





# Erkennung von Besucherströmen Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühlingssemester 2020


| | |
|---|---|
| Autoren: | Simon Styger, Daniel Bucher |
| Betreuer: | Prof. Dr. Luc Bläser |
| Projektpartner: | INS, Noser Engineering AG, Zoo Zürich |
| Experte: | Dr. Janos Zatonyi |
| Gegenleser: | Silvan Gehrig |

# 1 Abstract

Diese Arbeit ist in Zusammenarbeit mit der Firma Noser Engineering AG und dem Zoo Zürich entstanden. Der Zoo Zürich hat das Problem, dass seine Restaurants und Verpflegungsstände unterschiedlich stark frequentiert sind. Er möchte die Möglichkeit haben, seine Besucher auf weniger ausgelastete Verpflegungsmöglichkeiten umzuleiten. Ziel dieser Arbeit war es, mit Hilfe von Bildern einer handelsüblichen Kamera die Besucher der Restaurants zu zählen. Im Rahmen dieser Bachelorarbeit entstand so ein Prototyp, mit welchem Videodateien und Kamerastreams verarbeitet und die Informationen in einem einfachen Webfrontend visualisiert werden können.

Als Erstes wurde die Situation im Zoo analysiert und zwei verschiedene Ansätze bestimmt. Das Zählen am Eingang mit Hilfe einer virtuellen Schranke eignet sich dabei besser als das Zählen von sitzenden Personen. Aufgrund dieser Erkenntnisse wurde ein Prototyp unter der Verwendung von Deep Learning basierten Object Detection und Tracking Algorithmen entwickelt. Besonderes Augenmerk wurde dabei auf die Wahl des Detection Algorithmus gelegt, welcher in einem separaten Benchmarking bestimmt wurde.

Ergebnis: Die Zuverlässigkeit des Prototyps konnte aufgrund der COVID-19 Pandemie nicht im Zürich Zoo evaluiert werden. Aus diesem Grund haben wir den SAIVT Datensatz verwendet, welche diverse Aufnahmen von Personen bei Eingängen und auf Wegen beinhaltet. Wir haben unseren Prototypen auf Teilsequenzen von fünf Videos optimiert und auf fünf anderen Videos des Datensatzes evaluiert. Unser selbstdefiniertes Ziel war es, mindestens eine Genauigkeit von 80% zu erreichen. Mit einem Endergebnis von 90.72% konnten wir diesen Wert übertreffen.

# Inhaltsverzeichnis

# 2 Einleitung

Diese Arbeit ist in Zusammenarbeit mit der Firma Noser Engineering AG und dem Zoo Zürich entstanden. Der Zoo betreibt mehrere Restaurants und diverse Verpflegungsstände. Zwei Restaurants sind dabei jeweils zur Mittagszeit sehr beliebt und stark frequentiert. Andere Verpflegungsmöglichkeiten sind dagegen weniger ausgelastet. Aus diesem Grund möchte der Zoo bestimmen können, wie viele Besucher sich in den jeweiligen Restaurants und auf den Sitzterrassen aufhalten. Anhand dieser Daten könnten Besucher auf andere Restaurants umgeleitet werden. Der Zoo erhofft sich so, die Auslastung besser verteilen zu können.

## 2.1 Hintergrund

Die Information, wie viele Personen sich in einem Raum oder einem Aussenbereich befinden, ist für viele Firmen von Relevanz. So werden solche Technologien beim Flughafen, in Restaurants, im Detailhandel oder im öffentlichen Verkehr eingesetzt. In Freizeitparks werden die Personen beispielsweise mittels Drehkreuzen gezählt, um den Personenandrang auf die Bahnen zu steuern. Drehkreuze liefern eine exakte Personenanzahl. Für unseren Use Case sind sie jedoch weniger geeignet, da diese die Besuchererfahrung beeinträchtigen. Ein alternativer Ansatz wären Lichtschranken. Eine Zählung wird dabei mit jedem Unterbruch des Lichtsignals ausgelöst. Dies funktioniert jedoch nicht zuverlässig, wenn mehrere Personen gleichzeitig die Schranke passieren. Aus diesem Grund sind optische Verfahren mit Kameras beliebt. Diese erzielen eine hohe Genauigkeit (wenige False Positives und False Negatives) trotz der heuristischen Herangehensweise und schränken die Besucher nicht ein.

Für das Problem des visuellen Personenzählens gibt es zwei populäre Ansätze in der Literatur. Region of Interest (ROI) counting bezeichnet das Schätzen der Personenanzahl in einer bestimmten Region eines Bildes zu einer bestimmten Zeit. Line of Interest (LOI) counting dahingegen zählt die Anzahl der Personen, die eine virtuelle Schranke überschreiten.

### 2.1.1 Line of Interest counting

Für das LOI Counting und kommerzielle Produkte werden oft Kameras eingesetzt, die beim Eingang senkrecht auf den Boden gerichtet sind. Teilweise kommt dabei spezielle Hardware zum Einsatz. Tanner et al. [1] haben eine Time of Flight Sensor eingesetzt. Song et al. [2] und Hegner et al. [3] verwenden eine Kinect bzw. Prime-Sense Kamera, um Tiefenbilder zu erhalten. Das Vorgehen ist dabei oft dasselbe. Aufgrund der 3D Daten kann der Vordergrund gut vom Hintergrund getrennt werden. Die 3D Punktewolke, die daraus entsteht, wird anschliessend auf 2D Flächen projiziert. Mittels Anzahl Pixel pro Region und der Höhe der Pixel können Features wie Köpfe und Schultern ausfindig gemacht werden. Anhand der Features werden die Personen detektiert. Ein 2D Punkt pro Person erlaubt das Tracking der Personen, mit dem die Laufrichtung der Personen bestimmt wird. Für unsere Arbeit sind diese Ansätze weniger geeignet, da wir auf eine aufwändige Kalibration verzichten möchten und von unseren Industriepartnern explizit eine Lösung für Monokameras gewünscht wurde. Weiter sind Monokameras um einiges günstiger.

Eine Alternative ist es die Veränderung zwischen den Frames miteinzubeziehen. Barandiaran et al. [4] extrahieren die sich bewegenden Regionen mittels Image Differencing. Die Bewegungsrichtung wird anschliessend mittels Optical Flow rund um die Linie festgestellt. Mit Hilfe von mehreren Linien definieren sie eine Counting Zone, um eine robustere Zählung zu erhalten. Kim et al. [5] schlagen ein ähnliches, flowbasiertes System vor, bei dem die Kamera jedoch auch in einem nicht senkrechten Winkel positioniert werden kann. Um mit vielen Leuten umzugehen, wird die Zählung mit den Pixel des Vordergrunds gewichtet. Der Gewichtungsfaktor wird mit Hilfe einer Simulation angelernt.

Deep Learning hat in den letzten Jahren eine regelrechte Revolution in Gang gebracht. So setzen unter anderem Convolutional Neuronal Networks (CNNs) neue Massstäbe im Bereich Computer Vision. Es erstaunt somit nicht, dass es Arbeiten gibt, die versuchen die Schrankenzählung mit neuronalen Netzwerken direkt zu lösen. Cao et al. [6] verwenden ein Framework von mehreren solchen Netzwerken, die Bilder und Flow Informationen als zeitlicher

Ausschnitt (engl. temporal slice) verarbeiten. Denman et al. [7] vergleichen mehrere CNN Architekturen miteinander. Als Input verarbeiten sie dabei Optical Flow und Graufstufen Aufnahmen in räumlich-zeitlicher Darstellung (engl. spatio-temporal representation).

### 2.1.2 Region of Interest counting

Für das ROI counting ist dedizierte Hardware weniger üblich. Hegner et al. [3] bieten mit ihrer 3D Lösung die Möglichkeit mit Polygonen Regionen zu definieren und Personen darin zu zählen. Für ganze Räume oder Aussenbereiche wären in diesem Fall jedoch sehr viele Kameras notwendig.

Der Bewegungsansatz wird auch für ROI Zählungen aufgegriffen. Anstelle dass die Bewegungen über eine Linie analysiert werden, wird ein Ausschnitt einer Aufnahme oder ein komplettes Bild untersucht. Chan et. al [8] teilen die Aufnahme in Segmente von gleichartigen Bewegungen auf. Anhand von beispielsweise Kanten, Grössen und Formen der Segmente werden Features generiert. Mit Hilfe eines Gaussischen Prozesses wird die Zählung durchgeführt.
Eine Alternative sind dichtebasierte Ideen. Lempitsky und Zisserman [9] berechnen für jeden Pixel die Dichte. Die Werte des Pixels werden dabei mit einem trainierten Parameter multipliziert. Das Integral dieser Dichtefunktionen für einen bestimmten Bereich ergibt die Anzahl Personen. Anstelle von Bounding Boxes werden die Personen mit einem Punkt markiert.

Auch beim ROI Counting geht der Trend in Richtung Deep Learning. Chattopadhyay et al. [10] vergleichen mehrere neuronale Netzwerkarchitekturen miteinander. Sie schlagen dabei eine Lösung inspiriert von der menschlichen Simultanerfassung (engl. subitizing) vor. Menschen besitzen die Fähigkeit eine kleine Menge an Objekten (in der Regel bis zu 4) bestimmen zu können, ohne diese Zählen zu müssen. Ein grösseres Bild wird nun in kleinere Ausschnitte aufgeteilt, auf denen dies möglich ist.

## 2.2 Zielsetzung und Überblick

Aus Zeitgründen ist die Entwicklung von eigenen Bildverarbeitungsalgorithmen oder von neuronalen Netzwerkarchitekturen nicht möglich. Somit untersuchen wir in dieser Arbeit, wie wir bestehende Deep Learning Algorithmen und Modelle kombinieren können, um die Anzahl Restaurantbesucher im Zoo zu bestimmen. Dafür bieten sich Object Detection und Tracking Algorithmen an. Es gibt einige Open Source Implementationen und vortrainierte Modelle, die zur Verfügung stehen. Von unserem Industriepartner Noser Engineering AG wurde zudem die Microsoft Computer Vision API vorgeschlagen. [11]

In einem ersten Schritt legen wir in Kapitel 3 zwei verschiedene Lösungsansätze fest und untersuchen deren Vor- und Nachteile vor Ort. In Kapitel 5 implementieren wir den ermittelten Ansatz als Prototyp. Nach einigen Performance- und Genauigkeitsoptimierungen in Kapitel 6 und Abschnitt 7.3 überprüfen wir den Prototyp in Abschnitt 7.4 auf seine Genauigkeit. Unsere Industriepartner wünschen eine Aussage entsprechend einem Ampelschema. Um das Ergebnis messbar zu machen, definieren wir eine 80% Genauigkeit, die wir erreichen möchten. Ursprünglich ist die Evaluation im Zoo geplant gewesen. Aufgrund der COVID-19 Pandemie musste der Zoo Zürich geschlossen werden. Eine Evaluation vor Ort war somit nicht möglich. Aus diesem Grund definieren wir einen geeigneten Datensatz, um unser Verfahren möglichst aussagekräftig evaluieren zu können.

## 2.3 Klassifizierung, Detection und Recognition

Einige Begriffe wie Object Detection verwenden wir in dieser Arbeit häufig. Diese sind nicht immer klar oder werden falsch verwendet, deshalb gehen wir in diesem Abschnitt darauf ein.

Wenn man ein Bild erhält, dieses begutachtet und einer Kategorie zuordnet, spricht man von einer Klassifizierung. Dies ist in der Abbildung 2.1 links dargestellt. Man gibt dabei meistens in Prozent an, wie sicher man sich bei der Wahl ist. Neben der Klassifizierung kann man auch noch die Position des Objektes innerhalb des Bildes feststellen. Dies wird Lokalisierung genannt. Wenn wir nun mehrere Objekte auf einem Bild klassifizieren und lokalisieren möchten, wie in der unten aufgeführten Abbildung rechts ersichtlich, spricht man von einem Object Detection Problem. Die Objekte können dabei auch unterschiedlichen Kategorien angehören. Ein Gesicht (engl. Face) oder eine Person ist nun auch einfach eine Objektkategorie. [12]


Abbildung 2.1: Begriffserklärung

Detection wird oft mit Recognition verwechselt. Recognition bedeutet jedoch, dass ein Objekt wiedererkannt wird. Wie in Abbildung 2.2 dargestellt, wird bei Face Recognition ein Bild mit einer internen Datenbank abgeglichen. Die Datenbank kann dabei viele unterschiedliche Personen beinhalten, was den Abgleich erschwert. Als Output werden die erkannten Personen mit einer Wahrscheinlichkeit ausgegeben. Das Identifizieren von Personen wird für eine Zählung aber nicht benötigt. [12]

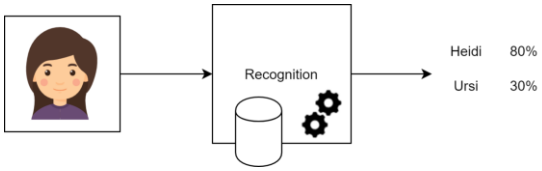


Abbildung 2.2: Recognition System

# 3 Ansätze

In diesem Abschnitt untersuchen wir nun konzeptionell, wie sich Personen im Zoo mit Hilfe der Ansätze aus der Literatur zählen lassen. Wir haben dafür in der Startphase unserer Bachelorarbeit mit allen involvierten Personen diskutiert und zwei Möglichkeiten definiert. Angelehnt an ROI wäre es eine Möglichkeit, Personen in einem bestimmten Bereich, bspw. im Essbereich der Restaurants, zu zählen. Eine Zählung an den Eingängen wäre ebenfalls möglich und orientiert sich somit am LOI-Ansatz. Beide Ansätze führen zur Verwendung von unterschiedlichen Technologien und haben Vor- sowie auch Nachteile.

Ziel dieser Untersuchung ist es, den vielversprechendsten Ansatz zu ermitteln, um diesen danach implementieren zu können. Um diese Untersuchung durchzuführen, wurden erste Testaufnahmen im Zoo gemacht. Damit wurden anschliessend Experimente durchgeführt, um den besten Ansatz für unseren Use Case zu finden.

## 3.1 Zählen im Essbereich

Die Idee des ersten Ansatzes besteht darin, die Personen innerhalb eines bestimmten Innen- oder Aussenbereichs zu zählen. Dabei werden fortlaufend Livebilder von einer oder mehreren Kameras ausgewertet und damit die aktuellen Besucherzahlen bestimmt. Im Optimalfall benötigt man für einen Raum eine Kamera, wie es in Abbildung 3.1 der Fall ist. Dieser Ansatz basiert auf der ROI-Idee aus der Literatur, wobei das Sichtfeld der Kamera die ROI definiert.

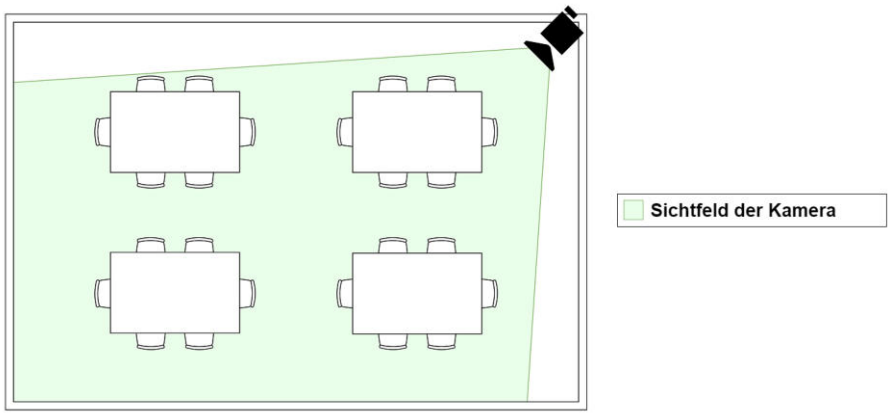


Abbildung 3.1: Abdeckung durch eine Kamera

Damit dieser Ansatz zuverlässige Daten liefert, muss nicht jedes einzelne Frame des Videostreams analysiert werden. Wenn man einige Frames pro Minute analysiert und aus den einzelnen Resultaten den Median berechnet, kann bereits eine Aussage über die Anzahl Personen getroffen werden. Somit muss die Auswertung nicht konstant erfolgen.

Durch die tiefere Anzahl Ausführungen, spielt die Ausführungszeit bei der Auswahl der verwendeten Technologien keine wichtige Rolle. Entscheidend für die Auswahl ist, dass der Algorithmus eine möglichst hohe Zuverlässigkeit aufweist.

Ein weiterer Vorteil dieses Ansatzes besteht darin, dass immer die aktuelle Situation dargestellt wird. Das bedeutet, dass Fehler aus einer Auswertung die nächsten Auswertungen nicht negativ beeinflussen. Fehler summieren sich somit nicht auf.

### 3.1.1 Störfaktoren

Gegen den Ansatz sprechen diverse Störfaktoren. Häufig ist es nicht möglich, einen ganzen Raum mit nur einer Kamera abzudecken. Die Form, Länge und Breite des Raumes spielen eine primäre Rolle. Wie man in Abbildung

3.2 sehen kann, können komplexe Raumformen das Sichtfeld stark verdecken. Zudem kann es in einem Raum viele unterschiedliche Störfaktoren geben. Das können bspw. Säulen, grosse Pflanzen oder andere Objekte sein. In Abbildung 3.2 kann man erkennen, wie das Sichtfeld durch eine Pflanze beeinträchtigt und durch eine Säule komplett verdeckt wird.

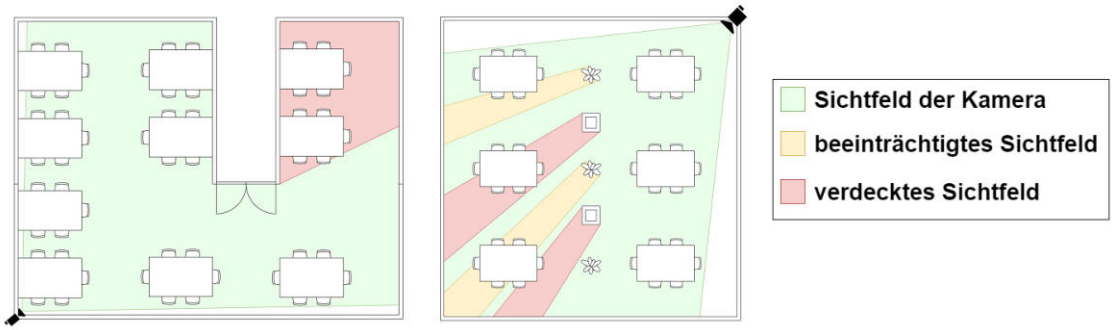


Abbildung 3.2: Die Kameras können nicht den ganzen Raum abdecken

Um die oben beschriebenen Störfaktoren zu eliminieren, können mehrere Kameras für einen Raum verwendet werden. Dadurch entsteht möglicherweise ein überlappendes Sichtfeld, was neue Herausforderungen mit sich bringt. In Abbildung 3.3 kann man diese Überlappung durch den dunkelgrünen Bereich erkennen. Wie man sehen kann wird Person A nur von Kamera 1 und Person B nur von Kamera 2 eingefangen. Dementsprechend werden beide Personen nur einmal gezählt. Aber was ist mit Person C? Sie befindet sich im Sichtfeld beider Kameras und wird möglicherweise doppelt erfasst.

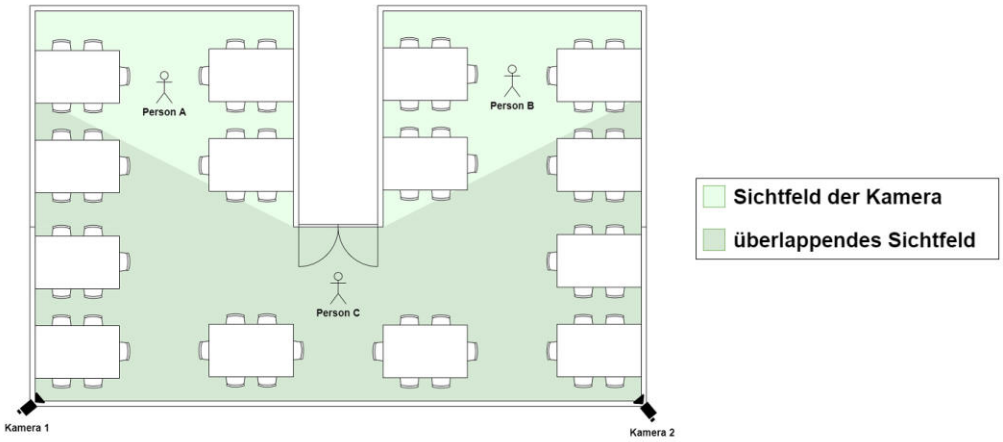


Abbildung 3.3: Zwei Kameras mit überlappenden Sichtfeldern

Im Optimalfall könnten die Kameras so positioniert werden, dass sich ihre Sichtfelder nicht überlappen. Dadurch wären doppelte Zählungen von Personen praktisch ausgeschlossen. Sollte dies allerdings nicht möglich sein, wäre zwingend eine zeitliche Synchronisierung der beiden Kameras nötig. Damit wird sicher gestellt, dass die analysierten Bilder der beiden Kameras zeitlich übereinstimmen. Damit durch Überlappung der Sichtfelder keine Mehrfachzählungen entstehen können, wäre eine aufwändige Kalibrierung wie bei den 3D-Kameras [1] [2] [3] notwendig.

Wir wollen einen Zähler implementieren, welcher in möglichst vielen Situationen zuverlässig funktioniert. Daher ist es wichtig, dass er auch ausserhalb eines Gebäudes verwendet werden kann. Draussen stehen in Restaurants häufig Sonnenschirme. Das sind ebenfalls Störfaktoren, welche das Sichtfeld beeinflussen. Wie man in Abbildung 3.4 erkennen kann, spielt die Höhe der Kameraposition hierbei eine entscheidende Rolle. Je höher die Kamera positioniert wird, desto mehr wird das Sichtfeld vom Sonnenschirm verdeckt.

