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

**Systembeschreibung**

[1. Ziel und Zweck des Dokumentes 3](#_Toc11100368)

[1.1. Projektbeschreibung 3](#_Toc11100369)

[1.1.1. Kurzbeschreibung des Projekts 3](#_Toc11100370)

[1.1.2. Zweck des Projekts 4](#_Toc11100371)

[1.1.3. Hintergrund, Problemstellung, Motivation für das Projekt 4](#_Toc11100372)

[1.1.4. Ziele des Projekts 4](#_Toc11100373)

[1.1.5. Erfolgskriterien 4](#_Toc11100374)

[2. Systemübersicht 5](#_Toc11100375)

[2.1. CNN 5](#_Toc11100376)

[2.2. Skripte 7](#_Toc11100377)

[2.2.1 cutImages.py 7](#_Toc11100378)

[2.2.2. Copy\_pictures.py 8](#_Toc11100383)

[2.2.3. Picture\_random\_sort.py 9](#_Toc11100384)

[2.2.4. picturesize.py 9](#_Toc11100385)

[3. Architektur und Designentscheide 10](#_Toc11100386)

[3.1. Daten (Mengengerüst & Strukturen) Felix 13](#_Toc11100387)

[4. Umgebungs-Anforderungen 16](#_Toc11100388)

[4.1. Technologie-Voraussetzungen 16](#_Toc11100389)

[4.2. Kooperierende Anwendungen und COTS-Komponenten 16](#_Toc11100390)

[5. Testplan 17](#_Toc11100391)

[5.1. Frede 17](#_Toc11100392)

[5.2. Pit 17](#_Toc11100393)

[5.3. Jascha 17](#_Toc11100394)

[5.4. Felix 17](#_Toc11100395)

[6. Projektabschluss 18](#_Toc11100396)

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 | In Bearbeitung |
| 0.3 | 09.06.2019 | Frederik Rieß | Erste Verbesserungen durchgeführt  Kapitel 2.1 fertiggestellt  Kapitel 2.2 begonnen (cutImages.py) | In Bearbeitung |
| 0.4 | 10.06.2019 | Felix Willrich | Kapitel 2 Skripte beschrieben | In Bearbeitung |

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.

