**[*Intelligente Parkplatzerkennung mit künstlichen neuronalen Netzwerken*]**

**Systembeschreibung**

[1. Ziel und Zweck des Dokumentes 4](#_Toc11437659)

[1.1. Projektbeschreibung 4](#_Toc11437660)

[1.1.1. Kurzbeschreibung des Projekts 4](#_Toc11437661)

[1.1.2. Zweck des Projekts 5](#_Toc11437662)

[1.1.3. Hintergrund, Problemstellung, Motivation für das Projekt 5](#_Toc11437663)

[1.1.4. Ziele des Projekts 5](#_Toc11437664)

[1.1.5. Erfolgskriterien 5](#_Toc11437665)

[2. Systemübersicht 6](#_Toc11437666)

[2.1. CNN 6](#_Toc11437667)

[2.2. Skripte 8](#_Toc11437668)

[2.2.1 cutImages.py 8](#_Toc11437669)

[2.2.2. Copy\_pictures.py 9](#_Toc11437674)

[2.2.3. Picture\_random\_sort.py 10](#_Toc11437675)

[2.2.4. picturesize.py 10](#_Toc11437676)

[3. Architektur und Designentscheide 11](#_Toc11437677)

[3.1. Daten (Mengengerüst & Strukturen) Felix 14](#_Toc11437678)

[4. Umgebungs-Anforderungen 17](#_Toc11437679)

[4.1. Technologie-Voraussetzungen 17](#_Toc11437680)

[4.2. Kooperierende Anwendungen und COTS-Komponenten 17](#_Toc11437681)

[5. Testplan 18](#_Toc11437682)

[5.1. Testverfahren 18](#_Toc11437683)

[5.2. Standardmodel 18](#_Toc11437684)

[5.3. Felix 19](#_Toc11437685)

[5.3.1. Rotation\_range 20](#_Toc11437686)

[5.3.2. width\_shift\_range 23](#_Toc11437687)

[5.3.3. shear\_range 26](#_Toc11437688)

[5.3.4. Flip Image 28](#_Toc11437689)

[5.4. Frede 31](#_Toc11437690)

[5.5. Pit 31](#_Toc11437691)

[5.6. Jascha 31](#_Toc11437692)

[6. Ausblick 32](#_Toc11437693)

[7. Abbildungsverzeichnis 33](#_Toc11437694)

[8. Tabellenverzeichnis 33](#_Toc11437695)

[9. Projektabschluss 34](#_Toc11437696)

Versionen:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rev. | Datum | Autor | Bemerkungen | Status |
| 0.1 | 14.03.2019 | Felix Willrich | 1. Entwurf + Eintragen aller Informationen | Abgeschlossen |
| 0.2 | 08.06.2019 | Felix Willrich | Kapitel 1,3.2, 5 angefangen | Abgeschlossen |
| 0.3 | 09.06.2019 | Frederik Rieß | Erste Verbesserungen durchgeführt  Kapitel 2.1 fertiggestellt  Kapitel 2.2 begonnen (cutImages.py) | Abgeschlossen |
| 0.4 | 10.06.2019 | Felix Willrich | Kapitel 2 Skripte beschrieben | Abgeschlossen |
| 0.5 | 14.06.2019 | Felix Willrich | Kapitel 5 geschrieben + Tabellenverzeichnis + Abbildungsverzeichnis + Allgemeine Formatierung | Abgeschlossen |

1. Ziel und Zweck des Dokumentes

Dieses Dokument beschreibt die Anforderungen der T-Systems on site services GmbH. Es handelt sich hierbei um die Systemdefinition, die der Auftragnehmer für den Auftraggeber (Kunde) erstellt, sodass der Kunde versteht und validieren kann, was das System leisten wird.

## Projektbeschreibung

Dieses Projekt wird im Rahmen des Modules „Teamprojekt“ durchgeführt. welches von Herrn Kircher und Frau Schiering doziert wird. Kunde für dieses Projekt ist Herr Philip May, welcher Angestellter bei der T-Systems on site GmbH ist und gleichzeitig die Rolle des Projektansprechpartners einnimmt.

### Kurzbeschreibung des Projekts

Das Projekt folgt einem gewissen Ablauf. Die Daten werden eingelesen, verarbeitet und ausgegeben. Diese drei Schritte werden anhand von folgendem Bild verständlich.

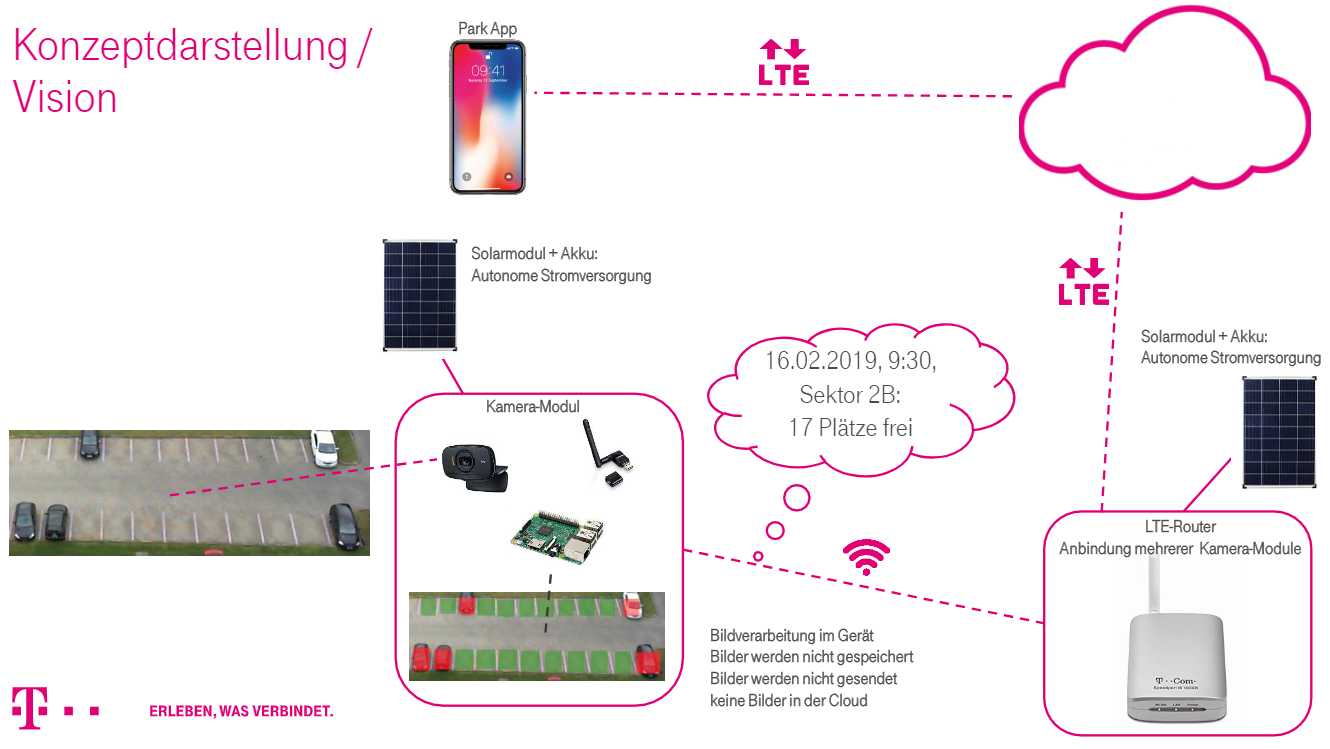


Abbildung 1: Produkt-Zyklus

Eine Kamera überträgt ein regelmäßig aufgezeichnetes Bild von einem Parkplatz an einen Mikrocontroller. Dieser verarbeitet die Daten, indem er das Bild in das konzipierte neuronale Netzwerk schickt. Anhand der Aufnahme soll bestimmt werden, wie viele Parkplätze frei sind. Diese Information wird an eine zentrale Stelle geschickt und weiterverteilt an eine Smartphone-App, damit die Nutzer zu jeder Zeit abrufen können, wohin sie fahren sollten. Die Bilder werden nicht vom Mikrocontroller weitergeschickt.

Da aufgrund der Zeit Abstriche gemacht werden müssen, werden wir uns in diesem Projekt auf die Erkennung von freien Parkplätzen konzentrieren. Das bedeutet, dass wir keine Live-Daten aus der Kamera bekommen werden, bzw. auch keine App erstellen werden, da dieses den Rahmen des Teamprojektes sprengen würde.

### Zweck des Projekts

Das Projekt soll in erster Instanz zur Erkennung von freien Park-Flächen eingesetzt werden und im Rahmen von Parkplätzen und Parkhäusern genutzt werden. Die Technik kann mit verschiedenen Inputdaten auf verschiedenste Felder ausgeweitet werden. Beispiele wären, Lagerbestände oder den Füllstand von verschiedenen Containern erkennen.

Während des Projekts ist deutlich geworden, dass die Anforderungen geändert werden müssen. Dies lag vornehmlich an dem Zeitaufwand bzw. an der Kommunikation mit dem Kunden. Es wurden verschiedene Annahmen getroffen und zum Sprint 2 hat sich herauskristallisiert, dass dieses Produkt vornehmlich zum Erkenntnisgewinn für den Kunden und uns gelten sollen. Damit keine Unklarheiten aufkommen, wurden die Ziele aus dem ersten Gespräch mit dem Kunden formuliert. Im weiteren Verlauf der Dokumentation werden die Testergebnisse und Erkenntnisse niedergeschrieben.

### Hintergrund, Problemstellung, Motivation für das Projekt

Die T-Systems on site services GmbH in Person von Philip May benutzt im produktiven Sektor verschiedene Machine Learning/Deep Learning Applikationen und möchten durch dieses Produkt in weitere Felder stoßen bzw. weitere Erkenntnisse darüber gewinnen.

