Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Christopher Czarnetzki

Konzeption und Entwicklung eines Deep Learning basierten Embedded Vision Systems zur Analyse von Laufwegen

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Christopher Czarnetzki

Konzeption und Entwicklung eines Deep Learning basierten Embedded Vision Systems zur Analyse von Laufwegen

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Masterstudiengang Automatisierung
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Jörg Dahlkemper
Zweitgutachter : Prof. Dr. Thomas Lehmann

Abgegeben am 20. Mai 2021

Christopher Czarnetzki

Thema der Masterarbeit

Konzeption und Entwicklung eines Deep Learning basierten Embedded Vision Systems zur Analyse von Laufwegen

Stichworte

Development-Board, Embedded Vision System, Edge-Computing, Deep Learning, YOLOv3, YOLO-Tiny, Objekterkennung, maschinelles Sehen, intelligente Videoanalyse

Kurzzusammenfassung

In dieser Arbeit werden die Entwicklung, Konzeption und die Realisierung eines Embedded Vision Systems beschrieben. Die Aufgabe dieses Systems ist die Laufweganalyse von Personen. Ein für den Einsatz von Machine Learning Anwendungen entworfenes Development-Board bildet die Basis des Systems. Die Laufweganalyse erfolgt über eine Deep Learning basierte Personenerkennung, für die das neuronale Netz YOLOv3-Tiny eingesetzt wird. Auf Grundlage der durch die Personenerkennung akquirierten Daten wird ermittelt, wie häufig bestimmte Teilbereiche eines überwachten Bereichs von Personen als Laufwege genutzt werden. Zur Veranschaulichung der Ergebnisse wird eine entsprechende Visualisierung erzeugt.

Christopher Czarnetzki

Title of the paper

Conception and development of a Deep Learning based embedded vision system for the analysis of walking paths

Keywords

Development-board, embedded vision system, edge-computing, Deep Learning, YOLOv3, YOLO-Tiny, object detection, computer vision, intelligent video analytics

Abstract

This thesis describes the development, conception and realization of an embedded vision system. The system task is to analyse the walking path of persons. A development board designed for the use of machine learning applications forms the basis of the system. The walking path analysis is achieved by a Deep Learning based person detection performed by the neural network YOLOv3-Tiny. Based on the data acquired by the person detection it is determined how often certain sub-areas of a monitored area are used by people as walking paths. A corresponding visualization is generated to illustrate the results.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

Afo.	Anforderung
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
AV	Average Precision
BB	Bounding Box
CNN	Convolutional Neural Network
COCO	Common Objects in Context
ConvNet	Convolutional Network
COTS	Commercial-off-the-shelf
CSI	Camera Serial Interface
CSV	Comma-separated values
CUDA	Compute Unified Device Architecture
DL	Deep Learning
DNN	Deep Neural Network
FCN	Fully Convolutional Network
FCNN	Fully-Connected Neural Network
FLOPS	Floating Point Operations Per Second
FN	False Neative
FP	False Positive
fps	frames per second
GPU	Graphics Processing Unit
HAW	Hochschule für Angewandte Wissenschaften

ILSVRC ImageNet Object Recognition Challenge

IoU Intersection over Union

KI Künstliche Intelligenz

KN Künstliche Neuronen

KNN Künstliches Neuronales Netz

mAP mean Average Precision

ML Machine Learning

MOT Multiple Object Tracking

NMS Non-Maximum Suppresion

o. J. ohne Jahr

ReLu Rectified Linear Unit

RNN Recurrent Neural Network

ROI Region of Interest

RPN Region Proposal Network

SDK Software Development Kit

SGD Stochastic Gradient Descent

SSD Single Shot Multibox Detector

SVM Support Vector Machine

TOPS Tera Operations Per Second

TP True Positive

TPU Tensor Processing Unit

VM Virtuelle Maschine

VOC Visual Object Classes

VPU Vision Processing Unit

VTB Visual Tracker Benchmark

YOLO You Only Look Once

1. Einleitung

Die Idee der künstlichen Intelligenz (KI) wurde in den 1950er Jahren mit der Frage geboren, ob Computer die Fähigkeit des “Denkens“ erlangen könnten. Dieses Thema beschäftigt seit dieser Zeit die Forschung und Wissenschaft. In den vergangenen Jahren haben die künstliche Intelligenz sowie deren Teilgebiet das Machine Learning (ML), als auch das Deep Learning (DL) als Zweig des maschinellen Lernens, eine bedeutsame Entwicklung durchlaufen.
Die KI hat beginnend in den Jahren 1960 und 1980 bereits zwei Phasen großer Erwartungen und großen Optimismus' durchlaufen. Heute befindet sich die KI mitten in einer dritten solchen Phase. In den 1960er und frühen 1970er Jahren haben zu hohe Erwartungen und falsche Prognosen in eine *symbolische KI* jedoch zu einer Abkehr von der künstlichen Intelligenz geführt. Zwei Beispiele falscher Prognosen und nicht erfüllbarer Erwartungen stammen von Marvin Minsky. Dieser sagte aus, das Problem der Schaffung künstlicher Intelligenz werde im Wesentlichen innerhalb einer Generation gelöst sein. Darüber hinaus sprach Minsky 1970 davon, es werde drei bis acht Jahre dauern, bis es Maschinen mit der allgemeinen Intelligenz auf dem Niveau eines durchschnittlich intelligenten Menschen geben werde. Die dementsprechend hohen Erwartungen konnten nicht erfüllt werden und führten zum Ende von Investitionen und Forschungstätigkeiten und dem Beginn eines *KI-Winters*[1]. Ähnlich verhielt es sich in den 1980er Jahren. Die symbolische KI erlebte in Form der sogenannten *Expertensysteme* einen neuen Aufschwung. Einige erste Erfolgsgeschichten lösten weltweite Investitionen und Entwicklungsbemühungen im Bereich dieser neuen Form der KI aus. Die neue Technologie brachte jedoch Nachteile wie teure Wartungen, eine schlechte Skalierbarkeit und begrenzte Anwendungsmöglichkeiten mit sich. Und so ging auch in dieser zweiten Phase Anfang der 1990er das Interesse an der KI verloren, mit dem Resultat eines weiteren KI-Winters (vgl. [Chollet, 2018a], S. 12).

Heute haben Entwicklungsarbeit und Investitionen in die Technologie von Hochleistungsrechner, Computerclustern und das Cloud-Computing sowie die große Menge verfügbarer Daten bedeutende Fortschritte im Bereich der künstlichen Intelligenz mitsamt des Durchbruchs des Deep Learnings bewirkt. Steigende Rechenleistung und deren bessere Verfügbarkeit bieten die Möglichkeit, existierende KI-Anwendungen zu optimieren und eröffnen darüber hinaus

[1] Eine Anspielung auf den nuklearen Winter, da diese Ereignisse kurz nach dem Höhepunkt des Kalten Krieges stattfanden (vgl. [Chollet, 2018b], S. 33)

neue Einsatzbereiche und Anwendungszwecke.
Als eine Folge dieser Entwicklung beliefern Technologiekonzerne den kommerziellen Markt in steigendem Umfang mit kompakten Embedded Systemen, Development-Modulen und -Boards sowie vollständigen Development-Kits, die den praktischen Einsatz von KI-Anwendungen unter Verwendung von ML- sowie DL-Methoden ermöglichen. Im Nachfolgenden werden stellvertretend für alle Entwicklungen dieses Marktsegments einige Lösungen führender Technologieunternehmen vorgestellt.
Der NVIDIA Konzern bietet mit der *Jetson Familie* Produkte in Form einzelner Module [NVIDIA, 2020b] oder vollständiger Development-Kits [NVIDIA, 2020a] an. Die NVIDIA Jetson Produkte arbeiten neben der CPU mit zusätzlichen Graphics Processing Units (GPUs), um die erforderliche Rechenleistung für mobile oder lokale (dezentrale) KI-Anwendungen bereitzustellen. Ähnliche Lösungen bietet Google über seine *Coral* Plattform an [Google, o. J.d]. Der Angebotsumfang beinhaltet unter anderem separate Module, Development-Boards und einen USB-Beschleuniger [Google, o. J.b]. Dieser USB-Beschleuniger kann als zusätzlicher Prozessor an ein bestehendes System angeschlossen werden und dieses mit additiver Rechenleistung versorgen. Im Gegensatz zu den Jetson Produkten beziehen die Lösungen der Coral Plattform ihre Leistung nicht aus GPUs, sondern den speziell für ML-Anwendungen entwickelten Tensor Processing Units (TPUs). Eine Alternative zu Googles USB-Beschleuniger stellt der Intel Movidius Neural Compute Stick dar [Intel, o. J.]. Dieser ist ebenfalls dafür vorgesehen einem bestehenden System zusätzliche Rechenleistung zu liefern, nutzt mit der Vision Processing Unit (VPU) jedoch eine weitere Alternative zu GPU und TPU.
In Verbindung mit einer geeigneten Kameratechnologie eignen sich diese Systeme beispielsweise für verschiedenste Applikationen im Bereich des *maschinellen Sehens* (engl. *computer vision*). Entsprechende Komplettlösungen werden bereits von verschiedenen Technikunternehmen angeboten (s. bspw. [TheImagingSource, o. J.] und [BaslerAG, o. J.]).
Die Entwicklung und Nutzung dieser Embedded Systeme und die daraus resultierende Anwendbarkeit des *Edge-Computings* ist eng mit der Entwicklung von Smart Cities, Smart Homes, intelligenten Überwachungssystemen und dem autonomen Fahren verknüpft (vgl. [Shi et al., 2016]). Die Objekterkennung im Zusammenhang einer intelligenten Videoanalyse in öffentlichen Einrichtungen und Bereichen ist ein typisches Anwendungsfeld. Häufig zielt diese Videoanalyse speziell auf Personen ab, um diese quantitativ zu erfassen oder Erkenntnisse aus deren Bewegungsverhalten abzuleiten.

Die konkreten Verwendungszwecke sind vielfältig und umfassen unter anderem

- das Identifizieren möglicher Gefahrenstellen in Bereichen die gemeinsam von Fahrzeugen und Personen genutzt werden,
- die Beobachtung der Laufwege von Personen unter dem Aspekt des *Social Distancing* (vgl. [Neuralet, o. J.])

- Analyse von Bewegungsmustern von Kunden im Einzelhandel,
- die einfache Personenzählung oder
- die sicherheitsbezogene Videoüberwachung (vgl. [Bosch, o. J.])

Das Spektrum möglicher Anwendungen und Anwendungsbereichen der KI-Technologie mitsamt den Methoden des Machine- und Deep-Learnings ist noch deutlich umfangreicher, weshalb davon ausgegangen werden kann, dass das derzeitige Interesse an der KI in der nächsten Zukunft nicht abnehmen wird.

1.1. Zielsetzung

In dieser Arbeit soll durch methodisches Vorgehen ein Embedded Vision System konzipiert und entwickelt werden. Dieses System soll es anhand von Videoaufzeichnungen ermöglichen, die Laufwege von Personen in öffentlich zugänglichen und verkehrsberuhigten Bereichen, wie beispielsweise dem Hochschulcampus der HAW Hamburg, zu analysieren.
Das Ziel dieser Analyse besteht darin, die von Personen stark und schwach frequentierten Teilbereiche innerhalb eines Gesamtaufnahmebereichs zu identifizieren und voneinander zu unterscheiden. Um in diesem Zusammenhang Personen zu erkennen und diese als solche zu klassifizieren, soll ein geeignetes künstliches neuronales Netz bzw. ein entsprechendes Deep Learning Modell (kurz Modell) eingesetzt werden.
Die Ergebnisse dieser Analyse sollen im Rahmen eines anderen Projektes die Einsatzplanung mobiler Transportplattformen unterstützen und dazu beitragen, Personen in einem möglichst geringen Maße zu behindern. Ein Datentransfer zu den mobilen Transportsystemen ist im Rahmen dieser Arbeit nicht vorgesehen. Im Kontext dieser Arbeit sollen die akquirierten Daten dazu verwendet werden, die Nutzungshäufigkeit der einzelnen Teilbereiche in Abhängigkeit zeitlicher Intervalle zu visualisieren.
Für die Realisierung des Embedded Vision Systems sowie die Erfüllung der notwendigen Funktionalitäten sind die Anforderungen an ein Development-Board, einen Datensatz und ein neuronales Netz bzw. Modell zu analysieren.
Im konzeptionellen Teil sind eine systematische Auswahl des Development-Boards, Datensatzes und Modells durchzuführen sowie eine zum Development-Board kompatible Kamera auszuwählen. Überdies ist ein Implementierungskonzept zu erarbeiten, welches die Analyse der Laufwege und die entsprechende Visualisierung ermöglicht. Das dementsprechend entwickelte Embedded Vision System soll als Stand-alone-Lösung die Bildinformationen aus Videoaufzeichnungen unter Nutzung des umgesetzten Implementierungskonzepts analysieren und eine entsprechende Visualisierung erzeugen können. In diesem Zusammenhang soll zum Nachweis der Funktionsfähigkeit, unter Annahme eines beispielhaften Szenarios,

ein Praxistest erfolgen und das Verfahren einer Laufweganalyse beschrieben werden.

1.2. Struktur der Arbeit

Diese Masterarbeit umfasst acht Kapitel. Das erste Kapitel enthält neben der allgemeinen Einleitung und der Zielsetzung diese Beschreibung der Struktur der Arbeit.
In Kapitel 2 werden die wesentlichen Grundlagen des maschinellen Lernens und der Objekterkennung beschrieben. Ferner werden ausgewählte Metriken und Bewertungsmöglichkeiten für die Objekterkennung erläutert. Den Abschluss dieses Kapitels bildet eine Erklärung zu Embedded Vision Systemen und dem Edge-Computing.
Im dritten Kapitel werden konkrete Methoden und Lösungen der Objekterkennung aufgeführt und deren charakteristische Eigenschaften benannt. Zudem werden, wo zutreffend, die Weiterentwicklungen einzelner Methoden beschrieben. Zum Schluss dieses Kapitel werden verwandte Arbeiten vorgestellt, die ähnliche Thematiken in Bezug auf das Thema dieser Arbeit behandeln.
In Kapitel 4 wird die Anforderungsanalyse als Basis der Konzeption sowie deren Realisierung dargestellt. Im Rahmen dieser Analyse beziehen sich die Anforderungen auf die wesentlichen Systemanteile hinsichtlich Hardware und Software als auch auf das Embedded Vision System im Gesamten. Sämtliche Anforderungen sind im letzten Abschnitt dieses Kapitels in einer Übersicht zusammengefasst.
Basierend auf der Anforderungsanalyse wird im Rahmen der Konzeption in Kapitel 5 der Prozess der Entscheidungsfindung samt Argumentation für die Auswahl der berücksichtigten Systemanteile erläutert. Des Weiteren wird das als Argumentationsgrundlage verwendete Vergleichs- und Bewertungssystem dargestellt. Hinsichtlich der vorgesehenen Funktion der Laufweganalyse umfasst Kapitel 5 überdies die Veranschaulichung des Implementierungskonzepts.
In Kapitel 6 wird im Nachfolgenden die Umsetzung der Konzeption respektive das realisierte Embedded Vision System samt Funktionalität vorgestellt. Hierbei werden im Detail die eingesetzte Hardware sowie die Softwareanteile beschrieben. Der Softwareanteil umfasst die Erläuterung des eingesetzten Modells und liefert überdies weitere Angaben zu Systembestandteilen, die die Software betreffen. Ferner werden die Zusammenstellung und Vorverarbeitung des Datensatzes sowie wesentliche Aspekte des Trainings wiedergegeben. Abschließend werden die Durchführung und das Verfahren der Laufweganalyse geschildert. Die Laufweganalyse wird dazu in einzelne Phasen unterteilt, welche jeweils detailliert erläutert werden.
Die Evaluation der erzielten Ergebnisse erfolgt in Kapitel 7. Darin enthalten ist die Interpretation der im Rahmen der durchgeführten Laufweganalyse erzeugten Visualisierung. Weiterhin wird eine Bewertung der Leistungsfähigkeit des Embedded Vision Systems vorgenommen.

Dazu werden gängige Metriken herangezogen.
Den Abschluss dieser Masterarbeit bildet das Kapitel 8, in dem zum einen die wesentlichen Inhalte und Erkenntnisse der Arbeit zusammengefasst, rekapituliert und im Allgemeinen beurteilt werden. Zum anderen werden mögliche Verbesserungen und Weiterentwicklungen thematisiert.

2. Grundlagen

Als Basis für die noch folgenden Hauptkapitel werden im Nachfolgenden fachliche Grundlagen erläutert, welche für die vorliegende Aufgabe relevant sind. Ferner beschreibt dieses Kapitel im Generellen die Art der eingesetzten Hardware, sprich das Embedded Vision System sowie das damit einhergehende Edge-Computing.

Das Ziel dieses Kapitels ist es, eine Verständnisgrundlage für die behandelten fachlichen Themengebiete sowie der Konzeption und des Systemdesigns aufzubauen.

2.1. Maschinelles Lernen

Das maschinelle Lernen, sprich das Machine Learning (ML), ist die Wissenschaft, einen Computer so zu programmieren, dass dieser aus den Daten lernen kann (vgl. [Gerón, 2019], S. 2). Um diese Aussage näher zu beschreiben, thematisiert dieses Kapitel die wesentlichen Charakteristiken und Konzepte des Machine Learnings sowie die elementaren Bestandteile auf denen das Machine Learning aufbaut.