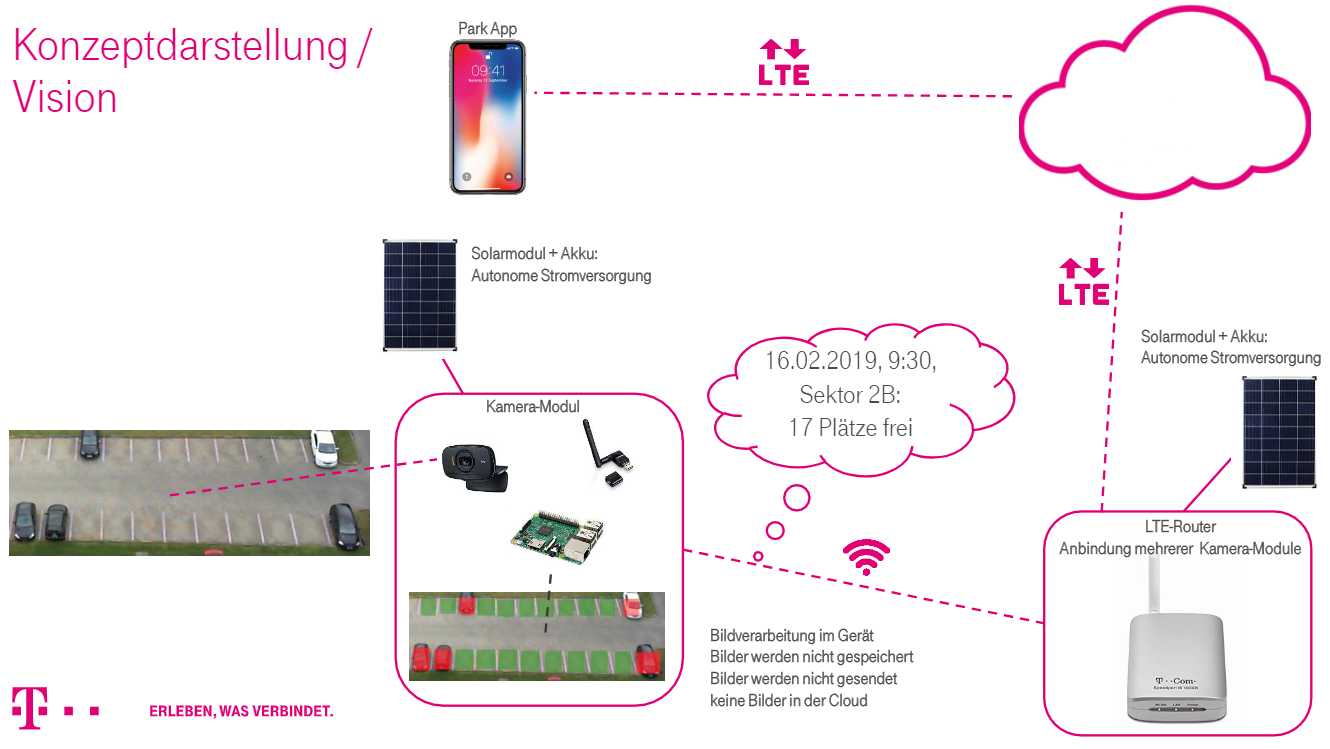


Abbildung 1: Produkt-Zyklus

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

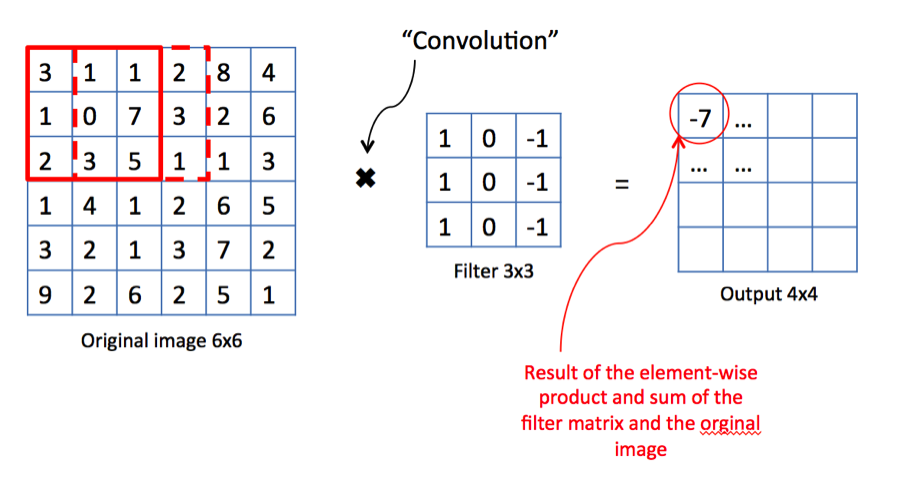
Sollte die gewünschte Genauigkeit erlangt worden sein, wird das Projekt als Erfolg bezeichnet. Weiterhin hinzukommen würden verschiedene Umgebungen, wie Schnee, Regen und andere Hindernisse wie Baulöcher oder Belegung von zwei Parkplätzen gleichzeitig. Sollten diese zusätzlichen Kriterien erfüllt werden,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

