Neben dem Einsatz von RCNN Detektoren zur *Objektdetektion* existiert ebenso ein Ansatz zur *instanzbasierten Segmentierung*, das *Mask R-CNN* Netz. Es nimmt zwei wichtige Anpassungen an der Architektur des *Faster R-CNN* Netzes vor. Da bei Segmentierungsproblemen eine genauere Abgrenzung von Objekt und Hintergrund notwendig ist, wird das *RoI Pooling Layer* durch ein *RoI Align Layer* ausgetauscht. Hierbei wird das Rundungsproblem beim Pooling behoben. Angenommen eine *RoI* von 16x16 Pixeln wird mit einem MEAN-Pooling Layer der Schrittweite Drei verarbeitet, so ergibt sich pro Pooling Schritt ein Einzugebiet von 5.33 Pixeln. Dieses wurde abgerundet auf 5 Pixel. Bei *RoI Align Layer* wird durch bilineare Interpolation der Wert des 5.33ten Pixels ermittelt und in das Pooling mit einbezogen. Dies ermöglicht eine genauere Segmentierung an den Grenzen eines Objektes [24].

Außerdem wird parallel zum RPN ein sogenanntes *Fully Convolutional Network* (FCN) eingesetzt, einem Netz, dass rein aus *Convolutional Layer*n besteht. Es dient, um für jede existierende Klasse eine pixelbasierte binäre Maske auszugeben, die für jeden Pixel die Zugehörigkeit zu einer Klasse bestimmt. Basierend auf dieser Maske werden die

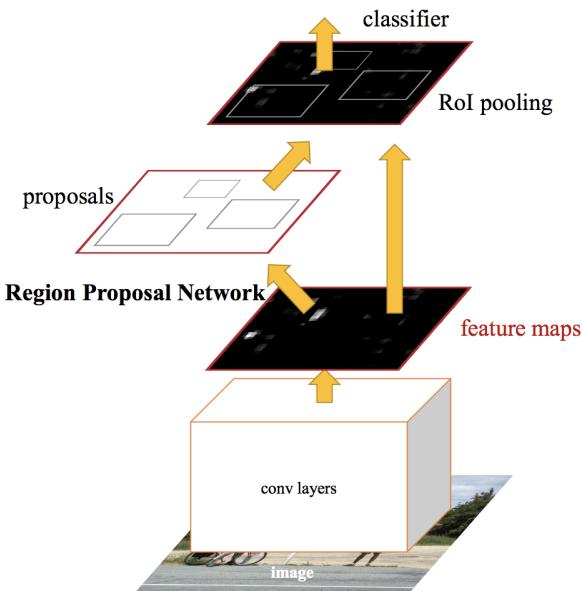


Abbildung 2.14.: Faster R-CNN [23]

detektierten Objekte anschließend farblich hervorgehoben [25].

## Single Shot MultiBox Detector

Zwar liefern die oben genannten Objektdetektoren akkurate Ergebnisse, allerdings sind sie als zu rechenintensiv und langsam einzustufen, als dass sie für Echtzeit Applikationen eingesetzt werden könnten. Der *Single Shot MultiBox Detector* (SSD) unterscheidet sich von vorhergehenden Modellen, wie beispielsweise den R-CNN Detektoren, dahingehend, dass er bewusst auf den Schritt der Generierung von Bounding Box Vorschlägen und des *Poolings* verzichtet, um wesentlich schneller ablaufen zu können als andere Objektdetektoren. Die Präzision der Klassifikationen bleibt hierbei erhalten, selbst Bilder niedriger Auflösung können weiterhin verarbeitet werden. Dem *SSD* genügt also ein einziges tiefes neuronales Netz zum Lokalisieren und Klassifizieren von Objekten. Wie der *SSD* aufgebaut ist und welche Ansätze er verfolgt, soll in diesem Unterkapitel erläutert werden [15].

Die Architektur des *SSD* zielt darauf ab, durch unterschiedlich große *Convolutional Layer Feature Maps* unterschiedlicher Skalierung in die Klassifikation mit einzufließen zu lassen.

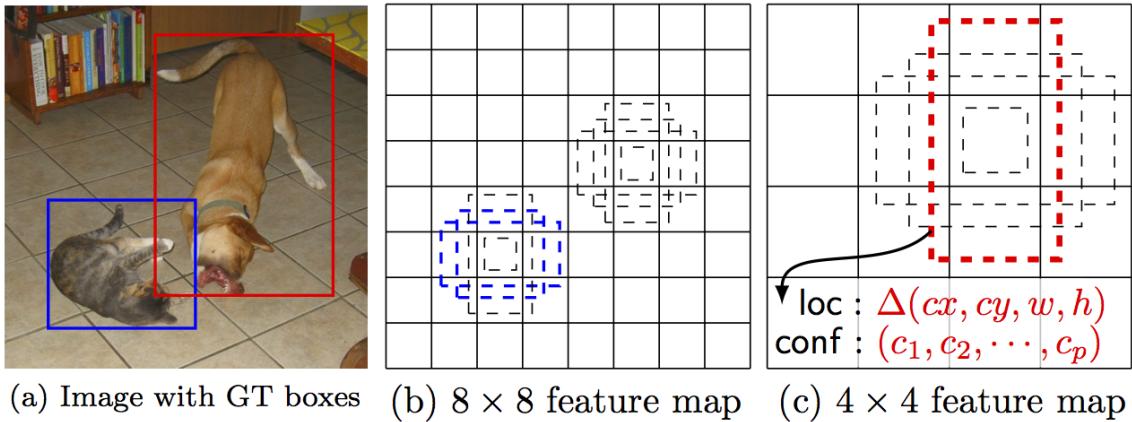


Abbildung 2.15.: SSD Bounding Box Proposals [15]

Anschaulich kann es sich vorgestellt werden, als werde das Bild in mehrere unterschiedlich große Gitterstrukturen unterteilt und die resultierenden Zellen jeweils einzeln klassifiziert. Dadurch ist es möglich, Objekte unterschiedlicher Größe zu erkennen. Für jede Zelle im Gitter wird eine gleiche Anzahl vordefinierter Bounding Boxen, die unterschiedliche Seitenverhältnisse aufweisen, definiert (siehe Abbildung 2.15). Daher entstammt der Name „MultiBox“[]. So wird sichergestellt, dass sowohl horizontal als auch vertikal ausgeprägte Objekte in der selben Zelle gleichzeitig erkannt werden können (siehe Abbildung 2.16) [15].