Für die Gruppe ergibt sich aufgrund von wenig Vorkenntnissen folgende Probleme:

* Neues Umfeld kennen lernen
* Geeignete Tools und Umgebung finden
* Datenbeschaffung zum Anlernen
* Prototypen erschaffen
* Genaue Erkennung implementieren
* Testumgebung

Die meisten Probleme werden oder wurden mit unserem Ansprechpartner besprochen und

teilweise aufgearbeitet.

Die Motivation zu diesem Projekt ergibt sich aus dem ersten Stichpunkt der Probleme. Die Gruppe möchte in ein neues, aufstrebendes und sehr interessantes Thema einsteigen und dabei gleichzeitig Praxiserfahrung sammeln.

### Ziele des Projekts

Das Projekt wird zuerst bis zu dem Schritt entwickelt, bis das Netz angelernt ist

und verschiedene Parkplatzsituationen erkannt werden. Eine Genauigkeit von 99% wird

angestrebt.

### Erfolgskriterien

Sollte die gewünschte Genauigkeit erlangt worden sein, wird das Projekt als Erfolg bezeichnet. Weiterhin hinzukommen würden verschiedene Umgebungen, wie Schnee, Regen und andere Hindernisse wie Baulöcher oder Belegung von zwei Parkplätzen gleichzeitig. Sollten diese zusätzlichen Kriterien erfüllt werden, wird das Projekt in vollem Umfang als Erfolg gewertet. Es wird eine möglichst genaue Erkennung mit allen unterschiedlichen Faktoren angestrebt.

2. Systemübersicht

In diesem Kapitel wird zuerst das Grundkonzepts eines CNNs beschrieben. Gleichzeitig werden hier die Skripte beschrieben, die für unser Projekt essential waren. Unser eigenes Netz wird unter dem Punkt 3 erklärt. Das ganze Projekt ist auf Github hochgeladen. <https://github.com/Scantraxx123/Parkplatzerkennung>

## 2.1. CNN

Convolutional Neural Networks haben sich bei vielen Aufgaben bewiesen, die der Bilderkennung dienen. Diese Netzwerke bestehen aus mehreren Convolutional und Max Pooling Layern und erkennen bestimmte Pattern (Ecken, Linien etc.) in Bildern. Je tiefer das CNN entwickelt ist, desto genauer kann das Netz bestimmte Objekte erkennen (in unserem Fall Parkplätze mit/ohne Autos).

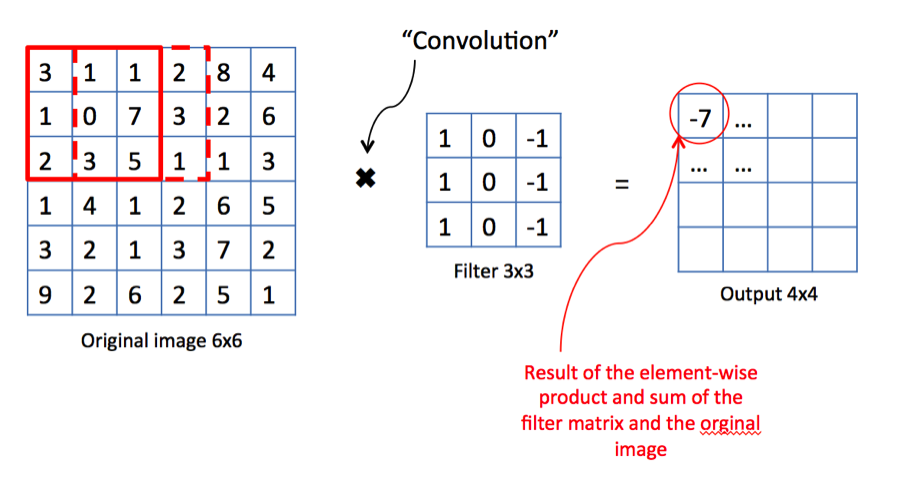
Jedes Convolutional Layer hat eine gewisse Anzahl an Filtern, die diese Pattern erkennen. Zunächst werden die zu trainierenden Bilder mit einer festgelegten Größe und einer Anzahl an Channel eingelesen. Dabei kann durch die Channel angegeben werden, dass das Model zum Beispiel mit RGB (3 Channel) oder Graustufen (1 Channel) trainiert wird. Das erste Convolutional Layer liest die Bilder also ein und lässt anschließend die verschiedenen Filter über jedes Bild laufen. Ein Filter kann dabei als kleine Matrix gesehen werden, die eine bestimmte Anzahl an Zeilen und Spalten hat (hier haben sich häufig Filter einer Dimension 3x3 bewiesen). Die Werte in dieser Matrix werden mit zufälligen Werten initialisiert. Der Filter „gleitet“ dann über jeden Block von 3x3 Pixeln des zu trainierenden Bildes. Aus dem Filter und dem Ausschnitt des Bildes wird dann das Punktprodukt gebildet und an eine bestimmte Stelle der Output-Matrix geschrieben. Dabei ist die Output-Matrix so groß, wie es Möglichkeiten für den Filter gibt, über das Bild zu gleiten. Die Tiefe dieser Matrix wird durch die Anzahl der Channel bestimmt. Die Abbildung verdeutlicht diesen Vorgang. 

Abbildung 2: Berechnung CNN

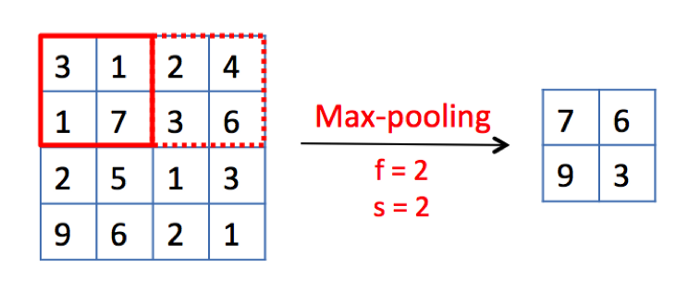
Anschließend wird das sogenannte Max Pooling angewandt. Dabei wird zunächst die Größe des Filters festgelegt, der über die vorherige Matrix laufen soll (häufig eine 2x2-Matrix). Zudem wird die Schrittweite festgelegt, die der Filter über das Bild laufen soll. Wenn nun der Filter über jeden Ausschnitt (2x2) des eingelesenen Bildes läuft, wird von diesen 4 Pixeln der höchste Wert ermittelt und in eine neue Matrix geschrieben. Die folgende Abbildung zeigt diesen Schritt. f steht dabei für die Größe des Filters und s für die Schrittweite.

Abbildung 3: Matrix-Berechnung

Wie zu sehen ist, wird durch diese Operation die Größe der ursprünglichen Matrix enorm reduziert. Im Gegensatz dazu wird die Anzahl der Filter immer höher.

Nach weiteren Convolutional und Max Pooling Layern wird ein sogenanntes Flatten genutzt, um die Matrix „abzuflachen“. Dies bedeutet einfach nur, dass die letzte Matrix in lediglich eine Spalte umgewandelt wird, um die Daten im Netz weiter zu verarbeiten. Anschließend folgen noch eine verschiedene Anzahl an Fully Connected Layern unterschiedlicher Größe und eine Funktion, wie die Daten am Ende auszugeben sind (in unserem Fall binär). Die Abbildung zeigt nochmals den kompletten Vorgang des CNNs mit einer abschließenden Softmax-Funktion, die für einen möglichen Output von mehr als zwei Kategorien sinnvoll zu verwenden ist.

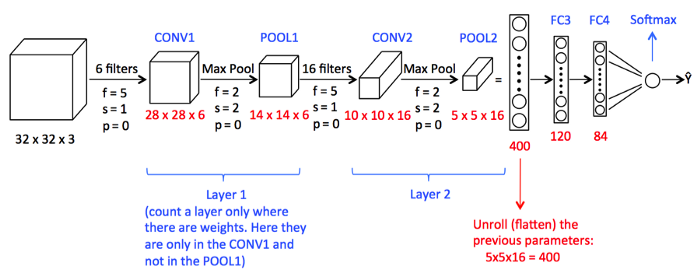


Abbildung 4: Layer-Berechnung

## 2.2. Skripte

In diesem Abschnitt werden alle Skripte beschrieben, die in unserem Workflow Verwendung gefunden haben.

### cutImages.py

Dieses Skript wird dazu verwendet, um aus einem aufgenommenen großen Parkplatz und der dazugehörigen XML-Datei die einzelnen Parkplätze auszuschneiden. Dies geschieht mit den Python-Bibliotheken os, cv2 und xml.etree.ElementTree.

Das Einlesen der Bilder und der XML-Dateien geschieht mit der Bibliothek os. Nach dem Einlesen des Bildes wird unter demselben Namen die XML-Datei gesucht. Aus dieser werden wiederum die Koordinaten der Parkplätze sowie der Tag «Occupied» mit der Bibliothek xml.etree.ElementTree ausgelesen. Durch die Koordinaten wird anschließend mithilfe der Bibliothek cv2 die größtmögliche rechteckige Kontur innerhalb der vier Pixel gewählt, da letztere nicht ganz rechteckig sind und leichte Verschiebungen aufweisen. Diese einzelnen Rechtecke werden dann ausgeschnitten und unter einem passenden Namen in einen Ordner kopiert. Der exakte Ort des Bildes hängt dabei davon ab, ob der Tag «Occupied» des Parkplatzes dabei 0 oder 1 war. Dementsprechend werden die einzelnen Bilder auf die Ordner «Empty» und «Occupied» kopiert. Zur Veranschaulichung der Daten werden auch die einzelnen Parkplätze auf dem großen Bild markiert. Dieses wird anschließend in denselben Ordner neben «Empty» und «Occupied» gelegt. Ein belegter Parkplatz wird dabei rot umrandet, während ein grüner Parkplatz entsprechend freie Parkplätze zeigt.

