Frankfurt University of Applied Sciences
Fachbereich 2 Studiengang Informatik

Bachelorthesis

Entwicklung eines intelligenten „Stein-Schere-Papier“-Agenten mit Hilfe von Bilderkennung

Eingereicht zum Erlangen des akademischen Grads
Bachelor of Science

Autor: Erhan Tokmak
MatNr. 956727

Version vom: 12. Februar 2020

1. Betreuer: Prof. Dr. Christian Baun
2. Betreuer: Prof. Dr. Matthias Deegener

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher weder gesamt noch in Teilen einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

$Unterschrift:$ $Ort, Datum:$

Danksagung

An dieser Stelle möchte ich mich bei meiner Familie und meinen Freunden bedanken, die mich über die Jahre unterstützt und begleitet haben. Herrn Prof. Baun gilt mein besonderer Dank für die Unterstützung bei der Themenfindung sowie die Begleitung während der Ausarbeitung. Weiterhin möchte ich mich hiermit bei Herrn Prof. Dr. Deegener bedanken, der sich als Korreferent zur Verfügung gestellt hat.

Zusammenfassung

Maschinelles Lernen ist in den letzten Jahren zu einem wesentlicher Bestandteil technischer Neuerungen geworden. Maschinen können daraufhin trainiert werden, eigenständig zu erkennen, zu verarbeiten und auszuwerten. Dies bietet ein enormes Potential und steht somit im Mittelpunkt der wirtschaftlichen und gesellschaftlichen Entwicklung des frühen 21. Jahrhunderts.

In diesem Projekt geht es darum mittels Gestenerkennung Handzeichen zu erkennen und anschließend auszuwerten. Die zu erkennenden Handzeichen sind aus dem bekannten Spiel Stein-Schere-Papier. Jedoch wird das Spiel um zwei Handzeichen erweitert, nämlich um Echse und Spock. Dies führt zu einer veränderten Dynamik des Spiels und die Komplexität wird insgesamt gesteigert. Zur Erreichung der Gestenerkennung, wird eine Maschine darauf trainiert, die aus dem Spiel bekannten Handzeichen zu erkennen. Dies wird mit der Hilfe eines Raspberry Pi und einer Kamera realisiert. Auf einer grafischen Oberfläche wird das erkannte Handzeichen angezeigt. Das Programm erwidert anschließend mit einem zufälligen Handzeichen. Die Anzeige des Gewinners erfolgt in Echtzeit.

Abstract

In recent years, machine learning has become an essential part of technical innovations. Machines can be trained to recognize, process and evaluate independently. This offers enormous potential and should be considered one of the major developments of the early 21st century in economics and in society.

This project is about recognizing hand gestures by using gesture recognition and then evaluating them. The hand gestures to be recognized originate from the well-known game rock-paper-scissors. However, the game is expanded by two hand gestures - namely lizard and Spock. This leads to a changed dynamic of the game and thereby the complexity increases. In order to achieve gesture recognition, a machine is trained to recognize the hand gestures of the game. This is achieved in corroboration of a Raspberry Pi and a camera. The recognized hand gestures are displayed on a graphical user interface. The program respond with a random hand gesture. The winner is displayed in real time.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	=	Application Programming Interface
ARM	=	Acorn Risc Machine
CNN	=	Convolutional neural network
CPU	=	Central Processing Unit
CSI	=	Camera serial interface
CSV	=	Comma-separated values
GB	=	Gigabyte
GHz	=	Gigahertz
GPU	=	Graphics processing unit
GUI	=	Graphical user interface
IEEE	=	Institute of Electrical and Electronics Engineers
I/O	=	Eingabe/Ausgabe
LAN	=	Local Area Network
MB	=	Megabyte
MicroSD	=	Kleine sichere digitale Speicherkarte
OpenCV	=	Open source computer vision
RAM	=	Random-Access Memory
SoC	=	System on chip
USB	=	Universal Serial Bus
WLAN	=	Wireless Local Area Network
mm	=	Millimeter
XML	=	Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Das Erkennen und Auswerten von Bildern oder direkten Aufnahmen wird immer wichtiger. Im Bereich Sicherheit und Überwachung werden zum Beispiel Gesichter von Personen erkannt, um Unbefugten den Zugriff oder das Eindringen zu verweigern. Durch Gesichtserkennung können beispielsweise Smartphones entsperrt oder auch gezielt Personen in der Öffentlichkeit verfolgt werden, um Bewegungsprofile zu erstellen.

Ein weiteres Anwendungsgebiet ist die Qualitätssicherung bei Produktionslinien, bei der durch Bilderkennung direkt Auffälligkeiten in der Produktionslinie erkannt und gemeldet werden, um gegebenenfalls fehlerhafte Chargen zu vermeiden.

Abhängig vom Einsatzbereich solcher Anwendungen muss eine Maschine speziell dafür trainiert werden. Für das Trainieren einer solchen Maschine benötigt man normalerweise leistungsstarke Prozessoren. Im Abschnitt Bewertung wird gezeigt, wie viel Zeit es in Anspruch nimmt, eine solche Maschine zu trainieren.

Die vorliegende Bachelorthesis befasst sich mit der Realisierung eines solchen Projektes mit Hilfe eines Einplatinencomputers, dem Raspberry Pi 4. Dieser Computer soll durch den Einsatz einer Kamera erkennen, welche der fünf möglichen Handbewegungen gerade gezeigt werden und durch eine Voraussage, das eingesetzte Handzeichen erwidern. Die Bilderkennung soll flüssig und genau ablaufen. Um diese Funktionalitäten zu ermöglichen, werden verschiedene Protokolle und Anwendungsschnittstellen verwendet, wie zum Beispiel OpenCV, Keras und Tensorflow, auf die im Abschnitt Design genauer eingegangen wird.

Die Erfindung des Spiels „Stein, Schere, Papier, Spock, Echse“ stammt von Sam Kass und Karen Bryla. Hier eine Übersicht der möglichen Spielzüge:

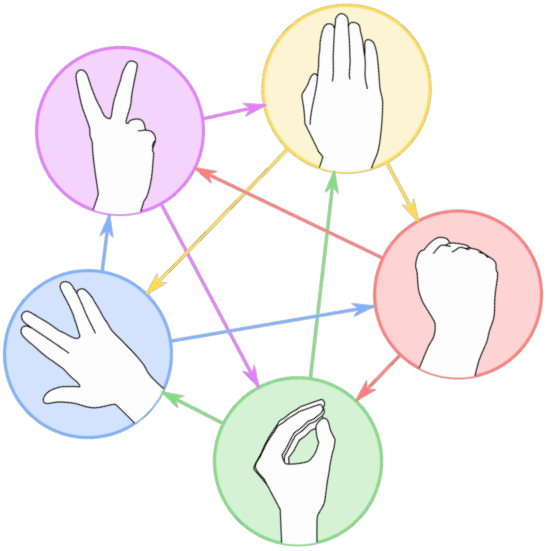

Abbildung 1: Spielablauf[1]

Damit das Spiel ausgeglichen und fair ist, wurde das Spiel so entworfen, dass jedes Handzeichen zwei Handzeichen besiegt und gegen zwei Handzeichen verliert. Die schafft die Möglichkeit, alle Handzeichen gleichmäßig einzusetzen, ohne das es zur Benachteiligung eines Zeichens kommt. Die genauen Spielregeln sind:

- Spock gewinnt gegen Schere und Stein verliert jedoch gegen Papier und Echse.
- Schere gewinnt gegen Papier und Echse verliert aber gegen Stein und Spock.
- Papier gewinnt gegen Spock und Stein verliert allerdings gegen Schere und Echse.
- Stein gewinnt gegen Echse und Schere, verliert jedoch gegen Papier und Spock.
- Echse gewinnt gegen Spock und Papier, verliert aber gegen Stein und Schere.

[1] Bildquelle:`https://commons.wikimedia.org/wiki/File:Pierre_ciseaux_feuille_l%C3%A9zard_spock.svg`

2 Stand der Technik

Dieses Kapitel geht auf den aktuellen Stand der Technik im Bereich Maschinelles Lernen ein und zeigt beispielhaft auf, welche Projekte zum Thema Gestenerkennung Inspiration für das Projekt dieser Thesis sind. Weiterhin wird die eingesetzte Hardware beschrieben, die es zur Realisierung des Projekts benötigt.

Anhand von zwei Anwendungsbeispielen von großen Konzernen soll nachfolgend die Relevanz von Maschinellem Sehen in der heutigen Zeit deutlich gemacht, sowie das Ausbaupotential aufgezeigt werden.

2.1 Anwendungsbeispiele Maschinelles Lernen

Die Amazon.com Inc. hat in den Vereinigten Staaten von Amerika eine Reihe an Lebensmittelläden unter dem Namen „Amazon Go“ eröffnet, die sich größtenteils auf Künstliche Intelligenz verlassen.

Ziel des Geschäftsmodells ist es, den Kunden den Einkauf durch die Abschaffung des manuellen Bezahlvorgangs zu erleichtern. Voraussetzung ist der Download einer Anwendung auf das Smartphone des Nutzers. Die Applikation fordert den Kunden beim Eintreten in den Supermarkt auf, sich mit einem Scanvorgang einzuchecken. Ab diesem Zeitpunkt wird der Kunde von Kameras erfasst und durchgehend bei seinem Gang durch das Geschäft verfolgt. Per Gesichtserkennung wird festgestellt, welcher Kunde welche Lebensmittel den Regalen entnimmt und auch gegebenenfalls wieder zurücklegt. Nach Beendigung des Einkaufs kann der Kunde ohne physischen Erfassungs- und Bezahlvorgang den Supermarkt verlassen. Die Abrechnung erfolgt dann automatisch über den Amazon-Zugang des Kunden. Diese Vorgehensweise minimiert den personellen Aufwand massiv. Dennoch kann Amazon Go nicht gänzlich auf Mitarbeiter verzichten. Das Auffüllen der Regale erfolgt weiterhin manuell. Außerdem unterstützen Mitarbeiter die Kunden bei ihrem Einkauf. Bisher wertet Amazon dieses neue Geschäftsmodell als Erfolg, auch wenn es an manchen Stellen noch Schwierigkeiten mit der Erkennung von Produkten oder der genauen Verfolgung dieser während des Einkaufsvorgangs gibt.

Auch in der Automotive Industrie gewinnt Maschinelles Sehen immer mehr an Präsenz. Die Weltgesundheitsorganisation (WHO) gibt an, dass es jährlich über 1,25 Millionen Verkehrstote gibt. Große Konzerne wie die Tesla Inc. oder Google LLC entwickeln selbstfahrende Autos mit dem Ziel, dies zu reduzieren. Durch selbstfahrende Autos kann menschliches Fehlverhalten, wie beispielsweise durch Unachtsamkeit oder Trunkenheit am Steuer, übergangen werden.

Google gibt an, dass ihr selbstfahrendes Auto, welches seit 2016 von dem Tochterunternehmen Waymo LLC betrieben und weiterentwickelt wird, dank Sensoren, Kameras, Laser, Sonar und Radar bis zu 300 Meter weit sehen kann. Zum Trainieren der Maschine wird das Fahrzeug täglich zu verschiedenen Zeiten und in verschiedenen Wetterbedingungen gefahren. Auswertungen zeigen, dass das Fahrzeug in für Menschen schlecht einsehbaren Situationen, wie zum Beispiel bei schlechten Wetterbedingungen, besser „sehen“ kann als Menschen.

Abbildung 2: Maschinelles Sehen am Beispiel des Google Autos[2]

Von der Software erfolgt eine Auswertung aller gesammelten Daten und es wird ein möglichst sicherer Weg geplottet. Mit den ausgewerteten Information versucht das Auto zusätzlich, nach der geltenden Straßenverkehrsordnung zu fahren.

2.2 Bisherige Projekte im Bereich Gestenerkennung

Die Grundidee für diese Thesis im Bereich der Gestenerkennung stammt von einem bereits existierenden Projekt von Julien de la Bruère-Terreault[5]. In seiner Arbeit geht es darum, eine Maschine zu Trainieren, die drei Handzeichen erkennen soll, „Stein“, „Schere“ und „Papier“[3].

[2]Bildquelle: `https://www.youtube.com/watch?v=ROAwXEqDk7k&t=60s`

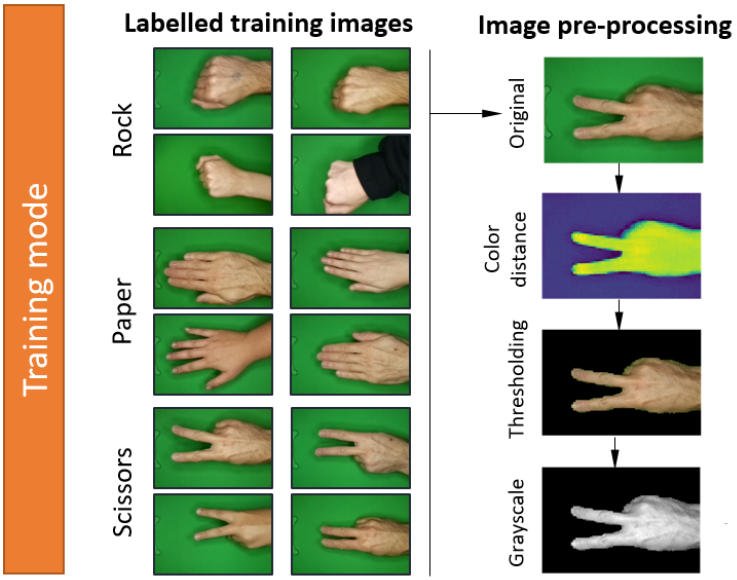

Abbildung 3: Projekt von Julien de la Bruère-Terreault[3]

Dafür wurde ein spezielles Gerüst gebaut, auf dem die Installation des Raspberry Pi mit der Kamera erfolgt. Damit die Maschine die Handzeichen besser erkennt, ist der Boden grün verkleidet und von oben mit einer konstanten Lichtquelle ausgerüstet [4].

Abbildung 4: Gerüst des Projekts von Julien de la Bruère-Terreault[4]

Die Schwachstelle des oben gezeigten Agenten liegt darin, dass er lediglich Handzeichen in der vorgegebenen Umgebung (grüner, ausgeleuchteter Hintergrund sowie immer gleichbleibende Entfernung zur Kamera) erkennt.

Ein anderes Projekt auf der Seite Github vom Benutzer SouravJohar[6] bietet eine Lösung mit Keras und SqueezeNet. Auch in seiner Arbeit sollen die Handzeichen Stein, Schere und Papier erkannt werden, jedoch wird das Modell mit SqueezeNet trainiert.

[3]Bildquelle: `https://github.com/DrGFreeman/rps-cv`
[4]Bildquelle: `https://github.com/DrGFreeman/rps-cv`

Dies hat zur Folge, dass weniger Leistung von dem Raspberry Pi verlangt wird und eine flüssigere Bildrate entsteht. In den Abschnitten Design und Implementierung wird dies detaillierter beschrieben.

Ziel dieser Thesis ist es, weitestgehend unabhängig von Umgebung und/oder Reichweite der Kamera die Handzeichen zu erkennen. Des Weiteren erfolgt die Erweiterung der möglichen Handzeichen von drei (Stein, Schere, Papier) auf fünf Handzeichen (Stein, Schere, Papier, Echse, Spock). Die Maschine soll zusätzlich erkennen und ausgeben, ob der Spieler oder die Maschine nach einem gespielten Durchgang gewonnen hat.

2.3 Raspberry Pi

Der Raspberry Pi ist ein Einplatinencomputer und wurde von das Raspberry Pi Foundation entwickelt. Raspberry Pis sind Ein-Chip-Systeme, die mit einem ARM Prozessor ausgestattet sind und die Entwicklung erfolgte mit der Absicht, Preisgünstige Computer zum Experimentieren und Programmieren zur Verfügung zu stellen. Es gibt neben den eigentlichen Computern auch ein großes Angebot an Zubehör, das die Funktionen erweitert, wie zum Beispiel Antennen, Kameras oder Messgeräte.

