# Können Computer CAPTCHAs lösen?


**Maturitätsarbeit von Jan Kümmerle, Klasse 6b**
**Kantonsschule Zimmerberg, Au ZH**

**Schuljahr: 2024/2025**



Betreuende Lehrperson: Gérald Huber
Korreferent: Philipp Moor

# Authentizitätserklärung
# Maturitätsarbeit

Name: ..... Vorname: .....

Klasse: ..... Betreuende Lehrperson: .....

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel

.....

selbstständig, ohne unerlaubte fremde Hilfe und in eigenen Worten angefertigt habe. Ich erkläre, dass ich keine anderen als die angegeben Quellen und Hilfsmittel (dazu gehören auch KI-Tools) verwendet habe und dass ich sowohl bei wörtlich übernommenen Aussagen (Zitaten) wie auch bei in eigenen Worten wiedergegebenen Aussagen und Gedanken anderer Autorinnen, Autoren oder KI-Tools (Paraphrasen) auf die Urheberschaft verwiesen habe. Alle Personen, die einen wesentlichen Beitrag zu dieser Arbeit geleistet haben, habe ich ebenfalls erwähnt.

Ort, Datum: ..... Unterschrift: .....

# I. Abstract

Die Fortschritte der künstlichen Intelligenz (*KI*) waren in den letzten Jahren so gross, dass Schlagzeilen wie „Experten bewerten die Gefahren durch KI ähnlich hoch wie bei Pandemien oder einem Atomkrieg“ (Deutschlandfunk, 2023), keine Seltenheit mehr sind. Gerade aufgrund der Fortschritte mächtiger *KI*-Tools stellte ich mir die Frage, ob Computer CAPTCHAs lösen können, die speziell dazu konzipiert sind, nur von Menschen gelöst werden zu können. Um diese Frage zu beantworten, beschäftigte ich mich zunächst mit dem Aufbau verschiedener Objekterkennungsmethoden, um sie dann praktisch an CAPTCHA Bildern zu testen. Ich beschränkte mich dabei auf die Lösung von CAPTCHAs, welche Fussgängerstreifen enthalten, da es mir v.a. um das Konzept ging. Das beste Computermodell konnte schliesslich jedes zweite CAPTCHA richtig lösen. Da auch Menschen CAPTCHAs nicht fehlerfrei lösen können, untersuchte ich auch, wie gut Menschen CAPTCHAs lösen. Überraschenderweise wurden von den Menschen im Durchschnitt nur 51.88% der CAPTCHAs identisch gelöst, was das Resultat des Computermodells noch eindrücklicher macht. Dieses Resultat zeigt zwar einerseits, dass mit Computern sehr viel möglich ist, was vor kurzem noch für unmöglich gehalten wurde, es stellt aber auch eine massive Gefahr dar. Webseiten verlassen sich auf CAPTCHAs, um sich vor Angriffen von Bots zu schützen. In der Zukunft werden also neue Sicherheitsmassnahmen erfunden und getroffen werden müssen, um das Internet frei von Bots zu halten.

# II. Vorwort

Ich möchte mich bei folgenden Personen bedanken, die meine Arbeit im Erstellungsprozess gelesen haben und mich auf Fehler aufmerksam machten, oder Verbesserungen vorschlugen:

- **Doris Walti Kümmerle**
- **Matthias Kümmerle**
- **Monika Kümmerle**
- **Karl Kümmerle**
- **Jonas von Sachs**

Mein Dank gilt ebenfalls meinen Probanden, welche meinen CAPTCHA Test durchgeführt haben.

Ich danke auch **Mike Mazurov**, der einen Datensatz mit Bildern von Fussgängerstreifen öffentlich zur Verfügung stellt.

Zuletzt möchte ich mich bei Herrn **Gérald Huber** bedanken, der mich während des ganzen Arbeitsprozesses als Betreuungsperson unterstützt hat.

# III. Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Motivation

Künstliche Intelligenz ist durch relativ leistungsfähige und leicht anzuwendende Systeme wie ChatGPT oder DALL·E seit einigen Jahren in weiten gesellschaftlichen Kreisen ein viel und zum Teil kontrovers diskutiertes Thema.

Da ich schon lange an Technik und Informatik interessiert bin, fragte ich mich, was hinter diesen bekannten Systemen steckt. Ich wollte also herausfinden, was *KI* im Kern ist, und ob sie ein Sicherheitsrisiko darstellt. Als dann auf einer Webseite ein CAPTCHA erschien, in welchem ich auf alle Kästchen mit Fussgängerstreifen klicken musste, um zu überprüfen, ob ich ein Mensch bin, kam mir die folgende Idee: Ich könnte versuchen, ein *KI*-basiertes Computermodell zu bauen, welches in der Lage ist, CAPTCHAs zu lösen. Ich überlegte mir nämlich, wie ironisch und fatal es wäre, wenn Computer den Test bestehen würden, der eigentlich dazu gedacht ist, Menschen von Maschinen zu unterscheiden. Dies könnte möglicherweise zu einem Sicherheitsrisiko werden. Nach einer ersten Recherche wurde mir schnell klar, dass es nicht einfach ein spezifisches *KI* Modell gibt, sondern dass viele verschiedene Methoden existieren, um spezifische Objekte in Bildern zu erkennen. Ich wollte nun den Unterschied der verschiedenen Methoden verstehen und herausfinden, welche die beste ist. Dieser Frage gehe ich also in meiner Maturitätsarbeit nach.

## 1.2 Ziele

**Leitfrage:**
Kann ein Computer ein CAPTCHA lösen, welches eigentlich genau dazu konzipiert wurde, nur von Menschen gelöst werden zu können?

**Teilfragen:**

1. Welche Faktoren beeinflussen die Effektivität und Genauigkeit verschiedener Computermodelle beim Lösen von CAPTCHAs?
2. Mit welcher Objekterkennungsmethode schafft es ein Computer, die meisten CAPTCHAs vollständig richtig zu lösen, also den Test zu bestehen?
3. Wie gut schneiden die Computermodelle im Vergleich mit Menschen ab?

Durch die Beantwortung der Teilfragen lassen sich Hypothesen klären wie etwa die Annahme, dass CAPTCHAs durch Computermodelle gelöst werden können, was ein Sicherheitsrisiko für uns darstellt. Eine weitere Hypothese wäre, dass die neuronalen Netzwerke, welche für den Durchbruch der uns bekannten *KI* verantwortlich sind, die besten Resultate liefern werden.

Das Ziel dieser Arbeit ist also, verschiedene computergestützte Objekterkennungsmethoden zu verstehen, theoretisch miteinander zu vergleichen und anschliessend in nachgestellten Versuchen zu testen. Die Resultate sollten anschliessend miteinander verglichen werden, um zu bestimmen, welche der Methoden sich am besten für die gewählte Art von CAPTCHAs eignet. Alle der getesteten Modelle einer Methode sollten vor der Evaluierung so weit optimiert worden sein, dass sie die bestmöglichen Ergebnisse erreichen. Der Optimierungsprozess wird genau beobachtet, dokumentiert und anschliessend analysiert.

# 2 Theoretische Grundlagen

In diesem Kapitel werde ich zuerst erklären, was genau ein CAPTCHA ist und dann die theoretischen Grundlagen hinter vier Methoden, mit welchen man mit Computern Objekte in Bildern erkennen kann, vorstellen. Das *Histogram of Oriented Gradients*, das *Template Matching* und *neuronale Netzwerke* sind drei voneinander komplett unabhängige Methoden. Die *Segmentierung* basiert auf neuronalen Netzwerken und ist somit sehr ähnlich zu diesen.

## 2.1 CAPTCHAs

CAPTCHA steht für „Completely Automated Public Turing test to tell Computers and Humans Apart“. CAPTCHAs werden also auf Webseiten verwendet, um Computer von Menschen zu unterscheiden, wobei die Tests für Computer schwierig und für Menschen einfach zu lösen sein sollten. Sie sollen also verhindern, dass Bots im Internet Abstimmungen verfälschen, Registrierungssysteme mit der Erstellung von gefälschten Accounts überfluten oder falsche Kommentare von Bots geschrieben werden.

Es gibt verschiedene Arten von CAPTCHAs, darunter die Text CAPTCHAs, Bild CAPTCHAs oder auch Audio CAPTCHAs (siehe Abb. 1). Es gibt aber auch immer mehr CAPTCHAs, bei welchen nur eine Box mit der Aufschrift „Ich bin kein Roboter“ angeklickt werden muss, ohne danach einen Test lösen zu müssen. Bei dieser Art von CAPTCHA werden die Aktionen des Benutzers wie z.B. die Mausbewegungen analysiert. Das funktioniert, da Menschen die Maus im Gegensatz zu Computern nicht perfekt geradlinig bewegen (vgl. Imperva, o.J.).

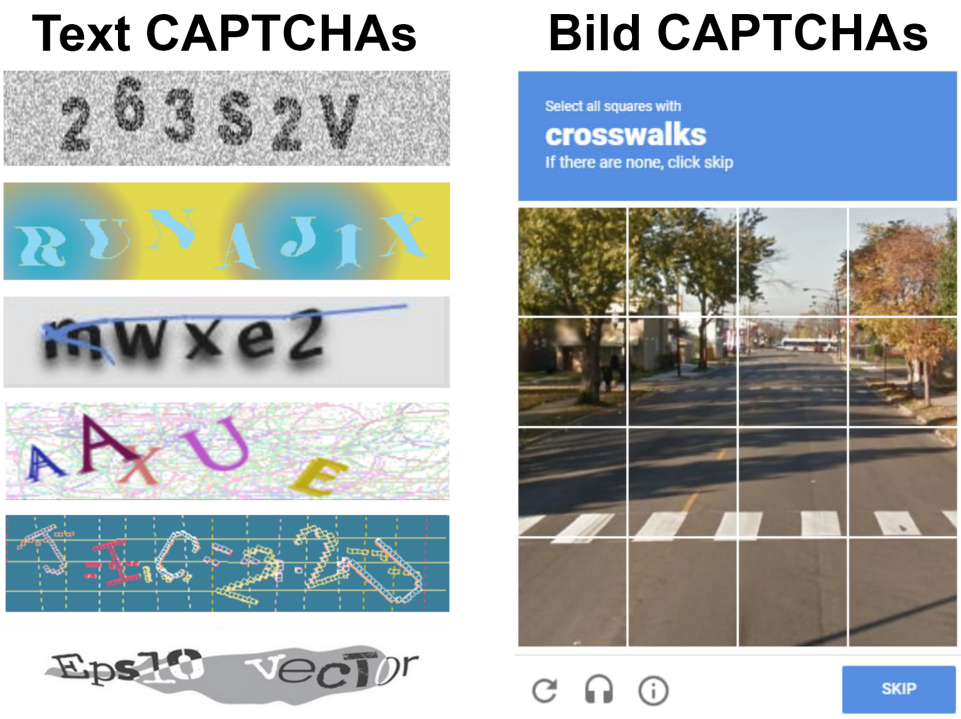



Abbildung 1: Verschiedene Arten von CAPTCHAs (vgl. Imperva, o.J. und Archana, 2024)

## 2.2 Histogram of Oriented Gradients (*HOG*)

Bei dieser Objekterkennungsmethode werden zuerst Histogramme aus „orientierten Gradienten“ berechnet, mit welchen anschliessend eine „Support Vector Machine“ trainiert wird. Mit dieser *Support Vector Machine* können dann Objekte erkannt werden.

### 2.2.1 Orientierte Bildgradienten

Gradienten beschreiben in der Mathematik, wie sich eine Funktion im Verhältnis zu ihren Argumenten und unabhängigen Variablen verändert. Auf ein Bild bezogen ist damit gemeint, wie sich die Helligkeitswerte der Pixel in eine Richtung verändern (vgl. Kang und Atul, 2019).

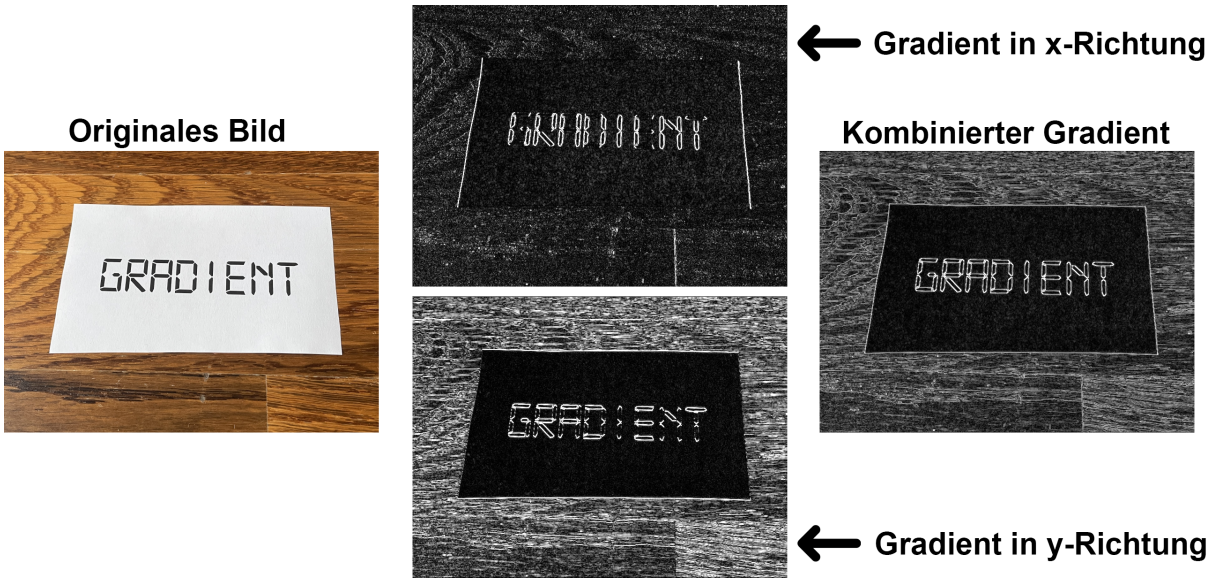



Abbildung 2: Beispiel eines Bildgradienten

An den Stellen, an denen eine grosse Veränderung der Pixelintensität zu sehen ist (Kanten), ist auch der Gradient hoch (in Abb. 2 mit weiss dargestellt).

Um ein Histogram of Oriented Gradients berechnen zu können, muss zuerst der Gradient für jedes Pixel des Bildes berechnet werden. Mit den Gleichungen in (1) wird der Gradient $G_x$ in x- und $G_y$ in y-Richtung berechnet. Die Variablen $r$ und $c$ stehen dabei für *row* und *column* und beziehen sich auf die Position eines Pixels im Bild. Das $I()$ steht für die Intensität des gefragten Pixels.

$$G_x(r,c) = I(r,c+1) - I(r,c-1); \quad G_y(r,c) = I(r-1,c) - I(r+1,c), \tag{1}$$

Anschliessend kann die Magnitude (Stärke) und der Winkel der Gradienten durch die Gleichungen (2) und (3) berechnet werden.

$$\text{Magnitude}(\mu) = \sqrt{G_x^2 + G_y^2} \tag{2}$$

$$\text{Winkel}(\theta) = \left|\tan^{-1}\left(\frac{G_y}{G_x}\right)\right| \tag{3}$$

Aus der Magnitude und der Richtung entsteht für jedes Pixel ein Vektor, der „Orientierte Bildgradient“ (siehe Abb. 3) (vgl. Tyagi, 2021).

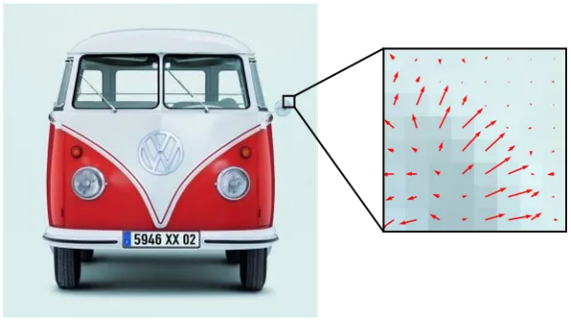


Abbildung 3: Orientierte Bildgradienten der Pixel (Nemutlu, 2022)

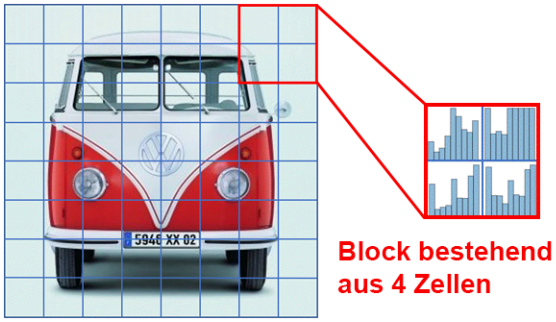



Abbildung 4: Histograms of Oriented Gradients (vgl. Nemutlu, 2022)

### 2.2.2 Berechnen des Histogramms

Ein Histogramm ist ein Diagramm, in welchem ein Datensatz durch rechteckige Balken dargestellt wird, wobei die Länge jedes Balkens proportional zur entsprechenden Häufigkeit des jeweiligen Wertes im Datensatz ist (vgl. Anderson, 2023).

Es wird nun für jede Zelle (die blauen Kästchen in Abb. 4) ein Histogramm erstellt. Es gibt verschiedene Methoden, um diese Histogramme zu erstellen, eine davon funktioniert wie in Abb. 5 dargestellt. Die Balken des Histogramms stellen jeweils eine Spannweite von Winkeln dar, die als „Bin“ bezeichnet werden. Wählt man neun *Bins*, so ergeben sich jeweils 180° : 9 = 20° Schritte. Der erste *Bin* stellt also die Winkel von 0°–20° dar, der zweite von 20°–40°. Dann wird die Magnitude jedes Pixels der untersuchten Zelle dem richtigen *Bin* angerechnet (vgl. Aishwarya, 2019).

Magnitude = 13.6
Orientierung = 30°

| **Magnitude** | | 13.6 | | | | |
|---|---|---|---|---|---|---|
| **Bin** | 0°-20° | 20°-40° | 40°-60° | 60°-80° | ... | 160°-180° |

Abbildung 5: Beispiel für die Erstellung eines Histograms of Oriented Gradients

Da die Bildgradienten sehr anfällig auf Lichtveränderungen sind, werden die Zellen mit den Histogrammen in grösseren Blöcken (rote Kästchen in Abb. 4) normalisiert. Die Grösse dieser Blöcke kann vom Benutzer definiert werden und wird „Block Size“ genannt. Diese Blöcke werden übers Bild geschoben, wobei die Distanz der Verschiebung auch festgelegt werden kann. Die Distanz, um welche der Block verschoben wird, nennt man „Block Stride“.

Es können noch weitere Parameter zur Bestimmung der Histogramme angegeben werden. Nachfolgend sind noch einmal die bekannten Parameter aufgeführt:

- **Cell Size:** Die Grösse der Zelle.
- **Block Size:** Die Grösse der Blöcke, über welche die Histogramme normalisiert werden.
- **Block Stride:** Die Distanz, um welche die Normalisierungsblöcke verschoben werden, bevor ein weiterer Normalisierungsschritt stattfindet.

### 2.2.3 Support Vector Machines (*SVM*)

Mithilfe von „Support Vector Machines“ (*SVM*) können jetzt Objekte anhand der *HOG*-Merkmale erkannt werden.

*SVM*s können zwei Gruppen Klassifizierungsprobleme lösen, also zwischen Option 1 und Option 2 unterscheiden. In meinem Fall wird zwischen Bildern mit und ohne Fussgängerstreifen unterschieden. Sie versuchen dabei, die beiden Gruppen mit einer „Hyperplane“ bestmöglich zu trennen (Abb. 6). Falls die Daten nicht linear sind, werden sie in höhere Dimensionen transformiert (wie in Abb. 7 gezeigt), wodurch die Möglichkeit entsteht, eine Hyperplane einzuzeichnen (vgl. Stecanella, 2017).

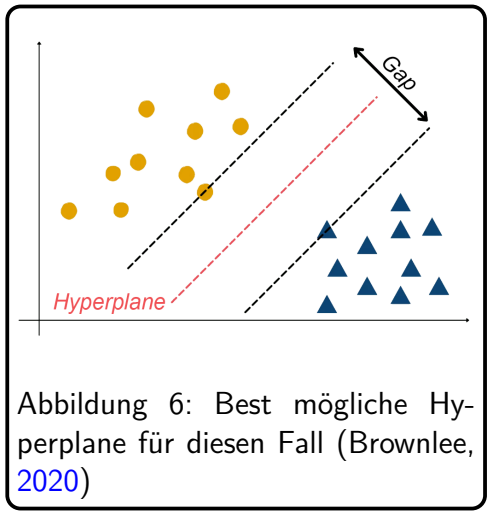



Abbildung 6: Best mögliche Hyperplane für diesen Fall (Brownlee, 2020)

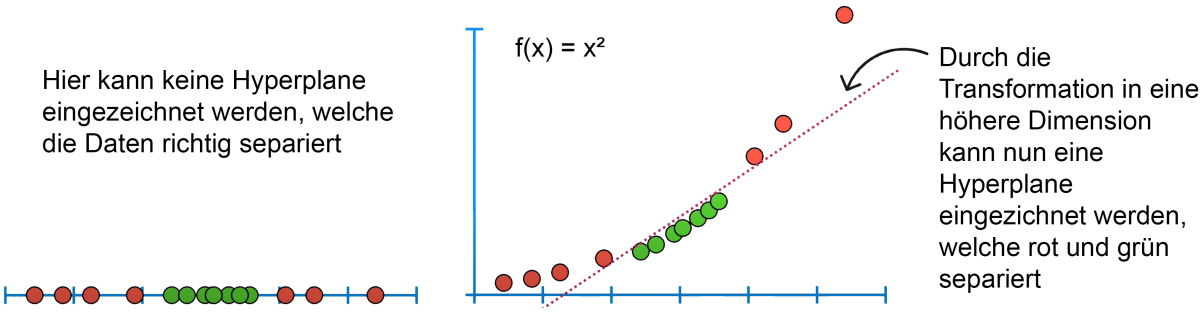



Abbildung 7: Hyperplane wird ermöglicht durch Versetzung in höhere Dimension (vgl. Starmer, 2020)

In der Praxis wird der *SVM*-Algorithmus durch sogenannte „Kernels“ implementiert. Kernels setzen die Daten in angemessene Dimensionen und berechnen eine geeignete Hyperplane (vgl. Brownlee, 2020).

In Abb. 8 wird ein unbekannter Punkt einer Gruppe zugeordnet. Da der unbekannte Punkt auf der Seite der orangen Punkte liegt, klassifiziert die *SVM* ihn auch als orange. Der eingezeichnete Wert $d$ ist die Distanz von der Hyperplane zum unbekannten Punkt und wird auch als „Threshold“ bezeichnet. Es gibt an, wie sicher sich das Modell ist, dass der Punkt zu den orangen Punkten gehört.

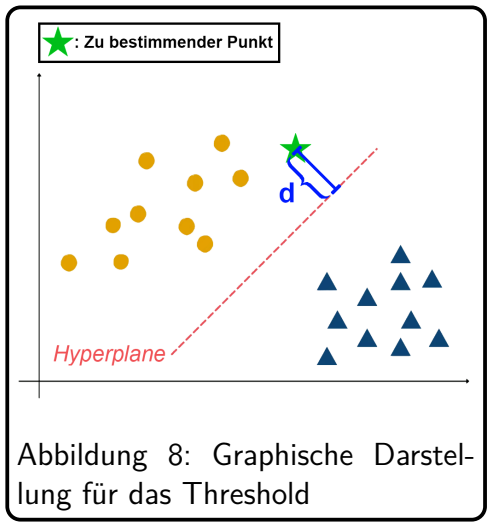



Abbildung 8: Graphische Darstellung für das Threshold

Bei der Bilderkennung müssen zuerst Teilbilder mit Objekt und ohne Objekt zu *HOG*s umgewandelt werden. Diese Histogramme werden dann in eine *SVM* gegeben, die anschliessend versucht, die Bilder mit dem Objekt von denen ohne Objekt mittels einer Hyperplane zu trennen. Schliesslich werden einzelne Regionen auf dem Bild angeschaut, und die *SVM* bestimmt dann, ob es sich an dieser Stelle um das Objekt handelt oder nicht.

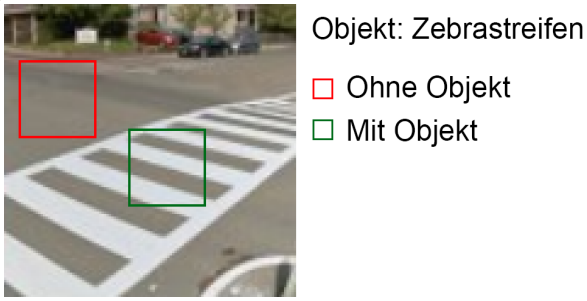


Abbildung 9: Region mit und ohne Objekt

Der Grund, weshalb die Bilder überhaupt zu *HOG*s umgewandelt werden, ist, dass es rechnerisch zu aufwendig wäre, eine Hyperplane für die Originalbilder zu berechnen. Die Histogramme sind nämlich viel kleiner als die Originalbilder, versuchen aber, die relevanten Informationen beizubehalten.

## 2.3 Template Matching

### 2.3.1 Idee hinter Template Matching

Beim Template Matching wird ein kleines Bild als Vorlage (Template) gewählt und anschliessend über das grosse Bild, in welchem etwas erkannt werden sollte, geschoben. Für jede Stelle der Vorlage auf dem Bild wird dann ein numerischer Vergleichswert berechnet. An den Stellen, an welchen der Vergleichswert am grössten ist, ist die Wahrscheinlichkeit, dass sich dort das gesuchte Objekt befindet, am höchsten (vgl. Adaptive Vision, o.J.).

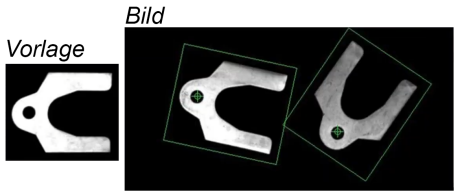


Abbildung 10: Template Matching (vgl. Orbis, 2020)

### 2.3.2 Berechnung des Vergleichswerts

Es gibt verschiedene Methoden, einen Vergleichswert zu berechnen, die „Cross-Correlation“ ist aber die häufigste, weswegen hier nur diese erklärt wird. Dabei wird das Template an einer Stelle auf dem Bild angesetzt. Nun wird jeder Pixelwert des Templates mit dem korrespondierenden Pixelwert des Bildes multipliziert. Anschliessend werden alle soeben berechneten Werte addiert. Das ist grafisch in Abb. 11 dargestellt.

$$\text{Cross-Correlation}(Image1, Image2) = \sum_{x,y} Image1(x,y) \cdot Image2(x,y) \tag{4}$$

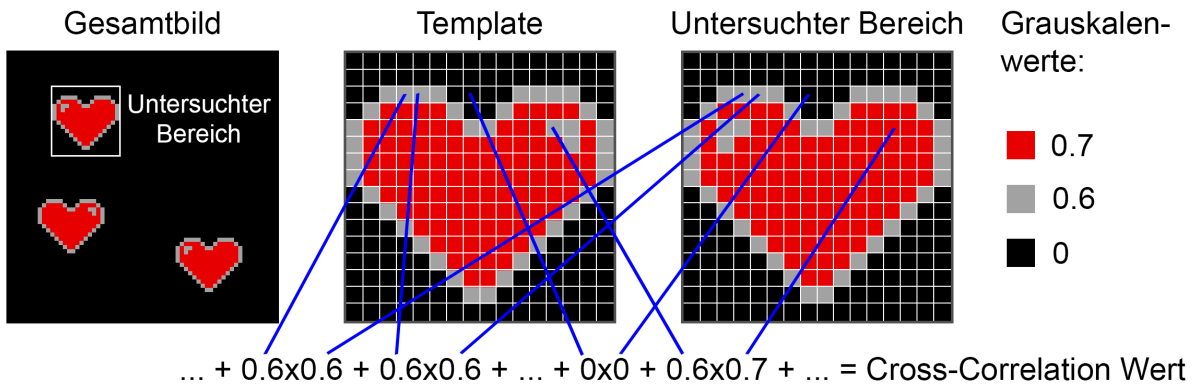



Abbildung 11: Graphische Darstellung der Berechnung des Cross-Correlation Werts

Das Problem hierbei ist jedoch, dass diese Summe in einem generell helleren Bild viel grösser ist. Um dieses Problem zu lösen, wurde die „Normalisierte Cross-Correlation (NCC)“ eingeführt. Bei dieser wird die Gesamtbelichtung des Bildes einbezogen, und der Endwert liegt zwischen -1 und 1, wobei 1 einer perfekten Übereinstimmung entspricht. Es geht hier vor allem darum, das Konzept zu verstehen, nicht darum, wie genau man den NCC-Wert tatsächlich berechnet.

$$NCC(Image1, Image2) = \frac{1}{N\sigma_1\sigma_2}\sum_{x,y}(Image1(x,y) - \overline{Image1}) \cdot (Image2(x,y) - \overline{Image2}) \tag{5}$$

| | |
|---|---|
| N | Anzahl Pixel |
| $\sigma_1$ und $\sigma_2$ | Standardabweichungen der Pixelintensitäten der Bilder, berechnet durch $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$ $x_i$: individuelle Pixelintensität; $\mu$: durchschnittliche Pixelintensität |
| Image(x,y) | Intensität des Pixels an der Koordinate (x,y) |
| $\overline{Image}$ | Durchschnittliche Pixelintensität im Bild |

Tabelle 1: Bedeutung der Variabeln bei NCC

### 2.3.3 Kriterien zur Bestimmung von Treffern

Es gibt zwei Kriterien, welche beide zutreffen müssen, um finale Treffer zu bestimmen (siehe Abb. 12):

- Der NCC-Wert muss grösser als eine bestimmte Zahl (z.B. 0.75) sein.
- Es muss sich um ein lokales Maximum handeln. D.h. der NCC-Wert dieses Pixels muss grösser sein als der NCC-Wert der benachbarten Pixel.

(vgl. Adaptive Vision, o.J.)

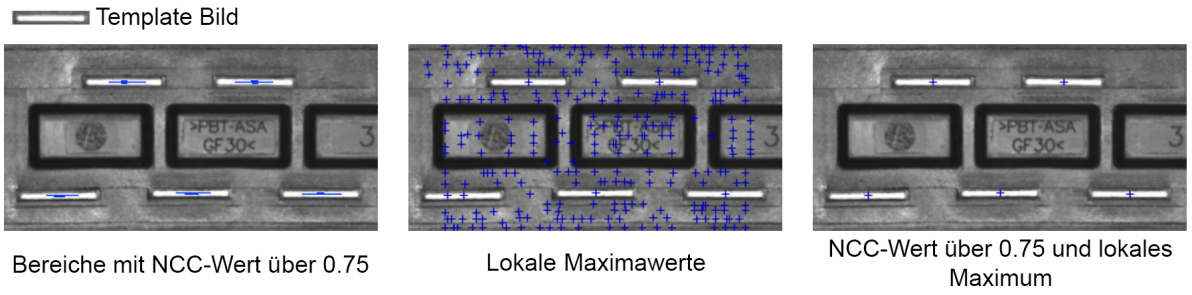



Abbildung 12: Identifikation von Übereinstimmungen (vgl. Adaptive Vision, o.J.)