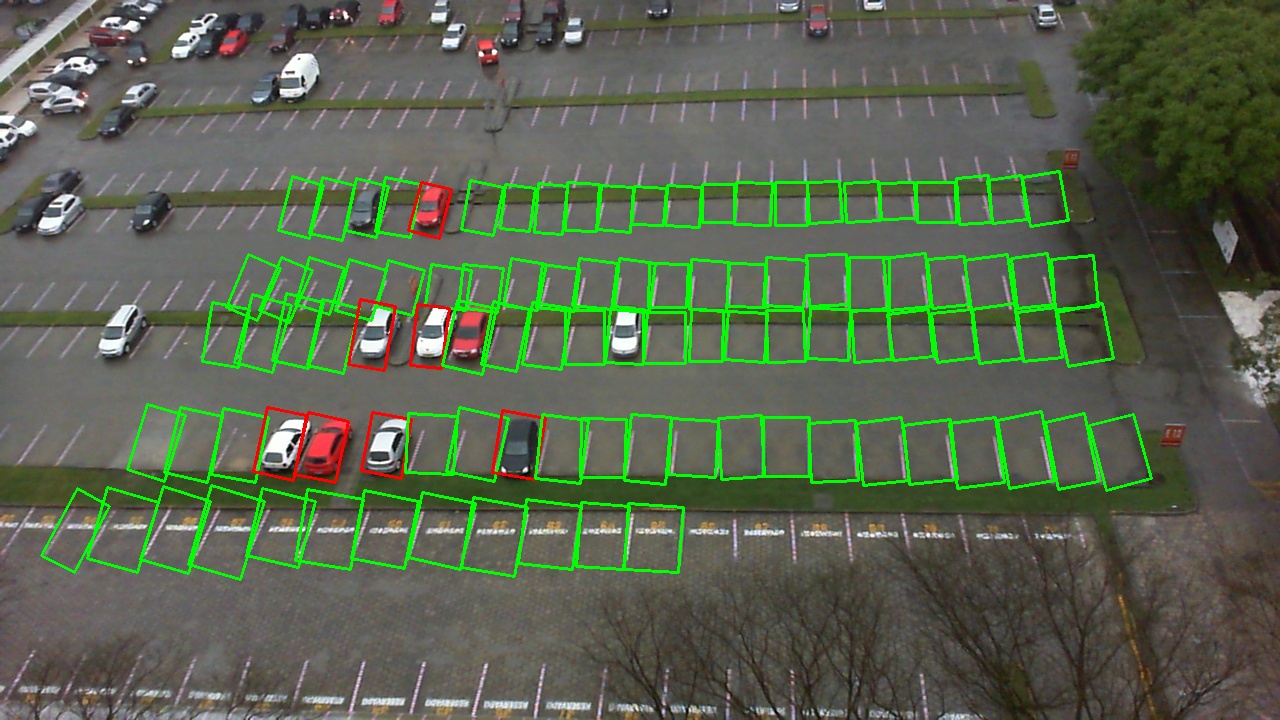


Abbildung 5:Datenfehler

Dabei ist aufgefallen, dass nicht alle Daten in der XML-Datei korrekt sind. Wie auf dem Bild zu sehen, haben einige Parkplätze falsche Einträge unter dem Tag «Occupied» und werden so falsch markiert. Da dies jedoch nicht bei allen Dateien auftritt ist der Einfluss zwar da, aber fällt bei der Masse an Daten kaum ins Gewicht.

Das Skript an sich ist für unser Projekt letzten Endes nicht von erheblicher Bedeutung gewesen, da es bei uns nur um die tatsächliche Erkenntnisgewinnung der zu trainierenden Daten ging und die Daten (Parkplätze) selbst schon einzeln ausgeschnitten waren. Allerdings kann es gut für zukünftige Arbeiten benutzt werden, wenn ein Bild mit einer XML-Datei vorliegt, von denen noch nicht die einzelnen Parkplätze ausgeschnitten wurden.



### Copy\_pictures.py

Damit jedes Bild von den unterschiedlichen Datensätzen in unserer festgelegten Struktur vorliegt, wurden zwei Skripte geschrieben, die uns dabei unterstützen. Copy\_pictures.py und CNR\_COPY.py. Beide Skripte funktionieren mit dem gleichen Mechanismus.

Es gibt einen Quellordner und zwei Zielordner (Empty,Occupied). Danach wird der Quellordner mit all seinen Unterordnen nach Bildern durchsucht. Im Fall von copy\_pictures.py, welches für das Parkinglot-Set genutzt worden ist, gibt es die Unterscheidung, ob das Bild aus dem Ordner „empty“ oder „occupied“ stammt. In dieser Fallunterscheidung wird berücksichtig in welchem Ordner die Bilder gespeichert werden müssen.

Bei dem Skript CNR\_COPY.py, welches für das CNR-Set benutzt worden ist, wird die .txt Datei mit den Pfaden und dem jeweiligen Status (Empty,Occupied) noch eingelesen. Dies hilft, wie bei dem Skript zuvor bei der Fallunterscheidung und gleichzeitig dem Einsortieren.

Bei beiden Skripten wurde die Bibliothek „shutil“ benutzt. Diese besitzt Kopierfunktionen die Python direkt ab Werk mitliefert.

Da uns nach dem ersten Durchlaufen der Skripte aufgefallen ist, dass ein Kopiervorgang von fast 700.000 Bildern bzw. 150.000 Bildern mehrere Stunden dauern kann, haben wir uns nach Alternativen umgesehen. Wir sind in der Recherche auf die Libary „pyfastcopy“( <https://pypi.org/project/pyfastcopy/>) gestoßen. Diese wird über den Packetmanager von Python(pip) installiert und ersetzt die alten shutil-Funktionen. Nach einem weiteren Durchlauf wurde der Kopiervorgang merklich beschleunigt.

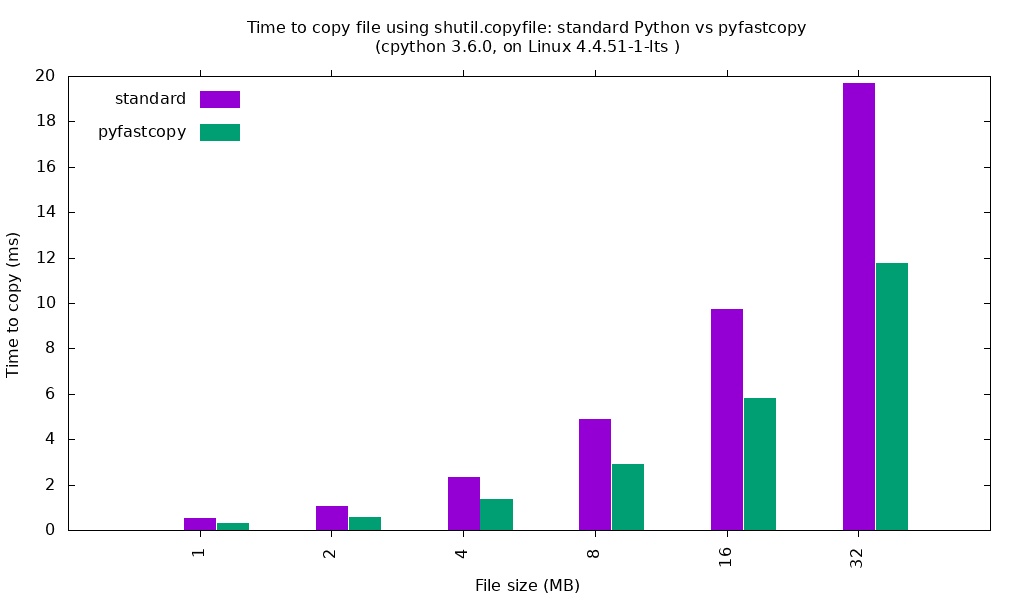


Abbildung 6: Vergleich pyfastcopy zu Standart

### Picture\_random\_sort.py

Einige Bildpakete setzten voraus, dass alle Bilder zufällig einsortiert werden, damit das Netz keine falschen Muster lernt. Um dies zu garantieren, wurde ein Skript geschrieben, welches alle Bilder zufällig einsortiert in die jeweiligen Ordner.

Die Basis des Skripts funktioniert wie das copy\_pictures.py Skript. Es werden Verzeichnisse durchsucht und die jeweiligen Bilder in die Ordner einsortiert. Der einzige Unterschied liegt im Zufälligkeitsmechanismus. Beim Durchsuchen der Ordner werden alle Pfade in eine Liste geschrieben. Nachdem diese Liste befüllt ist, wird mit der Funktion „shuffle“ die Liste verdreht. Dies ermöglicht beim Kopieren der Bilder eine zufällige Einsortierung.

### picturesize.py

Einer der Tests, die von uns in Sprint 2 und 3 durchgeführt worden sind, versucht herauszufinden, welche Größe von Bildern die Beste für das CNN ist. Damit das jeweilige Teammitglied einen Anhaltspunkt hat, welche Größen geeignet sein könnten, wurde ein Skript angefertigt, welches alle Größen ausliest und in eine Datei schreibt.

Die Libary „PIL“ besitzt diverse Funktionen, um Bilder auszulesen oder zu bearbeiten. Auch in diesem Skript werden alle Unterordner auf Bilder durchsucht. Sobald ein Bild gefunden worden ist, wird dies mit der Libary geöffnet und die Höhe und Weite der Bilder ein Dictionary geschrieben. Der Key ist die Höhe und Weite. Der Wert zu dem jeweiligen Key besteht aus einer Zahl, die aussagt wie häufig diese Bildgröße vorkommt. Alle Werte werden am Ende in .txt geschrieben. Außerdem wird der Durchschnitt berechnet aller Bilder.