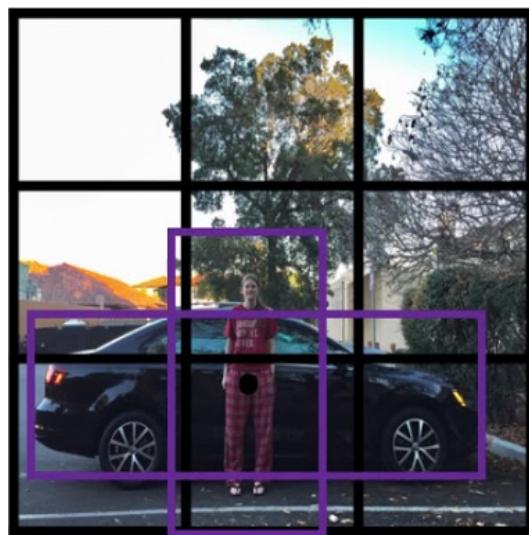


Abbildung 2.16.: Bounding Boxes [26]

Für jede dieser Bounding Boxen bestimmt der *SSD* Wahrscheinlichkeiten für Klassen-zugehörigkeiten als auch Verschiebungen der vordefinierten Bounding Box zur wahren Bounding Box des Objekts für jede Klasse. Die Kostenfunktion ist durch die gewichtete Summe des Lokalisationsverlustes und des Klassifikationsverlustes bestimmt. Während der Klassifikationverlust durch eine Softmax-Funktion bestimmt werden kann, wird der Lokalisationsverlust über die *Smooth L1* Funktion (2.9) bestimmt. Der Parameter  $l$  beschreibt die vorhergesagte Bounding Box, der Parameter  $g$  die originale Bounding Box nach den Trainingsdaten [15].

$$L_{loc}(l^j, g) = \sum_{j \in Pos}^n \sum_{i \in (x,y,w,h)}^m SM_{L1}(l_i^j - g_i)$$

$$SM_{L1}(l_i^j - g_i) = \begin{cases} 0.5x^2 & \text{wenn } |x| < 1 \\ |x| - 0.5 & \text{sonst} \end{cases} \quad (2.9)$$

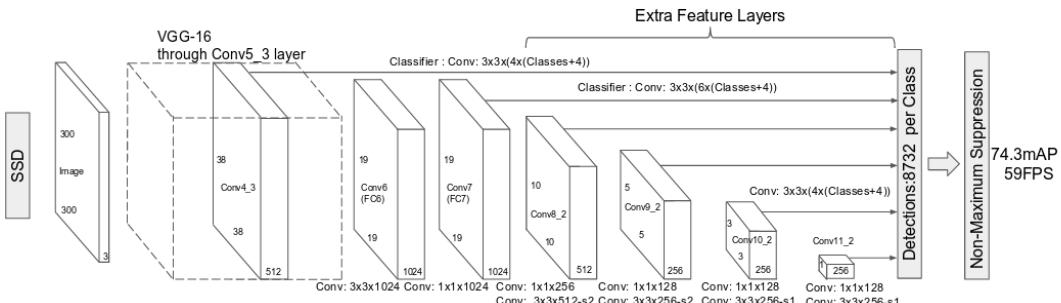


Abbildung 2.17.: SSD Architektur [15]

Technisch basiert der *SSD* auf der Idee eines *Feed-Forward Convolutional Networks*. Er benutzt ein *VGG-16* Basis Netzwerk<sup>1</sup>, dessen *Fully-Connected Layer* am Ende entfernt wurden. Die resultierende *Feature Map* wird nun einer Reihe von ständig kleiner werdenden *Convolutional Layern* unterzogen. Jedes *Convolutional Layer* kann eine feste Anzahl an Detektionen bestimmen. Eine Detektion wird durch eine Klassenangabe und die Lage einer vorhergesagten Bounding Box bestimmt. Eine Bounding Box wird wie bereits er-

<sup>1</sup> *VGG-16* ist ein auf dem Datensatz von *ImageNet* basierendes neuronales Netz, das bis zu 1000 unterschiedliche Kategorien klassifizieren kann [27].

läutert durch den linken oberen Eckpunkt  $P(x, y)$  und eine Höhe und Breite bestimmt. Bei  $c$  Klassen hat der *Feature Vektor* einer Detektion demnach die Größe  $c + 4$ . Bei einer *Feature Map* Größe von  $m \cdot n$  und  $k$  verschiedenen vordefinierten Bounding Boxen ergeben sich also  $m \cdot n \cdot k \cdot (c+4)$  verschiedene *Feature Vektoren* für eine Feature Map [15]. Diese *Feature Vektoren* werden nun an das Ende des Netzes zur Klassifikation weitergeleitet.

Dieser Vorgang wird für alle *Feature Maps* für alle *Convolutional Layer* durchgeführt. Die daraus folgende Menge an Detektionen wird durch ein *Non Maximum Suppression Layer* in ihrer Größe reduziert. Als Maß zur Filterung wird die *IoU* der detektierten Bounding Box zur wahren Bounding Box verwendet. Überschreitet diese einen Wert von 0.5, so ist diese der originalen Bounding Box zugeordnet. Demnach ist es auch möglich, dass eine originale Bounding Box mehreren vordefinierten Bounding Boxen zugeordnet werden kann [15].

Während der Trainingsprozesses des *SSD300*<sup>2</sup> mit PASCAL VOC2007 wurde eine Lernrate von  $\eta = 10^{-3}$  für das Mini-Batch Verfahren mit Batchgröße 32 und Moment  $\beta = 0.9$  verwendet. Die Gewichtungen wurden *Xavier* initialisiert. Nach 40.000 Iterationen wurde die Lernrate für 10.000 Iterationen auf  $\eta = 10^{-4}$  reduziert und schließlich auf  $\eta = 10^{-5}$  [15].