### 2.3.4 Lösung einfacher Probleme

Ein Problem des Template Matchings ist, dass die Objekte nur in genau der Grösse des Templates erkannt werden. Unterscheiden sich die Objekte in ihrer Grösse, ist auch der NCC-Wert klein. Dieses Problem kann aber mit „Image Pyramids“ gelöst werden. Dabei wird der gesamte Vorgang mit gleichen Templates verschiedener Grössen wiederholt (vgl. Goekcenaz, 2023).
Dasselbe Problem gibt es für verschiedene Orientierungen. Dieses Problem wird aber gleich gelöst, indem Templates mit verschiedenen Orientierungen getestet werden.

Kombiniert man die beiden Lösungen miteinander, erhält man ein relativ robustes Erkennungssystem (vgl. Adaptive Vision, o.J.).

## 2.4 Neuronale Netzwerke (*NN*s)

Künstliche neuronale Netzwerke sind eine abstrakte Repräsentation des menschlichen Nervensystems. Das menschliche Nervensystem beinhaltet Neuronen, welche über Axone miteinander kommunizieren. Aus einer Nervenzelle (biologisches Neuron) gehen viele Dendriten und ein langes Axon aus. Am Ende verästelt sich das Axon, wobei am Ende der Verästlungen die Synapsen sitzen. Die Synapsen eines Neurons verbinden sich mit den Dendriten anderer Neuronen. Die biologischen Neuronen empfangen von anderen Neuronen elektrische Signale und senden darauf ihre eigenen Signale weiter (vgl. Zaccone und Rezaul, 2018, S. 11-12).

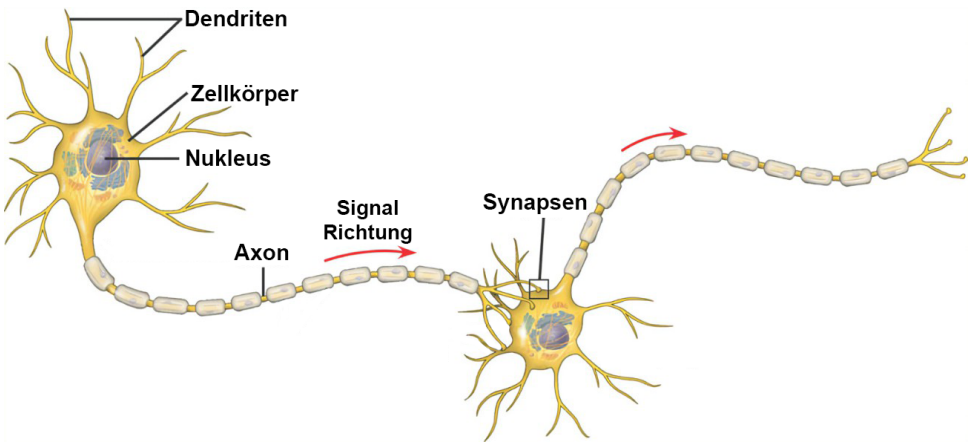



Abbildung 13: Schematische Abbildung menschlicher Neuronen (vgl. ResearchGate, o.J.)

### 2.4.1 Aufbau

Neuronale Netzwerke sind aus mehreren Ebenen aufgebaut, welche jeweils eine bestimmte Anzahl Neuronen enthalten (vgl. 3Blue1Brown, 2017). Jedes Neuron ist mit jedem Neuron der benachbarten Ebenen verbunden und berechnet aufgrund der empfangenen Signale ein eigenes Signal, welches das Neuron dann wie die biologischen Neuronen weiter gibt. Ein Neuron ist also ein mathematisches Modell, welches aus einem oder mehreren Inputs eine Zahl als Output berechnet (vgl. Nikhil, 2024). Die berechneten Outputs befinden sich normalerweise in einem Intervall, oft im Intervall [0, 1]. Am Anfang eines Netzwerkes steht die „Input Ebene“ und am Ende die „Output Ebene“. Dazwischen hat es versteckte Ebenen. Die Anzahl versteckter Ebenen und die Anzahl der Neuronen der verschiedenen Ebenen kann frei bestimmt werden.

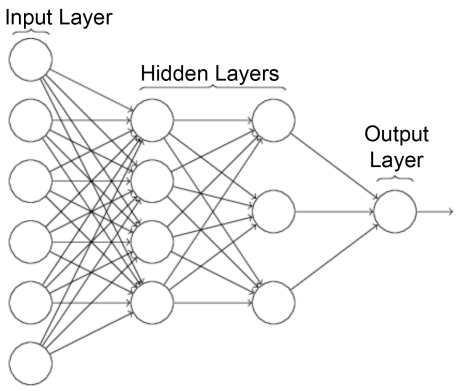



Abbildung 14: Aufbau eines neuronalen Netzes (vgl. Nielsen, 2015)

### 2.4.2 Funktionsweise neuronaler Netzwerke

Die Funktionsweise neuronaler Netzwerke wird hier durch ein Beispiel, in welchem die Zahlen von 0–9 erkannt werden sollen, veranschaulicht. Das Ganze ist stark vereinfacht, wobei die Grundidee der Funktionsweise von neuronalen Netzwerken aber beibehalten wird. Das Beispielnetzwerk nimmt als Input 28x28 Pixel grosse Bilder, auf denen jeweils eine Zahl abgebildet ist.

In der Inputebene hat es für jeden Pixel des Bildes ein Neuron ($28 \cdot 28 = 784$ Neuronen), wobei der Wert jedes Neurons den Grauskalenwert des dazugehörigen Pixels annimmt. Diese Grauskalenwerte sind Werte zwischen 0 für weiss und 1 für schwarz. In der Outputebene hat es für die möglichen Zahlen zwischen 0 und 9 jeweils genau ein Neuron. Das Output Neuron mit dem höchsten Wert ist laut dem Modell im Beispiel der Zahlenerkennung die wahrscheinlichste Zahl (vgl. 3Blue1Brown, 2017).

Neuronale Netzwerke benötigen die Ebenen, um das gewünschte Endresultat Schritt für Schritt herleiten zu können. Die Zahlen können beispielsweise in Einzelteile unterteilt werden (Abb. 15). Jedes dieser Einzelteile wird anschliessend einem Neuron zugeteilt. Wenn dann z.B. die Wahrscheinlichkeiten der Neuronen der Einzelteile der Zahl 9 hoch sind, so handelt es sich vermutlich um eine 9. Dies ist in der dritten Ebene des neuronalen Netzwerks in Abb. 17 dargestellt.

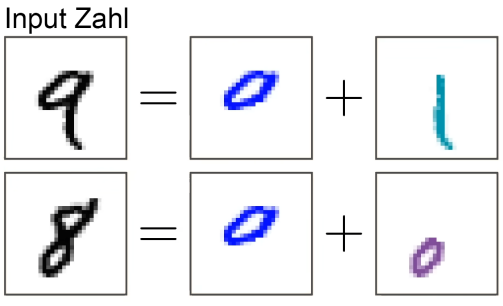



Abbildung 15: Unterteilung der Zahlen (vgl. 3Blue1Brown, 2017)

Mit einer weiteren Ebene können die Einzelteile in noch kleinere Strichsegmente unterteilt werden, wie in Abb. 16 beispielhaft für die obere Schlaufe der Zahl „9“ dargestellt ist. Dabei wird wieder jedes Strichsegment einem Neuron dieser weiteren Ebene zugeteilt. Wenn nun diese Neuronen, welche die obere Schlaufe bilden, alle hohe Wahrscheinlichkeiten haben, dann ist eine gute Übereinstimmung mit dem Einzelteil vorhanden und es handelt es sich vermutlich um diese Schlaufe. Dies ist in Abb. 17 in der zweiten Ebene illustriert.

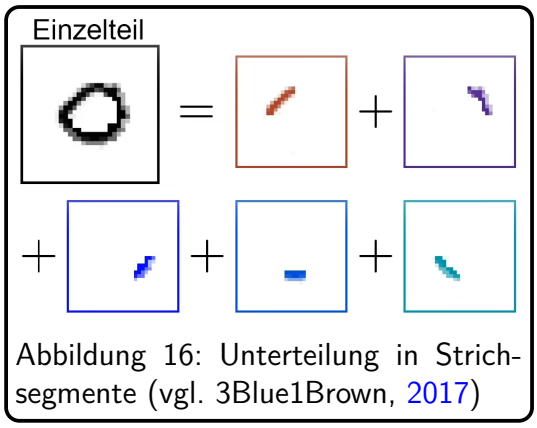



Abbildung 16: Unterteilung in Strichsegmente (vgl. 3Blue1Brown, 2017)

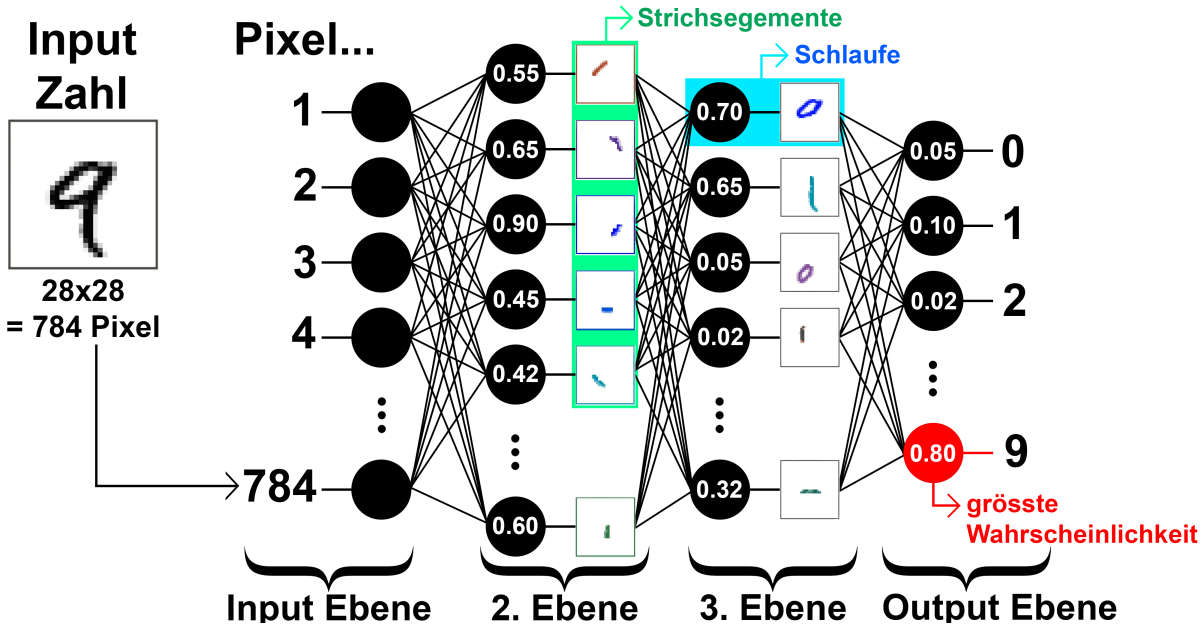



Abbildung 17: Die in Kapitel 2.4.2 aufgezeigte Funktionsweise neuronaler Netzwerke

Um die Werte der Neuronen, welche die Strichsegmente repräsentieren, zu bestimmen, muss die jeweilige Übereinstimmung mit der Input-Zahl ermittelt werden. Je stärker also das Strichsegment in der Input-Zahl ausgeprägt ist, desto grösser soll der Wert des korrespondierenden Neurons sein.

Für alle Pixel im Input-Bild werden Gewichtungen erstellt, welche Zahlen zwischen -1 und 1 annehmen können. Pixel innerhalb des Strichsegments bekommen eine Gewichtung von 1, Pixel, welche direkt ans Strichsegment angrenzen, die Gewichtung -1 und Pixel, welche weiter entfernt sind, die Gewichtung 0 (vgl. Abb. 18). Die tiefe Gewichtung von -1 an der Grenze bringt den Vorteil, dass die Grenze schärfer und das Gesamtresultat genauer wird.

Der Wert des Neurons wird nun ermittelt, indem der Grauskalenwert jedes Pixels der Input-Zahl mit der korrespondierenden Gewichtung des Pixels multipliziert und die Produkte aufsummiert werden.

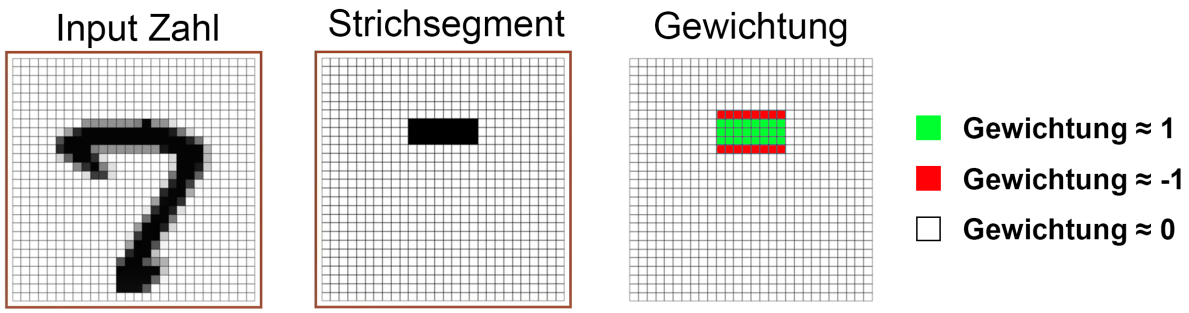



Abbildung 18: Visualisierung der unterschiedlichen Gewichtung von Pixeln (vgl. 3Blue1Brown, 2017)

Die Werte, die jetzt für die Neuronen, welche Kanten repräsentieren, herausgekommen sind, können irgendeinem Wert zwischen unendlich und -unendlich entsprechen. Da sie aber in diesem Beispiel Werte zwischen 0 und 1 sein müssen, gibt es sogenannte Aktivierungsfunktionen (vgl. 3Blue1Brown, 2017). Diese bringen die berechneten Kantenwerte der Neuronen in den gewünschten Bereich von 0–1, da es sich um eine Wahrscheinlichkeit handelt. Eine der häufigst verwendeten Aktivierungsfunktionen ist die „Sigmoid“ Funktion (vgl. Sharma et al., 2017).

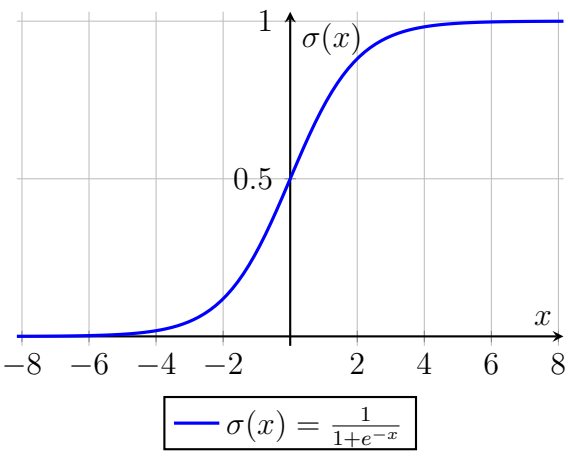



Abbildung 19: Sigmoid Funktionsgraph

Um also den Wert des nächsten Neurons mit der Sigmoid Funktion zu berechnen, nimmt man die Gleichung (6), wobei $w$ jeweils die dazugehörige Gewichtung und $a$ der Wert des vorherigen Neurons ist.

$$a_k^{(1)} = \sigma(w_1a_1 + w_2a_2 + w_3a_3 + ... + w_na_n) \tag{6}$$

In Abb. 20 sind nur die Gewichtungen für ein einzelnes Neuron eingezeichnet. Das Ganze wird aber selbstverständlich für alle Neuronen so gemacht.

Ein Neuron wird dann aktiviert, wenn dessen Wert $> 0.5$ ist, da mit der Sigmoid Aktivierungsfunktion der Wert 0 zu 0.5 wird ($\sigma(0) = 0.5$). Falls ein Neuron aber erst aktiviert werden soll, wenn dessen Wert vor der Aktivierungsfunktion grösser n ist, dann passt man die Gleichung so an:

$$a_k^{(1)} = \sigma(w_1a_1 + w_2a_2 + w_3a_3 + ... + w_na_n - n) \tag{7}$$

Den Wert n nennt man „Bias “.

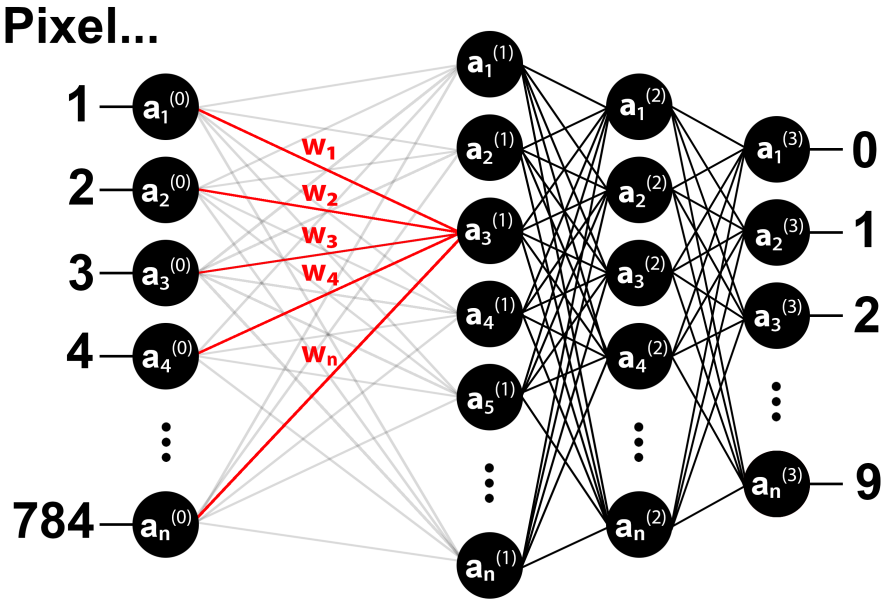



Abbildung 20: Gewichtungen in neuronalen Netzwerken

Unser Beispielnetzwerk hat 784 Neuronen in der Input Ebene, jeweils 16 in den 2 versteckten Ebenen und 10 in der Output Ebene. Jedes Neuron ist mit jedem Neuron der danebenliegenden Ebenen verbunden. Jede Verbindung hat eine Gewichtung $w$ und jedes Neuron, ausser diejenigen in der Input Ebene, jeweils noch ein *Bias*. In diesem vergleichsweise relativ kleinen Netzwerk können also bereits

$$((784 \cdot 16) + (16 \cdot 16) + (16 \cdot 10)\ Gewichte) + (16 + 16 + 10\ Biases) = 13'002$$

Einstellungen angepasst werden. Wenn nun davon gesprochen wird, dass ein Netzwerk lernt oder trainiert wird, dann geht es darum, die optimalen Werte der Gewichte und *Biases* zu finden (vgl. 3Blue1Brown, 2017).

### 2.4.3 Trainieren neuronaler Netzwerke

Der Lernprozess eines neuronalen Netzwerkes wird als iterativer Prozess konfiguriert. Das heisst, dass der Lernvorgang schrittweise erfolgt. Die Parameter des Modells werden basierend auf seiner Leistung an bekannten Beispielen aus dem Trainingsdatensatz angepasst. Das Ziel ist es, eine „Cost Funktion“, welche angibt, wie stark das Verhalten des Modells von dem gewünschten Verhalten abweicht, zu minimieren. Ein sehr häufig verwendeter Lernalgorithmus ist der „Backpropagation Algorithm“ (dt. Rückwärtsausbreitungsalgorithmus). Dieser Algorithmus funktioniert folgendermassen:

1. Allen Parametern des Netzwerks werden zufällige Werte zugeteilt.

2. In allen Trainingsfällen werden die folgenden Schritte befolgt.

**Vorwärtsausbreitung:**
Hier wird die *Cost Funktion* des Netzwerks berechnet (vgl. Zaccone und Rezaul, 2018). Eine häufig verwendete *Cost Funktion* ist die „Mean squared Error Cost Funktion". Dabei wird für jedes Trainingsbild die Abweichungen der Output Neuronen von den richtigen Werten quadriert und anschliessend aufsummiert. Anschliessend wird noch mit einem Faktor multipliziert, der die Anzahl Neuronen in der Output Ebene berücksichtigt (vgl. Nielsen, 2015).

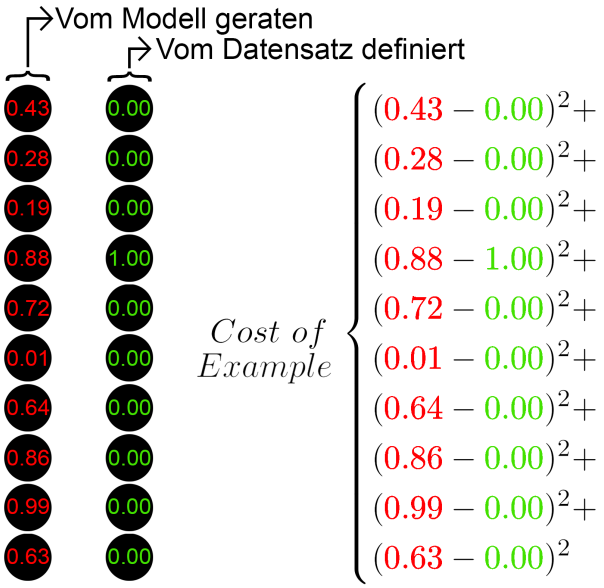



Abbildung 21: Berechnung der *Cost Funktion*

Beachtung der Anzahl Neuronen

$$C(w,b) = \frac{1}{2n}\sum_{i=1}^{n}(y(x_i)-a_i)^2 \quad (8)$$

| $C(w,b)$ | *Cost Funktion* |
|---|---|
| $w$ | Gewichte |
| $b$ | *Biases* |
| $n$ | Anzahl Bilder |
| $y(x)$ | definierter Wert |
| $a$ | geratener Wert |

Tabelle 2: Parameter der *Mean squared Error Cost Funktion*

**Rückwärtsausbreitung:**
Diese beginnt bei der Output Ebene und arbeitet sich bis zur Input Ebene zurück. Jedes Output Neuron sollte im Optimalfall genau dem Wert aus dem Datensatz entsprechen. Da mit der *Cost Funktion* berechnet wurde, wie sehr sich ein Neuron verändern sollte, kann dieses, um zum richtigen Wert zu gelangen, ausrechnen, welche Werte die Gewichtungen und *Biases* der Verbindungen zur zweitletzen Ebene haben sollten. Jetzt würde aber jedes Neuron der Output Ebene diese Gewichte und *Biases* verschieden anpassen, weswegen der Durchschnitt genommen wird. Zudem müssen noch alle anderen Bilder aus dem Datensatz betrachtet werden. Auch hier wird wieder der Durchschnitt genommen. So bewegt sich das Modell schrittweise in die richtige Richtung. Diese Art der Modelloptimierung nennt man „Gradient Descent" (vgl. Zaccone und Rezaul, 2018).

Da die Optimierung mit *Gradient Descent* aber sehr lange dauert, wird es in der Praxis leicht anders gehandhabt. Anstatt dass für jeden Schritt der Durchschnitt von allen Trainingsdaten berechnet wird, wird für jeden Schritt ein kleiner jeweils zufällig gemischter Teil des Datensatzes gewählt. In diesem kleinen Teil sind immer noch genügend Trainingsdaten vorhanden, um eine akkurate Repräsentation des ganzen Datensatzes zu bilden. Das Modell bewegt sich also bei einem Schritt trotzdem noch in die richtige Richtung, es ist einfach nicht mehr der direkteste Weg. Da die Berechnungszeiten aber deutlich kürzer sind, führt diese Methode trotzdem schneller zum Ziel. Diese Methode nennt man „Stochastic Gradient Descent" (vgl. 3Blue1Brown, 2017).

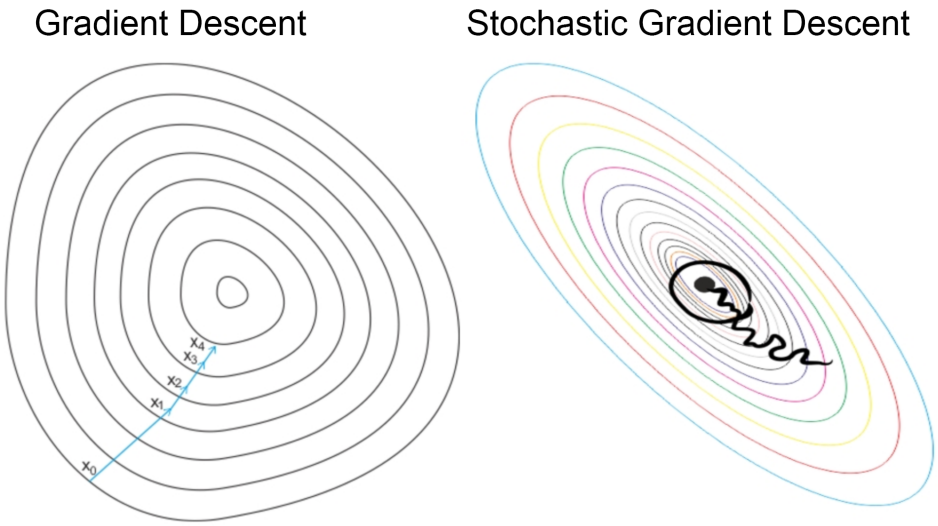



Abbildung 22: Vergleich von *Gradient Descent* mit *Stochastic Gradient Descent* (Zaccone und Rezaul, 2018)

Das Modell beginnt das Training mit den sog. Aufwärmschritten. Diese funktionieren gleich wie die normalen Trainingsschritte, nehmen allerdings kleinere Veränderungen an den Parametern des Modells vor. Aufwärmschritte helfen bei der Stabilisierung des Trainingprozesses und sind somit eine Starthilfe für das Training von *NN*s. Die Anzahl der Aufwärmschritte kann vom Benutzer des Modells bestimmt werden (vgl. Baeldung, 2024).

## 2.5 Segmentierung

### 2.5.1 Einführung

Bei der gewöhnlichen Objekterkennung werden jeweils nur rechteckige Regionen mit dem Objekt im Bild gesucht und in der Form von *Bounding Boxes* angegeben. Bei der Segmentierung hingegen werden pixelgenaue Masken für die Objekte erstellt.

Es gibt zwei Arten von Segmentierung, die „semantische Segmentierung“ und die „Segmentierung von Instanzen“ (vgl. Abb. 23).

**Semantische Segmentierung:**
Bei der semantischen Segmentierung wird jedes Pixel auf dem Bild einer Klasse zugeteilt. Klassen sind dabei die verschiedenen Objektarten, die erkannt werden sollen wie z.B. *Autos*, *Strasse* oder *Hintergrund*. Es gibt dabei immer mindestens zwei Klassen, ein Objekt und den Hintergrund.

**Segmentierung von Instanzen:**
Bei der Segmentierung von Instanzen wird aktiv nach einem Objekt gesucht, es werden also nur den Pixeln von gewünschten Objekten Klassen zugeteilt. Zudem werden auch Objekte derselben Klasse voneinander unterschieden, so z.B. *Auto 1* von *Auto 2* (vgl. IBM, o.J.).


Abbildung 23: Die zwei Arten von Segmentierung (IBM, o.J.)

Da für den weiteren Verlauf dieser Arbeit nur die semantische Segmentierung von Relevanz ist, wird hier nur diese weiterverfolgt.

### 2.5.2 Funktionsweise der Semantischen Segmentierung

Es gibt viele verschiedene Möglichkeiten, wie man Segmentierung angehen kann. Segmentierungsmodelle, die aus neuronalen Netzwerken bestehen, sind aber meist die besten, weswegen hier nur diese weiter verfolgt werden. Selbst hier gibt es noch etliche mögliche Aufbaustrukturen, aber die meisten ähneln sich stark. Diese Modelle bestehen aus zwei Prozessen, einem „Upsampling-“ und einem „Downsampling Prozess“.

Die beiden Prozesse machen die Segmentierung nicht genauer, sondern werden lediglich durchgeführt, um Zeit zu sparen. Müssten Segmentierungsmodelle ihre komplizierte Aufgabe mit den vollaufgelösten Bildern durchführen, wäre der Rechenaufwand zu gross. Weil die Segmentierungmodelle aber auch so sehr gute Resultate erlangen, überwiegt der Zeitvorteil der verlorenen Genauigkeit.

**Downsampling:**
Beim *Downsampling* wird die Auflösung der Bilder durch einen „Encoder“ verschlechtert. Meist verwendet man dazu in der Segmentierung ein Verfahren, welches sich „Max-Pooling“ nennt (vgl. Fezan, 2019). Beim *Max-Pooling* wird aus einem Pixelblock immer der höchste Wert für das schlechter aufgelöste Bild verwendet. Zudem wird die Position des Pixels mit dem grössten Wert innerhalb des Pixelblocks für später gespeichert. Diese Werte nennt man „Pooling Indices“, wobei 0 der oberen linken, 1 der unteren linken, 2 der oberen rechten und 3 der unteren rechten Ecke entspricht (vgl. Dhanush, 2023). Dieser Schritt verkleinert die Daten massiv, wodurch das anschliessend verwendete neuronale Netzwerk deutlich schneller trainiert werden kann.

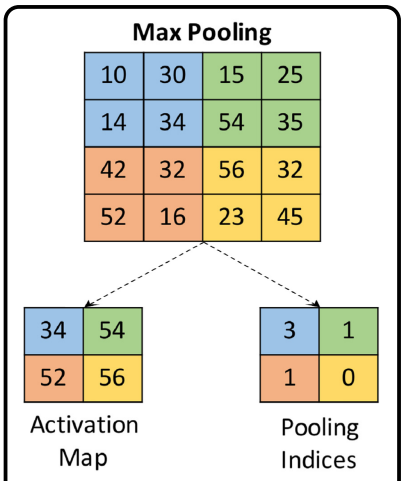



Abbildung 24: *Max-Pooling* Gonçalves et al., 2020

**Upsampling:**
In diesem Prozess wird das soeben vereinfachte Bild durch einen „Decoder“ wieder vergrössert. Dazu werden die Bilddimensionen in jedem Schritt verdoppelt, z.B. von einem 16x16 Bild zu einem 32x32 Bild. Das wird so oft wiederholt, bis das Bild wieder die Originalgrösse erreicht hat. Die Werte des verkleinerten Bilds werden nun an der vorher gespeicherten Stelle wieder eingefügt (siehe Abb. 25).
Das so entstandene Bild hat sehr viele Pixel mit dem Wert $Null$. Die Werte dieser Pixel werden nun durch ein neuronales Netzwerk, welches trainiert werden muss, geschätzt. Anschliessend berechnet ein weiteres neuronales Netzwerk für jeden Pixel die Wahrscheinlichkeit, zu einer bestimmten Klasse zu gehören. Dem Pixel wird die Klasse mit der grössten Wahrscheinlichkeit zugewiesen (vgl. Fezan, 2019).