2.1.1. Die Verfahren des maschinellen Lernens

Es existieren unterschiedliche Arten des Machine Learnings. Eine Möglichkeit, diese zu unterscheiden ist die Art des Lernverfahrens. Diese lassen sich in das überwachte (engl. *supervised*), unüberwachte (engl. *unsupervised*), teilüberwachte (engl. *semi-supervised*) sowie das verstärkende (engl. *reinforcement*) Lernen unterteilen.
Bei dem überwachten Lernen handelt es sich um die am häufigsten eingesetzte Variante des maschinellen Lernens (vgl. [Chollet, 2018b], S. 129). Hierbei werden Trainingsdaten verwendet, die aus Paaren von Ein- und Ausgabedaten bestehen. Für alle eingegebenen Daten ist der Zielwert bereits bekannt. Eine typische Aufgabe des überwachten Lernens ist die Klassifizierung. Klassische Eingangsdaten können hierbei beispielsweise in Form von Bildern oder bekannten Attributen einer Instanz vorliegen. Handelt es sich bei der Instanz beispielsweise um eine E-Mail, kann das Attribut „Spam“ oder „kein Spam“ lauten. Die Ausgabedaten sind die korrespondierenden Lösungen in Form von Labels, respektive der Klassenzuordnung. Im

Falle der Regression, welche als weitere typische Lernaufgabe gilt, bestehen die Eingangsdaten aus unterschiedlichen Merkmalen (engl. *features*) einer Instanz und die Ausgangsdaten aus stetigen nummerischen Zielwerten. Bei der Instanz kann es sich beispielsweise um ein Fahrzeug handeln, dessen Merkmale das Alter und die Fahrleistung sind. Diese Eingangsdaten sind mit den nummerischen Zielwerten, wie beispielsweise einem Verkaufspreis, verknüpft.
Neben der Klassifizierung und Regression umfasst das überwachte Lernen auch die für diese Arbeit relevante Objekterkennung. Die Eingangsdaten bestehen aus Bildern, auf denen Objekte dargestellt sind. In Bezug auf die vorliegende Arbeit wird die Objekterkennung als Kombination aus Klassifizierung und Regression betrachtet, bei welcher die Koordinaten eines Markierungsrahmens durch Vektorregression vorhergesagt werden (vgl. [Chollet, 2018b], S. 130).
Das unüberwachte Lernen unterscheidet sich von dem überwachten Lernen im Wesentlichen durch das Fehlen bereits klassifizierter Beispieldaten. Eine anschauliche und neben der Dimensionsreduktion gängige Methode ist das Bilden von Clustern (engl. *clustering*), wobei in den Eingabedaten nach gleichen Merkmalen und Ähnlichkeiten gesucht wird, mittels derer eine Verteilung der eingegangenen Daten in Untergruppen sprich Cluster ermöglicht. Zu den weiteren gängigen Methoden des unüberwachten Lernens zählen darüber hinaus die Anomalieerkennung und Novelty Detection, die Visualisierung und Dimensionsreduktion sowie das Lernen mit Assoziationsregeln (siehe auch [Gerón, 2020], S. 11f.).
Das teilüberwachte Lernen ist eine Mischform beider zuvor beschriebenen Verfahren. In den meisten Fällen handelt es sich bei den Algorithmen für das teilüberwachte Lernen um kombinierte Algorithmen des überwachten und unüberwachten Lernens. Die Datensätzen bestehen hier aus Eingabedaten für die nur teilweise Ausgabedaten gegeben sind. Dadurch kann der Aufwand zur Erstellung von annotierten Datensätzen reduziert, bzw. auch die Menge der Trainingsdaten erhöht werden.
Im Vergleich zu den drei vorgestellten Verfahren folgt das verstärkende Lernen einem anderen Ansatz. Es setzt auf das Erlernen einer Strategie, die sogenannte *policy*, die zu möglichst vielen oder hohen Belohnungen beim Ausführen einer Aufgabe führt. Die Strategie kann ein beliebiger Algorithmus sein. Das System, welches diese Strategie anwendet, wird als *Agent* bezeichnet. Dieser Agent führt Aktionen aus, interagiert also mit seiner Umgebung und erhält dafür mittels Belohnungsfunktion eine positive oder negative Rückmeldung. Abhängig davon wie eine Belohnung ausfällt, wird die Strategie korrigiert, bis die Belohnungsfunktion maximiert ist.

2.1.2. Das künstliche Neuron und künstliche neuronale Netz

Das künstliche Neuron (KN) oder auch *Perzeptron* ist der Grundbaustein des maschinellen Lernens. Im Grundsatz ist dieses künstliche Neuron eine mathematische Operation, die über

eine gewichtete Summe von Eingängen im Zusammenwirken mit einer Aktivierungsfunktion eine Ausgabe erzeugt. Zusätzlich zu der gewichteten Summe wird ein Bias-Terms b berücksichtigt (siehe auch [Koul et al., 2019] Abschnitt *Exciting Beginnings*) [1]:

$$z = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b = \sum_n w_n x_n + b = \mathbf{w} \cdot \mathbf{x} + b \tag{2.1}$$

Ist z positiv, wird der Ausgang y des künstlichen Neurons aktiviert und der Wert *1* ausgegeben, andernfalls *0* (vgl. [Nielsen, 2015], Kapitel 1):

$$y = \begin{cases} 1 & \text{falls } z > 0 \\ 0 & \text{falls } z \leq 0 \end{cases} \tag{2.2}$$

In Abbildung 2.1 ist exemplarisch der schematische Aufbau eines künstlichen Neurons dargestellt.

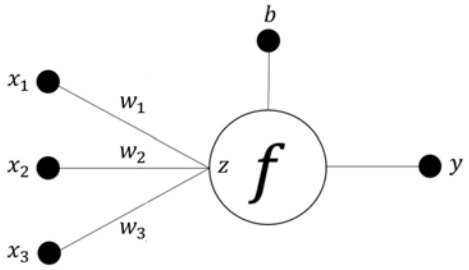


Abbildung 2.1.: Schematische Darstellung eines künstlichen Neurons nach [Koul et al., 2019], Kapitel 1

Durch das Zusammenfassen künstlicher Neuronen entsteht ein künstliches neuronales Netz (KNN). Dieses besteht aus einer beliebigen Anzahl von *Layern*, die auch als *Schichten* bezeichnet werden. Der Input Layer (auch Eingabeschicht) dient als Dateneingang in das KNN. Auf den Input Layer folgen ein oder mehrere *Hidden* Layer (deutsch verborgene Schichten), die mit den darin enthaltenen Neuronen den Dateneingang verarbeiten. Die Ergebnisse der Berechnungen werden jeweils an die Neuronen der nachfolgenden Layer übergeben. Dieses Vorgehen entspricht der Datenverarbeitung innerhalb eines *Feedforward Netzes*. In diesen Netzen sind die Eingänge eines Layers nur mit den Ausgängen des vorherigen Layers verbunden. Sind alle Neuronen eines Layers mit sämtlichen Neuronen des Folgelayers verbunden, spricht man von *vollständig verbundenen Netzen* (engl. *Fully-Connected Neural Networks*, kurz FCNNs). Deren einzelne Layer werden dementsprechend als Fully-Connected oder auch als *Dense* Layer bezeichnet (siehe auch [Chollet, 2018b], S. 403ff.). In Abbildung 2.2 ist ein solches Netz dargestellt.

[1] Konvention: In der vorliegenden Arbeit werden Vektoren als fettgedruckte Kleinbuchstaben dargestellt.

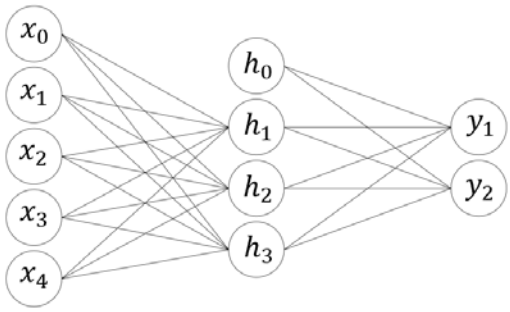


Abbildung 2.2.: Künstliches Neuronales Netz - Fully-Connected

Ein weiterer Typ eines KNNs ist das rekurrente neuronale Netz (engl. *Recurrent Neural Network*, kurz RNN). Dieses weist Rückkopplungen von Layern auf vorangegangene Layer auf und berücksichtigt so vorherige Zustände (vgl. auch [Chollet, 2018a], S. 196ff. oder [Goodfellow et al., 2016], Kap. 10).
Unabhängig von dem Typ werden KNNs mit zwei oder mehr Hidden Layern als *tiefe* neuronale Netze (engl. *Deep Neural Networks*, kurz DNNs) bezeichnet. Bei einem Netz mit weniger als zwei Hidden Layern handelt es sich um ein flaches neuronales Netz (engl. *Shallow Neural Network*). Der letzte Layer eines KNNs ist der Output Layer (auch Ausgabeschicht). Dieser generiert die abschließende Ausgabe des Netzes.

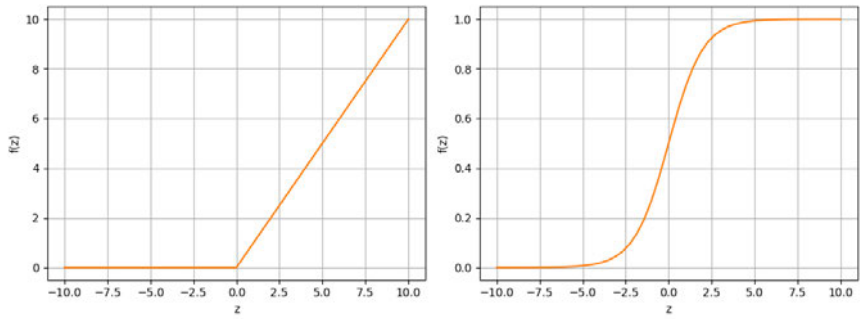


Abbildung 2.3.: ReLU- (links) und Sigmoid-Funktion (rechts)

Wie die Neuronen eines KNNs auf ihre Eingangsgrößen reagieren und aus diesen Ausgangsgrößen generieren, wird durch die Aktivierungsfunktion f bestimmt. Gängige Aktivierungsfunktionen sind die *ReLu*-Funktion (*Rectified Linear Unit*) (vgl. [Gerón, 2020], S. 293) oder auch die *Sigmoid*-Funktion. Diese sind in Abbildung 2.3 dargestellt. Die ReLu-Funktion setzt negative Werte auf null und lässt positive Werte unverändert:

$$f(z) = max(0, z) \tag{2.3}$$

Die Sigmoid-Funktion wird in der Regel in Verbindung mit dem Output Layer eingesetzt. Sie bildet Eingabewerte auf den Bereich [0,1] ab und kann daher genutzt werden, um beispielsweise Wahrscheinlichkeiten für die Zugehörigkeit zu einer bestimmten Klasse auszugeben. Die Sigmoid-Funktion ist definiert durch [Raschka, 2018], Seite 450:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

Das Ziel der bis zu dieser Stelle beschriebenen KNNs ist es, unter Einsatz erlernter Gewichtungen und Bias-Terme aufgabenspezifische Ausgaben zu erzeugen, die einem erwarteten Ergebnis entsprechen oder diesem möglichst nahe kommen. Die Schlüsselkonzepte zum Erlernen von Gewichtungen und Bias-Termen sind die *Kostenfunktion* sowie ein Optimierungsverfahren. Beide Konzepte werden in Kapitel 2.1.3 näher erläutert.

2.1.3. Kostenfunktion und Optimierungsverfahren

Das Trainieren eines KNNs und das damit einhergehende Erlernen von Gewichtungen und Bias-Termen erfolgt unter Anwendung der angesprochenen Kostenfunktion und dem Optimierungsverfahren, welches auf dem *Gradientenabstiegsverfahren* und dem *Backpropagation* Algorithmus beruht. Die Kostenfunktion sowie das Optimierungsverfahren werden innerhalb eines iterativen Trainingsprozesses eingesetzt, der mit einem trainierten Netz respektive Modell endet.
Das Zusammenwirken von Kostenfunktion, Gradientenverfahren und Backpropagation während eines Trainingsprozesses lässt sich folgendermaßen zusammenfassen. Nach einem Trainingsdurchlauf ermittelt die Kostenfunktion den Kostenwert, der als das Feedback-Signal für die nachfolgenden Schritte zur Optimierung der Gewichte und Bias-Terme fungiert. Die Backpropagation berechnet den Fehlergradienten sämtlicher Parameter eines Netzes. Nachdem diese bestimmt sind, werden die Parameter entgegen der Richtung der berechneten Gradienten über das ausgewählte Gradientenverfahren aktualisiert, um so die Kostenfunktion zu reduzieren. Dieser Ablauf wird wiederholt, bis eine vorgegebene Anzahl an Durchläufen erreicht ist.

Kostenfunktion

Die Kostenfunktion kann als das Qualitätsmaß während des Trainings betrachtet werden. Sie vergleicht die berechnete Ausgabe eines Modells mit dem tatsächlichen Ergebnis, führt also einen Soll-Ist-Vergleich durch, und gibt so an, wie gut ein Modell eine Aufgabe lösen kann. Das Ergebnis des Vergleichs ist ein *Kostenwert*, den es während eines Trainings durch das

Ermitteln geeigneter Gewichtungen und Bias-Terme zu minimieren gilt. In diesem Zusammenhang wird von der „Minimierung der Kostenfunktion“ gesprochen.

Am Beispiel der mittleren quadratischen Abweichung, kurz MSE, (engl. *mean squared error*) berechnet sich dieser Wert als Funktion aller Gewichte und Bias-Terme (vgl. [Nielsen, 2015], Kap. 1, Gleichung (6)):

$$C_{\mathbf{w},\mathbf{b}} = \frac{1}{2n} \sum_{x} (y_{(\mathbf{x})} - \mathbf{a})^2 \tag{2.5}$$

Der Faktor n gibt die Anzahl der eingegebenen Trainingsdaten an. Bei $y_{(\mathbf{x})}$ handelt es sich um die erwünschte vektorielle Ausgabe und bei $\mathbf{a} = f(z)$ um die jeweils korrespondierende berechnete Ausgabe des vorhergehenden Layers.
In der Regel weisen Kostenfunktionen einen hohen Anfangswert auf, da initiale Gewichtungen zufällig gewählt werden. Dies wird als „zufällige Initialisierung“ bezeichnet (vgl. [Gerón, 2020], S. 121). Die Aktualisierung der Modell-Parameter, also der Gewichtungen sowie Bias-Terme, bewirkt im Rahmen des Trainingsprozesses Änderungen der Ausgangsgrößen. Diese sollen zu der angesprochenen Minimierung der Kostenfunktion führen. Der niedrigste erreichbare Wert entspricht dem sogenannten *globalen Minimum*.

Gradientenabstiegsverfahren

Um das im vorangegangenen Abschnitt angesprochene globale Minimum zu erreichen, wird das Gradientenabstiegsverfahren eingesetzt. Dabei ist es erforderlich, den Gradienten der Kostenfunktion in Abhängigkeit der Gewichtungen zu kennen. Zur Optimierung der Kostenfunktion werden die Parameter unter Anwendung des Gradientenverfahrens in Richtung des abfallenden Gradienten, also des negativen Gradienten, geändert. Die mathematische Formulierung dieser Änderung kann beschrieben werden durch (vgl. [Raschka, 2018], S. 58):

$$\mathbf{w}' = \mathbf{w} + \triangle\mathbf{w} = \mathbf{w} - \eta \nabla C_{(\mathbf{w})} \tag{2.6}$$

Hierbei steht $\mathbf{w}'$ für neue Gewichtungen. Diese Gleichung gilt analog für die Bias-Terme. Alle Variablen w müssen dazu durch die Variablen b ersetzt werden.

Dieser Ablauf wird wiederholt, bis der Algorithmus einen minimalen oder zufriedenstellenden Wert annimmt. Ein wichtiger Hyperparameter ist in diesem Kontext die Lernrate η. Diese wird mit dem negativen Gradienten $\nabla C_{(\mathbf{w})}$ der Kostenfunktion multipliziert, woraus sich eine Änderung der Gewichtungen mit einer bestimmten Schrittweite ergibt. Wie in Abbildung 2.4 dargestellt, beeinflusst die Schrittweite maßgeblich die Entwicklung der Kostenfunktion. In der Abbildung steht Θ für den Parametervektor und $\hat{\Theta}$ für den Parametervektor, für den

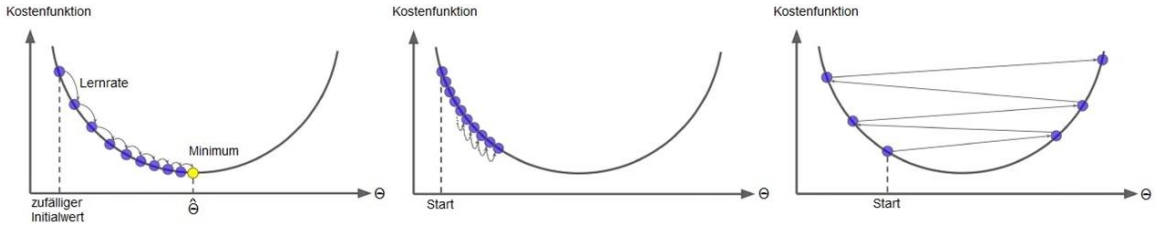


Abbildung 2.4.: Gradientenabstieg bei ideal (links), zu klein (mitte) oder zu groß (rechts) gewählter Schrittweite (vgl. [Gerón, 2019], S. 118f.)

die Kostenfunktion minimal wird. Während des Trainingsprozesses konvergiert die Kostenfunktion bei gut gewählter Schrittweite und der Optimierung der Parameter mit relativ wenig Iterationen zielgerichtet hin zum globalen Minimum.
Wird die Lernrate zu niedrig gewählt, ändern sich die Parameter nur in kleinen Schritten. Auch hier konvergiert die Kostenfunktion zum globalen Minimum, allerdings sind deutlich mehr Iterationen notwendig, was einen deutlich längeren Trainingsprozess zur Folge hat. Bei zu groß gewählter Lernrate besteht die Gefahr des Überspringens von Minima und einem Divergieren der Parameterwerte.