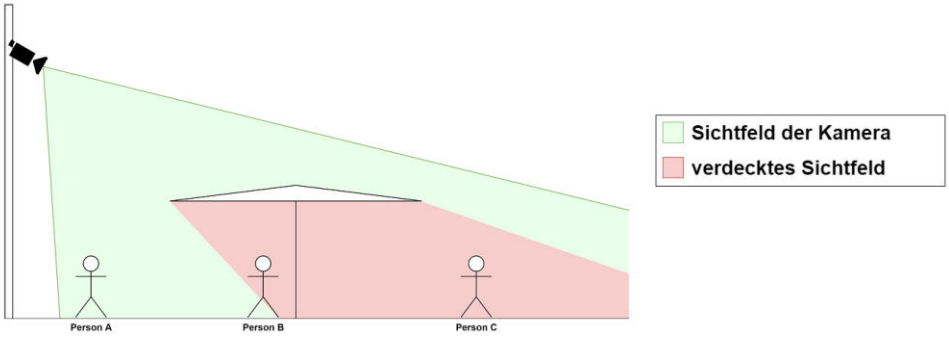


Abbildung 3.4: Beeinträchtigtes Sichtfeld durch einen Sonnenschirm

Dieses Problem lässt sich nicht einfach lösen, wenn man die Kamera tiefer positioniert. Denn sobald der Einfallswinkel zu klein wird, besteht das Risiko, dass das Sichtfeld von anderen, tiefer gelegenen Objekten wiederum beeinträchtigt wird. Wie in Abschnitt 3.1.1 bereits beschrieben, wäre das mit mehreren Kameras lösbar. Dadurch entsteht aber erneut das Problem von Mehrfachzählungen einer Person, wenn sie sich im Sichtfeld von zwei Kameras befindet.

Bei unserem Besuch im Zoo haben wir einige dieser Störfaktoren vor Ort identifiziert. Zum Beispiel besitzt das Pantanal eine sehr grosse Terasse, die auch verwinkelt ist. Dies führt zur Situation wie in Abbildung 3.3 beschrieben und zu einer zwingenden Verwendung von mehreren Kameras, deren Sichtfelder sich überlappen können. Weiter werden bei gutem Wetter Sonnenschirme verwendet, die wie vorhin beschrieben, das Sichtfeld einschränken können. Weiter haben wir identifiziert, dass die Räume der Mosoala und Pantanal Restaurants sehr gross sind, was ebenfalls eine Verwendung von mehreren Kameras erfordert.

### 3.1.2 Experimente

Trotz der Störfaktoren haben wir einige Experimente mit verschiedenen Detection Algorithmen durchgeführt. Die Resultate und Erkenntnisse werden in diesem Abschnitt erläutert.

Mit Hilfe von Detection Algorithmen können Objekte erkannt und lokalisiert werden. Dadurch wäre es möglich, die Zahl der anwesenden Personen zu bestimmen. Dabei wird jedes erkannte Objekt mit einer sogenannten Bounding Box umgeben, welche die Position in Rechteckform markiert. Die Anzahl dieser Bounding Boxen entspricht im Idealfall der Anzahl anwesender Personen. Um diesen Ansatz zu untersuchen, haben wir sowohl im Masoala, wie auch im Pantanal Restaurant Video-Aufnahmen durchgeführt. Die einzelnen Bilder wurden dann von den Algorithmen untersucht.

Als Erstes haben wir versucht mit der Computer Vision API von Microsoft Azure Objekte der Klasse Person zu detektieren. Dadurch dass die Personen mindestens 5% des gesamten Bildes ausmachen müssen und nicht zu nahe beieinander sein dürfen, liefert Azure sehr wenige Detections. Diese Beschränkungen lassen sich auch nicht konfigurieren. [13] Aufgrund der tiefen Detectionszahl und den Limitierungen von Azure, haben wir die Experimente mit dem Open Source Object Detector YOLO [14] erneut durchgeführt. YOLO lässt im Vergleich zu Azure eine Konfiguration zu.

Mit YOLO haben wir zwar einige Detections mehr erzielt, von der korrekten Anzahl sind wir aber auch damit noch weit entfernt. In Abbildung 3.5 ist ersichtlich, dass auch nahe Personen kaum erkannt werden. Das liegt daran, dass das verwendete Modell auf Fussgänger und nicht auf sitzende Personen trainiert ist. Auch andere Object Detection Algorithmen sind auf dem gleichen Datenset (COCO [15]) trainiert, weshalb auch mit diesen ähnliche Resultate zu erwarten sind.

Abbildung 3.5: Tiefe Anzahl von Detections beim ROI-Ansatz (aus Datenschutzgründen anonymisiert)

Anstelle der Kategorie Person haben wir mit RetinaFace [16] einen Face Detection Algorithmus ausprobiert. Mit diesem haben wir gleich viele Detections erhalten wie mit YOLO, jedoch sind drei davon nicht korrekt. Was weiter gegen den Einsatz eines Face Detection Algorithmus spricht, ist, dass die Gesichter nicht abgewandt zur Kamera sein dürfen. Somit wären mehrere Kameras notwendig, die wiederum synchronisiert werden müssten.

Die Resultate der einzelnen Algorithmen sind in Abbildung 3.6 ersichtlich. Wir haben dabei je fünf Bilder aus den beiden Restaurants genauer analysiert. Wie dabei ersichtlich ist, liegen alle drei Algorithmen weit unter den gezählten Resultaten. Azure erkennt 14, YOLO 44 und RetinaFace 41 der Personen über alle Bilder hinweg. Manuell haben wir 275 Personen gezählt.

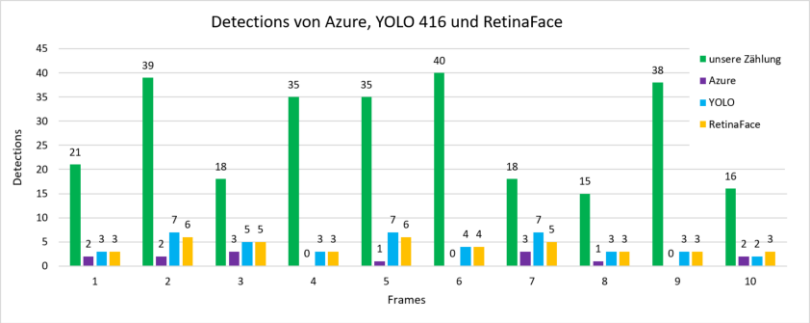


Abbildung 3.6: Resultate von Azure, YOLO und RetinaFace auf 10 unterschiedlichen Bildern im Zoo

#### 3.1.2.1 Head Detection

Wie wir in den Experimenten festgestellt haben, gibt es Probleme für die Objektkategorien "Person" und "Gesicht". Für die Detection von Personen fehlt uns der Unterkörper und für Gesichter müssen die Personen in die richtige Richtung blicken. Die Köpfe der Personen sind jedoch meistens erkennbar, somit müssten wir eine Detection für Köpfe machen können.

Wir haben leider kein trainiertes Modell für diese Kategorie gefunden. Aus diesem Grund müsste man ein bestehendes Modell wie YOLO nachtrainieren. Dieses Vorgehen wird Transfer Learning genannt. Dabei wird der letzte Layer eines bereits bestehenden Netzwerks ausgetauscht. Mit Bildern von Köpfen oder Oberkörpern könnte dem Netz in diesem Fall ein neuer, letzter Layer antrainiert werden. Transfer Learning eignet sich gut, wenn ein Netz auf eine neue, aber ähnliche Anforderung angewendet werden soll. Dafür würden wir sehr viele Trainingsdaten in Form von Bildern benötigen. Weiter würde das annotieren von Bilder und das Nachtrainieren des Modells viel Zeit in Anspruch nehmen.

#### 3.1.2.2 Preprocessing

Um die schlechten Zahlen der Object Detection zu verbessern, haben wir mit Alexander Gassmann alternative Technolgien besprochen. Mit Hilfe von Optical Flow und Background Subtraction könnten die Algorithmen wie [6] [7] in Form eines Preprocessings unterstützt werden. Wir haben beide Varianten angetestet.

Beim Ansatz mit Optical Flow werden die örtlichen und zeitlichen Bewegungsvektoren aufsummiert. Daraus können Bewegungen in aufeinander folgenden Bildern erkannt und somit die Anwesenheit von Personen geschätzt werden. Bei diesem Ansatz könnten gehende Personen eine Herausforderung darstellen, da durch sie grosse Bewegungsvektoren entstehen.

Background Subtraction versucht nicht primär Objekte zu erkennen. Durch die Verwendung einer binären Maske wird erkannt, welche Pixel von einem zuvor festgelegten Zustand abweichen. Wenn man also im Preprocessing ein Bild des Bereichs ohne Personen festlegen würde, könnte man anhand dieses Vergleichs feststellen, wo sich etwas verändert hat.

Abbildung 3.7: Ergebnisse von Optical Flow und Background Subtraction

Wie man in Abbildung 3.7 sehen kann, lässt sich mit Optical Flow nur sehr schwer eine Aussage zur Auslastung eines Raumes machen. Von den über zehn Personen in diesem Raum wurde lediglich in einem Bereich eine deutliche Bewegung erkannt. Sitzende Personen konnten fast kaum erkannt werden. Dies liegt vor allem daran, dass sich sitzende Personen nicht so stark bewegen wie stehende Personen. Folglich haben sie, wie bereits beschrieben, nur sehr kleine Bewegungsvektoren. Darum ist es schwierig, sitzende Personen zuverlässig zu zählen. Dies führte uns zum Schluss, Optical Flow nicht weiter zu verfolgen, da die erhofften Verbesserungen nicht erreicht werden konnten.

Ebenfalls in Abbildung 3.7 erkennt man rechts das Ergebnis der Background Subtraction. Hier lassen sich etwas mehr Personen erkennen. Auch auf anderen Aufnahmen hatten wir ähnliche Ergebnisse. Somit könnte Background Subtraction für die Erkennung von sitzenden Personen durchaus unterstützend wirken. Wir haben die Idee des Preprocessing aber nicht weiterverfolgt, da die existierenden Algorithmen keine Background Subtraction Resultate verarbeiten können. Man müsste also die bestehenden Netzwerk Architekturen anpassen und ein eigenes Modell neu antrainieren. Beide Schritte sind viel aufwändiger als Transfer Learning.

## 3.2 Zählung am Eingang

Ein weiterer Ansatz um Personen zu zählen ist, mit Hilfe einer Kamera eine visuelle Schranke am Eingang des Bereichs zu programmieren. Mit Hilfe dieser Schranke können die ein- und austretenden Personen gezählt werden. Damit dies möglich ist, muss der Weg einer Person während der Zeit, in der sie im Sichtfeld der Kamera ist, verfolgt werden. Dieser Ansatz ist somit stark an die bereits beschriebene LOI Zählung aus der Literatur angelehnt. In Abbildung 3.8 sieht man wie die Zählung anhand einer LOI funktionieren würde.

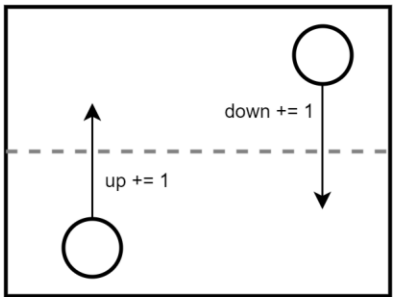


Abbildung 3.8: Bewegungen von Objekten über eine LOI

Dieser Ansatz ist sehr unabhängig vom Use Case. Das bedeutet, man kann damit nicht nur die Personen in einem Restaurant zählen, sondern überall wo sie einen Eingang oder Ausgang passieren.

Dabei filmen die Kameras nur die Eingänge zu einem Bereich und können optimal darauf ausgerichtet werden. Der Ansatz ist somit unabhängig von der Situation innerhalb des Bereichs, den man überwachen möchte. Das Sichtfeld am Eingang wird zudem durch weniger Störfaktoren beeinträchtigt. Ein weiterer Vorteil dieses Ansatzes besteht darin, dass durch die Verwendung von nur einer Kamera pro Eingang keine Überlappungen im Sichtfeld entstehen. Somit wird auch keine Kalibration benötigt.

### 3.2.1 Nachteile

Wir erwarten bei diesem Ansatz eine hohe Performance Auslastung, da die Videoübertragungen der einzelnen Eingänge in Echtzeit verarbeitet werden müssen. Die Personen müssen in der kurzen Zeit, in welcher sie sich im Sichtfeld der Kamera befinden, mindestens einmal erkannt werden. Somit ist eine deutlich höhere Ausführungs-Rate des Analyse-Algorithmus nötig.

Durch die erhöhte Ausführungs-Rate spielt der zeitliche Aspekt des Algorithmus ebenfalls eine Rolle. Bei der Auswahl des Algorithmus ist man somit etwas eingschränkt, da nicht jedes Modell eine derart hohe Ausführungs-Rate ermöglicht.

Dazu kommt, dass das Hauptproblem des ROI-Ansatzes auch mit diesem Ansatz nicht komplett eliminiert werden kann. Verdeckungen von Personen sind nach wie vor möglich, wenn sich mehrere Personen dicht zusammen oder gar hintereinander durch das Sichtfeld bewegen.

Ein weiterer Nachteil dieses Ansatzes besteht darin, dass sich Fehler über die Zeit aufsummieren können. Das bedeutet, dass falsche Zählungen über den gesamten Zeitraum bestehen bleiben und so das Resultat immer mehr verfälschen. Wir vermuten allerdings, dass diese Fehler in beide Richtungen über die Schranke auftreten und sich so mehr oder weniger ausgleichen sollten.

### 3.2.2 Experimente

Auch für diesen Ansatz haben wir mit dem Object Detector YOLO experimentiert. Wie man in Abbildung 3.9 erkennen kann, funktioniert dies sehr gut. Sämtlichen Personen wurden einer Bounding Box zugeordnet. Die Probleme die wir beim ROI-Ansatz erkennen konnten, existieren in diesem Ansatz praktisch nicht. Einerseits sind die Personen auf den Bildern deutlich grösser, da sie an der Kamera vorbei laufen müssen. Andererseits sind auf dem Bild fast ausschliesslich Fussgänger ersichtlich. Das entspricht genau dem, worauf die Modelle ausgelegt sind.

Die Detection Resultate sind sehr überzeugend. Darum wurde in einem nächsten Schritt ein detection basierter Tracking-Algorithmus eingesetzt, um die Personen durch das Bild zu verfolgen. Mit Hilfe dieses Algorithmus war es technisch sehr einfach möglich, die ein- und austretenden Personen zu zählen. Die bereits existierenden Modelle und Algorithmen, welche als Open Source zur Verfügung stehen, lieferten gute Resultate.

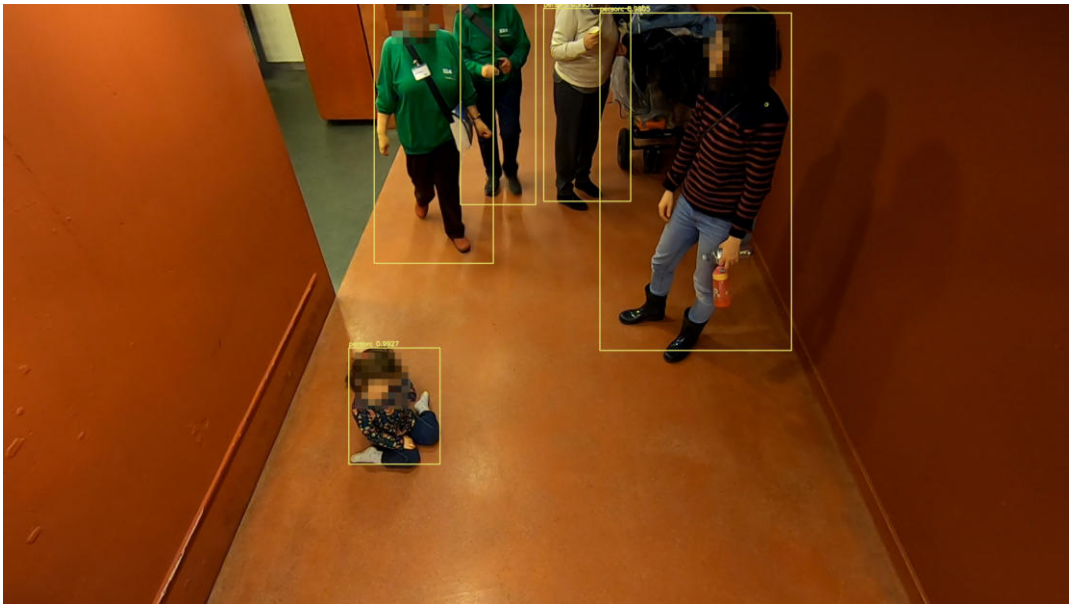

Abbildung 3.9: Hohe Anzahl von Detections beim LOI-Ansatz (aus Datenschutzgründen anonymisiert)

## 3.3 Entscheid

Die oben beschriebenen Ansätze wurden konzeptionell überprüft und im Zoo ersten Experimenten unterzogen. Die Störfaktoren beim ROI-basierten Ansatz sind ein grosser Nachteil, welche man nicht ohne weiteres lösen kann. Zudem gibt es keine Möglichkeit die Verwendung von mehreren Kameras technisch einfach umzusetzen. Dazu kommt die Zuverlässigkeit, welche bei diesem Ansatz dringend verbessert werden müsste. Auch mit der Verwendung von verschiedenen Technologien konnte nicht annähernd die Anzahl an Personen ermittelt werden, wie tatsächlich anwesend waren.

Der LOI-basierte Ansatz sieht hingegen sehr viel versprechend aus. Der eingesetzte Detection Algorithmus entdeckt grundsätzlich alle Personen und liefert somit deutlich zuverlässigere Resultate als beim ersten Ansatz. Zudem konnten mit Hilfe eines Tracking-Algorithmus bereits gute Resultate erzielt werden. Die Verdeckungen, welche beim ersten Ansatz ein grosser Nachteil sind, können hier mit einer guten Positionierung der Kamera verringert werden. Dafür nimmt man in Kauf, dass man entsprechend höhere Performance Requirements haben wird.

# 4 Implementation

In diesem Kapitel erklären wir die Implementation des LOI Counters. Für die Entscheidung, ob eine Person eine zuvor definierte Schranke übertreten hat, sind mehrere Algorithmen involviert.

In einem ersten Schritt müssen Personen auf Kamerabildern, sogenannten Frames, erkannt werden können. Für dieses Problem wird ein Object Detector benötigt. Dieser liefert nach jeder Ausführung eine Liste von Bounding Boxen und Kategorisierungen zurück. Anhand dieser Bounding Boxen sind wir in der Lage, die Personen auf einem Bild zu lokalisieren.

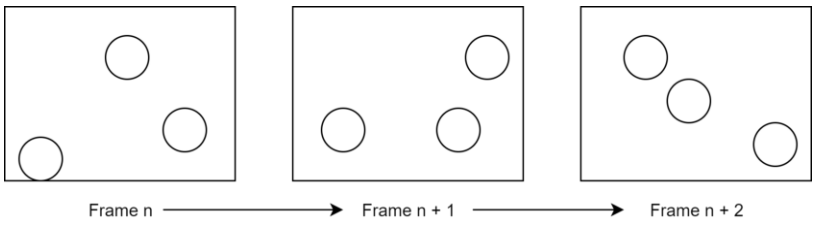


Abbildung 4.1: Lokalisierte Objekte in nacheinander folgenden Frames

In Abbildung 4.1 kann man erkennen, dass die Lokalisierung von Objekten nicht ausreichend ist, um festzustellen, welche Objekte sich wohin bewegen. Der Grund dafür ist, dass man mit einem Detection Algorithmus keine Zusammenhänge zwischen den Objekten in den einzelnen Frames herstellen kann. Ohne diese Zusammenhänge kann nicht bestimmt werden, in welche Richtung sich jemand bewegt. Um diese Verfolgung von Objekten zu ermöglichen, wird in einem zweiten Schritt ein Tracking Algorithmus benötigt. In Abbildung 4.2 zeigen wir, dass man anhand dieser Verfolgbarkeit feststellen kann, wohin sich Objekte seit dem letzten Frame bewegt haben.

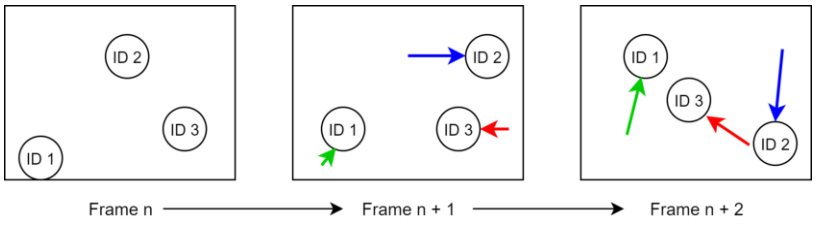


Abbildung 4.2: Bewegungen von Objekten mit Identifikation

Im einem dritten Schritt geht es nun darum, festzustellen, ob ein Objekt seit dem letzten Frame eine Schranke überquert hat. Dieser Schritt ist in Abbildung 4.3 visualisiert. Dabei wird für jedes Objekt bestimmt, ob sich der Mittelpunkt seiner Bounding Box im vorherigen Frame links oder rechts der Schranke befand. Wenn sich der Mittelpunkt der Bounding Box im neuen Frame auf der anderen Seite der Schranke befindet, so hat die Person die Schranke in dieser Zeit überquert. Dies lässt sich sehr einfach in einem Algorithmus umsetzen, der in Abbildung 4.4 ersichtlich ist.

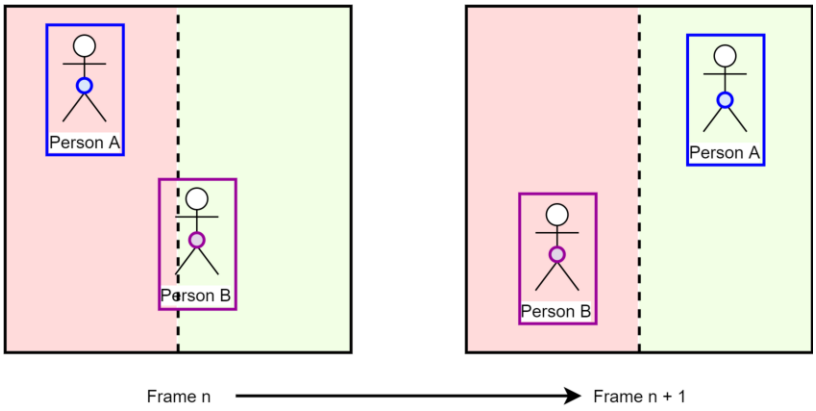


Abbildung 4.3: Konzept des Zählmechanismus

```python
def count(tracking_object, line_position):
    new_centre = tracking_object.calculate_centre()
    difference = 0
    was_left = tracking_object.last_centre < line_position
    was_right = tracking_object.last_centre >= line_position
    is_left = new_centre < line_position
    is_right = new_centre >= line_position
    if was_left and is_right:
        # Person went in
        difference = 1
    elif was_right and is_left:
        # Person went out
        difference = -1
    tracking_object.last_centre = new_centre
    return difference
```

Abbildung 4.4: Code des Zählmechanismus

## 4.1 Object Detection

Die erste Komponente unseres Prototypen besteht aus einem Deep Learning Modell für die Object Detection. Für die Anwendung von Deep Learning Algorithmen gibt es diverse Frameworks wie z.B. Tensorflow, PyTorch oder Caffe. Mit diesen lassen sich Netzwerkarchitekturen erstellen, Modelle trainieren und ausführen. Die Modelle besitzen dabei je nach Framework ein anderes Format, was den Wechsel zwischen Frameworks erschwert. Die populäre Computer Vision Library OpenCV besitzt seit Version 3.1 ein Deep Neural Network (DNN) Modul [17], mit welchem sich Modelle von diversen Frameworks einbinden und verwenden lassen. Wir haben dieses für unsere Implementation verwendet, da dies die Austauschbarkeit des Modells (auch Framework übergreifend) sicherstellt. Weiter reduzieren wir dadurch weitere Abhängigkeiten zu Third-Party Libaries.