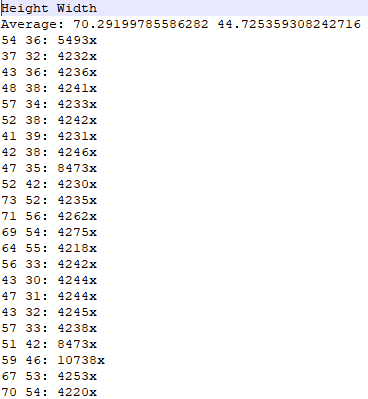


Abbildung 7: Größe Parklinglot Daten

1. Architektur und Designentscheide

**Programmiersprache und Umgebung**

Wieso Python? Wieso Jupyter Notebook? Wieso Colab?

**Datenerhebung**

Wieso diesen Datensatz?

Aufbau des Jupyter Notebooks

Aufbereitung der Daten:

Um die eingelesenen Daten optimal und effektiv zu bearbeiten, wurde die Klasse ImagedataGenerator genutzt. Diese wird von Keras zur Verfügung gestellt und liefert viele Möglichkeiten mit, die Bilder zu verarbeiten. Unter anderem können so die Daten eingelesen werden und gleich die Größe der Bilder angegeben werden. Außerdem kann über den ImageDataGenerator auch Augmentation betrieben werden, um Variationen in den einzelnen Bildern zu erzeugen. Ein weiter wichtiger Punkt war für diese Entscheidung, dass allein aufgrund der Ordnerstruktur beim Einlesen die Daten schon korrekt gelabelt werden.

Mit dieser Klasse können also viele zwingend notwendige Aufgaben in dem Projekt erledigt werden.

CNN:

Bei dem Model für das neuronale Netz handelt es sich um ein Convolutional Neural Network, mit dem schon eine relativ hohe Validation-Accuracy von 99% erreicht wird. Das Model wurde so gewählt, dass es relativ viele Pattern in den Bildern erkennt, aber zeitgleich soll das Trainieren des Models pro Epoche nicht allzu lange dauern.

Die Aufgabe war es nun aus diesem Model durch die nötigen Tests Erfahrungen zu erzielen, die Änderungen an selbem erzeugen. Das Model besteht aus mehreren Convolutional und Max Pooling Layern. Als Aktivierungsfunktion wurde für die Convolutional Layer die Funktion «relu» (Rectified Linear Unit, Abbildung x) genutzt. Diese ist die meist genutzte Aktivierungsfunktion bei Deep Learning Models. Wenn die Funktion einen negativen Input erhält, wird 0 zurückgegeben. Andernfalls wird der Input auch wieder zurückgegeben (f(x) = max(0, x)).

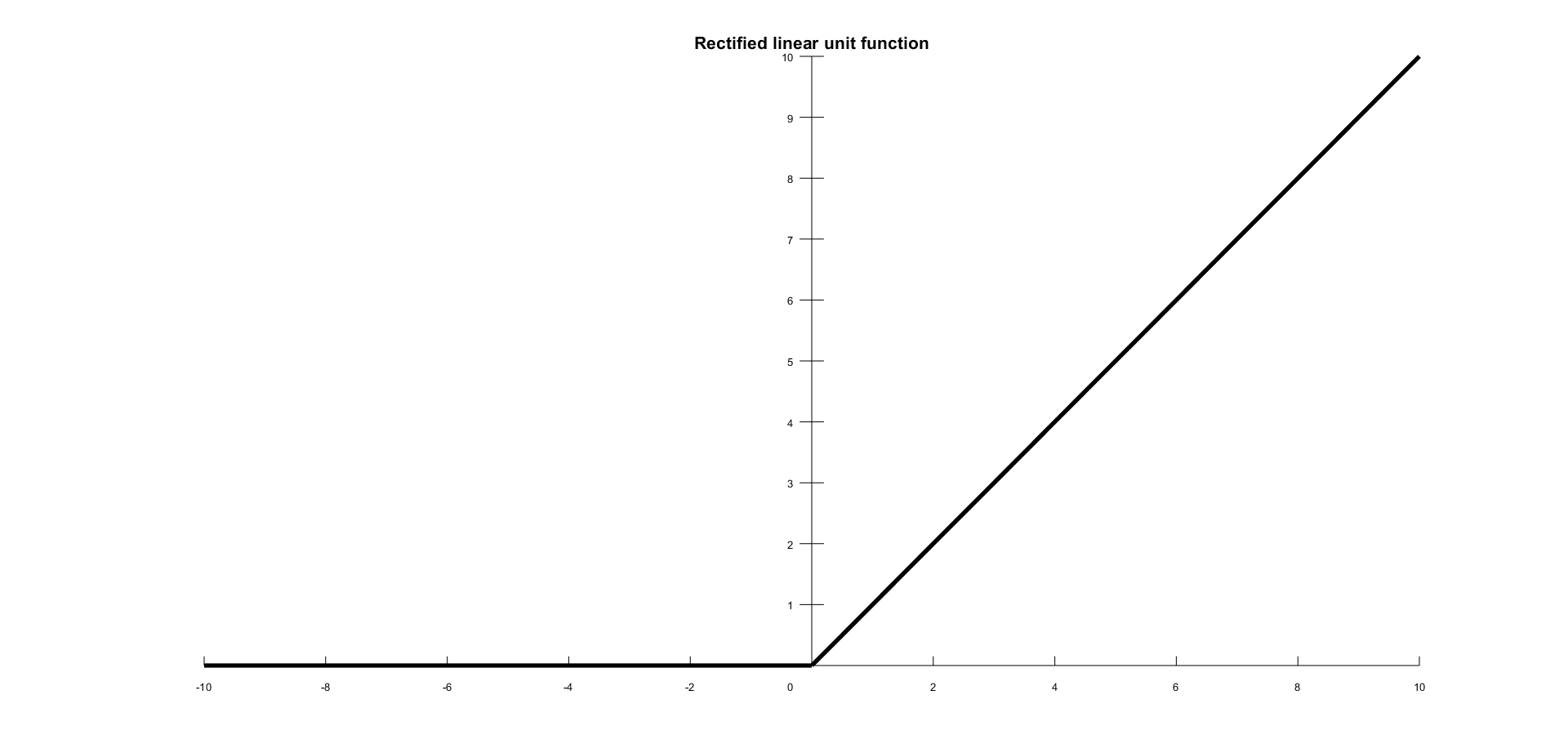


Abbildung 8: linear unit function

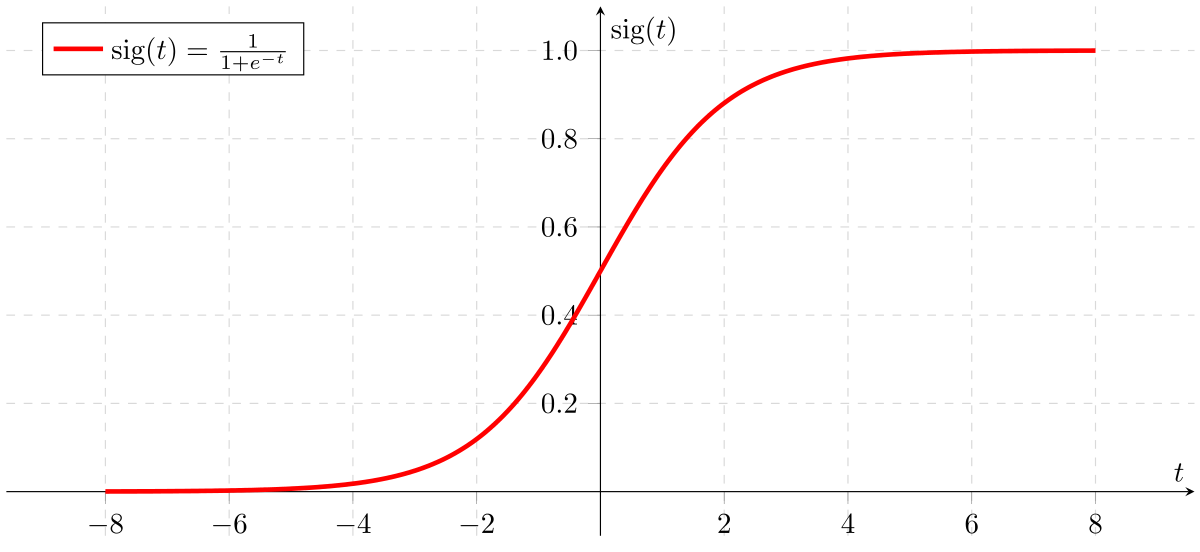
Da wir noch nicht viel Erfahrung mit neuronalen Netzen haben, orientierten wir uns also an dieser Funktion. Als letzte Aktivierungsfunktion musste jedoch die Sigmoid-Funktion gewählt werden, da wir aufgrund eines binären Problems (Empty oder Occupied) am Ende als Output einen Wert zwischen 0 und 1 haben. 

Abbildung 9: Sigmoid-Funktion

Dadurch, dass der Output nur zwischen 0 und 1 liegen kann, werden große Schwankungen beim Output verhindert und so Fehlerwerte vermieden. Zudem ist die Funktion an jeder Stelle differenzierbar, was insbesondere für die Backpropagation später von großem Nutzen ist.

Als Loss-Function wurde aufgrund des binären Problems die „binary\_crossentropy“ genommen. Der Optimizer für die Backpropagation ist „adam“, der für viele Problemstellungen im Deep Learning nützlich und immer eine gute Wahl ist. Für das Testen möglicher anderer Optimizer blieb leider keine Zeit. Desweiteren sind auch mehrere Dropout-Layer und BatchNormalization in dem Model zu finden. Diese werden später bei den Tests nochmals genauer erklärt.

