**DeepHorizon**

-Janis Leuenberger

Inhaltsverzeichnis

**Es wurden keine Einträge für das Inhaltsverzeichnis gefunden.**

# Einstieg

Das Forschungsfeld sowie die bereits ausgelieferten Lösungen in Verbindung mit maschinellem Lernen interessierten mich sehr. Ich versuchte mich schon ein wenig in ML vor allem zugunsten des Lernens. Als erstes probierte ich Keras. Dies ist eine open-source Python Bibliothek für ML. Sie verwendet als backend vor allem Tensorflow, ein von Google entwickeltes Framework. Keras ist mit der Priorität zur einsteiger- und benutzerfreundlichkeit entwickelt worden und ist somit ideal für Neulinge. Mit der Hilfe von Kleinprojekten, wie dem klassifizieren von Katzen und Hunden, lernte ich bereits wichtige tools wie Tensorboard und opencv.

# Planung und Entscheidung

In Voraussicht auf ein hoffentlich erfolgreiches Projekt wog ich verschiedene Möglichkeiten und Lösungen ab. Die wichtigsten werden im Folgenden beschrieben.

## Sprache & Frameworks

Tensorflow und die darauf basierenden Systeme sind die führenden Lösungen für ML. Dies zeigt der weit verbreitete Einsatz in Google internen System sowie von anderen namenhaften Firmen wie NVidia, eBay, Intel und Twitter. Alternativen wären zum Beispiel das von Facebook entwickelte PyTorch. Jedoch werde ich mich voraussichtlich auf TF fokussieren.

Um mit TF, welches aus Geschwindkeitsgründen mehrheitlich in C++, interagieren zu können braucht es eine Sprache. Es gibt viele mehr oder weniger unterstütze Möglichkeiten doch für mich kamen nur Python oder Javascript in Frage. Schlussentlich entschied ich mich für Python aus folgenden Gründen: Meiner Meinung nach ist die Syntax besser, TF wurde für und mit Python entwickelt, andere Bibliotheken setzen Python voraus (Bsp.: Keras, TFLearn).

## Lösungsansätze

### Algorithmus

Die einfachere Möglichkeit ein komplexes Problem zu lösen stellt ein Algorithmus dar. Ein grosser Vorteil wäre die Trainingszeit, die nicht benötigt wird. Zudem wäre er zuverlässig sofern der Code keine Fehler enthält. Nachteile sind die Spezialisierung einer solchen Applikation. Falls sich das Problem nur ein wenig ändern, hat der Algorithmus keine Chance mehr. Die Entwicklungsdauer für mein Problem wäre so gross und mathematisch kompliziert wodurch ein Algorithmus zumindest alleine keine Option ist.

### Convolutional Neural Network

Die logische Variante wäre ein Convolutional Neural Network, da es um image proccesing beziehungsweise image categorization geht. Hier würden Trainingsdaten gesammelt werden, mit der das Netzwerk lernt die Daten zu verstehen um zum gleichen Ergebnis zu kommen. Das Netzwerk wäre hier limitiert auf die Fähigkeiten des Fahres der Trainingsdaten.

### Reeinforcement Learning

Im reeinforcement learning fällt das Netzwerk Entscheidungen und wird dann entweder durch ein weiteres Netzwerk, sich selber oder durch einen Algorithmus bewertet. Dies kann zu einem sehr guten Ergebnis führen, jedoch ist das Entwickeln sehr viel schwieriger, vor allem für den «Bewerter», weil ich voraussichtlich nur mit Bildinformation arbeiten werde und nicht mit überlegenen Daten wie z.B. Postion zu Agenten, Abstand zu anderen Fahrzeugen oder zum Rand der Strasse.

# Algorithmus

## Bildschirmaufnahme

Wichtig: Hier wird Schritt für Schritt die Entwicklung beschrieben. Viele Zeilen wurden geschrieben und wieder verworfen. Alles kann nicht dokumentiert werden zudem werden nicht alle Version veröffentlich und veralteter Code wird nicht kommentiert.