Für die Anwendung des Gradientenabstiegsverfahrens bestehen unterschiedliche Möglichkeiten. Das Unterscheidungsmerkmal ist dabei die sogenannte *Batch-Größe*, die wie die Lernrate ein weiterer Hyperparameter eines Trainings ist.

Die erste Möglichkeit ist das *Batch-Gradientenverfahren*. Dabei wird nach jeder Änderung $\nabla C_{(w)}$ berechnet, um die Auswirkung der Änderung auf die Kostenfunktion festzustellen. Innerhalb eines Trainingsdurchlaufs wird dies für jeden einzelnen Datenpunkt ausgeführt. Das Berechnen der Gradienten und die Aktualisierung der Gewichtungen nach jedem Schritt über den gesamten Datensatz machen das Batch-Gradientenverfahren sehr präzise. Durch den hohen Rechenaufwand und eine lange Trainingsdauer eignet sich dieses Verfahren jedoch wenig für sehr große Datensätze.
Im Gegensatz dazu wird beim *stochastischen* Gradientenverfahren, kurz SGD (engl. *stochastic gradient descent*), lediglich der Gradient eines zufällig gewählten Datenpunktes errechnet. Aufgrund der niedrigen Anzahl notwendiger Berechnungen handelt es sich um ein sehr schnelles Verfahren, welches sich gut für große Datensätze eignet. Die Gefahr in lokalen Minima zu verbleiben, ist hier reduziert und die Wahrscheinlichkeit, das globale Minimum zu erreichen, folglich erhöht. Der für die stochastische Arbeitsweise charakteristische hohe Zufallsanteil macht dieses Verfahren jedoch sehr unregelmäßig. Dies äußert sich in dem sprunghaften Verhalten der Kostenfunktion. Zwar sinkt die Kostenfunktion im Mittel, doch zeigt sich das sprunghafte Verhalten bis um das Minimum herum.
Das am häufigsten eingesetzte sogenannte *Mini-Batch-Gradientenverfahren* stellt einen Kompromiss zwischen den zuvor beschriebenen Verfahren dar. Bei diesem Verfahren werden die Gradienten für sogenannte *Mini-Batches* berechnet. Hierbei handelt es sich um zu-

fällig gewählte Teilmengen eines Datensatzes. Abhängig von der Größe eines Mini-Batches und der entsprechenden Anzahl zu berechnender Gradienten ist nur die Aktualisierung einer reduzierte Menge von Gewichten und Bias-Termen notwendig.

Bezüglich dieses Verfahrens werden für die Berechnung von $\nabla C_{(w)}$ lediglich die jeweiligen Eingabedaten $X_1, X_2, \ldots, X_m$ des Mini-Batches herangezogen:

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j} \tag{2.7}$$

Die Aktualisierung der korrespondierenden Gewichte w_k und Bias-Terme b_l ergibt sich aus den Gleichungen 2.8 und 2.9. In beiden Gleichungen werden jeweils die Summen über den gesamten Eingabedatensatz des Mini-Batches gebildet.

$$w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k} \tag{2.8}$$

$$b_l' = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l} \tag{2.9}$$

Die hier aufgeführten Gleichungen 2.7, 2.8 und 2.9 entsprechen sinngemäß den Gleichungen (19), (20) und (21) aus [Nielsen, 2015], Kapitel 1.

Im Vergleich zum Batch-Gradientenverfahren ist dieses Verfahren schneller, da nicht bei jedem Schritt alle Gradienten berechnet werden müssen. Des Weiteren wird durch die Verwendung von Mini-Batches die Sprunghaftigkeit des SGD reduziert. Der Gradientenverlauf wird folglich geglättet und eine größere Annäherung an Minima erreicht.

Backpropagation

Der letzte und für einen Trainingsprozess sowie die Vorgänge in einem KNN bzw. Modell essentielle Bestandteil ist die Backpropagation. Hierbei wird beginnend beim letzten Layer ein KNN rückwärts bis zum ersten Layer durchlaufen. Der Fehlergradient der Kostenfunktion wird dabei über alle Gewichtungen rückwärts durch das KNN propagiert (vgl. [Gerón, 2020], S. 292). Es wird also schrittweise der jeweilige Anteil aller Parameter am Gesamtfehler erfasst. Für die Ausführung macht die Backpropagation von der Kettenregel aus der Differentialrechnung Gebrauch (vgl. [Chollet, 2018b], S. 81). Formal kann die Backpropagation mit vier grundlegenden Gleichungen beschrieben werden (vgl. [Nielsen, 2015], Kap. 2). Die erste Gleichung beschreibt den Fehler δ^L im Output Layer:

$$\delta^L = \bigtriangledown_a C \bigodot f^{'}(z^L) \tag{2.10}$$

Bei $\bigtriangledown_a C$ handelt es sich um den Vektor dessen Komponenten $\frac{\partial C}{\partial a_j^L}$ die partiellen Ableitungen der Kostenfunktion nach den Aktivierungsausgaben der einzelnen Neuronen der Layer sind. Der Term $f^{'}(z^L)$ gibt an, wie schnell sich die Aktivierungsfunktion f bei den einzelnen gewichteten Eingängen z ändert. Die Berechnung des Fehlers δ^l in Bezug auf den Fehler des nächsten Layers δ^{l+1} wird berechnet mittels:

$$\delta^L = ((W^{l+1})^T \delta^{l+1}) \bigodot f^{'}(z^l) \tag{2.11}$$

Dabei ist $(\mathbf{W}^{l+1})^T$ die transponierte Gewichtsmatrix des Layers (l+1). Die Änderungsrate einer Kostenfunktion in Bezug auf die Bias-Terme ergibt sich aus:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.12}$$

Die Änderungsrate der Kostenfunktion in Bezug auf alle Gewichtungen eines KNNs kann dann beschrieben werden durch:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{2.13}$$

Die Gleichungen 2.10 bis 2.13 finden sich sinngemäß in [Nielsen, 2015], Kapitel 2.

2.1.4. Deep Learning und das Convolutional Neural Network

Das Deep Learning (DL) nimmt unter den verschiedenen Teildisziplinen des maschinellen Lernens eine besondere Stellung ein. In diversen Aufgaben- und Anwendungsbereichen hat speziell das DL die Grundlage für bedeutende Durchbrüche und Fortschritte geschaffen. Dies trifft insbesondere für den Bereich der Sinneswahrnehmungen und die Technologie des maschinellen Sehens zu. Hier gilt das Deep Learning als der Stand der Technik und stellt bereits seit 2012 den Algorithmus der Wahl dar (vgl. [Chollet, 2018b], S. 40). Im gleichen Jahr liegt auch der Ursprung des großen industriellen Interesses am DL. Auslöser des Interesses war der Gewinn der *ImageNet Object Recognition Challenge* (ILSVRC) durch [Krizhevsky et al., 2017] (vgl. [Goodfellow et al., 2016], S. 371). Krizhevsky et al. setzten dabei ein sogenannte *Faltungsnetze* (engl. *Convolutional Neural Networks*, kurz CNNs) ein. Die CNNs

nehmen eine wesentliche Rolle ein, da diese im Kontext des maschinellen Sehens in praktisch allen Anwendungsgebieten eingesetzt werden (vgl. [Chollet, 2018b], S. 161).

Im Grundsatz wurde die CNN Netzarchitektur erstmals von Yann LeCun in [LeCun et al., 1989] beschrieben. Eine typische Architektur des CNNs ist in Abbildung 2.5 dargestellt. Nach dem Input-Layer folgt der Schlüsselbestandteil eines CNNs, bei dem es sich um einen Stapel von Faltungsschichten (engl. *Convolutional-Layer*, kurz *Conv-Layer*) handelt, die wechselweise mit *Pooling-Layern* auftreten. Ein gewöhnliches feedforward FCNN bildet in der Regel gemeinsam mit dem Output-Layer das Ende eines CNNs (vgl. [Gerón, 2020], S. 464).

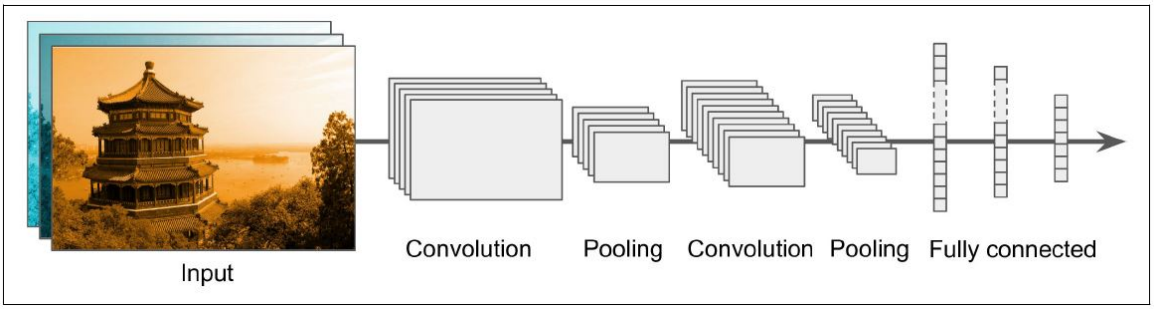


Abbildung 2.5.: Typische Architektur eines CNNs nach [Gerón, 2019], S. 477

Die Convolutional-Layer führen eine Operation aus, die als Faltung bezeichnet wird. Bei dieser wird mittels eines *Filters* (auch Convolutional-Kernel) und einem bestimmten Teil einer Eingabe respektive eines Eingabebildes ein Skalarprodukt gebildet und ein einzelner Ausgabewert generiert. Der Teil der Eingabe, der jeweils von einem Filter abgedeckt wird, wird als das Wahrnehmungsfeld (engl. *local receptive field*) bezeichnet. Der Filter wird schrittweise über die gesamte Eingabe, also über sämtliche Neuronen oder Pixel, verschoben und die Rechenoperation wiederholt (vgl. [Dadhich, 2018], Kapitel *Convolutional Neural Networks*, Abschnitt *The convolutional layer*). Das Resultat dieses Vorgangs ist eine *Feature Map*, die für das Erkennen eines bestimmten Merkmals zuständig ist. Jeder Filter generiert jeweils eine eigene Feature Map.
Zu den Hyperparametern eines Conv-Layers zählen die Anzahl der Filter, deren Größe, die *Schrittweite* (engl. *stride*) und die Strategie des *Paddings*. Die Schrittweite bestimmt um wie viele Neuronen oder Pixel ein Filter nach jeder einzelnen Operation verschoben wird. Über die Wahl der Padding-Strategie wird die Randbehandlung eines Bildes definiert. Die möglichen Strategien sind das *Full-*, *Same-* oder *Valid-Padding*. Dabei ist bei CNNs das Same-Padding die gebräuchlichste Strategie, da bei dieser die Höhe und Breite der Eingabe respektive des Eingabebildes erhalten werden kann und so eine komfortablere Gestaltung von Netzarchitekturen möglich ist (vgl. [Raschka, 2018], S. 495). Mit den vorab aufgeführten Parametern wird die Größe der Ausgabe eines Conv-Layers bestimmt. (vgl. [Dadhich, 2018], Kapitel *Convolutional Neural Networks*, Abschnitt *The convolutional layer*).

Neben dem Convolutional-Layer ist der Pooling-Layer ein weiterer typischer Bestandteil eines CNNs. Dieser weist keine eigenen Gewichte auf und bewirkt in einem Netzwerk eine Reduzierung der Rechenlast sowie der Parameteranzahl. Dazu zieht der Pooling-Layer jeweils einzelne *Subsamples*, äquivalent zur Größe des angewendeten Filters, aus seinem Eingang und generiert so ein verkleinertes Abbild des Eingangs. Abhängig von der Art des Poolings wird je Subsample nur ein bestimmter Wert ausgegeben. Ein *Max-Pooling-Layer* übergibt den höchsten Wert des Subsamples an die Ausgabe, während ein *Mean-Pooling-Layer* (auch *Average-Pooling-Layer*) den Mittelwert der einzelnen Werte bildet und diesen übergibt. Das Verhalten beider Pooling-Varianten ist anhand eines Zahlenbeispiels in Abbildung 2.6 demonstriert.

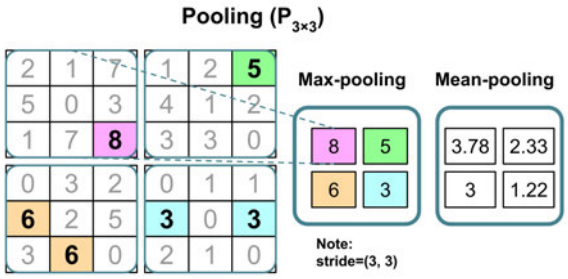


Abbildung 2.6.: Ausgaben einer Max- und Mean-Pooling Operation mit einem 3 x 3 Filter und einer Schrittweite von 3 (vgl. [Raschka, 2019])

Aus den Eigenschaften der beschriebene Layer lassen sich die Grundkonzepte und Vorteile eines CNNs gegenüber einem gewöhnlichen FCNN ableiten. Das erste Grundkonzept sind die begrenzten Wahrnehmungsfelder einzelner Neuronen. Diese führen zu einer deutlichen Reduzierung der Netzwerk-Parameter, da jedes Neuron nur mit den Neuronen innerhalb des eigenen Wahrnehmungsfeldes verbunden ist. Überdies reduziert sich auf diese Weise der auftretende Speicherbedarf. Das zweite Grundkonzept sind geteilte Parameter. Sämtliche Neuronen einer entstandenen Feature Map teilen sich die Parameter ihres korrespondieren den Filters. An dieser Stelle sei erwähnt, dass es sich bei den Werten eines Filters um die einzigen trainierbaren (Hyper-)Parameter des Conv-Layers handelt (vgl. [Dadhich, 2018], Kapitel *Convolutional Neural Networks*, Abschnitt *The convolutional layer*). Das Anwenden eines Filters auf alle Neuronen eines Layers liefert ferner den Vorteil, dass jedes erlernte Merkmale in jedem Bereich einer Eingabe wiedererkannt wird. Man spricht in diesem Zusammenhang von Translationsinvarianz, eine Eigenschaft, die ein CNN besonders effizient für Aufgaben der Sinneswahrnehmungen macht (vgl. [Chollet, 2018b], S. 173). Die Translationsinvarianz kann durch den Einsatz von Max-Pooling Layern innerhalb eines CNNs noch unterstützt werden (vgl. [Gerón, 2020], S. 461). Ein FCNN hingegen erkennt Merkmale nur in den Bereichen, in denen sie erlernt wurden. Eine weitere wichtige Eigenschaft eines CNNs ist die Entwicklung der Dimensionen der Conv-Layer. Üblicherweise werden diese durch das Anwenden von Filtern tiefer, weisen also mehr Feature Maps auf. Gleichzeitig reduzieren

sich aber die *Breite* und *Höhe*. Ein Filter erfasst folglich größere Bildbereiche und kann so *Low-Level-Merkmale* zu komplexeren *High-Level-Merkmalen* kombinieren, um abschließend über das feedforward FCNN respektive den Output-Layer Vorhersagen zu treffen.

2.2. Objekterkennung

Dieses Kapitel behandelt die Deep-Learning-basierte Objekterkennung. Bei der Aufgabe der Objekterkennung, die darin besteht, Objekte in einem Bild zu klassifizieren und zu lokalisieren, handelt es sich um eine typische Problemstellung des maschinellen Sehens. Eine üblich Ausgabe besteht aus einer *Bounding Box* (BB) zur Markierung eines Objekts sowie einer Klassenzuordnung, dargestellt beispielsweise durch ein Label. Eine hier nicht behandelte Alternative ist die *Semantische Segmentierung*. Bei dieser wird auf Pixelebene jedes einzelne Pixel klassifiziert und entsprechend seiner Klassenzugehörigkeit gekennzeichnet.

Als mögliche Schwierigkeiten nennt [Süße and Rodner, 2014] auf Seite 590 f. Verdeckungen von Objekten und einen Hintergrund der Objekt und Bildelemente enthält, die nicht erkannt werden sollen. Beides erschwert die Erkennung der Zielobjekte. Hinzu kommen Erschwernisse, die sich aus dem *Intraklassenabstand* und der *Interklassendistanz* ergeben können. Der Intraklassenabstand bezieht sich auf die variierende Darstellung der Objekte einer einzelnen Klasse zum Beispiel durch unterschiedliche Rotation, Skalierung oder andere Perspektiven, aus der sich Erschwernisse ergeben können. Probleme hinsichtlich der Interklassendistanz basieren auf zu großen Ähnlichkeiten zwischen Objekten unterschiedlicher Klassen.

Aufgrund der bekannten Vorteile eines CNNs ist es eine klassische Methode, diese für die Objekterkennung anzuwenden. Da hierbei das gleiche Objekt mehrfach an leicht unterschiedlichen Positionen erkannt werden kann, ist eine Nachbearbeitung erforderlich. Das entsprechende Verfahren wird als *Non-Maximum Supression* (NMS) bezeichnet. Mit der NMS werden alle überflüssigen Bounding Boxes entfernt, indem jeder BB ein *Objectness-Wert* zugewiesen wird, der die Wahrscheinlichkeit angibt, dass die BB tatsächlich ein bestimmtes Objekt markiert. Anschließend werden alle BBs verworfen, deren Objectness-Werte unterhalb eines Grenzwertes liegen oder die sich zu stark mit einer BB überlappen, die einen höheren Objectness-Wert aufweist (vgl. [Gerón, 2020], S. 488 ff.).