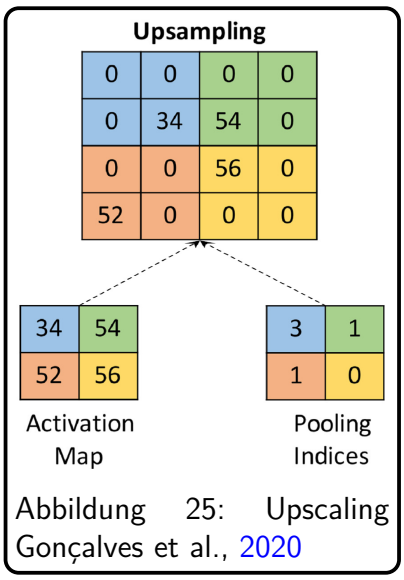


Abbildung 25: Upscaling Gonçalves et al., 2020

Die gesamte Struktur eines Modells für die semantische Segmentierung ist in Abb. 26 schematisch dargestellt.

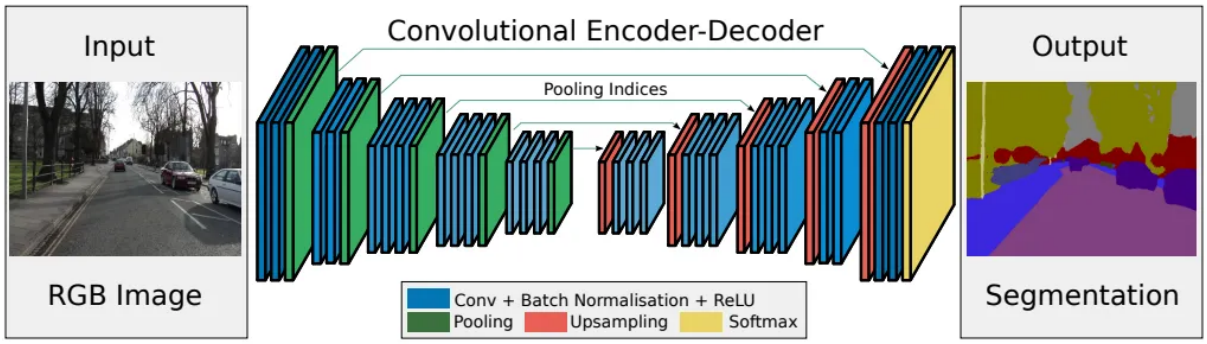


Abbildung 26: Gesamter Aufbau eines Segmentierungmodells (Fezan, 2019)

# 3 Methode

In diesem Abschnitt wird das praktische Vorgehen der Versuche der verschiedenen Objekterkennungsmethoden erläutert.

## 3.1 Erstellung eines Datensets

Um ein Computermodell trainieren zu können, braucht es einen Datensatz. Ein Datensatz ist eine grosse Sammlung geordneter und normierter Daten. Ich möchte in meiner Maturitätsarbeit einem Computer das Erkennen von Fussgängerstreifen auf Bildern beibringen, wozu ich also viele Bilder brauche, auf welchen Fussgängerstreifen abgebildet sind. Zudem müssen alle Bilder richtig beschriftet sein.

### 3.1.1 Sammeln von Bildern

Das Sammeln von Bildern ist an sich nicht schwierig, kann jedoch ein aufwendiger, mühseliger Prozess sein. Generell gibt es 2 Möglichkeiten, um an Bilder zu gelangen:

1. Man kann jemandem den Auftrag erteilen, eine gewisse Anzahl an Bildern zu sammeln/zu erstellen oder dies selbst tun. Das kann sehr kostspielig oder zeitaufwendig sein, besonders wenn man beachtet, dass es für einen guten Datensatz tausende von Bildern braucht. Gegebenenfalls führt jedoch kein Weg an dieser Methode vorbei.

2. Alternativ kann man das Internet nach einem bereits existierenden Datensatz durchforsten. Je nach Quelle braucht man dazu eine Lizenz oder kann ihn gratis herunterladen. Dazu muss dann noch beachtet werden, dass die Bilder allenfalls noch in das gewünschte Format gebracht werden müssen. Dies ist jedoch noch immer weniger Arbeit, als den gesamten Datensatz selbst zu erstellen.

In meinem spezifischen Fall konnte ich einen Datensatz mit 1240 Bildern mit Fussgängerstreifen von Mazurov, 2022 im Internet finden.

### 3.1.2 Labeling der Bilder

Um das Computermodell mit den Bildern trainieren zu können, muss es auch wissen, wo genau im Bild die Fussgängerstreifen sind. Die Bilder müssen also alle beschriftet werden. Diesen Prozess bezeichnet man als „Labeling“.
Das *Labeling* kann je nach Anforderungen und Ziel sehr unterschiedlich aussehen. Auch hier gilt wie zuvor, dass die Beschriftungen entweder in einem heruntergeladenen Datensatz inbegriffen sein können oder selbst erstellt werden müssen. Hier ist es aber öfters so, dass die Beschriftungen selbst gemacht werden müssen, damit sie genau den Anforderungen entsprechen.

Nun gilt aber noch zu klären, wie solche Beschriftungen am Beispiel der Fussgängerstreifen aussehen würden.

**Bounding Boxes:**
*Bounding Boxes* (dt. Begrenzungsrahmen) sind Rechtecke, welche um das gewünschte Objekt gesetzt werden. Sie wurden in dieser Arbeit für alle Objekterkennungsmethoden, ausser für die Segmentierung verwendet. Es gibt verschiedene Speicherformate für diese *Bounding Boxes*, es werden aber immer Werte gespeichert, mit welchen die *Bounding Boxes* eindeutig reproduzierbar sind. Ich habe in meinem Fall das „Pascal VOC Format“ verwendet. Darin angegeben sind unter anderem die x- und y-Koordinate der oberen linken und unteren rechten Ecke des Rechtecks. Es können auch mehrere *Bounding Boxes* pro Bild verwendet werden.
Dazu sind noch weitere Informationen wie z.B. eine Bezeichnung für das in der *Bounding Box* enthaltene Objekt enthalten. In meinem Fall wäre das bei allen „Crosswalk“.

Die Labels waren in meinem Datensatz nicht enthalten, weswegen ich alle Bilder selbst beschriften musste. Für diesen Prozess gibt es Programme, in denen man mit der Maus Rechtecke in die Bilder zeichnen kann, welche anschliessend vom Programm im Pascal VOC Format gespeichert werden.
In Abb. 27 ist ein Minimalbeispiel für ein Dokument im Pascal VOC Format abgebildet.

```
<annotation>
    <filename>Cross (1).png</filename>
    <object>
        <name>Crosswalk</name>
        <bndbox>
            <xmin>93</xmin>
            <ymin>0</ymin>
            <xmax>111</xmax>
            <ymax>17</ymax>
        </bndbox>
    </object>
</annotation>
```

Abbildung 27: Minimalbeispiel für ein Dokument im Pascal VOC Format

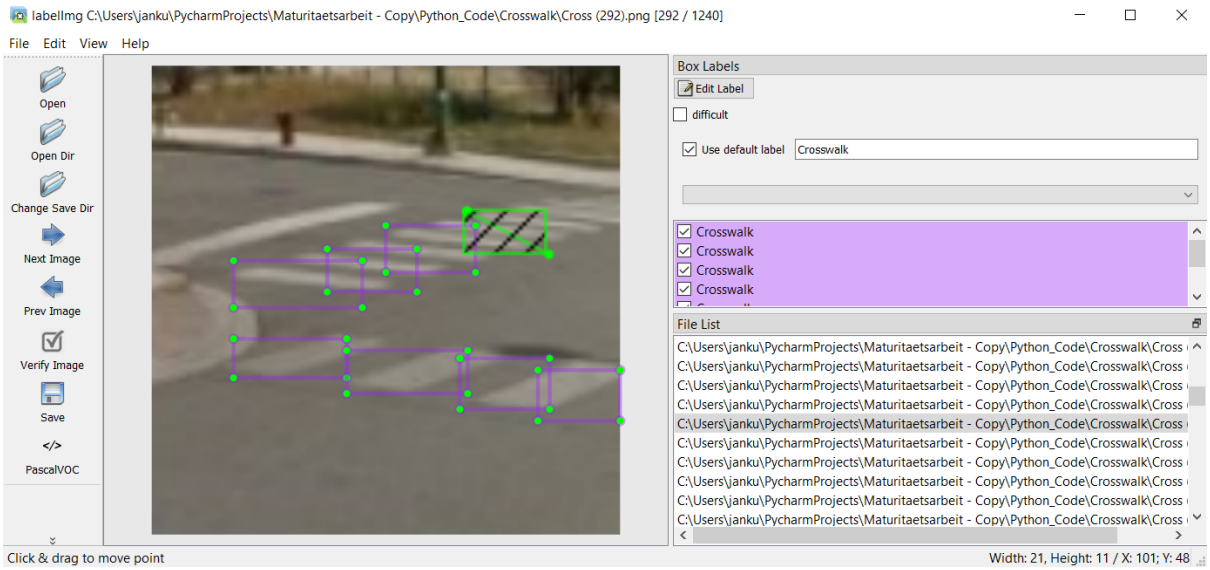



Abbildung 28: Der Prozess des *Labelings* mit *Bounding Boxes*

**Polygone:**
Für die Segmentierung müssen die Daten in einer Art von Polygon vorliegen. Polygone sind Vielecke, welche ebenfalls als Beschriftungsmethode verwendet werden können. Auch dafür gibt es verschiedene Speicherformate und Programme, um die Beschriftungen zu erstellen. Der Prozess ist aber vergleichbar mit dem der *Bounding Boxes*.

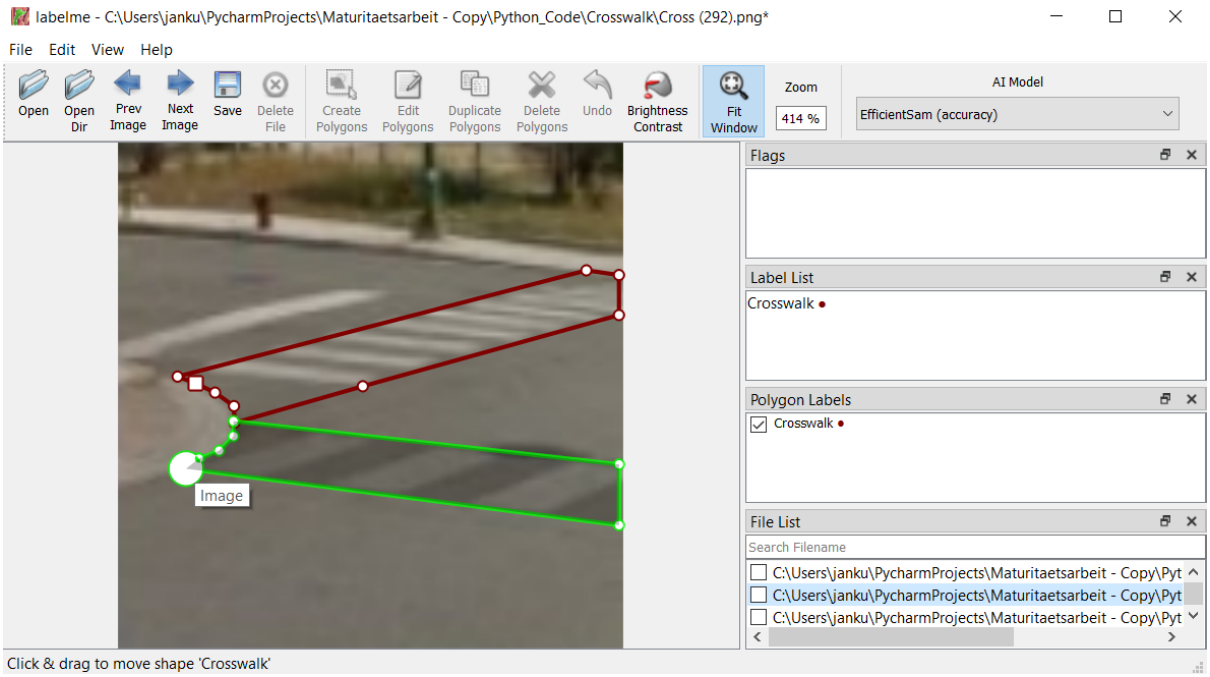



Abbildung 29: Der Prozess des *Labelings* mit Polygonen

### 3.1.3 Datenvorbereitung für spezifische Modelle

Nun muss noch beachtet werden, dass die Daten je nach Methode und verwendeter Bibliothek in einem anderen Format vorliegen müssen. Da das Pascal VOC Format jedoch stark standardisiert ist, gibt es meistens bereitstehende Skripts, um die Daten in das benötigte Format umzuwandeln. Dieses allfällige letzte Umwandeln der Daten ist also selten ein grosses Problem.

## 3.2 Finden der besten Methoden

Nun geht es darum, zu bestimmen, mit welcher der im Theorieteil erklärten Methoden vermutlich die besten Resultate erreicht werden.

### 3.2.1 Histogram of Oriented Gradients:

**Vorteile:**

- Das *HOG* ist durch die Arbeit in lokalen Zellen praktisch immun gegen photometrische Einflüsse wie die Belichtung oder Schattenwürfe im Bild (vgl. Wikipedia, 2023).
- *HOG*s sind für grosse Datensätze effizient skalierbar, da sie nur grundlegende Merkmale der Bilder analysieren.
- Die Interpretierbarkeit der Merkmale von *HOG*s ist simpel, da in den *HOG*s einfach die im Bild enthaltenen Kanten dargestellt sind (vgl. Jorge, 2023).

**Nachteile:**

- Die Histogramme verändern sich bei einer anderen Orientierung des Objektes komplett, weswegen Objekte mit unterschiedlichen Orientierungen nicht erkannt werden.
- In kontrastarmen Bilden sind die Kanten schlecht erkennbar, wodurch die Gradienten nicht stark sind. In diesen Bildern werden also auch die Resultate schlecht sein.

**Im Beispielfall:**
Der grösste Nachteil ist die Orientierung, welche beim Erkennen von Fussgängerstreifen in CAPTCHAs relevant ist. Zum Teil sind auch die Kontraste der weiter entfernten Fussgängerstreifen auf den Bildern sehr schlecht, was zu Schwierigkeiten führen kann.

Ansonsten stehen die Chancen meiner Einschätzung nach aber nicht so schlecht, da Fussgängerstreifen markante und regelmässige Kanten haben, welche von den Histogrammen erfasst werden können sollten.

### 3.2.2 Template Matching

**Vorteile:**

- Es braucht nur wenige Templates, wodurch die Vorarbeit, welche geleistet werden muss, sehr klein ist.
- Template Matching ist sehr vielfältig, denn es kann sowohl simple als auch komplizierte Strukturen erkennen.
- Solange das Template nicht zu gross ist, können Algorithmen Objekte in Echtzeit erkennen. Das Template Matching ist also verhältnismässig schnell.

**Nachteile:**

- Template Matching ist sehr sensibel gegenüber Variationen wie der Grösse oder der Orientierung der zu erkennenden Objekte. Werden zur Behebung dieses Problems mehrere Templates eingeführt, kann dies zu anderen Problemen führen. Teile des Hintergrunds könnten z.B. als Objekt erkannt werden. Zudem führt die Verwendung mehrerer Templates zu einer deutlichen Verlangsamung des Algorithmus.
- Bei grösseren Templates werden die auszuführenden Berechnungen deutlich umfangreicher und der Algorithmus somit langsamer.

**Im Beispielfall:**
Die Fussgängerstreifen in CAPTCHAs sind sehr unterschiedlich angeordnet, was die grösste Schwäche des Template Matchings trifft. Es müssten also Templates in verschiedenen Grössen und Orientierungen eingesetzt werden, was die Rechenzeit stark steigert. Zudem sind gewisse Fussgängerstreifen aufgrund des tiefen Kontrasts mithilfe von Templates kaum erkennbar.

Für die Erkennung von Fussgängerstreifen in CAPTCHAs mit Template Matching sind also die Nachteile deutlich grösser als die Vorteile.

Meine Erwartungen an das Template Matching sind also in diesem spezifischen Fall sehr gering.

### 3.2.3 Neuronale Netzwerke/Segmentierung

Da die Segmentierungsmodelle auf neuronalen Netzwerken basieren, gelten für beide sehr ähnliche Voraussetzungen. Die kleinen Unterschiede werden weiter unten kurz erläutert.

**Vorteile:**

- *NN*s sind selbstlernend, was bedeutet, dass sie eigenständig Wege zum Ziel entwickeln. Dabei stossen sie oft auf Lösungen, die für Menschen nicht offensichtlich wären.
- *NN*s haben eine effiziente Rauschfilterung der Daten. Sie können also die relevanten Informationen trotz Rauschen im Bild isolieren.
- *NN*s haben ein sehr breites Anwendungsspektrum, da sie ähnlich funktionieren wie ein menschliches Gehirn. Sobald sie trainiert sind, können sie sehr komplexe Aufgaben lösen (vgl. Musienko, 2022).

**Nachteile:**

- Die Hardware des Computers muss sehr gut sein, um *NN*s zu trainieren und Vorhersagen von ihnen zu erhalten, da die Aufgaben sehr rechenintensiv sind.
- *NN*s brauchen riesige Datenmengen (tausende Dateneinträge), um trainiert werden zu können. Der Aufwand, um an viele gute Daten zu kommen oder sogar selbst ein Datenset zu erstellen, kann riesig werden (vgl. Rawat, 2022).
- Die Person, welche ein *NN* trainiert, hat nur minimale Kontrolle über die Funktionsweise von neuronalen Netzwerken und somit auch auf die Genauigkeit des Netzwerks.

**Unterschied normaler *NN*s und der Segmentierung:**
Der Hauptunterschied liegt in der Form der Datenausgabe. Während die Vorschläge der normalen *NN*s in Form von *Bounding Boxes* sind, geben Segmentierungsmodelle die Vorschläge in der Form von Polygonen aus. Der Vorteil von Polygonen liegt darin, dass wirklich nur der Teil des Bildes, welcher wirklich das Objekt ist, als solches markiert wird. *Bounding Boxes* umgeben das ganze Objekt, wobei unumgehbar auch ein Teil des Hintergrunds mit eingeschlossen wird, was zu Ungenauigkeiten führen kann. Dafür ist die Aufgabe, Polygone zu erstellen etwas komplexer, weswegen möglicherweise auch wieder ungenauere Resultate entstehen können.

Eine Aussage darüber zu machen, welche Methode nun wirklich die bessere ist, ist also schwer. Meistens wählt man die Methode, deren Output die Form hat, die besser zur anschliessenden Anwendung passt.

**Im Beispielfall:**
Die Hardware ist in meinem Fall etwas limitierend, da mein Computer gewisse grosse Modelle nicht trainieren kann. Da aber die Auflösung meiner Trainingsbilder nicht gut ist, sind die grossen Modelle vermutlich überflüssig. Die limitierte Kontrolle könnte zu einem Problem werden, aber es ist schwer, darüber eine Aussage zu machen, da noch unklar ist, ob ich auf Probleme stossen werde. Falls aber ein Modell zu Problemen führen würde, könnte ich ein anderes testen. Dass neuronale Netzwerke sehr grosse Datenmengen brauchen, ist vermutlich das grösste Problem. Denn die knapp über 1000 zur Verfügung stehenden Bilder gelten beim Training von *NN*s nicht als grosse Datenmenge.

Die Rauschfilterung ist ein grosser Vorteil von *NN*s und spielt eine entscheidende Rolle beim Erkennen von Fussgängerstreifen in den Bildern. Denn die Bilder, welche in CAPTCHAs verwendet werden, sind extra so konzipiert, dass es schwer ist, die Fussgängerstreifen zu erkennen. Somit wird dieser Vorteil vermutlich zu sehr guten Ergebnissen führen.

## 3.3 Trainieren verschiedener Modelle

Nun geht es um die Umsetzung der Theorie in die Praxis, damit die Vermutungen, die gemacht wurden, überprüft werden können. Ich habe mich zudem dazu entschieden, das Template Matching nicht in die Realität umzusetzen, da die Nachteile die Vorteilen deutlich überwiegen und die anderen Methoden vielversprechender klingen. Ich konzentriere mich lieber auf eine gute Optimierung der anderen Methoden, anstatt noch mehr Methoden zu implementieren.

### 3.3.1 Histogram of Oriented Gradients

**Vorbereitungen:**
Wie in der Theorie erklärt, braucht es sowohl Regionen aus Bildern mit Fussgängerstreifen, als auch Regionen ohne Fussgängerstreifen, um eine *Support Vector Machine* trainieren zu können. Auch hier gilt, je mehr Regionen für die beiden Fälle zum Training vorhanden sind, desto besser wird die Leistung des Modells.

Dieser Prozess lässt sich Dank des beschrifteten Datensatzes automatisieren.

- **Regionen mit Fussgängerstreifen (Crosswalk-Regions):**
  Da im Datensatz alle Fussgängerstreifen angegeben sind, müssen diese lediglich noch durch ein Programm extrahiert und normalisiert werden. Normalisieren heisst hierbei, dass die Bilder alle zu Quadraten geschnitten werden müssen, da später bei der Erkennung auch immer quadratische Regionen untersucht werden.

- **Regionen ohne Fussgängerstreifen (Non-Crosswalk-Regions):**
  Hierzu habe ich ein Programm geschrieben, welches in jedem Bild durch das Ansetzen an zufälligen Orten versucht, eine 64x64 Pixel grosse Region ausfindig zu machen, die ausserhalb einer Crosswalk Region liegt. Für die zufällig ausgewählte Stelle wird also überprüft, ob sie sich mit einer Crosswalk-Region überschneidet. Ist dies nicht der Fall, wird diese Region als Non-Crosswalk-Region abgespeichert. Falls es aber doch eine Überschneidung gibt, wird es für eine andere zufällige Stelle erneut versucht. Scheitert das Programm 100 Mal daran, eine 64x64 grosse Region ohne Fussgängerstreifen zu finden, so versucht es dasselbe für eine 32x32 grosse Region, wovon es deutlich mehr gibt. Das Programm extrahiert dabei maximal 3 Regionen pro Bild und lässt gegebenenfalls ein Bild aus, wenn es selbst bei den 32x32 Pixel grossen Regionen keine passende Stelle findet.

**Trainieren der *SVM*:**
Bevor die *SVM* trainiert werden kann, müssen die *HOG* Features extrahiert werden. Dafür müssen die in der Theorie erwähnten Parameter festgelegt werden: *Kernel*, *Block Size*, *Block Stride* und die *Cell Size*. Es kommen noch ein paar weitere nicht erwähnte Parameter dazu.

Zudem habe ich verschiedene Kontrast- und Helligkeitswerte bei den Ausgangsbildern ausprobiert. Die Kontrast- und Helligkeitswerte können die Fussgängerstreifen in gewissen Bildern

stark hervorheben (Abb. 30: Serie 1), in anderen lassen sie die Fussgängerstreifen aber auch komplett verschwinden (Abb. 30: Serie 2), sodass er nicht mehr erkannt werden kann. Ich wollte also herausfinden, ob die Resultate mit diesen Werten noch etwas verbessert werden können.

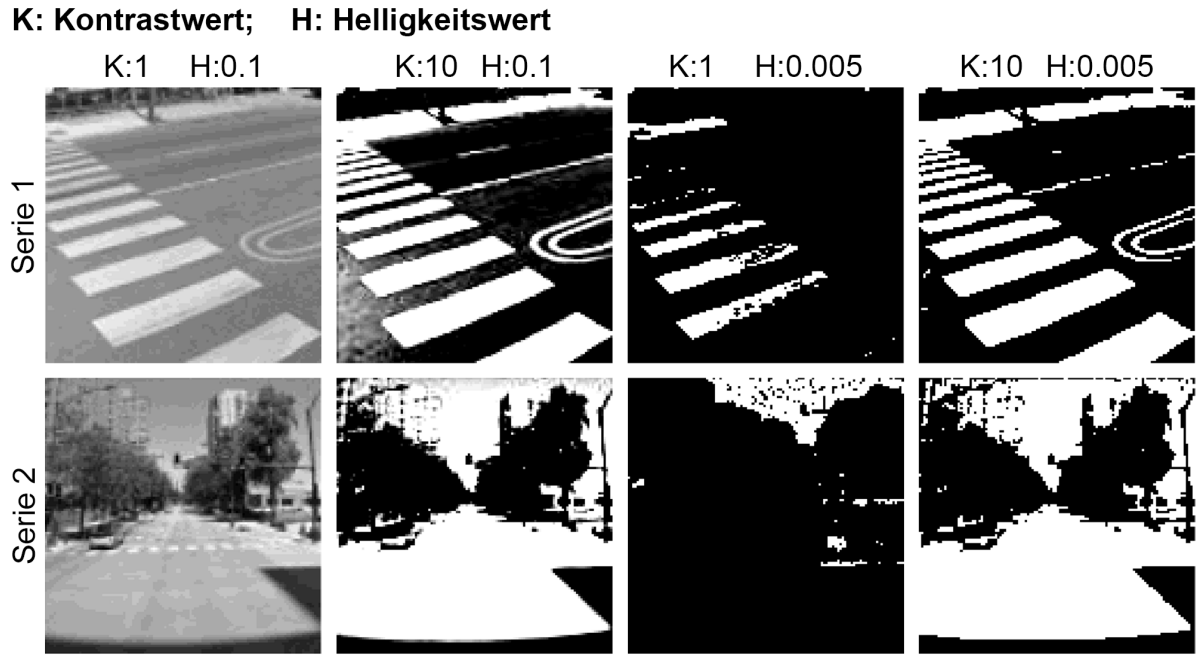



Abbildung 30: Veranschaulichung von Kontrast und Helligkeitsveränderungen

Um die besten Parameter zu finden, habe ich ein Skript geschrieben, welches all diese Parameter miteinander kombiniert und die Resultate in einer Excel Tabelle speichert. Gesamthaft wurden 3072 Kombinationen der Parameter getestet, um die besten zu finden. Jede Kombination resultiert in einem *HOG* Modell. In Tabelle 3 habe ich alle getesteten Parameterwerte angegeben.

| **Parameter** | **getestete Werte** |
|---|---|
| Kernel | „rbf“, „poly“ |
| Block Size | 8, 16 |
| Block Stride | 4, 8 |
| Cell Size | 4, 8 |
| Kontrastwerte | 1, 2, 3, 5, 7, 10 |
| Helligkeitswerte | 0.1, 0.05, 0.01, 0.005 |
| Thresholdwerte | -1, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1 |

Tabelle 3: Tabelle mit verwendeten Parametern

### 3.3.2 Neuronales Netzwerk

**Vorbereitungen:**
Es gibt mehrere Bibliotheken in Python, um mit neuronalen Netzwerken zu arbeiten. Ich entschied mich für eine Bibliothek namens TensorFlow, da diese sehr bekannt ist und ich schon etwas vertraut damit war.

Um neuronale Netzwerke zu trainieren, braucht man Trainingsbilder und Validierungsbilder. Das Netzwerk braucht die Validierungsbilder, damit es berechnen kann, wie gut seine Resultate schon sind. Zudem werden noch Testbilder benötigt, damit das Endmodell mit Bildern, welche das Modell noch nie gesehen hat, getestet werden kann. Dafür wurden alle Bilder aus dem Datensatz zufällig gemischt und anschliessend in Gruppen aufgeteilt. 80% der Bilder wurden dem Trainingsset, 10% dem Validierungsset und 10% dem Testset zugeteilt.

TensorFlow braucht die Daten, also im Fall der Objekterkennung die Bilder und die korrespondierenden Beschriftungen in einem spezifischen Format, einer *.tfrecord* Datei. Daher musste ich zuerst die Daten vom vorher erklärten Pascal VOC Format umwandeln. Dies war erstaunlicherweise mühsamer als erwartet, denn ich konnte kein Skript finden, welches die Daten vom Pascal VOC Format direkt in eine TensorFlow *.tfrecord* Datei umwandelt. So machte ich einen Zwischenschritt über ein CSV File. CSV Files sind tabellenartige Dokumente, in welchen die Daten wie in Abb. 31 angeordnet sind. Die Daten in dieses Format umzuwandeln, war mit einem kleinen Skript kein Problem. Um von den CSV Files zu den TensorFlow *.tfrecord* Files zu gelangen, verwendete ich eine leicht abgeänderte Version eines Skripts von EdjeElectronics, 2020, welches auf GitHub zur Verfügung gestellt wird. Somit hatte ich drei TensorFlow *.tfrecord* Dateien, eine Trainings- eine Validierungs- und eine Testdatei.

```
filename          width heigh  class      xmin ymin xmax ymax
Cross (1015).png  120   120    Crosswalk  13   54   115  76
Cross (1015).png  120   120    Crosswalk  39   38   77   43
Cross (1015).png  120   120    Crosswalk  2    41   31   52
Cross (1015).png  120   120    Crosswalk  94   39   114  44
Cross (1022).png  120   120    Crosswalk  6    70   57   109
Cross (1022).png  120   120    Crosswalk  43   79   89   120
Cross (1022).png  120   120    Crosswalk  80   87   120  120
Cross (1046).png  120   120    Crosswalk  1    72   31   88
Cross (1046).png  120   120    Crosswalk  29   70   59   82
Cross (1046).png  120   120    Crosswalk  56   67   94   79
Cross (1046).png  120   120    Crosswalk  90   62   120  74
```

Abbildung 31: Aufbau eines CSV Files

**Trainieren der neuronalen Netzwerke:**
In TensorFlow kann man neuronale Netzwerke komplett von Grund auf aufbauen oder man kann vorgefertigte Modelle importieren. Ich entschied mich dazu, bereits vorhandene Modelle zu importieren, da diese durch ihre Komplexität deutlich besser sind als das, was ich in kurzer Zeit selbst hätte aufbauen können. Ich wählte einige Modelle aus einer offiziellen TensorFlow Bibliothek namens „TensorFlow 2 Detection Model Zoo“ aus. In dieser Bibliothek gibt es ein paar komplett verschiedene Modelle, wobei es von einem Typ jeweils kleinere und grössere Versionen gibt. Grössere Versionen behalten dabei die Struktur des Typs, haben aber mehr Ebenen und Neuronen.

| Model name | Speed (ms) | COCO mAP | Outputs |
|---|---|---|---|
| CenterNet Resnet50 V2 512x512 | 27 | 29.5 | Boxes |
| CenterNet Resnet50 V2 Keypoints 512x512 | 30 | 27.6/48.2 | Boxes/Keypoints |
| CenterNet MobileNetV2 FPN 512x512 | 6 | 23.4 | Boxes |
| CenterNet MobileNetV2 FPN Keypoints 512x512 | 6 | 41.7 | Keypoints |
| EfficientDet D0 512x512 | 39 | 33.6 | Boxes |
| EfficientDet D1 640x640 | 54 | 38.4 | Boxes |
| EfficientDet D2 768x768 | 67 | 41.8 | Boxes |
| EfficientDet D3 896x896 | 95 | 45.4 | Boxes |

Abbildung 32: Auszug der Modelle aus *TensorFlow 2 Detection Model Zoo*

Welches dieser Modelle nun das beste ist, kommt stark auf den Anwendungsfall an. Die Modelle unterscheiden sich strukturell stark voneinander, es ist jedoch schwer, aus den angegebenen Informationen vorherzusagen, welches Modell in meinem Fall das beste wäre. So entschied ich mich dazu, dass ich möglichst viele Modelle ausprobieren werde. Ich beschränkte mich grösstenteils auf die kleineren, schnelleren Modelle, da die Bildauflösung meiner Bilder extrem schlecht ist und ich mit meinem Computer gewisse Modelle nicht trainieren konnte, da sie zu gross waren.