In den nächsten Abschnitten werden die einzelnen Teilschritte betreffend Object Detection erläutert. Vor und nach der Verwendung des Modells sind Pre- bzw. Postprocessing Schritte notwendig. Die Auswahl des Modells treffen wir in Kapitel 5 nach einem Benchmarking.

### 4.1.1 Preprocessing und Forward Pass

Im Preprocessing geht es darum, den Input für die Object Detection vorzubereiten. Dafür muss das Frame mit Hilfe von OpenCV zu einem BLOB umgewandelt werden. BLOB steht für Binary Large Object. Im folgenden Code wird zuerst ein vier-dimensionales BLOB generiert, welches als Input für den Forward Pass verwendet wird. Diese vier Dimensionen beschreiben die im BLOB enthaltenen Werte: [18] [19]

- Die Anzahl der im BLOB enthaltenen Bilder
- Die Anzahl der BGR-Kanäle (B=blau, G=grün, R=rot)
- Die Höhe des BLOBs
- Die Breite des BLOBs

Der Forward Pass repräsentiert die effektive Object Detection, auch Inferenz genannt. Die Zeit welche für diesen Schritt benötigt wird, nennt man dementsprechend Inferenzzeit. Diese Ausführung generiert schlussendlich den Output, welcher die Liste der Detections beinhaltet.

```
blob = cv2.dnn.blobFromImage(frame, scale, size, swapRB=True, crop=False)
network.setInput(blob)
output = network.forward(layer_names)
```

### 4.1.2 Filterung

Ist die Object Detection abgeschlossen, muss das Resultat im Postprocessing noch weiter verarbeitet werden. In einem ersten Schritt wird eine Filterung durchgeführt. Diese Filterung wird benötigt, da der Object Detector einen sehr grosse Liste (über 10'000 Elemente) von potentiellen Objekten zurück gibt. Diese Liste wird dann anhand der Wahrscheinlichkeiten gefiltert. So werden nur diejenigen Objekte weiterverwendet, die mit einer bestimmten Wahrscheinlichkeit auch wirklich Objekte sind. Durch diese Filterung resultiert eine kleinere Liste von tatsächlichen Detections, welche weiter verarbeitet wird. In unserem Fall filtern wir die Objekte zusätzlich nach Objekten der Klasse Person, da die Objekte anderer Klassen nicht benötigt werden.

Die Filterung lässt sich mit Hilfe des Confidence Thresholds konfigurieren. Häufig wird dieser bei 0.50 festgesetzt. In Abschnitt 7.3.1 ermitteln wir den besten Wert für unseren Use Case.

### 4.1.3 Non Maxima Suppression (NMS)

Der zweite Schritt im Postprocessing nennt sich Non Maxima Suppression (NMS). Dieser Algorithmus untersucht die erkannten Objekte und filtert diejenigen raus, welche sich zu stark überlappen. Damit löst man das Problem, dass ein Objekt, wie links in Abbildung 4.5, mehrere Detections verursachen kann. Die NMS filtert anhand der Dimensionen und der Wahrscheinlichkeit die überlappenden Bounding Boxen raus. Das Ergebnis erkennt man im rechten Teil der Abbildung 4.5. [20] [12]


Abbildung 4.5: Visualisierung der NMS (Frame aus TownCentre Dataset [21])

Dabei verwendet die NMS die sogenannte Intersection over Union Metrik (IoU). Bei der IoU wird für zwei Flächen berechnet, wie gross die Überlappung im Vergleich zur Gesamtfläche ist. Dies ist in Abbildung 4.6 ersichtlich. Ist

dieser Wert höher als ein bestimmter Grenzwert, wird davon ausgegangen, dass beide Bounding Boxen zur selben Person gehören. Dabei wird die Bounding Box mit der tieferen Confidence aus der Liste entfernt. Dieser NMS Threshold kann ebenfalls konfiguriert werden. Ein zu kleiner Wert kann zu False Negatives führen, da nahestehende Personen durch die NMS als ein Objekt behandelt werden. Den Einfluss des Thresholds auf Verdeckungen untersuchen wir in Abschnitt 7.3.3. Auch legen wir in diesem Kapitel den besten Threshold für unseren Use Case fest. [22] [12]

$$\text{Intersection over Union}(IoU) = \frac{A \cap B}{A \cup B}$$

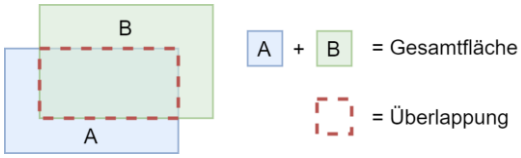


Abbildung 4.6: Intersection over Union

## 4.2 Tracking Komponente

Die Tracking Komponente ist für unseren Prototyp ebenfalls sehr wichtig, da sie für die Zusammenhänge zwischen den einzelnen Frames zuständig ist. Wie auch bei den Object Detectors gibt es eine Vielzahl an solchen Tracking Algorithmen, weshalb die Einordnung mit Hilfe von Kategorien hilfreich ist.

**Online/Offline Tracking** Online Tracking bezeichnet das sequentielle Abarbeiten der Frames, wohingegen Offline Tracking Frames in Form von Batches verarbeitet. Die Batchverarbeitung berücksichtigt, im Gegensatz zur sequentiellen Verarbeitung, auch die Informationen aus den zukünftigen Frames. Offline Tracking ist für unseren Use Case jedoch nicht sinnvoll. Die Batchauswertung der ganzen Aufnahme ist nicht möglich, da wir die Resultate laufend benötigen und nicht erst nach Betriebsende der Restaurants. Eine Aufteilung in mehrere Batches führt dazu, dass Zählungen verloren gehen können, wenn Personen über mehrere Batches hinweg die LOI überqueren. [23]

**Detection-Based Tracking (DBT), Detection-Free Tracking (DFT)** Trackers können auch anhand der Objektinitialisierung eingeteilt werden. DBT bezeichnet die Objekterkennung mit Hilfe eines Object Detectors, wohingegen bei DFT die Objekte manuell markiert werden müssen. Die manuelle Markierung hat den Vorteil, dass man keine Einschränkung bei den Objektkategorien hat und unabhängig von einem Detector ist. Für unsere Lösung ist DFT aber nicht praktikabel, da wir eine automatisierte Zählung erreichen möchten. [23]

**Single Object Tracking (SOT), Multiple Object Tracking (MOT)** SOT beschränkt sich auf das Tracking eines einzelnen Objektes und löst Herausforderungen wie Verdeckungen, Änderungen der Skalierung etc. MOT löst zwei weitere Probleme, nämlich das Merken der einzelnen Objekt-IDs und das Bestimmen der Anzahl Objekte. Da mehrere Personen gleichzeitig unsere Zähl-Schranke passieren können, sind wir auf die Verwendung eines MOT angewiesen. [23]

Zusammenfassend benötigen wir somit einen MOT Tracker, der einen Object Detector verwendet, sowie das sequentielle Abarbeiten der Frames als Online Tracker ermöglicht.

### 4.2.1 MOT Bestandteile

Das Ziel eines MOTs ist es mehrere Objekte zu erkennen (in unserem Fall mit Hilfe eines Object Detectors) und deren Identitäten in den fortlaufenden Frames wiederherzustellen. Dabei stellen sich zwei Fragen. Wie misst man die Übereinstimmung von Objekten zwischen den Frames und wie werden die Identitäten basierend auf diesen Informationen wiederhergestellt?

Die erste Frage befasst sich dabei mit der Modellierung der Objekte. Dafür gibt es verschiedene Ansätze, die kombiniert werden können: [23]

- Die visuelle Darstellung (engl. appearance model) kann z.B. mit Hilfe von Optical Flow, Punktwolken oder Tiefenbilder modelliert werden.
- Die Modellierung der Bewegung (engl. motion model) schätzt die potenziell nächste Position eines Objektes in den zukünftigen Frames und reduziert so den Suchraum.
- Das Interatkionsmodell (engl. interaction model) berücksichtigt, wie Objekte sich untereinander beeinflussen. Zum Beispiel passen Fussgänger ihre Laufwege, Laufgeschwindigkeit etc. aufgrund von anderen Passanten an.

Die zweite einleitende Frage behandelt ein Zuordnungsproblem der verschiedenen Informationen zu einer Identität. Dies wird oft mit Hilfe von Dynamic Programming oder statistischen Methoden gelöst.

### 4.2.2 Tracker in Libraries

Die einfachste Möglichkeit ein Tracker zu verwenden, wäre die Einbindung mit Hilfe einer Library. Aus diesem Grund haben wir zwei Libraries und ihre Tracker untersucht.

Die OpenCV Library bietet bereits einige Tracker wie GOTURN, MOSSE etc. als API [24] an. Bei diesen handelt sich um Single Object Tracker. OpenCV erweitert diese mit Hilfe einer MultiTracker Klasse, welche jedoch den SOT Tracker für jedes Objekt einzeln ohne Optimierungen ausführt. Dies führt dazu, dass die Auswertung umso langsamer wird, je mehr Objekte getrackt werden.

Experimente mit der Dlib Library [25], einer Library für diverse Machine Learning Anwendungen, und deren Correlation Tracker haben zum gleichen Ergebnis geführt. Auch dieser Algorithmus ist ein SOT Tracker und musste deshalb mit einer Liste zu einem MOT Tracker erweitert werden. Das Zuordnungsproblem haben wir so gelöst, indem wir die alten Mittelpunkte der Bounding Boxen mit den neuen anhand der euklidischen Distanz verglichen haben. Die kleinsten Abstände haben dabei dem gleichem Objekt entsprochen. Falls neue Detections keinen alten Detections zugeordnet werden konnten, wurden sie als neue Objekte registriert. Die Objekte wurden nach einer bestimmten Anzahl Frames ohne erneute Detection gelöscht. Für die Umsetzung sind wir dabei einem Tutorial von Dr. Adrian Rosebrock [26] gefolgt.

Neben den Performanceeinbussen haben wir noch einen weiteren Nachteil dieser Tracker festgestellt. Wir erhalten sehr viele ID Switches, da die getesteten Algorithmen nur die Bewegungen modellieren und somit nicht damit umgehen können, wenn Personen sich nahe beieinander befinden, verdeckt werden oder sich kreuzen.

### 4.2.3 Deep SORT

Da wir das Problem der ID-Switches für unseren Anwendungsfall lösen müssen, macht es Sinn auch noch die visuelle Repräsentation der Objekte mit zu berücksichtigen. Mit Deep SORT untersuchen wir einen Algorithmus, der sowohl visuelle (appearance based) als auch Bewegungsinformationen (motion based) verwendet. Durch diese Kombination erreicht Deep SORT ein sehr stabiles Tracking von Objekten. [27]

Wie alle Tracking Algorithmen verarbeitet auch Deep SORT die Detections, welche von der Object Detection geliefert werden. Für diese Detections werden in einem ersten Schritt die Appearance Deskriptoren generiert. Dies geschieht mit der Verwendung eines CNN. In Abschnitt 4.2.3.1 gehen wir detaillierter auf die Generierung dieser Deskriptoren ein. In einem zweiten Schritt werden die neuen Positionen der bereits existierenden Tracking Objekte vorhergesagt. Dies geschieht mit Hilfe eines Kalman Filters. Der Kalman Filter ist ein Algorithmus um den zukünftigen Zustand eines System möglichst optimal zu schätzen, ohne dass detailliert bekannt ist, wie das

System funktioniert. Da die Funktionsweise des System nicht bekannt ist, muss der Kalman Filter dafür die bisherigen Messungen verwenden. In unserem Anwendungsfall sind dies die Resultate von bisherigen Object Detection Durchläufen. Der Kalman Filter ermittelt anhand dieser Informationen den kleinst möglichen Fehler für die neuen Vorhersagen. [27] [28]

Deep SORT zählt für jedes Tracking Objekt, wie viele Frames seit der letzten erfolgreichen Detection vergangen sind. Dieses Alter wird bei jeder Kalman Vorhersage um eins erhöht. Kann die Detection durch die visuelle Assoziierung bestätigt werden, so wird ihr Alter wieder auf null gesetzt. Hat ein Tracking Objekt das maximale Alter erreicht, wird es gelöscht. [27]

Detections, die keinem bestehenden Tracking Objekt zugeordnet werden können, werden als temporäre Tracking Objekte erfasst. Damit diese Objekte zu effektiven Tracking Objekten werden können, müssen sie in den nächsten zwei Schritten durch je eine Detection bestätigt werden. Wird ein Tracking Objekt in den nächsten zwei Schritten nicht zweimal bestätigt, so wird es wieder gelöscht. [27]

#### 4.2.3.1 Appearance Deskriptoren

Durch die Verwendung von visuellen Informationen unterschiedet sich Deep SORT stark von anderen Trackern, die nur motion basiert sind. Wir beschrieben im folgenden Abschnitt, wie diese visuellen Informationen gewonnen werden.

Um die visuellen Informationen zu bekommen, werden als Basis die Bounding Boxen aus der Object Detection benötigt. Diese Bounding Boxen definieren die Positionen der Personen. In einem ersten Schritt, werden die Ausschnitte welche von den Bounding Boxen umfasst werden, als sogenannte Patches aus dem Bild extrahiert. Ein CNN untersucht die einzelnen Patches und gibt für jeden Patch ein Feature zurück, welches die visuellen Informationen beinhaltet. Dieser Prozess wird in Abbildung 4.7 stark vereinfacht dargestellt.

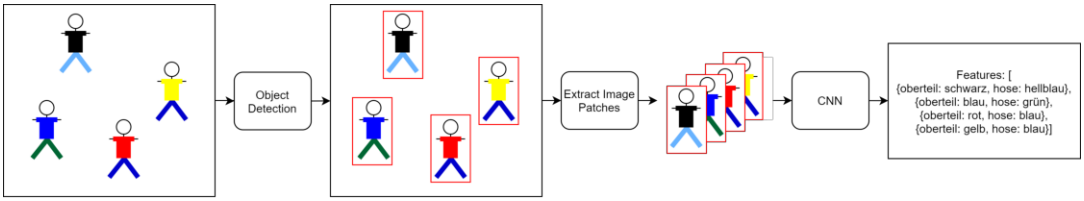


Abbildung 4.7: Generierung der Features

Für das oben genannte Netzwerk wird eine Tensorflow-Session benötigt. Die Inferenzzeit fällt mit durchschnittlich 2.1ms pro Frame auf einer Mittelklasse-Grafikkarte sehr niedrig aus.

Mit Deep SORT konnten wir sehr gute Tracking Resultate erzielen und die Anzahl der ID-Switches deutlich reduzieren. Deshalb verwenden wir den Deep SORT Tracker in unserem Prototyp.

## 4.3 Architektur Überblick

Der generelle Aufbau unseres LOI-Counters beschreiben wir in diesem Abschnitt und anhand der Abbildung 4.8. Wir verwenden als Programmiersprache Python, da die verwendete OpenCV Library eine Python API anbietet, sowie der Tracker in Python geschrieben wurde. Weiter ist Python im Computer Vision und Machine Learning Bereich sehr verbreitet.

In Python stellen einzelne Files Modules dar und mehrere Modules zusammen in einem Ordner sind ein Package. Das Module app ist der Entrypoint in unsere Applikation und parst die Argumente. Unsere Logik lässt sich in wenigen Modulen abbilden, weshalb wir diese zu einem Package `couting_backend` zusammenfassen. Wie in Abbildung 4.8 ersichtlich ist, führt das Module `people_counter` alle darin vorkommenden Modules zusammen.

Das Package `deep_sort` wurde vom Github Repository von Nicolai Wojke kopiert. [29] Dieses Package enthält ein Module `track`, welches die einzelnen Tracking Objekte repräsentiert. Um in unserem Prototypen zu zählen,

wie viele Tracking Objekte die LOI überqueren, wurde dafür ein Wrapper Module `count_track` implementiert, welches sich ebenfalls in diesem Package befindet.

Weiter interessant sind die Dateien welche vom `object_detector.py` verwendet werden. Die Dateien `yolo.weights` und `yolo.cfg` repräsentieren das Netz für die Object Detection. Die Datei `coco.names` beinhaltet eine Liste aller Klassen des Coco Datasets repräsentieren.

Die Datei `mars-small128.pb` wird vom `image_encoder.py` für die Featuregewinnung benötigt.

Der `communication_proxy.py` dient als Schnittstelle für die Kommunikation nach aussen. Die Implementation wird in Kapitel 8 detailliert beschrieben.

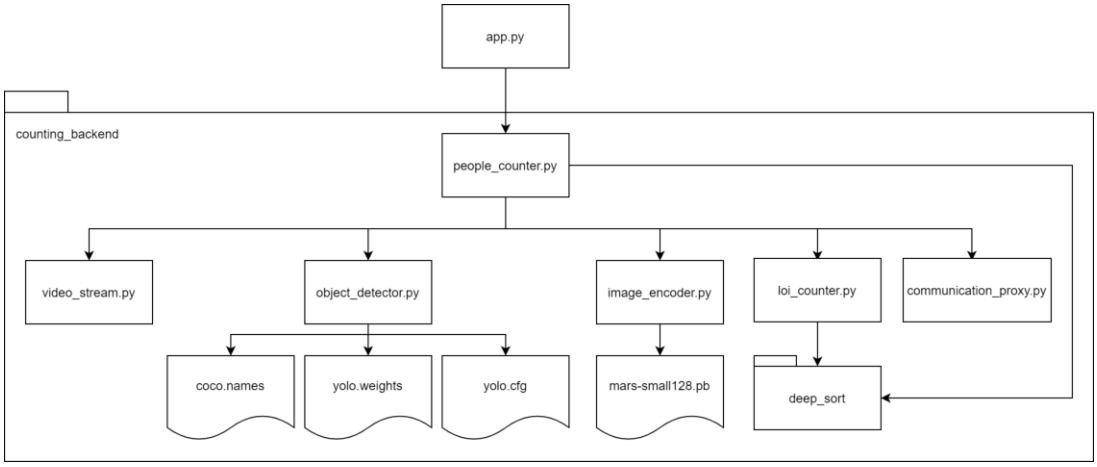


Abbildung 4.8: Architektur im Überblick

### 4.3.1 Zusammenspiel der Komponenten

In Abbildung 4.9 haben wir die Verarbeitung eines einzelnen Frames und das Zusammenspiel der einzelnen Modules visualisiert. Videos und Kameras können als Streams modelliert und mittels OpenCV abgefragt werden.

1. Im ersten Schritt wird über einen blockierenden Aufruf ein einzelnes Frame vom Kamera oder Videostream angefragt.
2. Das Frame wird an den People Counter übermittelt, sobald es verfügbar ist.
3. Dieses Frame wird zusammen mit dem Confidence und NMS Threshold an das Object Detector Module übergeben.
4. Das Object Detector Module führt dann den Forward Pass sowie die NMS durch. Danach liefert es die Bounding Boxen mit den dazugehörigen Confidences zurück an den People Counter.
5. Die Bounding Boxen übermittelt der People Counter dann zusammen mit dem aktuellen Frame an den Image Encoder.
6. Der Image Encoder generiert aus den Bounding Boxen die Features (Appearance Deskriptoren), wie in Abschnitt 4.2.3.1 beschrieben, und gibt diese an den People Counter zurück.
7. Danach werden die neuen Positionen der bisherigen Tracking Objekte vom Object Tracker mittles `predict` neu geschätzt.
8. Anschliessend werden die Appearance Deskriptoren mit den jeweiligen Bounding Boxen zu einer Liste von Detections zusammengeführt. Die Bounding Boxen werden für die Lokalisierung und die Features für die visuelle Identifizierung im Tracker benötigt.
9. Der People Counter übermittelt diese Liste von Detections an den Object Tracker. Dort werden die geschätzten Positionen bestätigt und neue Tracking Objekte erstellt.

10. Im Counter wird untersucht, ob ein Tracking Objekt aus der aktualisierten Liste die LOI übertreten hat.
11. Das Resultat aus den gezählten Linienüberquerungen liefert der Counter zurück an den People Counter.
12. Der People Counter übermittelt die Zählungsresultate an den Communication Proxy. Wie die Kommunikation abläuft wird in Kapitel 8 beschrieben.

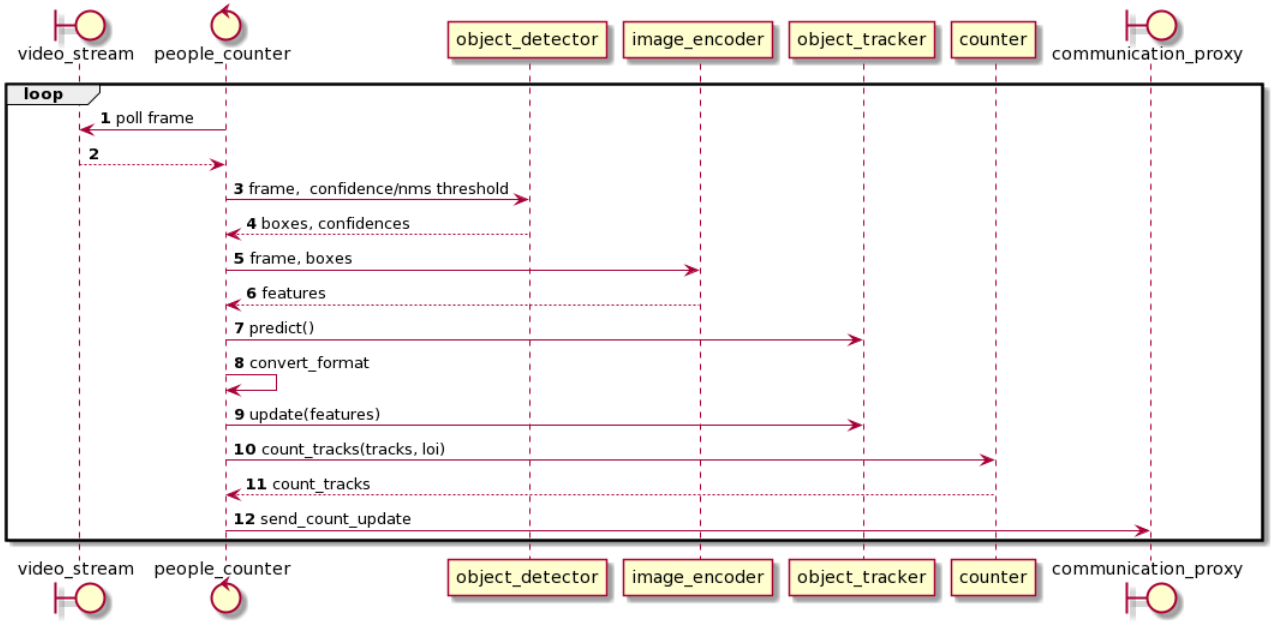


Abbildung 4.9: Verarbeitung eines Frames

# 5 Object Detector Vergleich

Die wichtigste Komponente unseres Counters ist der Object Detector. Ohne gute Detection Resultate ist das Tracking und die damit verbundene Zählung nicht möglich. Aus diesem Grund vergleichen wir in diesem Kapitel mehrere Detection Modelle miteinander und führen ein Benchmarking durch. Wir messen anhand einer eigenen Datensammlung die Inferenzzeit, sowie die Genauigkeit der Personenerkennung und Lokalisierung. Zu den veröffentlichten Algorithmen und Modellen werden zwar meistens Metriken angegeben, diese gelten jedoch über mehrere Objektkategorien hinweg. Beim Coco Datenset [15] [30], welches meistens als Referenz dient, wären dies 80 Kategorien. Weiter sind die Referenzdaten oft zu allgemein, da diese ein breites Spektrum an Bildern und Aktivitäten abdecken.