Mit diesem Verfahren erfolgt die Objekterkennung zuverlässig, jedoch langsam, da ein CNN mehrere Durchläufe über ein Bild ausführen muss. *Fully Convolutional Networks* (FCNs) beschleunigen dieses Verfahren. In FCNs werden die Dense-Layer am Ende der CNNs durch Conv-Layer ersetzt. Zum einen resultiert daraus, dass auf Grund der Eigenschaften eines Conv-Layers am Ende eines FCNs Bilder beliebiger Größe verarbeitet werden können. Zum

anderen steigert ein FCN die Effizienz, da es ein Bild direkt vor der Ausgabe nur einmal verarbeiten bzw. dieses nur einmalig ansehen muss (vgl. [Gerón, 2020], S. 490 f.).

Die Objekterkennung umfasst als gängige Hauptkategorien die zweistufigen *Two-Stage-Detektoren* sowie die einstufigen *One-Stage-Detektoren*. Entsprechend der Bezeichnung sind Two-Stage-Detektoren in zwei Hauptschritte unterteilt. Im ersten Schritt werden Merkmale aus einem Bild extrahiert und *Regions of Interest* (ROIs) in Form von Bounding Boxes vorgeschlagen. Eine ROI ist als der Bereich zu verstehen in dem sich ein bestimmtes Objekt befinden könnte. Im zweiten Schritt werden unter Berücksichtigung der extrahierten Merkmale und der vorgeschlagenen ROIs Ausgaben generiert. Das bedeutet, es werden eine oder mehrere finale BBs erzeugt und die zugehörigen Klassenwahrscheinlichkeiten berechnet. Zweistufige Detektoren weisen sich durch sehr genaue, jedoch verhältnismäßig langsame Vorhersagen aus (vgl. [Dadhich, 2018], Section *Methods for object detection*). One-Stage-Detektoren sind hingegen in der Lage, schnellere Vorhersagen zu treffen, da diese ein Netzwerk in einer einzelnen Architektur bilden und keinen Zwischenschritt ausführen. Für die Vorhersage von BBs muss ein Netzwerk also nur einmal durchlaufen werden, wobei das zwischenzeitliche Vorschlagen von ROIs entfällt. Damit sind die einstufigen Detektoren weniger komplex und eignen sich daher besonders für die Verwendung auf Embedded Systemen oder mobilen Endgeräten. Die Genauigkeit ist im Vergleich zu den zweistufigen Detektoren jedoch geringer und speziell kleine Objekte werden weniger gut erkannt.

2.3. Bewertungsverfahren und Metriken der Objekterkennung

Für die Bewertung der Leistungsfähigkeit und Qualität eines Modells oder Algorithmus' zur Objekterkennung existieren unterschiedliche Metriken und Kenngrößen. Simpel ausgedrückt, sollen die möglichen Bewertungskriterium eine Aussage darüber liefern, wie schnell, genau und sicher Objekte erkannt werden.

Die Leistungsfähigkeit kann mit der Geschwindigkeit verbunden werden, mit der ein Modell oder Algorithmus einzelne Frames verarbeiten kann. Daraus ergibt sich die Anzahl der verarbeiteten Frames pro Zeiteinheit, angegeben in *frames per second* (fps).

Ein wesentlicher Aspekt zur Beurteilung der Qualität ist der Grad der Überlappung von vorausgesagten Bounding Boxes und der *Ground Truth*, also den gelabelten, sprich tatsächlichen Bounding Boxes. Die entsprechende Metrik ist die *Intersection over Union* (IoU), auch bekannt als Jaccard Index. Diese ergibt sich nach [Rezatofighi et al., 2019] aus Gleichung 2.14. Darin wird die Schnittmenge der Fläche B einer vorausgesagten Bounding Box und der Ground Truth Fläche GT durch die Vereinigungsmenge von B und GT geteilt.

$$IoU = \frac{|B \cap GT|}{|B \cup GT|} \tag{2.14}$$

Die berechneten IoU-Werte werden eingesetzt, um aus einer Menge von Vorhersagen die *True Positives* (TP) und *False Positives* (FP) zu bestimmen. Wird die IoU als Bewertungsmetrik eingesetzt, ist es erforderlich, einen Schwellenwert zu bestimmen. Im Rahmen der PASCAL VOC Challenge wird beispielsweise ein Schwellenwert von $0,5$ verwendet und darauf basierend die *(mean) Average Precision* (mAP bzw. AP) berechnet (vgl. [Rezatofighi et al., 2019]). Die AP ist eine häufig genutzte Metrik zur Messung der Genauigkeit einer Objekterkennung. Hierbei spielen die *Precision* und der *Recall* eine wesentliche Rolle.
Mit der Precision wird angegeben, wie groß der Anteil der tatsächlichen Positives unter allen angegebenen Positives ist. Die Berechnung der Precision erfolgt mit Gleichung 2.15 (vgl. [Gerón, 2019], S. 91).

$$precision = \frac{TP}{TP + FP} \tag{2.15}$$

Der Recall gibt Auskunft darüber, wie groß der Anteil der erkannten Positives unter allen tatsächlich vorhandenen Positives ist, also wie vollständig Objekte erkannt werden. Berechnet wird dies mit Gleichung 2.16 (vgl. [Gerón, 2019], S. 91).

$$recall = \frac{TP}{TP + FN} \tag{2.16}$$

Die Abhängigkeit zwischen Precision und Recall ist mit der Precision-Recall-Kurve darstellbar. Tendenziell gilt, je höher die Precision, desto geringer der Recall und andersherum. Die AP entspricht der Fläche unter dieser Kurve. Abhängig vom Wettbewerb, in dem die AP zur Bewertung herangezogen wird, unterscheidet sich deren Berechnung. Im Rahmen der PASCAL VOC Challenge wird unter Beachtung eines festen IoU-Grenzwerts jeweils die maximale Precision bei einem Recall von 0 %, 10 %, 20 % und weiter bis 100 % ermittelt und aus den entsprechenden Werten der Mittelwert gebildet. In der COCO Competition wird die AP stattdessen mit unterschiedlichen IoU-Grenzwerten ermittelt.
Wird die AP für unterschiedliche Objektklassen berechnet und daraus der Mittelwert gebildet spricht man von der mean Average Precision (vgl. [Gerón, 2020], S. 494).

Bei den hier beschriebenen Bewertungskriterien fps, IoU und (m)AP handelt es sich um die wesentlichen Metriken mit denen in der Regel jedes Modell oder jeder Algorithmus zur Objekterkennung aussagekräftig bewertet werden kann.

2.4. Embedded Vision Systeme und das Edge-Computing

Unter einem Embedded System versteht man in der Regel ein eigenständiges kompaktes Rechensystem, das in ein größeres Gesamtsystem “eingebettet“, also integriert ist und darin dedizierte Funktionen ausführt. Verfügt ein solches System über eine zusätzliche Kameratechnologie, kann von einem “Embedded Vision System“ gesprochen werden.
Im Kontext dieser Arbeit wird ein Embedded Vision System als eine mobil oder dezentral einsetzbare Recheneinheit kleiner Bauform mit einem eigenen Betriebssystem verstanden, welche mit einem kompakten Aufnahmegerät ausgestattet ist. Das entsprechende übergeordnete Gesamtsystem wird an dieser Stelle nicht genauer spezifiziert, soll aber als ein “intelligentes“ Umfeld im Sinne einer Smart-City angesehen werden. Das übergeordnete Gesamtsystem stellt hier keine maschinelle Anlage oder etwas Äquivalentes dar.
Bei den Recheneinheiten handelt es sich üblicherweise um Single-Board-Computer oder Development-Boards. Diese zeichnen sich typischerweise durch kleine Bauformen, relativ geringe Leistungsaufnahmen aber auch begrenzte Hardware-Ressourcen, beispielsweise hinsichtlich Speicher und Rechenleistung aus. Gängige Aufnahmegeräte im nicht industriellen Umfeld sind häufig konventionelle USB-Kameras oder mittels Breitbandkabel und MIPI CSI Standard angeschlossene Kameramodule ohne Gehäuse.
Eine Anforderung an ein Embedded Vision System ist das Durchführen des Edge-Computing unter möglichst effektiver Ausnutzung der verfügbaren Ressourcen. Das Grundprinzip des Edge-Computings ist es, die Datenverarbeitung in der Nähe der Datenquelle durchzuführen (vgl. [Shi et al., 2016], Section II B.). Speziell in Bezug auf Anwendungen, die auf der AI oder insbesondere dem Deep Learning basieren, ist die Rechenleistung dabei die kritische Größe. Im Rahmen dieser Arbeit handelt es sich bei dem Edge-Computing demnach um die dezentrale Verarbeitung der Bilddaten aufgezeichneter Personen zur Laufweganalyse direkt auf der Recheneinheit, also “at the edge“.
Laut [Shi et al., 2016] bietet das Edge-Computing diverse Vorteile für den jeweiligen Anwendungszweck als auch das Gesamtsystem bzw. das entsprechende Netzwerk. Einige dieser Vorteile sind nachfolgend aufgeführt.

- Reduzierung der Netzwerk- und Bandbreitenauslastung durch einen geringeren Datenverkehr über ein Netzwerk
- Erhöhung der Datensicherheit und des Datenschutzes
- Schnellere Antwortzeiten und geringere Latenzen durch dezentrales Ausführen von Berechnungen und Algorithmen
- Steigerung der Echtzeitfähigkeit

3. Stand der Technik

In diesem Kapitel wird der Stand der Technik im Bereich der Objekterkennung anhand gängiger Algorithmen und Modelle beschrieben, die unter anderem für das Erkennen von Personen ausgelegt sind. Auf Grund der verhältnismäßig schnellen Entwicklung in diesem Technologiefeld ist jedoch zu berücksichtigen, dass sich der tatsächliche Stand der Technik dementsprechend sehr dynamisch entwickelt und jeweils eine begrenzte Gültigkeit aufweist. Vor diesem Hintergrund wird in den nachfolgenden Beschreibungen auch die historische Entwicklung einzelner Algorithmen und Modelle dargestellt.
Zum Abschluss dieses Kapitels werden einige verwandte Arbeiten vorgestellt, in denen die Objekterkennung in Verbindung mit Personen respektive Fußgänger behandelt wird.

3.1. Methoden der Objekterkennung

In den nachfolgenden Abschnitten werden drei populäre Methoden der Objekterkennung vorgestellt. Dabei werden die wesentlichen Funktionsprinzipien sowie die unterschiedlichen Entwicklungsstufen beschrieben.

3.1.1. R-CNN

Das R-CNN [Girshick et al., 2014] zählt zur Kategorie der Two-Stage-Detektoren und ist der erste CNN-basierte Objekt Detektor. Gleichzeitig hat das R-CNN bei der Anwendung auf den PASCAL VOC Datensatz als erstes bewiesen, dass DL-basierte Methoden bei der Objekterkennung enorme Leistungssteigerungen im Vergleich zu den Systemen bewirken, die auf anderen Methoden basieren. Ein R-CNN besteht aus vier Modulen, wobei das erste Modul Kategorie-unabhängige *Regionenvorschläge* (engl. *region proposals*) generiert. Dazu wird die *selective Search* Methode angewendet. Im zweiten Modul werden aus den vorgeschlagenen Regionen unter Anwendung eines CNNs Merkmalsvektoren extrahiert. Da die Fully-Connected Layer des CNNs Vektoren fester Länge fordern, werden die vorgeschlagenen Regionen auf die benötigte Größe verzerrt. Abschließend werden im dritten Modul Support Vektor Machines (SVMs) für die Objektklassifizierung eingesetzt, während Bounding Box Regressoren im vierten Modul die Vorhersage der Bounding Boxes erzeugen.

Eine weiterentwickelte Variante des R-CNN ist das *Fast R-CNN* [Girshick, 2015]. Dieses ist schneller und hat einen geringeren Speicherbedarf für das Speichern von Merkmalen. Hier werden Merkmale einmalig in einem Durchlauf aus dem gesamten Eingabebild extrahiert und einem Region of Interest (ROI) Pooling Layer zugeführt. Dieser extrahiert aus den eingegangenen ROIs unterschiedlicher Größe Feature Maps von gleicher Größe. Auf diese Weise entfallen Verzerrungen und räumliche Informationen der Merkmale bleiben erhalten. Ein weiterer Vorteil ergibt sich aus dem Training. Im Vergleich zum mehrstufigen Prozess des R-CNN, welcher ein Vortrainieren, eine Feinabstimmung sowie zwei weitere Stufen für die Klassifizierung mittels SVMs und die Bounding Box (BB) Regression umfasst, wendet Fast R-CNN einen einstufigen End-to-End Trainingsprozess an (vgl. [Jiao et al., 2019], Abschnitt 3).

Das *Faster R-CNN* ist eine nochmals verbesserte Variante. Dieses setzt sich aus zwei Modulen zusammen. Eines davon ist der Fast R-CNN Detektor. Das zweite Modul, aus dem die Verbesserung resultiert und welches die selective Search Methode ersetzt, ist ein *Region Proposal Network* (RPN). Bei diesem Netzwerk handelt es sich um eine Art FCN, welches für das Vorschlagen von Regionen eingesetzt wird und dabei für jeden Vorschlag, respektive jede Bounding Box einen Objectness-Wert ausgibt. Das RPN greift dabei auf Conv-Layer zu, welche es sich mit dem Detektor teilt. Daraus resultiert im Vergleich zur selective Search Methode eine reduzierte Zeit für das Vorschlagen der Regionen und damit ein insgesamt schnellerer Prozess der Objekterkennung. Neben der Geschwindigkeit verbessert sich auch die Qualität der Regionsvorschläge, wodurch sich zusätzlich die allgemeine Genauigkeit der Objekterkennung steigert (vgl. [Ren et al., 2017]).

In [He et al., 2018] wird eine weitere Architektur mit der Bezeichnung *Mask R-CNN* vorgestellt. Auf Grund des engen Bezugs zur semantischen Segmentierung sei an dieser Stelle nur auf diesen Lösungsansatz verwiesen.

3.1.2. SSD

Der Single Shot Multibox Detector (SSD) zählt zur Kategorie der One-Stage Detektoren und setzt sich aus einem Basisnetzwerk sowie einer zusätzlich ergänzten Hilfsstruktur zusammen. Das Basisnetzwerk entspricht einem gewöhnlichen feed-forward Netzwerk für Klassifizierungsaufgaben, an deren Ende sich anstelle der eigentlichen Klassifizierungslayer jedoch die erwähnte Hilfsstruktur befindet. Bei den Hilfsstrukturen handelt es sich um zusätzliche *Feature Layer*, auch *Convolutional Feature Layer*, deren Größe sich allmählich reduziert. Der Hintergrund ist es, das Erkennen und Vorhersagen von Objekten unterschiedlicher Größe zu ermöglichen. Die SSD-Architektur ist in Abbildung 3.1 dargestellt.

Das SSD Modell verzichtet auf den Schritt, bestimmte Regionen vorzuschlagen und verwendet stattdessen eine feste Anzahl von *Standard Boxen*, die sich ähnlich einem Raster in allen

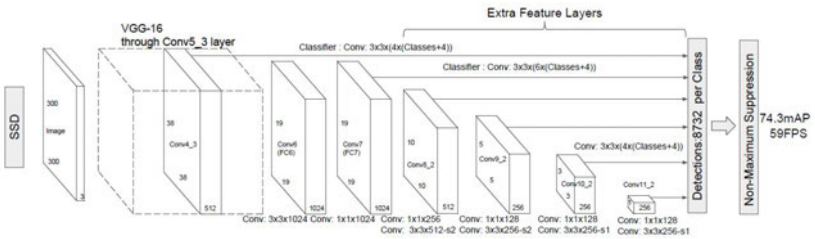


Abbildung 3.1.: Architektur des Single Shot Multibox Detectors nach [Liu et al., 2016]

Bereichen unterschiedlicher Feature Maps befinden. Ein bestimmter Satz von Standard Boxen, jede mit einem anderen Seitenverhältnis, ist dabei jeweils direkt übereinander angeordnet. Jeder dieser Sätze befindet sich zudem innerhalb einer Zelle, in die die Feature Maps unterteilt sind. Um Vorhersagen für die Objekterkennung treffen können und die entsprechenden Bounding Boxes sowie Wahrscheinlichkeiten für die Klassenzugehörigkeit eines Objektes auszugeben, werden sogenannte *Convolutional Filter* eingesetzt. Diese wandern über die Feature Maps und geben entweder Wahrscheinlichkeitswerte oder einen Offset aus. Bei dem Offset handelt es sich um die Verschiebung einer vorhergesagten Bounding Box in Relation zu einer Standard Box, deren Position sich wiederum in Relation zur Feature Map Position befindet.

Da innerhalb eines Netzdurchlaufs die Vorschläge der Bounding Boxes und Klassenzugehörigkeiten aus sämtlichen Feature Maps berechnet und am Ende des Netzwerks zusammen laufen, wird zum Abschluss das Non-Maximum Suppression Verfahren angewendet. Dadurch werden alle überflüssigen Bounding Boxes entfernt und die finale Ausgabe generiert.

Aus dem Verzicht auf einen separaten Schritt zum Vorschlagen von Regionen und dem Bündeln aller Rechenoperationen in einen einzelnen Netzwerkdurchlauf resultiert eine Steigerung der Geschwindigkeit der Vorhersagen im Vergleich zu den R-CNN Varianten, während gleichzeitig gute Genauigkeiten erreicht werden können. (vgl. [Liu et al., 2016]).