Wie genau das Model aufgebaut ist, ist der Abbildung x zu entnehmen.

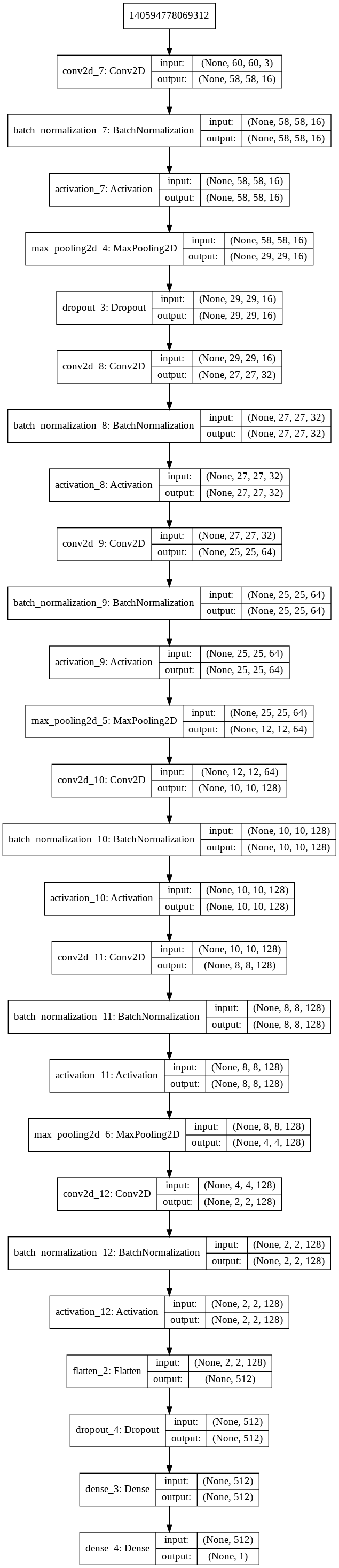


Abbildung 10: Verwendetes Netz mit Layern

## Daten (Mengengerüst & Strukturen) Felix

Alle Daten stammen aus zwei verschiedenen Quellen. Die erste Quelle (<https://web.inf.ufpr.br/vri/databases/parking-lot-database/>) wurde von unserem Kunden vorgegeben. Diese Daten stammen von der Informatik Fakultät der brasilianischen Universität «Universidade Federal do Paraná». In diesem Paket sind 2 Parkplätze über einen längeren Zeitraum aufgenommen worden. Ein Parkplatz davon aus zwei unterschiedlichen Kamerawinkeln. Dabei enthält das Paket die Aufteilung in Wettersituationen (sunny, cloudy, rainy), Tagen und in den jeweiligen Status des Parkplatzes (empty, occupied). Alle einzelnen Parkplätze wurden ausgeschnitten und in die einzelnen Ordner sortiert. Die ausgeschnittenen Bilder besitzen unterschiedliche Größen. Gleichzeitig gibt es die jeweiligen Gesamtbilder der Parkplätze mit einer dazugehörigen XML-Datei. Diese beinhaltet u.a. die Koordinaten, Größe sowie einen Booleanwert, ob der Parkplatz besetzt (1) oder frei ist (0). Insgesamt beinhaltet dieses Paket ca. 700.000 Bilder.

Der zweite Parkplatz (<http://cnrpark.it/>) ist bei Recherche zu Alternativen gefunden worden. Dieses Paket umfasst ca. 150.000 Bilder und nimmt Parkplätze mit neun verschiedenen Kameras auf. Auch hier wurden die Bilder in Wettersituation und Tage aufgeteilt. Der Unterschied in der Datenstruktur liegt dabei, dass die Bilder noch in Kameras aufgeteilt werden. Damit bestimmt werden kann, welche Parkplätze frei oder belegt sind, wurde eine .txt Datei angelegt mit Pfaden und einem Booleanwert.

Im Laufe des Projekts wurden diverse unterschiedliche Pakete angelegt, die verschiedenen Testsituationen zu Gute gekommen sind. Die Daten wurden jeweils aufgeteilt in train/validation Daten. Alle Daten wurden zufällig tageweise zugeordnet, damit keine doppelten Bilder vorkommen.

|  |  |
| --- | --- |
| **Name** | **Beschreibung** |
| UFPR04\_05\_train\_val\_test.zip | Zum Ende hin bevorzugtes Paket.  80/20/20 train/validation/test split  Insgesamt 66049 Bilder |
| PUC\_50k.zip | 50000 Bilder des PUC Parkplatzes vom Parkinglot Dataset in 80/20 Verteilung. |
| UFPR05\_50k.zip | 50000 Bilder des UFPR05 Parkplatzes vom Parkinglot Dataset in 80/20 Verteilung. |
| UFPR04\_50k.zip | 50000 Bilder des UFPR04 Parkplatzes vom Parkinglot Dataset in 80/20 Verteilung. |
| 50k\_All\_Parking\_Spaces.zip | 50000 Bilder von allen Parkplkätzen |
| ALL\_ALL.zip | Alle Parkplätze in 80/20 Split. 837991 Bilder |
| CNR\_TEST.zip | Paket, um den Testgenerator mit einem anderen Datensatz zu laden. Insgesamt 144965. Kein Split. |
| ALL\_100K.zip | Zufällige Verteilung der Bilder (100000) der Parkplätze PUC/UFPR05/UFPR04/CNR in 80/20 Verteilung. |
| ALL\_50K.zip | Zufällige Verteilung der Bilder (50000) der Parkplätze PUC/UFPR05/UFPR04/CNR in 80/20 Verteilung. |
| PUC\_UFPR05\_04\_50\_50.zip | Alle Bilder (695851) der Parkplätze PUC/UFPR05/UFPR04 vom Parkinglot Dataset in 50/50 Verteilung. |
| PUC\_50\_50.zip | Alle Bilder (424223) des Parkplatzes PUC vom Parkinglot Dataset in 50/50 Verteilung. |
| UFPR05\_50\_50.zip | Alle Bilder (165785) des Parkplatzes UFPR05 vom Parkinglot Dataset in 50/50 Verteilung. |
| UFPR04\_50\_50.zip | Alle Bilder (103522) des Parkplatzes UFPR04 vom Parkinglot Dataset in 50/50 Verteilung. |
| PUC.zip | Alle Bilder (424223) des Parkplatzes PUC vom Parkinglot Dataset in 80/20 Verteilung. |
| UFPR04\_05.zip | Alle Bilder (261956) der Parkplätze UFPR04/UFPR05 vom Parkinglot Dataset in 80/20 Verteilung. |
| UFPR05.zip | Alle Bilder (165785) des Parkplatzes UFPR05 vom Parkinglot Dataset in 80/20 Verteilung. |
| UFPR04.zip | Alle Bilder (103522) des Parkplatzes UFPR04 vom Parkinglot Dataset in 80/20 Verteilung. |

Tabelle 1: Daten-Aufteilung

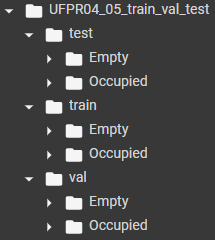
Alle Pakete besitzen eine einheitliche Datenstruktur. Im Stammverzeichnis der Pakete befinden sich die Unterordner train und val. Der train-Ordner beinhaltet alle Bilder, die das Netz lernen soll und der val-Ordner alle, um das Netz zu überprüfen ob es richtig lernt. In den Ordnern befinden sich nochmals die Ordner empty und occupied. Damit sind gleichzeitig alle Bilder mit Labels ausgestattet.

Abbildung : Ordnerstruktur Daten

Alle Pakete werden per Befehl aus OneDrive heruntergeladen, entpackt und in das Netz geladen. 

Abbildung 12: Befehle zum Herunterladen und Entpacken der Daten

Sobald diese Daten entpackt worden sind befindet sich die Struktur in dem jeweiligen Notebook. Damit die Bilder geladen werden können, wird programmatisch der Pfad angelegt.



Abbildung 13: Ordnerstruktur im Code

1. Umgebungs-Anforderungen

## Technologie-Voraussetzungen

Das System ist geschlossen aufgebaut und benötigt dadurch wenig Technologien. Im Folgenden werden die eingesetzten Technologien erklärt.

|  |  |
| --- | --- |
| Name | Beschreibung |
| Python 3.X | Grundlage jeglicher Programmierung |
| Keras | Paket für maschinelles Lernen in Python geschrieben |
| NumPy | Paket, um das Rechnen mit Matrizen und Vektoren zu vereinfachen, Geschrieben in NumPy und wird passiv mitgenutzt |
| Matplotlib | Library für die Darstellung von Diagrammen |
| Tensorflow | Keras stützt sich auf Tensorflow |
| Jupyter Notebook | Sozusagen die IDE |
| Foto Datenbank | <https://web.inf.ufpr.br/vri/databases/parking-lot-database/>  <http://cnrpark.it/>  Grundlage zum Anlernen |
| Google Colab | Hardware Grundlage |
| One Drive | Speichermedium für die Bildpakete |

Abbildung 14: Verwendete Technologie

Für unser Programm waren vor allem Keras bzw. Tensorflow und die Umgebung Google Colab wichtig. Keras bietet uns auf einer Hochsprachen-Ebene diverse Funktionen, um unser Netz zu konzipieren. Google Colab hingegen ist eine Plattform auf der potenten Hardware kostenlos zur Verfügung gestellt wird. Es können dort Notebooks hochgeladen werden und durchgeführt werden. Dies haben wir benutzt, da kein Teammitglied zuhause ausreichende Hardware zur Verfügung stehen hatte.