Schliesslich testete ich die folgenden Modelle. Die Benennung besteht jeweils aus dem Namen der zugrunde liegenden Architektur, der Version und der Auflösung der Input Bilder des Modells.

- EfficientDet D0 512x512
- CenterNet ResNet50 V2 512x512
- SSD MobileNet v2 320x320
- Faster R-CNN ResNet152 V1 640x640
- Faster R-CNN Inception ResNet V2 640x640

Damit die heruntergeladenen Modelle wissen, um welche Objekte es sich handelt, muss man eine sog. „Label Map“ erstellen. Das ist eine einfache Textdatei, die so aussieht, wie hier rechts. Da meine Modelle Fussgängerstreifen erkennen müssen, ist der *name* der Objekte „Crosswalk“.

```
item {
    id: 1
    name: "Crosswalk"
}
```

Die notwendigen Installationen führte ich nach den Anweisungen des Artikels von Morgunov, 2023 aus. TensorFlow stellt für die vorgefertigten Modelle ein Skript zur Verfügung, mit welchem man sie trainieren kann. Mann muss nur noch in einem Konfigurationsskript die folgenden Parameter anpassen:

- **num_classes:** Die Anzahl zu erkennenden Objekte, hier also 1, da ich nur Fussgängerstreifen erkennen will.

- **num_steps:** Die gewünschte Anzahl Trainingsschritte (Die getesteten Modell wurden alle bis zu 100'000 Schritten trainiert.)
- **warmup_steps:** Die gewünschte Anzahl Aufwärmschritte (Alle Modelle wurden mit 100 und 10'000 Aufwärmschritten getestet.)
- **fine_tune_checkpoint:** Der Pfad zur vortrainierten *ckpt-0* Datei, die zusammen mit dem Modell heruntergeladen wurde.
- **label_map_path:** Der Pfad zur Label Map Datei.
- **input_path:** Die Pfade zu den Validierungs- und Trainings *.tfrecord* Dateien.

Anschliessend musste ich beim Ausführen des Skripts noch den Pfad zu dieser Konfigurationsdatei sowie den Pfad, an welchen das Modell gespeichert werden sollte, angeben. Darüber hinaus sollte jeweils nach 100 Schritten ein Zwischenschritt gespeichert werden, damit später die Genauigkeit der Modelle mit der Anzahl Schritte verglichen werden konnte.

**Vorhersagen mit trainierten Modellen:**
Zuerst musste ich die gespeicherten Zwischenschritte in richtige Modelle umwandeln. TensorFlow stellt auch dafür ein Skript zur Verfügung. Da ich sehr viele Zwischenschritte zu tatsächlichen Modellen exportieren musste, schrieb ich ein Skript, welches dies automatisierte.

Die Modelle geben für jedes getestete Bild eine definierte Anzahl an *Bounding Boxes* mit einer dazugehörigen Wahrscheinlichkeit aus. Je höher die Wahrscheinlichkeit einer *Bounding Box*, desto eher handelt es sich laut dem Modell bei dieser *Bounding Box* tatsächlich um einen Fussgängerstreifen. Damit die unwahrscheinlichen *Bounding Boxes* herausgefiltert werden, definiert man einen Thresholdwert. Der Thresholdwert gibt die minimale Wahrscheinlichkeit an, die vorausgesetzt wird, damit eine *Bounding Box* wirklich als Ergebnis zählt.

Ich fügte meinem Skript dann einen Code-Block hinzu, welcher von den Modellen Vorhersagen für die Testbilder machen liess. Ich speicherte für jedes Modell die Resultate für 12 Thresholdwerte (0, 0.1, 0.15, 0.175, 0.2, 0.225, 0.25, 0.275, 0.3, 0.4, 0.45, 0.5), also jeweils die *Bounding Boxes*, die eine Wahrscheinlichkeit hatten, die grösser als der Thresholdwert war. Die *Bounding Boxes* wurden wieder im Pascal VOC Format abgespeichert.

### 3.3.3 Segmentierung mithilfe eines neuronalen Netzwerks

**Vorbereitungen:**
Da die getesteten Segmentierungsmodelle alle auf neuronalen Netzwerken basieren, stehen in etwa dieselben Bibliotheken zur Verfügung wie schon bei den neuronalen Netzwerken. So entschied ich mich auch hier für die *TensorFlow* Bibliothek. Es ist dementsprechend einiges gleich wie im Kapitel 3.3.2, weshalb ich hier nicht alles wiederholen werde. Die Dinge, welche aber anders als bei den normalen neuronalen Netzwerken waren, werden in diesem Kapitel erläutert.

Da die Daten für die Segmentierung grundlegend von den Begrenzungsrahmen abweichen, musste ich noch einmal neue *.tfrecord* Dateien erstellen. Die Segmentierungsdaten lagen nach der manuellen Beschriftung im *labelme* Format vor, in welchem die Koordinaten der Ecken der Polygons gespeichert sind. Aus diesen Daten erstellte ich zuerst „Binary Masks" (vgl. Abb. 33). Das sind einfache Bilddateien, in welchen die Pixel der Fussgängerstreifen den

Wert 1 und die Pixel des Hintergrunds den Wert 0 bekommen. Danach erstellte ich mit einem Python Skript aus den Originalbildern und den *Binary Masks* die benötigten *.tfrecord* Dateien.

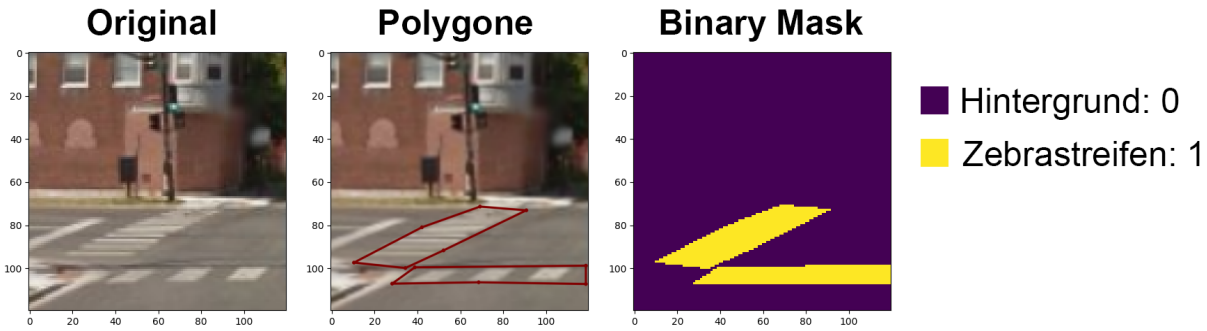



Abbildung 33: Vergleich zwischen *Originalbild*, *Polygonbeschriftung* und *Binary Mask*

**Trainieren von Segmentierungsmodellen:**
Hierbei folgte ich einem Tutorial von TensorFlow, o.J. und ergänzte teilweise etwas. Auch hier gibt es wieder viele verschiedene Modelle zur Auswahl. Getestet wurden die folgenden:

- DeepLabV3 MobileNetV2
- DeepLabV3Plus SegNet

Ähnlich wie bei den einfachen neuronalen Netzwerken musste ich auch hier die Modelle wieder konfigurieren. Gesetzt werden mussten Parameter wie die Anzahl Aufwärmschritte und Trainingsschritte, der Pfad zu den *.tfrecord* Dateien und weitere. Während des Trainings wurden auch hier wieder Zwischenschritte gespeichert.

**Vorhersagen mit trainierten Modellen:**
Die Zwischenschritte mussten auch hier wieder in richtige Modelle umgewandelt werden. Dazu schrieb ich ein Skript, welches die Zwischenschritte automatisch zu Modellen exportiert und anschliessend für jedes Modell eine Vorhersage für jedes Bild aus dem Testdatensatz erstellt. Die Modelle geben ihr Resultat in der Form einer *Binary Mask* zurück. Schliesslich hatte ich für jedes Modell und jedes Testbild eine *Binary Mask*.

## 3.4 Analyse

In diesem Kapitel erkläre ich, wie ich bei der Datenanalyse vorgegangen bin und wie ich meine Daten dafür vorbereitet habe.

### 3.4.1 Definition von Vergleichsmetriken

Die Modelle haben jeweils individuelle Auswertungsmetriken, welche ihre Genauigkeit angeben. Weil man diese aber nicht untereinander vergleichen kann, überlegte ich mir eigene Metriken. Da mein finales Ziel war, mit den Modellen CAPTCHAs zu lösen, bezog ich mich bei meinen eigenen Metriken direkt auf das Lösen von CAPTCHAs.

**Genauigkeit im Bild (*GiB*):**
Die Bilder sind in ein 4x4 Raster unterteilt. Ein einzelnes Kästchen kann entweder richtig oder falsch angewählt worden sein. Die Genauigkeit innerhalb eines Bilds entspricht dabei

dem prozentualen Anteil an richtig angewählten Kästchen verglichen mit der Gesamtanzahl an Kästchen.

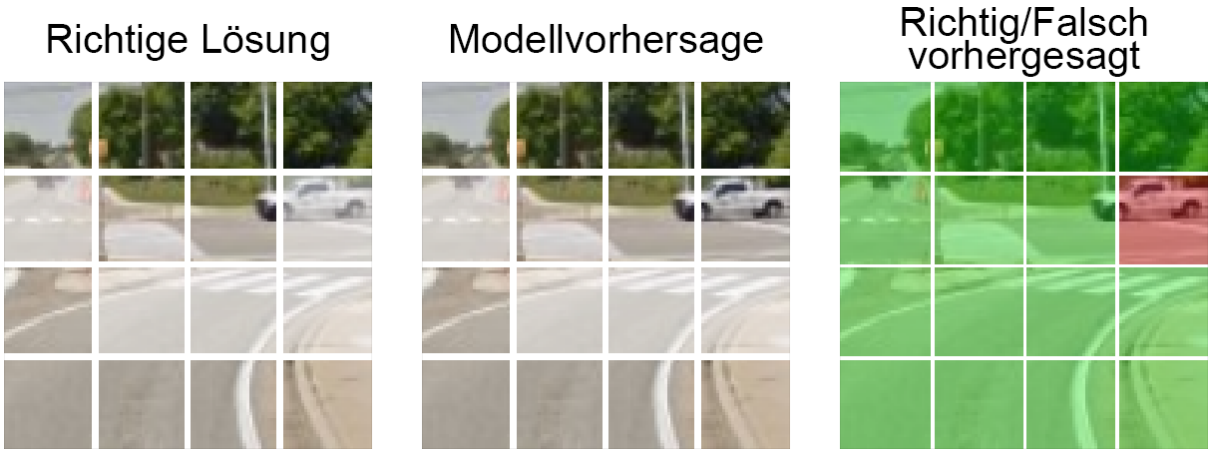



Abbildung 34: Genauigkeit innerhalb eines Bilds

Im Beispiel in Abb. 34 hat das Modell 15/16 Kästchen korrekt erkannt. Somit ist die *GiB* dieses Bilds $\frac{15}{16} \cdot 100 = 93.75\%$.

Diese Genauigkeit innerhalb des Bilds wird nun für jedes Bild des Testsatzes berechnet. Anschliessend wird der Durchschnitt aller Bilder genommen.

**Perfekte Genauigkeit (*pG*):**
Die *perfekte Genauigkeit* sagt aus, wie viele Bilder das Modell prozentual komplett richtig erkannt hat. Berechnet wird also:

$$\frac{Anzahl\ komplett\ richtiger\ Bilder}{Gesamtanzahl\ Bilder\ im\ Testdatensatz} \cdot 100 = Perfekte\ Genauigkeit\ [in\ \%]$$

Die *perfekte Genauigkeit* gibt also den prozentualen Anteil an, bei dem das Modell den CAPTCHA Test bestanden hätte.

### 3.4.2 Vorgehen bei der Analyse

Um die Genauigkeit der Modelle in Form der *pG* und der *GiB* bestimmen zu können, müssen die Vorhersagen in Form von *Bounding Boxes* und *Binary Masks* in die CAPTCHA Form umgewandelt werden. Mit CAPTCHA Form ist ein 4x4 Gitter gemeint, wobei jedes Kästchen entweder angewählt sein kann oder nicht. Für die Umwandlung schrieb ich zwei Skripts (eines für die *Bounding Boxes* und eines für die *Binary Masks*). Ein Kästchen des CAPTCHAs wurde angewählt, wenn eine *Bounding Box* bzw. ein Crosswalk Pixel der *Binary Mask* im Kästchen war (vgl. Abb. 35).

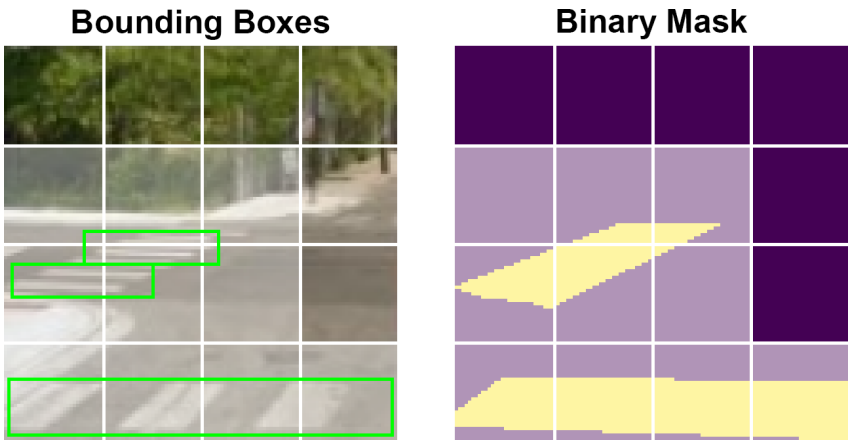



Abbildung 35: Umwandlung ins CAPTCHA Format

Ich begann damit, die verschiedenen Methoden einzeln auszuwerten. Dabei analysierte ich die verschiedenen Modelle und den Einfluss der getesteten Parameter.

**Histogram of Oriented Gradients:**
Bei der Analyse der *HOG* Modelle analysierte ich den Einfluss der getesteten Parameter auf die *pG* und die *GiB*. Ich habe zu den Daten jeweils Graphen erstellt, in welchen die ausgewerteten Daten visualisiert werden. Die analysierten Parameter sind:

- Thresholdwerte
- *HOG* Parameter
  - Block Size
  - Block Stride
  - Cell Size
- Vorbearbeitung der Bilder
  - Kontrastwerte
  - Helligkeitswerte

Zudem beschrieb ich bei jedem Parameter den Einfluss auf die Genauigkeiten und gab allfällige Erklärungen dafür. Anschliessend bezog ich mich noch spezifisch auf die besten *HOG* Modelle und überprüfte, ob die vorher bestimmten Tendenzen der Parameter auch auf die besten Modelle übertragbar sind.

**Neuronale Netzwerke:**
Bei den neuronalen Netzwerken wurde zuerst analysiert, welchen Einfluss die Thresholdwerte und die Anzahl Trainings- und Aufwärmschritte auf die *pG* und die *GiB* hatten. Zudem analysierte ich noch, ob die Vorhersagezeit eine Korrelation zu den Genauigkeiten hat. Mit der Vorhersagezeit ist gemeint, wie lange ein Modell durchschnittlich für eine Vorhersage gebraucht hat.

Danach verglich ich die getesteten Modelle miteinander, um herauszufinden, welches der Modelle das beste ist. Da ich das Limit testen wollte, liess ich das beste Modell noch bis zu

200'000 Trainingsschritten weiter trainieren und analysierte die weitere Entwicklung.

Auch bei den neuronalen Netzwerken arbeitete ich mit Graphen und Erklärungen.

**Segmentierung:**
Hier ging ich praktisch gleich vor wie bei den normalen neuronalen Netzwerken. Da es bei der semantischen Segmentierung aber kein Threshold gibt, konnte ich dieses natürlich nicht analysieren. Ich betrachtete auch zuerst über alle Modelle hinweg gesehen die Entwicklung der *pG* und der *GiB* in Bezug auf die Anzahl Trainings- und Aufwärmschritte.

Anschliessend verglich ich auch hier wieder die verschiedenen getesteten Modelle miteinander.

**Quervergleich zwischen verschiedener Methoden:**
In diesem Teil verglich ich das jeweils beste Modell der verschiedenen Methoden miteinander. So konnte ich herausfinden, ob das *HOG* die neuronalen Netzwerke oder die Segmentierung die besten Resultate liefert. Dazu verglich ich weitere Faktoren wie die Trainingszeit und die Vorhersagezeit der einzelnen Modelle miteinander.

## 3.5 Vergleich unter Menschen

Neben den Resultaten der Computermodelle, interessierte ich mich auch dafür, wie gleich überhaupt Menschen CAPTCHAs lösen. Denn teilweise ist man sich auch als Mensch unsicher, ob man das Kästchen nun noch anklicken soll, besonders wenn der Fussgängerstreifen gerade noch so knapp in ein neues Kästchen überragt.

Damit ich Menschenresultate miteinander vergleichen konnte, musste ich zuerst ein paar Testpersonen meine Test-CAPTCHAs lösen lassen. Ich gab den Menschen die gleichen 124 Test-CAPTCHAs, die auch meine Modelle lösen mussten. Ich schrieb also ein kleines Programm, in welchem die Testpersonen die CAPTCHAs lösen mussten. Sobald die Person auf weiter klickte, wurde das Resultat gespeichert und man konnte nicht mehr zurück gehen. Das machte ich so, da man in der echten Welt beim Lösen von CAPTCHAs nach dem Absenden auch nicht mehr zurück kann. Neben den Resultaten wurde zudem noch die Zeit, welche die Personen benötigten, um die CAPTCHAs zu lösen, gemessen. Ich habe 8 Personen den Test machen lassen, was mit meinen eigenen Resultaten zu 9 unterschiedlichen Personen führt.

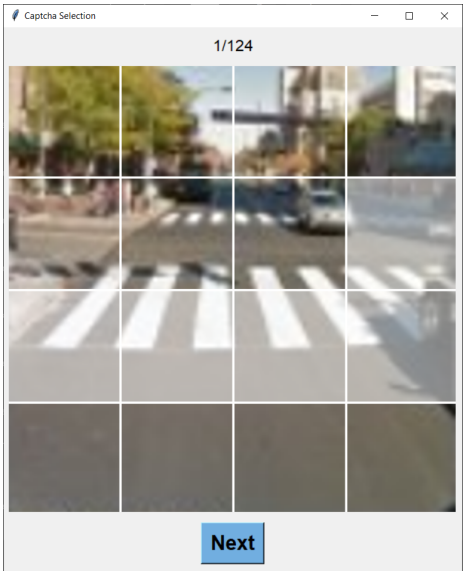


Abbildung 36: CAPTCHA Testprogramm

Da man immer nur zwei Personen miteinander vergleichen kann, verglich ich jede Person mit jeder anderen einzeln. Bei jedem Personenvergleich überprüfte ich für jedes Testbild, ob das CAPTCHA identisch gelöst wurde. Schliesslich berechnete ich die Prozentzahl identisch gelöster CAPTCHAs. Diese perfekte Übereinstimmungszahl ist vergleichbar mit der *perfekten*

*Genauigkeit*, die ich für die Modelle verwendete.

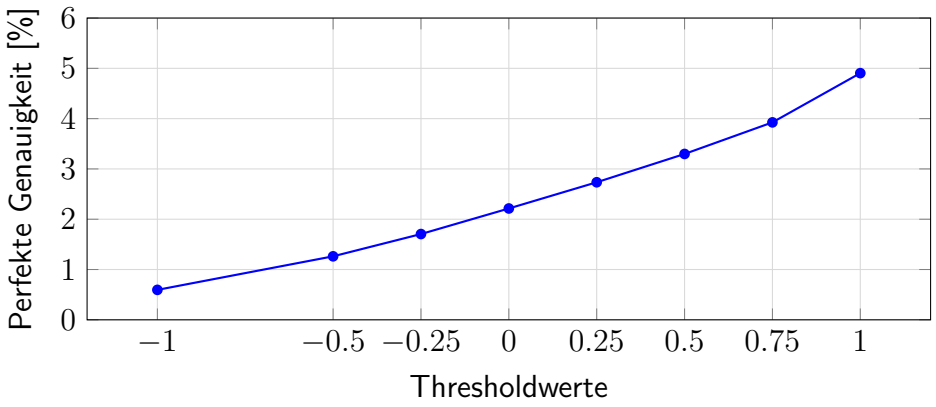
Bei jedem Personenvergleich berechnete ich zudem bei jedem Testbild die Übereinstimmung innerhalb des Bildes, was gleich funktioniert, wie die Berechnung der *Genauigkeit im Bild*. Dann berechnete ich auch hier von allen Vergleichen den Durchschnitt, was mir einen Wert gab, der vergleichbar mit der *GiB* ist, welche ich für die Computermodelle verwendete.

# 4 Darstellung der Ergebnisse

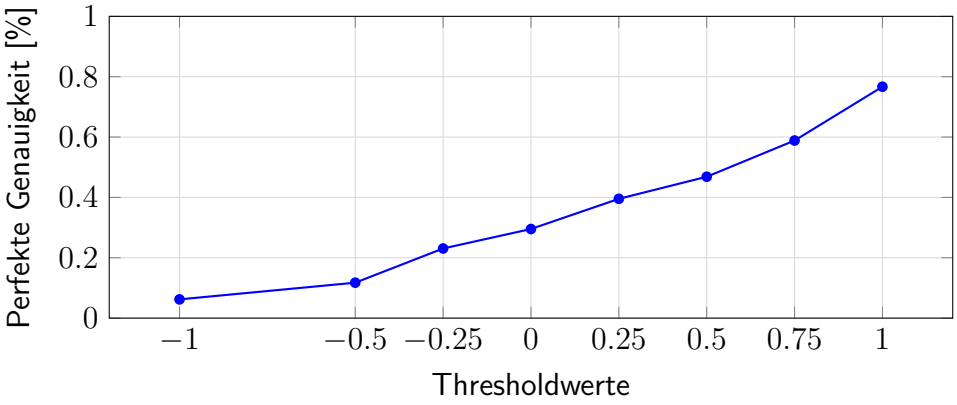
## 4.1 Histogram of Oriented Gradients

### 4.1.1 Thresholdanalyse

Im folgenden Diagramm ist die durchschnittliche *perfekte Genauigkeit* der 3072 verschiedenen Kombinationen für den jeweiligen Thresholdwert dargestellt.




Hierbei fiel mir auf, dass die Modelle mit Thresholdwert 1, obwohl sie die höchste *pG* haben, grössenteils überhaupt keine Fussgängerstreifen erkannten. Die höhere *perfekte Genauigkeit* kam also v.a. davon, dass es die 7 von 124 Bilder ohne Fussgängerstreifen aus dem Testsatz richtig erkannte. Ich wollte also schauen wie es aussieht, wenn ich die Bilder ohne Fussgängerstreifen aus dem Testsatz entferne. Dies ist im Graph unten abgebildet.



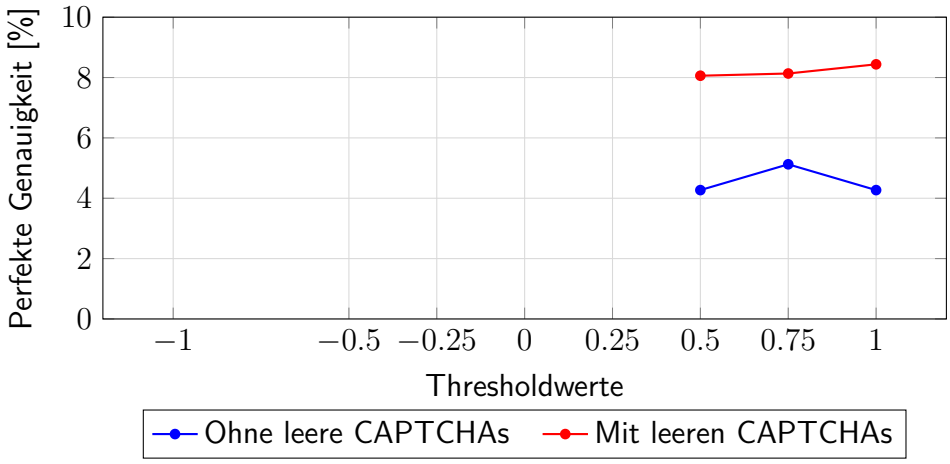

Ein grösseres Threshold führt zwar immer noch zu einer höheren *perfekten Genauigkeit*, man kann allerdings klar erkennen, dass diese generell deutlich kleiner ist.

Da ich mich besonders für die besten Modelle interessierte, betrachtete ich Modelle mit

- *perfekter Genauigkeit* $> 8\%$ für Resultate mit leeren CAPTCHAs
- *perfekter Genauigkeit* $> 4\%$ für Resultate ohne leere CAPTCHAs

separat. Mit leeren CAPTCHAs meine ich diejenigen, die keine Fussgängerstreifen im Bild haben. Wenn ich im Folgenden von den besten Modellen spreche, dann sind immer diejenigen gemeint, die diese Bedingungen erfüllen.

Graph für die besten Modelle




Da für Thresholdwerte kleiner als 0.5 kein Modell die entsprechende Mindestgenauigkeit erreichte, fehlen die Einträge dazu.

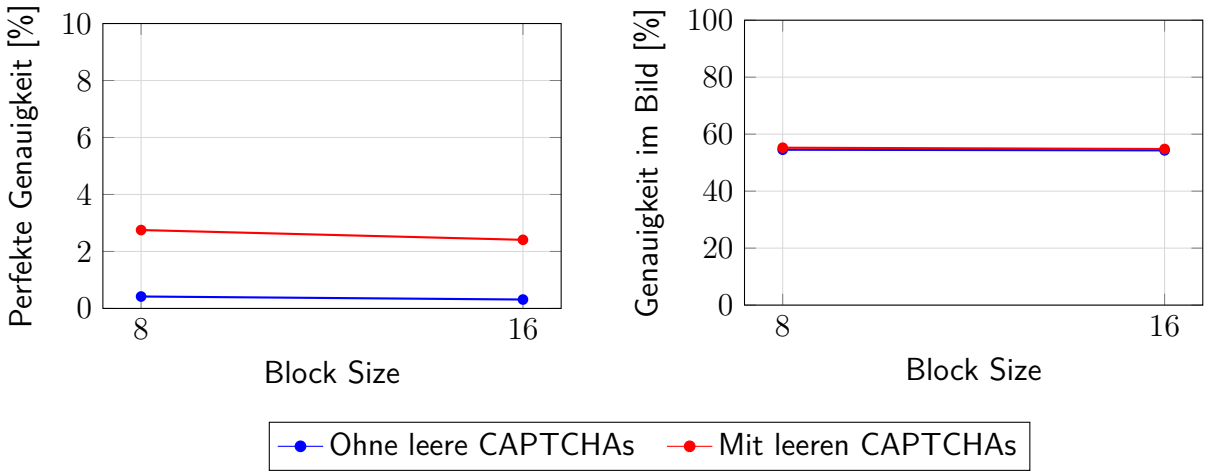
Auch bei den besten Modellen ist klar erkennbar, dass die leeren CAPTCHAs einen grossen Einfluss auf die *pG* haben. Denn noch immer ist der höchste getestete Thresholdwert der beste, da die leeren CAPTCHAs richtig erkannt werden. Betrachtet man die *pG* ohne leere CAPTCHAs, zeigt sich, dass der beste Thresholdwert bei 0.75 liegt. Die Modelle mit diesem Wert erkennen Fussgängerstreifen am zuverlässigsten. Ein Thresholdwert von 1 erscheint nur deshalb besser, weil er häufig verhindert, dass überhaupt etwas als Fussgängerstreifen erkannt wird, was die *pG* bei Anwesenheit leerer CAPTCHAs erhöht.

### 4.1.2 Analyse der *HOG* Parameter

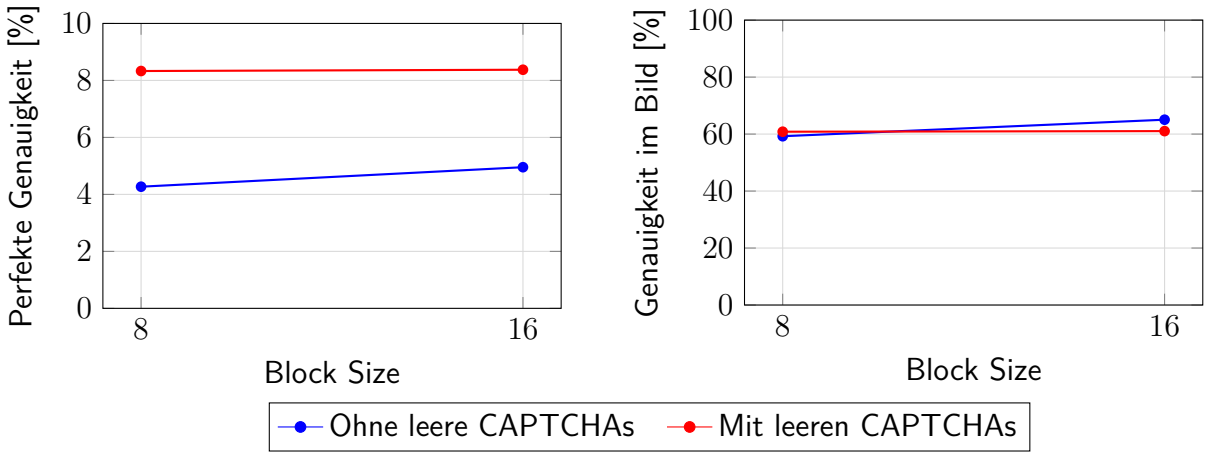
Auch in diesem Kapitel werde ich wegen des starken Einflusses der leeren CAPTCHAs, zwischen der Auswertung mit und ohne leere CAPTCHAs unterscheiden, damit die Resultate nicht verfälscht sind.