Die Wahl eines Object Detectors ist immer ein Tradeoff zwischen Inferenzzeit und Genauigkeit. Für unseren Use Case ist es wichtig, dass wir die Frames laufend verarbeiten können und es zu keiner Verzögerung im Betrieb kommt. Bei 30 Bilder pro Sekunde (entspricht 30 FPS), dürfte die komplette Verarbeitungsdauer inklusive Tracking und Zählung pro Frame nicht mehr als 33 ms betragen. Somit sind für uns Detectors interessant, die mit ihrer Inferenzzeit auf einer Mittelklasse-Grafikkarte unter diesem Wert liegen.

## 5.1 Unterschiedliche Netzwerkarchitekturen

Was alle Object Detectors gemeinsam haben, ist die Verwendung eines Backbone Netzwerkes. Dieses wird als eine Art der Vorverarbeitung dazu verwendet, aus Inputbildern Features zu generieren. Aus diesem Grund wird die Komponente auch Feature Extractor genannt. Oft wird dabei ein Classifier verwendet, bei dem der letzte Fully Connected Layer entfernt wurde. Wenn die Anforderungen in Richtung Genauigkeit gehen, wird ein tieferes, dichteres Netzwerk genommen. Für schnellere Inferenzzeit wird ein flacheres, weniger dichtes verwendet. Die Qualtität der Features ist dabei sehr wichtig, da diese die Obergrenze der Genauigkeit eines Detectors bestimmt. [31]

Unterschiede lassen sich anhand der weiteren Komponenten, wie in Abbildung 5.1 dargestellt, feststellen. Two Stage Detectors werten die vom Backbone Netzwerk erzeugten Features in einer separaten Region Proposals Komponente aus. Mittels Segmentierung (R-CNN) oder einem weiteren CNN (Fast R-CNN) werden interessante Regionen bestimmt. Dies führt zu einer hohen Lokalisierungs- und Objekterkennungsgenauigkeit, jedoch auf Kosten der Perfomance. One Stage Detectors verzichten auf diesen Zwischenschritt und bieten somit eine tiefere Inferenzzeit. Die Inferenzzeit kann sich zwischen den beiden Architekturvarianten um über 30 ms unterscheiden (SSD im Vergleich mit Mask-RCNN [32]). Two Stage Detectors übersteigen somit unsere Zeitlimite, weshalb für unsere Anwendung nur One Shot Detectors interessant sind. [31] [12]

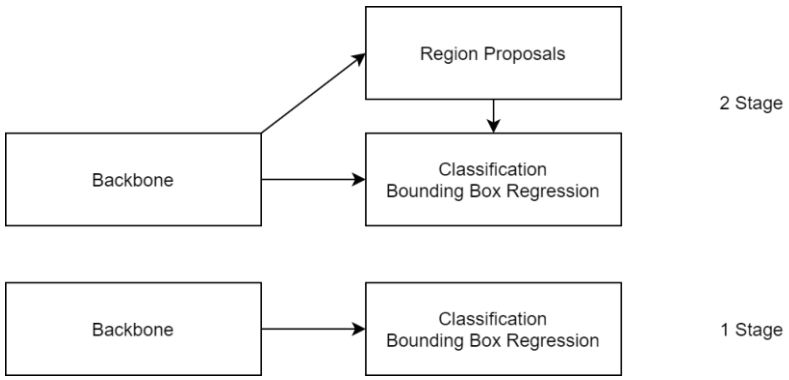


Abbildung 5.1: Netzwerkarchitekturen

## 5.2 Auswahl der Modelle

Researcher und Organisationen stellen vortrainierte Modelle oft im Internet zur Verfügung. Diese wollen wir für unser Projekt verwenden, da uns so das Trainieren eines eigenen Modells erspart bleibt.

Wie bereits in Abschnitt 4.1 erwähnt, verwenden wir für die Inferenz das DNN Modul der OpenCV Library. Modelle, die mit den Frameworks Caffe, TensorFlow, Torch oder Darknet erstellt wurden, werden zum aktuellen Zeitpunkt unterstützt [17]. Die meisten Frameworks führen dabei eine Zusammenstellung an vortrainierten Modellen, an welchen man sich orientieren kann [33] [32] [14] [34].

Aus den genannten Performancegründen legen wir den Fokus auf die One Stage Detectors. YOLO und SSD sind dabei die bekanntesten und verbreitetsten Algorithmen. YOLO (you only look once) [35] wurde von Redmon et al. entwickelt. Der ursprüngliche Algorithmus unterteilt die Inputbilder für die Weiterverarbeitung in ein Gitter mit mehreren Zellen. Für jedes Objekt ist eine Zelle zuständig, nämlich diese, in welcher sich der Mittelpunkt des Objektes befindet. Jede Zelle sagt mehrere Bounding Boxen und mehrere die Confidences pro Klasse für dieses Objekt voraus. YOLOv2 [36] führt Anchor Boxen ein, die es erlauben, mehrere Objekte innerhalb einer Zelle zu erkennen. Die aktuellste Version von Redmon et al., YOLOv3 [37] bietet weitere Verbesserungen wie Multilabel Klassifikation oder ein robusterer Feature Extractor. Für die Extrahierung der Features werden jeweils unterschiedliche Versionen des Darknet Netzwerkes eingesetzt, welches ebenfalls von Redmon stammt.

Für YOLOv3 gibt es verschiedene Varianten und publizierte Modelle [14]. Wir vergleichen das Standardmodell, mit einer verkleinerten Architektur, YOLOv3-tiny, sowie mit einem Modell, welches Spatial Pyramid Pooling (SPP) verwendet. Die Modelle akzeptieren unterschiedliche Bildgrössen als Input. Erste Versuche mit der grössten Einstellung (608 x 608 Pixel) haben gezeigt, dass die Inferenzzeit über unserem Requirement liegt. Aus diesem Grund verwenden wir in unserem Benchmarking die nächstkleinere Bildgrösse 416 x 416 Pixel. Grössere Bilder versprechen bessere Resultate, was gegen die Verwendung der kleinsten Bildeinstellung spricht.

Der SSD (SingleShot Multibox Detector) Algorithmus von Liu et al. [38] verwendet einen anderen Ansatz. Die Idee ist es, dass eine fixe Menge an unterschiedlichen Bounding Boxen über jede mögliche Bildstelle gelegt wird. Pro Bounding Box werden die Confidences und der Abstand zum Objekt berechnet. Dieses Vorgehen wird auf mehreren Layern wiederholt, um mit unterschiedlichen Objektgrössen umgehen zu können. Die Bounding Boxen unterscheiden sich dabei zwischen den Layern ebenfalls in der Skalierung und dem Seitenverhältnis. Tensorflow bietet in seinem Model Zoo [32] mehrere SDD Modelle an. In unserem Vergleich verwenden wir die Modelle mit MobileNetv2 und Inception als Backbone. MobileNet ist ein kleinerer Feature Extractor, der für Smartphones optimiert ist. Inception ist ein grösseres, genaueres Backbone Netzwerk und somit langsamer.

Um eine Einordnung zu den Two Stage Detectors zu erhalten, haben wir zudem den Mask RCNN Algorithmus getestet. Dieser wurde von He et al. [39] erfunden und basiert auf der Region Proposal Idee, wie wir sie im letzten Abschnitt beschrieben haben. Anstelle von Bounding Boxen arbeitet Mask RCNN jedoch mit Masken. In Tensorflows Model Zoo [32] gibt es ein trainiertes Modell, welches ebenfalls Inception als Backbone verwendet.

Von unserem Industriepartner Noser wurde die Microsoft Computer Vision API [40] vorgeschlagen, welche wir ebenfalls in unserem Vergleich aufnehmen. Leider sind diese Algorithmen Closed Source und nicht dokumentiert, weshalb wir keine Einordnung vornehmen können.

| **Algorithmus** | **Varianten** | **Backbone** |
|---|---|---|
| YOLOv3 | 416<br>tiny<br>spp | Darknet |
| SSD | | MobileNet v2<br>Inception v2 |
| MaskRCNN | | Inception v2 |
| Microsoft Computer Vision API | | |

Tabelle 5.1: Modelle

## 5.3 Zusammenstellung der Testdaten

Um überprüfen zu können, wie zuverlässig ein Modell bei der Analyse des jeweiligen Datensatzes ist, wird eine sogenannte Ground Truth benötigt. Die Ground Truth definiert mittels Annotationen, wo sich auf dem Bild Objekte einer bestimmten Klasse befinden. Je grösser die Übereinstimmung zwischen der Ground Truth und den durch ein Modell erkannten Objekten ist, desto besser ist dessen Zuverlässigkeit. Um eine exakte Aussage zur Zuverlässigkeit eines Modells machen zu können, ist die Qualität der Ground Truth hierbei aber von grosser Wichtigkeit. Bevor wir uns für die verwendeten vier Datensätze entschieden haben, wurde die Qualität der Ground Truth genauer untersucht. Dabei stellten wir fest, dass die Ground Truth sehr unterschiedlich umgesetzt wird. Manche Ground Truth Daten beinhalten nur Annotationen von komplett sichtbaren Objekten, andere gar von Objekten die praktisch komplett verdeckt oder nur halb bzw. noch gar nicht im Bild sind. Zudem unterliegen die Dimensionen der Bounding Boxen keinem Standard. Das ist ein weit verbreitetes Problem und wurde von Milan et al. bereits detailliert untersucht. [41]

Diese Unterschiede zwischen den einzelnen Ground Truth Daten machen es schwierig, eine exakte Aussage zur Zuverlässigkeit eines Modells zu treffen. Für unser Benchmarking haben wir darum die vier folgenden Datasets verwendet.

- RGB-D Pedestrian Dataset, EPFL [42]
- PETS09-S2L1, MOT Challenge 15 [43]
- MOT16-04, MOT Challenge 16 [44]
- Towncentre Dataset, Oxford University [21]

Grundsätzlich sollte jede Person die man von Auge erkennen kann auch annotiert sein. Nicht annotierte Personen würden zu False Positives führen, wenn sie von einem Modell dennoch erkannt werden. Wie man in Abbildung 5.2 erkennen kann, ist dies bei den folgenden vier Datasets sehr gut umgesetzt. Dies erlaubt eine einheitliche Evaluierung der verschiedenen Modelle.


Abbildung 5.2: Visualisierung der Ground Truth von MOT15 (1), EPFL (2), MOT16 (3) und Towncentre (4)

Hinzu kommt, dass diese Videos alle als Aufnahme von oben auf einen Weg oder eine Strasse aufgezeichnet wurden. Um Personen zuverlässig zählen zu können, sollten die Kameras im Zoo ebenfalls so angebracht werden, dass sie den Weg vor einem Eingang oder einem Gang filmen. Durch die Verwendung dieser Videos erreichen wir somit eine gute Approximation der Situation im Zoo.

## 5.4 Genauigkeitsmetrik

Die gängige Metrik für das Bestimmen der Genaugikeit eines Object Detectors ist die Average Precision (AP). Diese wird von Wettbewerben wie der COCO Challenge [15] oder der PASCAL Visual Object Classes (VOC) Challenge [45] verwendet. Die folgenden, grundlegenden Begriffe sind wichtig für das Verständnis:

**True Positives** Detections, die mit den Annotationen aus der Ground Truth übereinstimmen

**False Positives** Detections, die nicht in den Annotationen existieren

**False Negatives** Annotierte Objekte, die vom Object Detector nicht entdeckt wurden

Wie bereits beschrieben, klassifiziert ein Object Detector Objekte und lokalisiert diese mit Hilfe von Bounding Boxen. Die Genauigkeit der Bounding Boxen ist wichtig für das Tracking. Diese stimmen jedoch nicht exakt mit den Ground Truth Koordinaten überein. Wie wird diese Abweichung gehandhabt? In Kapitel 4.1.3 haben wir die IoU Metrik beschrieben. Mit dieser kann man auch die Überlappung einer gefundenen Bounding Box mit der Ground Truth bestimmen. Mit Hilfe eines Thresholds bestimmt man einen Toleranzbereich und teilt so die Objekte in True und False Positives auf. Pro Objekt kann es nur eine True Positive Detection geben. Die AP wird jeweils für einen bestimmten IoU Threshold berechnet und dementsprechend ausgewiesen (z.B. $AP_{50}$ wäre die Average Precision berechnet mit IoU Threshold 50). In unserem Benchmarking verwenden wir den Ansatz der COCO Challenge, welche die AP für 10 verschiedene Thresholds von 0.5 bis 0.95 (mit Abstand von jeweils 0.05) berechnet und diese mittelt. Dies führt dazu, dass Detectors mit besserer Lokalisierung weiter oben rangieren. [15]

Die Average Precision setzt sich aus Precision und Recall zusammen. Die Precision weist aus, wie gross der Anteil der True Positives unter allen gefundenen Objekten ist. Desto näher die Precision bei 1.0 ist, desto weniger False Positives gibt es und desto präzisien ist somit das Resultat. [46] [47]

$$\text{total gefunden} = \text{True Positives} + \text{False Positives}$$
$$\text{Precision } p = \frac{\text{True Positives}}{\text{total gefunden}}$$

Recall stellt dar, wie viele Objekte aus der Ground Truth tatsächlich durch den Detector gefunden wurden. Ist diese Metrik nahe bei 1.0, gibt es wenig False Negatives und der Detector erziehtl eine hohe Abdeckung. [46] [47]

$$\text{annotierte Objekte} = \text{True Positives} + \text{False Negatives}$$
$$\text{Recall } r = \frac{\text{True Positives}}{\text{annotierte Objekte}}$$

In Abbildung 5.3 ist ein typischer precision-recall Plot abgebildet. Grundsätzlich besteht ein Tradeoff zwischen Precision und Recall. Wie in der Abbildung ersichtlich ist, bedeuten höhere Recall Werte tendenziell tiefere Precision Werte und umgekehrt. Der Detector muss z.B. mehr Objekte erkennen, wenn ein hoher Recall Wert erzielt werden soll. Dies führt wiederum zu mehr False Positives und tieferer Precision. Die Qualität des Object Detectors lässt sich an der Fläche unter der Kurve messen. Je grösser diese ist, desto besser sind die Detection Resultate. Ein idealer Object Detector hätte ein konstanter Precision Wert von 1.0 bei ansteigendem Recall, was der grösstmöglichen Fläche entspricht. [46]

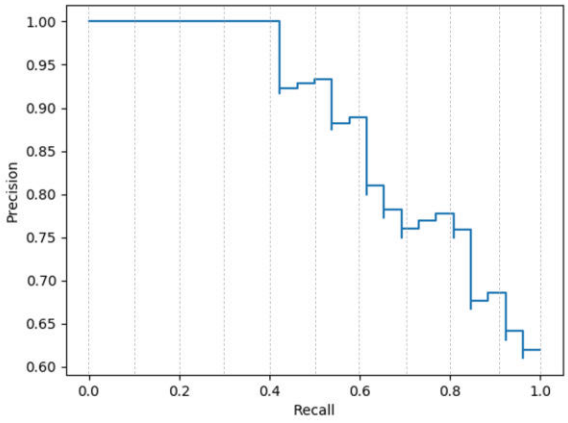


Abbildung 5.3: Beispiel für ein precision-recall Diagramm

Die AP fasst den Verlauf des precision-recall Diagramms zusammen. Die Precisionwerte werden an bestimmten Recall Stellen abgelesen und gemittelt. Die Levels sind der Abbildung 5.3 mit Hilfe der Linien dargestellt. Für die Berechnung der AP gilt die folgende Formel: [45]

$$AP = \frac{1}{\#Levels} \sum_{r \in Levels} p(r)$$

Die Pascal Challenge [45] verwendet 11 Levels. Das COCO Datenset [15], an welchem wir uns orientieren, hingegen verwendet eine viel feinere Diskretisierung von 101 Punkten. Anstelle der exakten Precision, wird diese meistens interpoliert. Dies wird getan, um Ausreisser von einzelnen Beispielen zu glätten. [45]

## 5.5 Benchmarking Resultate

Das Benchmarking wurde auf einer Maschine mit den folgenden Spezifikationen durchgeführt:

- Prozessor: Intel i7-5930K CPU 6 Core @ 3.7GHz
- Arbeitsspeicher: 64GB
- Grafikkarte: ZOTAC GeForce GTX 1080 AMP Edition
- Betriebssystem: Ubuntu 18.04.4 LTS
- Python-Version: 3.6
- OpenCV-Version: 4.6

OpenCV führt die Modelle für die Object Detection standardmässig über die CPU aus. Um die Geschwindigkeit der Modelle zu verbessern, gibt es die Möglichkeit OpenCV für die Grafikkarte zu kompilieren. Wir haben dafür diese Anleitung befolgt. [48] Damit wurden die Inferenzzeiten für alle Modelle auf OpenCV stark verkleinert. Bei YOLO 416 konnte die Inferenzzeit von 740 ms auf 20.7 ms gebracht werden. Das entspricht einer Beschleunigung um Faktor 35.74.

### 5.5.1 Zuverlässigkeit

Die Zuverlässigkeit eines Modells kann mit der oben beschriebenen AP angegeben werden. Wir haben die Modelle auf die vier bereits beschriebenen Datensätze angewandt und aus den Resultaten die AP berechnet. Dabei weichen einige unserer Messungen von den offiziellen Zahlen ab, da wir andere Daten verwendet haben. Weiter könnte die Qualität der Ground Truth einen starken Einfluss haben.

| | RGB-D EPFL | PETS09 MOT15 | MOT16-04 | Towncentre Oxford |
|---|---|---|---|---|
| YOLO 416 | 33.18 | 39.99 | 22.76 | 27.48 |
| YOLO SPP | 31.31 | 39.07 | 24.52 | 26.41 |
| MaskRCNN | 30.60 | 31.58 | 28.69 | 29.04 |
| SSD-Inception | 28.69 | 29.35 | 11.88 | 12.09 |
| SSD-Mobilenet V2 | 27.14 | 25.33 | 7.85 | 10.70 |
| Azure | 27.19 | 16.28 | 8.17 | 7.02 |
| Tiny YOLO | 18.48 | 13.95 | 5.47 | 3.20 |

Tabelle 5.2: AP der jeweiligen Modelle

Tabelle 5.2 zeigt, dass YOLO 416 (⌀ 30.85), YOLO SPP (⌀ 30.25) und MaskRCNN (⌀ 29.98) im Durchschnitt die höchste Zuverlässigkeit aufweisen konnten.

In Abbildung 5.4 beschreiben die Precision-Recall Kurven die Zuverlässigkeit der untersuchten Modelle. Wie in Abschnitt 5.4 bereits erwähnt, wurden die Kurven für jeweils 10 IoU Threshholds generiert, um zusätzlich auch die Zuverlässigkeit der Lokalisierung zu visualisieren. [49]

Wie man in Abbildung 5.4 erkennen kann bleibt die Precision bei YOLO SPP, YOLO 416 und MaskRCNN bei zunehmendem Recall lange stabil. Bei SSD-Inception und SSD-MobileNet fällt die Precision bei zunehmendem Recall gegen das Ende stark ab und bei Tiny YOLO und Azure sinkt die Precision bereits von Beginn an.