## Kooperierende Anwendungen und COTS-Komponenten

Das Notebook greift auf die hochgeladenen Pakete im OneDrive zu. OneDrive wird nur als Speichermedium genutzt, damit alle Teammitglieder jederzeit die Möglichkeiten haben, die Pakete herunterzuladen. Somit ist nur das Notebook von Nöten. Dazu wird ein Befehl ausgeführt, der sich per «wget» das Paket herunterlädt und danach entpackt. Die Pakete sind unter dem Punkt 3.2. genauer erklärt.

1. Testplan

Das Projekt sieht vor, wie Eingangs erklärt, ein neuronales Netzwerk aufzubauen zur Parkplatzerkennung. Da es in diesem Kontext keine sinnvollen Möglichkeiten gibt das Modell zu testen, wurden die Tests abgewandelt. Die Tests konzentrieren sich vor allem auf verschiedene Parameter und deren Wirkung auf das Netz und die Genauigkeit. Jedes Teammitglied hat sich dabei auf verschiedene Eigenschaften konzentriert. Diese Tests wurden ausdrücklich vom Kunden gewünscht.

## Testverfahren

Jede getestete Eigenschaft wurde unter einem bestimmten Verfahren geprüft. Jede Eigenschaft wurde in 30 Durchläufen getestet. In diesen 30 Durchläufen wurde das Prinzip des «early-stoppings» benutzt. Dies sagt aus, dass sobald nach 5 Epochen kein besseres Ergebnis erzielt worden ist, wird das Trainieren gestoppt wird. Die beste Validation Accuracy wurde dabei niedergeschrieben und zum Schluss in Konkurrenz mit den anderen Testergebnissen derselben Reihe gesetzt.



Abbildung 15: Early-Stopping

## Standardmodel

Das Standardmodel bietet die Grundlage für jegliches weitere Testen. Das Model wurde unter dem Punkt 3 erklärt. Jede weitere Eigenschaft ist eine Erweiterung zum Standardmodel. Dies bedeutet gleichermaßen, dass sich alle Testreihen mit dem Standardmodel messen lassen müssen. Das Ziel der einzelnen Tests besteht darin das Standardmodel zu erweitern bzw. zu verbessern.

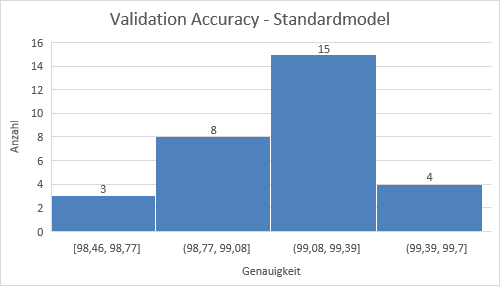


Abbildung 16: Validation Accuracy- Standardmodel - Histogram

Das Standardmodel ist in unserem Fall schon sehr genau. Ein Durchschnittswert von 99,08-99,39 wird zu 15 Mal erzielt in unseren Tests. Dies bedeutet für weitere Testreihen, dass wir versuchen werden die letzten Prozente aus der Simulation herauszukitzeln oder zumindest erkennen können, dass eine Veränderung des Netzes keine weiteren Vorteile für uns bringt.

Die hohe Genauigkeit kommt daher, dass unsere Testdaten relativ ähnlich sind und das Model teilweise Parkplätze trainiert, die er gleichzeitig auch validiert. Diese sind zwar von anderen Tagen bzw. anderen Uhrzeiten, aber der Parkplatz hat sich in der Zwischenzeit nicht grundlegend verändert.

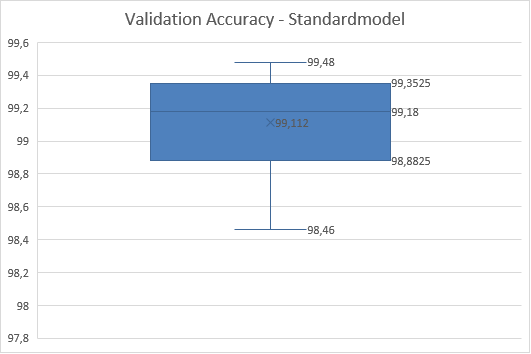


Abbildung 17: Validation Accuracy - Standardmodel - Boxplot

Als Referenzwert wird der Wert 99,112% angesehen.

## Felix

Das Framework Keras bietet die Möglichkeit über den sogenannten «ImageDataGenerator» (<https://keras.io/preprocessing/image/>) eine Augmentation (Veränderung/Bearbeitung) der Bilder durchzuführen. Dies soll dazu dienen mehr Testdaten zu erzeugen oder wie unserem Fall dem Modell unterschiedliche Arten von Bildern zuzuführen. Die Bilder können teilweise in schlechten Verhältnissen aufgenommen werden, wie z.B. eine Verzerrung, weil die Linse nicht fokussiert ist. Damit wir testen können, ob eine Augmentation das Netz positiv beeinflusst, wurden fünf verschiedene Eigenschaften getestet, wovon zwei Eigenschaften in unterschiedlichen Eigenschaften getestet worden sind. Zu beachten ist, im Standardmodel sind diese Eigenschaften nicht aktiv.

|  |  |
| --- | --- |
| **Eigenschaft** | **Wert** |
| rotation\_range | 45 |
| rotation\_range | 135 |
| width\_shift\_range | 9 |
| width\_shift\_range | 15 |
| horizontal\_flip | True |
| vertical\_flip | True |
| shear\_range | 9 |

Tabelle 2: Verwendete Eigenschaften Augmentation

### Rotation\_range

Die Eigenschaft «rotation\_range» bewirkt, dass das Bild sich um einen gewissen Grad dreht. Diese Eigenschaft wurde ausgewählt, da alle Parkplätze in verschiedenen Winkeln aufgenommen werden. Die Kamera kann nicht für jeden Parkplatz gerade ausgerichtet werden.

Damit das Netz sich auf verschieden gedrehte Bilder einstellen kann, wurden zwei verschiedene Bereiche ausgewählt. Zuerst 0-45 Grad und danach 0-135 Grad.

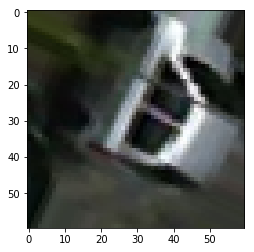


Abbildung : Beispiel rotation\_range = 135

Abbildung : Beispiel rotation\_range = 45

Während der Tests ist sehr schnell deutlich geworden, dass die Drehung der Bilder das Ergebnis verschlechtert um einen geringen Prozentsatz. Eine geringe Drehung kann mit dem Standardmodel mithalten, wobei eine starke Drehung das Ergebnis verschlechtert. Schaut man sich die Beispielbilder an, wird deutlich, dass die Bilder sich stärker verzerren durch die größere Rotation. Dies könnte dazu führen, dass das Netz die Bilder nicht mehr vernünftig erkennen kann.

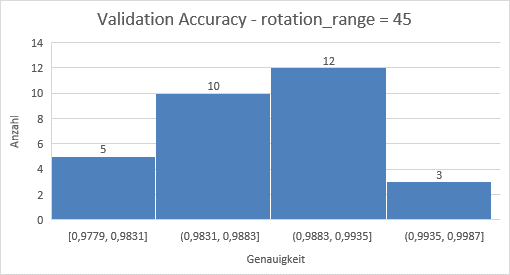


Abbildung 20: Validation Accuracy - rotation\_range = 45 - Histogram

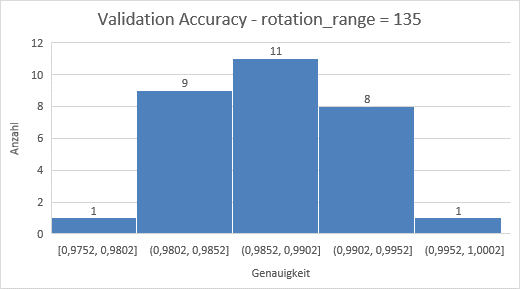


Abbildung 21: Validation Accuracy - rotation\_range = 135 - Histogram

Die Boxplots zeigen deutlich, in welchem Bereich die beiden Werte arbeiten. Der Durchschnitt liegt ca. 0,3% unter dem Standardmodel. Hierbei wird ersichtlich, dass eine Drehung der benutzten Daten für uns nicht sinnvoll ist. Sobald die rotation\_range hochgesetzt wird, verschlechterten sich die Ergebnisse auf allen Ebenen. Selbst die Ausreisser nach oben und unten werden schlechter.

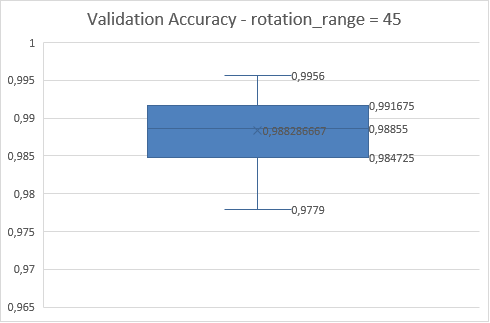


Abbildung 22: Validation Accuracy - rotation\_range = 45 - Boxplot

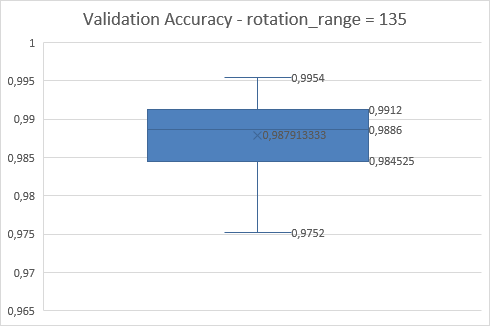


Abbildung 23: Validation Accuracy - rotation\_range = 135 - Boxplot

Somit ist die Eigenschaft «rotation-range» nicht sinnvoll für unser Model mit diesen Daten.

width\_shift\_range

Die Eigenschaft width\_shift\_range verschiebt das Bild um eine gewisse Anzahl von Pixeln nach links oder rechts. Da auch hiermit eine Verzerrung des Bildes simuliert wird, wurde diese Eigenschaft ausgewählt. Bei der Analyse der Ausgangsbilder wurde festgestellt, dass diverse Bilder eine leichte Verzerrung aufweisen.