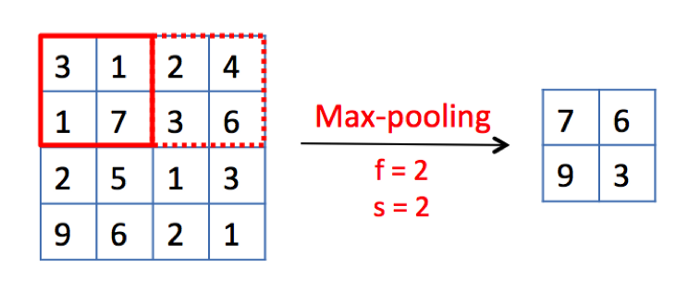
2. Systemübersicht

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

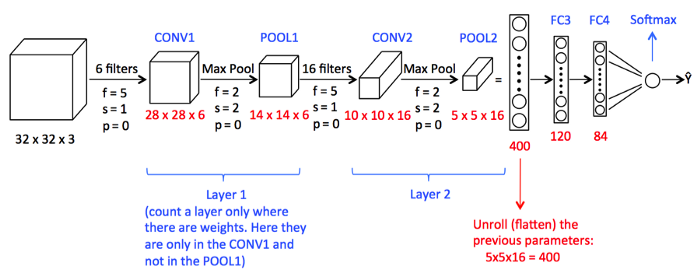
Convolutional Neural Networks haben sich bei vielen Aufgaben bewiesen, die der Bilderkennung dienen. Diese Netzwerke bestehen aus mehreren Convolutional und Max Pooling Layern und erkennen bestimmte Pattern (Ecken, Linien etc.) in Bildern. Je tiefer das CNN entwickelt ist, desto genauer kann das Netz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

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.

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.



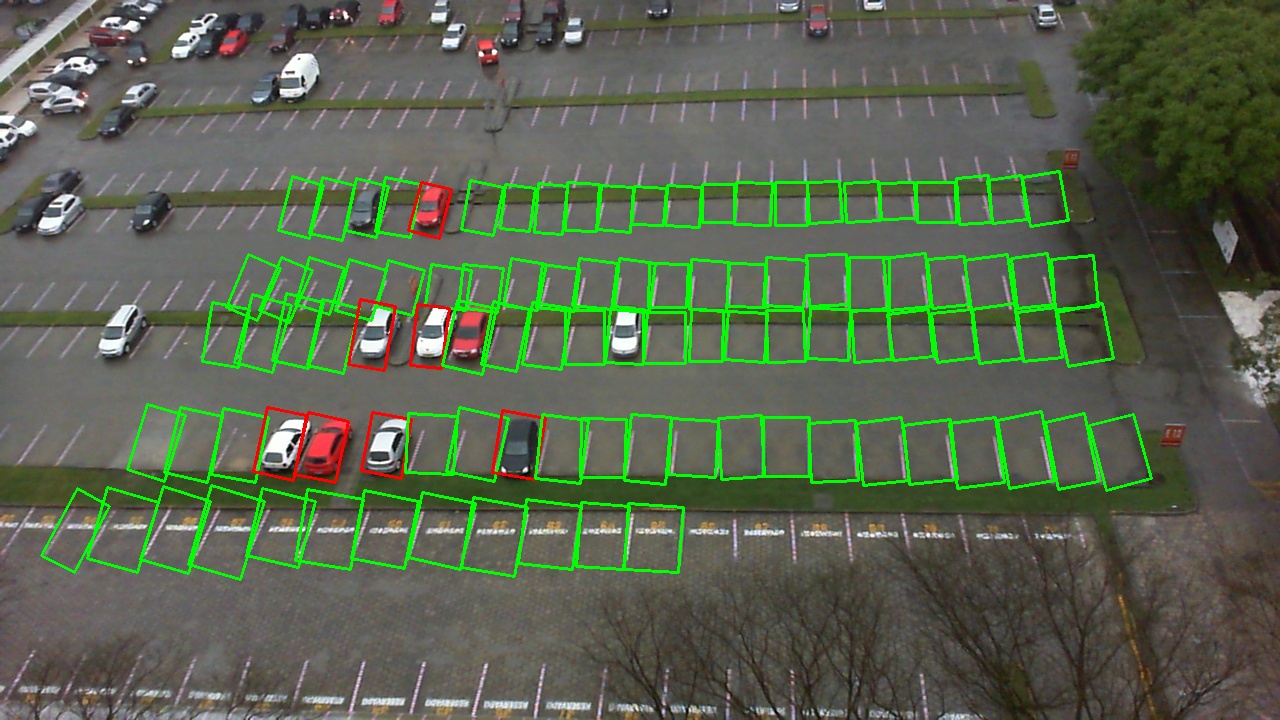
## 2.2. Skripte

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.