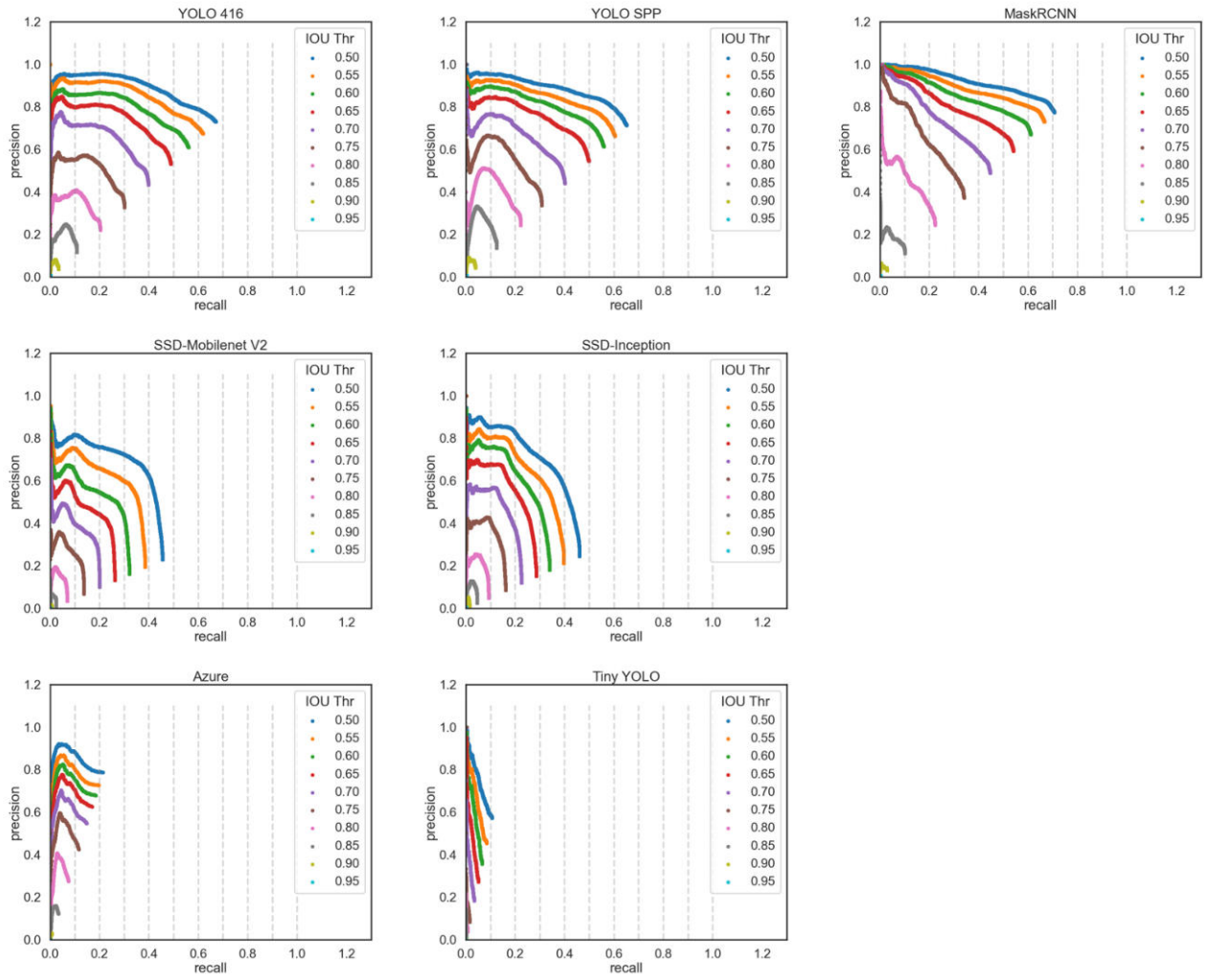


Abbildung 5.4: Die Precision-Recall-Kurven der einzelnen Modelle über alle vier Datensätze

### 5.5.2 Zeitliche Messung

Neben der Genauigkeit der Modelle, ist die Geschwindigkeit für unsere Auswahl ebenfalls von Interesse. Unsere Lösung soll in Echtzeit laufen können. Aus diesem Grund darf die Object Detection nicht länger als 33 ms dauern. Um zu untersuchen, welche Modelle diese Bedingung erfüllen, haben wir die Inferenzzeiten der unterschiedlichen Modelle über jeweils 1000 Frames genesseb. Azure wurde aus den zeitlichen Messungen ausgeschlossen. Da Azure bei der Untersuchung der Genauigkeit bereits eher schlecht abschnitt, haben wir entschieden, dieses Modell nicht weiter in Betracht zu ziehen.

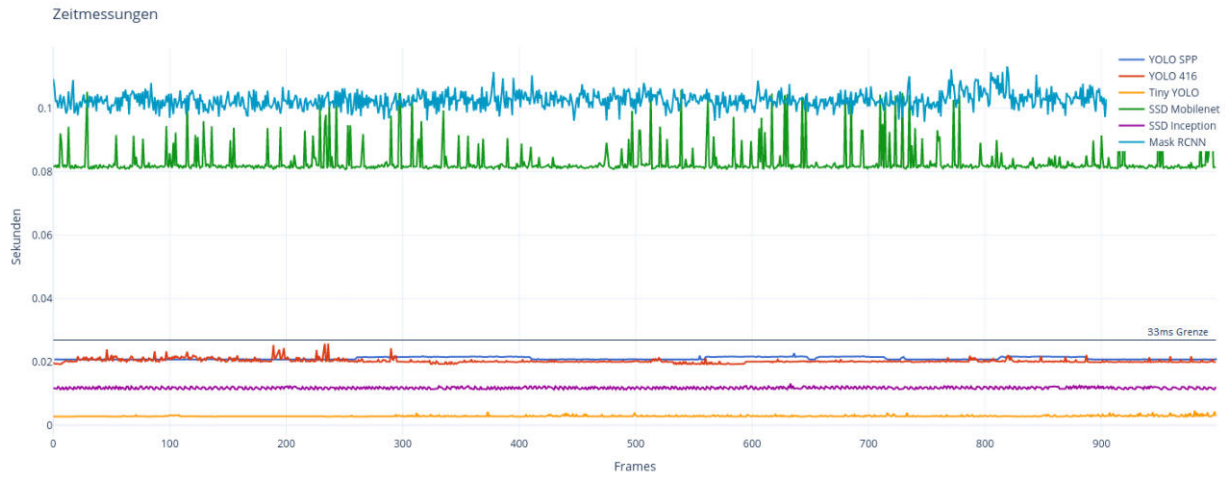


Abbildung 5.5: Zeitmessungen der Algorithmen

In Abbildung 5.5 werden die Laufzeiten der einzelnen Modelle über jeweils 1000 Analysen visualisiert. Um für unseren Anwendungsfall in Betracht gezogen zu werden, muss die durchschnittliche Inferenzzeit eines Modells unter 33 ms liegen. Mask RCNN mit durchschnittlich 102 ms und SSD Mobilenet mit durchschnittlich 83 ms sind somit zu langsam.

YOLO SPP und YOLO 416 weisen eine durchschnittliche Inferenzzeit von 21 ms bzw. 20.7 ms auf und sind somit schnell genug. Auch SSD Inception mit 11.7 ms und Tiny YOLO mit 2.8 ms erfüllen die Geschwindigkeitsanforderung.

## 5.6 Fazit

Aus den zeitlichen Messungen in Abbildung 5.5 bleiben noch YOLO SPP (21 ms), YOLO 416 (20.7 ms), SSD Inception (11.7 ms) und Tiny YOLO (2.8 ms) übrig. Sie alle weisen eine durchschnittliche Inferenzzeit von unter 33 ms auf. Aus diesen vier Modellen gilt es nun das mit der höchsten Genauigkeit auszuwählen. Über die vier Datensätze hinweg weisen YOLO 416 und YOLO SPP mit einer durchschnittlichen AP von 30.85 bzw. 30.25, die höchste Genauigkeit auf. SSD Inception mit 20.5 und Tiny YOLO mit 10.36 liegen dagegen um einiges tiefer.

Da für unser Prototyp sowohl die Genauigkeit als auch die Geschwindigkeit des Modells wichtig sind, entscheiden wir uns für die Verwendung von YOLO 416 als Detection Modell. Die durchschnittliche Inferenzzeit von 20.7 ms erreichen wir einen Puffer von 12.3 ms, um weitere Aufgaben wie Tracking, Filterung etc. ausführen zu können. Zudem ist es mit einer durchschnittlichen AP von 30.85 in unserem Benchmarking das genauste Modell.

# 6 Performance

Im letzten Kapitel haben wir den für uns passenden Object Detector ausgewählt und seine Inferenzzeit gemessen. Unter den Modellen welche die zeitliche Anforderung von unter 33 ms Inferenzzeit erfüllen, war YOLO 416 das genauste. In diesem Kapitel untersuchen wir die Performance des gesamten Prototyps und nehmen weitere Optimierungen vor, um die Zeitlimite von 33 ms pro Frame einhalten zu können.

Für die weiteren Optimierungen haben wir eine ausführlichere Messung anhand von 50'000 Frames eines HD-Videos (1280x720 Pixel) durchgeführt. Wie Abbildung 6.1 zeigt, benötigen die unterschiedlichen Phasen in der Frameverarbeitung unterschiedlich viel Zeit. Durchschnittlich werden 35.6 ms benötigt, somit liegt die Gesamtzeit zur Verarbeitung eines Frames bereits über der maximalen Grenze.

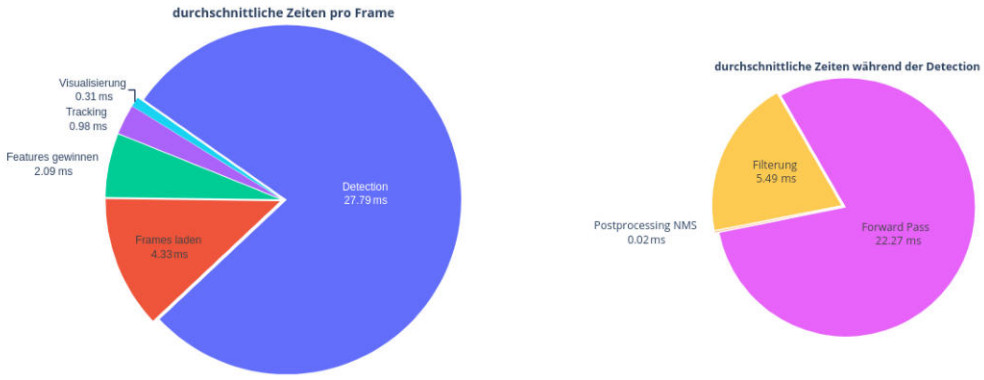


Abbildung 6.1: durchschnittliche Verarbeitungszeiten eines Frames in ms

In den folgenden Abschnitten beschreiben wir verschiedene Schritte, mit welchen wir die durchschnittliche Gesamtzeit für die Verarbeitung eines Frames zu verkleinern versuchen. Für die Feature Gewinnung, das Tracking selbst, sowie die Visualisierung der Tracking Objekte und der LOI werden lediglich 3.38 ms benötigt. Wenn wir in der Gesamtzeit unter die Grenze von 33 ms kommen, spielt auch die Dauer des Frame ladens keine Rolle. Das ist ein blockierender Aufruf und die Kamera kann nur so schnell Bilder übermitteln, wie sie sie auch aufzeichnet. Somit bleibt die Object Detection mit durchschnittlich 27.79 ms als einziger für Optimierungen übrig. In den folgenden Abschnitten zeigen wir einige Massnahmen auf, mit welchen wir diese Zeit reduzieren konnten.

## 6.1 Skipping

In Abbildung 6.1 kann man erkennen, dass die Object Detection den grössten Teil der Verarbeitungszeit ausmacht. Unsere Überlegung besteht nun darin, nicht jedes Frame von der Detection und dem Tracking verarbeiten zu lassen. Wenn nur jedes zweite oder gar dritte Frame verarbeitet würde, könnte die durchschnittliche Verarbeitungszeit deutlich reduziert werden. Für die Frames welche nicht durch von der Object Detection verarbeitet würden, würden nur die 4.33 ms für das Laden und die 0.31 ms für die Visualisierung anfallen, die restlichen gut 31ms fallen weg. In Abbildung 6.2 erkennt man, wie das vom Ablauf her funktionieren würde.

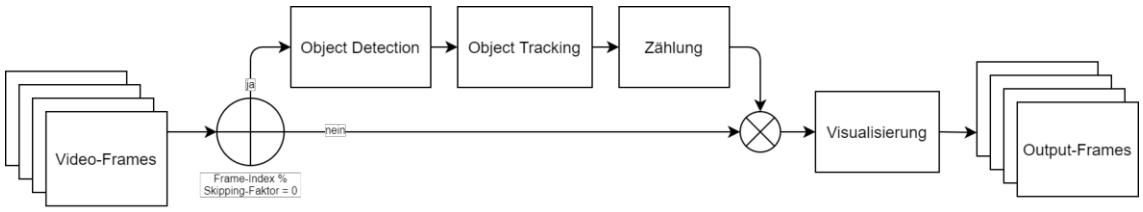


Abbildung 6.2: Verarbeitung mit Skipping

Erste Messungen zeigten, dass die durchschnittliche Verarbeitungszeit mit dem Skipping stark verringert werden kann. Wie in Abbildung 6.3 ersichtlich, sinkt sich die anfängliche Zeit von 35.6 ms mit einem Skipping Faktor von 2 auf 20ms.

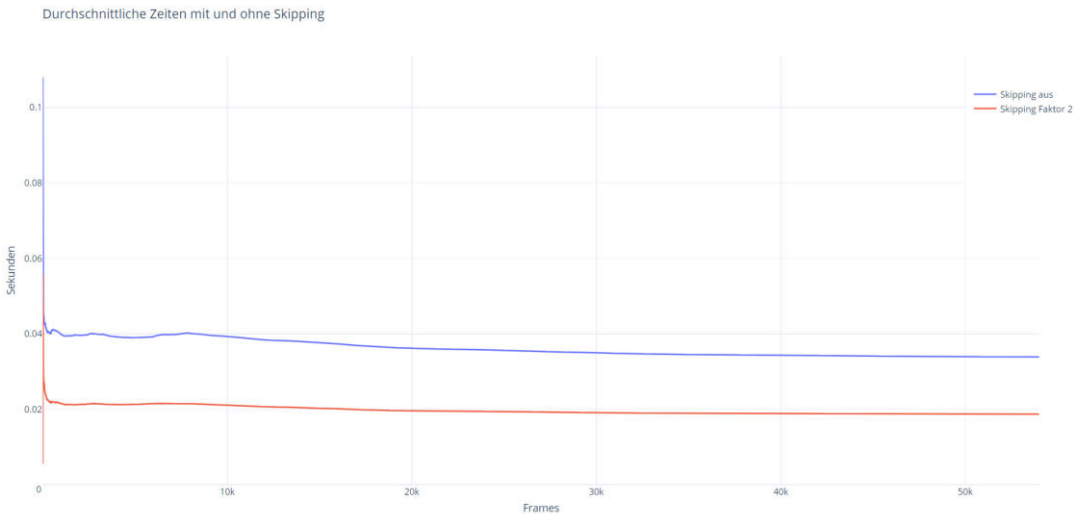


Abbildung 6.3: Durchschnittliche Zeiten für die Skipping Faktoren 1 und 2

Nun stellt sich die Frage, ob das Tracking durch das Skipping nicht beeinträchtigt wird. Schliesslich wird der Tracking Algorithmus bei einem Skipping Faktor von 2 nur noch halb so oft ausgeführt. Tatsächlich hat das Skipping einen Einfluss auf das Tracking, allerdings ist dieser Tradeoff aus Performance Gründen zwingend. Aus diesem Grund wählen wir den tiefst möglichen Faktor 2.

Um die Auswirkungen des Skippings auf die Genauigkeit unseres Prototypen zu untersuchen, verwenden wir mehrere Videos des SAIVT Datasets. [50] Dieses wird in Abschnitt 7.1 detailliert erläutert. Die Videos wurden einmal ohne und einmal mit Skipping Faktor 2 verarbeitet. In diesen Resultaten vergleichen wir die Anzahl falscher und korrekter Zählungen. Diese Untersuchung haben wir mit dem in Abschnitt 7.3.1 evaluierten Wahrscheinlichkeits Grenzwert ermittelt. Dabei stellten wir fest, dass sich das Skipping sogar leicht positiv auf die Zuverlässigkeit des Tracking Algorithmus auswirkt. Von 298 falschen und 30 verpassten Zählungen auf 892 korrekte Zählungen ohne Skipping, kommen wir mit einem Skipping Faktor von 2 nun auf 152 falsche und 41 verpasste auf 881 korrekte Zählungen. Dadurch dass die Object Detection weniger oft ausgeführt wird, verändern sich auch die Dimensionen der Bounding Boxen seltener. Das bedeutet, dass Bounding Boxen die genau über der LOI liegen, weniger oft Zählungen auslösen können. Die Resultate zeigen, dass einige Zählungen verpasst werden, die Reduktion von falschen Zählungen ist aber deutlich höher was zu einem besseren Gesamtresultat führt.

## 6.2 Auslagerung in C

Wie man in Abbildung 6.1 erkennen konnte, benötigt die Filterung während der Object Detection durchschnittlich 5.49 ms. Abbildung 6.4 zeigt, dass es dabei einige Ausreisser gibt, die länger dauerten als die Inferenz selbst. Wir vermuten, dass diese Ausreisser auf die grosse Liste an potentiellen Detections zurück zu führen ist, welche

verarbeitet werden muss. Bei der Iteration über derart grosse Listen können die Laufzeiten in unterschiedlichen Programmiersprachen stark unterschiedlich sein. Prof. Dr. Bläser hat uns darauf hin gewiesen, dass man solche Operationen am besten in C auslagert. Mittels Cython [51] lassen sich solche C Extensions einfach umsetzen. Man erweitert den Python Code um statische Typdeklarationen, lässt sich mit Hilfe von Cython den C Sourcecode erstellen und kompiliert diesen anschliessend mit Hilfe eines C Compilers.

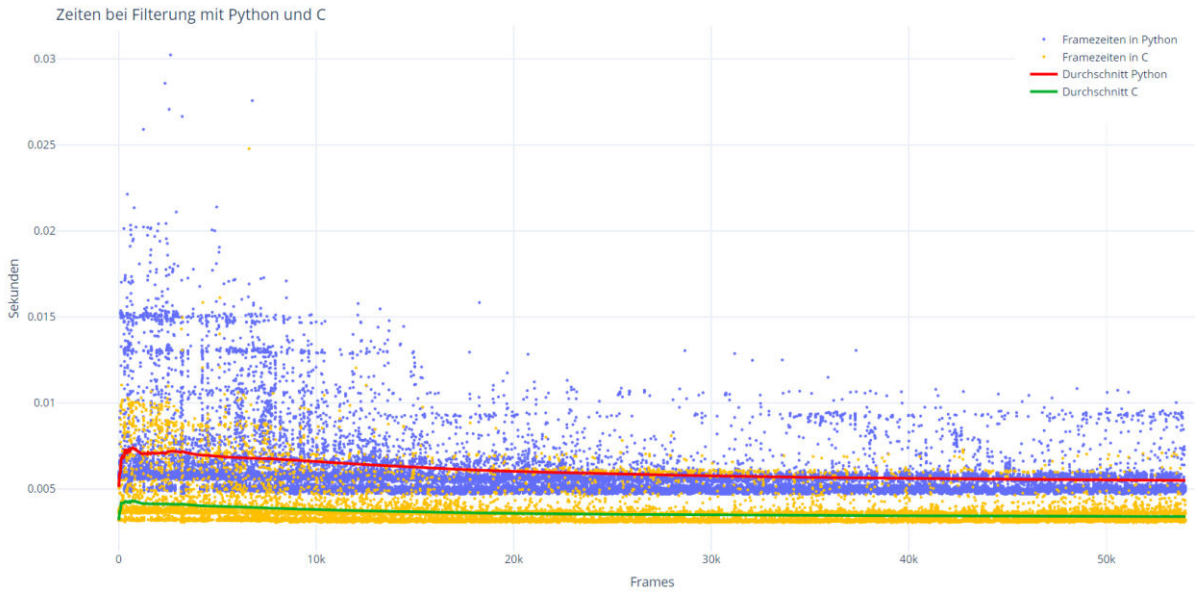


Abbildung 6.4: Zeiten bei Filterung mit Python und C

In Abbildung 6.4 sieht man wie sich die durchschnittliche Laufzeit verändert hat. Durch diese Auslagerung konnten wir die Filterung um über zwei Millisekunden, auf durchschnittlich 3.35 ms beschleunigen, was einem Faktor von ca. 1.6 entspricht. Zudem konnten damit die vielen Ausreisser deutlich verringert werden. In der durchschnittlichen Verarbeitungszeit spielen die damit eingesparten 2 ms keine grosse Rolle. Da die Logik des Prototyps durch diese Optimierung zudem nicht betroffen ist und wir den reinen Performanceaspekt erörten wollen.

## 6.3 Probleme mit Tensorflow

In unserem Prototypen laufen zwei Komponenten auf der Grafikkarte. Einerseits das OpenCV Netzwerk für die Object Detection und andererseits das Tensorflow Netzwerk für die Generierung der Features. Laut Tensorflow-Dokumentation ist es allerdings so, dass eine Session standardmässig 90% des verfügbaren Grafikspeichers alloziert. Das führte dazu, dass ein Prozess über 7 der verfügbaren 8 GB unserer Grafikkarte für sich belegte. Mit **allow-growth: true** kann die Tensorflow-Session allerdings so konfiguriert werden, dass nur so viel Speicher alloziert wird, wie unbedingt nötig. Sollte zur Laufzeit mehr Speicher benötigt werden, kann die Session selbständig den zusätzlich benötigten Speicher allozieren. Nach dieser Konfiguration alloziert eine Tensorflow-Session noch rund 2.1 GB Grafikspeicher. [52] [53]

## 6.4 Resultat

Zu Beginn erreichten wir eine durchschnittliche Verarbeitungszeit von 35.6 ms. Durch die oben beschriebenen Optimierungen kommen wir mit einem Skipping-Faktor von 2 auf eine durchschnittliche Verarbeitungszeit von 18.75 ms.

Durch die Konfiguration der Tensorflow Sessions haben wir nun auch die Möglichkeit, bis zu drei Streams parallel verarbeiten zu können. Dadurch erhöht sich allerdings die durchschnittliche Verarbeitungszeit. Wie man in Abbildung 6.5 erkennen kann, erreichen wir bei drei parallelen Prozessen eine durchschnittliche Verarbeitungszeit von 28.35 ms, was einer Rate von gut 35 FPS entspricht.

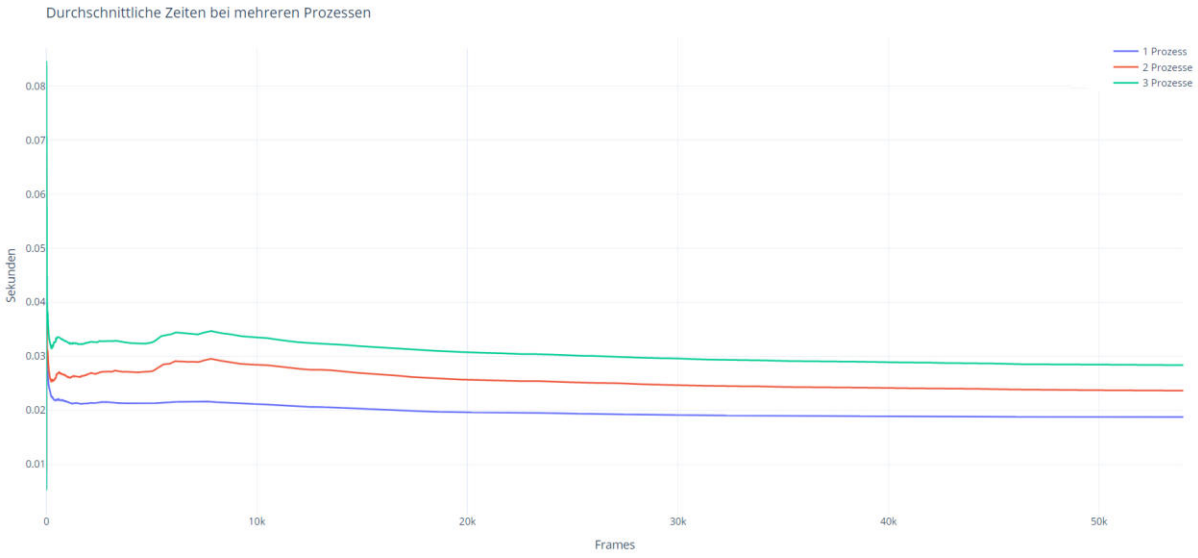


Abbildung 6.5: Durchschnittliche Zeiten für ein (18.75 ms), zwei (23.63 ms) und drei (28.35 ms) parallele Prozesse

# 7 Optimierungen und Evaluation

In diesem Kapitel untersuchen wir die Zuverlässigkeit unseres Zählers als Gesamtsystem. Somit testen wir keine einzelne Komponente wie in Kapitel 5, sondern das Zusammenspiel der Komponenten miteinander.