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

Abbildung 2.18.: Vergleich SSD [15]

Nach obiger Tabelle 2.18 ist eindeutig festzustellen, dass *SSD300* ein gutes Verhältnis zwischen Präzision und Reaktionsvermögen bewahrt. Durch den Verzicht auf den Schritt der Generierung von Bounding Box Vorschlägen und des *Poolings* kann *SSD* deutlich schneller ablaufen als die Vergleichsdetektoren, während durch das Vordefinieren von Bounding Boxen ebenso eine hohe Präzision erzielt werden kann [15].

<sup>2</sup>SSD300 verwendet Bilder der Auflösung 300x300 Pixel. Alternativ existiert ebenso SSD512 für Bilder der Auflösung 512x512 Pixel. Die Bilder können jedoch auch kleiner als die vorgegebene Auflösung gewählt werden.

Allerdings ergibt sich vor allem für kleine Objekte ein erschwertes Detektionsvermögen, da diese in den höherliegenden Convolutional Layern untergehen. Als Lösung hierfür kann eine erhöhte Inputgröße gewählt werden (vgl. *SSD512*) oder *Data Augmentation* für den Lernprozess angewandt werden. Weitere mögliche Fehler können falsche Positivbefunde aufgrund von fehlerhafter Lokalisation sein, Verwechslung mit ähnlichen Kategorien oder dem Hintergrund sowie weitere [15].

Letztendlich lässt sich *SSD* als schneller Objektdetektor beurteilen, der nicht nur einfach zu trainieren und integrieren ist, sondern ebenso ein gutes Verhältnis zwischen Lokalisationspräzision und Echtzeitvermögen schafft.

## You Only Look Once

Der Algorithmus *You Only Look Once* (YOLO) ist ein weiterer Objekterkennungsalgorithmus und betrachtet statt separaten Bildregionen das komplette Bild. Er benutzt nur ein neuronales Netz, um Bounding Boxen und Wahrscheinlichkeiten für bestimmte Klassen vorherzusagen.

Hierzu wird ein  $S \times S$  Gitter über das Bild gelegt. Für jedes Feld im Gitter werden  $B$  Bounding Boxen erzeugt. Jede Box besitzt neben den zum Gitterfeld relativen Positionswerten einen Wert, der die Vorhersage der jeweiligen Klasse und die Präzision der Box repräsentiert. Dieser Wert wird als *confidence score* bezeichnet und wird durch die Multiplikation der Wahrscheinlichkeit für eine Klasse mit der *IoU*, also die Präzision der berechneten Box im Verhältnis zu der Box aus den vortrainierten Testdaten festgelegt [28].

Aus der Menge an Bounding Boxen werden schließlich mit Hilfe eines festgelegten Schwellwertes die Boxen mit lokalisierten Objekten bestimmt (siehe Abbildung 2.19).

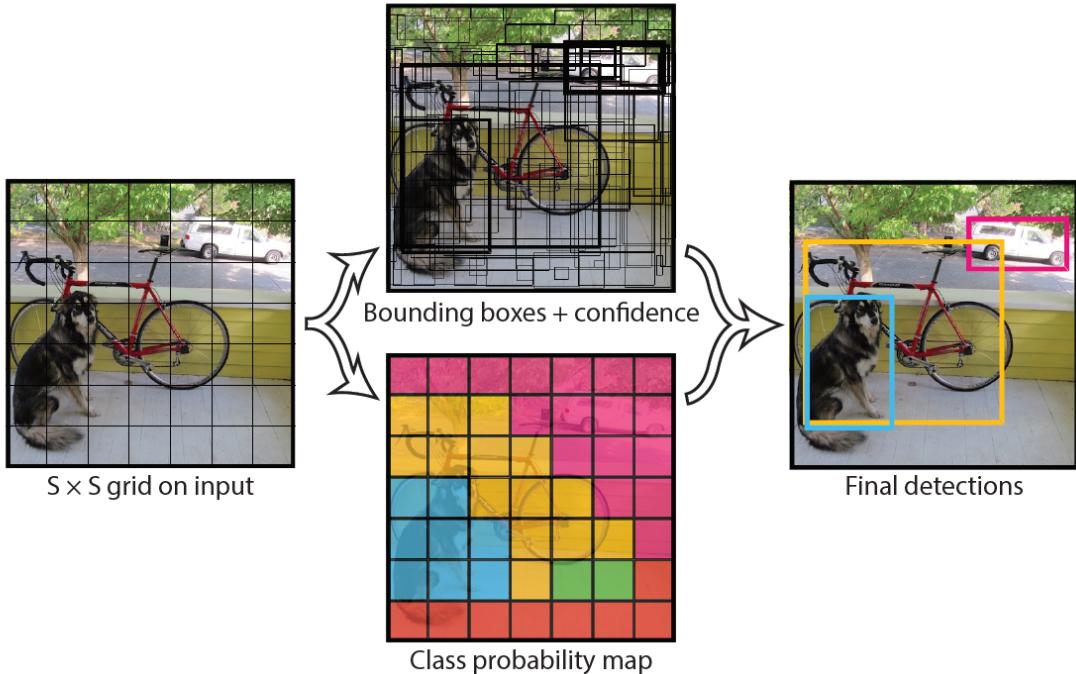


Abbildung 2.19.: Vereinfachte Darstellung der Objekterkennung mit dem YOLO Algorithmus [28]

Die vorhergesagten Werte werden in einem  $S \times S \times (B * 5 + C)$  Tensor kodiert, wobei  $S$  und  $B$  wie zuvor beschrieben durch das Gitter und die Bounding Boxen festgelegt sind und  $C$  die Anzahl der Klassen beschreibt. Abbildung 2.20 zeigt den Aufbau des CNN von *YOLO* für die Detektion. Es besteht aus 24 Convolutional Layern gefolgt von zwei *Fully-Connected* Layern. Für die Genauigkeit bei der Detektion wird die Auflösung des Eingangsbildes verdoppelt [28].