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

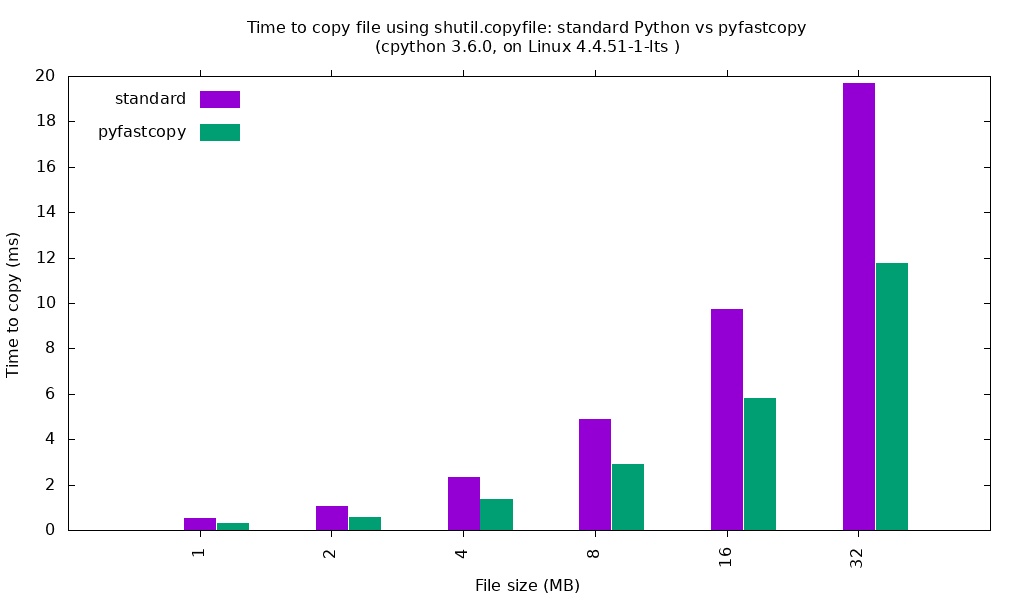
Damit jedes Bild von den unterschiedlichen Datensätzen in unserer festgelegten Struktur vorliegt, wurden zwei Skripte geschrieben, die uns dabei unterstützen. Copy\_pictures.py und CNR\_COPY.py. Beide Skripte funktionieren mit dem gleichen Mechanismus.

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.



### Picture\_random\_sort.py

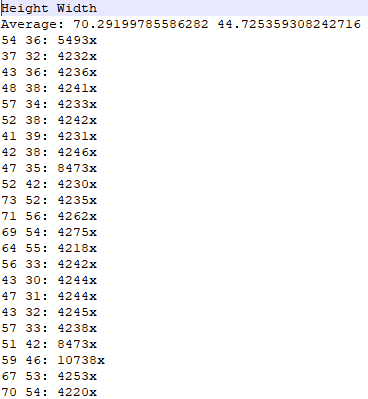
Einige Bildpakete setzten voraus, dass alle Bilder zufällig einsortiert werden, damit das Netz keine falschen Muster lernt. Um dies zu garantieren, wurde ein Skript geschrieben, welches alle Bilder zufällig einsortiert in die jeweiligen Ordner.

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.



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:

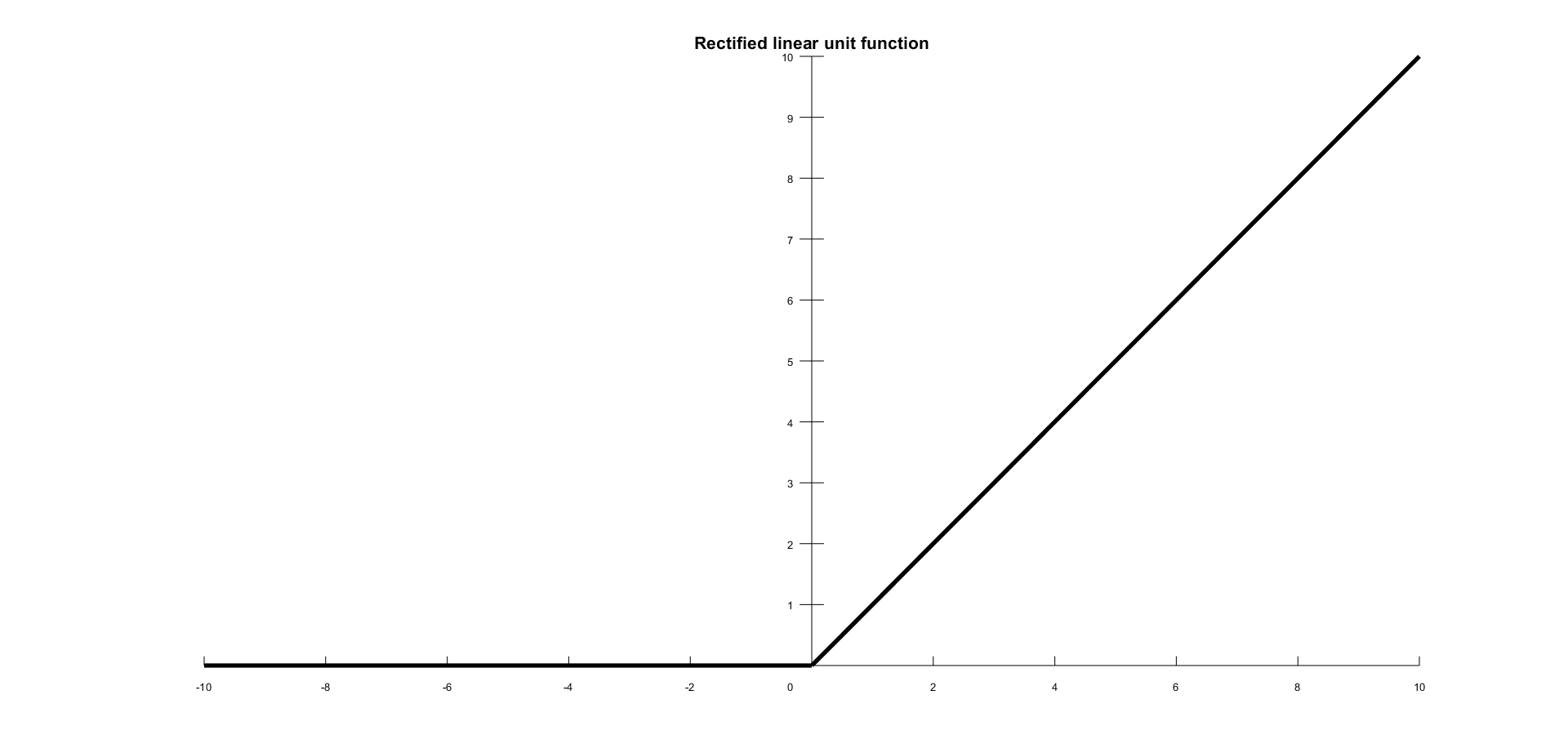
Um die eingelesenen Daten optimal und effektiv zu bearbeiten, wurde die Klasse ImagedataGenerator genutzt. Diese wird von Keras zur Verfügung gestellt und liefert viele Möglichkeiten mit, die Bilder zu verarbeiten. Unter anderem können so die Daten eingelesen werden und gleich die Größe der Bilder angegeben 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.