Unser Vorgehen ähnelt dem Trainieren und Evaluieren eines Machine Learning Modells. Wir teilen in einem ersten Schritt unsere Testdaten in ein Trainingsset und Evaluationsset auf. Auf Basis des Trainingssets optimieren wir unser System und prüfen dessen Zuverlässigkeit mit Hilfe einer Metrik. Ziel ist es bei den unterschiedlichen Optimierungen die beste Konfiguration zu bestimmen. Wenn wir diese gefunden haben, wird die Evaluation auf einem anderen Datenset einmalig ausgeführt und bestimmt, ob wir die 80% Zuverlässigkeit erreicht haben.

Da der Zoo aufgrund der COVID19-Pandemie geschlossen wurde, war es nicht möglich, genügend Videoaufnahmen im Zoo zu tätigen. Um die Zuverlässigkeit unseres Prototyps zu untersuchen, haben wir darum nach einem Datensatz gesucht, welchen wir für unsere Untersuchungen verwenden können. Der SAIVT Datensatz bietet 14 verschiedene Videoaufnahmen von Wegen und Eingängen auf einem Hochschul-Gelände. Diese Videoaufnahmen dauern jeweils zwei Stunden und bieten somit genügend Material um unseren Prototyp zu untersuchen. [50]

## 7.1 Trennung der Testdaten

Wichtig für das Trainieren ist es, dass die Trainings- und Evaluationsdaten sauber getrennt werden und von unterschiedlichen Aufnahmen stammen. Ansonsten kann nicht bestimmt werden, ob das System overfitted ist. Overfitting bedeutet, dass ein Machine Learning System die Testdaten auswendig lernt. Dies führt dazu, dass sehr gute Resultate auf den Testdaten erzielt werden, das System jedoch auf anderen Daten nicht anwendbar ist. [54]

Als Testvideos haben wir vier verschiedene Videosequenzen aus der oben genannten Datensammlung ausgewählt. Diese sind in Abbildung 7.1 ersichtlich. Da für die Zählung keine Ground Truth existiert, müssen wir diese manuell überprüfen. Um den Aufwand im Rahmen zu halten, haben wir aus den gewählten Videos jeweils die ersten 30 Minuten extrahiert. Weiter konnten wir bei unserem einmaligen Besuch im Zoo eine 45 minütige Aufnahme tätigen, die wir ebenfalls in unseren Testdaten verwenden.

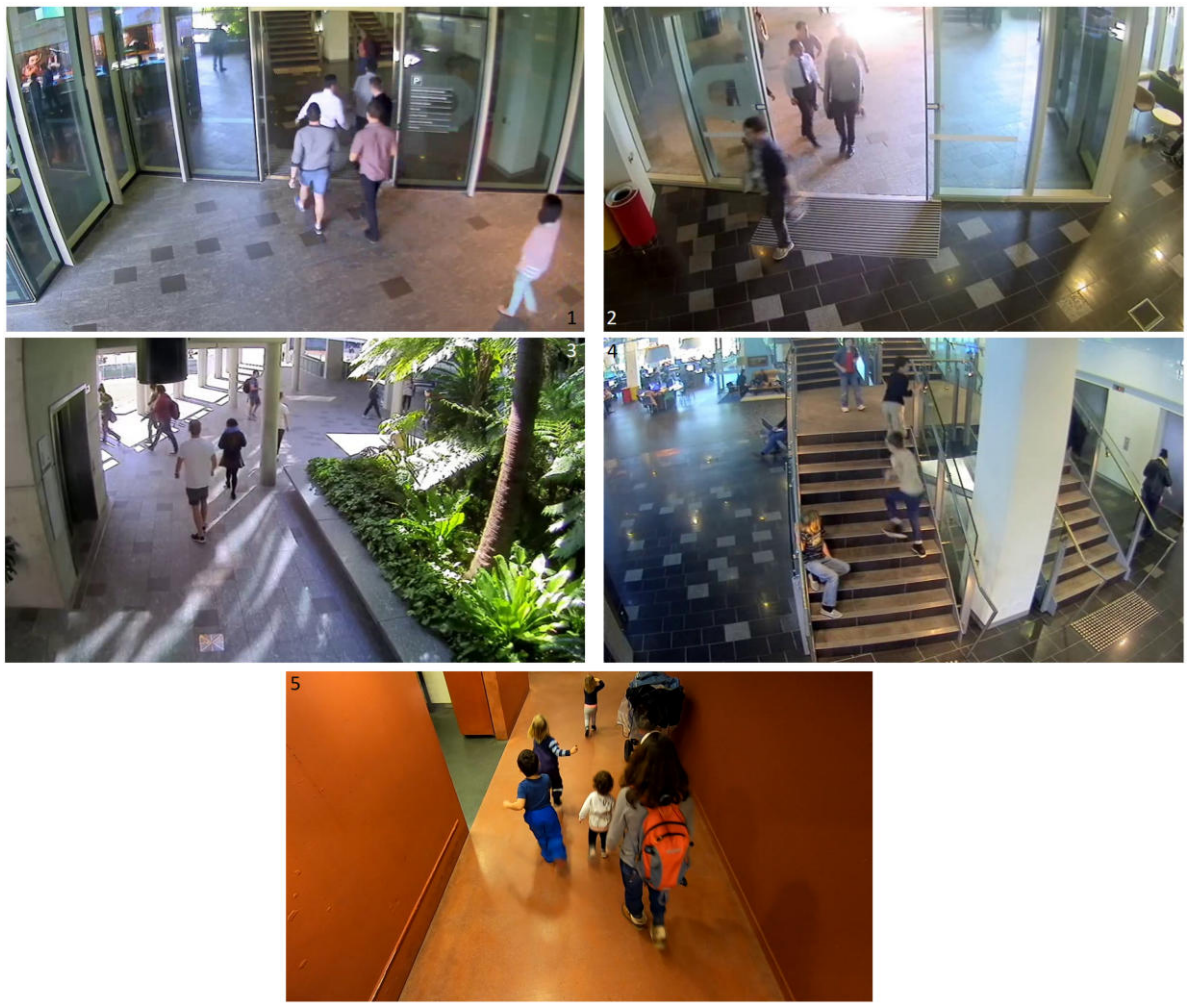


Abbildung 7.1: Ausschnitte der Videos Eingang 1 (1), Eingang 2 (2), Lift (3), Treppenhaus(4) und Masoala (5)

Für die Evaluation haben wir fünf weitere Videos aus dem SAIVT Datensatz ausgewählt. Diese sind in Abbildung 7.2 ersichtlich. Die Evaluation haben wir dabei pro Video auf eine Dauer von 1.5 Stunden beschränkt.

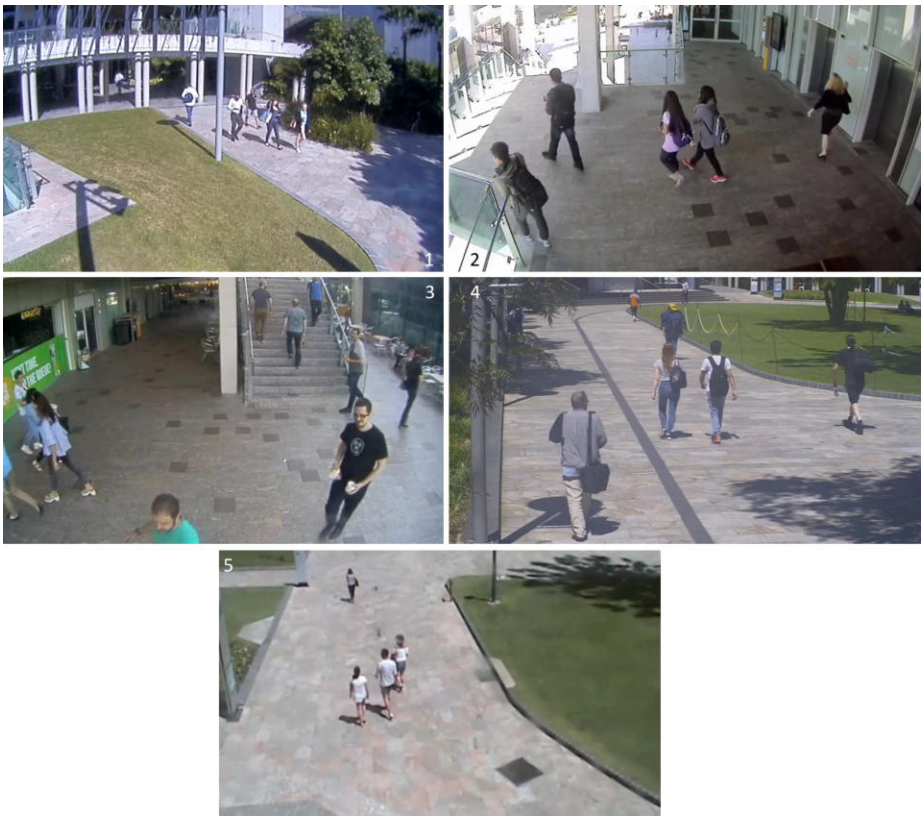


Abbildung 7.2: Ausschnitte der Videos Weg zum Eingang (1), äusserer Lift (2), Treppe (3), Hauptweg 1 (4) und Hauptweg 2 (5)

## 7.2 Genauigkeitsmetrik

Unser eigenes Ziel war es ja eine Genauigkeit von 80% zu erreichen. Doch wie berechnet man diese überhaupt? Genauigkeit beschreibt die Nähe einer Messung zu einem bestimmten Wert. Formal ist sie definiert als: [47]

$$\text{Genauigkeit} = 1 - \text{Fehlerrate}$$

Die Fehlerrate ist nun je nach Anwendung unterschiedlich. In Abschnitt 5.4 haben wir Begriffe wie False Positives und Negatives für Object Detectors beschrieben. Diese Begriffe verwenden wir für die Fehlerrate und definieren sie somit neu für den Zähler als Gesamtsystem:

**False Positive** Zählung die erfolgt ist, ohne dass eine Person die Schranke überschritten hat

**False Negative** eine Person hat die Schranke überschritten, ohne dass eine Zählung ausgelöst wurde

**True Positive** eine korrekte Zählung

Alle False Positives und Negatives zusammen ergeben den Gesamtfehler. Die Fehlerrate erhalten wir, wenn wir die Fehler ins Verhältnis zu allen erwarteten Zählungen setzen. Dies ergibt die folgende Formel:

$$\text{Fehlerrate} = \frac{\text{False Negatives} + \text{False Positives}}{\text{False Negatives} + \text{True Positives}}$$

## 7.3 Optimierungen

In den oben beschriebenen Test-Videos gibt es unterschiedliche Fälle, in welchen die Zählung nicht zu 100% korrekt ist. Wir haben diese Fälle untersucht und dabei nach Optimierungen gesucht, welche die Probleme lösen können. Diese Optimierungen und ihre Resultate werden in den folgenden Absätzen beschrieben. Das Ziel der Optimierungen ist es, die Genauigkeit über den gesamten Datensatz zu verbessern und dementsprechend die Fehlerrate zu minimieren.

### 7.3.1 Confidence Threshold

Zuerst gilt es, die Object Detection so zu konfigurieren, dass dem Tracking Algorithmus die bestmöglichen Resultate übermittelt werden. Mit dem Confidence Threshold kann die Filterung von potentiellen Detections konfiguriert werden. Häufig wird dafür der Wert 0.50 verwendet. Wir haben diesen mit den zwei Werten 0.35 und 0.20 verglichen. In Abbildung 7.3 kann man erkennen, dass die besten Resultate mit einem Wert von 0.35 erzielt werden konnten. Die Genauigkeit liegt dabei bei 78.06%. Beim Threshold 0.20 gibt es zwar weniger False Negatives, dafür deutlich mehr False Positives. Die Genauigkeit liegt bei 78.63%. Der Ausgangswert von 0.5 erzielt etwas weniger False Positives, dafür deutlich mehr False Negatives. Daraus resultiert eine Genauigkeit von 76.78%.

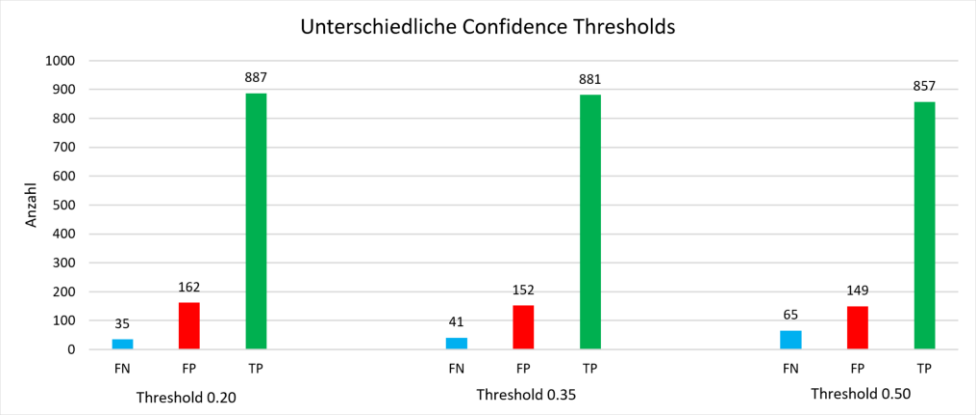


Abbildung 7.3: Auswirkung von unterschiedlichen Confidence Threshold

Durch die obige Evaluation werden wir ab hier mit einem Threshold von 0.35 arbeiten. Daraus entsteht die Ausgangslage, welche in Tabelle 7.1 aufgeführt ist. Wie man erkennen kann, erzielen wir bereits sehr hohe Werte.

| Dataset | False Negative | False Positive | True Positive | Genauigkeit |
|---|---|---|---|---|
| Eingang 1 | 16 | 60 | 346 | 79.00% |
| Eingang 2 | 16 | 27 | 349 | 88.21% |
| Lift | 1 | 23 | 75 | 68.42% |
| Treppenhaus | 6 | 31 | 72 | 52.56% |
| Masoala Restaurant | 2 | 11 | 39 | 68.29% |
| Total | 41 | 152 | 881 | 79.06% |

Tabelle 7.1: Ausgangslage auf den unterschiedlichen Videosequenzen

### 7.3.2 Sprünge auf der Linie

Wie man in Tabelle 7.1 erkennen kann, gibt es deutlich mehr False Positives als False Negatives. Als erstes wollen wir also die False Positives reduzieren, möglichst ohne die False Negatives zu erhöhen. Wir haben die Ursachen der False Positives genauer untersucht und konnten feststellen, dass einige Personen sehr viele zusätzliche Zählungen in kurzer Zeit auslösen. Dies geschieht, da sich die Dimensionen der Bounding Boxen zwischen einzelnen Frames teilweise so stark verändern, dass der Mittelpunkt mehrere Male über die LOI springt. Auf eine korrekte Zählung (True Positive) kamen teilweise bis zu sechs fehlerhafte Zählungen (False Positive).

#### 7.3.2.1 Alter der Zählung

Um die vielen falschen Zählungen in kurzer Zeit zu reduzieren, implementieren wir eine Alterung der Zählungen. Wenn ein Tracking Objekt die LOI überquert, so wird sein Alter auf 0 gesetzt. Mit jeder Aktualisierung wird das Alter inkrementiert. Solange dieses unter einem bestimmten Grenzwert liegt, kann das Objekt nicht erneut gezählt werden. Abbildung 7.4 zeigt, wie sich die Anzahl an False Positives bereits bei einem Alterungswert von 2 stark reduziert. Bei einem Grenzwert von 6 konnte eine erneute Verbesserung erzielt werden. Die letzte Messung liegt bei einem Alterungswert von 10, wo wir die False Positives erneut reduzieren konnten. Die False Positives konnten um 77.6% (von 152 auf 34) reduziert werden, wobei die False Negatives nur um 9.75% anstiegen. Um die False Negatives nicht noch weiter zu erhöhen, wurden keine Messungen mit höheren Werten durchgeführt. Über unsere Datenbasis konnte die totale Genauigkeit von anfänglich 79.06% auf 91.43% verbessert werden.

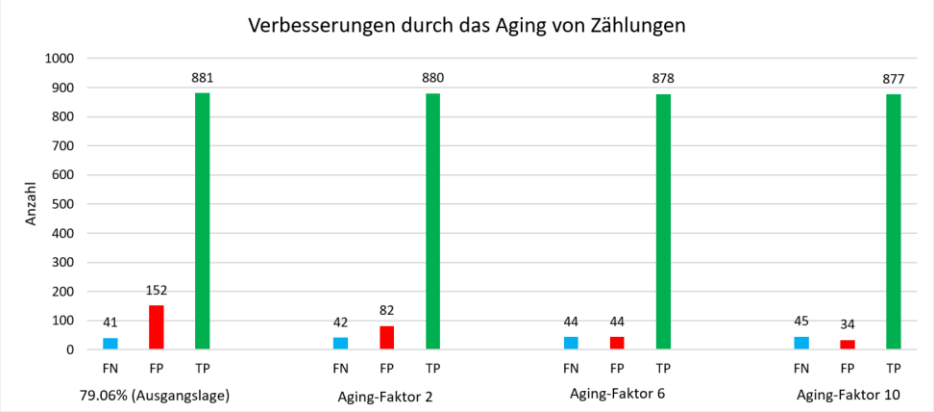


Abbildung 7.4: Reduktion der False Positives durch die Alterung der Zählungen

#### 7.3.2.2 Drei LOIs

Die Anzahl der False Positives konnte durch die Alterung der Zählung bereits stark verringert werden. Trotzdem gibt es noch immer bestimmte Fälle, in welchen eine Person sehr viele falsche Zählungen verursacht. Vor allem im Lift Video gibt es einen Edge Case, wo sich eine Person exakt auf die LOI setzt und bereits durch kleine Bewegungen sehr viele False Positives auslöst. Diese Situation ist in Abbildung 7.5 ersichtlich.

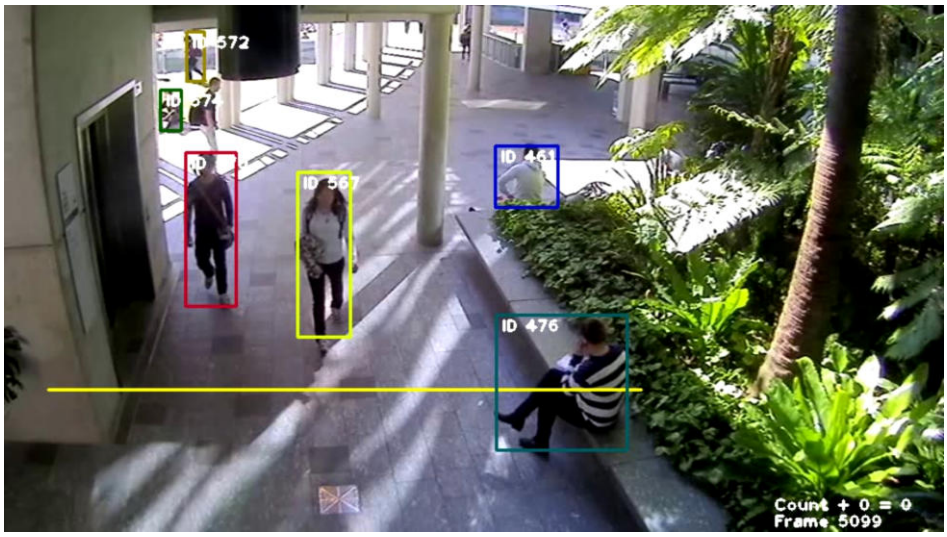


Abbildung 7.5: Person auf der LOI verursacht viele False Positives

Um zu verhindern, das solche Spezialfälle viele False Positives auslösen, implementieren wir drei beieinander liegende LOI. Um eine Zählung auszulösen, muss ein Objekt alle drei LOI überqueren. Somit werden nur Personen, welche

die Schranke normal überqueren, eine Zählung auslösen. Personen, die sich aus irgendeinem Grund direkt auf der LOI befinden, werden aber weniger False Positives verursachen. Wir haben mit unterschiedlichen Pixel Distanzen zwischen den LOI experimentiert. Die Ergebnisse dieser Untersuchungen sind in Abbildung 7.6 ersichtlich.

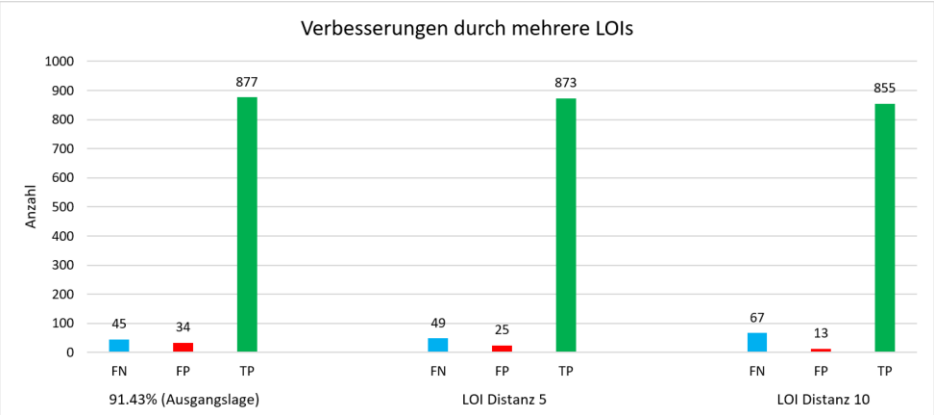


Abbildung 7.6: Reduktion der False Positives durch die Implementierung von mehreren LOIs

Die False Positives konnten auf Kosten der False Negatives erneut verringert werden. Mit einer LOI Distanz von 5 konnte die Genauigkeit von 91.43% auf 91.97% noch leicht verbessert werden. Bei einer Distanz von 10 sinkt die Metrik im Vergleich zur Ausgangslage allerdings leicht auf 91.32%. Zwar konnten die False Positives nochmals klar reduziert werden. Der starke Anstieg an False Negatives führt aber trotzdem zu einer Verschlechterung des Resultats. Somit entscheiden wir uns für eine LOI Distanz von 5 Pixeln.