In dem Netzwerk in Abbildung 2.20 wird ein Bild mit einer Auflösung von  $224 \times 224$  Pixeln verwendet und die vorhergesagten Werte im  $7 \times 7 \times 30$  Tensor ausgegeben.

Die mittlerweile dritte und aktuelle Version von *YOLO* weist enorme Verbesserungen auf, gerade im Bezug auf die Erkennung von sehr kleinen Objekten wie zum Beispiel einzelne Vögel in einem Schwarm [29].

TODO Yolo v3: Veränderung neuronales Netz

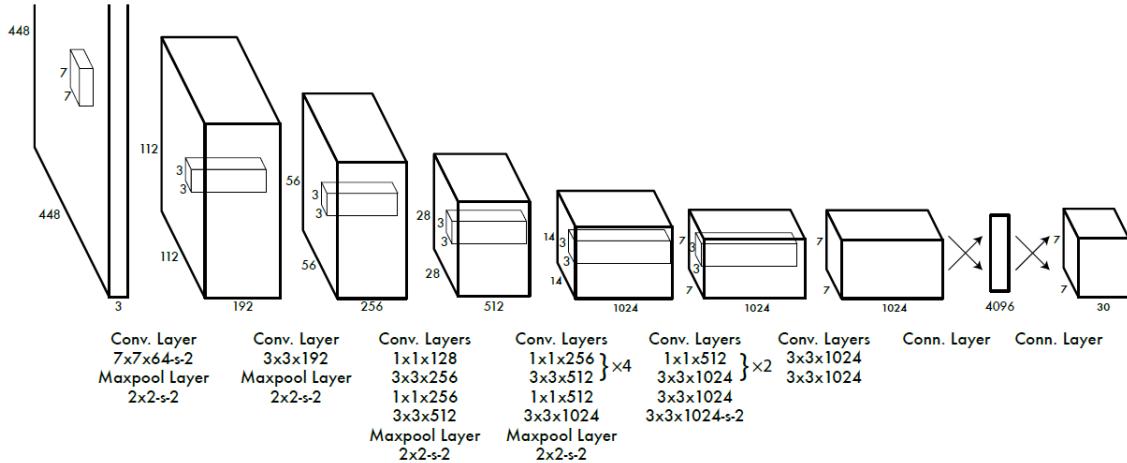


Abbildung 2.20.: YOLO Architektur [28]

## 2.6. Cloud Infrastruktur

Das Trainieren eines *Deep Learning* Modells ist gerade bei großen CNN Architekturen rechenaufwendig. Tensor Operationen wie Matrixmultiplikationen und Konvolutionen erfordern im Rahmen des maschinellen Lernens hohe Parallelisierung und Taktfrequenzen, um in absehbarer Zeit gute Ergebnisse zu liefern. Die Rechenkapazität normaler Desktop-PCs reicht meist nicht aus, um performantes *Deep Learning* betreiben zu können.

Abhilfe bieten Software-as-a-Service (SaaS) bzw. Platform-as-a-Service (PaaS) Angebote wie *Amazon SageMaker*, *Google Cloud Platform Cloud AI* oder *Azure ML Services* oder aber auch Start-ups wie *FloydHub*. Diese bieten Infrastruktur in unterschiedlichen Zonen je nach Standpunkt der Rechenzentren zum Trainieren an sowie eine Plattform zum Verwalten der *Deep Learning* Prozesse.

## Trainingshardware

Gerade GPUs bieten sich aufgrund ihres hohen Parallelisierungsvermögens gegenüber herkömmlichen CPUs an. Insbesondere *NVIDIA* nimmt hierbei eine Vorreiterrolle in der Produktion von Server-GPUs ein. Die *Compute Unified Device Architecture* (CUDA) von *NVIDIA* ermöglicht als Programmiermodell und paralleler Computing Plattform das Auslagern von Rechenprozessen auf GPUs. Das *CUDA Toolkit* beinhaltet GPU beschleu-

nigte Bibliotheken, einen Compiler, Entwicklungswerkzeuge sowie die eigentliche *CUDA* Laufzeit und wird von vielen *Deep Learning* Bibliotheken genutzt, wie z.B. *PyTorch* [30] [31].

Werden gängige Tesla GPUs, die Kunden als PaaS-Angebot zur Verfügung gestellt werden, nach ihrer Rechenleistung verglichen, so ergibt sich Tabelle 2.6 [32].

	K80	P100	T4	V100	GTX 1080	TITAN RTX
CUDA Cores	2496	3584	2560	5120	2560	4608
Tensor Cores	/	/	320	640	/	576
TeraFLOPS (Single Precision)	4,113	9,526	8,141	14,13	8,873	16,31
Memory Bandwidth (in GB/sec)	240,6	732,2	320	897	320,3	672
Suggested Power Supply Unit	700	600	350	600	450	600

Tabelle 2.1.: Vergleich von GPUs nach Rechenleistung

Auch wurden die Desktop-Grafikkarten *GeForce GTX 1080* sowie die *Titan RTX* in den Vergleich mit aufgenommen. Der Vergleich dient später dazu, um eine begründete Kosten-Nutzen Abwägung zwischen einem Trainieren der Modelle auf der Cloud oder lokal vollziehen zu können. Hauptentscheidungskriterien sind hierbei der Grad der möglichen Parallelisierung und die reine Rechenleistung im Verhältnis zum Stromverbrauch.

Neben GPUs existieren seit 2015 die von Google entwickelten *Tensor Processing Units* (TPUs). Diese Art von Spezialhardware erreicht pro TPU-Kern eine Rechenleistung von bis zu 92 TOPS [33]. Werden 2048 solcher TPU-Kerne zu einem TPU-Pod zusammen geschlossen, so ergibt sich eine Rechenleistung von über 100 PetaFLOPS [34]. Zudem ist die größere Rechenleistung gleichzeitig effizienter als herkömmliche GPUs (siehe Abbildung 2.21)

Eine weitere Steigerung versprechen Microsofts Field Programmable Gate Arrays (FPGAs), die allerdings nicht weiter im Rahmen dieser Arbeit betrachtet werden sollen [36].