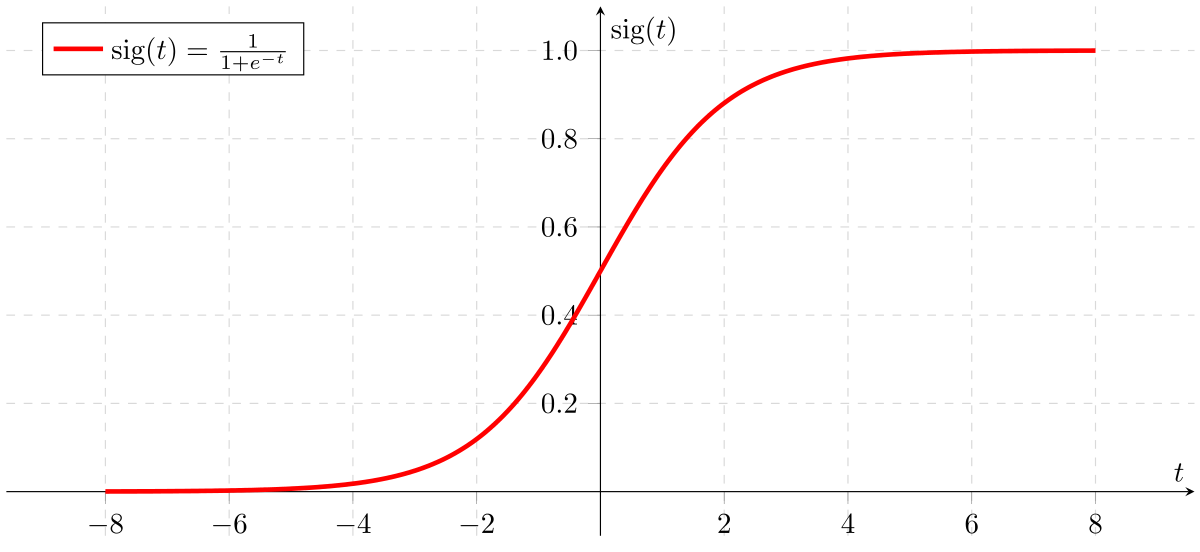
Mit dieser Klasse können also viele zwingend notwendige Aufgaben in dem Projekt erledigt werden.

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)).

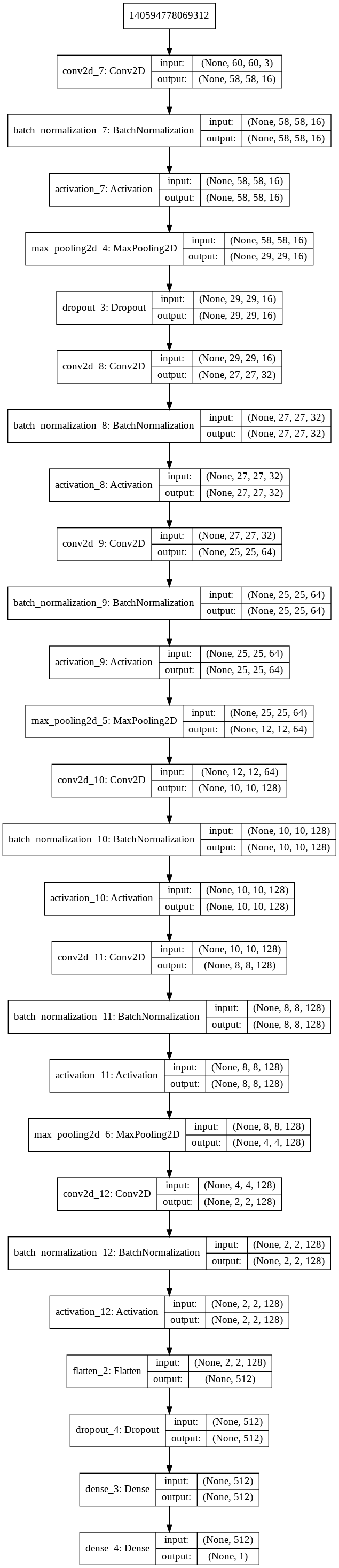


Da wir noch nicht viel Erfahrung mit neuronalen Netzen haben, orientierten wir uns also an dieser Funktion. Als letzte Aktivierungsfunktion musste jedoch die Sigmoid-Funktion gewählt werden, da wir aufgrund eines binären Problems (Empty oder Occupied) am Ende als Output einen Wert zwischen 0 und 1 haben. 

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.



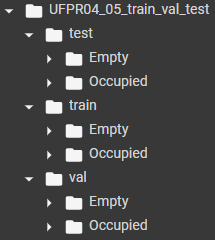
## Daten (Mengengerüst & Strukturen) Felix

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. |

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.

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

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.



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 |

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.

## Frede

## Pit

## Jascha

## Felix

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**