3.1.3. YOLO

Ein weiterer populärer One-Stage-Detektor für die Objekterkennung ist YOLO (*You Only Look Once*). YOLO ist als ein CNN implementiert und behandelt die Objekterkennung als ein Reggressionsproblem. Das Netzwerk sagt im Rahmen einer einzelnen Auswertung Bounding Boxes (bzw. deren Koordinaten) sowie Klassenwahrscheinlichkeiten direkt aus einem Gesamtbild vorher. Ein Eingangsbild wird dabei in ein Raster von S x S Zellen geteilt. Eine Zelle ist jeweils für das Erkennen eines Objektes zuständig, dessen Mittelpunkt sich innerhalb der jeweiligen Zelle befindet. Die Ausgabe einer einzelnen Zelle besteht aus B Bounding

Boxes und einem korrespondierenden *Konfidenzwert* (engl. *confidence score*). Dieser Wert ergibt sich aus der Wahrscheinlichkeit, dass eine Box ein Objekt enthält und der IoU zwischen Bounding Box und einer *Ground Truth Box*. Die Ausgabe umfasst ferner die Angabe von Klassenwahrscheinlichkeiten der möglichen Objektklassen. Mit der YOLO Architektur können zwar schnelle Vorhersagen getroffen werden, die erstmals als echtzeitfähig bezeichnet wurden, allerdings ist die Genauigkeit beim Lokalisieren speziell kleiner Objekte eine Schwachstelle (vgl. [Redmon et al., 2016]).

YOLOv2 ist eine Weiterentwicklung der ersten YOLO Architektur, die speziell den niedrigen *Recall* und die Schwächen bei der Lokalisierung verbessern soll. Mit den Weiterentwicklungen geht zudem eine Steigerung der *Mean Average Precision* (mAP) einher. Die Verbesserungen basieren auf unterschiedlichen Maßnahmen wie der Batchnormalisierung, angewendet auf die Convolutional-Layer (Conv-Layer) und einem ergänzten Prozess zur Feinabstimmung des Klassifizierungsnetzwerks. Des Weiteren verwirft YOLOv2 die Fully Connected Layer der YOLO-Architektur und nutzt stattdessen *Anchor Boxes*. Diese werden für die Vorhersage der Bounding Boxes eingesetzt und dienen als Referenz für deren Offset. Klassenvorhersagen und Objectness-Werte werden ebenfalls auf die Anchor Boxes bezogen, beziehungsweise werden diesen zugewiesen. Diese Maßnahme reduziert die mAP zwar minimal, vereinfacht dem Netzwerk aber den Lernvorgang und verbessert zudem den Recall. Eine zusätzliche Verbesserung der mAP bewirkt der Einsatz des K-Means Clustering auf die Bounding Boxes eines Trainingsdatensatzes. Hieraus resultieren besser skalierte und dimensionierte Anchor Boxes, die ein vereinfachtes Training erlauben und die Objekterkennung verbessern. Eine weitere Neuerung ist ein Klassifizierungs-Backbone mit der Bezeichnung *Darknet-19*. Diese nutzt 19 Conv-Layer und fünf Max-Pooling Layer und benötigt damit bei gleichbleibender Genauigkeit weniger Operationen für das Verarbeiten eines Bildes (vgl. [Redmon and Farhadi, 2016]).

Im Vergleich zu YOLOv2 liefert YOLOv3 lediglich kleinere Modifikationen, die die neue Version etwas größer und genauer machen, jedoch auch langsamer. Zu den wichtigsten Änderungen zählt das neue Netzwerk zur Merkmalsextraktion, genannt *Darknet-53*. Es verwendet nicht mehr 19 sondern 53 Conv-Layer, unter anderem mit den Dimensionen 1 x 1 und 3 x 3. Des Weiteren nutzt YOLOv3 eine multilabel Klassifizierung mit unabhängigen logistischen Klassifizierern, um sich damit Datensätzen anzupassen, die viele sich überlappende Label aufweisen (vgl. [Jiao et al., 2019], S. 6). Eine weitere Neuerung ist die Anwendung drei unterschiedlich skalierter Feature Maps für die Vorhersage von Bounding Boxes und dementsprechend unterschiedlich großen Objekten.

Abschließend wird über die in Abbildung 3.2 dargestellte YOLOv3-Architektur durch den letzten Conv-Layer ein 3-D Tensor ausgegeben, der die Klassenwahrscheinlichkeiten, Objectness-Werte und Bounding Boxes, respektive deren Koordinaten beinhaltet (vgl. [Redmon and Farhadi, 2018]).

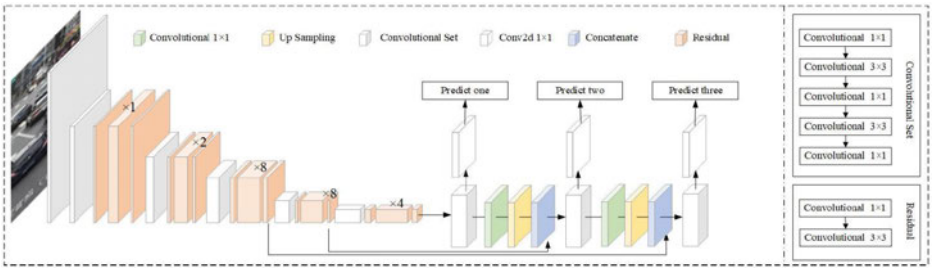


Abbildung 3.2.: Yolov3-Architektur aus [Mao et al.]

Es existieren bereits weitere YOLO Versionen die an dieser Stelle jedoch nicht vorgestellt werden, da das wesentliche Funktionsprinzip sowie die YOLO-Entwicklung anhand der ersten drei Versionen anschaulich darstellbar ist.

3.2. Verwandte Arbeiten

In diesem Kapitel werden Arbeiten bzw. Projekte vorgestellt, die ähnliche Aufgabenstellungen im Vergleich zum Thema dieser Masterarbeit behandeln. Zwei Beispiele zielen darauf ab, den Einsatz von Embedded Systemen zum Zweck der Objekterkennung als auch der Objektverfolgung zu beschreiben und zu beurteilen.

Projekt „Egde-Based Street Object Detection“ [Nagaraj et al., 2017]

In diesem Projekt verwenden die Autoren einen NVIDIA Jetson TX2 und Bildmaterial von Verkehrsüberwachungskameras. Die Aufgabe dieses Embedded Systems besteht in der Objekterkennung im Straßenverkehr mittels Edge-Computing und dies in Echtzeit. Das eingesetzte DL-Modell, hier YOLO in Verbindung mit dem DarkNet Framework, erkennt 14 Objekte, darunter Fußgänger und Personengruppen, und erreicht eine durchschnittliche Genauigkeit von 25 %. Als alternativen Lösungsansatz nutzen die Autoren *DetectNet* mit dem *DIGITS* Framework [Tao et al., 2016], erzielen damit aber schlechtere Resultate in Bezug auf die Erkennung vieler Objekte.

Projekt „Deep Learning-Based Real-Time Multiple-Object Detection and Tracking from Aerial Imagery via a Flying Robot with GPU-Based Embedded Devices“ [Hossain and Lee, 2019]

Hier werden unterschiedliche echtzeitfähige Embedded Systeme sowie unterschiedliche DL-Algorithmen zur luftgestützten Objekterkennung und Objektverfolgung an Bord eines Multicopters getestet.
Bei den Embedded Systemen handelt es sich um die Development-Boards TX1, TX2 und AGX Xavier der NVIDIA Jetson Familie. Im Rahmen des Projektes verwenden die Autoren zudem weitere Embedded Systeme, auf die an dieser Stelle jedoch nicht näher eingegangen wird. Bei der eingesetzten Kamera handelt es sich um eine 5 MP USB 3.0 Kamera.
Die Autoren setzen für die Aufgabe der Objekterkennung die nachfolgenden Modelle ein:

- YOLOv2, YOLOv2 Tiny
- YOLOv3, YOLOv3 Tiny
- SSD
- Faster R-CNN

Für die Objektverfolgung wird DeepSORT in Verbindung mit YOLOv3 eingesetzt.

Die Untersuchungen haben gezeigt, dass sich bei der Objekterkennung hinsichtlich der Geschwindigkeit als Bewertungskriterium die besten Resultate mit dem leistungsstärksten Embedded System, dem AGX Xavier, und dem SSD Modell erreichen lassen. Hiermit kann eine Framerate von 35-48 fps erzielt werden. Dementsprechend zeigt der AGX Xavier mit 10 fps auch bei der Objektverfolgung die besten Resultate. Auch bei der Genauigkeit schneidet der Xavier am besten ab.
Ferner zeigt sich, dass die leistungsschwächeren Geräte, speziell der TX1, für die Objekterkennung teilweise auf die Nutzung der Tiny-Varianten (vgl. [Hossain and Lee, 2019], Abschnitt 3.1.3) beschränkt sind, um annehmbare Framerates erreichen zu können. Aufgrund der geringeren Anzahl an Layern, die die Tiny-Varianten aufweisen, verschlechterte sich folglich die Genauigkeit. Daraus resultiert wiederum eine schlechte bis fehlende Eignung für die Objektverfolgung.

4. Anforderungsanalyse

Die in diesem Kapitel beschriebene Anforderungsanalyse dient als Grundlage für die Konzeption und Realisierung des Embedded Vision Systems. Die aufgeführten Anforderungen basieren auf der Zielsetzung aus Kapitel 1.1, respektive der dort formulierten Aufgabe und den zu realisierenden Funktionalitäten.
Die Anforderungsanalyse ist in Teilanalysen untergliedert, die sich auf das Embedded Vision System in Gänze, sowie im Einzelnen auf die auszuwählenden Systemelemente, also das Development-Board, den Datensatz und das Modell beziehen. Hinsichtlich des Begriffs "Modell" sei hier darauf hingewiesen, dass dieser sinngemäß ein künstliches neuronales Netz oder auch einen DL Algorithmus umfasst. Die Teilanalysen werden in den nachfolgenden Unterkapiteln 4.1, 4.2, 4.3 und 4.4 dargestellt.
In dem Kapitel 4.5 sind die Kernaussagen sämtlicher Anforderungen der vorangegangenen Kapitel in einer Übersicht zusammengefasst. Den einzelnen Anforderungen ist jeweils eine individuelle ID zugewiesen. Diese ist in Klammern gesetzt und besteht aus der Ziffer *A* für "Anforderung", der Kapitelnummer und einer laufenden Nummer. So handelt es sich bei (*A.4.2-01*) beispielsweise um die erste Anforderung an das Development-Board in Kapitel 4.2. Diese IDs finden sich ebenfalls in den Teilanalysen, um einen eindeutigen Bezug zwischen der Anforderungsübersicht und den ausformulierten Anforderungen herzustellen.

4.1. Anforderungen an das Gesamtsystem

In diesem Unterkapitel werden die Anforderungen beschrieben, die sich an das Embedded Vision System im Ganzen stellen. Hierbei werden primär die Installation bzw. Einbringung des Systems in den Einsatzbereich sowie die elementaren Aspekte zur Realisierung der Funktionalität betrachtet. Das Ziel ist es zunächst, auf übergeordneter Ebene zu vermitteln, welche grundsätzlichen Anforderungen zu erfüllen sind, um die in dieser Arbeit gestellte Aufgabe realisieren zu können. Detaillierte Anforderungen an einzelne Bestandteile bzw. Elemente des Systems folgen in den anschließenden Unterkapiteln.
Um eine Analyse der Laufwege von Personen durchführen zu können, muss das Embedded Vision System in einer exponierten Lage montiert oder positioniert sein. Dabei ist hinsichtlich der Höhe zu beachten, dass Personen aus einem geeigneten Aufnahmewinkel (A.4.1-01)

dargestellt werden. Als Richtwert ist eine Schrägansicht aus einem 45°-Winkel mit einer Toleranz von $\pm 15°$ in Bezug auf die Vertikale vorgesehen. Dieser stellt hinsichtlich der Kameraperspektive den Mittelwert zwischen der Normalansicht auf Augenhöhe und der Aufsicht aus einem 90°-Winkel (*Top Shot*) dar. Ferner ist zu beachten, dass der aufgezeichnete Gesamtbereich einer geeigneten Fläche entspricht (A.4.1-02). Ein zu geringer Aufnahmebereich erlaubt keine sinnvolle Laufweganalyse, während ein zu großer Bereich bewirkt, dass Personen klein bzw. mit wenigen Pixeln dargestellt werden und die Personenerkennung erschwert wird. Um im Rahmen dieser Arbeit ausreichend lange Laufwege erfassen und analysieren zu können, wird eine Fläche von $100\,m^2$ als Richtwert bestimmt. Dabei soll keine Seitenlänge geringer als 8 m sein.

Überdies stellen sich die nachfolgend aufgeführten Anforderungen an den funktionsbezogenen Anteil des Embedded Vision Systems. Zunächst ist es erforderlich, innerhalb des Aufnahmebereichs einzelne Teilbereiche bzw. Regions of Interest zu identifizieren und mittels geeigneter Implementierung festzulegen sowie zu visualisieren (A.4.1-03). Der Zweck besteht darin, im Rahmen der Analyse Bereiche zu ignorieren, in denen kein Verkehr von Transportplattformen oder sonstigen Fahrzeugen zu erwarten ist und dementsprechend mit hoher Wahrscheinlichkeit keine Behinderungen von Personen auftreten werden. Für die Teilbereiche gilt es, eine plausible Größe zu bestimmen (A.4.1-04). Als Richtwert soll von $20\,m^2$ ausgegangen werden, wobei keine Seitenlängen unter 3,5 m liegen sollte. Eine zu feine Unterteilung führt gegebenenfalls zu einer starken Fragmentierung bezüglich der Identifizierung häufig oder selten genutzter Teilbereiche. Auf diese Weise entsteht zwar eine große Informationsmenge, die Aussagekraft der Laufweganalyse kann jedoch negativ beeinflusst werden. Zu große Teilbereiche reduzieren die Aussagekraft der Analyse ebenfalls, da eine zu große Verallgemeinerung stattfindet. Daraus resultiert ein Informationsverlust, da sämtliche Informationen zu möglicherweise stark unterschiedlich frequentierten Laufwegen innerhalb dieser Teilbereiche nicht berücksichtigt werden. In beiden Fällen bietet eine Laufweganalyse lediglich eine geringe Unterstützung für die Wegeplanung mobiler Transportplattformen. Perspektivische Verzerrungen sind grundsätzlich zu beachten und bei der Implementierung zu berücksichtigen (A.4.1-05).
Damit die Laufweganalyse realisiert werden kann, ist es notwendig, erfassen zu können, in welchen Teilbereichen sich aufgezeichnete Personen befinden (A.4.1-06). Der gesamte Analyseprozess hat in Abhängigkeit zeitlicher Intervalle zu erfolgen (A.4.1-07).

4.2. Anforderungen an das Development-Board

Als Basis des zu entwickelnden Embedded Vision Systems ist ein speziell für KI-Anwendungen entwickeltes Development-Board zu bestimmen (A.4.2-01), welches als

Bestandteil eines hardwareseitig betriebsbereiten Development-Kits verfügbar ist (A.4.2-02). Es ist nicht vorgesehen, ein eigenes System aus einzelnen Komponenten wie beispielsweise Trägerboard, Ein-Chip-System, Prozessor, Spannungsversorgung und sonstiger Peripherie zu entwerfen und zu montieren. Leistungsbezogen ist es erforderlich, ein neuronales Netz auf dem Development-Board einsetzen zu können (A.4.2-03). Dazu sollte dieses mit einer geeigneten Processing Unit ausgestattet sein. Ferner ist die Verfügbarkeit einer Kamera erforderlich, die mit dem Development-Board kompatibel ist (A.4.2-04) und nach dem Plug-and-Play Prinzip direkt betrieben werden kann. Die Kompatibilität gewährleistet einen geringen Einrichtungsaufwand und beugt möglichen Problemen bei der Inbetriebnahme vor.