## FloydHub

FloydHub, ein kalifornisches Start-up, bietet eine Data-Science Plattform zum Trainieren und Bereitstellen von *Deep Learning* Applikationen. FloydHub erlaubt es Anwendern,

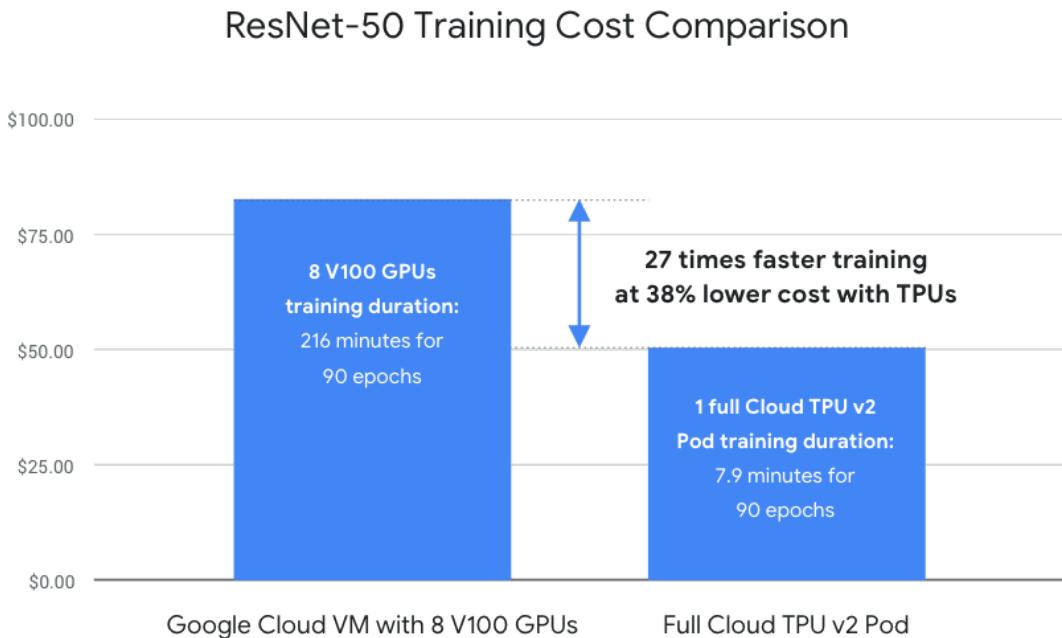


Abbildung 2.21.: Vergleich V100 - TPU Pod [35]

sich auf reines *Deep Learning* zu konzentrieren, während es die Arbeit rund um den *Deep Learning* Lebenszyklus abnimmt. Hierzu gehört das Bereitstellen der entsprechenden Hardware, das Installieren von Treibern oder das Integrieren verschiedener *Deep Learning* Bibliotheken, wie *TensorFlow*, *PyTorch* oder *Keras* [37].

Mit Hilfe des von FloydHub bereitgestellten Command Line Interfaces (CLI) kann ein lokales Projekt zu einem FloydHub Projekt initialisiert werden. Anschließend können anhand einer Konfigurationsdatei Einstellungen über das Training spezifiziert werden. Alternativ können diese auch über das CLI festgelegt werden.

```

1 machine: gpu2
2 env: pytorch-1.4
3 input:
4   - destination: input
5     source: <username>/datasets/smartwarehousessd/3
6     - <username>/datasets/smartwarehousessd/3:ssd
7 description: Job to train the SSD
8 max_runtime: 3600

```

```
9 command: python train.py
```

Listing 2.2: Konfigurationsdatei zum Trainingsjob

Hierbei kann zwischen der K80 (gpu) oder der V100 (gpu2) GPU für das Training gewählt werden. Auch die *Deep Learning* Laufzeitumgebung muss spezifiziert werden. Bei Bedarf auch zusätzliche Bibliotheken in einer *floyd\_requirements.txt*-Datei zur Installation mit angegeben werden. Anschließend muss der Datensatz referenziert werden, mit dem das Modell trainiert werden soll.

Dieser Datensatz wird separat hochgeladen, da sich dieser im Gegensatz zum Programmcode nur selten ändert. FloydHub implementiert auf seiner Plattform eine Art Pfadsystem, unter dem Datensätze und Projekte abgespeichert werden. Diese Pfade werden in der Konfigurations-Datei zur Referenzierung genutzt. Um auch im Programmcode auf den Datensatz zuzugreifen, muss ein Mountname definiert werden. In obigen Beispiel wird dem Datensatz unter Verzeichnis *<username>/datasets/smartwarehousessd/3* der Mountname *ssd* gegeben. Das Verzeichnis zum Einlesen der Daten ist anschließend im Code unter */floyd/input/ssd/* erreichbar.

Mit dem CLI Befehl *floyd run* wird der Programm Code auf die Plattform hochgeladen und der in der Konfigurationsdatei angegebene Befehl ausgeführt. Daraufhin wird ein Job erstellt, versioniert und ausgeführt. Während der Job ausgeführt wird, wird dem Nutzer ein Einblick in die Konsolenausgabe gewährt sowie in Metriken zur Hardwareauslastung. In der Jobhistorie kann im Nachhinein jeder Job mit dem damals aktuellen Programmcode und Datensatz eingesehen werden. Auch Datensätze werden versioniert. Schreibrechte sind auf das Verzeichnis */floyd/home* begrenzt, hier können Zwischenspeicherpunkte des Modells abgelegt werden.

FloydHub betreibt ein zeitbasiertes Zahlungsmodell. So kosten 10 Stunden Laufzeit auf einer K80 12\$, auf einer V100 bereits 42\$. Zusätzlich werden monatliche Account-Gebühren berechnet. Je nach Account kann eine unterschiedliche Anzahl an Projekten erstellt und Speicherplatz verwendet werden. Die *Beginner* Ausstattung von einem Projekt und 10 GB Speicher ist allerdings kostenfrei.