Die False Positives konnten wir erfolgreich von ursprünglich 152 auf 25 reduzieren. Die False Negatives haben leicht von 41 auf 49 zugenommen. In den nächsten Schritten versuchen wir diese Anzahl zu reduzieren.

### 7.3.3 Verdeckungen

Vor allem bei den zwei Videos beim Eingang kommt es häufig vor, dass Personen die leicht verdeckt sind, bereits nicht mehr erkannt und somit auch nicht getrackt werden. Das führt zu sehr viel fehlenden Zählungen, sprich False Negatives. Bereits in Abschnitt 4.1.3 sind wir darauf eingegangen, dass die NMS dabei eine grosse Rolle spielen könnte. Aus diesem Grund haben wir Experimente mit unterschiedlichen NMS Thresholds durchgeführt. In den bisherigen Untersuchungen lag der Wert bei 0.25. Wir haben die Versuche mit 0.5 und 0.75 sowie ohne NMS durchgeführt. Ziel dieser Untersuchungen war es, den Threshold zu finden, welcher das Resultat aus dem vorherigen Abschnitt in Hinsicht auf die False Negatives verbessern kann.

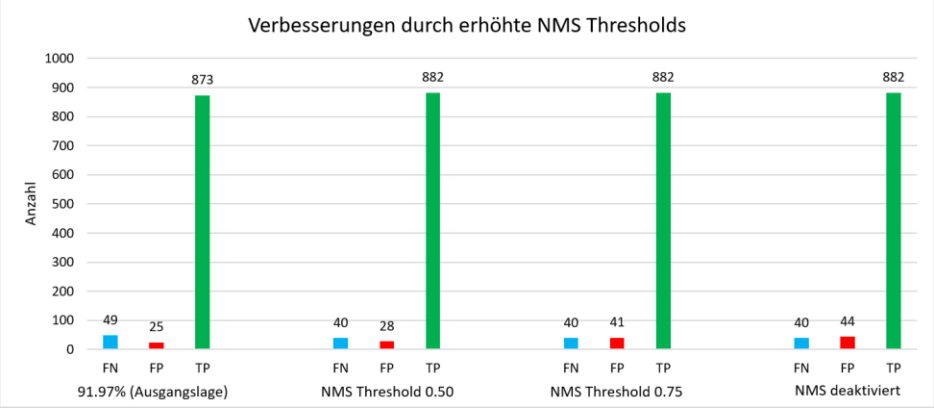


Abbildung 7.7: Ergebnisse mit unterschiedlichen NMS Threshold Werten

Abbildung 7.7 zeigt die Resultate, welche mit den unterschiedlichen NMS Thresholds erreicht werden konnten. Die Erhöhung von 0.25 auf 0.5 konnte die False Negatives von 49 auf 40 reduzieren. Die False Positives sind

dabei nur minimim um 3 angestiegen, was schlussendlich zu einer Genauigkeit von 92.62% führt. Die Erhöhung des Thresholds auf 0.7 und die komplette Deaktivierung der NMS führten gesamthaft zu schlechteren Resultaten.

## 7.4 Evaluation

Ziel der Evaluation ist es, zu bestimmen, ob unser Prototyp eine Genauigkeit von 80% erreicht. Diesen Wert wurde zu Beginn dieser Bachelorarbeit festgelegt. Wir haben unseren Prototyp auf die extrahierten, 90 minütigen Evaluationsdaten angewendet und überprüfen in diesem Abschnitt die dabei entstandene Zählung. Die in Abschnitt 7.1 genannten Evaluationsvideos wurden vom Prototypen noch nie verarbeitet. Wenn dieser auch hier eine Genauigkeit von 80% erreicht, so kann man davon ausgehen, dass der Prototyp in unterschiedlichen Situationen robust funktionieren wird.

Während der Optimierung haben wir einige Versuche durchgeführt, um die besten Konfigurationswerte für unseren Prototypen zu evaluieren. Für die Evaluation wurden somit die folgenden Werte verwendet:

- Skipping-Faktor: 2
- Confidence Threshold: 0.35
- NMS Threshold: 0.50
- Distanz zwischen den LOI: 5 Pixel
- Aging Faktor für Zählungen: 10

Für die Evaluation haben wir die Linienpositionen aus den Konfigurationen des SAIVT Datasets übernommen.

### 7.4.1 Auswertung

In der folgenden Tabelle 7.2 erkennt man die erreichten Zahlen und Resultate der Evaluation. Jedes Video wurde von uns manuell analysiert und die korrekte Anzahl bestimmt. Diese sind in der Tabelle mit "korrekt" vermerkt. Wie ersichtlich ist, haben wir das Ziel von 80% bei weitem übertroffen. Total aufsummiert erreichen wir eine Genauigkeit von 90.72%. Weiter lässt sich anhand der Spalten Differenz und Differenz korrekt feststellen, dass sich die Fehler teilweise gegenseitig aufheben, was zu einer noch geringerer Abweichung führt.

Die Genauigkeit von 90.72% erreichen wir auf Videosequenzen, welche für die Optimierungen nicht verwendet wurden. Somit kann man erkennen, dass wir mit diesen Optimierungen kein Overfitting des System produziert haben. Die Genauigkeit sollte somit auch auf anderen, neuen Situationen erreichbar sein.

| Dataset | hoch/runter | Differenz | hoch/runter kor. | Differenz kor. | FN | FP | TP | Genauigkeit |
|---|---|---|---|---|---|---|---|---|
| äussere Treppen | 295/299 | -4 | 304/316 | -12 | 35 | 9 | 585 | 92.90% |
| Weg z. Eingang | 385/464 | -79 | 396/469 | -73 | 47 | 31 | 817 | 90.97% |
| Hauptweg 1 | 194/111 | 83 | 201/131 | 70 | 20 | 13 | 292 | 89.42% |
| Hauptweg 2 | 110/144 | -34 | 119/154 | -35 | 27 | 8 | 246 | 87.17% |
| äusserer Lift | 370/388 | -18 | 370/388 | -18 | 33 | 33 | 659 | 90.46% |
| Total | 1354/1406 | -52 | 1390/1458 | -68 | 162 | 94 | 2599 | 90.72% |

Tabelle 7.2: Aufstellung der Zuverlässigkeiten des Prototyps

Durch die kleinen Differenzen zwischen der Zählung des Prototyps und der korrekten Zählung kann man zudem erkennen, dass sich die Fehler tatsächlich aufheben. Somit bestätigt sich unsere Vermutung aus Abschnitt 3.2.1 und die dort beschriebene Fehleraufsummierung ist tatsächlich kein grosses Problem.

# 8 Installation

Wie setzt man unseren Counter nun konkret vor Ort ein? Für unsere Abschlusspräsentation haben wir einen Demonstrator entwickelt. Dieser könnte als Inspiration für zukünftige Weiterentwicklungen und eine allfällige Installation im Zoo dienen. Weiter gehen wir in diesem Kapitel auf die Themen Datenschutz und Kosten einer Installation ein.

In Abbildung 8.1 haben wir die Integration des Zählers in das Demo-Gesamtsystem dargestellt. Wie wir in Kapitel 6 beschrieben haben, benötigt die Verarbeitung eines Frames ca. 29 ms. Um Realtime Zählungen durchführen zu können, wird somit ein Counter pro Kamerastream benötigt. Im gleichen Kapitel haben wir weiter festgestellt, dass bis zu drei Counter auf unserer Mittelklasse Grafikkarte laufen gelassen werden können, was die Parallelisierung auf einem Rechner erlaubt. Die Counter haben wir als Prozesse modelliert. Wir haben uns gegen Threads entschieden, da auf CPython (Referenzinterpreter von Python) nicht mehrere Threads gleichzeitig laufen können. Dies ist dem Global Interpreter Lock (GIL) geschuldet. [55] Weiter spricht für eine Modellierung als Prozesse, dass diese so potenziell über mehrere Rechner verteilt werden könnten. So könnte z.B. Edge Computing betrieben und die Bilder bereits bei der Kamera ausgewertet werden.

Die Coordinator Komponente ist zuständig für das Aufsummieren der Veränderungen. Die Anzahl Personen pro Restaurant werden somit in dieser Komponente bestimmt. Das Aufsummieren könnte für die Demo auch in der dritten Komponente, dem Webdashboard, erfolgen. Dies ist jedoch nicht mehr sinnvoll, wenn Besucher via App auf die Informationen zugreifen möchten. Der Coordinator könnte auch der Authentication und Authorisation dienen.

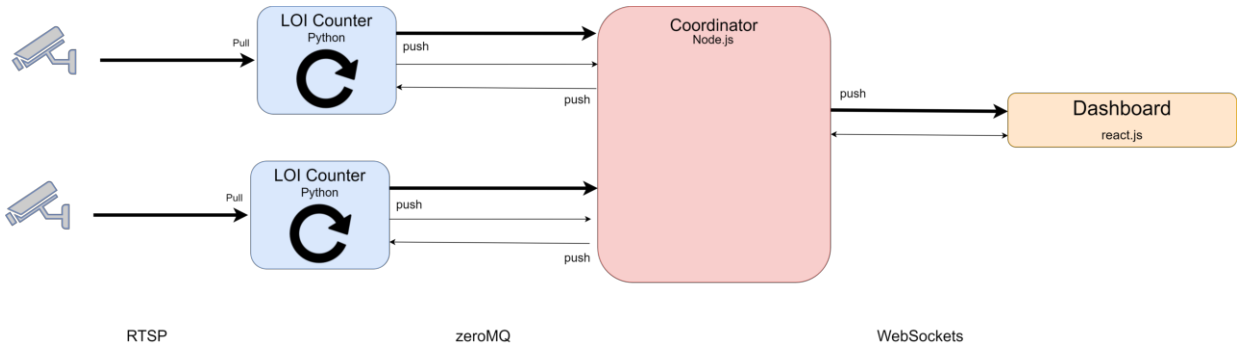


Abbildung 8.1: Übersicht über das Gesamtsystem

Die dritte Komponente, das Webdashboard, stellt die Managementbenutzeroberfläche dar. In Abbildung 8.2 ist diese als Screenshot ersichtlich. Die Kamerabilder, auf welchen die Bounding Boxen der Personen eingezeichnet sind, werden an die Single Page Application übertragen und dort dargestellt. Aufgrund der Bilder kann man die Linie nachjustieren. Diese Kalibration ist, wie wir in Abschnitt 8.2 noch weiter beschrieben, bei der Installation notwendig, um gute Zählresultate zu erhalten. Für die Kalibration stehen Buttons zur Verfügung, mit denen man die Linie vertikal und horizontal verschieben kann. Weiter kann man die Länge mit dem Plus und Minus Button anpassen. Die aktuelle Auslastung, sowie die generelle Kapazität eines Raumes wird oben rechts dargestellt. Die Auslastung als Ampel haben wir rechts mit den drei farbigen Kreisen visualisiert.

Für die Demonstration führen wir alle drei Komponenten auf demselben Rechner aus. Wir verwenden denselben Computer wie im Benchmarking (siehe Abschnitt 5.5). Die Counter können parallel auf der Grafikkarte ausgeführt werden.

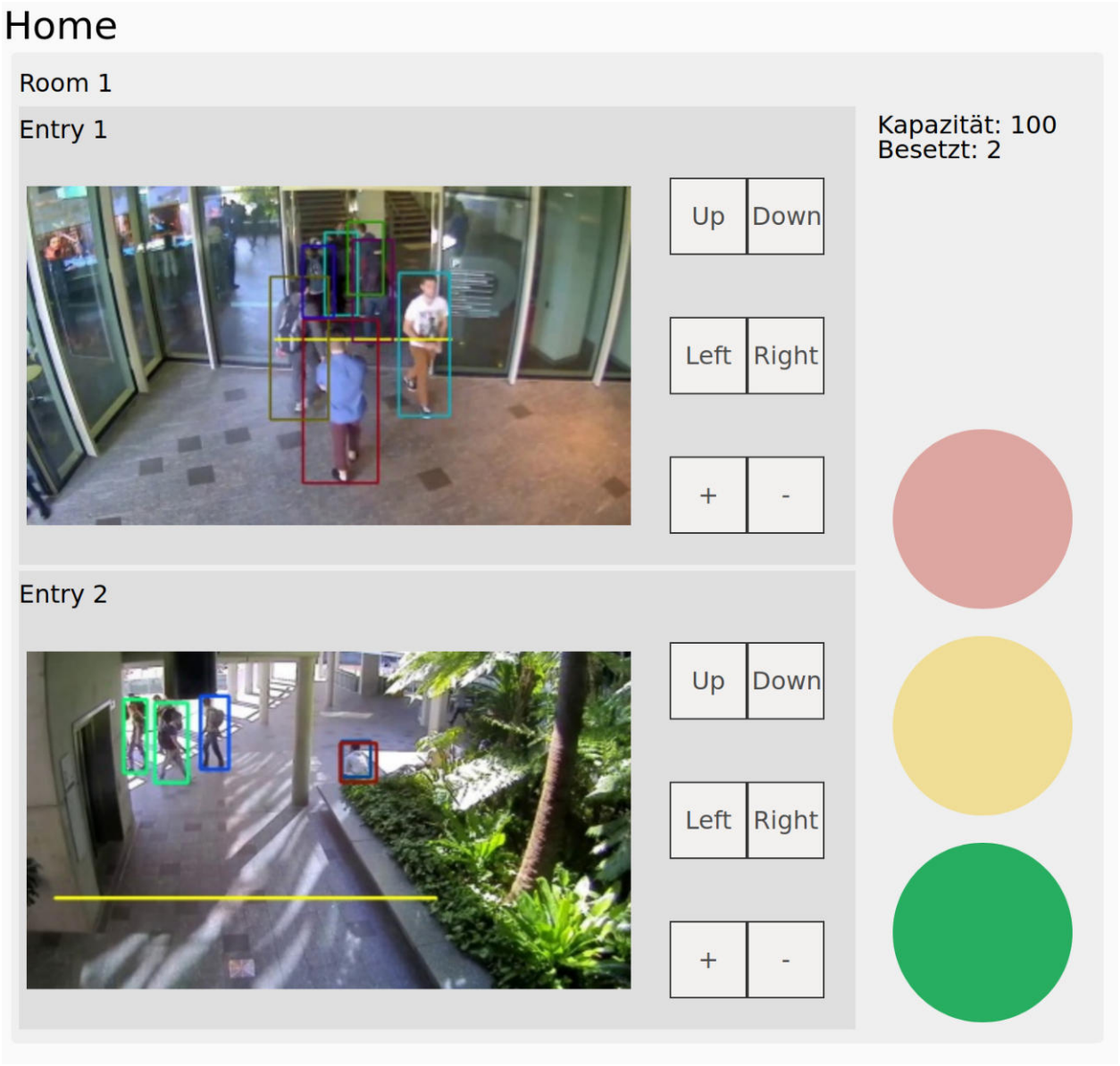


Abbildung 8.2: Screenshot des Frontends

## 8.1 Kommunikation

Für die Kommunikation zwischen den einzelnen Komponenten werden verschiedene Netzwerkprotokolle, wie in Abbildung 8.1 dargestellt, verwendet. Die Zähler pullen die Frames von den Kameras via Real Time Streaming Protocol (RTSP). RSTP lässt sich einfach mit OpenCV verwenden, in dem man die IP, Username, Passwort und Port angibt. RSTP wird von vielen Kameras unterstützt, unter anderem auch von den uns verwendeteten Burgwächter DOME 303 [56]. Die Kameras buffern die Frames. Somit führen einzelne Performanceausreisser (Verarbeitungsdauer von über 33ms) zu keinem Frameverlust.

Die Interprozesskommunikation zwischen den Counters und dem Coordinator erfolgt mit Hilfe der ZeroMQ Library. So können wir Message Queues verwenden, um asynchron zwischen den Komponenten zu kommunizieren. ZeroMQ kommt im Gegensatz zu anderen Messaging Systemen wie RabbitMQ ohne Broker aus. Ein Broker wäre ein zusätzlicher Prozess, der als zentrale Instanz die Message Queues überwacht und gegebenenfalls Nachrichten perisistiert, wenn Empfänger nicht erreichbar sind. Wir verzichten darauf, um unseren Prototypen schlank zu halten und nicht weiteren Netzwerkverkehr zu erhalten.

In Abbildung 8.1 sind die drei Message Queue Verbindungen pro Counter dargestellt. Der Counter fungiert als Publisher für die Image und Zähler Queue. Er pushed so seine Nachrichten an den Coordinator. Als Subscriber erhält der Counter Updates betreffend Linieneinstellungen. Dadurch dass er beim Versenden der Frames und Zählungen nicht auf Acknowledgement Nachrichten warten muss, wird er nicht an einer konstanten Frameverarbeitung gehindert. Die fehlenden Acknowledgements führen jedoch dazu, dass Nachrichten verloren gehen können. Bei der Übertragung der Frames (dicke Linien in Abbildung) ist dies weniger kritisch, da diese nur als Hilfe für die Konfiguration dienen. Bei den Zählerupdates ist dies jedoch wichtiger. Aus Zeitgründen haben wir dies so gelöst, dass die Zähler die Werte ebenfalls aufsummieren und jeweils ihr Total übertragen. So erhält der Coordinator die richtige Zahl beim nächsten Update, auch wenn eine Nachricht zuvor verloren gegangen ist.

Ein weiterer Punkt sind die vielen Frames, die dem Coordinator übertragen werden. Um zu verhindern, dass der Coordinator mit Bildern überschwemmt wird, haben wir den Queuebuffer limitiert. ZeroMQ nennt diese Limitation Highwater Mark [57]. Wenn die Limite erreicht ist, verwirft der Publisher die Nachrichten. Standardmässig ist kein Limit gesetzt.

Die Kommunikation zwischen Coordinator und Dashboard erfolgt mit der Kommunikationslibrary socket.io, welche WebSockets als Protokoll verwendet. Wir haben uns dafür entschieden, da ZeroMQ nicht direkt in einer clientseitigen Applikation verwendet werden kann. WebSockets haben gegenüber HTTP den Vorteil, dass die Verbindung zwischen Browser und Coordinator offen bleibt und so Änderungen vom Coordinator gepushed werden können.

### 8.1.1 Asynchrone Modellierung der Zähler

Damit die Zähler beim Warten auf Nachrichten aus der Settingsqueue nicht blockieren, haben wir die Applikation asynchron modelliert. Seit Python 3.4 gibt es das Async IO Package. Mit Hilfe dieses Packages und der `async/await` Syntax, die man aus anderen Programmiersprachen kennt, haben wir die Asynchronität umgesetzt. Async IO basiert anders als z.B. C# auf Koroutinen und der kooperativen Multitasking Idee. Dies bedeutet, anstelle, dass das Betriebssystem das Scheduling übernimmt, müssen die Tasks die Kontrolle selbständig abgeben. Wenn diese nie blockieren, laufen diese unendlich lange weiter und andere Tasks kommen nie zur Ausführung. [55]

## 8.2 Position der Linie

Sowohl während den Optimierungen als auch in der Evaluation konnten wir feststellen, wie sehr die Zählung von der Position der LOI abhängt. Die perfekte Position für die Linie zu finden ist beinahe unmöglich. Fehler lassen sich nicht ausschliessen und werden immer auftreten. Dennoch möchten wir einige Überlegungen dazu anstellen. Wir erklären, was wir im Laufe dieser Bachelorarbeit erkannt haben und was es bei der Wahl der LOI Position zu beachten gibt, um eine möglichst präzise Zählung zu ermöglichen.

Grundsätzlich kann die LOI beliebig positioniert werden. Die Zählung funktioniert sowohl bei einer waagrechten als auch bei einer senkrechten Ausrichtung. Somit ist es auch möglich Personen zu zählen, wenn die Kamera nicht frontal auf den Weg oder Eingang gerichtet ist. Allerdings konnten wir während der Implementation feststellen, dass die Anzahl der False Negatives dadurch ansteigt. Dies ist darauf zurück zu führen, dass es öfter zu Verdeckungen kommt und die Object Detection dadurch beeinträchtigt wird. Wir empfehlen darum, wann immer möglich, die Kamera frontal auf den Weg oder Eingang zu richten um die LOI waagrecht positionieren zu können.

Die Linie darf auf keinen Fall zu nahe am Rand des Bildausschnitts sein. Sollte dies der Fall sein, so kann es vorkommen, dass die Object Detection nicht genügend Zeit hat, um die bestätigte Tracking Objekte zu instantiieren, bevor die Person die LOI überquert hat. Die Gefahr von False Negatives wird grösser, je näher die LOI sich am Rand befindet. Dazu kommt, dass sich die Bounding Box in kurzer Zeit stark verändern kann, wenn beispielsweise der Kopf oder die Füsse einer Person nicht mehr im Bild sind. Durch diese Veränderungen können ebenfalls fehlerhafte Zählungen ausgelöst werden, wenn sich die LOI zu nahe am Bildrand befindet.

Weiter zu beachten ist, dass sich Personen bei Türen oder Eingängen häufig gegenseitig verdecken. Vor bzw. nach dem Eingang haben Personen meist einen grösseren Abstand zueinander, was die Gefahr einer Verdeckung verringert. Aus diesem Grund empfehlen wir, die LOI nicht genau auf dem Eingang sondern etwas davor zu positionieren.

Wichtig ist, dass die LOI möglichst auf Durchgängen positioniert wird, wo Personen keinen Grund haben stehen bleiben. Bleiben Personen exakt auf der LOI stehen, so besteht die Gefahr von False Positives. Die Alterung welche wir in den Optimierungen implementieren, kann das nur bedingt verhindern. Abbildung 8.3 zeigt ein Beispiel für eine solche Situation im Evaluationsvideo von Hauptweg 2.


Abbildung 8.3: Schlechte Linienpositionen führen zu fehlerhaften Zählungen

In dieser Situation betrachtet eine Personengruppe für einige Zeit die Informationstafel am linken Bildrand. Einzelne Verdeckungen und kleine Bewegungen der Personen führen zu Veränderungen der Bounding Boxen. Dadurch werden hier zwei False Positives verursacht. Die rote Linie markiert eine bessere Positionierung, welche weitere solche Fehler verhindern würde.

Da sich die Situation von Anwendung zu Anwendung anders darstellt, empfehlen wir einige Zeit in die optimale Positionierung der LOI zu investieren.

## 8.3 Datenschutz

Bevor eine Installation vorgenommen werden kann, muss der Datenschutz sichergestellt sein. Wir sind keine Juristen und sind somit nicht im Stande rechtliche Aussagen betreffend Datenschutz zu machen. Wir können jedoch einen Kommentar aus technischer Sicht abgeben.