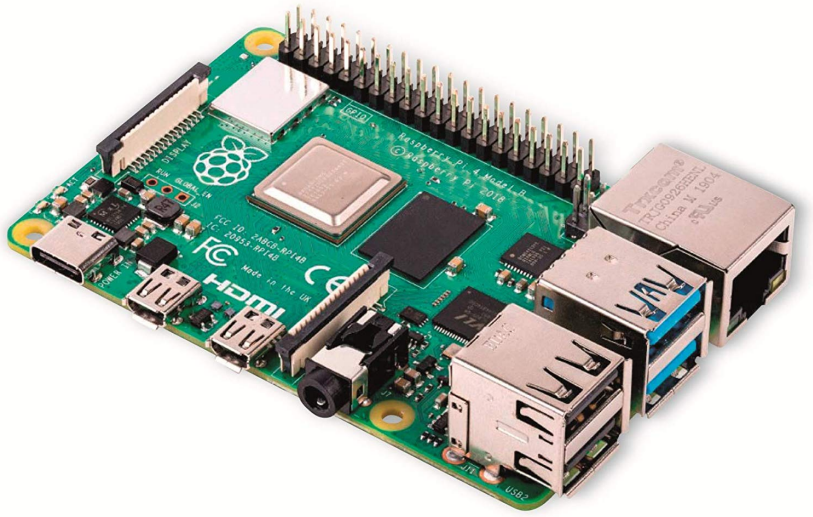

Abbildung 5: Raspberry Pi 4 Model B[5]

Für das Projekt wird der Raspberry Pi 4 Model B verwendet. Der Raspberry Pi 4 Model B kam im Juni 2019 auf den Markt. Die Spezifikation des Computers sind in der nachfolgenden Tabelle 1 abgebildet.

Bezeichnung	Typ
SoC	Broadcom BCM2711
CPU	4x ARM Quad-Core-Cortex-A72, 1,5 GHz
GPU	Broadcom Dual Core VideoCore VI
RAM	LPDDR4-SDRAM 4GB
Speicherkarte	SanDisk Ultra 64GB microSDXC
Netzwerk	10/100/1000 Ethernet, 2,4 GHz und 5,0 GHz IEEE 802,11ac WLAN

Tabelle 1: Raspberry Pi Hardwarekomponente

Eine der verwendeten Kameras ist die offizielle Raspberry Pi Kamera V2.0. Die Kamera kann mit allen bisherigen Raspberry Pi Geräten mit einem Flachbandkabel über die serielle Schnittstelle (CSI) verbunden werden. Der Bildsensor ermöglicht es, Bilder mit einer Auflösung von 3280 x 2464 Pixeln und Videos mit 1920x1080 Pixeln aufzunehmen.

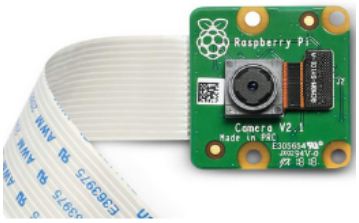

Abbildung 6: Raspberry Pi Kamera 2.0[6]

Die zweite Kamera, die genutzt wird, ist die Microsoft LifeCam VX-3000, die über USB an den Raspberry Pi angeschlossen werden kann.

Abbildung 7: Microsoft LifeCam VX-3000[7]

[6]Bildquelle:https://www.amazon.de/LABISTS-Offizielle-Raspberry-V2-0-Kamera-Modul/dp/B07VRJKYYB/ref=sr_1_7?__mk_de_DE=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=pi+cam+2.0&qid=1580655160&sr=8-7

[7]Bildquelle:https://www.amazon.de/Microsoft-LifeCam-VX-3000-original-Handelsverpackung/dp/B000GEU6TA/ref=sr_1_2?__mk_de_DE=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=Microsoft+LifeCam+VX-3000&qid=1580654999&sr=8-2

Nachfolgenden ein Vergleich der technischen Parameter der beiden eingesetzten Kameras.

	Microsoft LifeCam XV-3000	Raspberry Pi Kamera v2.0
Bildauflösung	1280 x 960 Pixel (1,3 Megapixel)	3280x2464 Pixel (8 Megapixel)
Videoauflösung	480p30	1080p30
Größe	73mm x 55mm x 53mm	25mm x 23mm x 9mm
Verbindung	USB	Flachbandkabel
Preis	40,71 €	26,99 €

Tabelle 2: Kameravergleich

2.4 Python

Die Programmiersprache für das Projekt ist Python. Python gehört zu den beliebtesten Programmiersprachen für Maschinelles Lernen nach Angaben von GitHub. Deswegen gibt es viele Anlaufpunkte, die es ermöglichen, die Sprache zu lernen. Die Programmiersprache ist auch deswegen so beliebt, weil sie relativ wenige Schlüsselwörter benötigt und die Syntax auf Übersichtlichkeit optimiert ist. Dies hat zur Folge, dass Python-Skripte kürzer formuliert werden als in anderen Sprachen. Das Trennen von Programmblöcken oder Methoden funktioniert in Python durch Einrücken und wird mit Doppelpunkten initialisiert. Das folgende Beispiel

```
1 for i in range(10):
2         print("Hallo Welt")
```

gibt zehn Zeilen lang den Text „Hallo Welt“ aus. In anderen Sprachen erfolgt diese Initialisierung oft durch geschweifte Klammern. Das gleiche Beispiel in der Programmiersprache C++ ist länger und die Einrückung dient nur der Leserlichkeit.

```
1 #include <iostream>
2 using namespace std;
3
4 int main () {
5    for( int a = 0; a <=10; a ++ ) {
6       cout << "Hallo Welt"<< endl;
7    }
8    return 0;
9 }
```

Bei längeren Programmen kommt es schnell zu einer Anhäufung an Zeilen in dem Programmcode. Auch deswegen werden Prototypen oft schnell mit Python programmiert. Für komplexere Anwendungen werden jedoch meist effizientere Sprachen eingesetzt, da Python im Vergleich eher langsam ist.

2.5 Open source Computer Vision Library

Open source Computer Vision Library, auch OpenCV genannt, ist eine quelloffene Software-Bibliothek mit über 2.500 Algorithmen für die Bildverarbeitung, Video I/O, High-level GUI Objekterkennung und Maschinelles Lernen. Die Bibliothek findet Verwendung in verschiedenen Anwendungsfeldern, wie zum Beispiel Gesichtserkennung, Objekterkennung, mobile Roboter oder Gestenerkennung. Es gibt Schnittstellen für die Programmiersprachen C++, C , Python, Java und MATLAB. OpenCV läuft auf allen gängigen Betriebssystemen und bietet dadurch vielen Entwicklern einen umfangreichen Zugang zur Bibliothek.

OpenCV kann von Deep Learning Frameworks, wie auch Tensorflow, vortrainierte Netze einlesen und einen Forward Pass[16] ausführen. Beim Forward Pass werden die Eingabedaten durch das Netzwerk geschickt. Jede versteckte Schicht im Netzwerk akzeptiert die Eingabedaten, verarbeitet diese Daten und gibt sie weiter zur folgenden Schicht.

3 Design

Im folgenden Kapitel werden Lösungsansätze, genutzte Programme und Vorgehensweisen beschrieben, die man zur Einrichtung der Entwicklungsumgebung benötigt.

3.1 Einrichtung der Entwicklungsumgebung

Zunächst ist es notwendig, ein Betriebssystem auf eine MicroSD-Karte zu installieren. Hierfür wird eine Speicherkarte mit 64 Gigabyte verwendet, auf der die Installation des Betriebssystems Raspbian erfolgt. Die Karte wird über ein Kartenlesegerät mit einen Computer verbunden. Zur Installation auf die Speicherkarte wird die Software BalenaEtcher[13] benutzt. Nach der Installation kann die Speicherkarte in den Raspberry Pi eingesteckt werden. Anschließend muss das Gerät beim Starten des Raspberry Pis eingestellt werden. Die Einstellungen sind wie folgt:

- Gerätename: Pi (beliebig wählbar)
- Passwort: beliebig wählbar
- Sprache und Tastatur-Layout (Deutsch)
- WLAN wird nicht verwendet, da das Gerät über ein LAN-Kabel verbunden ist.

Im letzten Schritt sucht das Gerät nach möglichen Updates und installiert diese automatisch. Zum Ausnutzen der vollständigen Speicherkapazität der MicroSD-Karte erfolgt über die Konsole des Raspberry Pis der Aufruf des Konfigurationsmenüs. Dafür wird der Befehl

```
1 $ sudo raspi-config
```

benutzt. Anschließend wird im Menü unter Advanced Options das Expand Filesystem angezeigt. Die Aktivierung der Einstellung ergibt sich durch das Drücken der Eingabetaste am Computer. Die USB-Kamera wird automatisch erkannt und ist direkt einsatzbereit. Die Raspberry Pi Kamera muss zusätzlich im Konfigurationsmenü aktiviert werden. Im Untermenü *Interface Options* wird die Option *Camera* angezeigt und kann hier durch das Drücken der Eingabetaste aktiviert werden. Die Auswahl *Finish* führt zum Speichern der getätigten Einstellungen, damit kann der Raspberry Pi neu gestartet werden.

Bei der Eingabe des Befehls *df -h* in der Konsole wird nun angezeigt, dass die gesamte Speicherkarte in Verwendung ist. Bei der Installation werden auch diverse Programme installiert, die für das Projekt irrelevant sind und unnötig viel Speicherplatz einnehmen. Die Deinstallation für nicht benötigte Software erfolgt mit den Befehlen

```
1 $ sudo apt-get purge wolfram-engine
2 $ sudo apt-get purge libreoffice*
3 $ sudo apt-get clean
4 $ sudo apt-get autoremove
```

Listing 1: Deinstallierung von Programmen die nicht notwendig sind

Somit wird 1 Gigabyte an Speicherkapazität zurückgewonnen. Damit ist alles vorbereitet und bereit zum Start.

3.2 Tensorflow Anwendungsschnittstelle als Lösungsansatz

Tensorflow[14] ist die quelloffene Software-Bibliothek von Google für numerische Berechnungen und Maschinelles Lernen. Sie bietet unterschiedliche Modelle und Algorithmen für Mehrschichtiges Lernen. Unter Mehrschichtigem Lernen versteht man die Bildung von künstlichen neuronalen Netzen. Die neuronalen Netze haben zwischen Eingabeschicht und Ausgabeschicht viele versteckte Zwischenschichten (siehe Abbildung 8).

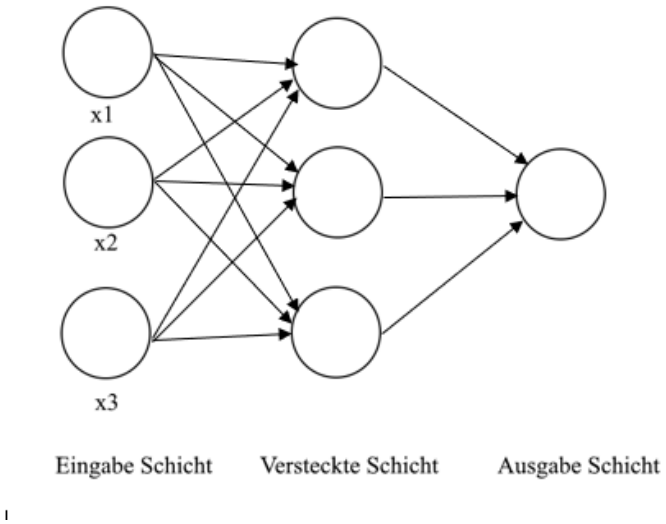

Abbildung 8: Beispiel eines künstlichen neuronalen Netzes

In diesen Schichten wird nach wichtigen Merkmalen wie Form, Farbe und Strukturen und deren Relation zueinander gefiltert. Wenn sich beispielsweise die erste Ebene um die Helligkeit und Farben kümmert, verwendet die nächste Ebene den Output der vorherigen Ebene, um Flächen, Kurven, Linien oder Kanten zu erkennen. So erfolgt die immer weitere Abstrahierung der Information in jeder Ebene. Aus Linien entstehen geometrische Strukturen und daraus bilden sich zum Beispiel Gesichter oder Bilder von Tieren. Das Filtern von solchen Details wurde früher fest vergeben. Heutzutage erfolgt der Einsatz von Convolutional Neural Networks (übersetzt: gefaltete neuronale Netze), um diese Neuronen oder Filter selbständig mit dem Trainingsdatensatz zu definieren.

Das grundlegende Element von Tensorflow ist der sogenannte Graph. Der Graph ist ein mathematisches Problem, welches mithilfe eines gerichteten Diagrams abstrahiert dargestellt wird. Das Diagramm setzt sich aus Knoten und Kanten zusammen, die miteinander verbunden sind. Die Knoten repräsentieren mathematische Operationen und Daten in Tensorflow. Dadurch wird das Modell für das Maschinelle Lernen selbst entwickelt und nicht, wie in anderen Bibliotheken, mit voreingestellten Modellen. Jedoch müssen in den Graphen die Gewichtungen variiert werden. Dies geschieht durch die iterative Zuführung von Trainingsdaten an den Rechner. Dadurch wird die Ausgabe so verändert, dass sich der Graph dem anvisierten Ausgabewert immer weiter nähert.

3.3 MASK-RCNN

Tensorflow bietet eine Reihe von vortrainierten Erkennungsmodellen, wie auch Tensorflow detection model zoo. Tensorflow detection model zoo ist eine Implementierung von Mask R-CNN[17] in Python 3, Keras und Tensorflow. Das Modell generiert Rahmen und Segmentierungsmasken für jede Instanz eines Objektes im Bild. Es basiert auf dem Feature Pyramid Network (FPN) und einem RasNet101 backbone. In dem GitHub Verzeichnis „Tensorflow detection model zoo“ stehen verschiedene Modelle zur Auswahl. Diese sind aufgeteilt in Hardwarekomponenten und Geschwindigkeit. Beispielsweise ist *ssd_mobilenet_v2_coco* optimiert für mobile Geräte und bietet eine nicht so hohe Genauigkeit wie *faster_rcnn_inception_v2_coco*. Die höhere Genauigkeit ist jedoch auch mit mehr Rechenleistung beim Trainieren verbunden.
Das Modell *faster_rcnn_inception_v2_coco* gibt an, in 58 Millisekunden einen mittleren Präzisionswert (engl. mean Average Precision oder auch mAP) von 28 zu haben. Die Geschwindigkeit wurde mit einer GTX TITAN X Grafikkarte und Bildern, die eine Größe von 600x600 Pixel haben, gemessen.

3.4 Loss-Funktion

Maschinen lernen mittels einer Loss-Funktion[19]. Mit dieser Methode wird das Modell mit dem gegebenen Datensatz bewertet. Wenn die Vorhersagen zu sehr von dem originalen Ergebnis abweichen, gibt die Loss-Funktion eine hohe Zahl aus. Mithilfe von Optimierungsfunktionen lernt die Loss-Funktion, Fehler in der Vorhersage zu reduzieren. In diesem Abschnitt erfolgt die nähere Erläuterung der Funktionen.

Die Einteilung von Loss Funktionen erfolgt in zwei Kategorien, *Regression losses* and *Classification losses*.

Classification-Loss:

Mit dieser Methode wird mit einer Reihe von endlichen Daten eine Vorhersage getroffen, beispielsweise die Kategorisierung von Handzeichen anhand von Bildern.

Regression-Loss:

Diese Methode befasst sich mit der Vorhersage von kontinuierlichen Werten, beispielsweise der Vorhersage vom Kaufpreis einer Wohnung anhand der Anzahl der Zimmer, der Größe der Zimmer und der Grundfläche.