## 3. Konzeption

Um den *SSD* und den *YOLO* Objektdetektor miteinander vergleichbar zu machen und um deren Potential zum industriellen Einsatz zu bewerten, müssen konkrete Bewertungskriterien eingeführt werden. Mit Hilfe dieser Metriken werden die beiden Objektdetektoren anhand eines Benchmark Datensatzes initial verglichen, bevor der eigentliche Datensatz eingeführt wird. Dieser spiegelt Verhältnisse der Echtzeitumgebung wider.

### 3.1. Bewertungskriterien

#### Präzision

Zur Messung der Präzision wird die Metrik *mAP* verwendet. Dies garantiert eine gute Vergleichbarkeit mit den veröffentlichten Leistungsmerkmalen der Objektdetektoren.

#### Reaktionsvermögen

Um eine Verarbeitung in Echtzeit zu ermöglichen muss gewährleistet sein, dass die Interferenz mit dem Modell mit der eingehenden Bildrate einhergeht. Als Maßstab dafür dient die *Frames Per Second* (FPS) Zahl. Echtzeitfähigkeit im Projekt ist so definiert, dass die Bildrate der erzeugten Bilder mit den eingezeichneten Bounding Boxen nach der Interferenz mindestens so hoch sein muss wie die des eingehenden Bildstroms.

#### Trainingsverhalten

Unter dem Punkt Trainingsverhalten wird zusammengefasst, wie schnell sich die einzelnen Modelle mit den unterschiedlichen Objektdetektoren trainieren lassen. Hierbei wird besonderer Fokus darauf gelegt, wie viele Trainingsepochen notwendig sind, bis die Fehlerfunktion des neuronalen Netzes konvergiert. Es soll aber auch betrachtet werden, wie mit doppelt erkannten Objekten während des Trainingsprozesses umgegangen wird.

## Interaktionsverhalten

Im Zuge der Evaluation des Interaktionsverhalten werden drei Kriterien betrachtet. Darunter fällt, wie die Objektdetektoren bei besonderen Beleuchtungsverhältnissen abschneiden. Dies beinhaltet sowohl unterbeleuchtete als auch überbeleuchtete Gegenden. Daneben sollen ebenso extreme Blicklagen ein Faktor sein, um zu evaluieren, wie gut sich die trainierten Objektdetektoren für den industriellen Einsatz eignen. Dies umfasst sowohl unter welcher Entfernung die Objekte betrachtet werden, als auch unter welchen Winkel. Zuletzt muss bewertet werden, wie gut Objekte erkannt werden können, die nicht vollständig zu erkennen sind. Dies ist der Fall, wenn Objekte hintereinander angeordnet sind oder durch andere Beilagen verdeckt sind.

### 3.2. Vergleich der Objektdetektoren

- Synthetischen Datensatz einführen (Benchmarkdatensatz)
- Effekte/Probleme aufzeigen
- Bewertungskriterien bestärken
- Problemlösungsverfahren aufzeigen (z.B. Anpassung von Hyperparametern)
- Variantendiskussion (Welches Verfahren eignet sich hinsichtlich Bewertungskriterien am besten)
- Keine Bewertung der Detektoren!

Probleme:

- Mehrfach-Detektionen werden alle als positiv angesehen

### 3.3. Echtzeitumgebung

Das *Smart Warehouse* lehnt sich an ein großes Warenhaus an, bei dem Produkte nicht in Kartons verpackt, sondern als ganzes auf Regalen angeordnet sind, ähnlich wie bei *Baumarkt* oder *Selgros*. Im Rahmen des Projektes wurde sich exemplarisch auf Flaschen konzentriert, dabei wurden neun Kategorien bestimmt (siehe Abbildung 3.1).



Abbildung 3.1.: Die neun Kategorien

Der Datensatz besteht aus 1000 annotierten Bildern. Die Bilder besitzen eine Auflösung von 2112x4608 Pixels mit einer Farbtiefe von 24 Bit. Alle neuen Kategorien sind nahezu gleich häufig im Datensatz vorhanden.

Im initialen Datensatz sind auf 75% der Bilder die Objekte der jeweiligen Kategorien einzeln und klar erkennbar abgebildet. Hierdurch wird erhofft, dass Modell zunächst auf die Muster der jeweiligen Objekte zu trainieren. In 12,5% der Bilder sind die Objekte der jeweiligen Kategorien ebenso einzeln, allerdings mit unterschiedlichen Hintergründen, Beleuchtungsverhältnissen, Blickwinkeln und Entfernung abgebildet. Je nach Umgebung wurden Bilder dieses Anteils als schwer erkennbar markiert. Um das Warenhaus

zu simulieren, sind in den letzten 12,5% der Bilder die Objekte auf Regalen angeordnet, jeweils hintereinander oder in Getränkekästen (siehe Abbildung 3.2).