Bei den nachfolgend formulierten Anforderungen ist zu beachten, dass sich diese stellenweise nicht auf Absolutwerte beziehen, sondern qualitativ zu verstehen sind. Die aufgeführten Anforderungen stellen Kriterien dar, anhand derer die Daten der einzelnen Development-Boards miteinander verglichen und infolgedessen bewertet werden sollen.
In Hinsicht auf die technischen und physikalischen Eigenschaften soll das Development-Board einen möglichst geringen Energiebedarf (A.4.2-05) von maximal 10 W und eine geringe erforderliche Versorgungsspannung (A.4.2-06) von maximal 5 VDC aufweisen. Die Abmessungen (A.4.2-07) sollen eine Länge und Breite von jeweils 100 mm nicht überschreiten und das Gewicht (A.4.2-08) weniger als 150 g betragen. Die jeweiligen Hintergründe dieser Anforderungen werden im Nachfolgenden beschrieben.
In Abhängigkeit der gewählten Spannungsquelle kann der Energiebedarf unmittelbar die Länge der Einsatzzeit des Gesamtsystems bestimmen. Speziell im Batterie- oder Akkubetrieb handelt es sich bei dem Energiebedarf um eine kritische Größe. Die Höhe der mindestens erforderlichen Versorgungsspannung sollte möglichst gering sein, da diese ein bestimmender Faktor für die Dimensionierung der Spannungsquelle ist. Auch hier handelt es sich speziell bei einem Batterie- beziehungsweise Akku-Betrieb um einen wichtigen Aspekt. Bei den angegebenen 5 VDC handelt es sich um einen typischen Wert für elektrische Kleingeräte. Geringe Abmessungen und ein niedriges Gewicht erhöhen die Flexibilität hinsichtlich möglicher Einsatz- und Montageorte sowie der Installationsmöglichkeiten. Darüber hinaus handelt es sich bei einem geringen Formfaktor um ein klassische Anforderung, die sich an ein Embedded System stellt. Die genannten Werte orientieren sich grob an dem Raspberry Pi (4), der als klassischer Single-Board-Computer betrachtet werden kann. In diesem Zusammenhang ist das geforderter Maximalgewicht von 150 g jedoch deutlich höher angesetzt, da beispielsweise auch Kühlkörper berücksichtigt werden sollen.
Neben den zuvor geschilderten Anforderungen sind außerdem ein großer Arbeitsspeicher (A.4.2-09) sowie eine hohe Rechen- beziehungsweise Prozessorleistung (A.4.2-10) entscheidende Anforderungen, die an das Development-Board gestellt. Der verfügbare Arbeitsspeicher sollte zumindest 1 GB betragen und die Rechenleistung bei mindestens 0,5 TFLOPS (engl. für *Tera Floating Point Operations Per Second*) liegen. Durch einen großen Arbeitsspeicher sollen Geschwindigkeitseinbußen infolge großer Datenaufkommen

oder mehrerer zeitgleich ablaufender Prozesse und Tasks möglichst gering gehalten werden. Eine hohe Rechenleistung ist ein entscheidender Faktor für die Einsetzbarkeit eines neuronales Netzes. Abhängig von der verfügbaren Rechenleistung kann unter Berücksichtigung der typischen Einschränkungen eines Embedded Systems für die vorliegende Aufgabe ein mehr oder weniger tiefes, rechenintensives und komplexes neuronales Netz bzw. Modell ausgewählt werden. Damit stehen unmittelbar wichtige Metriken im Zusammenhang, wie beispielsweise die Genauigkeit, mit der Personen erkannt werden oder die Geschwindigkeit bei der Bildverarbeitung.
In Bezug auf den Softwareanteil soll das Development-Board eine möglichst große Anzahl an DL-Frameworks unterstützen bzw. für deren Anwendung geeignet sein (A.4.2-11). Daraus ergibt sich der Vorteil, ein Framework einsetzen zu können, welches sich entsprechend der jeweiligen Eigenschaften bestmöglich für die im Rahmen dieser Arbeit zu implementierenden Funktionalitäten eignet. Mögliche Einschränkungen, die einzelne oder wenige Frameworks beispielsweise durch nicht vorhandene Module aufweisen, können durch eine größere Auswahl umgangen werden. Ferner ist zu beachten, dass sämtliche Softwareanteile aus Open Source Quellen beziehbar sind (A.4.2-12). Development-Boards, die den Einsatz kostenpflichtiger und proprietäre Software erfordern, sind nicht zu berücksichtigt. Die Investitionskosten für das Development-Board respektive Development-Kit an sich sind möglichst gering zu halten und sollen einen finanziellen Rahmen von maximal €500,00 nicht überschreiten (A.4.2-13).
Zusammengefasst beziehen sich die zuvor aufgeführten Anforderungen auf einen Großteil aller Aspekte und Eigenschaften, die ein technisches System aufweist. Da ein einzelnes Development-Board realistischerweise nicht in allen Bereichen optimale Ergebnisse liefern wird, werden die Anforderungen in die Kategorien I und II unterteilt und gewichtet. Dadurch werden die Anforderungen hinsichtlich ihrer Relevanz unterschieden. Anforderungen der Kategorie I sind für die vorliegende Arbeit von unmittelbarer Relevanz oder gelten als eine Grundbedingung. Diese müssen ausnahmslos erfüllt sein, damit ein Development-Board innerhalb des Auswahlverfahrens berücksichtigt wird. So werden nicht relevante und ungeeignete Hardware-Systeme im Sinne einer Vorauswahl ausgeschlossen. Die entsprechenden Anforderungen sind:

- A.4.2-01 Spezielle KI-Hardware in Form eines Development-Boards
- A.4.2-02 Verfügbar als Bestandteil eines Development-Kits
- A.4.2-03 Eignung für den Einsatz neuronaler Netze
- A.4.2-04 Verfügbarkeit einer kompatiblen Kamera
- A.4.2-13 Investitionskosten unterhalb des Richtwertes von €500,00

Die übrigen Anforderungen zählen zu der Kategorie II. Abhängig von der Relevanz für die vorliegende Arbeit erhalten diese Anforderungen durch die Vergabe der Gewichtungen unterschiedliche Prioritäten. Die Gewichtungen werden in der Anforderungsübersicht in Unterkapitel 4.5 aufgeführt und erläutert.

4.3. Anforderungen an den Datensatz zur Personenerkennung

Im Nachfolgenden werden die Anforderungen an den Datensatz beschrieben, der im Rahmen dieser Arbeit genutzt werden soll. Hierzu ist ein Datensatz zu wählen, welcher für die Objekterkennung oder -verfolgung vorgesehen ist (A.4.3-01). Entsprechend der Aufgabe dieser Arbeit soll es sich dabei um einen speziell für Personen oder Personengruppen bestimmten Datensatz handeln (A.4.3-02). In diesem Zusammenhang sind Datensätze zu vermeiden, deren Bilder vermehrt Gruppen mit hoher Personendichte darstellen, auf denen dementsprechend zu Großteilen verdeckte Personen abgebildet werden (A.4.3-03). Als Gruppe wird hier eine Personenmenge von fünf oder mehr eng beieinander stehender Personen verstanden. Ein Datensatz weist dann vermehrt Gruppen auf, wenn dies in etwa über 20 % (Richtwert) der Bilder eines Datensatzes der Fall ist[1]. Des Weiteren muss der Datensatz vorverarbeitet sein, also aus vollständig oder zu Großteilen gelabelten Bilddaten bestehen. Hier ist die Vorverarbeitung als das Vorhandensein von Ground Truth Informationen bzw. den entsprechenden Bounding Boxes zu verstehen (A.4.3-04). Eine zeitaufwändige Nachbearbeitung des Datensatzes soll dadurch vermieden oder in einem geringen Umfang gehalten werden.
Die nachfolgenden Anforderungen leiten sich im Wesentlichen aus der Kameraperspektive des Embedded Vision Systems ab, die aus der vorgesehenen Position bzw. dem Einsatzort in exponierter Lage resultiert. Entsprechend des Richtwerts für die Kameraperspektive von 45° ist ein Datensatz zu wählen, der Personen aus einem ähnlichen Winkel abbildet. Damit soll erreicht werden, dass das Modell die speziellen Erscheinungsmerkmale von Personen aus dieser Perspektive erlernen kann (A.4.3-05). Im Zusammenhang mit der Höhe des Einsatzortes ist ebenfalls zu beachten, dass die Größe der im Datensatzes dargestellten Personen tendenziell der erwarteten Größe der Personen entspricht, die während der praktischen Laufweganalyse erfasst werden (A.4.3-06). Damit ein mit dem Datensatz trainiertes Modell nicht zu speziell auf eine bestimmte Personendarstellung abgestimmt wird, ist zu beachten, dass die Größe der Personen innerhalb des Datensatzes trotz der groben Vorgabe einer Personengröße in einem gewissen Rahmen variiert. Außerdem muss beachtet werden, dass unterschiedliche Seiten der Personen dargestellt werden, um auch in dieser Hinsicht eine

[1] Da eine Überprüfung dieser Art bei großen Datensätzen nicht realisierbar ist, ist hinsichtlich dieser Anforderungen eine Abschätzung mittels Stichprobe und oberflächlicher Bewertung zulässig.

Variation der optischen Merkmale sicherzustellen (A.4.3-07). Zusammengefasst ist es demnach erforderlich, einen Datensatz mit repräsentativen Bilddaten zu verwenden, um später gute Vorhersagen des Modells zu ermöglichen. Ein weiterer Aspekt ist das Aufnahmeumfeld bzw. der Hintergrund, vor dem Personen dargestellt werden. Um einen Bezug zu etwaigen Einsatzorten des Embedded Vision Systems herzustellen, soll es sich um verkehrsberuhigte Wege, Straßen, oder ähnliche Bereiche mit sporadischem oder keinem Verkehrsaufkommen handeln, bei denen wechselnde Lichtverhältnisse vorteilhaft sind (A.4.3-08).
Um den Download großer Datenmengen zu vermeiden und darüber hinaus die Möglichkeit besteht, den Datensatz bei Bedarf lokal auf dem Embedded Vision System sichern zu können, darf der Speicherbedarf keinen unverhältnismäßig hohen Anteil des insgesamt verfügbaren Speicherplatzes beanspruchen. Als Richtwert wird hier ein Speicherbedarf von maximal 1 GB bestimmt (A.4.3-09).

4.4. Anforderungen an das Modell

Innerhalb dieses Unterkapitels werden die Anforderungen formuliert, die sich im Kontext dieser Arbeit an ein Modell stellen. Diese Anforderungen orientieren sich neben dem vorgesehenen Verwendungszweck überdies an den charakteristischen Beschränkungen eines Embedded Systems. Diese weisen in der Regel einen verhältnismäßig geringen Arbeitsspeicher sowie eine begrenzte Rechenleistung auf. Diese Parameter sind bei einem *Commercial-off-the-shelf* (kurz COTS) Produkt nicht beeinflussbar, ohne eine Änderung am Produkt (hier das Development-Board) vorzunehmen.
Damit das Modell einen möglichst geringen Bedarf an Arbeitsspeicher und Rechenleistung aufweist, ist ein geeigneter Objektdetektor-Typ zu verwenden. Einstufige Detektoren weisen im Vergleich zu zweistufigen Detektoren eine einfachere Architektur und geringere Komplexität auf und haben so einen positiven Effekt auf den erforderlichen Arbeitsspeicher und die erforderliche Rechenleistung. Aus diesem Grund ist ein einstufiger Detektor (A.4.4-01) für das Embedded Vision System auszuwählen. Darüber hinaus soll ein Modell verwendet werden, welches eine geringe Anzahl an (Hidden) Layern aufweist. Als Richtwert wird hier eine Anzahl von 20 Layern angesetzt (A.4.4-02). Ein weiteres Kriterium ist die Anzahl der Modellparameter. Diese sollte möglichst gering sein (A.4.4-03), um auf diese Weise Latenzen bei der Ausführung des Modells sowie den erforderlichen Speicherbedarf zu reduzieren.
Hinsichtlich des Verwendungszwecks der Objekterkennung existieren besonders etablierte respektive häufig für diese Aufgabe eingesetzte Modelle. Ein entsprechendes Modell ist auch im Rahmen dieser Arbeit auszuwählen (A.4.4-04). Das auszuwählende Modell soll darüber hinaus speziell für mobile oder embedded Geräte geeignet sein (A.4.4-05). Außerdem ist es erforderlich, dass das Modell mehrere Personen simultan kann (A.4.4-06).
Zwei grundsätzliche Anforderungen an ein Modell beziehen sich auf die Aspekte Genauigkeit

und Geschwindigkeit. Zum einen soll ein Modell eine möglichst hohe Genauigkeit aufweisen, um bei einem gegebenen Eingang eine exakte Objekterkennung als Ausgabe zu liefern und zum anderen sollte ein Modell schnell sein, also eine größtmögliche Anzahl an Bildern pro Sekunde verarbeiten können, um sich für Anwendungen mit Echtzeitanforderungen zu eignen. Hier gilt es, abhängig von dem jeweiligen Verwendungszweck, einen geeigneten Kompromiss zu finden, da die Verbesserung des einen Aspekts tendenziell negative Auswirkungen auf den jeweils anderen Aspekt hat. Im Rahmen der vorliegenden Arbeit ist es erforderlich, dass das Modell eine ausreichende Geschwindigkeit, um die Positionen bewegter Personen in möglichst kurzen Abständen zu erfassen. Dies ist erforderlich, damit Personen innerhalb einzelner Teilbereiche erkannt wird, bevor diese den entsprechenden Teilbereich wieder verlassen. Unter der Annahme einer Schrittgeschwindigkeit von 1,4 m/s (ca. 5 km/h) sollen Personen mindestens alle 0,5 m einmal durch das Modell erfasst bzw. erkannt werden. Bei der angenommenen Schrittgeschwindigkeit benötigt eine Person ca. 0,36 s um sich einen halben Meter fortzubewegen. Das Modell dementsprechend eine Verarbeitungsgeschwindigkeit von mindestens 2,78 fps ermöglichen. Dieser Wert stellt aufgerundet auf 3 fps eine weitere Anforderung an das Modell dar (A.4.4-07). Hierbei handelt es sich zwar um einen geringen Wert, da die Laufweganalyse jedoch nicht in Echtzeit erfolgt, sondern auf aufgezeichneten Videosequenzen basiert, ist eine höhere Verarbeitungsgeschwindigkeit nicht erforderlich. Damit die Laufweganalyse hinsichtlich der erfassten Personenzahlen ein ausreichend verlässliches Ergebnis liefert, sollen Personen mit einer möglichst hohen Sicherheit erkannt werden (A.4.4-08). Ein letzter Aspekt ist der Support des Modells (A.4.4-09). Hierzu zählen die Verfügbarkeit des Modells und vortrainierter Gewichte, der Umfang vorhandener Fachpublikationen mit Implementierungsbeispielen, die Menge an Recherchematerial sowie die Präsenz auf einschlägigen Webseiten wie beispielsweise GitHub.

4.5. Anforderungsübersicht

Die in den vorangegangenen Unterkapiteln formulierten Anforderungen sind in den Tabellen 4.1, 4.2, 4.3 und 4.4 zusammengefasst. Neben den Anforderungs-IDs beinhalten die Tabellen in jeder Zeile eine Kurzbeschreibung der einzelnen Anforderungen sowie einen Gewichtungsfaktor (kurz Gew.fkt.) entsprechend der jeweiligen Relevanz.
Der Zweck dieses Unterkapitels besteht darin, eine Übersicht aller Anforderungen zu liefern und darüber hinaus einen Eindruck zu vermitteln, welche Kriterien und Aspekte bei der Entwicklung und Konzeption eines Embedded Vision Systems zu beachten und zu berücksichtigen sind. Die aufgeführten Gewichtungsfaktoren dienen dazu, die unterschiedlichen Prioritäten der einzelnen Anforderungen für die vorliegende Arbeit zu verdeutlichen.
Die in Tabelle 4.1 aufgeführten Anforderungen sind gleichwertig respektive neutral gewichtet, da es sich bei diesen um elementare Anforderungen handelt, die nicht für den Vergleich

unterschiedlicher Lösungsmöglichkeiten vorgesehen sind.

Tabelle 4.1.: Übersicht der Anforderungen an das Gesamtsystem und den korrespondierenden Gewichtungsfaktoren

ID	Anforderung	Gew.fkt.
A.4.1-01	Geeigneter Aufnahmewinkel für die Laufweganalyse (Richtwert: 45°, ± 15°)	1
A.4.1-02	Geeignete Größe des Aufnahmebereichs (Richtwert: Fläche mind. 100 m^2, keine Seite kürzer als 8 m)	1
A.4.1-03	Teilbereiche identifizieren, festlegen und visualisieren	1
A.4.1-04	Geeignete Größe der Teilbereiche (Richtwert: 20 m^2, keine Seite kürzer als 3,5 m)	1
A.4.1-05	Perspektivische Verzerrungen berücksichtigen	1
A.4.1-06	Erfassen, in welchem Teilbereich sich Personen befinden	1
A.4.1-07	Zeitliche Intervalle für den Analyseprozess	1

Die unterschiedlichen Gewichtungen bzgl. der Anforderungen an das Development-Board in Tabelle 4.2 basieren auf nachfolgenden Begründungen.
Die Rechen- und Prozessorleistung (A.4.2-10) stellt ein sehr wichtiges Kriterium für ein Embedded Vision System dar, da diese maßgeblich die Einsatzmöglichkeiten neuronaler Netze bzw. Modelle und deren Leistungsfähigkeit hinsichtlich gängiger Metriken wie der Verarbeitungsgeschwindigkeit von Bildsequenzen oder der Genauigkeit bei der Objekterkennung mitbestimmt. Die Anforderung (A.4.2-10) ist demzufolge höher gewichtet. Eng damit verbunden und ebenfalls höher gewichtet ist der verfügbare Arbeitsspeicher (A.4.2-09). Zur Laufzeit eines neuronalen Netzes respektive Modells können schnell große Anteile des Arbeitsspeichers allokiert werden. Ist nicht ausreichend Speicher, vorhanden läuft das System *out of memory* und ist nicht mehr lauffähig. Darüber hinaus wird auch für die Anzahl der unterstützen und nutzbaren Frameworks (A.4.2-11) eine höhere Gewichtung bestimmt. Diese Anforderung wird als wichtiges Kriterium für Aufgaben mit einem Entwicklungscharakter betrachtet und bedeutet sowohl während des Aufbauprozesses eines Embedded Vision Systems als auch bei möglichen Weiterentwicklungen eine größere Flexibilität sowie weniger Einschränkungen. Die Kriterien A.4.2-07 (geringe Abmessungen) und A.4.2-08 (niedriges Gewicht) sind niedriger gewichtet, da das Embedded Vision System nicht innerhalb eines räumlich begrenzten Systems mit einem möglichst niedrigen Gesamtgewicht wie beispielsweise einem Multicopter oder einem fahrerlosen Transportsystem eingesetzt wird. An dem vorgesehenen stationären Einsatzort sind beide Aspekte unkritisch. Um in diesem Zusammenhang dennoch den typischen Eigenschaften eines Embedded Systems zu genügen, werden diese Anforderungen nicht verworfen.