**Block Size:**

Graphen für alle Modelle



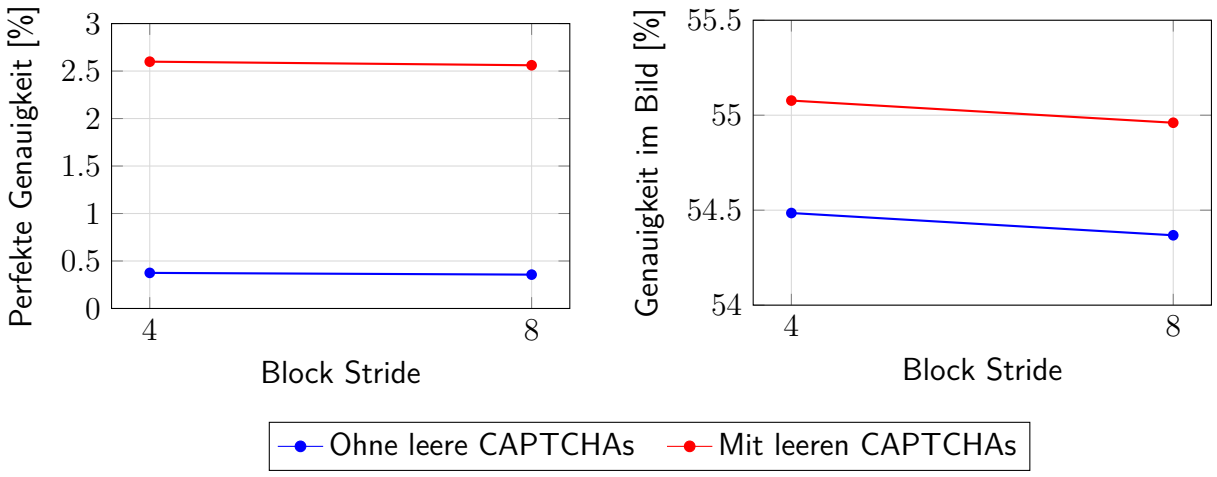

Graphen für die besten Modelle



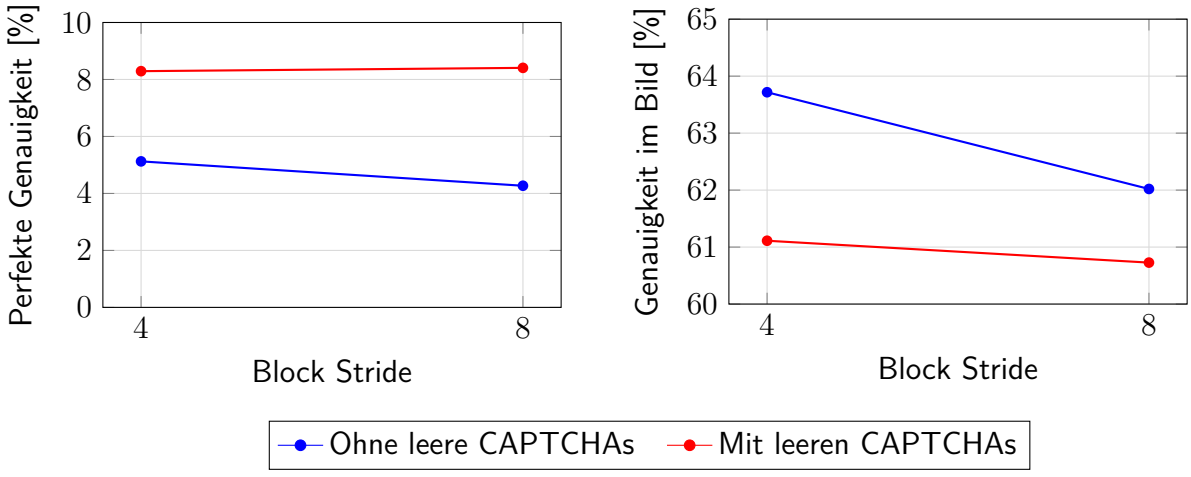

Über alle Modelle gesehen war *Block Size* 8 besser. Wenn man allerdings nur die besten Modelle anschaut, dann war *Block Size* 16 besser. Bei den besten Modellen wurden die Bilder also in grösseren Blöcken normalisiert. Besonders gross war der Einfluss der *Block Size* allerdings nicht, weshalb der Unterschied aller Modelle und der besten Modelle auch Zufall sein könnte.

**Block Stride:**

Graphen für alle Modelle



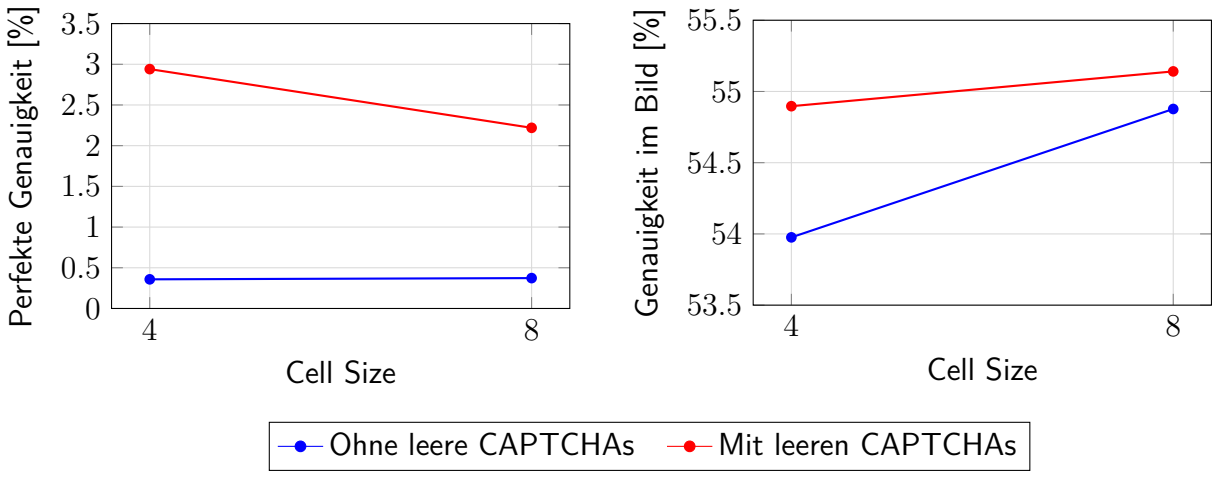

Graphen für die besten Modelle



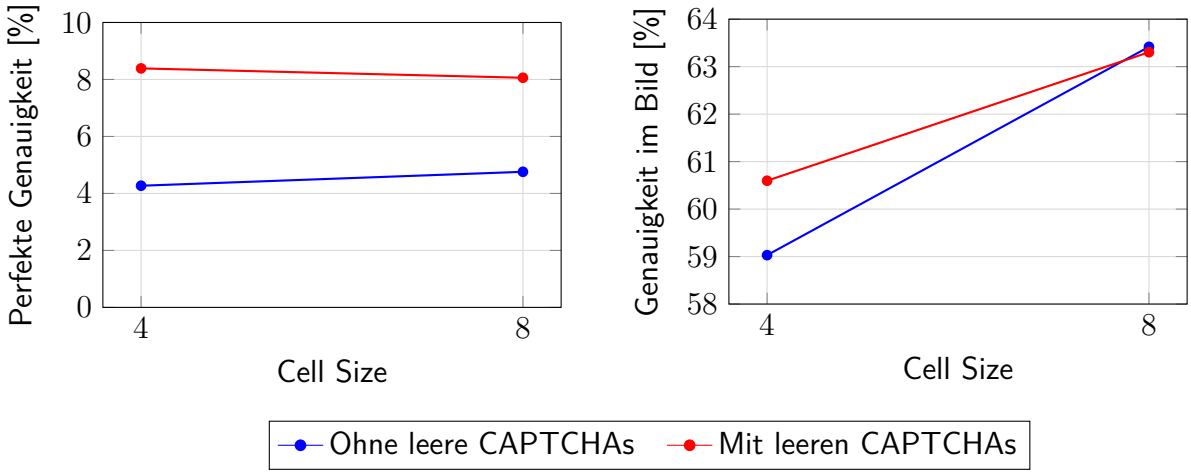

Ein *Block Stride* von 4 führte in praktisch jedem Fall zu einer höheren Genauigkeit. Mit einem kleineren *Block Stride* werden die Bilder mehr normalisiert, was hier zu mehr erkannten Fussgängerstreifen führte. So kann man sich auch erklären, dass der Graph mit leeren CAPTCHAs bei den besten Modellen bei *Block Stride* 4 etwas schlechter ist. Denn diese Modelle setzen ja v.a. darauf, dass sie überhaupt keine Fussgängerstreifen erkennen. Werden durchschnittlich aber mehr Fussgängerstreifen in den Bildern erkannt, so ist es auch wahrscheinlicher, dass in den leeren CAPTCHAs ein Fussgängerstreifen erkannt wird und diese somit falsch sind.

**Cell Size:**

Graphen für alle Modelle




Graphen für die besten Modelle



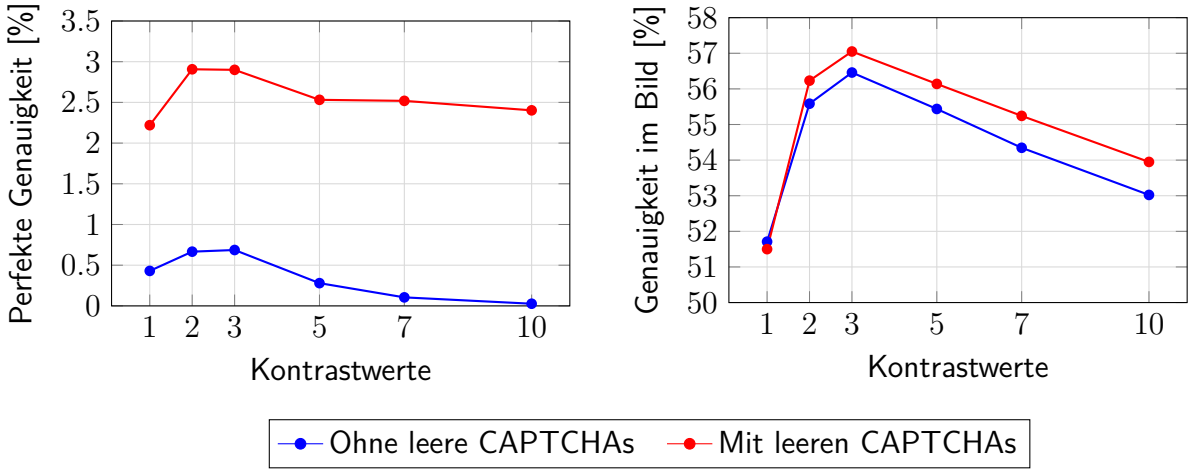

Die *GiB* ist mit *Cell Size* 8 immer etwas höher. Die *pG* dagegen verhält sich wieder etwas unterschiedlich. Generell scheint es aber so, dass bei einer *Cell Size* von 8 eher mehr Fussgängerstreifen erkannt werden, weshalb die *pG*, wenn leere CAPTCHAs mit einbezogen werden, eher tiefer ist. Bei den besten Modellen (ohne leere CAPTCHAs) ist *Cell Size* 8 etwas besser.

### 4.1.3 Vorbearbeitungsparameter der Bilder

In diesem Unterkapitel zeige ich den Effekt der Vorbearbeitung der Bilder auf die Genauigkeit auf. Spezifisch sind hier die getesteten Kontrast- und Helligkeitswerte gemeint.

**Kontrast:**

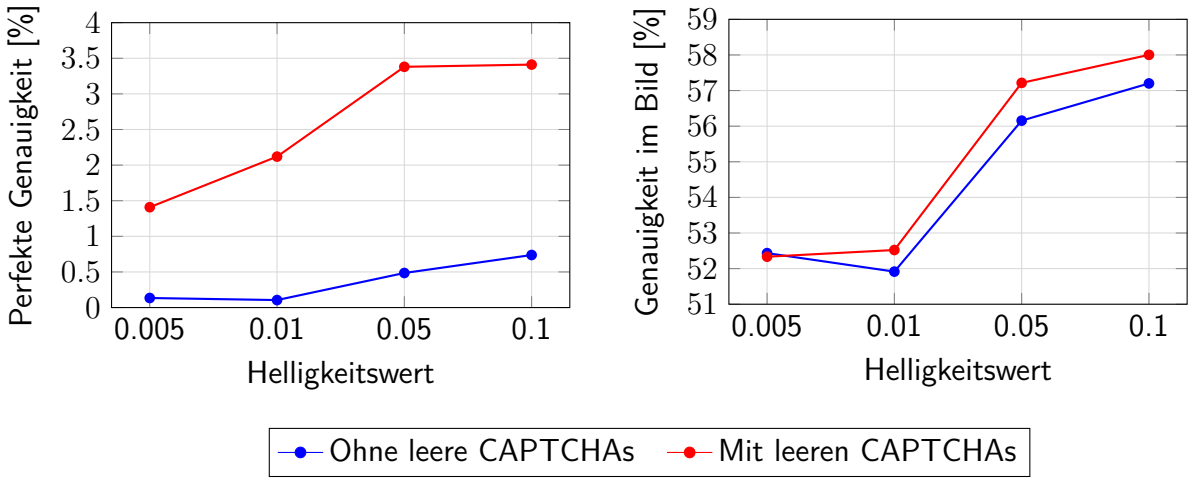
Graphen für alle Modelle




Sowohl die *pG*, als auch die *GiB* sind am höchsten bei Kontrastwerten im Bereich von 2-3. Hier lasse ich die Graphen für die besten Modelle weg, da sie keinen Mehrwert bringen.

**Helligkeit:**

Graphen für alle Modelle

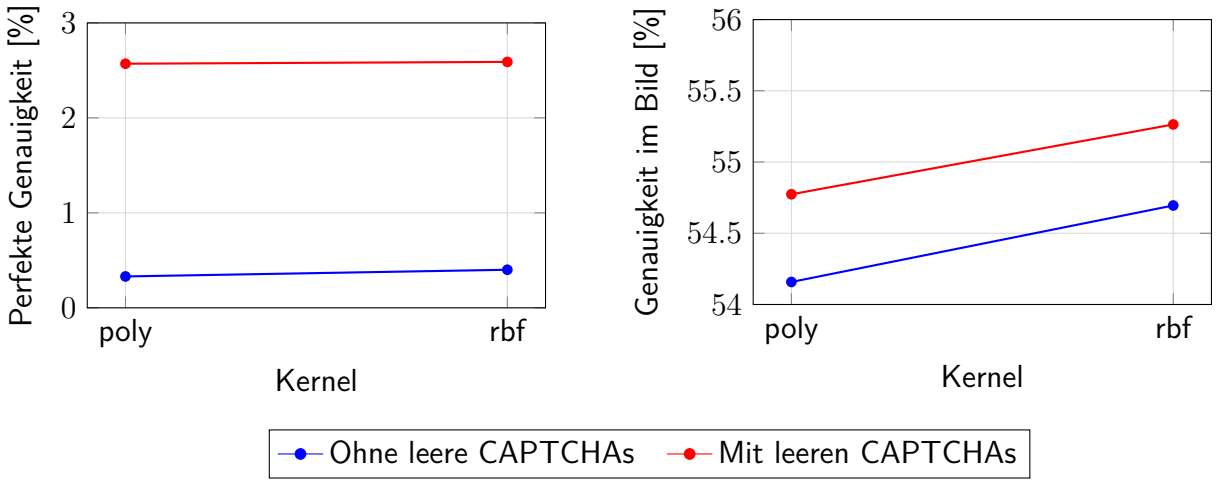



Die Modelle erzielten die besten Ergebnisse, wenn die Helligkeitswerte gross waren, was einer kleinen Helligkeitsveränderung der Bilder entspricht. Auch hier bringen die Graphen für die besten Modelle keine neuen Erkenntnisse, weswegen ich sie nicht aufzeige.

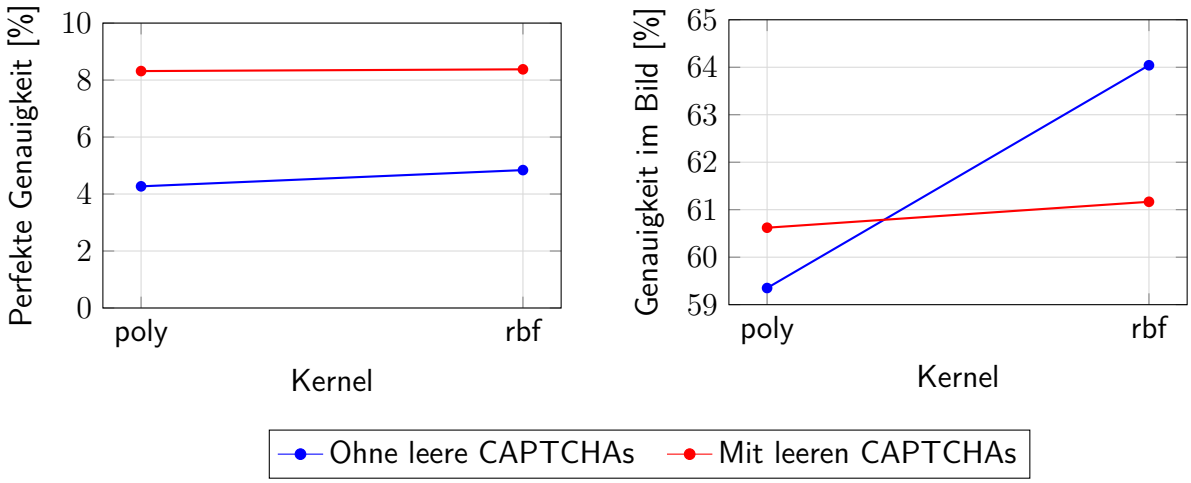
### 4.1.4 Einfluss des Kernels auf die Genauigkeit

In diesem Unterkapitel wird der Einfluss der beiden getesteten Kernels auf die Genauigkeit betrachtet.

Graphen für alle Modelle




Graphen für die besten Modelle




Das „rbf“ Kernel war durchschnittlich etwas besser. Besonders auf die *pG* hatten die beiden Kernels aber praktisch keinen Einfluss. Bei den besten Modellen ohne den leeren CAPTCHAs ist die *GiB* mit dem „rbf“ Kernel aber deutlich besser.

### 4.1.5 Heraussuchen des besten Modells

**Mit leeren CAPTCHAs:**
Von den 3072 getesteten Kombinationen, waren 15 mit einer *perfekten Genauigkeit* von 8.87% die besten. Ich werde nicht alle 15 Parameterkombinationen der Modelle einzeln auflisten, ich erstelle aber eine Tabelle, in welcher die häufigsten Parameter der 15 besten Modelle aufgezählt werden.

| Parameter | Auswertung |
|---|---|
| Kernel | in 9 Fällen: rbf; in 6 Fällen poly |
| Block Size | in 8 Fällen: 8; in 7 Fällen 16 |
| Block Stride | in 9 Fällen: 8, in 6 Fällen 4 |
| Cell Size | in allen 15 Fällen: 4 |
| Kontrastwerte | in 6 Fällen: 3; in 5 Fällen: 1; in 4 Fällen: 2 |
| Helligkeitswerte | in 13 Fällen: 0.1; in 2 Fällen: 0.05 |
| Threshold | in 14 Fällen: 1; in 1 Fall: 0.75 |

Tabelle 4: Beste Parameter für *HOG*-Modelle mit leeren CAPTCHAs

Die besten Parameter stimmen alle mit den vorherigen Erkenntnissen überein. Nur dass die *Cell Size* in allen 15/15 Fällen 4 ist, war etwas unerwartet. Sie war zwar bei den besten Modellen mit leeren CAPTCHAs ein wenig besser, aber nicht so massiv.

**Ohne leere CAPTCHAs:**
Von den 3072 getesteten Kombinationen waren hier zwei Modelle mit einer *perfekten Genauigkeit* von 5.98% die besten. In der Tabelle sind die Parameter der zwei besten Modelle aufgeführt.

| Parameter | Auswertung |
|---|---|
| Kernel | in beiden Fällen: rbf |
| Block Size | in beiden Fällen: 16 |
| Block Stride | in beiden Fällen: 4 |
| Cell Size | in beiden Fällen: 8 |
| Kontrastwerte | in 1 Fall: 2; in 1 Fall: 3 |
| Helligkeitswerte | in beiden Fällen 0.1 |
| Threshold | in beiden Fällen 0.75 |

Tabelle 5: Beste Parameter für *HOG*-Modelle ohne leere CAPTCHAs

Diese Parameter stimmen genau mit den vorherigen Erkenntnissen überein. Die beiden Kombinationen stimmten in allen Parametern überein, ausser im Kontrastwert, wobei sich aber auch dieser nur leicht veränderte.

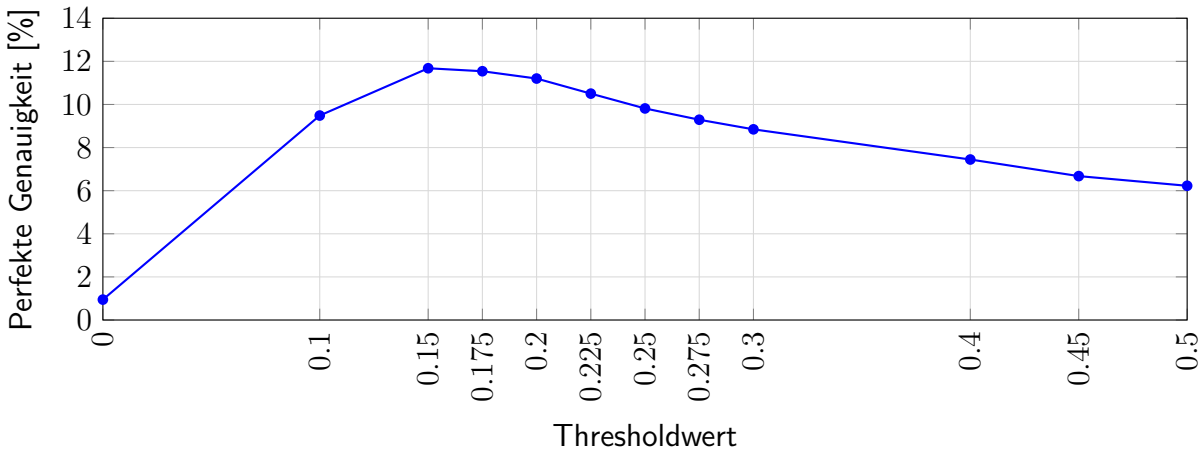
Da ich bei den anderen Modellen die leeren CAPTCHAs immer berücksichtigte, nehme ich auch hier die Werte mit leeren CAPTCHAs. Das beste Modell, welches mit *Histograms of Oriented gradients* und *Support Vector Machines* trainiert wurde, erreichte also eine *perfekte Genauigkeit* von **8.87%** und eine *Genauigkeit im Bild* von **61.24%**.

## 4.2 Neuronale Netzwerke

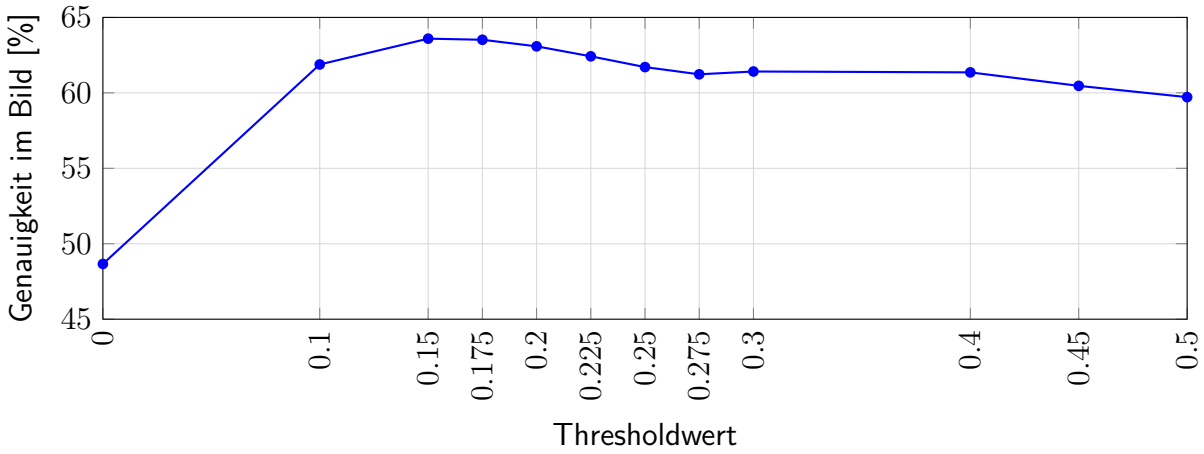
### 4.2.1 Generelle Analyse aller Modelle

**Threshold:**

Durchschnittliche *pG* für Thresholdwerte über alle Modelle

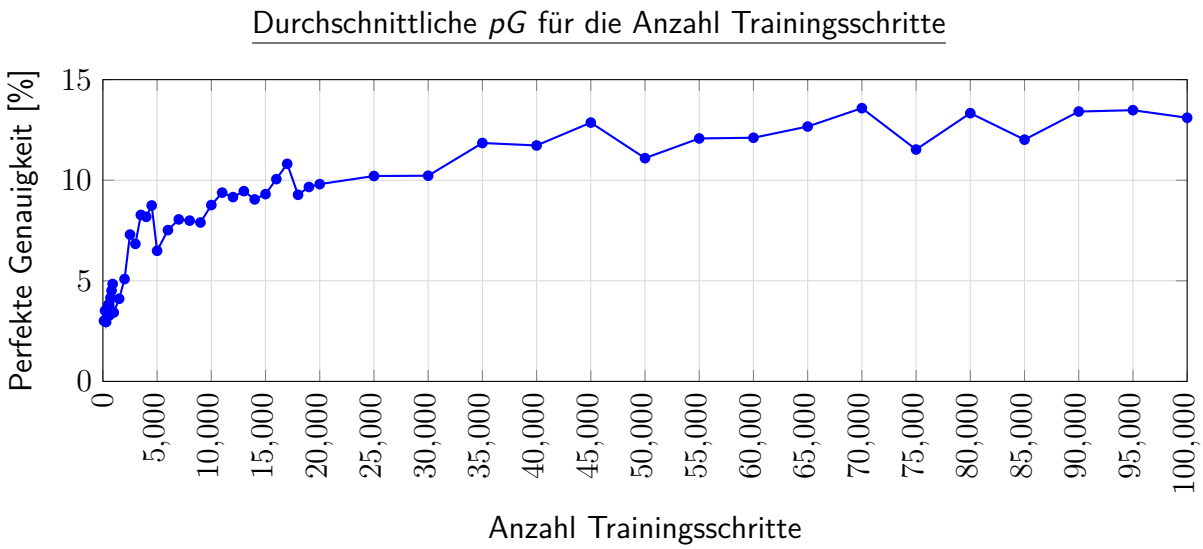


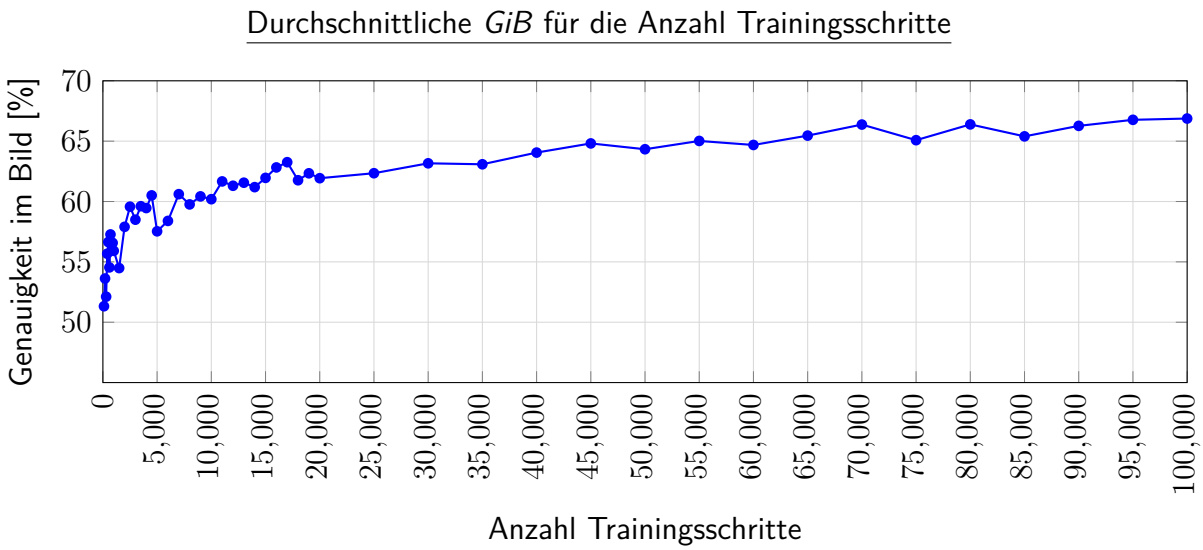

Durchschnittliche *GiB* für Thresholdwerte über alle Modelle




Die Resultate der Modelle waren beim Thresholdwert von 0 mit grossem Abstand am schlechtesten und bei einem Thresholdwert von 0.15 am besten. Nach dem Wert 0.15 nahm sowohl die *pG* als auch die *GiB* wieder langsam ab.

**Trainingsschritte:**

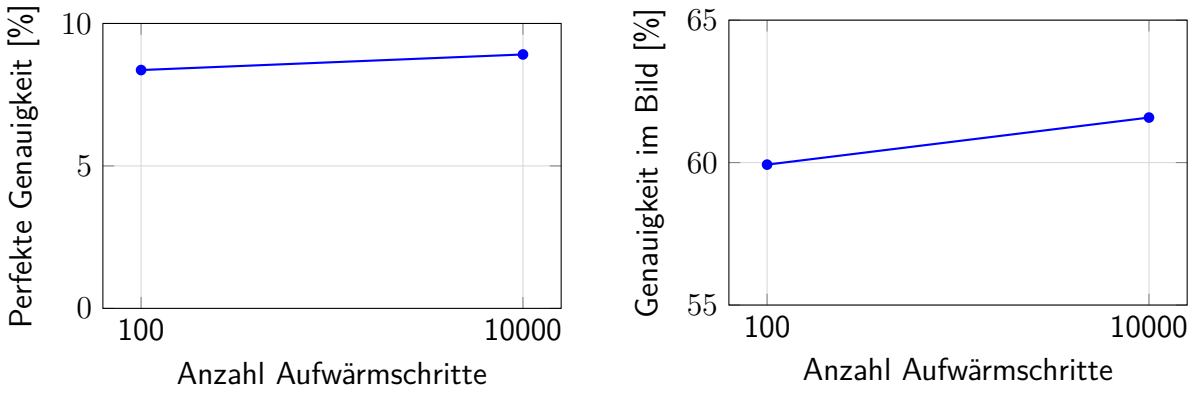



Mit wachsender Anzahl Trainingsschritte sind die Genauigkeiten höher. Zudem kann man gut sehen, dass die Genauigkeiten zu Beginn deutlich schneller steigen als am Schluss. Das ist auch so zu erwarten, denn das Netzwerk startet am Anfang mit zufälligen Parametern, welche im Verlauf immer besser optimiert werden. Am Anfang ist es somit einfach, das Modell zu verbessern. Je mehr Trainingsschritte es aber werden, desto optimierter sind die Parameter bereits und es ist schwieriger für das Modell, sie noch deutlich zu verbessern.