Die Eingänge von Gebäude wie Restaurants werden gelegentlich bereits mit Überwachungskameras überwacht. Beim Einsatz dieser werden die Aufnahmen für eine bestimmte Zeitdauer gespeichert, um bei Diebstählen etc. beigezogen zu werden. Unsere Lösung benötigt keine Speicherung der Aufnahmen (auch nicht temporär), sondern verarbeitet diese fortlaufend. Unser Zähler sollte somit überall verwendet werden können, wo auch Überwachungskameras erlaubt sind. Weiter erhalten keine Drittanbieter Zugang zu den Bilddaten, da wir keine Clound Angebote nutzen. Dies wäre beim Einsatz der Azure Computer Vision API [40] anders. Die Personenerkennung, sowie das Tracking erfolgt rein lokal auf einem Rechner. Zudem verwenden wir keine Face Recognition Algorithmen oder ähnliches, weshalb uns die Identität der Personen nie bekannt wird. Durch all diese Punkte sind wir der Meinung, dass unser System nicht anders als eine Lichtschranke oder andere Sensoren behandelt werden muss.

## 8.4 Kosten

Gerade im Hinblick auf die Skalierbarkeit und die produktive Verwendung dieses Prototypen, ist eine Kostenzusammestellung sowohl für die Noser Engineering als auch für den Zoo Zürich von Interesse.

- Eine GeForce GTX 1080 mit 8 GB Grafikspeicher kostet aktuell zwischen 550.- und 600.- CHF.
- Die beiden Burgwächter Kameras, welche im Demoaufbau verwendet werden, kosteten jeweils 129.90 CHF.
- Hinzu kommen noch die restlichen Komponente des verwendeten Computers. Eine ähnliche CPU kostet aktuell noch gut 330.- CHF. Bezüglich Arbeitsspeicher vermuten wir, dass die selbe Leistung auch mit deutlich weniger RAM erreicht werden kann. Die Kosten für 32 GB Arbeitsspeicher belaufen sich aktuell auf rund 150.- CHF.

Die Kosten für den ganzen Demoaufbau, inklusive weiterer Komponenten wie Mainboard oder CPU-Kühler, belaufen sich damit auf ca. 1750.- CHF. Dazu kommen noch weitere Kosten für die Installation und den Unterhalt des gesamten Systems.

Dieser Preis bezieht sich auf die Verwendung von zwei Kameras. Mit dem hier beschriebenen Computer ist die Verwendung von mehr Kameras aus Gründen der Performance nicht möglich. Um mehr Eingänge und Restaurants zu überwachen, wird also neben mehr Kameras auch ein zweiter oder sogar dritter Computer benötigt, was die Kosten weiter erhöht.

# 9 Schlussfolgerung

Aufgrund der COVID-19 Pandemie und der damit verbundenen Schliessung des Zoos konnten wir leider keine Evaluation unseres Prototypen vor Ort durchführen. Die Resultate, die wir auf unserem zusammengesetzten Datensatz erzielen konnten, stimmen uns jedoch optimistisch. Wir haben auf den Videos eine Genauigkeit von 90.72% erzielt, was das gesetzte Ziel von 80% übertrifft. Somit sehen wir durchaus Potenzial für ein videobasiertes Zählsystem.

Wir haben uns bei dieser Arbeit stark auf den Object Detector Algorithmus konzentriert, da dieser für unser System essentiell ist. Aus Zeitgründen ist dabei die Evaluation des Tracking Algorithmus ein wenig zu kurz gekommen. Hier hätten wir sicher noch mehr vergleichen können. Es ist jedoch auch schwierig sich einen Überblick zu beschaffen, da es keine aktuelle Review Paper gibt.

Weiter hätte sich das Senden der Zählupdates potenziell robuster lösen lassen. Wie in Abschnitt 8.1 beschrieben, führen die einzelnen Counters ebenfalls einen State, der sie bei jedem Update übertragen. Falls eine Zählung nicht bei der Coordinatorkomponente ankommt, wird die Änderung beim nächsten Event mitgesendet. Dieses Vorgehen ist immer noch fehleranfällig. Zum Beispiel kann der zeitliche Abstand zwischen einer verlorenen Nachricht und der nächsten empfangenen Nachricht gross sein, was dazu führt, dass die User für eine lange Zeit einen alten Stand sehen.

## 9.1 Ausblick

In unserer Bachelorarbeit legen wir mit dem Prototypen eine Grundlage für weitere Projekte in diesem Bereich. Es gibt noch viel Potenzial für Optimierungen und Erweiterungen. Auf diese gehen wir in diesem Abschnitt ein.

### 9.1.1 Fehleraufsummierung

In Abschnitt 3.2.1 haben wir das Problem der Fehlersummierung bereits angesprochen. Unsere Vermutung, dass sich Fehler in beide Richtungen über die Schranke aufheben, konnte in Abschnitt 7.4 bestätigt werden. Dennoch lohnt es sich, Überlegungen zu einer allfälligen Optimierung anzustellen. Im Zoo macht es durchaus Sinn, den Zähl-Prozess jeweils am Morgen neu zu starten. Dadurch würde der Zähler auf 0 gesetzt und alle somit auch alle fehlerhaften Zählungen eliminiert. Mit Alexander Gassmann haben wir noch weitere Ideen diskutiert:

- Die Aufenthaltsdauer der Besucher im Restaurant könnte miteinbezogen werden. Mit Hilfe eines mathematischen Modells könnten nicht gezählte Austritte korrigiert werden.
- Eine zweite Idee ist, den Counter mit anderen Sensoren zu kombinieren. Beispielsweise könnte man dafür den Lärmpegel innerhalb eines Raums messen und einen Abgleich mit den gezählten Personen machen. Eine weitere Idee wäre die Transaktionen der Kassen im Restaurant heranzuziehen. Damit könnte ebenfalls überprüft werden, ob die registrierten Zahlungen den gezählten Leuten entsprechen. Die Tippgeschwindigkeit oder die Abstände zwischen den Transaktionen könnten ebenfalls hilfreiche Metriken sein.

### 9.1.2 Performance

Für unseren Prototypen haben wir Python als Programmiersprache verwendet. Python gilt im Vergleich zu hardwarenahen Programmiersprachen als langsam. Wir sind trotzdem der Meinung, dass ein Rewriting des Prototypen in einer Programmiersprache wie C++ nicht viel Performancegewinn verspricht, da die verwendeten Libraries wie OpenCV und NumPy bereits stark optimiert sind und die Inferenzzeit des Object Detectors der grösste Anteil an der Verarbeitungszeit ausmacht.

Die Inferenzzeit lässt sich nur mit einem schnelleren Modell noch weiter verkleinern. Was man aus Entwicklersicht jedoch noch tun könnte, wäre das Timing der Detection intelligenter zu gestalten. Zum Beispiel in dem man die Object Detection nur laufen lässt, wenn sich der Tracker nicht mehr sicher über die Lokalisierung eines Objektes ist oder wenn potenziell neue Objekte im Bild erscheinen. Ding und Wong schlagen ein solche Confidence Trigger

Detection in ihrem Paper vor. [58]

Wenn man trotzdem noch mehr Performance aus dem Python Code rauskitzeln möchte, könnte man anstelle von CPython PyPy verwenden. Der PyPy Interpreter nutzt einen Just In Time Compiler, welcher viel ausgeführten Code zur Laufzeit in Maschinencode umwandelt.

### 9.1.3 Tracker Netzwerk

Für unseren Prototypen verwenden wir zwei verschiedene neuronale Netzwerke. Die Object Detection erfolgt mit dem YOLO Algorithmus und der Tracker verwendet ein Netzwerk um Features für die ID-Zuordnung zu generieren. Das Detection Modell haben wir über das OpenCV-DNN Modul eingebunden. Aus Software Engineering Perspektive macht es Sinn auch den Feature Extractor des Trackers mit Hilfe der OpenCV Library zu verwenden. Die aktuell bestehenden Abhängigkeiten zu Tensorflow könnten so eliminiert werden. Die Verwendung von Tensorflow führt zu sehr vielen weiteren Abhängigkeiten, die mitinstalliert werden. Diese würden dann ebenfalls entfallen.

Die Portierung des Modells auf das OpenCV-DNN Modul ist nicht ganz trivial, da zuerst die Kompabilität anaylisiert werden muss. Das Modul unterstützt nicht alle Tensorflow Funktionalitäten. Weiter ist die Generierung einer Konfigurationsdatei notwendig. [59]

### 9.1.4 Linien und weitere Formen

Wie in Abschnitt 8.2 beschrieben, eignen sich horizontale Linien am besten für die Zählung. Unser Prototyp haben wir dementsprechend auch nur für eine horizontale Ausrichtung implementiert. Den Prototypen könnte man nun um beliebige Linienausrichtungen erweitern und dementsprechend Controls für z.B. Rotationen anbieten. Der Abstand zwischen den einzelnen Linien könnte man ebenfalls flexibel konfigurierbar machen. Wir haben in Abschnitt 7.3.2.2 diesen Wert mit 5 Pixeln bestimmt. Eventuell ist dies nicht der ideale Wert für jede Situation.

Hegner et al. [3] bieten mit ihrer Lösung neben Linien auch verschiedene Polygone an, mit deren Hilfe gezählt werden kann. Unser Counter liesse sich ebenfalls um weitere Formen erweitern.

### 9.1.5 Weitere Verwendungszwecke

Der Use Case liesse sich auch auf den ganzen Zoo ausweiten. Wenn man den ganzen Zoo in Sektoren unterteilen würde, könnte man so die Verteilung der Gäste im gesamten Zoo messen. Dafür müssten die Grenzen dieser Sektoren mit Kameras überwacht werden. Die daraus gewonnenen Daten liessen sich als Heatmap auf der Zoo Karte visualisieren. Um den ganzen Zoo in solche Sektoren unterteilen zu können, wären aber deutlich mehr Kameras und mehrere Rechnerinstanzen zur Verarbeitung der vielen Videostreams notwendig.

Unsere Lösung könnte auch für Anwendungen ausserhalb des Zoos interessant sein. Während der COVID-19 Pandemie müssen Einkaufsläden und andere öffentliche Einrichtungen vermehrt auf die Anzahl der Personen achten, welche sich in einem bestimmten Bereich befinden. Dabei gibt es eine maximale Anzahl welche von der Gesamtfläche des Raums abhängig ist. Das von uns implementierte Ampelsystem bietet eine einfache Visualisierung mit welcher ersichtlich ist, ob diese Maximalzahl bereits überschritten ist oder nicht.

# Danksagung

Wir kamen ohne jegliches Vorwissen in ein äusserst spannendes, sehr komplexes Gebiet und durften viele neue Dinge lernen. Wir bedanken uns bei allen Personen, die uns während dieser Bachelorarbeit unterstützt haben. Insbesondere bei unserem fachlichen Betreuer Prof. Dr. Luc Bläser und bei Alexander Gassmann von der Noser Engineering AG. Sie unterstützten uns tatkräftig und konnten einige sehr hilfreiche Hinweise und Inputs geben. Weiter bedanken wir uns bei Dominik Ryser vom Zoo Zürich für die unkomplizierte Zusammenarbeit. Bei Martin Stypinski bedanken wir uns für seine wertvollen Ratschläge und Hinweise sowie die Review unseres Codes. Ebenfalls danken wir Christoph Amrein für seine Code Review und Silvan Gehrig für die Ratschläge während dieser Arbeit. Bei allen involvierten Personen der Noser Engineering möchten wir uns ebenfalls bedanken. Ebenfalls bedanken wir uns bei unserem Experten Dr. Janos Zatonyi. Und zu guter Letzt haben sich Laura Maier, David Styger und Rita Bucher die Mühe gemacht und unsere Arbeit gegen gelesen. Besten Dank.

# Glossar

**AP** Average Precision. 26–28, 50

**API** Application Programming Interface. 5, 9

**BLOB** Binary Large Object. 16

**CNN** Convolutional Neural Network. 4, 5, 18, 19, 22

**CPU** Central Processing Unit (Prozessor). 26

**DBT** Detection-Based Tracking. 17

**DFT** Detection-Free Tracking. 17

**DNN** Deep Neural Networks. 15, 23

**FPS** Frames per Second. 22, 31

**IoU** Intersection over Union. 16, 17, 27

**LOI** Line of Interest. 4, 7, 11–14, 19–21, 29, 30, 36–39, 42, 43, 49

**MOT** Multiple Object Tracking. 17, 18

**NMS** Non Maxima Suppression. 16, 20, 38, 39, 49

**ROI** Region of Interest. 4, 5, 7, 10, 12, 13, 49

**SOT** Single Object Tracking. 17, 18

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Literaturverzeichnis


[1] R. Tanner u. a. „People Detection and Tracking with TOF Sensor“. In: *2008 IEEE Fifth International Conference on Advanced Video and Signal Based Surveillance*. 2008, S. 356–361.

[2] HuanSheng Song u. a. „Benchmark data and method for real-time people counting in cluttered scenes using depth sensors“. In: *CoRR* abs/1804.04339 (2018). arXiv: 1804.04339. URL: http://arxiv.org/abs/1804.04339.

[3] Robert Hegner u. a. „Scalable, self-organizing 3D camera network for non-intrusive people tracking and counting“. In: Okt. 2014, S. 3405–3407. DOI: 10.1109/ICIP.2014.7025689.

[4] Javier Barandiarán, Berta Murguia und Fernando Boto. „Real-Time People Counting Using Multiple Lines“. In: Juni 2008, S. 159–162. ISBN: 978-0-7695-3344-5. DOI: 10.1109/WIAMIS.2008.27.

[5] Byeoung-su Kim u. a. „A Method of Counting Pedestrians in Crowded Scenes“. In: Bd. 5227. Sep. 2008, S. 1117–1126. DOI: 10.1007/978-3-540-85984-0_134.

[6] Lijun Cao u. a. „Large scale crowd analysis based on convolutional neural network“. In: *Pattern Recognition* 48.10 (2015). Discriminative Feature Learning from Big Data for Visual Recognition, S. 3016–3024. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2015.04.001. URL: http://www.sciencedirect.com/science/article/pii/S0031320315001259.

[7] Simon Denman u. a. „Scene Invariant Virtual Gates using DNNs“. In: *IEEE Transactions on Circuits and Systems for Video Technology* PP (Okt. 2018), S. 1–1. DOI: 10.1109/TCSVT.2018.2874649.

[8] A. B. Chan, Zhang-Sheng John Liang und N. Vasconcelos. „Privacy preserving crowd monitoring: Counting people without people models or tracking“. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, S. 1–7.

[9] Victor Lempitsky und Andrew Zisserman. „Learning To Count Objects in Images.“ In: Jan. 2010, S. 1324–1332.

[10] Prithvijit Chattopadhyay u. a. „Counting Everyday Objects in Everyday Scenes“. In: *CoRR* abs/1604.03505 (2016). arXiv: 1604.03505. URL: http://arxiv.org/abs/1604.03505.

[11] *Azure Cognitive Services*. URL: https://azure.microsoft.com/en-us/services/cognitive-services/. (Zugegriffen am 17.02.2020).

[12] *Neural Networks and Deep Learning (Course 4 of the Deep Learning Specialization)*. URL: https://www.youtube.com/playlist?list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF. (Zugegriffen am 28.03.2020).

[13] *Detect common objects in images*. URL: https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/concept-object-detection. (Zugegriffen am 17.02.2020).

[14] *YOLO: Real-Time Object Detection*. URL: https://pjreddie.com/darknet/yolo/. (Zugegriffen am 17.02.2020).

[15] *COCO - Common Objects in Context*. URL: http://cocodataset.org/. (Zugegriffen am 28.03.2020).

[16] Jiankang Deng u. a. „RetinaFace: Single-stage Dense Face Localisation in the Wild“. In: *CoRR* abs/1905.00641 (2019). URL: http://arxiv.org/abs/1905.00641.

[17] *Deep Learning in OpenCV*. URL: https://github.com/opencv/opencv/wiki/Deep-Learning-in-OpenCV. (Zugegriffen am 15.04.2020).

[18] *Function Documentation - blobFromImage()*. URL: https://docs.opencv.org/3.4/d6/d0f/group__dnn.html#ga29f34df9376379a603acd8df581ac8d7. (Zugegriffen am 01.05.2020).

[19] *Deep learning: How OpenCV's blobFromImage works*. URL: https://www.pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/. (Zugegriffen am 01.05.2020).

[20] *Non-Maximum Suppression for Object Detection in Python*. URL: https://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-python/. (Zugegriffen am 01.05.2020).

[21] *Coarse Gaze Estimation in Visual Surveillance*. URL: http://www.robots.ox.ac.uk/ActiveVision/Research/Projects/2009bbenfold_headpose/project.html#datasets. (Zugegriffen am 17.02.2020).

[22] *YOLO object detection with OpenCV*. URL: https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/. (Zugegriffen am 01.05.2020).

[23] Luo Wenhan, Zhao Xiaowei und Kim Tae-Kyun. „Multiple Object Tracking: A Review". In: *CoRR* (2014). URL: http://arxiv.org/abs/1409.7618.

[24] *OpenCV Tracking API*. URL: https://docs.opencv.org/3.4/d9/df8/group__tracking.html. (Zugegriffen am 24.05.2020).

[25] *Dlib Library*. URL: http://dlib.net/. (Zugegriffen am 24.05.2020).

[26] *Simple object tracking with OpenCV*. URL: https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/. (Zugegriffen am 11.06.2020).

[27] Nicolai Wojke, Alex Bewley und Dietrich Paulus. „Simple Online and Realtime Tracking with a Deep Association Metric". In: *CoRR* abs/1703.07402 (2017). URL: http://arxiv.org/abs/1703.07402.

[28] R. E. Kalman. „A New Approach to Linear Filtering And Prediction Problems". In: *ASME Journal of Basic Engineering* (1960).

[29] Nicolai Wojke. *Deep SORT*. URL: https://github.com/nwojke/deep_sort. (Zugegriffen am 20.03.2020).

[30] Tsung-Yi Lin u. a. „Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). URL: http://arxiv.org/abs/1405.0312.

[31] Licheng Jiao u. a. „A Survey of Deep Learning-based Object Detection". In: *CoRR* abs/1907.09408 (2019). arXiv: 1907.09408. URL: http://arxiv.org/abs/1907.09408.

[32] *Tensorflow Detection Model Zoo*. URL: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. (Zugegriffen am 15.04.2020).

[33] *Caffe Model Zoo*. URL: https://github.com/BVLC/caffe/wiki/Model-Zoo. (Zugegriffen am 15.04.2020).

[34] *ONNX Model Zoo*. URL: https://github.com/onnx/models. (Zugegriffen am 15.04.2020).

[35] Joseph Redmon u. a. „You Only Look Once: Unified, Real-Time Object Detection". In: *arXiv* (2015).

[36] Joseph Redmon und Ali Farhadi. „YOLO9000: Better, Faster, Stronger". In: *CoRR* abs/1612.08242 (2016).

[37] Joseph Redmon und Ali Farhadi. „YOLOv3: An Incremental Improvement". In: *arXiv* (2018).

[38] Wei Liu u. a. „SSD: Single Shot MultiBox Detector". In: *CoRR* abs/1512.02325 (2015).

[39] Kaiming He u. a. „Mask R-CNN". In: *CoRR* abs/1703.06870 (2017). URL: http://arxiv.org/abs/1703.06870.

[40] *Azure Computer Vision*. URL: https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/. (Zugegriffen am 17.02.2020).

[41] Anton Milan, Konrad Schindler und Stefan Roth. „Challenges of Ground Truth Evaluation of Multi-Target Tracking". In: (). URL: https://ethz.ch/content/dam/ethz/special-interest/baug/igp/photogrammetry-remote-sensing-dam/documents/pdf/anton13cvprw.pdf.

[42] *RGB-D Pedestrian Dataset*. URL: https://www.epfl.ch/labs/cvlab/data/data-rgbd-pedestrian/. (Zugegriffen am 30.03.2020).

[43] *2D MOT 2015*. URL: https://motchallenge.net/data/2D_MOT_2015/. (Zugegriffen am 30.03.2020).

[44] *MOT16*. URL: https://motchallenge.net/data/MOT16/. (Zugegriffen am 30.03.2020).

[45] Mark Everingham u. a. „The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 88 (2009), S. 303–338.

[46] *Precision-Recall*. URL: https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#precision-recall. (Zugegriffen am 01.06.2020).

[47] Guido Schuster. *Vorlesungs- und Übungsunterlagen Deep Learning*. Hochschule für Technik, Rapperswil. Durchführung 2020.

[48] Adrian Rosebrock. *How to use OpenCV's 'dnn' module with NVIDIA GPUs, CUDA, and cuDNN*. URL: https://www.pyimagesearch.com/2020/02/03/how-to-use-opencvs-dnn-module-with-nvidia-gpus-cuda-and-cudnn/. (Zugegriffen am 07.04.2020).

[49] Timothy Carlen. *Understanding the mAP Evaluation Metric for Object Detection*. URL: https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3. (Zugegriffen am 13.04.2020).

[50] *SAIVT-VirtualGate Database*. URL: https://github.com/qutsaivt/SAIVT-VirtualGate. (Zugegriffen am 30.03.2020).

[51] *C-Extensions for Python*. URL: https://cython.org/. (Zugegriffen am 20.05.2020).

[52] *Limiting GPU memory growth*. URL: https://www.tensorflow.org/guide/gpu#limiting_gpu_memory_growth. (Zugegriffen am 01.04.2020).

[53] *Using allow_growth memory option in Tensorflow and Keras*. URL: https://kobkrit.com/using-allow-growth-memory-option-in-tensorflow-and-keras-dc8c8081bc96. (Zugegriffen am 01.04.2020).

[54] Oliver Augenstein. *Vorlesungs- und Übungsunterlagen Mathematical Foundations For Machine Learning*. Hochschule für Technik, Rapperswil. Durchführung 2019.

[55] John Hunt. *A Beginners Guide to Python 3 Programming*. 1st. Springer Publishing Company, Incorporated, 2019. ISBN: 3030202895.

[56] *Burg Wächter DOME 303 39050*. URL: https://www.conrad.ch/de/p/burg-waechter-dome-303-39050-lan-wlan-ip-ueberwachungskamera-2048-x-1536-pixel-1524044.html. (Zugegriffen am 01.05.2020).

[57] *ZeroMQ Socket Options*. URL: http://api.zeromq.org/2-1:zmq-setsockopt. (Zugegriffen am 04.06.2020).

[58] Zhicheng Ding und Edward Wong. „Confidence Trigger Detection: An Approach to Build Real-time Tracking-by-detection System“. In: (2019). arXiv: 1902.00615. URL: https://arxiv.org/abs/1902.00615.

[59] *TensorFlow Object Detection API*. URL: https://github.com/opencv/opencv/wiki/TensorFlow-Object-Detection-API. (Zugegriffen am 04.06.2020).