Tabelle 4.2.: Übersicht der Anforderungen an das Development-Board und den korrespondierenden Gewichtungsfaktoren

ID	Anforderung	Gew.fkt.
A.4.2-01	Spezielle KI-Hardware in Form eines Development-Boards	1
A.4.2-02	Verfügbar als Bestandteil eines Development-Kits	1
A.4.2-03	Eignung für den Einsatz neuronaler Netze	1
A.4.2-04	Verfügbarkeit einer kompatiblen Kamera	1
A.4.2-05	Geringer Energiebedarf (max. 10 W)	1
A.4.2-06	Geringe Versorgungsspannung (max. 5 VDC)	1
A.4.2-07	Geringe Abmessungen (max. 100 mm x 100 mm, L x B)	0,5
A.4.2-08	Niedriges Gewicht ($<$ 150 g)	0,5
A.4.2-09	Großer Arbeitsspeicher ($\geq$ 1 GB)	1,5
A.4.2-10	Hohe Rechen- bzw. Prozessorleistung ($>$ 0,5 TFLOPS)	1,5
A.4.2-11	Große Anzahl unterstützter / nutzbarer Frameworks	1,5
A.4.2-12	Open Source / keine Bindung an kostenpflichtige, proprietäre Software	1
A.4.2-13	Investitionskosten unterhalb des Richtwertes von €500,00	1

Die übrigen Anforderungen erhalten eine neutrale Gewichtung, da es sich bei diesen um Aspekte handelt, die für das Embedded Vision Systems von Bedeutung sind, jedoch weder eine übergeordnete Priorität aufweisen noch als zweitrangig zu betrachten sind.

Im Zusammenhang mit der Auswahl des Datensatzes erhalten die Anforderungen A.4.3-04, A.4.3-07 und A.4.3-09 eine höhere Gewichtung. Ohne ausreichend vorhandene Ground Truth Daten bzw. Bounding Boxes (A.4.3-04) kann abhängig von der Größe des Datensatzes ein enormer und für die vorliegende Arbeit unverhältnismäßiger Zeitaufwand entstehen. Die Variation der Personengröße und -orientierung (A.4.3-07) hat für die generelle Nutzbarkeit eines mit dem Datensatz trainierten Modells eine hohe Bedeutung. Um eine generelle Nutzbarkeit zu gewährleisten, muss ein Modell eine ausreichend große Variation an Merkmalen von Personen erlernen. Da ein Embedded System üblicherweise einen begrenzten Speicherplatz bietet, ist der Aspekt des erforderlichen Speicherbedarfs des Datensatzes, ebenfalls höher gewichtet. Die Anforderungen A.4.3-01 und A.4.3-02 sind als grundlegende Anforderungen zu betrachten, die erfüllt sein müssen, damit ein Datensatz im Auswahlverfahren berücksichtigt wird. Ein höhere oder niedrigere Gewichtung ist dementsprechend nicht erforderlich. Die übrigen Anforderungen werden weder als übermäßig noch als wenig bedeutsam eingestuft und demnach neutral gewichtet.

Die wichtigste Anforderung an das Modell ist die spezielle Eignung für mobile oder embedded Geräte (A.4.4-05), da der Einsatz eines embedded Gerätes, hier das Development-Kit, einen wesentlichen Aspekt dieser Arbeit darstellt. Ein weiterer höher gewichteter Aspekt ist

Tabelle 4.3.: Übersicht der Anforderungen an den Datensatz und den korrespondierenden Gewichtungsfaktoren

ID	Anforderung	Gew.fkt.
A.4.3-01	Datensatz für Objekterkennung oder -verfolgung	1
A.4.3-02	Spezieller Datensatz für Personen oder Personengruppen	1
A.4.3-03	Wenig verdeckte Personen (dichte Personengruppen in max. 20 % der Bilder)	1
A.4.3-04	Ground Truth Daten bzw. Bounding Boxes	1,5
A.4.3-05	Darstellungswinkel von Personen ca. 45° ($\pm$ 15°)	1
A.4.3-06	Praxisnahe Personengröße	1
A.4.3-07	Variation der Personengröße und -orientierung	1,5
A.4.3-08	Aufnahmeumfeld mit Bezug zu etwaigen Einsatzorten und wechselnden Lichtverhältnissen	1
A.4.3-09	Verhältnismäßiger Speicherbedarf	1,5

die Wahl eines Modells für die Objekterkennung (A.4.4-04), da auch die Objekterkennung ein zentrales Thema dieser Arbeit ist. Darüber hinaus sollte auch die Verarbeitungsgeschwindigkeit ausreichend hoch sein (A.4.4-07), damit Personen rechtzeitig in den richtigen Teilbereichen erkannt werden. Zudem wird die Anforderung A.4.4-08 höher gewichtet, da Die Anforderungen A.4.4-01, A.4.4-02 und A.4.4-03 erhalten neutrale Bewertungen, da diese in der Regel bereits in Modellen entsprechend (A.4.4-05) berücksichtigt werden. Den übrigen Anforderungen wird ebenfalls eine neutrale Gewichtung zugewiesen, da diese keine überdurchschnittlich hohe Relevanz aufweisen.

Tabelle 4.4.: Übersicht der Anforderungen an das Modell und den korrespondierenden Gewichtungsfaktoren

ID	Anforderung	Gew.fkt.
A.4.4-01	Einstufiger Detektor	1
A.4.4-02	Geringe Anzahl der Layer (Richtwert: 20)	1
A.4.4-03	Anzahl der Modellparameter	1
A.4.4-04	Modell für die Objekterkennung	1,5
A.4.4-05	Modell für mobile oder embedded Geräte	2
A.4.4-06	Simultane Personenerkennung	1
A.4.4-07	Verarbeitungsgeschwindigkeit (3 fps)	1
A.4.4-08	Support	1,5

5. Konzeption

Die in diesem Kapitel beschriebene Konzeption basiert auf den in der Anforderungsanalyse formulierten Anforderungen. Entsprechend der einzelnen Teilanalysen werden in diesem konzeptionellen Teil die Verfahren zur Auswahl des Development-Boards samt kompatibler Kamera, des Datensatzes und des Modells sowie weiterer Softwareanteile in den Unterkapiteln 5.1, 5.2 und 5.3 beschrieben und die jeweiligen Entscheidungen begründet. Die einzelnen Unterkapitel enthalten eine Referenz zu dem Kapitel, in dem die zugehörigen Anforderungen formuliert sind, um so den Zusammenhang zwischen Anforderungsanalyse und der korrespondierenden Konzeption herzustellen. Die referenzierten Anforderungen können darüber hinaus in der Anforderungsübersicht in Unterkapitel 4.5 nachgelesen werden. Des Weiteren ist in diesem konzeptionellen Teil der Entwurf des für die Laufweganalyse erforderlichen Implementierungsanteils beschrieben.
Die in Unterkapitel 4.1 formulierten Anforderungen an das Gesamtsystem werden im Rahmen des Kapitels 6 behandelt bzw. deren Umsetzung dort dargestellt.

5.1. Auswahl des Development-Boards und der Kamera

Im Rahmen dieses Unterkapitels wird das Auswahlverfahren des Development-Boards beschrieben. Dabei wird Bezug auf die Anforderungen des Unterkapitels 4.2 genommen. Für die Entscheidungsfindung werden innerhalb des Auswahlverfahrens Bewertungspunkte an die berücksichtigten Development-Boards vergeben. Die Höhe der vergebenen Punktzahlen orientiert sich an dem Erfüllungsgrad einer Anforderung oder daran, wie gut ein Development-Board eine Anforderung im Verhältnis zu anderen Boards erfüllt. Die Punktevergabe erfolgt nach dem in Tabelle 5.1 beschriebenen Schema. Diese Bewertungen werden mit den Gewichtungen entsprechend Tabelle 4.2 multipliziert und aus den einzelnen Ergebnissen das Gesamtergebnis gebildet. Innerhalb der Tabelle 5.2 finden sich die Gewichtungsfaktoren in Klammern nach den zugehörigen Anforderungs-IDs.
Nach der Auswahl des Development-Boards wird unter Beachtung der Anforderung A.4.2-04 eine Kamera ausgewählt, die mit dem Development-Board kompatibel ist.

5.1.1. Auswahl des Development-Boards

Tabelle 5.1.: Vergabeschema der Bewertungspunkte für Development-Boards

Bewertung	Beschreibung
++	Zwei Bewertungspunkte werden addiert; die Anforderung ist optimal erfüllt;
+	Ein Bewertungspunkt wird addiert; die Anforderung ist erfüllt;
o	Keine Vergabe eines Bewertungspunktes; die Anforderung ist lediglich im Mindesten erfüllt;

Im Rahmen einer allgemeinen Online-Recherche[1] und einer anschließenden Bewertung der infrage kommenden Development-Boards *Jetson Nano*, *Jetson TX2*, *Google Coral Dev Board*, *Open-Q 605 SBC Dev. Kit* und dem *HummingBoard Pulse* in Bezug auf die Anforderungen der Kategorie I (siehe Unterkapitel 4.2), schneiden der Jetson Nano und das Coral Dev Board mit dem besten Gesamtergebnis ab. Die übrigen Development-Boards erzielen speziell bezüglich des preislichen Aspekts geringe Punktzahlen. Das weitere Auswahlverfahren fokussiert sich dementsprechend auf einen Vergleich zwischen dem Jetson Nano und dem Coral Dev Board. Bei dem Vergleich der angeführten technischen Eigenschaften und Daten ist zu berücksichtigen, dass der Jetson Nano und das Coral Dev Board einem Technologiefeld zuzuordnen sind, welches sich dynamisch weiterentwickelt. Dementsprechend können die hier genannten Daten gegebenenfalls auch kurz- bis mittelfristig nicht mehr den aktuellen Stand der Technik repräsentieren. Da die Entscheidung für eines der Development-Boards mit einem Kauf des entsprechenden Development-Kits verbunden ist, können mögliche technische Weiterentwicklungen im Rahmen dieser Arbeit nicht für einen erneuten Vergleich herangezogen werden.

Wie in der Tabelle 5.2 ersichtlich, erzielt der Jetson Nano in Bezug auf die Anforderungen der Kategorie II das bessere Gesamtergebnis und wird dementsprechend als Development-Board bzw. Development-Kit für das Embedded Vision System eingesetzt. Speziell die größere Anzahl der nutzbaren Frameworks (A.4.2-11) spricht für den Jetson Nano [Franklin, 2019]. Während dieser eine größere Auswahl ermöglicht, unterstützt das Coral Dev Board [Google, o. J.c] ausschließlich das Framework *TensorFlow Lite* [Google, o. J.a]. Für den Jetson Nano spricht des Weiteren der größere Arbeitsspeicher (A.4.2-09) von 4 GB RAM im Vergleich zu 1 GB RAM beim Coral Dev Board. Die Vorteile des Coral Dev Boards liegen in dem geringeren Energiebedarf (A.4.2-05), der mit 2 W ledig-

[1] Diese beinhaltete eine Voruntersuchung mehrerer Development-Boards unterschiedlicher Hersteller (siehe Anhang A.2). Die Anforderungen der Kategorie I (siehe Unterkapitel 4.2) haben dabei dazu gedient nicht zulässige Development-Boards auszuschließen.

Tabelle 5.2.: Vergleich der Development-Boards Jetson Nano und Coral Dev Board bzgl. der Anforderungen der Kategorie II

	Development-Board	
Anforderungs-ID	**Jetson Nano**	**Coral Dev Board**
A.4.2-05 (1) Energiebedarf	o	+
A.4.2-06 (1) Versorgungsspannung	++	++
A.4.2-07 (0,5) Abmessungen	o	+
A.4.2-08 (0,5) Gewicht	++ (nicht berücksichtigt)	/ (nicht berücksichtigt)
A.4.2-09 (1,5) RAM	+	o
A.4.2-10 (1,5) GFLOPS / TOPS	o	+
A.4.2-11 (1,5) Frameworksupport	++	o
A.4.2-12 (1) Open Source	+	+
Gesamtergebnis	**7,5**	**6**

lich 20 % bis 40 % des Energiebedarf des Nanos beträgt und in den geringeren Abmessungen (A.4.2-07). Diese betragen 88 mm x 60 mm (Länge x Breite) beim Coral Dev Board und 80 mm x 100 mm (Länge x Breite) beim Jetson Nano. Das Gewicht (A.4.2-08) wird aufgrund der fehlenden Angabe bei dem Coral Dev Board in dem Vergleich als Kriterium nicht berücksichtigt und demnach nicht bewertet. Die bei dem Jetson Nano aufgeführte Bewertung des Gewichts ergibt sich aus dem Vergleich zu den nicht mehr aufgeführten Development-Boards. Aus Gründen der Vollständigkeit ist das Kriterium des Gewichts hier aufgeführt, auch wenn es keine Berücksichtigung findet.
Hinsichtlich der Rechen- bzw. Prozessorleistung (A.4.2-10) liegen die Angaben beider Development-Boards in unterschiedlichen Einheiten vor. Während für den Jetson Nano die Einheit *GFLOPS* (engl. *Floating Point Operations Per Second*) verwendet wird, wird die Rechenleistung des Coral Dev Boards in der Einheit *TOPS* (engl. *Tera Operations Per Second*) angegeben. Um in Bezug auf das Kriterium der Rechenleistung eine Bewertung vornehmen zu können, werden veröffentlichte Vergleiche herangezogen.
Ein solcher Vergleich findet sich in [NVIDIA, 2019a] in Form einer Tabelle (siehe Anhang A.1), in welcher die Vorhersage Performance (*inference performance*) verschiedener Single-Board-Computer anhand der *frames per second* (fps) verglichen wird. Ein direkter

Vergleich zwischen dem Jetson Nano und dem Coral Dev Board wird darin für die Modelle MobileNet-v2 (300 x 300), SSD Mobilenet-V2 (300 x 300) und Inception V4 (299 x 299) aufgeführt. Die größte Relevanz haben dabei die erzielten Ergebnisse bei der Verwendung des SSD Mobilenet-V2 Modells. Bei diesem Modell handelt es sich um eines, welches sich speziell für die Aufgabe der Objekterkennung auf Embedded Geräten und Single-Board-Computern eignet. Bezüglich dieses Modells erreicht das Coral Dev Board mit 48 fps ein besseres Ergebnis als der Jetson Nano, der ein Resultat von 39 fps erreicht. Hinsichtlich des MobileNet-v2 Modells erreicht das Coral Dev Board 130 fps und schneidet damit etwa doppelt so gut ab wie der Jetson Nano (64 fps). Lediglich beim Modell Inception V4 liegt der Jetson Nano mit 11 fps, im Vergleich zu den erreichten 9 fps des Coral Dev Boards, leicht vorne. Da die beiden zuletzt genannten Modelle laut Tabelle für die Klassifizierung vorgesehen sind, werden diese bei der Bewertung geringfügig niedriger gewichtet. Zusammengefasst zeigt sich hinsichtlich dieses Vergleichs, dass das Coral Dev Board eine deutlich bessere Leistung als der Jetson Nano aufweist. Speziell das bessere Resultat bei der Verwendung des SSD Mobilenet-V2 Modells für die Objekterkennung spricht für das Coral Dev Board und führt hier zu der besseren Bewertung.
Ein anderer Vergleich (siehe Weiss [2019]) bezieht sich auf den Jetson Nano und den Coral USB-Beschleuniger[2] in Verbindung mit einem Desktop-Computer mit Intel i7 CPU. Das nachfolgend beschriebe Resultat nimmt aus diesem Grund keinen direkten Einfluss auf das getroffene Vergleichsergebnis zwischen dem Jetson Nano und dem Coral Dev Board, soll aber dazu dienen das bessere Abschneiden des Coral Dev Boards durch eine zusätzliche Anmerkung zu stützen. Hinsichtlich der Geschwindigkeit erzielt die Kombination aus Desktop Computer und dem Coral USB-Beschleuniger bei der Verwendung des SSD Mobilenet-V2 Modells mit 137 fps ein erheblich besseres Resultat als der Jetson Nano, welcher 20 fps erreicht. Da dieses Ergebnis sehr weit von den zuvor aufgeführten Vergleichsergebnissen abweicht, sollen diese Werte nur hintergründig betrachtet werden. Für diese Arbeit ist jedoch die Erkenntnis relevant, dass die speziell für das Inferencing ausgelegte *Edge TPU*, welche auch im Coral Dev Board eingesetzt wird, zu besseren Ergebnissen in Bezug auf die Leistungsfähigkeit beiträgt. Bei der Edge TPU handelt es sich um einen anwendungsspezifischen integrierten Schaltkreis bzw. Mikrochip (kurz ASIC, für *Application-Specific Integrated Circuit*).

5.1.2. Auswahl der Kamera

Im Rahmen der vorliegenden Arbeit ist für die Auswahl der Kamera kein umfangreiches Auswahl- und Bewertungsverfahren vorgesehen. Die wesentliche Anforderung ist die Kompatibilität zum ausgewählten Development-Board. Die entsprechende Auswahl der Kamera mitsamt den wesentlichen Entscheidungskriterien ist im Nachfolgenden dargelegt.

[2] https://coral.ai/products/accelerator

Die Webseite https://elinux.org/Jetson_Nano liefert im Abschnitt *Cameras* eine Übersicht erhältlicher USB-Kameras und Kameramodule, die nach dem Plug-and-Play Prinzip in Verbindung mit dem Jetson Nano einsetzbar sind. Diese Übersicht enthält unter anderem das Raspberry Pi Kameramodul V2, welches für das im Rahmen dieser Arbeit konzipierte Embedded Vision System ausgewählt wird. Da das Kameramodul überdies in der Produktvorstellung des Jetson Nanos (vgl. [Franklin, 2019]) genannt wird, kann es als eine geeignete Kameralösung betrachtet werden.
Das Kameramodul verwendet die für mobile Anwendungen und Embedded Systeme verbreitete MIPI CSI-2 Schnittstelle und wird mittels Flachbandkabel an die entsprechende Schnittstelle des Jetson Nanos angeschlossen. Da es sich um ein Kameramodul handelt, trägt es hinsichtlich des Formfaktors und Gewichts dazu bei, eine kompakte Bauform und ein geringes Gewicht des Gesamtsystems zu erhalten. Das typischerweise verhältnismäßig kurze Flachbandkabel stellt dementsprechend keinen Nachteil dar. Mit einer Bildauflösung von 8 Megapixeln und einer Sensor-Auflösung von 3280 x 2464 Pixeln werden hinsichtlich der Auflösung keine Einschränkungen erwartet, die sich nachteilig auf den Anwendungszweck auswirken. Ein positiver Aspekt, den das Raspberry Pi Kameramodul neben den technischen Eigenschaften aufweist, ist darüber hinaus die preiswerte Verfügbarkeit (vgl. [RaspberryPi, o. J.]).