Vereinfacht ausgedrückt sollte die Punktzahl an richtig kategorisierten Daten größer sein als jene, deren Kategorisierung falsch ist. Wenn sich jedoch die Loss-Funktion zu sehr verkleinert, passt sich die Maschine zu sehr an die Trainingsdaten an. Dies nennt sich auch overfitting (siehe Abbildung 9). In einem solchen Fall wird eine hohe Genauigkeit angegeben, die Maschine scheitert aber im Testlauf. Wenn die Loss-Funktion zu hoch ist und sich nicht verkleinert, ist das ein Zeichen von underfitting (siehe Abbildung 9). In dem Fall verbessert sich das Modell nicht und kann im Testlauf die Eingabedaten nicht richtig kategorisieren.

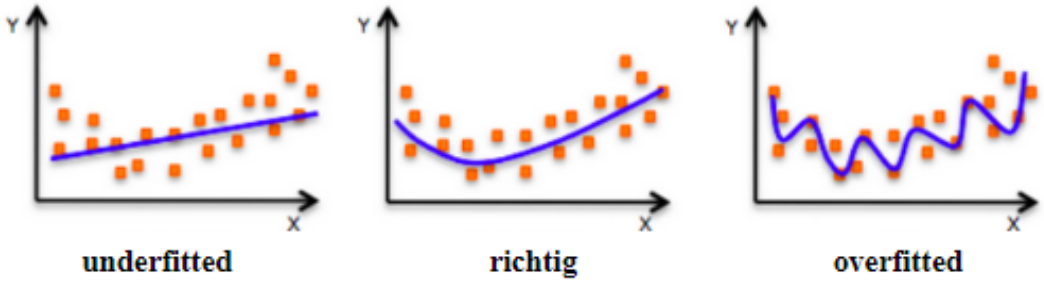

Abbildung 9: Unterschiede zwischen underfitting und overfitting[8]

Um Herauszufinden, ob das Modell overfitted oder underfitted ist, wird der Datensatz in Training und Test aufgeteilt, wobei Training 80 Prozent des Datensatzes beinhaltet und Test 20 Prozent. So wird das Modell beim Trainieren mit dem Testdatensatz begutachtet und bewertet.

Um dagegen zu agieren, können einige Maßnahmen getroffen werden[20].

- **Cross-validation** hilft präventiv gegen overfitting, mithilfe der Erstellung von "mini train-test splits". Während des Trainings werden immer wieder einige Daten zurückgehalten und dann zum Testen verwendet.
- **Eine größere Menge an Daten** verwenden. Bei einer zu geringen Anzahl an Trainingsdaten kann das Modell nicht gut trainiert werden.

[8]Bildquelle: `https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/`

- **Vorzeitiges Stoppen**, wenn während der Trainingsphase der Loss-Wert immer kleiner wird aber bei der Bewertung mit dem Testdatensatz der Loss-Wert steigt, sollte der Trainingsvorgang frühzeitig beendet werden.

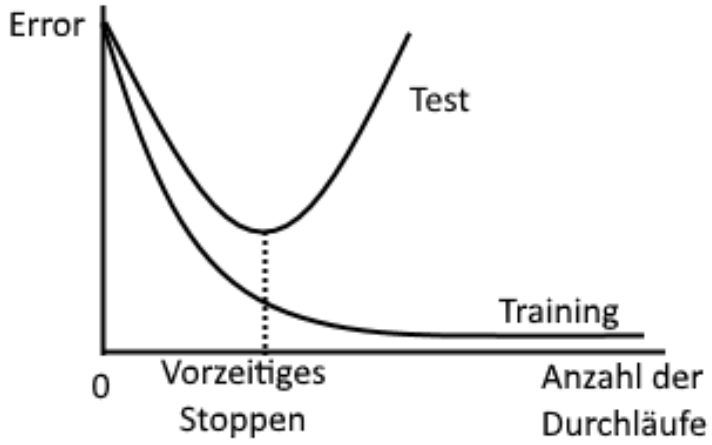

Abbildung 10: Vorzeitiges Stoppen

- **Dropout**: Diese Methode ignoriert während der Trainingsphase im Neuronalen Netz per Zufall Knoten. Dadurch wird die Information nicht immer von denselben Knoten bearbeitet, die Gewichtung auf bestimmten Knoten wird nicht zu stark und es kommt zu einer Nutzung des gesamten Netzes.

3.5 Keras in Verbindung mit SqueezeNet

Keras ist eine quelloffene Software Bibliothek für Mehrschichtiges Lernen[21]. Keras bietet eine Schnittstelle für Tensorflow und andere Backends. Seit dem Erscheinen von Tensorflow 1.4 ist Keras Teil der Tensorflow Kern-Anwendungsschnittstelle. Allerdings wird die Bibliothek ungebunden weitergeführt. Die Bibliothek stellt eine high-level Anwendungsschnittstelle zur Verfügung, welche die Programmierung von neuronalen Netzen stark vereinfacht. Die Vorteile von Keras sind

- das schnelle und einfache Erstellen neuronaler Netze.
- Modularität, Erweiterbarkeit und Benutzerfreundlichkeit.
- die Unterstützung von sowohl rekurrenten als auch konvolutionalen Netzwerken beziehungsweise deren Kombination.
- und die Unterstützung des Multi-Input-Trainings, Multi-Output-Trainings, sowie des GPUs und CPUs.

Durch die Modularität lassen sich Module beliebig kombinieren und konfigurieren. So können schnell neue Modelle geschaffen werden, die leicht als neue Funktionen oder Klassen hinzugefügt werden können.

3.5.1 SqueezeNet

Keras bietet den Zugriff auf das mehrschichtige neuronale Netz namens SqueezeNet[22] für Maschinelles Sehen. SqueezeNet gibt die Möglichkeit, ein sich faltendes neuronales Netzwerk (Convolutional Neural Network) aufzubauen. Die Vorteile von SqueezeNet sind,

- dass kleinere Convolutional Neural Networks während des Trainings weniger Kommunikation zwischen Servern erfordern.
- dass kleinere Convolutional Neural Networks weniger Bandbreite zum Übertragen von neuen Modellen benötigen.
- dass kleinere CNNs leichter auf FPGA (Feld programmierbare Gatter-Anordnung) und anderer Hardware mit begrenztem Arbeitsspeicher bereitgestellt werden können.

Squeezenet bietet ebenso wie AlexNet ein hohes Level an Genauigkeit, (mit dem Datensatz „ImageNet“) benötigt jedoch 50 Parameter weniger, nach der Kompression sogar 500 Parameter weniger. In diesem Fall ist SqueezeNet außerdem dreimal schneller als AlexNet (siehe Abbildung 11).

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al. 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al. 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al. 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	**50x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	**363x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	**510x**	57.5%	80.3%

Abbildung 11: SqueezeNet und AlexNet Vergleich[9]

Dies wird durch das Ersetzen von 3x3 Filtern gegen 1x1 Filter erreicht. Daraus erfolgt eine 9-fache Reduzierung der Menge an Parametern. In einem sogenannten „fire module“ gibt es ein „squeeze layer“ und ein „expand layer“. Die erste Schicht besteht aus 1x1 Filtern und im „expand layer“ wird eine Kombination aus 1x1 und 3x3 Filtern eingesetzt. Es wird versucht, die Eingabe in die 3x3 Filter zu reduzieren, um die Anzahl an Parametern gering zu halten. Um jedoch eine gute Genauigkeit beizubehalten, werden zum Schluss die 3x3 Filter benutzt.

Das Modell von SqueezeNet ist 4,8 Megabyte groß und nach der Kompression 0,47 Megabyte. Somit wird das Modell insgesamt 510 mal kleiner als das Modell von AlexNet (240 Megabyte ohne Kompression) und kann trotzdem mit der Genauigkeit mithalten.

[9]Bildquelle: `https://arxiv.org/pdf/1602.07360.pdf`

4 Implementierung

Der folgende Abschnitt beschreibt die konkrete Projektphase, welche für diese Thesis durchgeführt wird. Das Projekt hat zum Ziel, die Komplexität von Maschinellem Lernen vereinfacht anhand des bekannten Spiels „Stein, Schere, Papier“ mit der Erweiterung um „Spock und Echse“ zu erläutern, um für den Anwender ein greifbares und nachvollziehbares Beispiel der daraus hervorgehenden Möglichkeiten aufzuzeigen.

Die Verwendung des Raspberry Pis soll veranschaulichen, dass man für Maschinelles Lernen keine Großrechner benötigt. Auf dem Linux basierten Betriebssystem lassen sich Bibliotheken einfach installieren. Die meisten Anleitungen basieren auf Linux und setzen es auch als System voraus. Mit einer USB-Kamera ausgestattet, soll es dem Raspberry Pi 4 möglich sein, Übertragungen direkt festzuhalten.

4.1 Lösungsansatz Tensorflow

Um die Maschine zu trainieren, wird die Tensorflow Anwendungsschnittstelle für Python verwendet. Tensorflow bietet direkt nach dem Herunterladen Möglichkeiten, einen Agenten zu trainieren. Zum Herunterladen beziehungsweise Klonen des Tensorflow Models benutzt man den Befehl:

```
1 $ git clone https://github.com/tensorflow/models
```

Die Objekterkennung und das Training der Maschine benötigen diverse Bibliotheken. Die quelloffenen Bibliotheken sind ausgestattet mit Algorithmen für Maschinelles Lernen und Maschinelles Sehen. Nachfolgend die Bibliotheken, welche zu installieren sind:

- Protobuf 3.0.0
- Python-tk
- Pillow 1.0
- lxml
- tf Slim
- Jupyter notebook
- Matplotlib
- Tensorflow (>=1.12.0)
- Cython
- contextlib2

- cocoapi
- opencv-python.

Die Installation unter Raspbian wird mit dem Befehl:

```
1 $ pip install
```

oder

```
1 $ sudo apt-get install
```

ausgeführt. Für die Installation von *COCO API* müssen die Daten manuell heruntergeladen werden, speziell die Version für Python. Das Herunterladen und Installieren kann auch mit dem Befehl:

```
1 $ git clone https://github.com/cocodataset/cocoapi.git
2 $ cd cocoapi/PythonAPI
3 $ make
4 $ cp -r pycocotools <verzeichnis_zu_tensorflow>/models/research/
```

erfolgen. Bevor Tensorflow genutzt werden kann ist es notwendig, die Protobuf Bibliothek zu kompilieren. Das Ausführen des Befehls

```
1 $ protoc object_detection/protos/*.proto --python_out=.
```

im tensorflow/models/research Verzeichnis ist hierfür nötig.

Anschließend müssen die Ordner *research*, *object_detection* und *slim* in den Systemumgebungsvariablen mit folgenden Befehlen hinzugefügt werden:

```
1 $ export PYTHONPATH=$PYTHONPATH:<PFAD_ZU_TF>/TensorFlow/models/
    research
2 $ export PYTHONPATH=$PYTHONPATH:<PFAD_ZU_TF>/TensorFlow/models/
    research/object_detection
3 $ export PYTHONPATH=$PYTHONPATH:<PFAD_ZU_TF>/TensorFlow/models/
    research/slim
```

Im den Verzeichnissen /TensorFlow/models/research/slim und /TensorFlow/models/research ist die Ausführung der jeweiligen *setup.py* Dateien notwendig. Für die Installation werden die Befehle

```
1 $ python setup.py build
2 $ python setup.py install
```

in der Konsole ausgeführt.

Nach der Installation der Tensorflow-Anwendungsschnittstelle benötigt es für das Trainieren des Agenten einen Datensatz. Dafür werden im Vorfeld Fotos unter verschiedenen Lichtbedingungen und Umgebungen gemacht. Zum Testen werden circa 100 Bilder von jeweils einem Handzeichen gemacht. Da die Bilder mit einem Smartphone gemacht werden und die voreingestellte Auflösung zu groß ist (4032x3024 Pixel), ist zusätzlich ein Python-Skript[24] in Verwendung, um die Auflösung zu reduzieren.

```
1  from PIL import Image
2  import os
3  import argparse
4
5  def rescale_images(directory, size):
6          for img in os.listdir(directory):
7                  im = Image.open(directory+img)
8                  im_resized = im.resize(size, Image.ANTIALIAS)
9                  im_resized.save(directory+img)
10 if __name__ == '__main__':
11         parser = argparse.ArgumentParser(description="Rescale
               images")
12         parser.add_argument('-d', '--directory', type=str,
               required=True, help='Directory containing the images')
13         parser.add_argument('-s', '--size', type=int, nargs=2,
               required=True, metavar=('width', 'height'), help='
               Image size')
14         args = parser.parse_args()
15         rescale_images(args.directory, args.size)
```

Listing 2: Das Python-Skript `transform_image_resolution.py` zur Reduzierung der Auflösung

Der folgende Befehl reduziert die Auflösung auf 800x600 Pixel.

```
1 $ python transform_image_resolution.py -d images/ -s 800 600
```

Anschließend erfolgt die Kategorisierung der Bilder. Zum Kategorisieren wird die Software Labelimg[26] verwendet. Die Software kennzeichnet die Handzeichen mit einem halbtransparenten Kasten. Der Kasten erhält ein Label (Stein, Schere oder Papier). So muss jedes Bild gekennzeichnet werden.

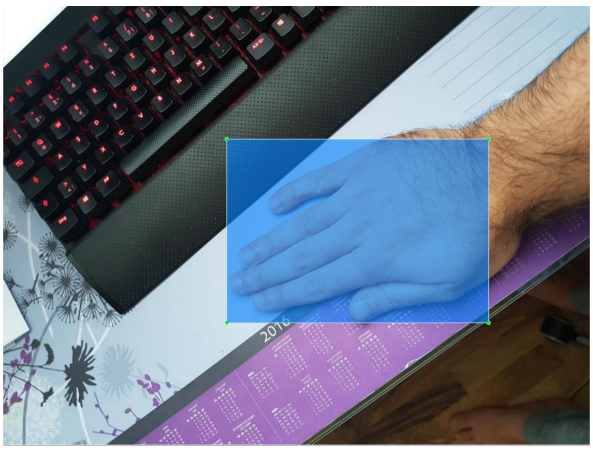

Abbildung 12: Kennzeichnung der Bilder mit der Software Labelimg

[9]Python-Skript Quelle: `https://github.com/datitran/raccoon_dataset`

Nach der Kennzeichnung aller Bilder, erfolgt die Aufteilung in zwei Gruppen. Die erste Gruppe, die 80 Prozent der Bilder enthält, wird zum Trainieren verwendet, die restlichen Bilder dienen als Testbilder.

Im nächsten Schritt müssen die XML Dateien in eine CSV Datei konvertiert werden. Die Konvertierung erfolgt mit dem Python-Skript *xml_to_csv.py*, jedoch mit folgender Änderung:

```
# Alt:
def main():
        image_path = os.path.join(os.getcwd(), 'annotations')
        xml_df = xml_to_csv(image_path)
        xml_df.to_csv('raccoon_labels.csv', index=None)
        print('Successfully converted xml to csv.')
# Neu:
def main():
        for folder in ['train', 'test']:
                image_path = os.path.join(os.getcwd(), ('images/'
                    + folder))
                xml_df = xml_to_csv(image_path)
                xml_df.to_csv(('images/'+folder+'_labels.csv'),
                    index=None)
        print('Successfully converted xml to csv.')
```

Listing 3: Das Python-Skript `xml_to_csv.py` zur Konvertierung der XML Dateien in CSV Dateien

Das Skript geht beide Ordner durch und speichert am Ende zwei CSV Dateien ab, *train_labels.csv* und *test_labels.csv*.

Tensorflow braucht sogenannte TFRecord Dateien, um einen Agenten zu trainieren. TFRecord Dateien sind das native Tensorflow-Binärformat zum Speichern von Daten. Um die TFrecord Dateien zu erhalten, nutzt man ein Python-Skript. Vor der Ausführung des Skriptes müssen noch einige Zeilen verändert werden. Damit das Skript funktioniert, müssen die richtigen Label angegeben werden, nämlich jene, die beim Kennzeichnen der Bilder vergeben wurden.