**Aufwärmschritte:**

Durchschnittliche Genauigkeiten für die Anzahl Aufwärmschritte

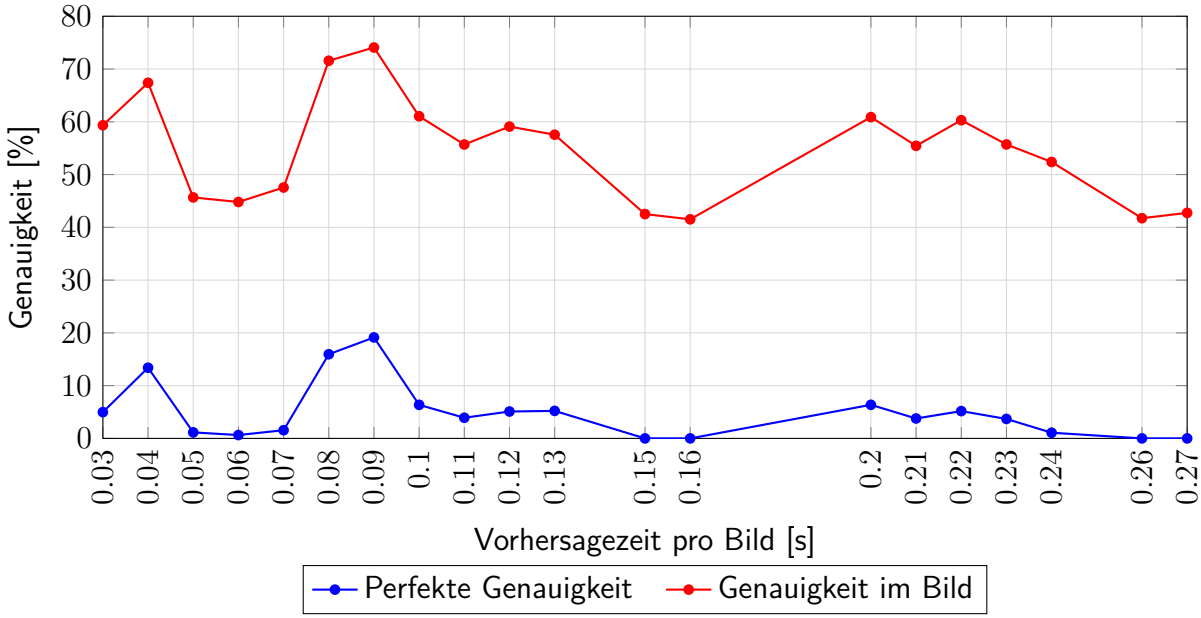



Meine Modelle hatten höhere Genauigkeiten mit 10'000 als mit 100 Aufwärmschritten. Es scheint also in meinem Fall zu besseren Resultaten zu führen, wenn die Modelle etwas länger mit kleinen Lernraten trainieren, bevor sie zu grösseren übergehen. Die Modelle mit 10'000 Aufwärmschritten hatten so länger Zeit, sich zu stabilisieren.

**Zeitaufwand pro Vorhersage:**

Es wurde bei jedem Modell gemessen, wie lange es pro Bild dauert, eine Vorhersage zu erlangen. Grössere Modelle brauchen länger, um eine Vorhersage zu machen als kleinere. Die Zeiten für eine Vorhersage wurden auf hundertstel Sekunden gerundet.

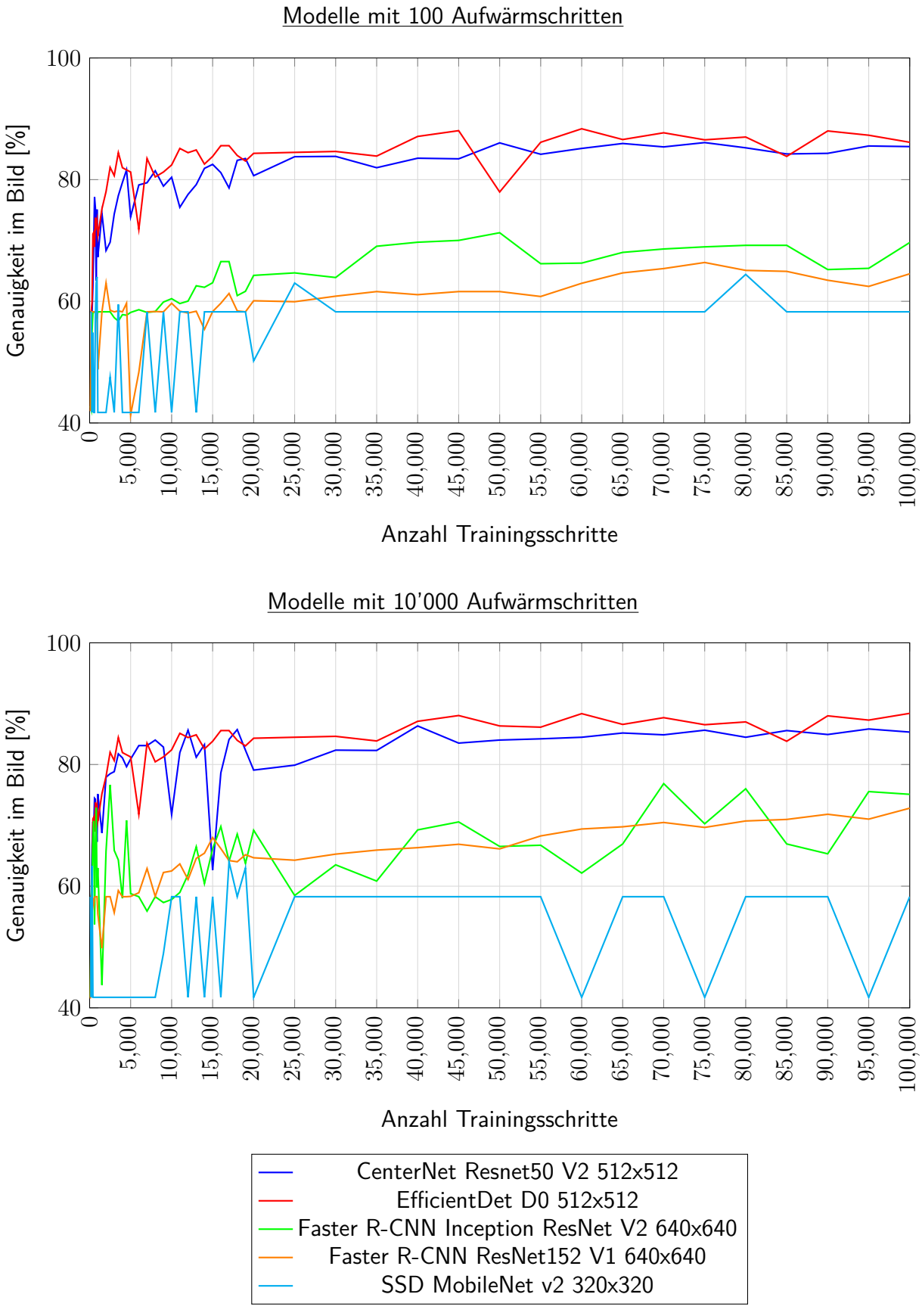
Durchschnittliche *pG* und *GiB* für die Vorhersagezeit pro Bild




Die höchsten Genauigkeiten wurden wider der Erwartung bei Zeiten von 0.08 - 0.09 Sekunden erreicht. Naheliegend wäre vorerst, dass die grossen Modelle, die etwas länger für ihre Vorhersagen brauchen, auch bessere Resultate erzielen. Vermutlich trat bei den grösseren Modellen der „Overfitting"-Effekt auf. Das ist, wenn ein neuronales Netzwerk zu viele Details, inklusive das Bildrauschen aus den Trainingsdaten, lernt, was zu schlechten Resultaten bei ungesehenen Daten führt (vgl. Shiv, 2020).

### 4.2.2 Vergleich zwischen verschiedenen Modellen

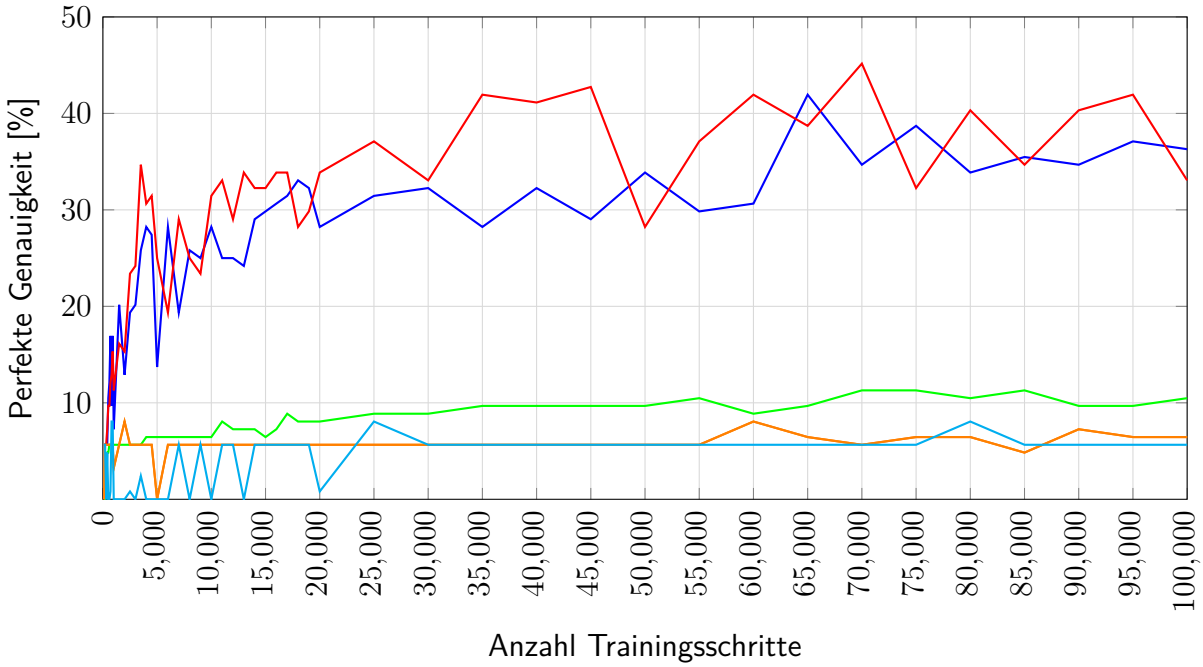
**Trainingsschritte:**



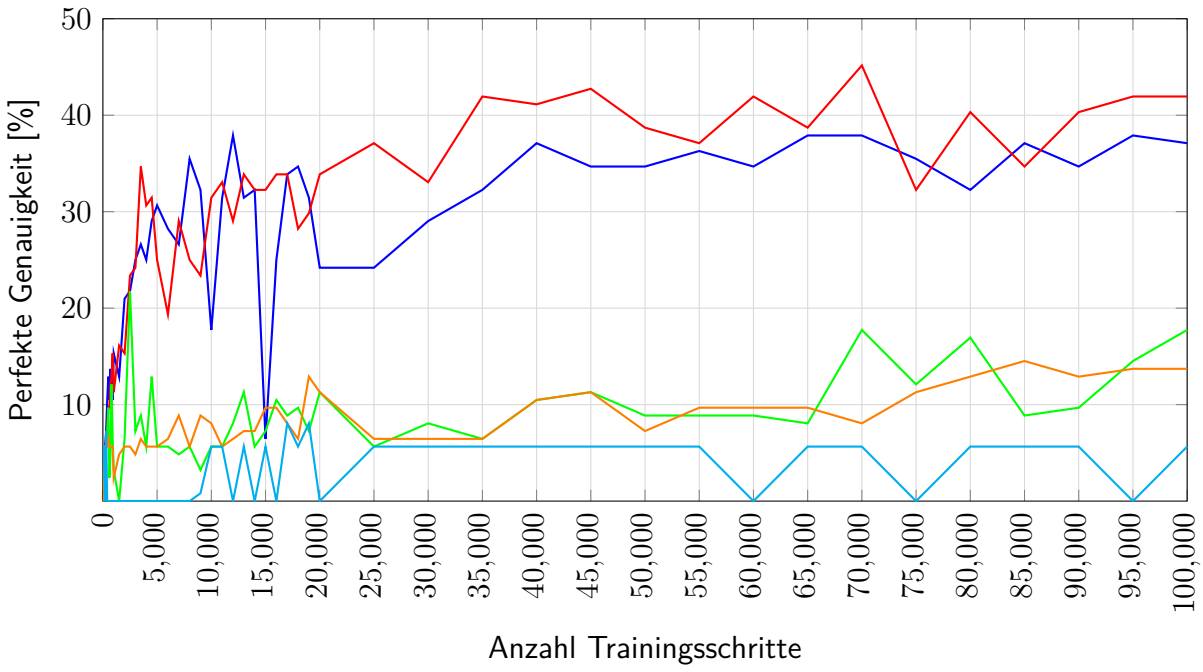
Sowohl bei den Modellen mit 100 als auch bei den Modellen mit 10'000 Aufwärmschritten waren zwei Modellarchitekturen deutlich besser als die anderen. Nämlich *CenterNet Resnet50 V2 512x512* und *EfficientDet D0 512x512*, wobei das zweite noch etwas besser abschnitt. Bei diesen beiden Modellen stabilisierte sich die *GiB* nach 20'000 Trainingsschritten relativ stark.

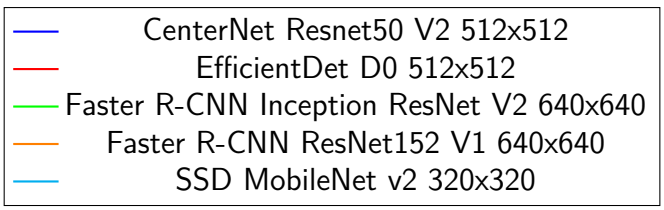
*Faster R-CNN Inception ResNet V2 640x640* und *SSD MobileNet v2 320x320* schwanken ziemlich stark, wobei die *GiB* bei 100 Aufwärmschritten noch etwas stabiler war. Die *GiB* für das *Faster R-CNN ResNet152 V1 640x640* Modell schien nach 100'000 Trainingsschritten immer noch leicht am steigen zu sein.

Modelle mit 100 Aufwärmschritten



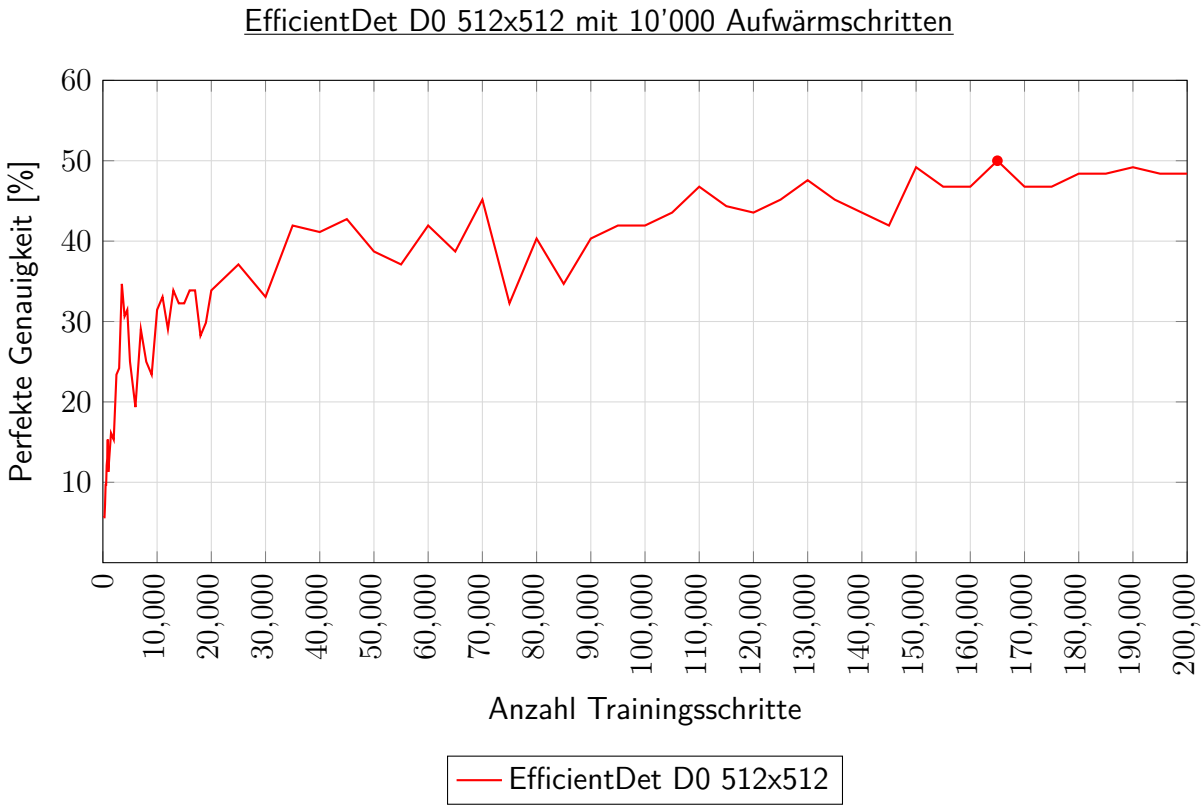

Modelle mit 10'000 Aufwärmschritten



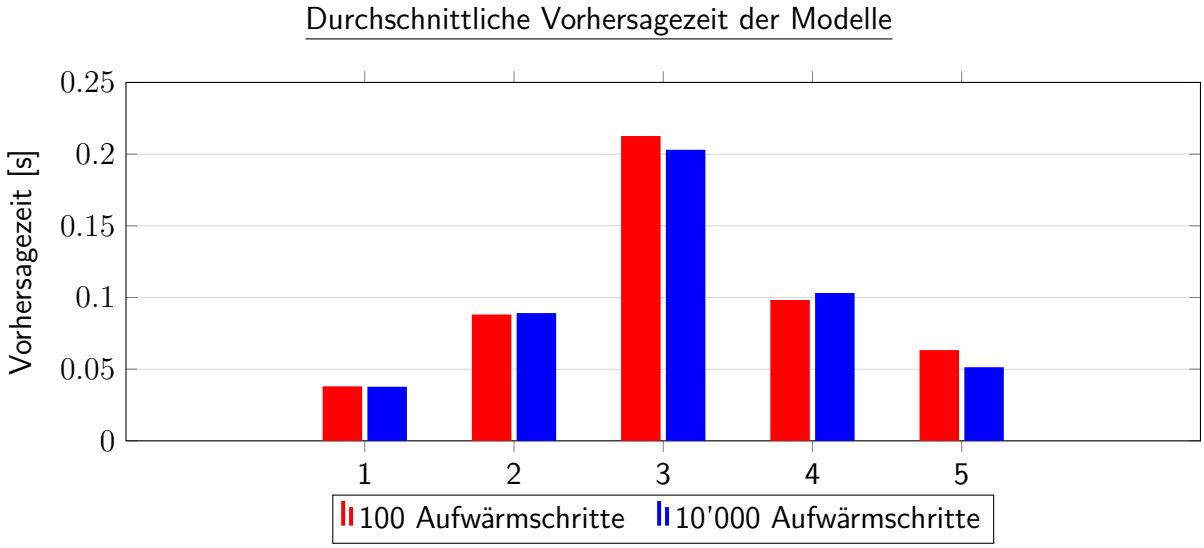
Gleich wie bei der *GiB* waren *CenterNet Resnet50 V2 512x512* und *EfficientDet D0 512x512* in Bezug auf die *pG* mit Abstand die besten. Auch hier war *EfficientDet D0 512x512* ein wenig besser. Dieses erreichte nach 70'000 Schritten eine *perfekte Genauigkeit* von 45.16%, was bis zu 100'000 Schritten das Maximum der *pG* darstellte. Zufällig war dies sowohl mit 100 als auch mit 10'000 Aufwärmschritten genau bei 70'000 Schritten der Fall.
Die drei restlichen Modelle waren deutlich schlechter, wobei besonders das *SSD MobileNet v2 320x320* Modell schlechte Ergebnisse erzielte. Ich untersuchte im folgenden Teil v.a. das beste Modell, da ich Hoffnungen hatte, dieses noch etwas zu verbessern.

Obwohl das beste Modell (*EfficientDet D0 512x512*) kaum noch Fortschritte zu machen schien, liess ich das Modell mit 10'000 Aufwärmschritten noch bis zu 200'000 Trainingsschritten weiter trainieren. Dies machte ich nur mit diesem einen Modell, da hier die Hoffnungen am grössten waren.




Tatsächlich verbesserte sich das Modell noch etwas. Bei 165'000 Schritten erzielte das Modell das beste Resultat mit einer *perfekten Genauigkeit* von 50%. Somit erkannte das Modell rund jedes zweite Bild vollständig richtig. Bei weiteren 21% der Bilder stimmten 15/16 Kästchen innerhalb des Bildes, was bedeutet, dass das Modell bei diesen Bildern nur sehr knapp daneben lag.

**Vorhersagezeiten:**
Die nächsten zwei Balkendiagramme demonstrieren die durchschnittliche Vorhersagezeit pro Bild für die verschiedenen Modelle. Es wurden nur die Vorhersagezeiten des jeweils besten Modells einer Architektur genommen. Zudem wird zwischen den Modellen mit 100 und 10'000 Aufwärmschritten unterschieden.




| Nummer | Modell |
|---|---|
| 1 | CenterNet Resnet50 V2 512x512 |
| 2 | EfficientDet D0 512x512 |
| 3 | Faster R-CNN Inception ResNet V2 640x640 |
| 4 | Faster R-CNN ResNet152 V1 640x640 |
| 5 | SSD MobileNet v2 320x320 |

Die Vorhersagezeiten variieren stark zwischen den einzelnen Modellen. *CenterNet Resnet50 V2 512x512* und *SSD MobileNet v2 320x320* sind die schnellsten Modelle. *EfficientDet D0 512x512* und *Faster R-CNN ResNet152 V1 640x640* brauchen für eine Vorhersage etwa doppelt so lange. *Faster R-CNN Inception ResNet V2 640x640* ist mit grossem Abstand das langsamste Modell.

Die Anzahl Aufwärmschritte scheint keinen Einfluss auf die Vorhersagezeit zu haben, denn diese unterscheiden sich zwischen den beiden untersuchten Anzahlen an Aufwärmschritten nur wenig. Die kleinen Unterschiede könnten auch von der Temperatur der PC Hardware abhängen, oder davon, was sonst noch im Hintergrund am Computer erledigt wurde.

Die Genauigkeit scheint auch bei separater Betrachtung der Modelle keinen Zusammenhang mit den Vorhersagezeiten zu haben. Vermutlich würden diese Resultate bei Bildern mit besserer Auflösung aber ganz anders aussehen. In meinem spezifischen Fall waren die grösseren, langsameren Modelle aber nicht genauer als die kleinen.

### 4.2.3 Bestes neuronales Netzwerk

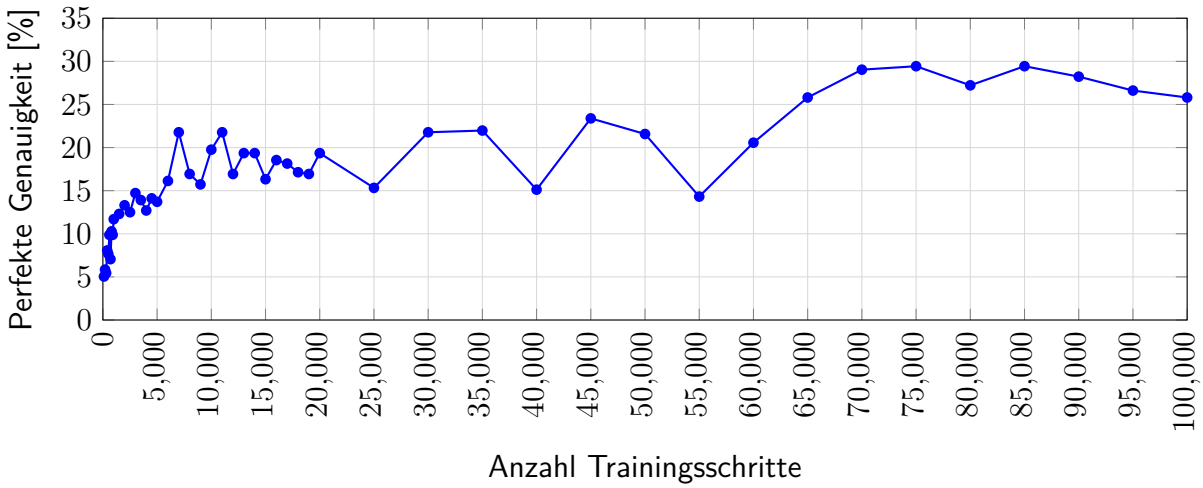
Viele trainierte Modelle erzielten sehr unterschiedliche Resultate. Einige davon waren sogar relativ schlecht, in Anbetracht, dass die Modelle stundenlang trainiert wurden. Das beste Modell, welches auf einem neuronalen Netzwerk basiert, ist das *EfficientDet D0 512x512* Modell nach 165'000 Trainingsschritten. Dieses erreichte eine *perfekte Genauigkeit* von **50%** und eine *Genauigkeit im Bild* von **89.42%**, was ziemlich eindrücklich ist.

## 4.3 Segmentierungsmodelle

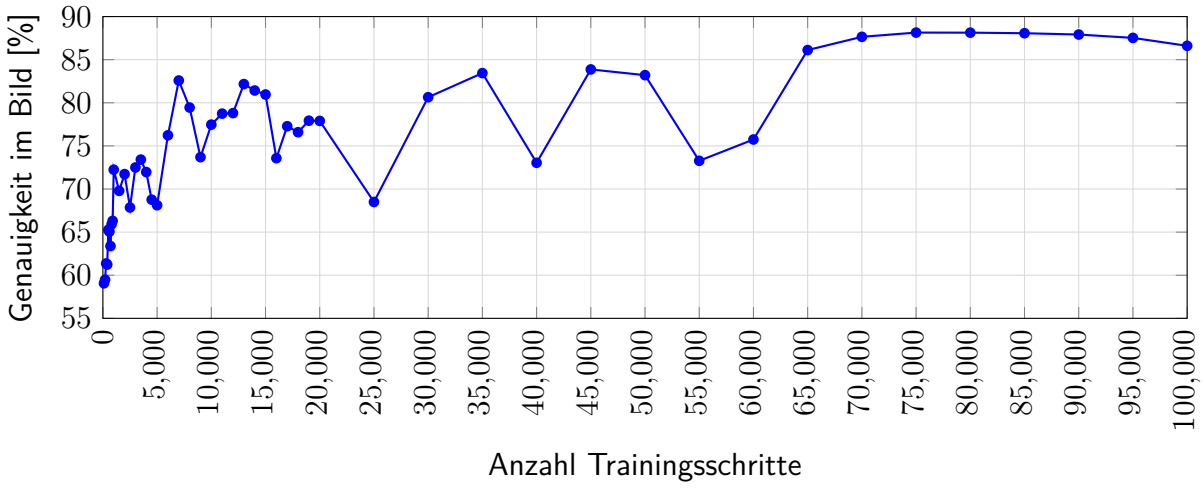
### 4.3.1 Generelle Analyse aller Modelle

**Trainingsschritte:**

Durchschnittliche *pG* für die Anzahl Trainingsschritte



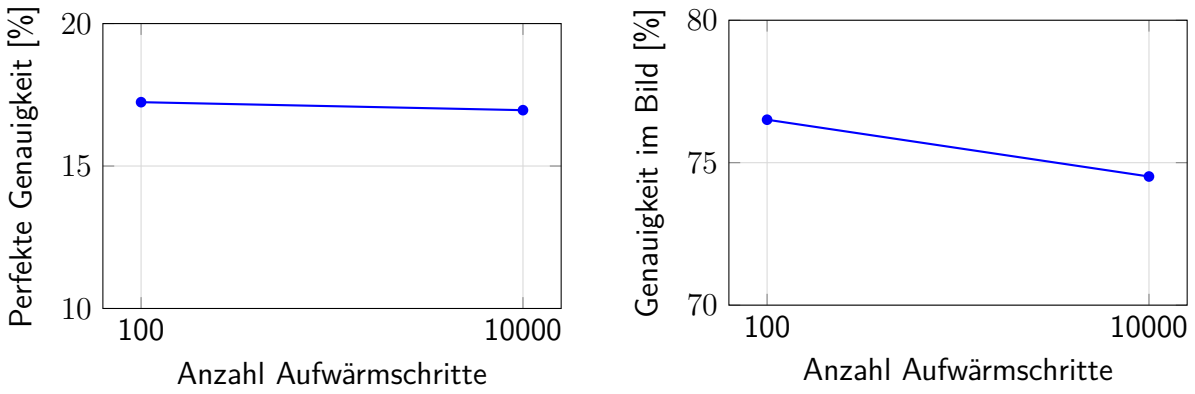

Durchschnittliche *GiB* für die Anzahl Trainingsschritte




Durchschnittlich ist sowohl die *pG*, als auch die *GiB* mit mehr Trainingsschritten höher. Der Graph für die *pG* sieht dabei sehr ähnlich aus wie der Graph der *GiB*. Auch hier kann man sehen, dass die Genauigkeiten zu Beginn stark zunehmen. Je länger dann trainiert wird, desto langsamer wird der Fortschritt.