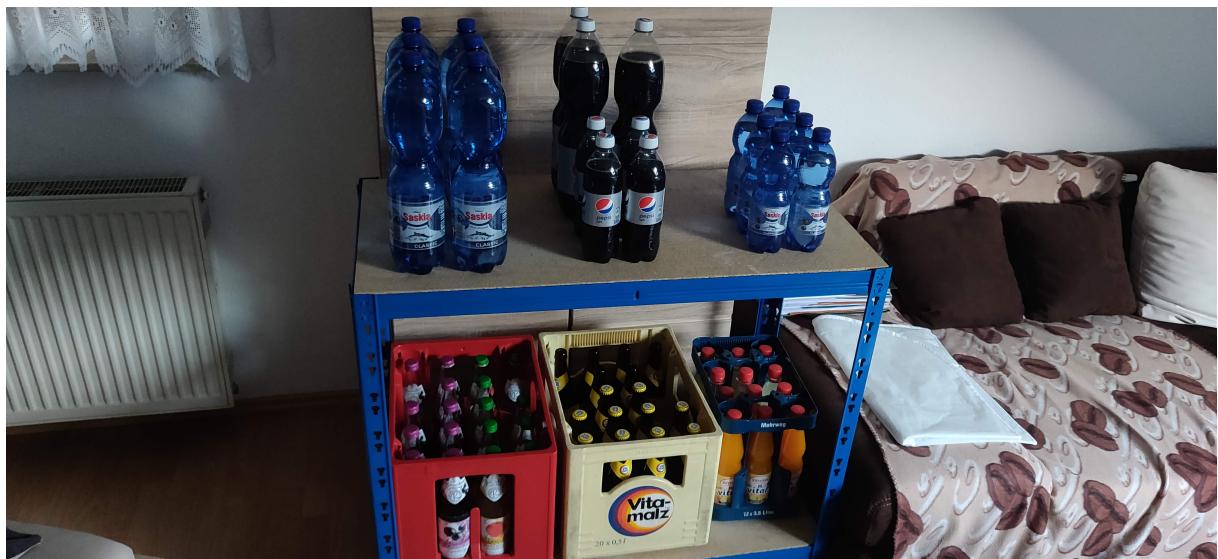


Abbildung 3.2.: SmartWarehouse Regal

# 4. Realisierung

## 4.1. Trainieren der Objektdetektoren

- Auswahl Cloud oder lokales Trainieren?
  - Amazon, Azure, etc. zu teuer
  - GCP erlaubt keine Quota
  - FloydHub ist zum Training ungeeignet (Unterstützt kein Darknet und auf die Dauer zu teuer)
  - Höchstens V100, aber den Aufwand nicht wert (Kosten/Nutzen Abwägung aus Zeit- und Geldgründen)
- Datensatz: Aus Zeitgründen nicht größer
- LabelImg
- Bilder mit Labeln in der Arbeit
- Was liefert die Arbeit, was bisher noch nicht bekannt war (Mehrwert, auch wenn etwas nicht geht)?
- Soll beweisen, dass etwas entstanden ist (Krassen Eindruck vermitteln)
- Keine Ergebnisse evaluieren
- conv -> Feature Vektor für fully connected layer classification
- down sampling -> pool layer
- komplett durchlesen (dopplungen, alles zur richtigen zeit eingeführt)
- Warum nur die drei detektoren, warum keine segmentierung?
- Auswahl detektor, low fps rcnn

## **4.2. Drohnen Anbindung**

## **4.3. Dashboard Entwicklung**

## 5. Ergebnisse

- Ergebnisse auswerten

## 6. Bewertung

- Ergebnisse bewerten und diskutieren
- Was liefert die Arbeit, was bisher noch nicht bekannt war (Mehrwert, auch wenn etwas nicht geht)?

## 7. Zusammenfassung und Ausblick

- Klare Darstellung, was die Arbeit geliefert hat
- Ca. 2-4 Anpunkte: Zukünftige Ziele

# Literaturverzeichnis

- [1] Jonathan Hui. *mAP (mean Average Precision) for Object Detection.* Medium, 2018. URL: [https://medium.com/@jonathan\\_hui/map-mean-average-precision-for-object-detection-45c121a31173](https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173) (Einsichtnahme: 15.02.2020).
- [2] Aurélien Géron. *Machine Learning mit Scikit-Learn & TensorFlow: Konzepte, Tools und Techniken für intelligente Systeme: Übersetzung von Kristian Rother.* 1. Aufl. Heidelberg: dpunkt.verlag GmbH, 2018.
- [3] MathWorks. *Deep Learning: Drei Dinge, die Sie wissen sollten.* MathWorks, 2019. URL: <https://de.mathworks.com/discovery/deep-learning.html> (Einsichtnahme: 12.10.2019).
- [4] dokts. innovation. *inventAIRyX.* dokts. innovation Homepage, 2019. URL: <https://www.doks-innovation.com/inventory-x/> (Einsichtnahme: 26.10.2019).
- [5] Ravindra Parmar. *Detection and Segmentation through ConvNets.* Towards Data Science, 2018. URL: <https://towardsdatascience.com/detection-and-segmentation-through-convnets-47aa42de27ea> (Einsichtnahme: 07.03.2020).
- [6] Priya Dwivedi. *Semantic Segmentation — Popular Architectures.* Towards Data Science, 2019. URL: <https://towardsdatascience.com/semantic-segmentation-popular-architectures-dff0a75f39d0> (Einsichtnahme: 07.03.2020).
- [7] Philippe Lucidarme. *Simplest perceptron update rules demonstration.* Homepage Blog, 2017. URL: <https://www.lucidarme.me/simplest-perceptron-update-rules-demonstration/> (Einsichtnahme: 26.10.2019).
- [8] Wikipedia. *Deep Learning.* Wikipedia, 2019. URL: [https://de.wikipedia.org/wiki/Deep\\_Learning](https://de.wikipedia.org/wiki/Deep_Learning) (Einsichtnahme: 27.01.2019).
- [9] David E. Rumelhart / Geoffrey E. Hinton / Ronald J. Williams. *Learning Internal Representations by Error Propagation.* Hrsg. von University of California, San Diego. 09/1985. URL: <https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf> (Einsichtnahme: 26.10.2019).