[9]Python-Skript Quelle: `https://github.com/datitran/raccoon_dataset`

```
def class_text_to_int(row_label):
        if row_label == 'paper':
                return 1
        elif row_label == 'rock':
                return 2
        elif row_label == 'scissors':
                return 3
        else:
                return None
```

Listing 4: Das Python-Skript `generate_tfrecord.py` zum Generieren der Tfrecord Dateien

Nach der Durchführung der Änderungen generiert man mit dem Befehlen

```
$ python generate_tfrecord.py --csv_input=images\train_labels.csv
    --image_dir=images\train --output_path=train.record
$ python generate_tfrecord.py --csv_input=images\test_labels.csv
    --image_dir=images\test --output_path=test.record
```

folgende zwei TFRecord Dateien: *train.record* und *test.record*.

Im nächsten Schritt benötigt es eine sogenannte „label map“. In dieser Datei wird jedem Namen eine ID zugeordnet. Die hier zugeordneten IDs müssen, wie auch im vorherigen Skript *generate_tfrecord.py* angegeben, vergeben werden.

```
item {
        id: 1
        name: 'paper'
}
item {
        id: 2
        name: 'rock'
}
item {
        id: 3
        name: 'scissors'
}
```

Listing 5: Die Datei `label_map.pbtxt` zur Zuordnung von IDs

[9]Python-Skript Quelle: `https://github.com/datitran/raccoon_dataset`

Im letzten Schritt wird die Konfigurationsdatei benötigt. Wie im Abschnitt MASK-RCNN beschrieben wird unter diesen Bedingungen das Modell *ssd_mobilenet_v2_coco* verwendet. Im Verzeichnis des Modells befindet sich eine Konfigurationsdatei. Es ist erforderlich, diese Datei auf den neuen Datensatz anzupassen. Nachfolgend ist beschrieben, wie die Anpassung erfolgt:

- Die Zahl der Klassen reduziert sich von 90 auf die Anzahl an Objekten, die erkannt werden sollen. Hier sind es drei, nämlich Stein, Schere und Papier.
- Unter *fine_tune_checkpoint* wird das Verzeichnis angegeben, in welches die model.ckpt-Datei kopiert wurde.
- Unter *input_path* wird das Verzeichnis der Datei *train.record* und unter *label_map_path* das Verzeichnis der *labelmap.pbtxt*-Datei angegeben.
- In der Klasse *eval_input_reader* muss der Vorgang wiederholt werden, diesmal aber mit der Datei *test.record*.
- In der Klasse *eval_config* unter *num_examples* wird die Menge an Testbildern angegeben, also 71.

Beim Versuch das Training zu starten, friert der Raspberry Pi ein, weil der Raspberry Pi nicht genug Systemspeicher zu Verfügung hat. Deswegen wurde das Training auf einen Desktop-Computer ausgelagert.
Zum Installieren der Anforderungen wurde mit der Software *Anaconda Navigator[33]* alles noch einmal aufgesetzt. Durch die höhere Rechenleistung kann man auch die Konfigurationsdatei *ssd_mobilenet_v2_coco* mit *faster_rcnn_inception_v2_pets.config* ersetzen. Nach der Installation kann man die Datei *train.py* mit dem Befehl:

```
1 $ python train.py --logtostderr --train_dir=training/ --
      pipeline_config_path=training/pipeline.config
```

ausführen. Nach dem Ausführen wird in der Konsole der Loss-Wert und die Anzahl von Schritten angezeigt (siehe Abbildung 13).

```
INFO:tensorflow:Recording summary at step 0.
I0120 12:25:21.567651  8384 supervisor.py:1050] Recording summary at step 0.
INFO:tensorflow:global step 1: loss = 2.5321 (24.652 sec/step)
I0120 12:25:26.203200 14764 learning.py:507] global step 1: loss = 2.5321 (24.652 sec/step)
INFO:tensorflow:global step 2: loss = 1.8821 (5.840 sec/step)
I0120 12:25:32.253417 14764 learning.py:507] global step 2: loss = 1.8821 (5.840 sec/step)
INFO:tensorflow:global step 3: loss = 2.8770 (6.126 sec/step)
I0120 12:25:38.381385 14764 learning.py:507] global step 3: loss = 2.8770 (6.126 sec/step)
INFO:tensorflow:global step 4: loss = 1.9782 (5.814 sec/step)
I0120 12:25:44.199047 14764 learning.py:507] global step 4: loss = 1.9782 (5.814 sec/step)
INFO:tensorflow:global step 5: loss = 1.9884 (5.949 sec/step)
I0120 12:25:50.151149 14764 learning.py:507] global step 5: loss = 1.9884 (5.949 sec/step)
INFO:tensorflow:global step 6: loss = 1.2789 (5.999 sec/step)
I0120 12:25:56.153027 14764 learning.py:507] global step 6: loss = 1.2789 (5.999 sec/step)
INFO:tensorflow:global step 7: loss = 1.6821 (5.795 sec/step)
I0120 12:26:01.949838 14764 learning.py:507] global step 7: loss = 1.6821 (5.795 sec/step)
INFO:tensorflow:global step 8: loss = 1.6821 (6.071 sec/step)
I0120 12:26:08.023815 14764 learning.py:507] global step 8: loss = 1.6821 (6.071 sec/step)
INFO:tensorflow:global step 9: loss = 2.4036 (5.762 sec/step)
I0120 12:26:13.786450 14764 learning.py:507] global step 9: loss = 2.4036 (5.762 sec/step)
INFO:tensorflow:global step 10: loss = 0.7316 (5.793 sec/step)
I0120 12:26:19.582898 14764 learning.py:507] global step 10: loss = 0.7316 (5.793 sec/step)
INFO:tensorflow:global step 11: loss = 1.4541 (6.157 sec/step)
I0120 12:26:25.742303 14764 learning.py:507] global step 11: loss = 1.4541 (6.157 sec/step)
INFO:tensorflow:global step 12: loss = 0.6520 (5.842 sec/step)
I0120 12:26:31.585615 14764 learning.py:507] global step 12: loss = 0.6520 (5.842 sec/step)
INFO:tensorflow:global step 13: loss = 1.3957 (5.636 sec/step)
I0120 12:26:37.223468 14764 learning.py:507] global step 13: loss = 1.3957 (5.636 sec/step)
INFO:tensorflow:global step 14: loss = 0.8844 (6.040 sec/step)
I0120 12:26:43.265877 14764 learning.py:507] global step 14: loss = 0.8844 (6.040 sec/step)
```

Abbildung 13: Trainingsablauf

Alle fünf Minuten wird ein Checkpoint mit dem momentanen Loss-Wert gesetzt. In einer zweiten Konsole wird mit dem Befehl

```
1 $ tensorboard --logdir=training
```

der Trainingsverlauf veranschaulicht. Der Reiter *loss* zeigt eine Zusammenfassung des Trainingsvorgangs. Der Loss-Wert liegt nach ungefähr 2848 Schritten bei 0,050, das bedeutet bei einer Fehlerrate von 5 Prozent.

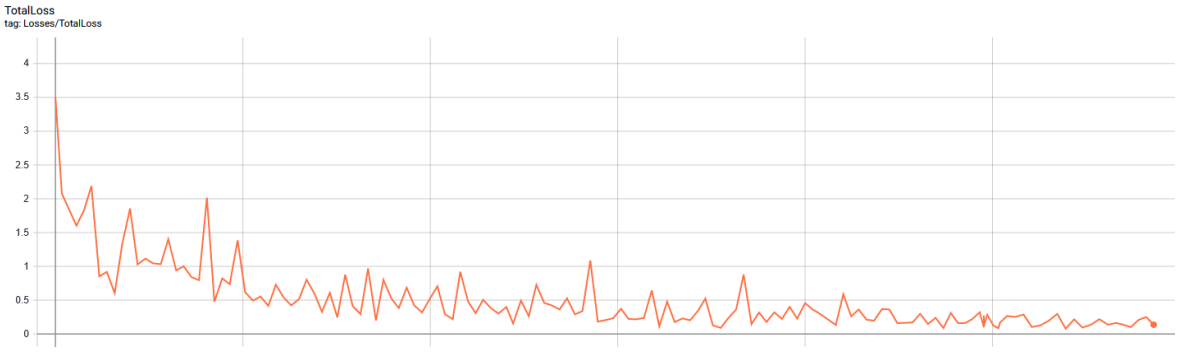

Abbildung 14: Graph zum Trainingsablauf

Eine Fehlerrate von 5 Prozent ist ideal zum Testen. Bevor das Modell jedoch getestet werden kann, muss ein Inferenzgraph generiert werden. Um einen solchen Graphen zu erstellen, nimmt man den letzten Checkpoint und generiert mit dem folgenden Befehl den Graphen:

```
$ python export_inference_graph.py --input_type image_tensor --
    pipeline_config_path training/faster_rcnn_inception_v2_pets.
    config --trained_checkpoint_prefix training/model.ckpt-2848 --
    output_directory inference_graph
```

Für das Testen des Modells sind mehrere Python-Skripte im Tensoflow-Verzeichnis *Object_detection_webcam.py*, *Object_detection_image.py* und *Object_detection_video.py* hinterlegt.
Während der Testphase hat der Agent Probleme beim Erkennen der Handzeichen.

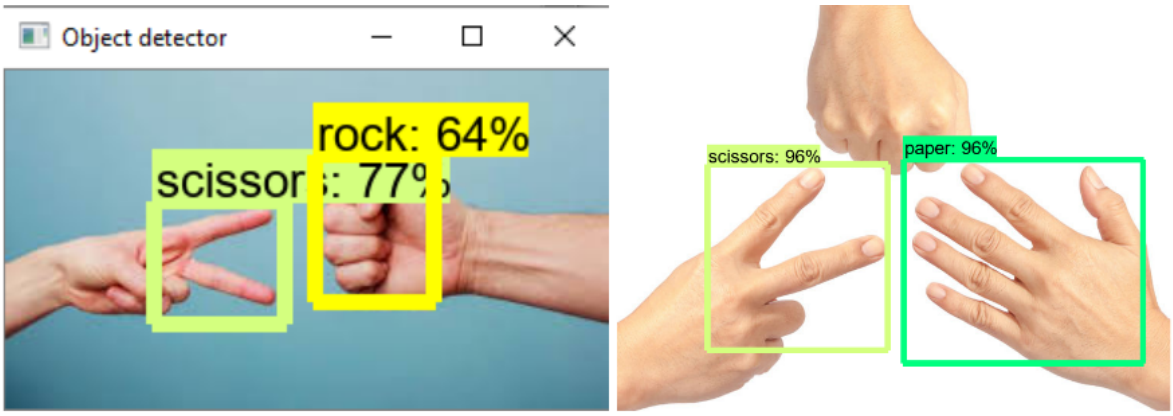

Abbildung 15: Modelltest mit Bildern

Bei Direktaufnahmen ist die Bildrate von zwei Bildern in der Sekunde zu gering. Der Raspberry Pi hat nicht genug Rechenleistung, um eine flüssige Bildrate zu erzeugen. Um die Bildrate und Genauigkeit zu erhöhen, wird eine Alternative verwendet, nämlich das Framework Keras. Keras bietet für mobile Geräte die Möglichkeit, mit SqueezeNet ein mehrschichtiges neuronales Netz aufzubauen, ohne Verlust von Genauigkeit und mit weniger Rechenleistung.

4.2 Verwendung von Keras und SqueezeNet

Die Alternative zur Tensorflow-Anwendungsschnittstelle ist Keras mit SqueezeNet. Der GitHub Benutzer SouravJohar hat einen Stein-Schere-Papier Agenten mit Keras entwickelt. Das Projekt dient als Inspiration, diese Arbeit mit SqueezeNet weiterzuentwickeln. In SouravJohar's Projekt geht es um das Erkennen der Handzeichen Stein, Schere und Papier. Dafür benutzt er einen Desktop Computer mit einem auf Linux basierten Betriebssystem. In dem Projekt werden die offenen Bibliotheken OpenCV, Tensorflow und Keras verwendet.

[9]Bildquelle: `https://stock.adobe.com/de/contributor/203803473/lolostock`
[9]Bildquelle: `https://www.shutterstock.com/de/g/somyot+pattana`

Um die Ausführung zu ermöglichen, sind folgende Anforderungen zu installieren:

- absl-py==0.7.1
- astor==0.8.0
- gast==0.2.2
- grpcio==1.21.1
- h5py==2.9.0
- joblib==0.13.2
- Keras==2.1.1
- Keras-Applications==1.0.7
- Keras-Preprocessing==1.0.9
- keras-squeezenet==0.4
- Markdown==3.1.1
- mock==3.0.5
- numpy==1.16.3
- opencv-python==4.1.0.25
- protobuf==3.7.1
- PyYAML==5.1
- scikit-learn==0.21.2
- scipy==1.3.0
- six==1.12.0
- tensorboard==1.13.1
- tensorflow==1.13.1
- tensorflow-estimator==1.13.0
- termcolor==1.1.0
- Werkzeug==0.15.4

Zum Ausführen des Projektes benötigt es zunächst einen Datensatz. Im Verzeichnis ist ein Python-Skript mit dem Namen *gather_images.py* abgelegt. Das Skript verwendet OpenCV, um Bilder aufzunehmen und abzuspeichern. Zum Ausführen des Skriptes verwendet man den Befehl

```
1 $ python gather_images.py <name> <zahl der bilder die aufgenommen
      werden sollen>
```

Nach dem Ausführen öffnet sich ein Fenster mit einer Direktübertragung der Kamera. In dem Fenster ist ein Kasten abgebildet. Durch das Drücken der „A“-Taste wird ein Ordner erstellt und die Anzahl an Bildern abgespeichert, die vorher im Befehl definiert wurden. Jedoch wird nur das gespeichert, was sich innerhalb des Kastens befindet. Am oberen linken Fensterrand erscheint eine Anzeige mit der Anzahl der erzeugten Bilder.

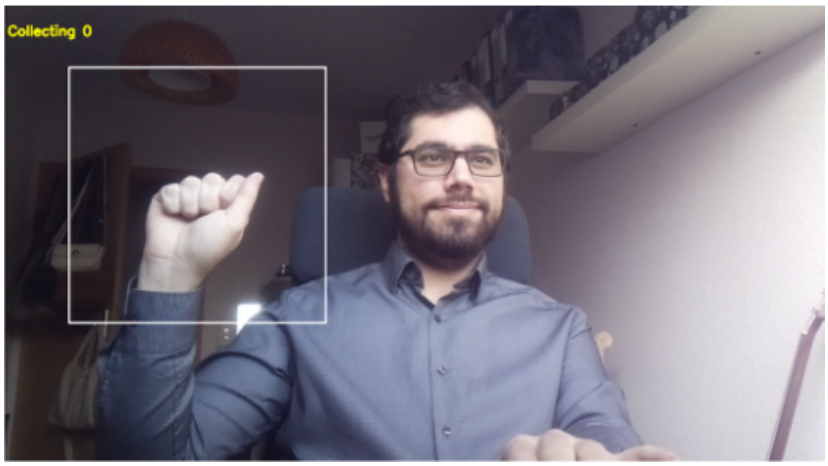

Abbildung 16: Python-Skript *gather_images.py* zum Speichern von Bildern

Im ersten Durchgang werden 200 Bilder des jeweiligen Handzeichens abgespeichert. Damit der Agent den Hintergrund nicht auch klassifiziert, werden 200 Bilder vom Hintergrund mit verschiedenen Gegenständen im Bild unter dem Begriff *None* abgespeichert. Der Agent sollte nun die Bilder als *None* klassifizieren, bei denen er kein Handzeichen erkennt. Alle Bilder werden mit einer Auflösung von 277x277 Pixel abgespeichert. SqueezeNet akzeptiert hier keine andere Auflösung.