**Aufwärmschritte:**

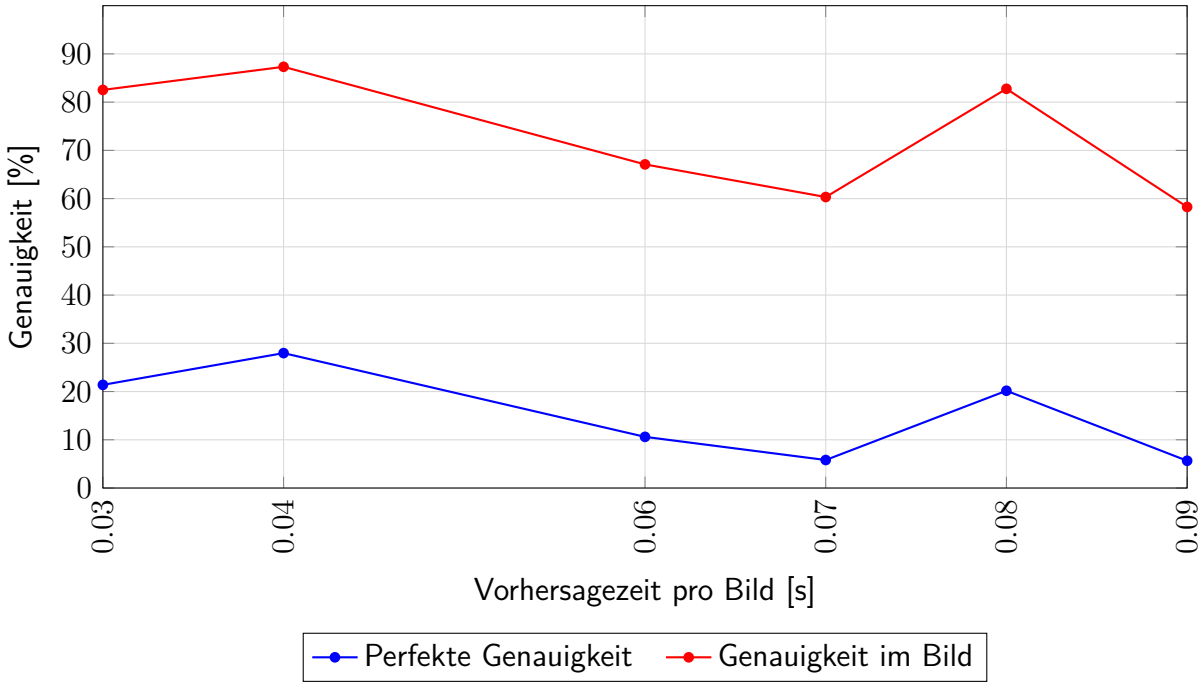
Durchschnittliche Genauigkeiten für die Anzahl Aufwärmschritte




Anders als bei den normalen neuronalen Netzwerken waren hier die Modelle mit 100 Aufwärmschritten etwas besser. Diese Modelle brauchten vermutlich nicht so lange, um sich zu stabilisieren und profitierten davon, schneller zu einer grösseren Lernrate überzugehen.

**Zeitaufwand pro Vorhersage:**
Auch hier wurde bei jedem Modell gemessen, wie lange es pro Bild dauert, um eine Vorhersage zu erlangen. Die Zeiten für eine Vorhersage wurden auf hundertstel Sekunden gerundet.

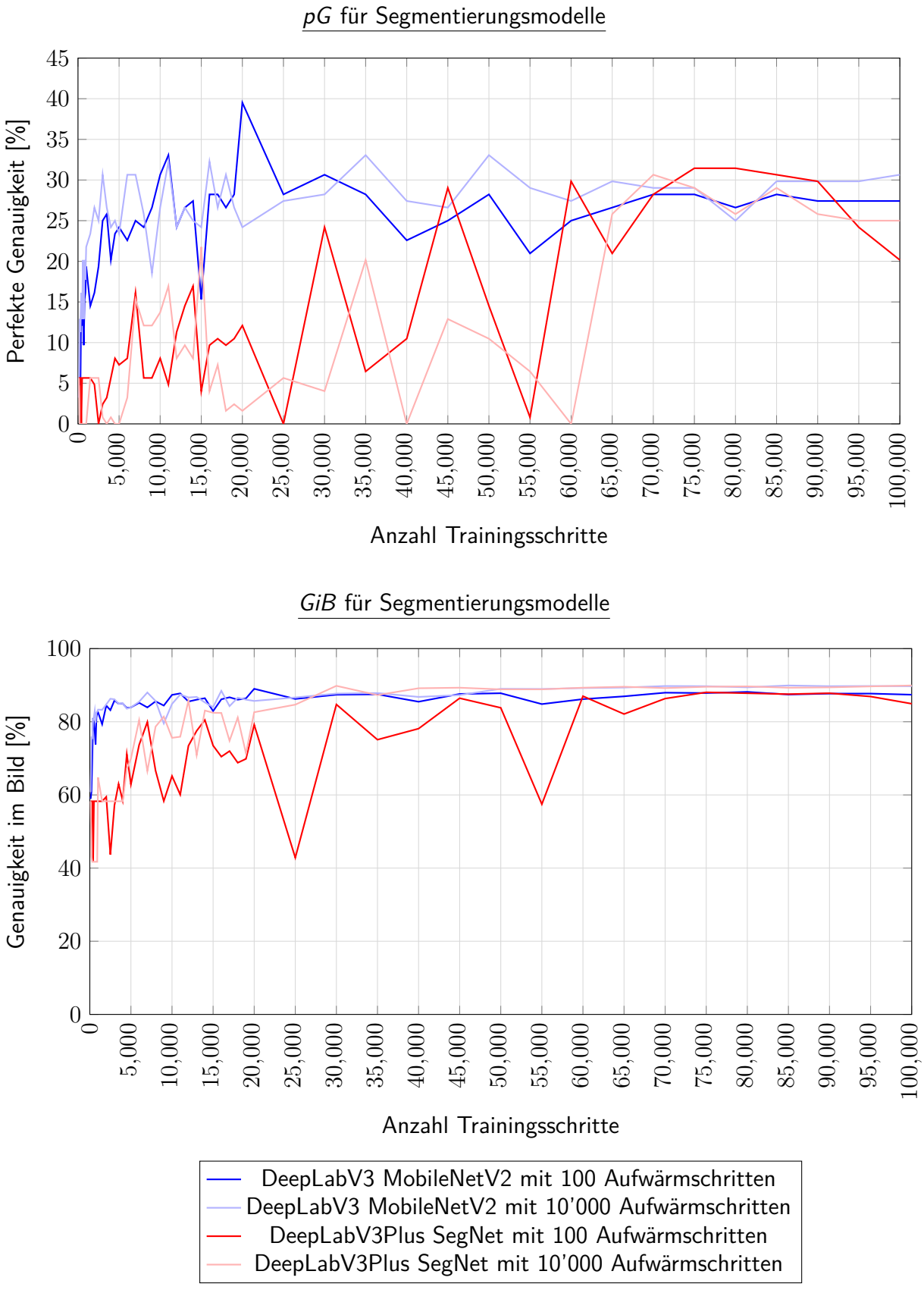
Durchschnittliche *pG* und *GiB* für die Vorhersagezeit pro Bild




Auch hier scheint es keinen Zusammenhang zwischen der Vorhersagezeit und der Genauigkeit zu geben. Vermutlich liegt es auch hier daran, dass die grösseren Modelle nicht profitieren können, da die Bilder schlecht aufgelöst sind.

### 4.3.2 Vergleich zwischen verschiedenen Modellen

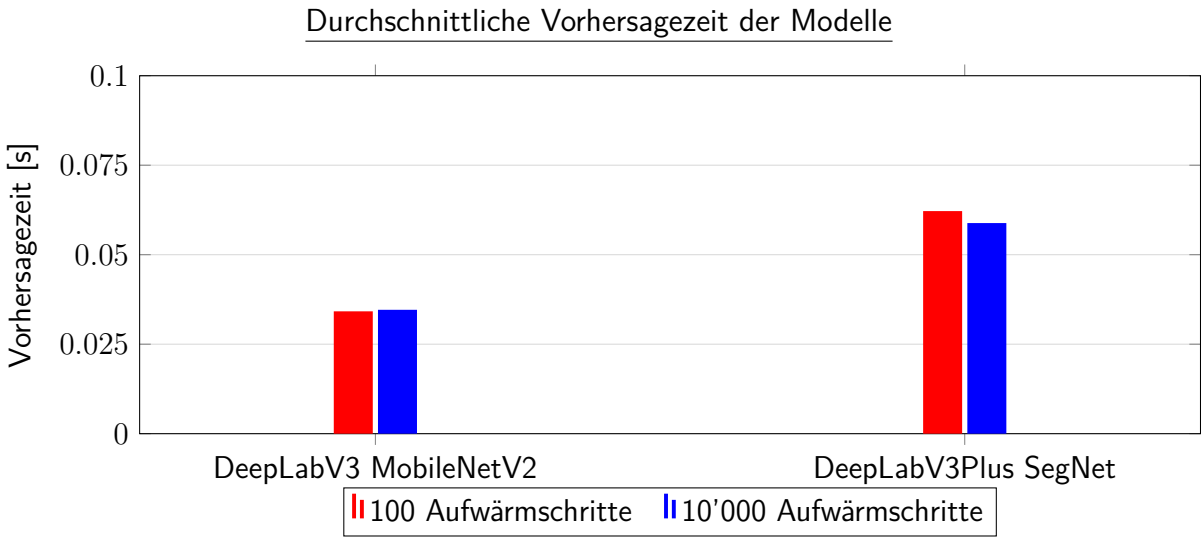
**Trainingsschritte:**



Bei Betrachtung der *pG* stabilisierten sich die beiden *DeepLabV3 MobileNetV2* Modelle deutlich schneller als die beiden *DeepLabV3Plus SegNet* Modelle, welche bis zu ca. 60'000 Trainingsschritten sehr stark schwankten. Nach 100'000 Trainingsschritten, waren aber alle Modelle vergleichbar gut, wobei die beiden *DeepLabV3 MobileNetV2* Modelle ein wenig besser waren. Das *DeepLabV3 MobileNetV2* Modell mit 100 Aufwärmschritten erreichte nach 20'000 Trainingsschritten durch Zufall ein deutlich überdurchschnittliches Resultat mit einer *pG* von 39.52%.

Bei der *GiB* sieht es sehr ähnlich aus. Auch diese wurde bei den *DeepLabV3Plus SegNet* Modellen erst nach etwa 60'000 Trainingsschritten stabil, war aber schlussendlich praktisch identisch zu den *DeepLabV3 MobilenetV2* Modellen.

Ich entschied mich dagegen, eines dieser Modelle noch weiter zu trainieren, da sich die *pG* kaum noch veränderte und das beste Resultat durch Zufall erreicht wurde. Man sollte aber trotzdem beachten, dass diese Modelle eventuell noch etwas besser hätten sein können, wenn sie noch weiter trainiert worden wären.




Die *DeepLabV3Plus SegNet* Modelle sind etwa 75% langsamer als die *DeepLabV3Plus MobileNetV2* Modelle. Die Vorhersagezeit hat also keinen Einfluss auf die Genauigkeit der Modelle, da beide ähnlich gut waren, wobei das schnellere Modell sogar noch etwas besser war.

Die Anzahl Aufwärmschritte scheint auch hier keinen Einfluss auf die Vorhersagezeit der Modelle zu haben.

### 4.3.3 Bestes Segmentierungsmodell

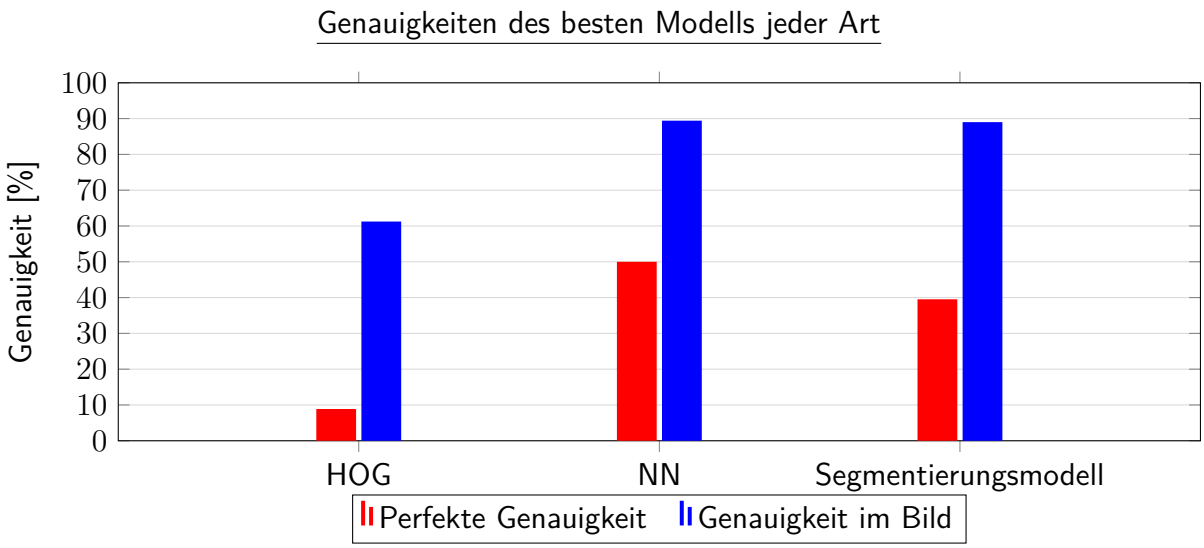
Das beste von mir trainierte Modell war also das *DeepLabV3 MobileNetV2* mit 100 Aufwärmschritten nach 20'000 Trainingsschritten. Dieses schnitt mit einer *perfekten Genauigkeit* von **39.52%** und einer *Genauigkeit im Bild* von **89.01%** sehr respektabel ab.

## 4.4 Vergleich zwischen Methoden

In diesem Kapitel vergleiche ich das jeweils beste Modell der untersuchten Methoden miteinander.

### 4.4.1 Genauigkeiten der Modellarten

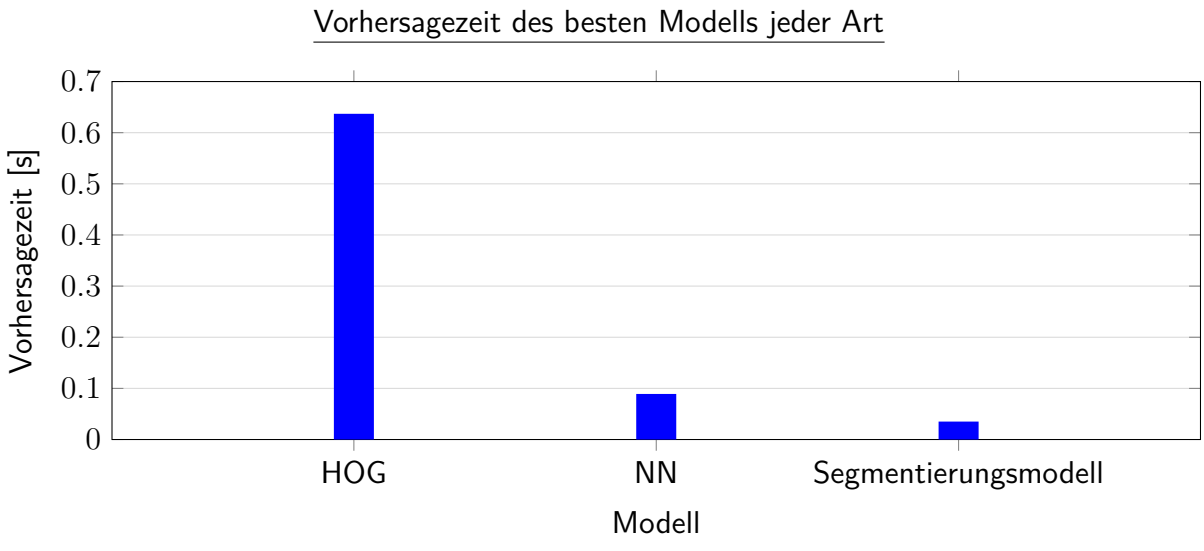
Ich wählte jeweils das Modell mit der höchsten *perfekten Genauigkeit* und nahm die korrespondierende *Genauigkeit im Bild*. Es gab also bei allen Modellarten noch Modelle mit leicht höheren *Genauigkeiten im Bild*. Da jedoch bei einem CAPTCHA alle Kästchen stimmen müssen, um es richtig zu lösen, orientierte ich mich an der *perfekten Genauigkeit*.




Die *HOG* Methode ist mit grossem Abstand die schlechteste der getesteten Methoden. Mit einer *perfekten Genauigkeit* von nur gerade 8.87% ist sie praktisch unbrauchbar. Auch die *Genauigkeit im Bild* (61.24%) überzeugt nicht, da es pro Kästchen im Bild nur zwei Optionen gibt. Würden die Kästchen rein zufällig angewählt werden, wäre nämlich bereits eine *GiB* von 50% zu erwarten.

Die beiden auf neuronalen Netzwerken aufgebauten Methoden sind deutlich besser. Die normale Objekterkennung mit *Bounding Boxes* ist bezogen auf die *pG* noch ein gutes Stück besser als das Segmentierungsmodell. Die *GiB* ist für beide fast identisch. Die normalen *NN*s sind 0.41% besser, wobei beide nur knapp unter 90% liegen.

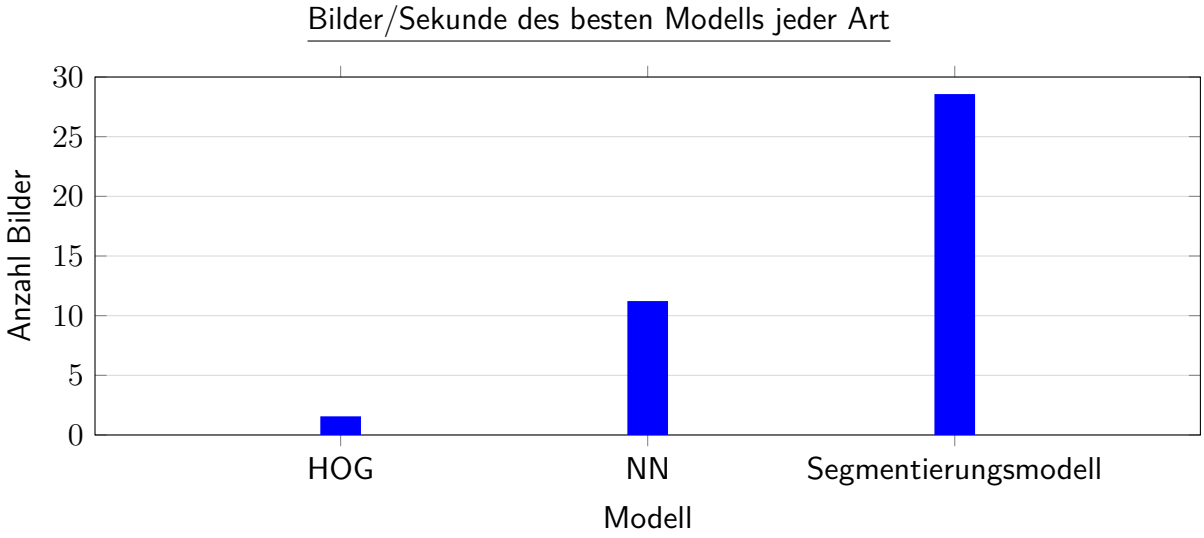
### 4.4.2 Vorhersagezeiten



Nicht nur war das *HOG* Modell mit Abstand das schlechteste, es ist auch mit Abstand das langsamste. Es ist etwa 7x langsamer als das beste *NN* und sogar 18x langsamer als das beste Segmentierungsmodell.

Der einzige Vorteil des *HOG* Modells lag in der Trainingszeit oder, anders ausgedrückt, der Zeit, bis es einsatzbereit war. Während das *NN* etwa 15 Stunden und das Segmentierungsmodell 6 Stunden für das Training benötigten, war das Training der *SVM* nach zwei Sekunden abgeschlossen. Das Training war also 27'000 Mal schneller als beim besten neuronalen Netzwerk.

Trotzdem ist dieser Vorteil klein, da die Modelle jeweils nur ein Mal trainiert werden müssen. 15 Stunden sind zwar nicht wenig, aber wenn dafür die Resultate brauchbar sind, ist es auch nicht besonders viel. Zudem sind die anderen beiden Modellarten, sobald sie einmal trainiert sind, auch deutlich schneller als das beste *HOG* Modell.




Wenn man die Geschwindigkeit in Bildern pro Sekunde angibt, wird der Unterschied noch eindrücklicher. Während man mit dem Segmentierungsmodell mit fast 30 Bildern pro Sekunde praktisch in Echtzeit eine Erkennung durchführen kann, ist dies mit dem *HOG* Modell unmöglich.

Zusammenfassend kann man sagen, dass Computermodelle CAPTCHAs schon relativ gut lösen können. Da man bei CAPTCHAs jeweils auch mehrere Versuche hat, wäre es auch nicht problematisch, wenn der Computer ein paar Versuche braucht. Mit guten neuronalen Netzwerken könnten Bild CAPTCHAs also gelöst werden.

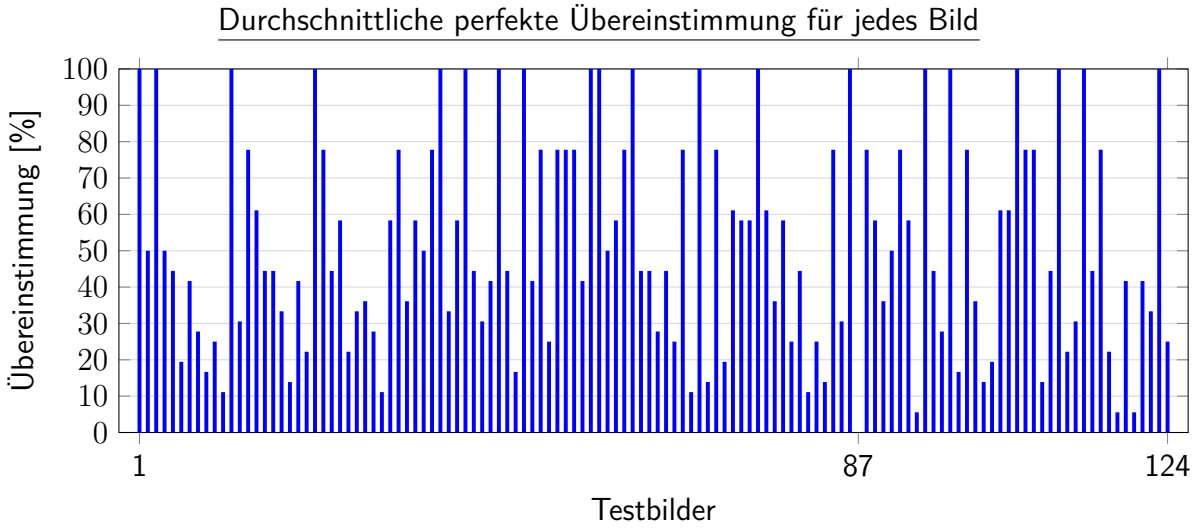
## 4.5 Vergleich zwischen Menschenantworten

Entgegen meinen Erwartungen wurden die CAPTCHAs von Person zu Person sehr verschieden gelöst. Die durchschnittliche perfekte Übereinstimmung, also die Prozentzahl von CAPTCHAs, die von zwei Personen jeweils genau gleich gelöst wurden, war nämlich nur gerade 51.88%.

Bei vielen CAPTCHAs ist den Menschen klar, welche Kästchen sie anklicken sollten, es gibt jedoch immer wieder solche, bei denen das Objekt knapp an der Grenze eines Kästchens liegt. Bei diesen CAPTCHAs ist man sich dann meist nicht sicher, ob dieses Kästchen nun noch

dazu gehört oder nicht. Mich interessierte es also, ob es auch bei meinen Testergebnissen Bilder gab, welche praktisch von allen gleich gelöst wurden, und solche, die sehr unterschiedlich gelöst wurden. Dazu habe ich für jedes Bild bei jedem Vergleich zwischen zwei Personen gespeichert, ob das CAPTCHA gleich oder anders gelöst wurde. Dann habe ich für jedes Bild berechnet, wie viele Prozent das CAPTCHA gleich gelöst haben.

Im Graph unten habe ich für alle 124 Bilder die perfekte Übereinstimmung in Prozent dargestellt.




Wie erwartet gab es viele Bilder, bei denen sich die Testpersonen unstrittig über die Lösung waren. Es gab hingegen auch viele Bilder, bei welchen sich weniger als 50% der Personenpärchen einig über die Lösung waren. Besonders spannend fand ich das Bild 87, bei welchem jede einzelne Person eine andere Lösung angegeben hat. In diesem Bild haben die beiden oberen linken Kästchen gerade noch eine kleine, aber undeutliche Überlappung, weswegen anwählen oder nicht beides valide Optionen sind. Zudem hat es in der unteren rechten Ecke Tramgeleise, welche eventuell auch als Trottoir angesehen werden könnten.

Abbildung 37: Bild 87

Vielleicht hätten die Personen ähnlicher geantwortet, wenn zu Beginn definiert worden wäre, was als Fussgängerstreifen gilt und ob ein Kästchen auch angewählt werden muss, wenn der Fussgängerstreifen nur knapp ins nächste Kästchen übergeht. Da dies in der Wirklichkeit aber auch unbekannt ist, entschied ich mich dagegen, etwas zu sagen. Es ist auch möglich, dass höher aufgelöste Bilder zu höheren Übereinstimmungsraten geführt hätten. Dann müsste man aber auch annehmen, dass die Computermodelle etwas besser abgeschnitten hätten.

Zieht man also in Betracht, dass die Übereinstimmung der CAPTCHAs selbst bei Menschen nur knapp über 50% liegt, ist das Resultat der Computermodelle noch eindrücklicher.

Die *Genauigkeit im Bild* lag bei den Menschen durchschnittlich bei 93.36%. Diese ist also etwas höher als bei den Computermodellen, woraus man schliessen kann, dass den Computermodellen

doch noch etwas mehr Fussgängerstreifen entgehen als den Menschen. Der Unterschied beträgt aber auch hier nur gerade ca. 3%.

Die durchschnittliche Zeit, welche die Menschen pro CAPTCHA gebraucht haben, war knapp über 5 Sekunden. Diese Zeit sagt aber nicht sehr viel aus, da Menschen die Fussgängerstreifen deutlich schneller identifizieren, aber eine gewisse Zeit brauchen, um sich durch die Kästchen zu klicken. Trotzdem ist es spannend zu sehen, dass die Computermodelle bei ähnlicher Genauigkeit über 50x schneller waren als die Menschen.

# 5 Diskussion der Ergebnisse

## 5.1 Interpretation der Ergebnisse in Bezug auf die Fragestellungen

Mit meinen Untersuchungen konnte ich beide meiner Hypothesen bestätigen. Computermodelle schaffen es tatsächlich, CAPTCHAs zu lösen oder lösen diese zumindest praktisch gleich gut wie die Menschen. Die Bestätigung der Hypothese, dass neuronale Netzwerke die besten Resultate liefern, liegt im Einklang mit dem plötzlichen enormen Aufschwung von künstlicher Intelligenz. Denn die meisten *KI* Modelle, so unter anderem auch ChatGPT und DALL·E, sind auf neuronalen Netzwerken aufgebaut (vgl. Aajush, 2023). Die Tatsache, dass der Erfolg neuronaler Netzwerke genau jetzt erfolgt, ist darauf zurückzuführen, dass Computer inzwischen die benötigte Rechenleistung haben, um *NN*s dieser Grösse trainieren zu können.

Die Leitfrage und die Teilfragen konnte ich durch meine Untersuchungen beantworten. Ich rechnete damit, dass Computermodelle in der Lage sind, CAPTCHAs zu lösen, besonders da Modelle wie DALL·E sogar in der Lage sind, ganze Bilder zu generieren. Was mich hingegen überraschte, war, dass Menschen in meinem Test nur knapp mehr als die Hälfte aller CAPTCHAs gleich lösten. Das wirft die Frage auf, ob CAPTCHAs überhaupt geeignet sind, um Menschen von Computern zu unterscheiden.

## 5.2 Kritische Beurteilung des methodischen Vorgehens

Ich investierte viel Zeit in das *Labeling* des Datensatzes, und trotzdem wäre ein grösserer Datensatz vermutlich die einfachste Anpassung gewesen, um zu noch besseren Resultaten zu gelangen. Vor allem hätte ich von einem Konzept namens „Data Augmentation“ profitieren können. Die *Data Augmentation* ist eine Technik, die verwendet wird, um die Grösse eines Datensatzes künstlich zu vergrössern. Dies kann durch leichte Bildtransformationen, wie etwa das Drehen, Spiegeln oder Abänderungen von Helligkeit und Kontrast der vorhandenen Bilder erreicht werden. Die *Data Augmentation* bringt also Variation in die Daten, wodurch das Modell mit mehr Daten trainiert werden kann (vgl. Abhishek, 2024). Mit dieser Technik hätte ich meinen Datensatz mit wenig Arbeit um ein Vielfaches vergrössern können.

Zudem hätte ich mehr darauf eingehen können, welche Art von Fehlern meine Computermodelle machen und diese mit der Art von Fehlern die Menschen begehen, vergleichen können. Während viele Fehler der Computermodelle daher stammen, dass sie einen Fussgängerstreifen komplett übersehen haben, liegt die Varianz bei den Menschen eher in kleineren Details, z.B. ob ein Kästchen noch angewählt werden sollte, weil ein Fussgängerstreifen potentiell noch knapp in dieses Kästchen übergeht.

Weiter hätte ich untersuchen können, ob sich die Resultate signifikant ändern, wenn man die Modelle mit anderen Trainingsdaten trainiert, oder mit anderen Testdaten getestet hätte. Damit meine ich nicht, dass ich einen komplett anderen Datensatz hätte verwenden können, sondern dass ich verschiedene zufällige Aufteilungen der Daten in die Test-, Validierungs- und Testsätze hätte ausprobieren können.

## 5.3 Fazit

Die verschiedenen Objekterkennungsmethoden erreichten ganz unterschiedliche Resultate. In dieser Tabelle sind die höchsten erreichten *perfekten Genauigkeiten* der jeweiligen Methoden mit der dazugehörigen *Genauigkeit im Bild*. Die höchsten Werte sind jeweils grün, die tiefsten rot.

| **Methode** | ***pG*** | ***GiB*** |
|---|---|---|
| Histogram of Oriented Gradients | 8.87% | 61.24% |
| Neuronale Netzwerke | 50% | 89.42% |
| Segmentierungsmodelle | 39.52% | 89.01% |

Tabelle 6: Beste Resultate für die jeweiligen Methoden

Die schlechte Leistung der *HOG*-Modelle ist vermutlich darauf zurückzuführen, dass die *HOG*s auf Helligkeitsveränderungen im Bild beruhen. Der schwache Kontrast zwischen Strasse und Fussgängerstreifen führt häufig zu einem entsprechend schwachen Bildgradienten. Ein weiterer Grund ist sicherlich auch, dass diese Objekterkennungsmethode immer spezifische Bildabschnitte einzeln betrachtet und somit keinen Kontext zum restlichen Bild hat. Selbst als Mensch ist es teilweise unmöglich, zu sagen, ob es sich in einem Bildausschnitt um einen Fussgängerstreifen handelt, wenn die Umgebung des Bildausschnittes unbekannt ist. Somit ist es für den *SVM* Algorithmus auch unmöglich, eindeutige Klassifikationen zu machen. Der letzte Grund, den ich gefunden habe, ist, dass Fussgängerstreifen sehr verschiedene Anordnungen in den Bildern haben und die *SVM* so aus den Daten nur schwer Muster erkennen kann.