- [10] Xavier Glorot, Y. B. „Understanding the difficulty of training deep feedforward neural networks“. Diss. Montréal: Universite de Montréal, 2010. URL: <http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf> (Einsichtnahme: 26. 10. 2019).
- [11] Imad Dabbura. *Gradient Descent Algorithm and Its Variants*. Towards Data Science, 2017. URL: <https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3> (Einsichtnahme: 26. 10. 2019).
- [12] Danqing Liu. *A Practical Guide to ReLU*. Medium, 2017. URL: <https://medium.com/@danieling/a-practical-guide-to-relu-b83ca804f1f7> (Einsichtnahme: 26. 10. 2019).
- [13] Renu Khandelwal. *COCO and Pascal VOC data format for Object detection*. Towards Data Science, 2019. URL: <https://towardsdatascience.com/coco-data-format-for-object-detection-a4c5eaf518c5> (Einsichtnahme: 02. 02. 2020).
- [14] Arun Ponnusamy. *Preparing Custom Dataset for Training YOLO Object Detector*. Arun Ponnusamy Homepage, 2019. URL: <https://www.arunponnusamy.com/preparing-custom-dataset-for-training-yolo-object-detector.html> (Einsichtnahme: 02. 02. 2020).
- [15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Chang-Yang Fu, Alexander C. Berg. „SSD: Single Shot MultiBox Detector“. Diss. 2016-12-29. URL: <https://arxiv.org/pdf/1512.02325.pdf> (Einsichtnahme: 02. 11. 2019).
- [16] Mark Everingham, J. W. *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Development Kit*. 2007. URL: <http://host.robots.ox.ac.uk/pascal/VOC/voc2007/html/doc/voc.html#SECTION00030000000000000000> (Einsichtnahme: 08. 02. 2020).
- [17] Mark Everingham, J. W. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit*. 2012. URL: [http://host.robots.ox.ac.uk/pascal/VOC/voc2012/html/doc/devkit\\_doc.html#SECTION00030000000000000000](http://host.robots.ox.ac.uk/pascal/VOC/voc2012/html/doc/devkit_doc.html#SECTION00030000000000000000) (Einsichtnahme: 08. 02. 2020).
- [18] Daphne Cornelisse. *An intuitive guide to Convolutional Neural Networks*. freeCodeCamp Homepage, 2018. URL: <https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050> (Einsichtnahme: 26. 10. 2019).

- [19] Abhineet Saxena. *Convolutional Neural Networks (CNNs): An Illustrated Explanation*. XRDS Crossroads - The ACM Magazine for Students, 2016. URL: <https://blog.xrds.acm.org/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/> (Einsichtnahme: 26. 10. 2019).
- [20] Leonadro Araujo Santos. *Pooling Layer: Introduction*. GitBook, 2018. URL: [https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/pooling\\_layer.html](https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/pooling_layer.html) (Einsichtnahme: 26. 10. 2019).
- [21] Tarang Shah. *Measuring Object Detection models - mAP - What is Mean Average Precision?* Tarang Shabs Blog, 2018. URL: <https://tarangshah.com/blog/2018-01-27/what-is-map-understanding-the-statistic-of-choice-for-comparing-object-detection-models/> (Einsichtnahme: 15. 02. 2020).
- [22] Adrian Rosebrock. *Intersection over Union (IoU) for object detection*. pyimagesearch, 2016. URL: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/> (Einsichtnahme: 02. 11. 2019).
- [23] Rohith Gandhi. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. Towards Data Science, 2018. URL: <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e> (Einsichtnahme: 08. 03. 2020).
- [24] Umer Farooq. *From R-CNN to Mask R-CNN*. Medium, 2018. URL: [https://medium.com/@umerfarooq\\_26378/from-r-cnn-to-mask-r-cnn-d6367b196cf](https://medium.com/@umerfarooq_26378/from-r-cnn-to-mask-r-cnn-d6367b196cf) (Einsichtnahme: 08. 03. 2020).
- [25] Dhruv Parthasarathy. *A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN*. Medium, 2017. URL: <https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4> (Einsichtnahme: 08. 03. 2020).
- [26] Andrew Ng. *Anchor Boxes*. Coursera, 2019. URL: <https://www.coursera.org/lecture/convolutional-neural-networks/anchor-boxes-yNwO0> (Einsichtnahme: 02. 11. 2019).
- [27] MathWorks. *vgg16*. MathWorks Homepage, 2019. URL: <https://de.mathworks.com/help/deeplearning/ref/vgg16.html;jsessionid=bf0fea41a0c7700184672711881f> (Einsichtnahme: 02. 11. 2019).

- [28] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, Ali Farhadi. „You Only Look Once: Unified, Real-Time Object Detection“. Diss. 2019. URL: <https://arxiv.org/pdf/1506.02640.pdf> (Einsichtnahme: 10.11.2019).
- [29] Joseph Redmon, A. F. „YOLOv3: An Incremental Improvement“. In: (2018). URL: <https://pjreddie.com/media/files/papers/YOLOv3.pdf> (Einsichtnahme: 04.02.2020).
- [30] NVIDIA. *TRAIN MODELS FASTER*. NVIDIA Homepage, 2020. URL: <https://developer.nvidia.com/cuda-zone> (Einsichtnahme: 09.02.2020).
- [31] PyTorch. *GET STARTED*. PyTorch Homepage, 2020. URL: <https://pytorch.org/get-started/locally/> (Einsichtnahme: 09.02.2020).
- [32] TechPowerUp. *GPU Specs Database*. TechPowerUp Homepage, 2020. URL: <https://www.techpowerup.com/gpu-specs/> (Einsichtnahme: 09.02.2020).
- [33] Harald Bögeholz. *Künstliche Intelligenz: Architektur und Performance von Googles KI-Chip TPU*. Heise Online, 2017. URL: <https://www.heise.de/newsticker/meldung/Kuenstliche-Intelligenz-Architektur-und-Performance-von-Googles-KI-Chip-TPU-3676312.html> (Einsichtnahme: 09.02.2020).
- [34] Google Cloud. *Systemarchitektur*. Google Cloud Dokumentation, 2020. URL: <https://cloud.google.com/tpu/docs/system-architecture> (Einsichtnahme: 09.02.2020).
- [35] Google Cloud. *Cloud TPU*. Google Cloud Homepage, 2020. URL: <https://cloud.google.com/tpu/?hl=de> (Einsichtnahme: 09.02.2020).
- [36] Karl Freund. *Microsoft: FPGA Wins Versus Google TPUs For AI*. Forbes, 2017. URL: <https://www.forbes.com/sites/moorinsights/2017/08/28/microsoft-fpga-wins-versus-google-tpus-for-ai/#781e2bf53904> (Einsichtnahme: 09.02.2020).
- [37] FloydHub. *Home*. FloydHub Documentation, 2020. URL: <https://docs.floydhub.com/> (Einsichtnahme: 15.02.2020).

# **A. Anhang**

## **Abbildungen**