Im Anschluss erfolgt die Ausführung der Pythondatei *train.py*. Im Vorfeld muss die Anpassung der Datei an zwei Stellen erfolgen. Zur Differenzierung der verschiedenen Handzeichen wird ein *Dictionary* mit dem Namen *CLASS_MAP* verwendet. Dictionaries sind assoziative Felder die aus Schlüssel-Objekt-Paaren bestehen.

```
CLASS_MAP = {
        "rock": 0,
        "paper": 1,
        "scissors": 2,
        "none": 3,
        "spock": 4,
        "echse": 5
}
```

Listing 6: Das *Dictionary CLASS_MAP* im Python-Skript `train.py`.

Dieses Dictionary besteht aus sechs unterschiedlichen Schlüsseln, nämlich rock, paper, scissors, none, spock und echse. Allen Schlüsseln wird ein Wert von null bis fünf zugeordnet. Die nächste Stelle, die zu bearbeiten ist, ist der Trainingsvorgang. Beim Trainieren lädt man alle Bilder in den Arbeitsspeicher. Das ist beim Raspberry Pi 4 mit 800 Bildern problemlos möglich. Um aber die Handzeichen besser zu erkennen, benötigt es mehr als 800 Bilder. Die Bilder werden in sogenannte *Batches* eingeteilt. Das heißt die Bilder werden mit folgendem Befehl in Gruppen geladen.

```
model.fit(np.array(data), np.array(labels), batch_size=20,
    validation_split=0.2, epochs=10)
```

Listing 7: Das Python-Skript `train.py`. Regulierung von geladenen Bildern mithilfe von *batch_size und validation_split* zum Bewerten des Modells

Bei einem Datensatz von über 1.000 Bildern gibt es die Fehlermeldung, dass der Systemspeicher nicht ausreicht und das Programm beendet wird. Um dieses Problem zu beheben, fügt man hier noch den Parameter *batch_size=20*[7] hinzu. Nach diesen Änderungen kann *train.py* mit dem Befehl

```
$ python3 train.py
```

ausgeführt werden. Jeder Epoch dauert bei 3.200 Bildern mit *batch_size=20* ungefähr 15 Minuten. In jedem Epoch wird jedes Bild durch das künstliche neuronale Netz vorwärts und dann rückwärts gesendet. Während des Trainings des Modells wird schnell deutlich, dass das Modell overfitted ist, da die Genauigkeit innerhalb vier Epochs bei 100 Prozent liegt und sich der Loss-Wert im Minusbereich befindet. Um dem entgegenzuwirken, muss man weitere versteckte Schichten implementieren, die Lernrate weiter reduzieren sowie zwei weitere Dropouts[3.4] einfügen. Zum Bewerten des Modells wird der Datensatz in *training* und *test* aufgeteilt.

Das Problem ist, dass beim Trennen der Daten die letzten 20 Prozent genommen und somit ein bis zwei Kategorien als Testdaten benutzt werden. Nach dem Vermischen des Datensatzes werden diese nun problemlos getrennt und zeigen beim Trainieren keine Genauigkeit von Null mehr an. Die Loss-Werte[3.4] gehen in kleineren Schritten herunter anstatt sich zu erhöhen und bei der Bewertung mit den Testdaten zeigt sich eine langsame Besserung des Loss-Wertes und der Genauigkeit.

Damit das Modell auch in anderen Umgebungen und Lichtbedingungen besser funktioniert, verwendet man die Klasse *ImageDataGenerator*. Die Klasse hilft bei kleineren Datensätzen Bilder zu editieren, bevor diese durch das neuronale Netz gesendet werden.

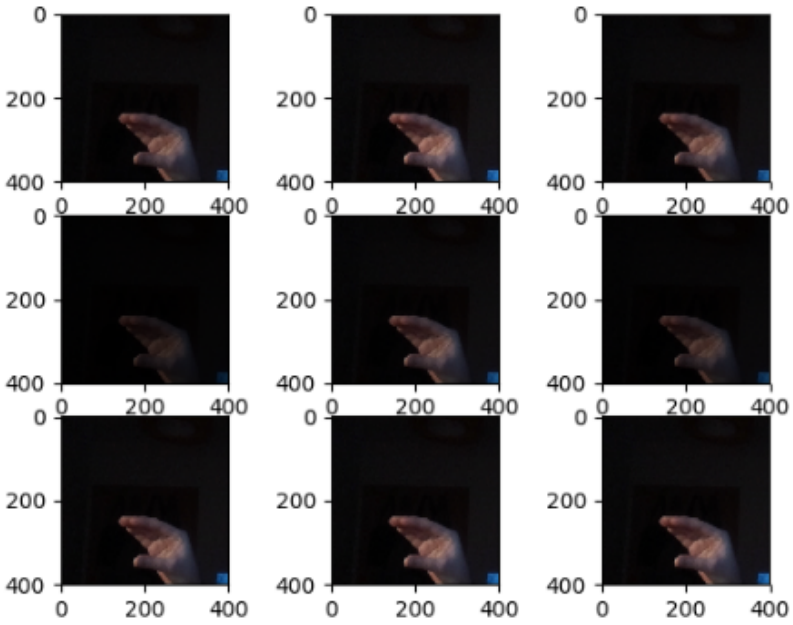

Abbildung 17: Veränderung der Helligkeit mit der *ImageDataGenerator* Klasse

Die Klasse bietet Funktionen zum Rotieren, Spiegeln, Zoomen oder zur Veränderung der Helligkeit von Bildern (siehe Abbildung 17). Nach dem Editieren werden so aus einem Bild beispielsweise neun verschiedene Bilder und das Modell setzt sich mit Daten, welche sich nicht sehr ähnlich sind, auseinander.

Nach dem Trainieren des Agenten speichert man das Modell ab und kann es anschließend verwenden.

Bevor der Agent eingesetzt werden kann, muss die Datei *play.py* ebenfalls angepasst werden. Beim Trainieren wird der Datensatz so bearbeitet, dass kategoriale Variablen nicht mehr vorkommen, da die Algorithmen nur Zahlen akzeptieren. Deswegen werden die Labels Stein, Schere, Papier, Echse, Spock und None nummeriert. Dies wird mit der Zeile

```
labels = np_utils.to_categorical(labels)
```

Listing 8: Das Python-Skript `train.py`. Entfernen von kategorischen Werten

erreicht. Die Zuweisung von Identifikatoren wird auch *Label Encoding* genannt. Die Tabelle 3 zeigt beispielhaft, wie so etwas aussehen kann.

Label	ID	Kodierung
Stein	0	001
Papier	1	010
Schere	2	011
None	3	100
Spock	4	101
Echse	5	110

Tabelle 3: Beispiel: Labelkodierung

Die Zeile sowie die Spalte „Label“ dienen lediglich der Leserlichkeit. Nach der Zuweisung besteht die Tabelle nur noch aus Zahlen. Bei der Objekterkennung muss man dies nun wieder rückgängig machen. Dafür erstellt man das Dictionary, *REV_CLASS_MAP*.

```
REV_CLASS_MAP = {
        0: "rock",
        1: "paper",
        2: "scissors",
        3: "none",
        4: "echse",
        5: "spock"
}
```

Listing 9: Dekodierung der Labels

Hier wird den IDs wieder das dazugehörige Label zugeordnet. Anschließend muss man die Spielstruktur erweitern. Jedes Handzeichen hat zwei Gewinnmöglichkeiten und zwei Möglichkeiten zu verlieren.

Die Spielregeln, welche hier noch zusätzlich implementiert werden müssen, lauten:

- Spock gewinnt gegen Schere und Stein,
- Schere gewinnt gegen Echse,
- Papier gewinnt gegen Spock,
- Stein gewinnt gegen Echse,
- Echse gewinnt gegen Papier und Spock.

Zum Beispiel wird das Handzeichen Spock mit *if*-Abfragen und dem Vergleichsoperator so realisiert, dass jede Möglichkeit abgedeckt ist.

```
1 if move1 == "spock":
2         if move2 == "paper":
3                 return "Computer"
4         if move2 == "rock":
5                 return "User"
6         if move2 == "scissors":
7                 return "User"
8         if move2 == "echse":
9                 return "Computer"
```

Listing 10: Abdeckung der möglichen Züge für das Handzeichen Spock

Im oben gezeigten Beispiel steht die Variable *move1* für die Hand des Spielers, *move2* für die Hand des Agenten. Mit dem Handzeichen Spock gewinnt der Spieler nur dann, wenn der Agent die Handzeichen Stein oder Schere benutzt.

Im nächsten Schritt ermöglicht man es dem Agenten, die neuen Handzeichen zu benutzen. Dafür musste die Zeile

```
1 computer_move_name = choice(['rock', 'paper', 'scissors','echse' ,
    'spock'])
```

Listing 11: Das Python-Skript `play.py`ermöglicht dem Agenten die neuen Handzeichen zu benutzen

mit den neuen Handzeichen Echse und Spock erweitert werden.

Beim Versuch die Datei *play.py* auszuführen, wird der Fehler

```
1 $ ValueError: could not broadcast input array from shape
    (400,400,3) into shape (380,0,3)
```

ausgegeben.

Dieser Fehler erscheint, da die Auflösung der Kamera nicht angegeben wird. Um eine Auflösung von 1280x720 zu erhalten, werden die zwei Zeilen

```
1 cap.set(3,1280)
2 cap.set(4,720)
```

Listing 12: Das Python-Skript `play.py`. Bildschirmauflösung anpassen

hinzugefügt. Beim Ausführen der Datei *play.py* öffnet sich nun ein Fenster mit zwei Kästen. Im linken Kasten werden die Handzeichen des Spielers erfasst, im Rechten werden die Handzeichen des Agenten, hier Computer genannt, angezeigt (siehe Abbildung 18).

Abbildung 18: Demonstration des Spiels

Über dem Kasten des Spielers wird das Handzeichen angezeigt, welches erkannt wurde und über dem Kasten des Agenten, das Handzeichen mit dem dieser erwidert. Der Gewinner wird unten in roter Schrift angezeigt.

5 Bewertung

In diesem Kapitel werden die beiden beschriebenen Lösungsansätze noch einmal begutachtet. Die Probleme, die entstanden sind sowie deren Lösungen werden hier noch einmal kurz beschrieben und bewertet.

5.1 Lösungsansatz mit der Tensorflow-Anwendungsschnittstelle

Die Tensorflow Anwendungsschnittstelle zeigt sowohl positive als auch negative Resultate. Tensorflow bietet durch eine detaillierte Dokumentation eine gute Hilfestellung und Basis für dieses Projekt. Auch durch die weite Verbreitung stehen viele Anleitungen zum Programmierern zur Verfügung.

Jedoch werden in diesen Anleitungen meist alte Versionen von Tensorflow verwendet, zum Beispiel Tensorflow 1.14 oder 1.13. Da bei der Bearbeitung dieses Projekts die Version 2.0 genutzt wird, sind viele Klassen und Funktionen umbenannt und geben oftmals Fehler aus. Bei der Verwendung von Tensorflow 1.14 muss man bei der Installation der anderen Bibliotheken darauf achten, dass diese auch kompatibel sind.

Dadurch, dass alle Bilder beim Erstellen eines Modells gelabelt werden müssen, entsteht ein erheblicher Zeitaufwand. Bei einem Datensatz von 300 Bildern ist der Zeitaufwand noch vertretbar, jedoch bei über 1.000 Bildern nicht mehr zu empfehlen. Beim Testen des Modells fällt schnell auf, dass der Raspberry Pi der großen Datenmenge nicht standhalten kann. Es kommt zum Speicherüberlauf und das Modell kann nicht trainiert werden. Somit erfolgte das Training des Modells auf einem Desktop-Computer. Ausgestattet mit 16 Gigabyte Arbeitsspeicher und einem Intel Core i5-2400 Prozessor, wurde das Modell 36 Stunden lang trainiert. Anschließend wurde das Modell auf den Raspberry Pi übertragen und getestet.

Die Bildrate sinkt auf zwei Bilder in der Sekunde. Beim Einsatz der USB-Kamera läuft zwar das Programm aber das Gerät reagiert nicht mehr. Um das Programm zu schließen, muss die Kamera entfernt werden, damit das Programm abstürzt. Alternativ kann der Raspberry Pi neu gestartet werden. Während das Programm läuft, liefert der Agent auch mit 300 Bildern Ergebnisse und erkennt in einigen Fällen die Handzeichen Stein, Schere und Papier. Durch das Erhöhen des Datensatzes würde das Modell wahrscheinlich genauer werden, es wurde jedoch wegen der Bildrate und den Problemen beim Trainieren nach anderen Lösungsansätzen gesucht.

5.2 Lösungsansatz mit Keras und SqueezeNet

Keras in Verbindung mit SqueezeNet zeigt erhebliche Fortschritte im Vergleich zur Tensorflow Anwendungsschnittstelle. Das Framework Keras bietet ein großes Repertoir an Hilfsmitteln und eine einfache Möglichkeit diese einzubinden.

Beim Trainieren des Modells hat der Raspberry Pi oft die Fehlermeldung ausgegeben, dass der Systemspeicher überlaufen ist. Dies hat zur Folge, dass der Raspberry Pi einfriert und nicht mehr reagiert. Dem konnte damit entgegengewirkt werden, dass Gruppierungen von jeweils 20 Bildern aus dem gesamten Datensatz geladen wurden. Generell sollte man die Empfehlung aussprechen, dass während des Trainings keine weiteren Programme auf dem Raspberry Pi laufen, da dieser mit dem Training vollkommen ausgelastet ist.

Es kam beim Trainieren weiterhin zu dem Problem von overfitting. Um dies zu lösen, wurden Dropouts und weitere versteckte Schichten implementiert. Auch der Datensatz wurde von den anfänglichen 800 Bildern auf 6.000 Bilder erhöht. Die Trainingsphase hat bei 800 Bildern ungefähr vier Minuten pro Epoch gedauert. Bei einem Datensatz von 6.000 Bildern hat es am Ende pro Epoch ungefähr 23 Minuten gedauert. Bei dem Versuch das Modell mit der Klasse *ImageDataGenerator* zu verbessern ist aufgefallen, dass es leider zu keiner Verbesserung geführt hat. Auch hatte die Klasse Probleme, mit der Größe des Datensatzes ab 1.000 Bildern. Aus diesem Grund wurde die Klasse wieder gestrichen.

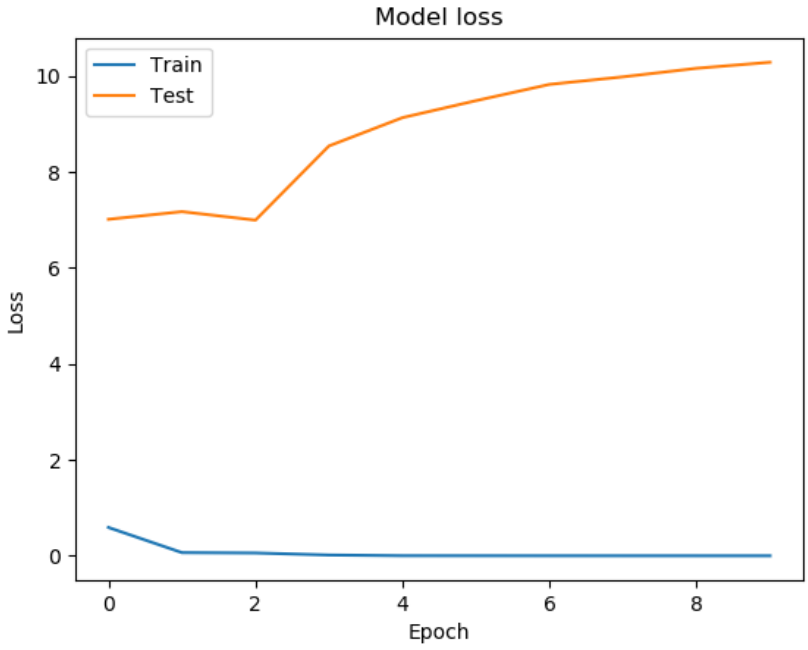

Abbildung 19: SqueezeNet: overfitting Problem anhand des Loss-Wertes

Da der Datensatz immer in Training und Test aufgeteilt wurde, kam es beim Testen immer wieder vor, dass ein Handzeichen, auch nach dem Mischen nicht erkannt wurde. Deswegen musste der Datensatz auf 6.000 Bilder erhöht werden. Dies erwies sich für die Testzwecke als ausreichend.