Der erste Schritt war das Bild auszulesen. Dafür benutzte ich PIL (Python Imaging Library), welche sehr benutzerfreundlich ist. NumPy ist für wissenschaftliches programmieren unumgänglich. Es beinhaltet sehr wichtige Objekte wie N-Dimensionale Arrays. NumPy wird für dieses Projekte immer wichtiger. Zunächst wird es aber nur für das konvertieren von Objekten verwendet. Opencv ist wie NumPy essenziell für dieses Projekt, da es einfache und effiziente Funktionen zur Bildverarbeitung hat.

import numpy as np

from PIL import ImageGrab

import cv2

import time

last\_time = time.time()

while(True):

    # Record a screenshot of the top left corner of the screen. Size: 900 x 650

    screen = np.array(ImageGrab.grab(bbox=(0,30, 900,680)))

    # Show the screenshot in a new window.

    cv2.imshow('Press q to quit', cv2.cvtColor(screen, cv2.COLOR\_BGR2RGB))

    if cv2.waitKey(25) & 0xFF == ord('q'):

        cv2.destroyAllWindows()

        break

    # Calculate time used for the processing.

    print(f"Loop took: {time.time() -last\_time} seconds")

    last\_time = time.time()

Das sieht dann irgendwie so aus. Links ist das Fenster mit dem Spiel "Forza Horizon 3", in der Mitte das Fenster, in welchem der zuvor aufgenommene Screenshot angezeigt wird und rechts ist die Konsole in der angezeigt wird wie lange der while-loop brauchte um die Bildschirmaufnahme und Anzeige zu erledigen. Durchschnittlich sind das etwas 0.09 Sekunden was in eine FPS (Bilder pro Sekunde) von ca. 11 resultiert. Das ist eine sehr niedrige Zahl. Im Vergleich: Das spielt läuft mit ca. 140 FPS.

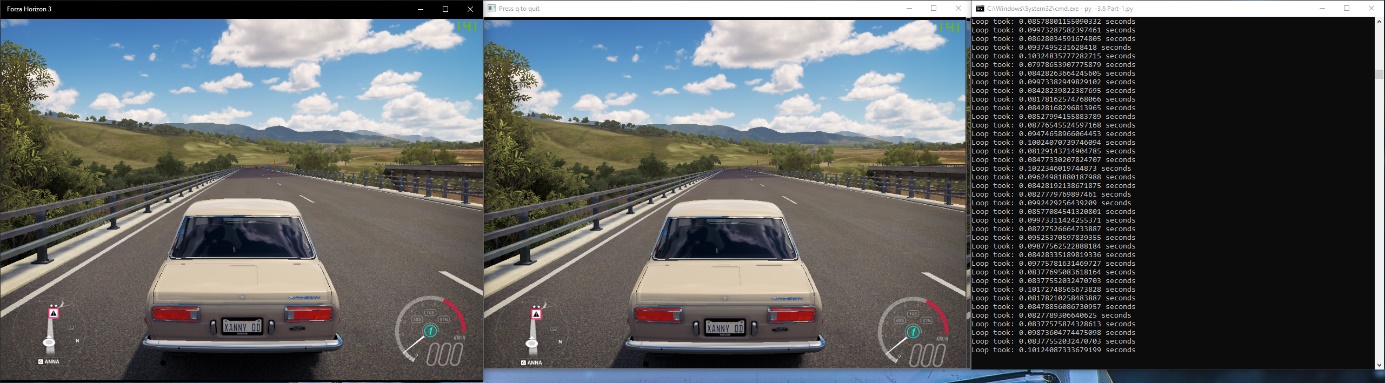


Abbildung : Bildschirmaufnahme, Quelle: Screenshot

## Kantendetektion

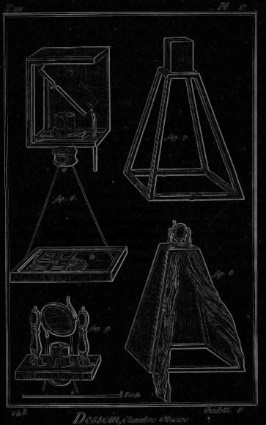
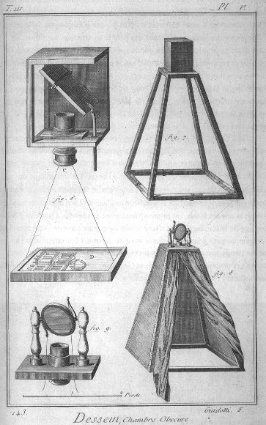


Abbildung : Beispiel Kantendetektion, Quelle https://de.wikipedia.org/wiki/Kantendetektion

Die Bildschirmaufnahme liefert die Grundlage für die Bildverarbeitung, die nun implementiert werden kann. Die Kantendetektion wäre ein guter Anfang um die riesige Menge von Daten, die jede Zentel Sekunde aufgenommen wird auf das Wichtigste zu reduzieren. Dabei werden mit mathematisch komplizierten Algorithmen Flächen voneinander zu trennen sofern sie sich genügend unterscheiden. Ich benutzte den von John F. Canny (nicht zu verwechseln mit John F. Kennedy) entwickelten "Canny algorithm". Die folgende Funktion erledigt dies.