Auch hier wurden zwei verschiedene Bereiche ausgewählt, um den Einfluss zu simulieren. Zuerst 0-9 und danach 0-15. Die Bereiche geben um wie viele Pixel die Bilder verschoben werden dürfen.

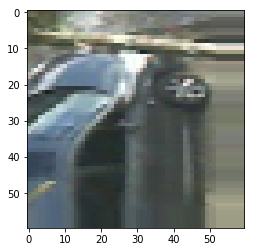
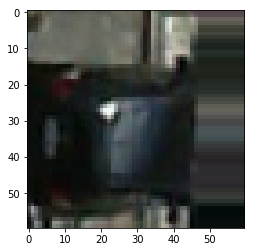


Abbildung : Beispiel width\_shift\_range = 15

Abbildung : Beispiel width\_shift\_range = 9

Bei der width\_shift\_range wird es schwieriger eine eindeutige Aussage zu treffen. Zuerst betrachten wir die Verschiebung um neun Pixel. Zuerst einmal wird deutlich, dass die Verschiebung sehr unterschiedlich ausfällt. In 16 von den 30 Fällen liegt sie unterhalb des Durchschnittswerts des Standardmodels. Gleichzeitig liegen aber auch 14 von 30 Fällen oberhalb des Ergebnisses. Daraus könnte man schließen, dass die Reihenfolge der Bilder wie diese trainiert werden eine Veränderung herbeiführen können. Die Reihenfolge wie die Bilder trainiert werden erfolgt zufällig.

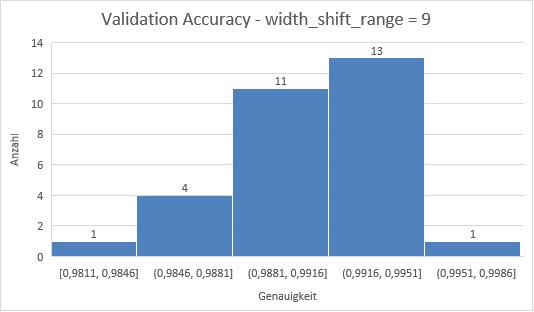


Abbildung 26: Validation Accuracy - width\_shift\_range = 9 - Histogram

Schaut man sich hingegen eine größere Verschiebung der Bilder an, wird auch hier deutlich, dass die Ergebnisse sehr kontrovers sind. Es scheint, dass die Bilder fast schon zufällig erkannt werden. Betrachtet man das Beispielbild entsteht eine starke Verzerrung des Bildes, sobald das Bild um 15 Pixel verschoben wird. Das Auto ist zu erkennen, aber es scheint, als könnte das Bild nicht komplett zugeordnet werden. Hierzu müsste eine tiefere Analyse der Ergebnisse durchgeführt werden.

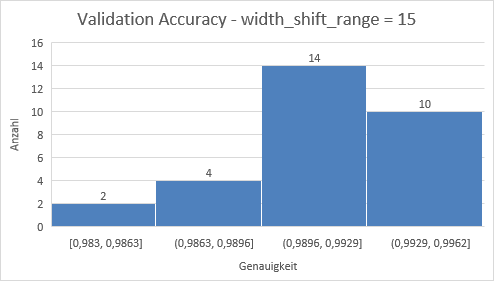


Abbildung 27: Validation Accuracy - width\_shift\_range = 15 - Histogram

Werden die Durchschnittswerte betrachtet, scheint eine größere Verzerrung das Netz sogar voranzubringen in der Erkennung. Daraus könnte man ableiten, dass eine Verzerrung für das Netz geeignet ist.

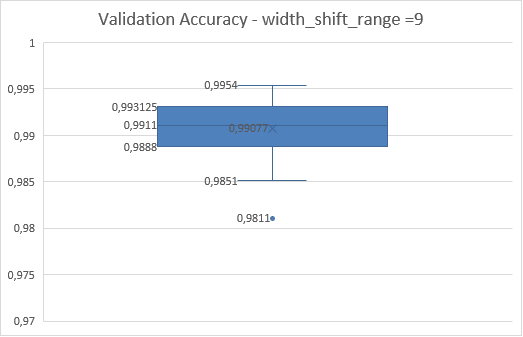


Abbildung 28: Validation Accuracy - width\_shift\_range = 9 - Boxplot

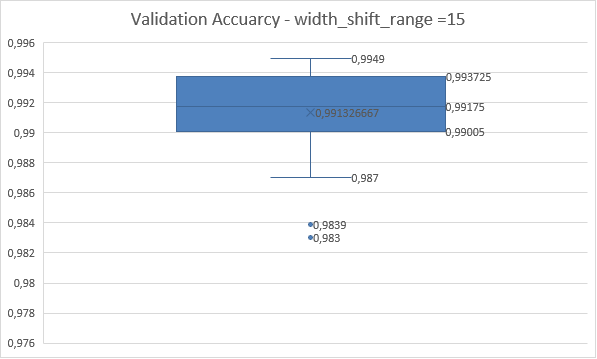


Abbildung 29: Validation Accuarcy - width\_shift\_range = 15 - Boxplot

### shear\_range

Die Eigenschaft Shear-Range bearbeitet das Bild mit einer sogennante Scheren Optik. Hierzu wird die X-Achse festgesetzt und alles oberhalb der Achse nach rechts oder links in einer trapezähnlichen Form verschoben.

Aufgrund von fehlender Zeit wurde nur eine Größe getestet. Diese belief sich auf neun Pixel.

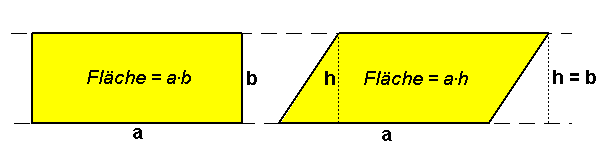


Abbildung 30: Geometriebesipiel shear-range

Wie in den vorrangegangenen Eigenschaften wurde diese Eigenschaft ausgewählt, da das Bild verzerrt wird und davon ausgegangen wird, dass nicht alle Bilder in einem perfekten Format bzw. Aufnahmestatus vorliegen.

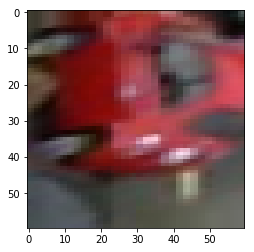


Abbildung 31: Beispiel shear-range

Bei der Betrachtung der Ergebnisse fällt direkt auf, dass die Genauigkeit allgemein sehr hoch ausfällt. Allein 21 von 30 Ergebnissen liegen über 99%. Wohingegen das niedrigste Ergebnis bei 98,71% liegt.

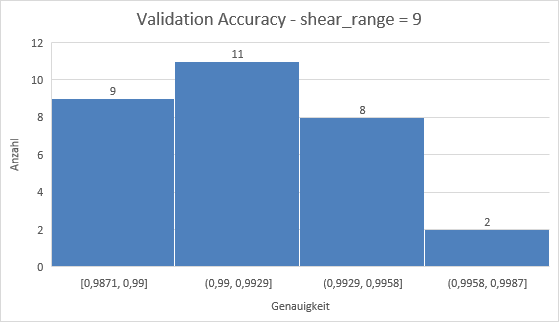


Abbildung 32: Validation Accuracy - shear\_range 9 - Histogram

Nehmen wir den Box-Plot der 30 Testergebnisse hinzu, kann direkt erkannt werden, dass die durchschnittliche Genauigkeit von 99,15% über dem Ergebnis des Standardmodels liegt. Darauf lässt sich ableiten, dass auch die Shear-Range ein Gewinn erzielen kann. Dieser fällt marignal aus, aber um die letzten Prozent herauszuholen wäre dies ein geeigneter Parameter.

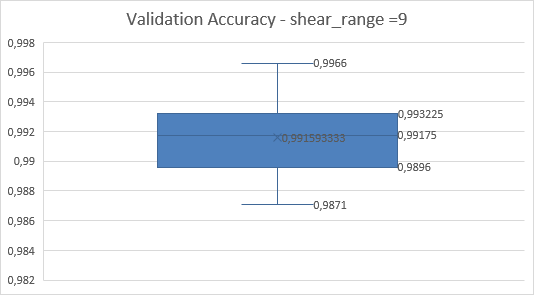


Abbildung 33: Validation Accuracy - shear\_range = 9 – Boxplot

Um die Grenzen der Shear-Range zu testen, sollten noch weitere Tests mit höheren Werten durchgeführt werden.

### Flip Image

Als letzte Eigenschaft wurde das drehen von Bildern getestet. Dazu gibt es zwei Eigenschaften. Zuerst das komplette Drehen horizontal und danach vertikal. Dazu wird das ganze Bild an der horizontalen oder vertikalen Linie gespiegelt. Diese Eigenschaften wurden ausgewählt, da jegliche Parkplätze in unterschiedlichen Lagen aufgenommen werwerden können.

Abbildung : Beispiel vertical\_flip =true

Abbildung : Beispiel horizontal\_flip = true

Sobald die Werte des Histogramms betrachtet werden, fällt auf das sowohl beim horizontal\_flip als auch der vertical\_flip die Genauigkeit gegenüber dem Standardmodel nachlässt. Die Konzentration ist vor allem im Bereich knapp unter 99%. Dies ist dahingehend erstaunlich, da das Drehen das Bild nicht verzerrt und somit das Bild vom Netz «normal» erkannt werden sollte. Das Netz sollte jede glich die Möglichkeit haben unterschiedliche Situationen genauer erkennen zu können.