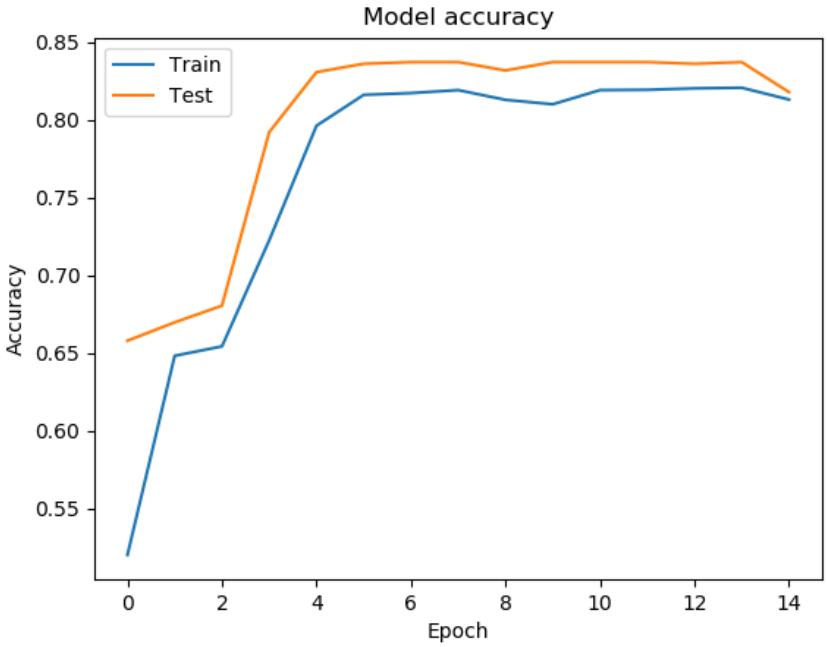

Abbildung 20: Verbesserung der Genauigkeit nach Umgestaltung des Modells

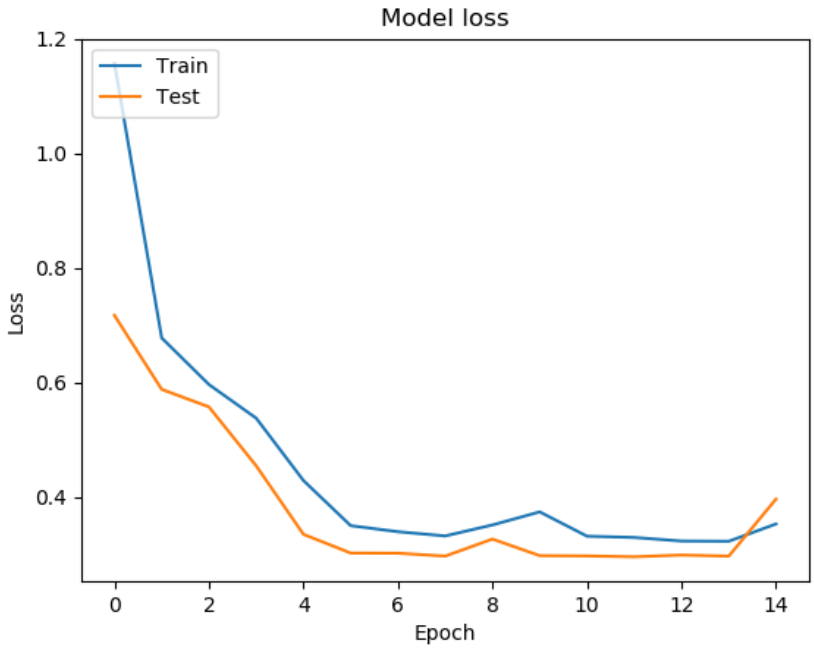

Abbildung 21: Verbesserung des Loss-Wertes nach der Umgestaltung des Modells

Das Modell konnte am Ende des Prozesses auch Bilder erkennen, die nicht im Datensatz vorhanden waren, vorausgesetzt sie hatten in etwa das gleiche Format.

Beim Testen des Spiels erkennt der Agent nun die Handzeichen des Spielers, erwidert mit einem eigenen Handzeichen und das Spiel läuft flüssig mit einer Bildrate von 32 Bildern pro Sekunde.

6 Ausblick

Da die Tensorflow-Anwendungsschnittstelle mehr Leistung verlangt, als der Raspberry Pi 4 bieten kann, könnte das Projekt auch noch einmal mit dem Framework Tensorflow lite ausprobiert werden.

Vorstellbar wäre auch eine Erweiterung, welche die Vorhersage des Computers verbessert. Dafür könnte man versuchen, Muster im Spielstil zu erkennen. Dies könnte man erreichen, indem die Handzeichen und das Ergebnis bei jedem Spielzug in einer Tabelle gespeichert werden. Hieraus könnten Muster abgeleitet werden, um Vorhersagen zu treffen, die die Maschine öfter gegen den Spieler gewinnen lassen.

Die Nutzung des Raspberry Pi 4 zeigt, dass dieser imstande ist, Modelle zu bauen und flüssig wiederzugeben. Mit diesem Wissen kann der Quellcode so verändert werden, dass auch ganz andere Projekte daraus entstehen können. Das Projekt kann beispielsweise die Basis sein, Gesichter oder sogar Tiere zu erkennen, wenn der Datensatz groß genug ist.

Literaturverzeichnis

[1] Wikipedia: *„Schere, Stein, Papier“*.
https://de.wikipedia.org/wiki/Schere,_Stein,_Papier.
Version: 03.02.2020

[2] Amazon.com Inc.: *„Amazon Go“*.
https://www.amazon.com/b?ie=UTF8&node=16008589011.
Version: 03.02.2020

[3] Waymo: *„Das Waymo Auto“*.
https://waymo.com/. Version: 03.02.2020

[4] World Health Organisation: *„Global status report on road safety 2018“*.
https://www.who.int/violence_injury_prevention/road_safety_status/2018/en/#. Version: 03.02.2020

[5] GitHub: *„Stein-Schere-Papier Projekt“*.
https://github.com/DrGFreeman/rps-cv. Version: 03.02.2020

[6] SouravJohar: *„Keras und SqueezeNet Modell für Stein-Schere-Papier“*.
https://github.com/SouravJohar/rock-paper-scissors.
Version: 03.02.2020

[7] Wikipedia: *„Raspberry Pi“*.
https://de.wikipedia.org/wiki/Raspberry_Pi. Version: 05.02.2020

[8] Raspberrypi: *„Raspberry Pi 4 Tech Specs“*.
https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/. Version: 05.02.2020

[9] TeknoTUT: *„Facial Recognition with Raspberry Pi and OpenCV“*.
https://www.teknotut.com/en/facial-recognition-with-raspberry-pi-and-opencv/. Version: 05.02.2020

[10] OpenCV: *„About“*.
https://opencv.org/about/. Version: 05.02.2020

[11] Wikipedia: *„Python (Programmiersprache)“*.
https://de.wikipedia.org/wiki/Python_(Programmiersprache).
Version: 05.02.2020

[12] GitHub: *„The State of the Octoverse: machine learning“*.
https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/. Version: 05.02.2020

[13] BalenaEtcher: *„Software zum installieren von Betriebssystemen auf Speicherkarten“*.
https://www.balena.io/etcher/. Version: 03.02.2020

[14] Tensorflow:
https://www.tensorflow.org/. Version: 03.02.2020

[15] TENSORFLOW: *„TensorFlow 2: Convolutional Neural Networks (CNN) and Image Classification“*.
https://techbrij.com/tensorflow-cnn-image-classification. Version: 03.02.2020

[16] TOWARDSDATASCIENCE: *„Forward propagation in neural networks — Simplified math and code version“*.
https://towardsdatascience.com/forward-propagation-in-neural-networks-simplified-math-and-code-version-bbcfef6f9250. Version: 05.02.2020

[17] CORNELL UNIVERSITY, Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick: *„Mask R-CNN“*.
https://arxiv.org/abs/1703.06870. Version: 03.02.2020

[18] MASK-RCNN: *„Mask R-CNN for Object Detection and Segmentation“*.
https://modelzoo.co/model/mask-r-cnn-keras. Version: 03.02.2020

[19] TOWARDSDATASCIENCE: *„Common Loss functions in machine learning“*.
https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23. Version: 03.02.2020

[20] ELITE DATA SCIENCE: *„Overfitting in Machine Learning: What It Is and How to Prevent It“*.
https://elitedatascience.com/overfitting-in-machine-learning#how-to-prevent. Version: 03.02.2020

[21] KERAS: *„Keras: The Python Deep Learning library“*.
https://keras.io/. Version: 03.02.2020

[22] CORNELL UNIVERSITY, Forrest N. Iandola1, Song Han, Matthew W. Moskewicz1, Khalid Ashraf1, William J. Dally, Kurt Keutzer1: *„SqueezeNet: AlexNet-Level accuracy with 50X fewer Parameters and <0.5MB Model size“*.
https://arxiv.org/pdf/1602.07360.pdf. Version: 03.02.2020

[23] PYIMAGESEARCH: *„Installing Keras with TensorFlow backend“*.
https://www.pyimagesearch.com/2016/11/14/installing-keras-with-tensorflow-backend/. Version: 05.02.2020

[24] TOWARDSDATASCIENCE: *„Creating your own object detector“*.
https://towardsdatascience.com/creating-your-own-object-detector-ad69dda69c85. Version: 05.02.2020

[25] TOWARDSDATASCIENCE: *„Live Object Detection“*.
https://towardsdatascience.com/live-object-detection-26cd50cceffd. Version: 05.02.2020

[26] GITHUB: *„LabelImg Software“*.
https://github.com/tzutalin/labelImg. Version: 03.02.2020

[27] GITHUB: *„Python-Skripte zum erstellen von CSV und Tfrecord Dateien“*.
https://github.com/datitran/raccoon_dataset . Version: 03.02.2020

[28] GitHub: *„Cocoapi Datenset“*.
https://github.com/cocodataset/cocoapi. Version: 103.02.2020

[29] GitHub: *„Tensorflow detection model zoo“*.
https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. Version: 03.02.2020

[30] Medium: *„mAP (mean Average Precision) for Object Detection“*.
https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173. Version: 05.02.2020

[31] Wikipedia: *„Deep Learning“*.
https://de.wikipedia.org/wiki/Deep_Learning. Version: 05.02.2020

[32] Cornell University, Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, Serge Belongie: *„Feature Pyramid Networks for Object Detection“*.
https://de.wikipedia.org/wiki/Deep_Learning. Version: 09.02.2020

[33] Anaconda Navigator:
https://docs.anaconda.com/anaconda/navigator/. Version: 09.02.2020

Anhang

Der Anhang enthält den geschriebenen und benutzten Quellcode und andere Dateien die benötigt wurden für die Lösungsansätze.

Quellcode für den Lösungsansatz mit Tensorflow

transform_image_resolution.py

```
from PIL import Image
import os
import argparse

def rescale_images(directory, size):
  for img in os.listdir(directory):
        im = Image.open(directory+img)
        im_resized = im.resize(size, Image.ANTIALIAS)
        im_resized.save(directory+img)
if __name__ == '__main__':
        parser = argparse.ArgumentParser(description="Rescale  images")
        parser.add_argument('-d', '--directory', type=str, required=True,
            help='Directory containing the images')
        parser.add_argument('-s', '--size', type=int, nargs=2, required=
           True, metavar=('width', 'height'), help='Image size')
        args = parser.parse_args()
        rescale_images(args.directory, args.size)
```

xml_to_csv.py

```
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET


def xml_to_csv(path):
  xml_list = []
  for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
          value = (root.find('filename').text,
          int(root.find('size')[0].text),
          int(root.find('size')[1].text),
          member[0].text,
          int(member[4][0].text),
          int(member[4][1].text),
          int(member[4][2].text),
          int(member[4][3].text)
        )
        xml_list.append(value)
  column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin',
      'xmax', 'ymax']
  xml_df = pd.DataFrame(xml_list, columns=column_name)
  return xml_df
```

```
26
27 def main():
28   for folder in ['train', 'test']:
29         image_path = os.path.join(os.getcwd(), (folder))
30         xml_df = xml_to_csv(image_path)
31         xml_df.to_csv((folder+'_labels.csv'), index=None)
32         print('Successfully converted xml to csv.')
33
34
35 main()
```

generate_tfrecord.py

```
1 """
2 Usage:
3 # From tensorflow/models/
4 # Create train data:
5 python generate_tfrecord.py --csv_input=data/train_labels.csv --
      output_path=train.record
6
7 # Create test data:
8 python generate_tfrecord.py --csv_input=data/test_labels.csv --
      output_path=test.record
9 """
10 from __future__ import division
11 from __future__ import print_function
12 from __future__ import absolute_import
13
14 import os
15 import io
16 import pandas as pd
17 #import tensorflow as tf
18 import tensorflow as tf
19
20 from PIL import Image
21 from object_detection.utils import dataset_util
22 from collections import namedtuple, OrderedDict
23
24 flags = tf.app.flags
25 flags.DEFINE_string('csv_input', 'C:/Users/sevto/Documents/OpenCV/
      ba_arbeit/img/test_labels.csv', 'Path to the CSV input')
26 flags.DEFINE_string('output_path', 'test.record', 'Path to output
      TFRecord')
27 flags.DEFINE_string('image_dir', 'C:/Users/sevto/Documents/OpenCV/
      ba_arbeit/img/test', 'Path to images')
28 FLAGS = flags.FLAGS
29
30
31 # Label map mit eigenen Objekten/labeln definieren
```

```
32 def class_text_to_int(row_label):
33     if row_label == 'paper':
34         return 1
35     elif row_label == 'rock':
36         return 2
37     elif row_label == 'scissors':
38         return 3
39     else:
40         return None
41 
42 
43 def split(df, group):
44     data = namedtuple('data', ['filename', 'object'])
45     gb = df.groupby(group)
46     return [data(filename, gb.get_group(x)) for filename, x in zip(gb.
           groups.keys(), gb.groups)]
47 
48 
49 def create_tf_example(group, path):
50     with tf.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), '
           rb') as fid:
51         encoded_jpg = fid.read()
52     encoded_jpg_io = io.BytesIO(encoded_jpg)
53     image = Image.open(encoded_jpg_io)
54     width, height = image.size
55 
56 filename = group.filename.encode('utf8')
57     image_format = b'jpg'
58     xmins = []
59     xmaxs = []
60     ymins = []
61     ymaxs = []
62     classes_text = []
63     classes = []
64 
65     for index, row in group.object.iterrows():
66         xmins.append(row['xmin'] / width)
67         xmaxs.append(row['xmax'] / width)
68         ymins.append(row['ymin'] / height)
69         ymaxs.append(row['ymax'] / height)
70         classes_text.append(row['class'].encode('utf8'))
71         classes.append(class_text_to_int(row['class']))
72 
73     tf_example = tf.train.Example(features=tf.train.Features(feature={
74         'image/height': dataset_util.int64_feature(height),
75         'image/width': dataset_util.int64_feature(width),
76         'image/filename': dataset_util.bytes_feature(filename),
77         'image/source_id': dataset_util.bytes_feature(filename),
```

```
78          'image/encoded': dataset_util.bytes_feature(encoded_jpg),
79          'image/format': dataset_util.bytes_feature(image_format),
80          'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
81          'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
82          'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
83          'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
84          'image/object/class/text': dataset_util.bytes_list_feature(
                classes_text),
85          'image/object/class/label': dataset_util.int64_list_feature(
                classes),
86      }))
87      return tf_example
88
89
90  def main(_):
91      writer = tf.python_io.TFRecordWriter(FLAGS.output_path)
92      path = os.path.join(FLAGS.image_dir)
93      examples = pd.read_csv(FLAGS.csv_input)
94      grouped = split(examples, 'filename')
95      for group in grouped:
96          tf_example = create_tf_example(group, path)
97          writer.write(tf_example.SerializeToString())
98
99      writer.close()
100     output_path = os.path.join(os.getcwd(), FLAGS.output_path)
101     print('Successfully created the TFRecords: {}'.format(output_path))
102
103
104 if __name__ == '__main__':
105     tf.app.run()
```