Da die leeren CAPTCHAs die Resultate verfälschten, interpretiere ich nur die Resultate der Modelle ohne die leeren CAPTCHAs. Bezüglich der *HOG*-Parameter fällt auf, dass die besten Resultate entstanden sind, als die *HOG*-Merkmale am meisten normalisiert wurden. Mehr normalisiert bedeutet hier, dass die Histogramme über grössere *Zellen* und grössere *Blöcke* berechnet wurden, während der *Block Stride* kürzer war. Vermutlich führte die stärkere Normalisierung zu besseren Resultaten, da das Rauschen in den Bildern sehr stark ist. Durch die Normalisierung wird dieses Rauschen etwas herausgefiltert, was wichtiger zu sein scheint als die kleinen Details, die durch die stärkere Normalisierung verloren gehen.
Das Kontrastwertoptimum wurde bei 3 erreicht, was wahrscheinlich der Fall ist, da die Originalbilder sehr kontrastarm sind. Wird der Kontrast etwas erhöht, sind die Gradienten stärker. Als der Kontrast aber zu stark erhöht wurde, gingen wieder zu viele Details im Bild verloren, weshalb die Genauigkeit vermutlich wieder etwas abnahm. Kleinere Helligkeitswerte führten vermutlich zu schlechteren Resultaten, da zu viele Fussgängerstreifen komplett unkenntlich gemacht wurden.

Die Interpretation für das Optimum des Thresholdwertes gilt sowohl für die *HOG* Methode als auch für die normalen *NN*s. Der Grund, weshalb die Resultate bei tiefen Thresholdwerten sehr schlecht sind, wird dabei liegen, dass es sehr viele mögliche Fussgängerstreifen mit einem tiefen Wahrscheinlichkeitswert gibt. Bei tiefen Thresholdwerten denken die Modelle dann, dass es praktisch überall im Bild Fussgängerstreifen hat und würden fast immer alle 16/16 Kästchen anwählen. Das führt dann logischerweise zu schlechten Resultaten. Wird der Thresholdwert grösser, so gibt es immer weniger mögliche Fussgängerstreifen mit einer Wahrscheinlichkeit, die grösser ist als der Thresholdwert. Ab einem gewissen Punkt werden also praktisch keine Fussgängerstreifen mehr erkannt, was natürlich auch zu schlechteren Resultaten führt. Irgendwo in der Mitte liegt dann das Optimum.

Der Hauptgrund, weshalb die Objekterkennungsmethoden, die auf *NN*s beruhen, deutlich besser waren, ist vermutlich, dass bei diesen Methoden der Kontext der Bilder miteinbezogen wurde. Zudem werden neben den Bildgradienten auch andere Informationen aus den Bildern extrahiert, welche auch hilfreich für die Erkennung sind. Welche Informationen das genau sind, kann man aus den Netzwerken aber leider nicht herauslesen.

Die *NN*s, welche mit *Bounding Boxes* gearbeitet haben, waren vermutlich etwas besser, da diese Modelle etwas weniger komplex sind als die Segmentierungsmodelle. Während die *NN*s mit *Bounding Boxes* die Fussgängerstreifen nur grob erkennen mussten, mussten die Segmentierungsmodelle pixelgenaue Masken erstellen, wozu sie viel kleinere Details im Bild miteinbeziehen mussten. Zudem wurden die Bilder bei den Segmentierungsmodellen zuerst verkleinert, wodurch Details verloren gingen. Da der Vorteil der pixelgenauen Masken der Segmentierungsmodelle beim Lösen von CAPTCHAs keinen wirklichen Vorteil mit sich bringt, führte die höhere Komplexität vermutlich zu schlechteren Resultaten.

Da in meinen Versuchen kein offensichtlicher Zusammenhang zwischen der Genauigkeiten und der Anzahl Aufwärmschritte bestand, ist es auch schwer, etwas daraus zu interpretieren. Die optimale Anzahl Aufwärmschritte variierte zwischen den Modellen stark, was es schwer macht, ein absolutes Optimum zu bestimmen. Da die unterschiedliche Anzahl Aufwärmschritte aber v.a. die Genauigkeit der ersten Trainingsschritte beeinflusste, hatte die Anzahl Aufwärmschritte keinen grossen Einfluss auf das Endresultat. Nach 100'000 Trainingsschritten waren sowohl die Modelle mit 100 als auch diejenigen mit 10'000 Aufwärmschritten praktisch gleich gut. Trainiert man das Modell also genügend lange, scheint die Anzahl Aufwärmschritte mehr oder weniger irrelevant zu sein.

Insgesamt wurden die Resultate der Modelle mit mehr Trainingsschritten immer besser, wobei die Verbesserungen der Resultate mit mehr Trainingsschritten immer kleiner wurden. Das macht so auch Sinn, denn mit mehr Trainingsschritten fanden auch mehr Optimierungen der Parameter statt. Je länger das Modell bereits trainiert wurde, desto optimierter sind die Parameter bereits und das Verbesserungspotential wird kleiner. Somit wird die Verbesserung mit mehr Trainingsschritten kleiner.

Bei den normalen *NN*s wurden fünf verschiedene Modellarchitekturen getestet, wobei zwei deutlich besser waren als die anderen. Das könnte viele verschiedene Ursachen haben, eine ist aber vermutlich die Auflösung der Input Bilder, die bei beiden der besseren Modellen 512x512 war. Von den anderen drei Modellen, hatte eines eine Input Auflösung von 320x320, die restlichen zwei hatten eine Input Auflösung von 640x640. Das Modell mit der Input Auflösung

von 320x320 war vermutlich nicht in der Lage, genügend Details aufzunehmen, da es darauf ausgelegt ist, möglichst schnell zu sein. Die Modelle mit der Input Auflösung von 640x640 wurden wahrscheinlich Opfer vom Overfitting-Effekt und erzielten deswegen schlechtere Resultate.

Bei den Segmentierungsmodellen waren nach den 100'000 Trainingsschritten beide Modellarchitekturen vergleichbar gut. Die kleinen Unterschiede stammen vermutlich vom grundlegenden Aufbau der Modelle.

Die erstaunlich kleine perfekte Übereinstimmung in den Antworten von Menschen stammt von sehr kleinen Abweichungen in der Auswahl der Kästchen. Die Übereinstimmung im Bild war bei den Menschen 93.36%, was bedeutet, dass durchschnittlich weniger als ein Kästchen pro Bild anders angewählt wurde. Es scheint, dass die perfekte Übereinstimmung gerade wegen den CAPTCHAs so gering ist, bei denen Menschen nicht wissen, ob das eine Kästchen noch zur Auswahl dazu gehört.

Computermodelle lösen CAPTCHAs also fast gleich gut wie Menschen. Die perfekte Übereinstimmung aller CAPTCHAs bei den Menschen war nur 1.88% höher als die *perfekte Genauigkeit* beim besten Computermodell.
Die Übereinstimmung im Bild war bei den Menschen aber 3.94% höher als die *Genauigkeit im Bild* beim bestem Computermodell, woraus man schliessen kann, dass dem Computermodell doch noch etwas mehr Fussgängerstreifen entgehen als den Menschen.

## 5.4 Blick auf Text CAPTCHAs

Neben der Art von CAPTCHAs, die in dieser Arbeit behandelt wurden, gibt es viele weitere Arten. Eine andere, weit verbreitete CAPTCHA Art sind die Text CAPTCHAs (vgl. Abb. 38).
Im Artikel von Wan et al., 2024 wurde genau diese Art von CAPTCHAs behandelt.

Abbildung 38: Text basiertes CAPTCHA Thatcher, 2009

Der Artikel besagt, dass die textbasierten CAPTCHAs die meist verwendeten Sicherheitssysteme sind, um Webseiten vor Angriffen zu sichern. Um aber Schwachstellen der textbasierten CAPTCHAs zu finden, wurden Modelle entwickelt, welche diese Art von CAPTCHAs lösen können. Es wurden komplizierte neuronale Netzwerke trainiert, welche weitaus besser sind als diejenigen, die ich in meiner Arbeit trainierte. Anschliessend wurden die Modelle an zwei verschiedenen Testdatensätzen getestet.

Die besten Modelle, welche von den Autoren des Artikels trainiert wurden, konnten beim ersten Datensatz über 98% und beim zweiten Datensatz knapp weniger als 99% aller CAPTCHAs lösen. Diese Zahlen sind natürlich sehr eindrücklich und zeigen, dass Computer diese Art von CAPTCHA praktisch fehlerfrei lösen können. In dem Artikel wurden keine Versuche mit Menschen durchgeführt, wobei ich mir aber gut vorstellen könnte, dass Computer diese Art von CAPTCHA besser lösen können als Menschen.

Kombiniert man die Resultate dieses Artikels mit den Resultaten meiner Untersuchungen,

wird klar, dass das CAPTCHA keine sichere Methode mehr ist, um Webseiten vor Bots und Angriffen zu schützen. Dies ist besonders der Fall, weil die Modelle, welche CAPTCHAs lösen können, auch immer robuster gegenüber Veränderungen und Anpassungen der CAPTCHAs werden. In Zukunft werden also neue Sicherheitsmassnahmen erfunden und durchgesetzt werden müssen.

# 6 Zusammenfassung

Die Technologie, welche im letzten Jahr die grösste Veränderung für mein Leben darstellte, war ohne Frage die künstliche Intelligenz. ChatGPT hilft mir jetzt beispielsweise beim Zusammenfassen von Texten und bei der Vorbereitung auf Mathematikprüfungen, während ich mit DALL·E Bilder für Präsentationen generieren kann. Da ich mich seit längerem für die Informatik interessiere, wollte ich wissen, was hinter diesen grossen *KI* Systemen steckt und ob ich selbst eines bauen kann. Die spezifische Idee mit den CAPTCHAs kam mir durch Zufall, als ich einem auf einer Webseite begegnet bin. Denn ich fragte mich plötzlich, ob Computer diese lösen können, wenn sie auch viel kompliziertere Dinge machen können, wie ganze Texte oder Bilder generieren. Somit hatte ich meine Idee, ein Computermodell zu bauen, welches CAPTCHAs lösen kann.

Da es mir dabei vor allem um das Konzept ging, habe ich mich auf Bild CAPTCHAs mit Fussgängerstreifen beschränkt, obwohl ich mir der vielen anderen Arten bewusst war. Ich wollte mich lieber auf diese eine Art von CAPTCHAs konzentrieren, als viele verschiedene Arten, dafür aber oberflächlicher, zu behandeln. Herausfinden wollte ich vor allem, wie gut Computermodelle CAPTCHAs lösen können und in wie vielen Fällen sie die Tests tatsächlich bestehen würden.

Da ich zudem verstehen wollte, weshalb die künstliche Intelligenz ihren Aufschwung gerade jetzt hat und dieser nicht schon vor einigen Jahren geschah, habe ich mich auch mit älteren Methoden wie dem *Template Matching* und dem *Histogram of Oriented gradients* auseinandergesetzt.

Bevor ich diese Modelle zu implementieren begann, musste ich zuerst einen Datensatz erstellen. Glücklicherweise konnte ich geeignete Bilder auf dem Internet finden und musste diese nur noch beschriften. Ich beschriftete sie auf zwei verschiedene Arten, zum einen mit *Bounding Boxes*, zum anderen mit *Polygonen*. Anschliessend musste ich die Daten noch in die modellspezifischen Formate umwandeln.

Dann erst konnte ich mich richtig an das Implementieren der Methoden machen. Für alle in der Theorie behandelten Methoden, ausser für das *Template Matching*, schrieb ich ein Skript, welches zuerst verschiedene Modelle trainierte. Anschliessend wurden die verschiedenen Modelle ausgewertet, wobei die Daten für die spätere Analyse in einer Excel Datei abgespeichert wurden.

Die vorliegenden Daten wertete ich zuerst innerhalb der verschiedenen Methoden aus. Die Modelle, welche auf *HOG*s basierten, waren insgesamt relativ schlecht. Am meisten beeinflusst wurden deren Resultate vom Threshold, da diese Modelle von den leeren CAPTCHAs profitierten. Die neuronalen Netzwerke wurden auch stark vom Threshold beeinflusst, wobei

gewisse Thresholdwerte aber zu deutlich besseren Resultaten führten. Zudem war die Korrelation zwischen der Anzahl Trainingsschritten und der Genauigkeiten deutlich ersichtlich. Da die Segmentierung auch auf neuronalen Netzwerken basiert, waren die Erkenntnisse bei der Segmentierung praktisch gleich.

In meinem Anwendungsfall war das *HOG* ziemlich unbrauchbar, wobei man das so nicht generalisieren kann. In gewissen Fällen wie der Personenerkennung, die bei Sicherheitskameras von Bedeutung ist, funktioniert die Objekterkennung mit *HOG* sogar sehr gut (vgl. Terayama et al., 2009). Die neuronalen Netzwerke schnitten in meinen Versuchen aber viel besser ab. Während das beste *HOG* Modell eine *perfekte Genauigkeit* von 8.87% hatte, erreichte das beste neuronale Netzwerk eine *perfekte Genauigkeit* von 50%. Das beste Segmentierungsmodell lag mit 39.52% in der Mitte, womit es aber immer noch deutlich besser war als das beste *HOG* Modell.

Da ich selbst manchmal Mühe mit CAPTCHAs habe, dachte ich mir, dass es spannend wäre, einen Versuch mit Menschen durchzuführen. Ich wollte herausfinden, wie identisch Menschen CAPTCHAs lösen. Dazu liess ich neun Testpersonen 124 CAPTCHAs lösen und wertete diese anschliessend aus. Die Menschen lösten mit einer *perfekten Übereinstimmung* von 51.88% nur gerade etwa jedes zweite CAPTCHA gleich. Dieses Resultat überraschte mich sehr.

Das bedeutet somit aber auch, dass Menschen insgesamt CAPTCHAs gar nicht zwingend besser lösen als Computer. Dieses Resultat freut mich einerseits, da ich ein erfolgreiches Modell gebaut habe, zugleich beunruhigt es mich aber auch ein wenig. Denn somit ist die Methode, mit welcher sich viele Webseiten im Internet schützen, nicht mehr sicher. Auch bei anderen Arten von CAPTCHAs wie den Text CAPTCHAs haben Computer vergleichbare Erfolge, denn Computermodelle schaffen es, bei dieser Art von CAPTCHA über 98% richtig zu lösen. In Zukunft werden also andere Sicherheitsmassnahmen ergriffen werden müssen, um Computerangriffe von Bots zu verhindern.

Zukünftige CAPTCHA Systeme werden sich vermutlich eher auf das Verhalten des Benutzers auf einer Webseite konzentrieren wie z.B. die Mausbewegungen oder die Länge der Klicks, die sich bei Menschen immer leicht verändert, während es bei Computern stets konstant ist. CAPTCHAs könnten eventuell auch durch „Verifiable Credentials“, die in einer „Blockchain“ gespeichert sind, ersetzt werden. Diese *Verifiable Credentials* sind eine Art digitale ID, die wegen des Speichern in einer *Blockchain* nicht gefälscht werden kann (vgl. Conversation, 2024).

Spannend wäre noch gewesen, ob und wie sich die Resultate verbessert hätten, wenn ich meinen Datensatz durch *Data Augmentation* vergrössert oder einen ganz anderen Testsatz verwendet hätte. Potentiell hätte dies zu noch besseren Resultaten geführt. Ebenfalls könnte man Modelle, welche andere Objekte wie z.B. Ampeln erkennen sollten, trainieren und anschliessend miteinander vergleichen. Vielleicht würde man dabei auf Objekte stossen, bei welchen die Resultate deutlich anders sind, als diejenigen für die Fussgängerstreifen.

Ich bin gespannt, welche technologischen Fortschritte die kommenden Jahre bringen werden und hoffe, dass wir sowohl die Chancen als auch die Risiken der künstlichen Intelligenz verantwortungsbewusst angehen werden.

# 7 Quellenverzeichnis

## 7.1 Literaturverzeichnis

3Blue1Brown. (2017). *Neural Networks*. Verfügbar 30. April 2024 unter https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

Aajush, M. (2023). *Artificial Intelligence Generative AI: The Idea Behind CHATGPT, Dall-E, Midjourney and More*. Verfügbar 24. November 2024 unter https://www.unite.ai/generative-ai-the-idea-behind-chatgpt-dall-e-midjourney-and-more/

Abhishek, J. (2024). *Data Augmenation*. Verfügbar 19. November 2024 unter https://medium.com/@abhishekjainindore24/data-augmentation-00c72f5f4c54

Adaptive Vision. (o.J.). *Template Matching*. Verfügbar 24. April 2024 unter https://docs.adaptive-vision.com/4.7/studio/machine_vision_guide/TemplateMatching.html

Aishwarya, S. (2019). *Feature Engineering for Images: A Valuable Introduction to the HOG Feature Descriptor*. Verfügbar 10. November 2024 unter https://medium.com/analytics-vidhya/feature-engineering-for-images-a-valuable-introduction-to-the-hog-feature-descriptor-9f27512e5361

Anderson, B. (2023). *Histogramm*. Verfügbar 25. April 2024 unter https://statorials.org/de/histogramm/

Archana. (2024). *How to automate CAPTCHA?* Verfügbar 25. November 2024 unter https://www.guvi.io/blog/how-to-automate-captcha-in-selenium/

Baeldung. (2024). *What Does Learning Rate Warm-up Mean?* Verfügbar 1. Dezember 2024 unter https://www.baeldung.com/cs/learning-rate-warm-up

Brownlee, J. (2020). *Support Vector Machines for Machine Learning*. Verfügbar 26. April 2024 unter https://machinelearningmastery.com/support-vector-machines-for-machine-learning/

Conversation, T. (2024). *AI is changing the future of CAPTCHAs*. Verfügbar 24. November 2024 unter https://www.fastcompany.com/91167056/ai-changing-future-captcha

Deutschlandfunk. (2023). *Experten warnen vor „Risiko einer Vernichtung“*. Verfügbar 24. November 2024 unter https://www.deutschlandfunk.de/experten-warnen-vor-risiko-einer-vernichtung-110.html

Dhanush, K. (2023). *MAX POOLING*. Verfügbar 30. Oktober 2024 unter https://medium.com/@danushidk507/max-pooling-ef545993b6e4

EdjeElectronics. (2020). *generate_tfrecord.py*. Verfügbar 17. Oktober 2024 unter https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10/blob/master/generate_tfrecord.py

Fezan. (2019). *Understanding of Semantic Segmentation & How Segnet Model work to perform Semantic Segmentation*. Verfügbar 30. Oktober 2024 unter https://medium.

com/@fezancs/understanding-of-semantic-segmentation-how-segnet-model-work-to-perform-semantic-segmentation-5c426112e499

Goekcenaz, A. (2023). *Image Pyramids and Edges | Image Processing 8*. Verfügbar 25. April 2024 unter https://medium.com/@gokcenazakyol/image-pyramids-and-edges-image-processing-8-20e8016f484a

Gonçalves, D., Weber, V., Pistori, J., Gomes, R., Araujo, A., Pereira, M., Gonçalves, W., & Pistori, H. (2020). Carcass image segmentation using CNN-based methods. *Information Processing in Agriculture*, *8*. https://doi.org/10.1016/j.inpa.2020.11.004

IBM. (o.J.). *What is image Segmentation?* Verfügbar 30. April 2024 unter https://www.ibm.com/topics/image-segmentation

Imperva. (o.J.). *CAPTCHA*. Verfügbar 25. November 2024 unter https://www.imperva.com/learn/application-security/what-is-captcha/

Jorge, L. (2023). *Histogram of Oriented Gradient (HOG): A Comprehensive Guide to Object Detection and Recognition*. Verfügbar 5. Mai 2024 unter https://medium.com/@livajorge7/histogram-of-oriented-gradient-hog-a-comprehensive-guide-to-object-detection-and-recognition-48513a1346fb

Kang & Atul. (2019). *Understanding Image Gradients*. Verfügbar 25. April 2024 unter https://theailearner.com/2019/05/11/understanding-image-gradients/

Mazurov, M. (2022). *Google Recaptcha Image Dataset*. Verfügbar 2. Dezember 2024 unter https://www.kaggle.com/datasets/mikhailma/test-dataset

Morgunov, A. (2023). *How to Train Your Own Object Detector Using TensorFlow Object Detection API*. Verfügbar 3. November 2024 unter https://neptune.ai/blog/how-to-train-your-own-object-detector-using-tensorflow-object-detection-api

Musienko, Y. (2022). *VOR- UND NACHTEILE DER NEURONALEN NETZWERKARCHITEKTUR*. Verfügbar 5. Mai 2024 unter https://merehead.com/de/blog/vor-nachteile-neuronalen-netzwerkarchitektur/

Nemutlu, D. (2022). *HOG Feature Descriptor*. Verfügbar 25. August 2024 unter https://medium.com/@dnemutlu/hog-feature-descriptor-263313c3b40d

Nielsen, M. (2015). *Neural Networks and Deep Learning*. Verfügbar 22. April 2024 unter https://books.google.ch/books?id=STDBswEACAAJ

Nikhil, B. (2024). *Neurons in Neural Networks*. Verfügbar 22. April 2024 unter https://www.baeldung.com/cs/neural-networks-neurons

Orbis, L. (2020). *Template Matching*. Verfügbar 13. Oktober 2024 unter https://www.youtube.com/watch?v=sbjoQt118sM&ab_channel=LearningOrbis

Rawat, S. (2022). *Advantages and Disadvantages of Neural Networks*. Verfügbar 5. Mai 2024 unter https://www.analyticssteps.com/blogs/advantages-and-disadvantages-neural-networks

ResearchGate. (o.J.). *Multi-Objective Process Optimization for Overpressure Reflow Soldering in Electronics Production - Scientific Figure on ResearchGate*. Verfügbar 22. September 2024 unter https://www.researchgate.net/figure/Simplified-model-of-a-biological-neuron-130-Mathematical-modelling-of-neural-network_fig15_348487699

Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Sci*, *6*(12), 310–316. https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf

Shiv, V. (2020). *The Perfect Fit for a DNN*. Verfügbar 7. November 2024 unter https://medium.com/analytics-vidhya/the-perfect-fit-for-a-dnn-596954c9ea39

Starmer, J. (2020). *Support Vector Machines Part 1 (of 3): Main Ideas!!!* Verfügbar 27. August 2024 unter https://www.youtube.com/watch?v=efR1C6CvhmE&t=743s

Stecanella, B. (2017). *Support Vector Machines (SVM) Algorithm Explained*. Verfügbar 26. April 2024 unter https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

TensorFlow. (o.J.). *Semantic Segmentation with Model Garden*. Verfügbar 3. November 2024 unter https://www.tensorflow.org/tfmodels/vision/semantic_segmentation

Terayama, M., Shin, J., & Chang, W.-D. (2009). Object Detection using Histogram of Oriented Gradients.

Thatcher, J. (2009). *CAPTCHAs, CAPTCHAs everywhere*. Verfügbar 19. November 2024 unter https://www.jimthatcher.com/captchas.htm

Tyagi, M. (2021). *HOG (Histogram of Oriented Gradients): An Overview*. Verfügbar 25. April 2024 unter https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f

Wan, X., Johari, J., & Ruslan, F. A. (2024). Adaptive CAPTCHA: A CRNN-Based Text CAPTCHA Solver with Adaptive Fusion Filter Networks. *Applied Sciences*, *14*(12). https://doi.org/10.3390/app14125016

Wikipedia. (2023). *Histogram of oriented gradients*. Verfügbar 5. Mai 2024 unter https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients

Zaccone, G., & Rezaul, K. (2018). *Deep Learning with TensorFlow* (2. Auflage). Packt Publishing.

## 7.2 Abbildungsverzeichnis

## 7.3 Tabellenverzeichnis

# Anhang

## Verwendete Dokumente und Code-Dateien

Da die Code Skripts, die in dieser Arbeit verwendet wurden sehr lang sind, verzichte ich darauf, alle in diesem Dokument aufzuführen. Dasselbe gilt für die Auswertungstabellen, da sie teilweise aus über 20 Reihen und 5000 Zeilen bestehen. Es wurden aber alle in der Arbeit verwendeten Dateien in einem GitHub-Repository zusammengestellt und sind unter folgendem Link verfügbar: https://github.com/Zeptics/Maturarbeit_Jan_K.

Im GitHub-Repository befindet sich eine *README.md*-Datei, in der die Struktur von dem Repository erklärt wird.

## Verwendung von *KI-/LMM*-Tools

Während dem Programmieren wurde gelegentlich die Unterstützung von ChatGPT in Anspruch genommen. Den Grossteil der Programmierarbeit übernahm ich aber selbst. ChatGPT ist das einzige *KI*-Tool, welches während dieser Arbeit verwendet wurde und dies im Zeitraum von März bis November 2024. Die Modelle, welche in diesem Zeitraum zur Verfügung standen und verwendet wurden sind: *GPT-3.5 Turbo*, *GPT-4o* und *GPT-4o mini*. Es handelt sich dabei bei allen Modellen um Versionen von *ChatGPT* von *OpenAI* und waren oder sind verfügbar unter: https://chatgpt.com/.

**Vorgehensweise beim Gebrauch von ChatGPT:**
Grundsätzlich habe ich ChatGPT für diese Punkte verwendet:

- Generieren von Anfangspunkten, auf welchen ich das restliche Programm aufbauen konnte.
- Fehleranalyse
- Kleinere Anfragen, wenn ich nicht wusste, wie ich etwas implementieren kann.

Unten sind meine verwendeten Prompts angegeben. In den Chatverläufen ging es oft mehrere Male hin und her, bis endlich ein funktionierender Codeblock entstand. Zudem fragte ich teilweise nach, was der Code genau macht, damit ich verstehe, was ChatGPT generierte.

**Prompts:**

HOG:

1. how to build an object detection model with histogram of oriented gradients
2. I have an image data set and I want to do some object detection with hogs and svms. How do I start
3. I want to do it with open cv. In what format should my bounding boxes be?
4. give me an example code in python of how I would extract the hog for every image using open cv
5. In what format and where are the hog features saved?

6. what is this?
7. how do I install pickle in python 3.10
8. In this code you said you were going to save the hog features using pickle. Where are they saved to exactly
9. Now I extracted the hog features and found the saved file. Can a model as this even do proper detection or can It only do classification?
10. I have rather complex images in which I want to detect crosswalks. I have bounding boxes around the crosswalks. can I use hog and svm combination to train this
11. Obviously not all crosswalks have the same height and with. So how would I convert them (stretching or cutting them) and can you write a python script for it?
12. is the x, y the top left corner of the image or the center
13. change the following script to save the image to an ouput dir: [*Script*]
14. how to do a for loop iterating over every line in a txt doc
15. If I have a string ïmage.png", how can i strip the .png and replace it with a .txt
16. what does this do: if __name__ == "__main__:
17. You said I would have to create the hog features for both crosswalks, and non crosswalks. I now did it for the crosswalks. I have one big pickle file containing all of the hog features for the crosswalks. How are they labelled / is this file formatted?
18. I did it using the following code. It is a combination of my own work and stuff you provided. How is it formatted here? [*Script*]
19. where exactly in my code should the provided snipped be
20. Is there a way to get parts of the images that do not contain crosswalks (everything that isn't inside of a bounding box)? (with a python script)
21. I fixed some stuff: [*Script*]
22. I don't quite get what this achieves. It is only making the crosswalks black now, but how does this help with the problem of non crosswalk images? don't these black boxes disturb the hog features?
23. Cant you make a script that extracts like 3 crops, scaled to 64x64 pixels for every image. And by checking the bounding boxes, make sure those 3 crops do not contain any crosswalks
24. It works great, except that in some images there are no 64x64 spaces without crosswalks. So please change the script to look for 32x32 spaces after 20 failed attempts of finding one for 64x64. Then again after failing 20 times try it with 16x16. If that fails too after 20 attempts, just skip the image.

Neuronale Netzwerke:

*Chat 1:*

1. I trained a neural net model from tensorflow model zoo. I have a checkpoint directory now and would like to get the predictions. I think that you first have to save the model using this command:

   [*Script*]

   but I don't know how to continue. can you please help me

2. How can I run cmd promts from within a python script
3. When I do that, in what directory will the cmd be in?
4. Okay thanks.
5. Whenever I train a model from tensorflow model zoo, it only keeps the files for the last 5 checkpoints. Can I somehow keep all of them?
6. if I take the manual approach, will I have to run the file simultaneously to the training process?

*Chat 2:*

1. how to train a model on tensorflow with your gpu?
2. how can I make sure tensorflow uses my gpu?
3. How do I install tensorflow with gpu support?
4. How do I check if my GPU is being recognized by TensorFlow?
5. I want to install tensorflow on a virtual environment. How do I do that?
6. This is my pipeline.config file:

   [*Script*]
7. how do I activate the virtual environment in windows after installation?
8. I have installed the virtual environment. How do I activate it now?
9. I get this error when trying to run my model, how can I fix it?

   [*Error Message*]
10. This is my pipeline.config file:

    [*Script*]
11. I get this error with my adjusted pipeline.config file:

    [*Error Message*]

Segmentierung:

*Chat 1:*

1. I have a dataset of images and segmentation annotations. The annotations are in the labelme format. I want to create trimap images for these annotations. How can I do this? The trimaps have color values of 1, 2, 3 on a scale from 0 to 255.
2. Can you write me a script that opens a png and checks the shape of that png?
3. I also want to see the amount of color channels.
4. With which package do you use .shape()?

*Chat 2:*

1. I have a bunch of trimap pngs. when i try to open them it doesnt work however. How can i view them?
2. I get this error: [*Error Message*]

CAPTCHA Testprogramm für Menschen:

1. I want to write a program which allows the user to solve captchas. I have input images. Now in the program, it should open the image, then depending on what part of the image the user is clicking, it should put a white transparent square on that part (showing the user that he selected that part). The part should obviously be one of the 16 squares in the 4x4 grid layout. If the user clicks on a selected part of the image again, it should remove the white overlay and deselect that part. Could you write me a python script that does just that
2. Traceback (most recent call last): [*Error Message*]
3. I added a button to the window with pack(). How can I change the size of that button as well as the text size?
4. if i use pack, it always puts the new object at the bottom. Is there a way to replace one instead of packing it new
5. I upgraded the script slightly so that i can skip through images. now the selection part on the first image works perfectly, however the selection part doesn't work for the second image and further. What is wrong with the script: [*Script*]
6. I upgraded my program again. I added that it displays grid lines over the image. It always shows them except for when I am first launching the program (only before either clicking next or on one of the fields on the image. How can I solve this? [*Script*]
7. I tried that as well but for some reason it didn't work
8. how to track the time that the program has been running for?