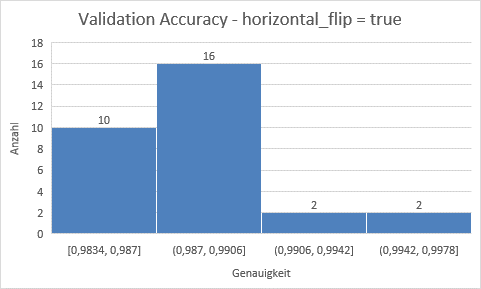


Abbildung 36: Validation Accuracy - horizontal\_flip = true - Histogram

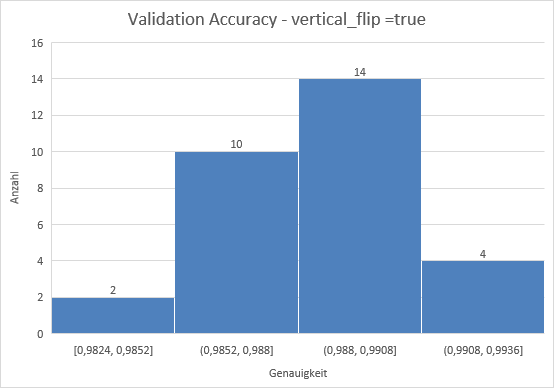


Abbildung 37: Validation Accuracy - vertical\_flip = true - Histogram

Werden die Box-Plots der Eigenschaften betrachtet, fällt auch hier auf, dass die Durchschnittswerte deutlich unter dem Standardmodel liegen. Dies führt zu der Annahme das diese Eigenschaften nicht für unser Model geeignet sind.

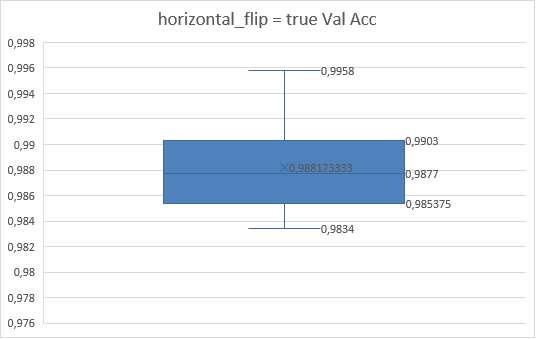


Abbildung 38: Validation Accuracy - horizontal\_flip = true - Boxplot

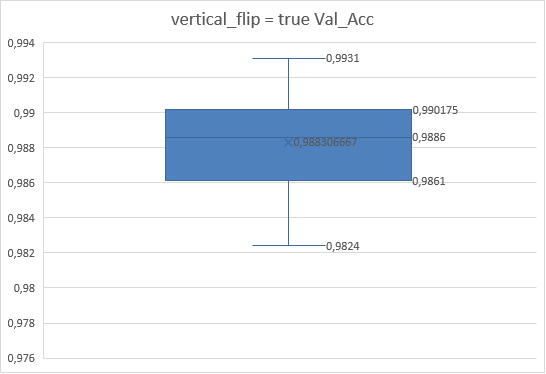


Abbildung 39: Validation Accuracy - vertical\_flip = true – Boxplot

Da diese Annahme für uns zurzeit noch nicht ganz verständlich ist, müssten auch hier weitere Tests durchgeführt werden. Es sollte geschaut werden, ob eine Veränderung am Netz notwendig ist, um eine höhere Genauigkeit zu erzielen bzw. es sollte erörtert werden woran es liegen kann, dass das CNN die Bilder um ca. 0,3-0,5% schlechter erkennt.

## Frede

## Pit

## Jascha

1. Ausblick
2. Abbildungsverzeichnis

[Abbildung 1: Produkt-Zyklus 3](#_Toc11437620)

[Abbildung 2: Berechnung CNN 5](#_Toc11437621)

[Abbildung 3: Matrix-Berechnung 6](#_Toc11437622)

[Abbildung 4: Layer-Berechnung 6](#_Toc11437623)

[Abbildung 5:Datenfehler 7](#_Toc11437624)

[Abbildung 6: Vergleich pyfastcopy zu Standart 8](#_Toc11437625)

[Abbildung 7: Größe Parklinglot Daten 9](#_Toc11437626)

[Abbildung 8: linear unit function 11](#_Toc11437627)

[Abbildung 9: Sigmoid-Funktion 11](#_Toc11437628)

[Abbildung 10: Verwendetes Netz mit Layern 12](#_Toc11437629)

[Abbildung 11: Ordnerstruktur Daten 14](https://d.docs.live.net/11f71a1654195330/Studium/Studium/6.%20Semester/Teamprojekt/Parkplatzerkennung/Dokumente_Teamprojekt/Projektbeschreibung_A15.docx#_Toc11437630)

[Abbildung 12: Befehle zum Herunterladen und Entpacken der Daten 15](#_Toc11437631)

[Abbildung 13: Ordnerstruktur im Code 15](#_Toc11437632)

[Abbildung 14: Verwendete Technologie 16](#_Toc11437633)

[Abbildung 15: Early-Stopping 17](#_Toc11437634)

[Abbildung 16: Validation Accuracy- Standardmodel - Histogram 17](#_Toc11437635)

[Abbildung 17: Validation Accuracy - Standardmodel - Boxplot 18](#_Toc11437636)

[Abbildung 18: Beispiel rotation\_range = 135 19](https://d.docs.live.net/11f71a1654195330/Studium/Studium/6.%20Semester/Teamprojekt/Parkplatzerkennung/Dokumente_Teamprojekt/Projektbeschreibung_A15.docx#_Toc11437637)

[Abbildung 19: Beispiel rotation\_range = 45 19](https://d.docs.live.net/11f71a1654195330/Studium/Studium/6.%20Semester/Teamprojekt/Parkplatzerkennung/Dokumente_Teamprojekt/Projektbeschreibung_A15.docx#_Toc11437638)

[Abbildung 20: Validation Accuracy - rotation\_range = 45 - Histogram 19](#_Toc11437639)

[Abbildung 21: Validation Accuracy - rotation\_range = 135 - Histogram 20](#_Toc11437640)

[Abbildung 22: Validation Accuracy - rotation\_range = 45 - Boxplot 20](#_Toc11437641)

[Abbildung 23: Validation Accuracy - rotation\_range = 135 - Boxplot 21](#_Toc11437642)

[Abbildung 24: Beispiel width\_shift\_range = 15 22](https://d.docs.live.net/11f71a1654195330/Studium/Studium/6.%20Semester/Teamprojekt/Parkplatzerkennung/Dokumente_Teamprojekt/Projektbeschreibung_A15.docx#_Toc11437643)

[Abbildung 25: Beispiel width\_shift\_range = 9 22](https://d.docs.live.net/11f71a1654195330/Studium/Studium/6.%20Semester/Teamprojekt/Parkplatzerkennung/Dokumente_Teamprojekt/Projektbeschreibung_A15.docx#_Toc11437644)

[Abbildung 26: Validation Accuracy - width\_shift\_range = 9 - Histogram 22](#_Toc11437645)

[Abbildung 27: Validation Accuracy - width\_shift\_range = 15 - Histogram 23](#_Toc11437646)

[Abbildung 28: Validation Accuracy - width\_shift\_range = 9 - Boxplot 23](#_Toc11437647)

[Abbildung 29: Validation Accuarcy - width\_shift\_range = 15 - Boxplot 24](#_Toc11437648)

[Abbildung 30: Geometriebesipiel shear-range 25](#_Toc11437649)

[Abbildung 31: Beispiel shear-range 25](#_Toc11437650)

[Abbildung 32: Validation Accuracy - shear\_range 9 - Histogram 26](#_Toc11437651)

[Abbildung 33: Validation Accuracy - shear\_range = 9 – Boxplot 26](#_Toc11437652)

[Abbildung 34: Beispiel vertical\_flip =true 27](https://d.docs.live.net/11f71a1654195330/Studium/Studium/6.%20Semester/Teamprojekt/Parkplatzerkennung/Dokumente_Teamprojekt/Projektbeschreibung_A15.docx#_Toc11437653)

[Abbildung 35: Beispiel horizontal\_flip = true 27](https://d.docs.live.net/11f71a1654195330/Studium/Studium/6.%20Semester/Teamprojekt/Parkplatzerkennung/Dokumente_Teamprojekt/Projektbeschreibung_A15.docx#_Toc11437654)

[Abbildung 36: Validation Accuracy - horizontal\_flip = true - Histogram 27](#_Toc11437655)

[Abbildung 37: Validation Accuracy - vertical\_flip = true - Histogram 28](#_Toc11437656)

[Abbildung 38: Validation Accuracy - horizontal\_flip = true - Boxplot 28](#_Toc11437657)

[Abbildung 39: Validation Accuracy - vertical\_flip = true – Boxplot 29](#_Toc11437658)

1. Tabellenverzeichnis

[Tabelle 1: Daten-Aufteilung 14](#_Toc11437611)

[Tabelle 2: Verwendete Eigenschaften Augmentation 18](#_Toc11437612)

1. Projektabschluss

*Ein letztes Update der Projektmanagementvorlage auf den Projekt-Schlussstand, dabei sind insb die folgenden Absätze und Kapitel zu aktualisieren*

**Absatz 2.2 Projektkontrolle und Projektsteuerung**

* *Getroffene Maßnahmen und ihre Auswirkungen im Projektverlauf.*

**PM - Anhänge**

<für den Projektabschluss sind abzugeben >

* Sprintpläne
* Sprintreview-Protokolle
* Meilensteinberichte
* **Aufgaben und Zeitaufwände aller Teammitglieder dargestellt über den Verlauf des Projekts**