labelmap.pbtxt

```
1  item {
2    id: 1
3    name: 'paper'
4  }
5
6  item {
7    id: 2
8    name: 'rock'
9  }
10
11 item {
12   id: 3
13   name: 'scissors'
14 }
```

faster_rcnn_inception_v2_pets.config

```
# Faster R-CNN with Inception v2, configured for Oxford-IIIT Pets Dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
# should be configured.

model {
  faster_rcnn {
        num_classes: 3
        image_resizer {
          keep_aspect_ratio_resizer {
                min_dimension: 600
                max_dimension: 1024
          }
        }
        feature_extractor {
          type: 'faster_rcnn_inception_v2'
          first_stage_features_stride: 16
        }
        first_stage_anchor_generator {
          grid_anchor_generator {
                scales: [0.25, 0.5, 1.0, 2.0]
                aspect_ratios: [0.5, 1.0, 2.0]
                height_stride: 16
                width_stride: 16
          }
        }
        first_stage_box_predictor_conv_hyperparams {
          op: CONV
          regularizer {
                l2_regularizer {
                  weight: 0.0
                }
          }
          initializer {
                truncated_normal_initializer {
                  stddev: 0.01
                }
          }
        }
        first_stage_nms_score_threshold: 0.0
        first_stage_nms_iou_threshold: 0.7
        first_stage_max_proposals: 300
```

```
44          first_stage_localization_loss_weight: 2.0
45          first_stage_objectness_loss_weight: 1.0
46          initial_crop_size: 14
47          maxpool_kernel_size: 2
48          maxpool_stride: 2
49          second_stage_box_predictor {
50            mask_rcnn_box_predictor {
51              use_dropout: false
52                  dropout_keep_probability: 1.0
53                  fc_hyperparams {
54                    op: FC
55                    regularizer {
56                          l2_regularizer {
57                            weight: 0.0
58                          }
59                    }
60                    initializer {
61                          variance_scaling_initializer {
62                          factor: 1.0
63                          uniform: true
64                          mode: FAN_AVG
65                          }
66                    }
67                  }
68          }
69    }
70    second_stage_post_processing {
71          batch_non_max_suppression {
72            score_threshold: 0.0
73            iou_threshold: 0.6
74            max_detections_per_class: 100
75            max_total_detections: 300
76          }
77          score_converter: SOFTMAX
78    }
79    second_stage_localization_loss_weight: 2.0
80    second_stage_classification_loss_weight: 1.0
81    }
82  }
83
84  train_config: {
85    batch_size: 1
86    optimizer {
87          momentum_optimizer: {
88            learning_rate: {
89                  manual_step_learning_rate {
90                    initial_learning_rate: 0.0002
91                          schedule {
```

```
92                              step: 900000
93                              learning_rate: .00002
94                            }
95                            schedule {
96                              step: 1200000
97                              learning_rate: .000002
98                            }
99                        }
100         }
101         momentum_optimizer_value: 0.9
102   }
103   use_moving_average: false
104 }
105 gradient_clipping_by_norm: 10.0
106 fine_tune_checkpoint: "C:/Users/sevto/Documents/tensor1.12/models/
      research/object_detection/faster_rcnn_inception_v2_coco/model.ckpt"
107 from_detection_checkpoint: true
108 load_all_detection_checkpoint_vars: true
109 # Note: The below line limits the training process to 200K steps, which
      we
110 # empirically found to be sufficient enough to train the pets dataset.
      This
111 # effectively bypasses the learning rate schedule (the learning rate will
112 # never decay). Remove the below line to train indefinitely.
113 num_steps: 200000
114 data_augmentation_options {
115   random_horizontal_flip {
116   }
117 }
118 }
119
120
121 train_input_reader: {
122   tf_record_input_reader {
123   input_path: "C:/Users/sevto/Documents/tensor1.12/models/research/
        object_detection/train.record"
124   }
125   label_map_path: "C:/Users/sevto/Documents/tensor1.12/models/research/
        object_detection/training/labelmap.pbtxt"
126 }
127
128 eval_config: {
129   metrics_set: "coco_detection_metrics"
130   num_examples: 71
131 }
132
133 eval_input_reader: {
134   tf_record_input_reader {
```

```
135         input_path: "C:/Users/sevto/Documents/tensor1.12/models/research/
                object_detection/test.record"
136   }
137   label_map_path: "C:/Users/sevto/Documents/tensor1.12/models/research/
          object_detection/training/labelmap.pbtxt"
138   shuffle: false
139   num_readers: 1
140 }
```

train.py

```
1  # Copyright 2017 The TensorFlow Authors. All Rights Reserved.
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  #     http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied
       .
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14 #
       ==========================================================================
15
16 r"""Training executable for detection models.
17
18 This executable is used to train DetectionModels. There are two ways of
19 configuring the training job:
20
21 1) A single pipeline_pb2.TrainEvalPipelineConfig configuration file
22 can be specified by --pipeline_config_path.
23
24 Example usage:
25         ./train \
26                 --logtostderr \
27                 --train_dir=path/to/train_dir \
28                 --pipeline_config_path=pipeline_config.pbtxt
29
30 2) Three configuration files can be provided: a model_pb2.DetectionModel
31 configuration file to define what type of DetectionModel is being trained
       , an
32 input_reader_pb2.InputReader file to specify what training data will be
       used and
33 a train_pb2.TrainConfig file to configure training parameters.
```

```
34
35 Example usage:
36         ./train \
37                 --logtostderr \
38                 --train_dir=path/to/train_dir \
39                 --model_config_path=model_config.pbtxt \
40                 --train_config_path=train_config.pbtxt \
41                 --input_config_path=train_input_config.pbtxt
42 """
43
44 import functools
45 import json
46 import os
47 import tensorflow as tf
48 from tensorflow.contrib import framework as contrib_framework
49
50 from object_detection.builders import dataset_builder
51 from object_detection.builders import graph_rewriter_builder
52 from object_detection.builders import model_builder
53 from object_detection.legacy import trainer
54 from object_detection.utils import config_util
55
56 tf.logging.set_verbosity(tf.logging.INFO)
57
58 flags = tf.app.flags
59 flags.DEFINE_string('master', '', 'Name of the TensorFlow master to use.'
    )
60 flags.DEFINE_integer('task', 0, 'task id')
61 flags.DEFINE_integer('num_clones', 1, 'Number of clones to deploy per
    worker.')
62 flags.DEFINE_boolean('clone_on_cpu', False,'Force clones to be deployed
    on CPU.  Note that even if ''set to False (allowing ops to run on gpu)
    , some ops may ''still be run on the CPU if they have no GPU kernel.')
63 flags.DEFINE_integer('worker_replicas', 1, 'Number of worker+trainer ''
    replicas.')
64 flags.DEFINE_integer('ps_tasks', 0,'Number of parameter server tasks. If
    None, does not use ''a parameter server.')
65 flags.DEFINE_string('train_dir', '','Directory to save the checkpoints
    and training summaries.')
66
67 flags.DEFINE_string('pipeline_config_path', '','Path to a pipeline_pb2.
    TrainEvalPipelineConfig config ''file. If provided, other configs are
    ignored')
68
69 flags.DEFINE_string('train_config_path', '','Path to a train_pb2.
    TrainConfig config file.')
70 flags.DEFINE_string('input_config_path', '','Path to an input_reader_pb2.
    InputReader config file.')
```

```
71 flags.DEFINE_string('model_config_path', '','Path to a model_pb2.
       DetectionModel config file.')
72
73 FLAGS = flags.FLAGS
74
75
76 @contrib_framework.deprecated(None, 'Use object_detection/model_main.py.'
       )
77 def main(_):
78   assert FLAGS.train_dir, '`train_dir` is missing.'
79   if FLAGS.task == 0: tf.gfile.MakeDirs(FLAGS.train_dir)
80   if FLAGS.pipeline_config_path:
81     configs = config_util.get_configs_from_pipeline_file(
82           FLAGS.pipeline_config_path)
83         if FLAGS.task == 0:
84                 tf.gfile.Copy(FLAGS.pipeline_config_path,
85                   os.path.join(FLAGS.train_dir, 'pipeline.config'),
                          overwrite=True)
86   else:
87     configs = config_util.get_configs_from_multiple_files(
88           model_config_path=FLAGS.model_config_path,
89           train_config_path=FLAGS.train_config_path,
90           train_input_config_path=FLAGS.input_config_path)
91         if FLAGS.task == 0:
92           for name, config in [('model.config', FLAGS.model_config_path)
                 ,('train.config', FLAGS.train_config_path),('input.config',
                FLAGS.input_config_path)]:
93             tf.gfile.Copy(config, os.path.join(FLAGS.train_dir, name),
                    overwrite=True)
94
95   model_config = configs['model']
96   train_config = configs['train_config']
97   input_config = configs['train_input_config']
98
99   model_fn = functools.partial(
100         model_builder.build,
101         model_config=model_config,
102         is_training=True)
103
104   def get_next(config):
105         return dataset_builder.make_initializable_iterator(
106           dataset_builder.build(config)).get_next()
107
108   create_input_dict_fn = functools.partial(get_next, input_config)
109
110   env = json.loads(os.environ.get('TF_CONFIG', '{}'))
111   cluster_data = env.get('cluster', None)
112   cluster = tf.train.ClusterSpec(cluster_data) if cluster_data else None
```

```
113   task_data = env.get('task', None) or {'type': 'master', 'index': 0}
114   task_info = type('TaskSpec', (object,), task_data)
115
116   # Parameters for a single worker.
117   ps_tasks = 0
118   worker_replicas = 1
119   worker_job_name = 'lonely_worker'
120   task = 0
121   is_chief = True
122   master = ''
123
124   if cluster_data and 'worker' in cluster_data:
125         # Number of total worker replicas include "worker"s and the "
              master".
126         worker_replicas = len(cluster_data['worker']) + 1
127   if cluster_data and 'ps' in cluster_data:
128         ps_tasks = len(cluster_data['ps'])
129
130   if worker_replicas > 1 and ps_tasks < 1:
131         raise ValueError('At least 1 ps task is needed for distributed
              training.')
132
133   if worker_replicas >= 1 and ps_tasks > 0:
134         # Set up distributed training.
135         server = tf.train.Server(tf.train.ClusterSpec(cluster), protocol=
              'grpc',job_name=task_info.type,task_index=task_info.index)
136         if task_info.type == 'ps':
137           server.join()
138           return
139
140         worker_job_name = '%s/task:%d' % (task_info.type, task_info.index
              )
141         task = task_info.index
142         is_chief = (task_info.type == 'master')
143         master = server.target
144
145   graph_rewriter_fn = None
146   if 'graph_rewriter_config' in configs:
147         graph_rewriter_fn = graph_rewriter_builder.build(configs['
              graph_rewriter_config'], is_training=True)
148
149   trainer.train(
150         create_input_dict_fn,
151         model_fn,
152         train_config,
153         master,
154         task,
155         FLAGS.num_clones,
```

```
156             worker_replicas,
157             FLAGS.clone_on_cpu,
158             ps_tasks,
159             worker_job_name,
160             is_chief,
161             FLAGS.train_dir,
162             graph_hook_fn=graph_rewriter_fn)
163
164
165 if __name__ == '__main__':
166 tf.app.run()
```

Object_detection_image.py

```
1 ######## Image Object Detection Using Tensorflow-trained Classifier
      #########
2 #
3 # Author: Evan Juras
4 # Date: 1/15/18
5 # Description:
6 # This program uses a TensorFlow-trained neural network to perform object
      detection.
7 # It loads the classifier and uses it to perform object detection on an
      image.
8 # It draws boxes, scores, and labels around the objects of interest in
      the image.
9
10 ## Some of the code is copied from Google's example at
11 ## https://github.com/tensorflow/models/blob/master/research/
      object_detection/object_detection_tutorial.ipynb
12
13 ## and some is copied from Dat Tran's example at
14 ## https://github.com/datitran/object_detector_app/blob/master/
      object_detection_app.py
15
16 ## but I changed it to make it more understandable to me.
17
18 # Import packages
19 import os
20 import cv2
21 import numpy as np
22 import tensorflow as tf
23 import sys
24
25 # This is needed since the notebook is stored in the object_detection
      folder.
26 sys.path.append("..")
27
28 # Import utilites
```

```
29 from utils import label_map_util
30 from utils import visualization_utils as vis_util
31
32 # Name of the directory containing the object detection module we're
       using
33 MODEL_NAME = 'inference_graph'
34 IMAGE_NAME = 'images/test/depositphotos.jpg'
35
36 # Grab path to current working directory
37 CWD_PATH = os.getcwd()
38
39 # Path to frozen detection graph .pb file, which contains the model that
       is used
40 # for object detection.
41 PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.
       pb')
42
43 # Path to label map file
44 PATH_TO_LABELS = os.path.join(CWD_PATH,'training','labelmap.pbtxt')
45
46 # Path to image
47 PATH_TO_IMAGE = os.path.join(CWD_PATH,IMAGE_NAME)
48
49 # Number of classes the object detector can identify
50 NUM_CLASSES = 3
51
52 # Load the label map.
53 # Label maps map indices to category names, so that when our convolution
54 # network predicts `5`, we know that this corresponds to `king`.
55 # Here we use internal utility functions, but anything that returns a
56 # dictionary mapping integers to appropriate string labels would be fine
57 label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
58 categories = label_map_util.convert_label_map_to_categories(label_map,
       max_num_classes=NUM_CLASSES, use_display_name=True)
59 category_index = label_map_util.create_category_index(categories)
60
61 # Load the Tensorflow model into memory.
62 detection_graph = tf.Graph()
63 with detection_graph.as_default():
64   od_graph_def = tf.GraphDef()
65   with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
66         serialized_graph = fid.read()
67         od_graph_def.ParseFromString(serialized_graph)
68         tf.import_graph_def(od_graph_def, name='')
69
70   sess = tf.Session(graph=detection_graph)
71
72 # Define input and output tensors (i.e. data) for the object detection
```

```
    classifier
73
74 # Input tensor is the image
75 image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
76
77 # Output tensors are the detection boxes, scores, and classes
78 # Each box represents a part of the image where a particular object was
       detected
79 detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
80
81 # Each score represents level of confidence for each of the objects.
82 # The score is shown on the result image, together with the class label.
83 detection_scores = detection_graph.get_tensor_by_name('detection_scores:0
       ')
84 detection_classes = detection_graph.get_tensor_by_name('detection_classes
       :0')
85
86 # Number of objects detected
87 num_detections = detection_graph.get_tensor_by_name('num_detections:0')
88
89 # Load image using OpenCV and
90 # expand image dimensions to have shape: [1, None, None, 3]
91 # i.e. a single-column array, where each item in the column has the pixel
       RGB value
92 image = cv2.imread(PATH_TO_IMAGE)
93 image_expanded = np.expand_dims(image, axis=0)
94
95 # Perform the actual detection by running the model with the image as
       input
96 (boxes, scores, classes, num) = sess.run([detection_boxes,
       detection_scores, detection_classes, num_detections],feed_dict={
       image_tensor: image_expanded})
97
98 # Draw the results of the detection (aka 'visulaize the results')
99
100 vis_util.visualize_boxes_and_labels_on_image_array(image,np.squeeze(boxes
       ),np.squeeze(classes).astype(np.int32),np.squeeze(scores),
       category_index,use_normalized_coordinates=True,line_thickness=8,
       min_score_thresh=0.60)
101
102 # All the results have been drawn on image. Now display the image.
103 cv2.imshow('Object detector', image)
104
105 # Press any key to close the image
106 cv2.waitKey(0)
107
108 # Clean up
109 cv2.destroyAllWindows()
```