# Function, which uses cv2 to find edges

def process\_img(original\_image):

    processed\_img = cv2.cvtColor(original\_image, cv2.COLOR\_BGR2GRAY)

    processed\_img = cv2.Canny(processed\_img, threshold1=200, threshold2=300)

    return processed\_img

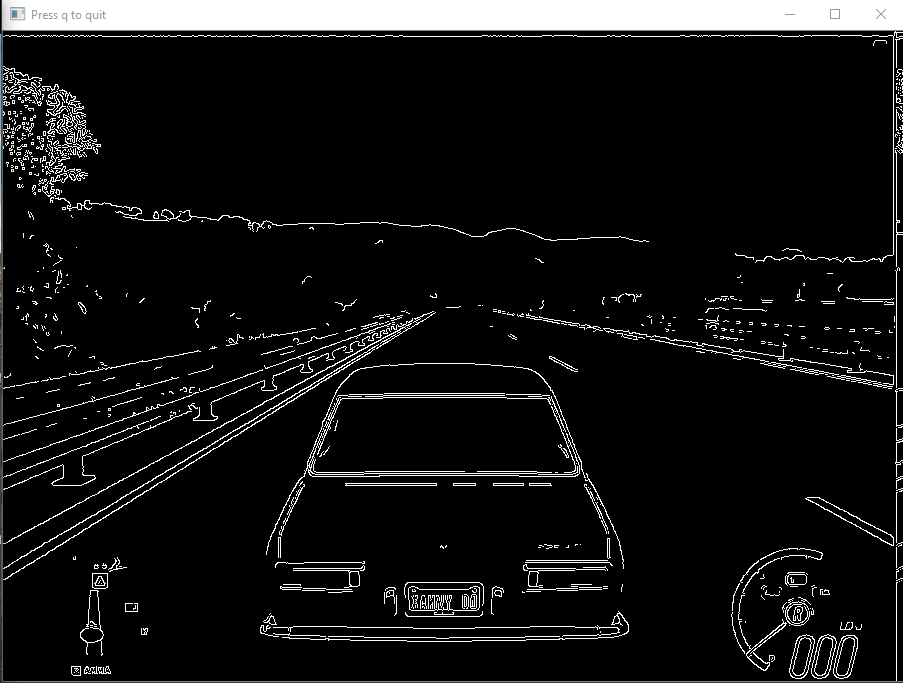


Abbildung : Candy Algorithmus angewendet, Quelle: Screenshot

## Bereich von Interesse



Abbildung : Beispiel Region of interest, Quelle: Screenshot

Region of Interest oder Bereich von Interesse beschreibt den Teil, meist von einem Bild, welcher die meisten Informationen enthält. In einem Fall können keine nützlichen Informationen aus dem Himmel gewonnen werden, deshalb wird dieser weggelassen um unnötige Leistungsverschwendung zu verhindern. Man könnte auch das Auto, den Radar und die Geschwindigkeitsanzeige "wegschneiden", wie im zweiten Bild ersichtlich. Momentan wird darauf verzichtet, weil später vermutlich die "first person" vom Auto verwendet wird.



Abbildung : First person Sicht, Quelle: Screenshot

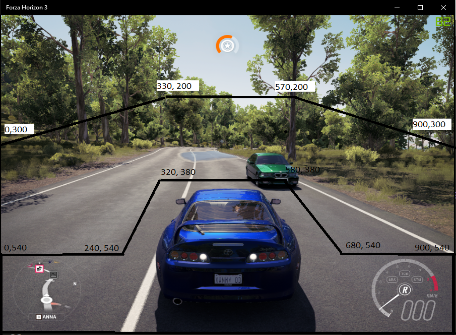


Abbildung : Beispiel erweiterte region of interest, Quelle: Screenshot

Zuerst wird ein numpy Array erstellt mit der gleichen Grösse wie das übergebene Bild, welcher gefüllt ist mit Nullen. Danach füllt es den Bereich des Arrays welcher als die Ecken eines Polygons mit der Variable "vertices" (=Ecken) übergeben wurde mit der Zahl 255. Somit haben wir eine "Maske", die bei dem Bereich von Interesse den Wert 255 hat um "unwichtigen" Bereich, dem Himmel den Wert 0 "cv2.bitwise\_and" erledigt den Rest indem es nur die Elemente des Arrays (vom Bild) behält welche bei der Maske den Wert 255 haben. Die Anderen werden auf 0 beziehungsweise Schwarz gesetzt werden.

# Region of interest

def roi(img, vertices):

    mask = np.zeros\_like(img)

    cv2.fillPoly(mask, vertices, 255)

    masked = cv2.bitwise\_and(img, mask)

    return masked

vertices = np.array(

    [[5,680], [5,300], [330,200], [570,200], [895,300], [895,680]],

    np.int32)

processed\_img = roi(processed\_img, [vertices])

## Liniendetektion

Der nächste Schritt ist die Kanten, welche mit dem Canny Algorithmus erkannt wurden auf relevante Linien. Ein Beispiel wäre für eine Wichtige ist die weisse Linie, welche am Rand der Strasse verläuft. Wenn man weiss wie sich die Linie in Relation zum Fahrzeug befindet kann herausgefunden werden ob man sich zu nahe am rechten beziehungsweise am linken Strassenrand befindet. Im Gegensatz wäre der Horizon eine Linie, die uns nichts Sinnvolles mitteilen kann. Zum Glück hat opencv bereits eine Funktion welche als Eingabe ein Bild annimmt, auf dem bereits eine Kantendetektion ausgeführt wurde. Der Rückgabewert ist eine Liste von Linien, die der Algorithmus gefunden hat. Dabei kann der threshold (=Schwelle), die minimale Länge einer Linie, die maximale Unterbrechung in einer Linie und weiteres angegeben werden. Dann können die Linien mit der hauseigenen Funktion "draw\_lines()" auf das Bild gezeichnet werden.

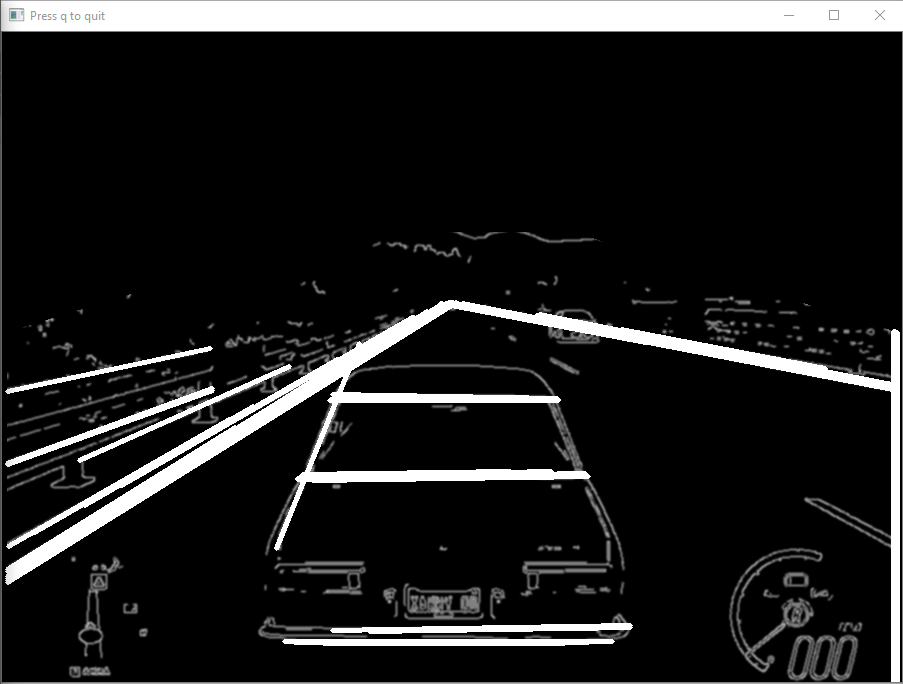


Abbildung : Liniendetektion schlechtes Beispiel, Quelle: Screenshot

lines = cv2.HoughLinesP(processed\_img, 1, np.pi/180, 30, np.array([]), 200, 5)

draw\_lines(processed\_img, lines)