Object_detection_video.py

```
1  ######## Video Object Detection Using Tensorflow-trained Classifier
       #########
2  #
3  # Author: Evan Juras
4  # Date: 1/16/18
5  # Description:
6  # This program uses a TensorFlow-trained classifier to perform object
       detection.
7  # It loads the classifier and uses it to perform object detection on a
       video.
8  # It draws boxes, scores, and labels around the objects of interest in
       each
9  # frame of the video.
10
11 ## Some of the code is copied from Google's example at
12 ## https://github.com/tensorflow/models/blob/master/research/
       object_detection/object_detection_tutorial.ipynb
13
14 ## and some is copied from Dat Tran's example at
15 ## https://github.com/datitran/object_detector_app/blob/master/
       object_detection_app.py
16
17 ## but I changed it to make it more understandable to me.
18
19 # Import packages
20 import os
21 import cv2
22 import numpy as np
23 import tensorflow as tf
24 import sys
25
26 # This is needed since the notebook is stored in the object_detection
       folder.
27 sys.path.append("..")
28
29 # Import utilites
30 from utils import label_map_util
31 from utils import visualization_utils as vis_util
32
33 # Name of the directory containing the object detection module we're
       using
34 MODEL_NAME = 'inference_graph'
35 VIDEO_NAME = 'test.mov'
36
37 # Grab path to current working directory
38 CWD_PATH = os.getcwd()
39
```

```
# Path to frozen detection graph .pb file, which contains the model that is used
# for object detection.
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')

# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,'training','labelmap.pbtxt')

# Path to video
PATH_TO_VIDEO = os.path.join(CWD_PATH,VIDEO_NAME)

# Number of classes the object detector can identify
NUM_CLASSES = 6

# Load the label map.
# Label maps map indices to category names, so that when our convolution
# network predicts `5`, we know that this corresponds to `king`.
# Here we use internal utility functions, but anything that returns a
# dictionary mapping integers to appropriate string labels would be fine
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

# Load the Tensorflow model into memory.
detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

  sess = tf.Session(graph=detection_graph)

# Define input and output tensors (i.e. data) for the object detection classifier

# Input tensor is the image
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')

# Output tensors are the detection boxes, scores, and classes
# Each box represents a part of the image where a particular object was detected
detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')

# Each score represents level of confidence for each of the objects.
```

```
83 # The score is shown on the result image, together with the class label.
84 detection_scores = detection_graph.get_tensor_by_name('detection_scores:0
       ')
85 detection_classes = detection_graph.get_tensor_by_name('detection_classes
       :0')
86
87 # Number of objects detected
88 num_detections = detection_graph.get_tensor_by_name('num_detections:0')
89
90 # Open video file
91 video = cv2.VideoCapture(PATH_TO_VIDEO)
92
93 while(video.isOpened()):
94
95   # Acquire frame and expand frame dimensions to have shape: [1, None,
         None, 3]
96   # i.e. a single-column array, where each item in the column has the
         pixel RGB value
97   ret, frame = video.read()
98   frame_expanded = np.expand_dims(frame, axis=0)
99
100  # Perform the actual detection by running the model with the image as
         input
101  (boxes, scores, classes, num) = sess.run([detection_boxes,
         detection_scores, detection_classes, num_detections],feed_dict={
         image_tensor: frame_expanded})
102
103  # Draw the results of the detection (aka 'visulaize the results')
104  vis_util.visualize_boxes_and_labels_on_image_array(frame,np.squeeze(
         boxes),np.squeeze(classes).astype(np.int32),np.squeeze(scores),
         category_index,use_normalized_coordinates=True,line_thickness=8,
         min_score_thresh=0.60)
105
106  # All the results have been drawn on the frame, so it's time to display
       it.
107  cv2.imshow('Object detector', frame)
108
109  # Press 'q' to quit
110  if cv2.waitKey(1) == ord('q'):
111        break
112
113 # Clean up
114 video.release()
115 cv2.destroyAllWindows()
```

Quellcode für den Lösungsansatz mit Keras und SqueezeNet

gather_images.py

```
desc = '''Script to gather data images with a particular label.

Usage: python gather_images.py <label_name> <num_samples>

The script will collect <num_samples> number of images and store them
in its own directory.

Only the portion of the image within the box displayed
will be captured and stored.

Press 'a' to start/pause the image collecting process.
Press 'q' to quit.

'''

import cv2
import os
import sys
# Abfangen ob der Befehl zum ausfuehren der Datei richtig eingeben wurde
try:
  label_name = sys.argv[1]
  num_samples = int(sys.argv[2])
except:
  print("Argumente fehlen.")
  print(desc)
  exit(-1)
# Speicherort fuer die Bilder
IMG_SAVE_PATH = 'image_data'
IMG_CLASS_PATH = os.path.join(IMG_SAVE_PATH, label_name)
%#Versuch angegebenen Ordner zu erstellen
try:
  os.mkdir(IMG_SAVE_PATH)
except FileExistsError:
  pass
try:
  os.mkdir(IMG_CLASS_PATH)
except FileExistsError:
  print("{} Speicherort existiert bereits.".format(IMG_CLASS_PATH))
  print("Alle Bilder werden zu den bereits existierenden Bildern dazu
      gespeichert")
#Zum erkennen der Kamera
cap = cv2.VideoCapture(0)

start = False
count = 0
```

```
#Aufloesung vom Bild(1280x720)
cap.set(3,1280)
cap.set(4,720)
#Start der direkt Uebertragung
while True:
  ret, frame = cap.read()
  if not ret:
        continue
  #Zum Zaehlen der bereits aufgenommenen Bildern
  if count == num_samples:
        break
  #Kasten Zeichnen
  cv2.rectangle(frame, (100, 100), (500, 500), (255, 255, 255), 2)
  #Speichern der Bilder innerhalb des Kreises
  if start:
        roi = frame[100:500, 100:500]
        save_path = os.path.join(IMG_CLASS_PATH, '{}.jpg'.format(count +
            1))
        cv2.imwrite(save_path, roi)
        count += 1
  #Anzeige der Aufgenommenen Bilder
  font = cv2.FONT_HERSHEY_SIMPLEX
  cv2.putText(frame, "Aufgenommen {}".format(count),(5, 50), font, 0.7,
      (0, 255, 255), 2, cv2.LINE_AA)
  cv2.imshow("Aufgenommene Bilder", frame)
  #Beim pressen der a-Taste wird das Programm gestartet oder Pausiert
  k = cv2.waitKey(10)
  if k == ord('a'):
        start = not start
  #Zum schliessen die Taste q druecken
  if k == ord('q'):
        break

print("\n{} Bild(er) gespeichert im Verzeichnis {}".format(count,
    IMG_CLASS_PATH))
cap.release()
cv2.destroyAllWindows()
```

train.py

```
import cv2
import numpy as np
from keras_squeezenet import SqueezeNet
from keras.optimizers import Adam
from keras.utils import np_utils
from keras.layers import Activation, Dropout, Convolution2D,
    GlobalAveragePooling2D
from keras import layers
from keras.models import Sequential
```

```
import tensorflow as tf
import os
import matplotlib.pyplot as plt
import pandas as pd

#Bilder Verzeichnis
IMG_SAVE_PATH = 'image_data'
#Objekt-Schluessel-Paar
CLASS_MAP = {
  "rock": 0,
  "paper": 1,
  "scissors": 2,
  "none": 3,
  "echse":4,
  "spock":5
}
#Laenge der CLASS_MAP
NUM_CLASSES = len(CLASS_MAP)

# Zum Editieren der Bilder um aus einem kleinen Datensatz das Beste zu
    machen
# gen = ImageDataGenerator(rotation_range=15,
                           # width_shift_range=0.1,
                           # height_shift_range=0.1,
                           # shear_range=0.01,
                           # zoom_range=[0.9, 1.25],
                           # horizontal_flip=True,
                           # vertical_flip=False,
                           # fill_mode='reflect',
                           # data_format='channels_last',
                           # brightness_range=[0.5, 1.5])

#Liste an Schluesseln aus der CLASS_MAP
def mapper(val):
return CLASS_MAP[val]

#Modell Einstellungen
def get_model():
  model = Sequential([
        SqueezeNet(input_shape=(227, 227, 3), include_top=False),
        layers.Dense(4, activation ='relu'),
        Dropout(0.4),
        Convolution2D(NUM_CLASSES, (1, 1), padding='valid'),
        Activation('relu'),
        GlobalAveragePooling2D(),
        Activation('softmax')
  ])
  return model
```

```
56
57
58 # Bilder aus dem Verzeichnis laden
59 dataset = []
60 for directory in os.listdir(IMG_SAVE_PATH):
61   path = os.path.join(IMG_SAVE_PATH, directory)
62   if not os.path.isdir(path):
63         continue
64   for item in os.listdir(path):
65         # Das versteckten dateien ignoriert werden
66         if item.startswith("."):
67           continue
68         img = cv2.imread(os.path.join(path, item))
69         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
70         img = cv2.resize(img, (227, 227))
71         dataset.append([img, directory])
72
73
74 '''
75 dataset = [
76 [[...], 'rock'],
77 [[...], 'paper'],
78 ...
79 ]
80 '''
81 #Vermischen des Datensatzes
82 from random import shuffle
83 dataset= np.random.permutation(dataset)
84 data, labels = zip(*dataset)
85 labels = list(map(mapper, labels))
86
87
88 # Label Encoding fuer die kategorischen Werte
89 labels = np_utils.to_categorical(labels)
90
91 # Auswahl des Modells und die Angabe der Lernrate
92 model = get_model()
93 model.compile(
94   optimizer=Adam(lr=0.0001),
95   loss='categorical_crossentropy',
96   metrics=['categorical_accuracy']
97 )
98 #Callback zum Fruezeitegen stoppen, wenn sich das Modell nicht mehr
     verbessert
99 callback1 = tf.keras.callbacks.EarlyStopping(monitor='
     categorical_accuracy', patience=4)
100 # Training Starten
```

```
history = model.fit(np.array(data), np.array(labels), validation_split
    =0.15, epochs=15, batch_size=20, callbacks=[callback1], shuffle=True,
    verbose=1)

#Plotten der Trainingsergebnisse(val_categorical_accuracy)
plt.plot(history.history['categorical_accuracy'])
plt.plot(history.history['val_categorical_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
#
# Plotten der Trainingsergebnisse(val_loss)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
# Modell abspeichern
model.save("rock-paper-scissors-model.h5")
```

play.py

```
from keras.models import load_model
import cv2
import numpy as np
from random import choice
#Label Encoding wieder rueckgaengig machen
REV_CLASS_MAP = {
  0: "rock",
  1: "paper",
  2: "scissors",
  3: "none",
  4: "echse",
  5: "spock"
}


#Speichern der Schluessel(rock,paper....)
def mapper(val):
  return REV_CLASS_MAP[val]

#Gewinnregeln
def calculate_winner(move1, move2):
  if move1 == move2:
        return "Unentschieden"
```

```
24
25  if move1 == "rock":
26        if move2 == "scissors":
27          return "Spieler"
28        if move2 == "paper":
29          return "Computer"
30        if move2 == "echse":
31          return "Spieler"
32        if move2 == "spock":
33          return "Computer"
34
35  if move1 == "paper":
36        if move2 == "rock":
37          return "Spieler"
38        if move2 == "scissors":
39          return "Computer"
40        if move2 == "echse":
41          return "Computer"
42        if move2 == "spock":
43          return "Spieler"
44
45  if move1 == "scissors":
46    if move2 == "paper":
47          return "Spieler"
48        if move2 == "rock":
49          return "Computer"
50        if move2 == "echse":
51          return "Spieler"
52        if move2 == "spock":
53          return "Computer"
54
55  if move1 == "echse":
56        if move2 == "paper":
57          return "Spieler"
58        if move2 == "rock":
59          return "Computer"
60        if move2 == "scissors":
61          return "Computer"
62        if move2 == "spock":
63          return "Spieler"
64
65  if move1 == "spock":
66        if move2 == "paper":
67          return "Computer"
68        if move2 == "rock":
69          return "Spieler"
70        if move2 == "scissors":
71          return "Spieler"
```

```
72          if move2 == "echse":
73            return "Computer"
74
75  # Das Modell laden
76  model = load_model("rock-paper-scissors-model.h5")
77  # Zum erkennen der Kamera
78  cap = cv2.VideoCapture(0)
79
80  prev_move = None
81  # Aufloesung einstellen
82  cap.set(3,1280)
83  cap.set(4,720)
84  # Direktaufnahme wird gestartet
85  while True:
86    ret, frame = cap.read()
87    if not ret:
88          continue
89
90    # Rechteck fuer den Spieler
91    cv2.rectangle(frame, (100, 100), (500, 500), (255, 255, 255), 2)
92    # rectangle for computer to play
93    cv2.rectangle(frame, (800, 100), (1200, 500), (255, 255, 255), 2)
94
95    # Rechteck fuer den Gegner
96    roi = frame[100:500, 100:500]
97    img = cv2.cvtColor(roi, cv2.COLOR_BGR2RGB)
98    img = cv2.resize(img, (227, 227))
99
100   # Versuchen die Hand des Spielers zu erkennen
101   pred = model.predict(np.array([img]))
102   move_code = np.argmax(pred[0])
103   user_move_name = mapper(move_code)
104
105   # Berechnung wer gewonnen hat
106   if prev_move != user_move_name:
107         if user_move_name != "none":
108           computer_move_name = choice(['rock', 'paper', 'scissors','echse
                  ' ,'spock'])
109           winner = calculate_winner(user_move_name, computer_move_name)
110         else:
111           computer_move_name = "none"
112           winner = "Warte..."
113   prev_move = user_move_name
114
115   # Ausgabe der Berechnung
116   font = cv2.FONT_HERSHEY_SIMPLEX
117   cv2.putText(frame, "Spieler: " + user_move_name,(50, 50), font, 1.2,
          (255, 255, 255), 2, cv2.LINE_AA)
```

```
118     cv2.putText(frame, "Computer: " + computer_move_name,(750, 50), font,
            1.2, (255, 255, 255), 2, cv2.LINE_AA)
119     cv2.putText(frame, "Gewinner: " + winner,(400, 600), font, 2, (0, 0,
            255), 4, cv2.LINE_AA)
120
121 # Zum berechnen der Bildrate
122 #
123 #       import time
124 #       (major_ver, minor_ver, subminor_ver) = (cv2.__version__).split('.')
125 #
126 #       if int(major_ver)  < 3 :
127 #           fps = cap.get(cv2.cv.CV_CAP_PROP_FPS)
128 #           print ("Frames per second using video.get(cv2.cv.
        CV_CAP_PROP_FPS): {0}".format(fps))
129 #       else :
130 #           fps = cap.get(cv2.CAP_PROP_FPS)
131 #           print ("Frames per second using video.get(cv2.CAP_PROP_FPS) :
        {0}".format(fps))
132 #
133 #       num_frames = 120;
134 #       print ("Capturing {0} frames".format(num_frames))
135 #       start = time.time()
136 #
137 #       for i in range(0, num_frames) :
138 #           ret, frame = cap.read()
139 #
140 #       end = time.time()
141 #
142 #       seconds = end - start
143 #       print ("Time taken : {0} seconds".format(seconds))
144 #
145 #       fps  = num_frames / seconds;
146 #       print ("Estimated frames per second : {0}".format(fps))
147
148     # Zum Anzeigen der Hanzeichen vom Computer
149     if computer_move_name != "none":
150           icon = cv2.imread("images/{}.jpg".format(computer_move_name))
151           icon = cv2.resize(icon, (400, 400))
152           frame[100:500, 800:1200] = icon
153
154     cv2.imshow("Rock Paper Scissors", frame)
155     # Zum schliessen q druecken
156     k = cv2.waitKey(10)
157     if k == ord('q'):
158           break
159
160 cap.release()
161 cv2.destroyAllWindows()
```