Abbildung : Liniendetektion gutes Beispiel, Quelle: Screenshot

## Das Erwachen der Macht

Um zu veranschaulichen was bist jetzt erreicht worden ist habe ich einen primitiven Algorithmus geschrieben welcher anhand der gefunden Linien entscheidet ob nach rechts, links oder nicht gelenkt werden soll. Dabei werden links und rechts Pixel nach ihrer Helligkeit überprüft. Die rote Linie zeigt den ungefähren Pfad. Beispiel: Auf der rechten Linie wird ein Pixel mit dem Farbwert 255 gefunden und auf der linken Seite wird kein Pixel mit diesem Farbwert gefunden. Dies bedeutet der Algorithmus ist mit hoher Wahrscheinlichkeit zu nahe am rechten Strassenrand darum wird der Tastendruck der Taste "A" simuliert und das Auto bewegt sich nach links bis zum nächsten Screenshot. Da dies nur ein Test ist wird die Geschwindigkeit noch von mir kontrolliert. Dabei ist noch zu erwähnen die FPS hat sicher durch die ganze Bildverarbeitung verschlechtert. Momentan sind es zwischen 7 und 10 FPS. Für den momentanen Zustand des Projektes reicht dies jedoch sollte es für die Zukunft nach Verbessrungen gesucht werden. Kleine Vorschau: YTVID

## Das Einschlafen der Macht

# Neural Networks erklärt

## Layers

Hier werde ich die für DeepHorizon benutzen neuralen ebenen beschreiben

### Convolutional Layer

### Dense Layer

### Dropout Layer

Ein grosses Problem bei convolutional neural networks stellt das sogenannte «overfitting» dar. In einfachen Worten beschreibt es das «auswendig lernen» der Trainingsdaten. Dies kann sehr schnell passiert und das Resultat ist fatal.

Bei der Entwicklung von neural networks strebt man immer die generalization(möglichst gute Ergebnisse bei unbekannten Szenarien) an. Dies wird sehr schnell zerstört wenn man ein model länger trainiert(obwohl eigentlich längeres training = bessers Ergebniss). Zum Glück gibt es Gegenmassnahmen. Sogenanntes regularizing.

Eine bekannte Möglichkeit ist das Trainieren und von mehrere verschiedenen models, die dann Zusammen ein Ergebnis präsentieren. Jedoch ist dies vergleichsweise sehr Rechen- und Zeitintensiv.

Eine neuere Möglichkeit ist das sogenannte «dropout». Hier wird der Effekt von mehreren Netzwerken in einem simuliert. Das geschieht durch das zufällige weglassen von Knotenpunkten. Salopp gesagt lässt man das Netzwerk bestimmte Teile von sich nicht benutzen. Dadurch kann Netzwerk nicht auf bestimmte Verbindungen vertrauen, wodurch es verschiedene Möglichkeiten lernt ein Problem zu lösen. «dropout» ist sehr effektiv und wenig Rechenintensiv.

### Softmax

Softmax ist eine mathematische Funktion, die das Ergebnis von einem neural network in Wahrscheinlichkeiten ändert.

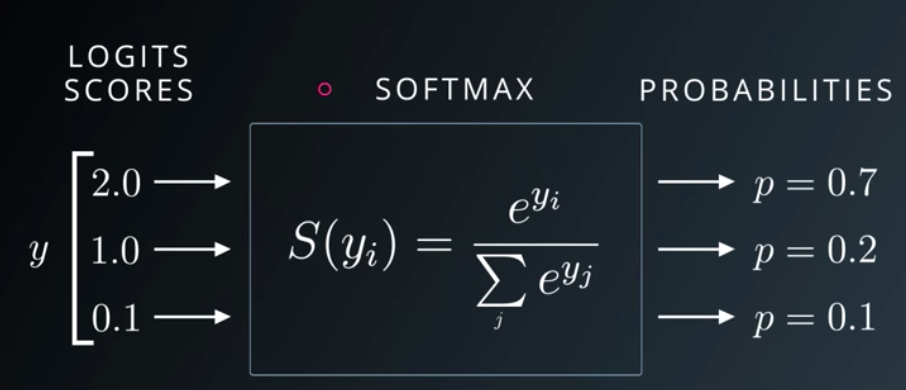


Abbildung : Softmax Visualisierung,

Quelle: https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d

Das Bild beschreibt einen softmax Prozess. Die Logits scores sind die Ergebnisse, die das Netzwerk bis jetzt produziert hat.

## Activations

### Rectified linear

# Sources:

A Gentle Introduction to Dropout for Regularizing Deep Neural Network, 26.10.2019 <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>

Understand the Softmax Function in Minutes, 26.10.2019 <https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d>

<https://www.tensorflow.org/>

<https://pillow.readthedocs.io/en/stable/>