5.2. Auswahl des Datensatzes zur Personenerkennung

Für die Auswahl des Datensatzes werden die Anforderungen aus dem Unterkapitel 4.3 herangezogen. Darauf basierend wird nachfolgend das Auswahlverfahren beschrieben.
Die Anzahl verfügbarer Datensätze ist sehr umfangreich, da neben einschlägig bekannten und großen Datensätzen, wie beispielsweise dem *Microsoft COCO* (*Common Objects in Context*), *PASCAL VOC* (*PASCAL Visual Object Classes*), *KITTI* oder *ImageNet* Datensatz diverse Anwendungs-spezifische Datensätze auf Webseiten wie *Kaggle*[3] zur Verfügung gestellt werden. Dementsprechend groß ist auch die Anzahl der Datensätze, die auf die Objekterkennung generell, als auch auf das Erkennen von Personen im Speziellen abzielen. Um das hier beschriebene Auswahlverfahren in einem engen Rahmen durchzuführen, begrenzt sich die Auswahl auf die folgenden Datensätze[4]:

- PETS09-S2L1 (MOT15) [MOT, o. J.a]
- MOT17-04 (MOT17DET) [MOT, o. J.b]
- MOT20-03 (MOT20) [MOT, o. J.c], [MOT, o. J.d]

[3] https://www.kaggle.com/datasets
[4] In Anhang **??** findet sich eine Übersicht der im Rahmen einer Vorauswahl betrachteten Datensätze.

- WiderPerson: A Diverse Dataset for Dense Pedestrian Detection in the Wild [Zhang et al., 2019]
- Crowds, Jogging, Subway, Human3 (Visual Tracker Benchmark) [VisualTrackerBenchmark, o. J.]

Bei den drei erstgenannten Datensätzen handelt es sich jeweils um einen von mehreren Trainingsdatensätzen, die im Rahmen der *Multiple Object Tracking Challenge* (MOT Challenge) des in Klammern genannten Jahres für die Teilnahme an dem Wettbewerb zur Verfügung standen. *WiderPerson* ist ein Personen-Datensatz, der eine große Variation an dargestellten Szenarien, speziell außerhalb des Straßenverkehrs, liefert. Der letztgenannte Datensatz ist eine Zusammenstellung einzelner Datensätze bzw. Sequenzen mit den Bezeichnungen „Crowds“, „Jogging“, „Subway“ und „Human3“, die auf der *Visual Tracker Benchmark* Website verfügbar sind und Personen in unterschiedlichen Umfeldern darstellen. Die fünf Datensätze ähneln sich in den wesentlichen Eigenschaften, unterscheiden sich stellenweise aber auch deutlich voneinander. Dadurch soll vermieden werden, bereits vor dem Vergleichs- und Bewertungsverfahren eine zu genaue Lösung vorzugeben.

Tabelle 5.3.: Vergabeschema der Bewertungspunkte für Datensätze

Bewertung	**Beschreibung**
++	Zwei Bewertungspunkte werden addiert; die Anforderung wird verhältnismäßig gut erfüllt;
+	Ein Bewertungspunkt wird addiert; die Anforderung wird zufriedenstellend erfüllt;
o	Keine Vergabe eines Bewertungspunktes; die Anforderung wird verhältnismäßig schlecht erfüllt;

Alle aufgeführten Datensätze sind für die Objekterkennung bzw. -verfolgung vorgesehen und stellen Personen in den Fokus. Die Anforderungen A.4.3-01 und A.4.3-02 gelten somit als erfüllt. Für das weitere Auswahlverfahren werden die noch offenen Anforderungen A.4.3-03 bis A.4.3-09 herangezogen. Dabei werden zunächst die für eine Bewertung relevanten Angaben und Werte der einzelnen Datensätze in den Tabellen 5.4 und 5.5 aufgeführt. Anschließend erfolgt innerhalb der Bewertungstabelle 5.6 unter Beachtung der Erläuterung in Tabelle 5.3 und den Gewichtungen aus Tabelle 4.3 eine entsprechende Punktevergabe. Die Gewichtungen sind innerhalb der Bewertungstabelle in Klammern hinter den korrespondierenden Anforderungs-IDs aufgeführt.

Entsprechend dieses Bewertungsverfahrens erzielt der Datensatz PETS09-S2L1 mit leichten Vorteilen gegenüber dem MOT17-04 Datensatz das beste Gesamtergebnis. Der PETS09-S2L1 Datensatz schneidet bei der Anforderung A.4.3-03 besser ab, da in diesem keine Bilder mit zu hohen Personendichten und den dementsprechend vermehrten Verdeckungen

Tabelle 5.4.: Beschreibung der Datensatzeigenschaften und Werte der MOT Datensätze in Bezug auf die gestellten Anforderungen

Datensatz	**Anforderungs-ID und Beschreibung**
PETS09-S2L1 (MOT15)	A.4.3-03: Selten verdeckte Personen A.4.3-04: Ground Truth A.4.3-05: Darstellungswinkel der Personen ca. 30° A.4.3-06: Treffende Personengröße A.4.3-07: Max. Größendifferenz zwischen kleinster und größter dargestellter Person ca. 60 %; Darstellung der Personen aus allen Richtungen A.4.3-08: Straßeneinmündung mit sporadischem Verkehraufkommen und Rasenflächen bei Tageslicht A.4.3-09: 94 MB; 795 (Trainings-) Bilder; (alle MOT15 Trainingsdatensätze: 1,3 GB)
MOT17-04 (MOT17DET)	A.4.3-03: Regelmäßig verdeckte Personen A.4.3-04: Ground Truth A.4.3-05: Darstellungswinkel der Personen ca. 40° A.4.3-06: Treffende Personengröße A.4.3-07: Max. Größendifferenz zwischen kleinster und größter dargestellter Person ca. 50 %; Darstellung der Personen größtenteils von vorne und hinten, gelegentlich von der Seite A.4.3-08: Fußgängerpassage bei künstlicher Beleuchtung (hell) A.4.3-09: 236 MB; 1050 (Trainigs-) Bilder; (alle MOT17DET Trainingsdatensätze: 1,9 GB)
MOT20-03 (MOT20)	A.4.3-03: Häufig verdeckte Personen A.4.3-04: Ground Truth A.4.3-05: Darstellungswinkel der Personen ca. 65° A.4.3-06: Tendenziell zu geringe Personengröße A.4.3-07: Max. Größendifferenz zwischen kleinster und größter dargestellter Person ca. 30 %; Darstellung der Personen größtenteils von hinten A.4.3-08: Verkehrsberuhigter Bereich vor einem Stadion bei künstlicher Beleuchtung (Dämmerung) A.4.3-09: 552 MB; 2.405 (Trainings-)Bilder; (alle MOT20 Trainingsdatensätze: 4,7 GB)

Tabelle 5.5.: Beschreibung der Datensatzeigenschaften und Werte des WiderPerson und VTB Datensatzes in Bezug auf die gestellten Anforderungen

Datensatz	**Anforderungs-ID und Beschreibung**
WiderPerson: A Diverse Dataset for Dense Pedestrian Detection in the Wild	A.4.3-03: Regelmäßig verdeckte Personen A.4.3-04: Ground Truth A.4.3-05: Darstellungswinkel der Personen sehr variierend; i. d. R. Normalansicht A.4.3-06: Stark variierende Personengrößen A.4.3-07: Sehr stark variierende Personengrößen zwischen einzelnen Bildern; Darstellung der Personen aus allen Richtungen A.4.3-08: Stark variierende Aufnahmeumfelder A.4.3-09: / MB; 13.382 Bilder
Crowds, Jogging, Subway, Human3 (Visual Tracker Benchmark, kurz VTB)	A.4.3-03: Selten verdeckte Personen A.4.3-04: Ground Truth A.4.3-05: Darstellungswinkel der Personen variierend von ca. 30° bis 80° A.4.3-06: Tendenziell passende Personengröße, aber regelmäßig abweichend A.4.3-07: Deutlich variierende Personengrößen; Darstellung der Personen aus allen Richtungen A.4.3-08: Gehweg, Rasenflächen, verkehrsberuhigter Bahnhofsbereich, Fußgängerüberweg (alles bei Tageslicht) A.4.3-09: 173,6 MB; 2.528 Bilder

von Personen auftreten. Die Bilder des MOT17-04 Datensatzes weisen hingegen mit ca. 40° einen Aufnahmewinkel auf, der am geringsten von dem als Richtwert gesetzten 45°-Winkel (A.4.3-05) abweicht. Da für diesen Richtwert jedoch ein verhältnismäßig großer Toleranzbereich von $\pm 15°$ vorgesehen ist, erhält der PETS09-S2L1 Datensatz trotz der größeren Winkelabweichung eine gleiche Bewertung. Bezüglich der Anforderung A.4.3-08 schneidet der MOT17-04 besser als der PETS09-S2L1 Datensatz ab, da durch die künstliche Beleuchtung unterschiedliche Lichtverhältnisse in einzelnen Bildbereichen entstehen.
Der zusammengestellte Visual Tracker Benchmark Datensatz (Crowds, Jogging, Subway, Human3) erhält ebenfalls ein gutes Ergebnis, hat gegenüber den zuvor genannten Datensätzen jedoch entscheidende Nachteile. Insbesondere wegen zu stark variierender Aufnahmewinkel (A.4.3-05), Personengrößen (A.4.3-06) und den Größen der Aufnahmebereiche (A.4.3-07).
Die wesentlichen Nachteile des MOT20-03 Datensatzes liegen in der zu häufig auftretenden hohen Personendichte mit vielen verdeckten Personen (A.4.3-03), der tendenziell zu kleinen Darstellung von Personen, die zumeist lediglich von hinten abgebildet sind (A.4.3-07), als

Tabelle 5.6.: Bewertung der Datensätze

Anforderungs-ID	**MOT15**	**MOT17DET**	**MOT20**	**WiderPerson**	**VTB**
A.4.3-03 (1) Wenig Verdeckung	++	o	o	o	++
A.4.3-04 (1,5) Gelabelt mit Bounding Box	++	++	++	++	++
A.4.3-05 (1) Darstellungswinkel	+	+	+	o	o
A.4.3-06 (1) Personengröße	++	++	+	o	+
A.4.3-07 (1,5) Größenvariation	++	++	o	o	+
A.4.3-08 (1) Varianz Umgebung	+	++	o	o	+
A.4.3-09 (1,5) Speicherbedarf	+	+	+	o	+
Ergebnis	**13,5**	**12,5**	**6,5**	**3**	**10**

auch in den dunklen Lichtverhältnissen (A.4.3-08).
Der WiderPerson Datensatz erhält insgesamt die wenigsten Punkte, da dieser grundsätzlich dafür ausgelegt ist, eine große Variation hinsichtlich des Aufnahmewinkels sowie der Größe und Orientierung von Personen zu bieten. Überdies liegt ein Fokus dieses Datensatz auf der Darstellung von Personengruppen und Menschenmengen mit einer hohen Dichte und vielen Verdeckungen. Die im Vergleich zu den anderen Datensätzen verhältnismäßig hohe Anzahl der verfügbaren Bilder lässt zudem auf einen entsprechend hohen Speicherbedarf (A.4.3-09) schließen. Aus diesem Grund erhält der WiderPerson Datensatz auch bei dieser Anforderung eine geringere Bewertung als die gleich bewerteten übrigen Datensätze.

Basierend auf dem vorliegenden Vergleichsergebnis wird der Datensatz PETS09-S2L1 ausgewählt. Damit der Datensatz in einem begrenzten Maß speziell an den Einsatzort angepasst ist, wird dieser um dort aufgenommene Bilder ergänzt. Das Ziel dieser Maßnahme ist die Verbesserung der Personenerkennung innerhalb des Einsatzortes. Der Anteil dieser Bilder wird auf maximal 25 % der Gesamtgröße des Datensatzes begrenzt, da das mit diesem Datensatz trainierte Modell weiterhin generell und ortsunabhängig in Aufnahmeumfeldern entsprechend A.4.3-08 einsetzbar bleiben soll. Ferner ist der Anteil dieser Bilder begrenzt, da diese manuell mit Bounding Boxes und Labels versehen werden müssen. Um dabei der Anforderung A.4.3-04 nicht zu widersprechen, wird ein begrenzter Aufwand eingehalten.
Die eigens generierten Bilddaten stammen aus verschiedenen aufgezeichneten Videose-

Abbildung 5.1.: Bilder des verwendeten Datensatzes. Oben: Bilder aus dem PETS09-S2L1 Datensatz. Unten: Mit dem Embedded Vision System aufgezeichnete Bilder des eigenen Datensatzes

quenzen. Diese Videosequenzen sind mittels eines Open Source Tools, hier der *Free Video to JPG Converter v. 5.0.101 build 201*, in eine bestimmte Anzahl einzelner Frames respektive Bilder zerlegt. Im Rahmen dieser Arbeit handelt es sich um 235 generierte Bilder. Der entsprechende Speicherbedarf liegt bei 31 MB. Die Bilder zeigen Personen aus einem Aufnahmewinkel von ca. 50° und in unterschiedlichen Größen und Orientierungen. Durch einen Versatz des Aufnahmegerätes zwischen einzelnen Videosequenzen weisen die Bilder zudem eine gewisse Varianz hinsichtlich der Perspektive auf. Da die am Einsatzort aufgenommenen Bilder im Vergleich zu den Bildern des PETS09-S2L1 Datensatzes verhältnismäßig dunkel sind, verstärkt sich in Bezug auf den gesamten Datensatz hinsichtlich Anforderungen A.4.3-08 vorteilhafterweise der Wechsel der Lichtverhältnisse. In Abbildung 5.1 sind exemplarisch einige Bilder des Gesamtdatensatzes dargestellt.
Für das Kennzeichnen der in den Bildern des Aufnahmebereichs dargestellten Personen wird das grafische Tool *LabelImg*[5] eingesetzt. Dieses ist einfach und pragmatisch bedienbar und ermöglicht auf diese Weise ein relativ schnelles Generieren von Bounding Boxes und Labels.
Da es eine ausreichend große Auswahl an frei erhältlicher und funktional gleichermaßen geeigneter Software-Tools für das Zerlegen von Videosequenzen in einzelne Frames sowie für das Erstellen von Bounding Boxes und Labels gibt, beruht die Auswahl dieser Tools hier auf persönlichen Präferenzen.

Tabelle 5.7.: Bewertung der Datensätze

Anforderungs-ID	**YOLOv3-Tiny**	**SSD-MobileNet v2**
A.4.4-01 (1) Einstufiger Detektor	+	+
A.4.4-02 (1) Geringe Anzahl der Layer (Richtwert: 20)	++	+
A.4.4-03 (1) Anzahl der Modellparameterunter 10 Mio.	+	++
A.4.4-04 (1,5) Modell für die Objekterkennung	+	+
A.4.4-05 (2) Modell für mobile / embedded Geräte	+	+
A.4.4-06 (1) Simultane Personenerkennung	+	+
A.4.4-07 (1) Verarbeitungs- geschwindigkeit (3 fps)	+	++
A.4.4-08 (1,5) Support	++	+
Ergebnis	**12,5**	**11,5**

5.3. Auswahl des Modells und weiterer Softwareanteile

Hinsichtlich der in Tabelle 5.7 aufgeführten Bewertungen gilt auch hier das in Tabelle 5.3 erklärte Bewertungsverfahren.

Innerhalb dieses Unterkapitels wird unter Berücksichtigung der in Kapitel 4.4 genannten Anforderungen das Auswahlverfahren des Modells für das Embedded Vision System beschrieben. Dazu erfolgt ein Vergleich mitsamt einer Bewertung. Das Modell mit der höchsten Punktzahl wird ausgewählt.

Da die finale Auswahl des Modells im Rahmen dieses konzeptionellen Teils auf zwei favorisierte Modelle beschränkt werden soll, erfolgt zunächst eine Vorbetrachtung verschiedener Modelle und Architekturen. Dabei werden mit Blick auf die gestellten Anforderungen jeweils unterschiedliche Aspekte betrachtet, um nicht infrage kommende Lösungen auszuschließen. Betrachtet man zunächst die Anforderung nach Modellen, die speziell für die Objekterkennung (A.4.4-04) geeignet sind, handelt es sich bei R-CNN, SSD und YOLO um etablierte

[5] https://pypi.org/project/labelImg/

Lösungen. Als zweistufiger Detektor ist R-CNN hinsichtlich der Anforderung A.4.4-01 jedoch nicht geeignet. Dagegen stellen SSD und YOLO als einstufige Detektoren geeignete Lösungen dar. Eine Berücksichtigung von Modellen, die auf bekannten Architekturen wie beispielsweise LeNet-5 oder AlexNet beruhen, können ebenfalls von weiteren Betrachtungen ausgeschlossen werden, da diese die Anforderung A.4.4-02 (geringe Anzahl an Layern, Richtwert von 20) nicht erfüllen. Weitere mögliche Modelle sind *InceptionV4* sowie *Xception*. Diese weisen mit jeweils mehr als 20 Millionen Parametern (A.4.4-03) jedoch eine zu hohe Anzahl auf und sind aus diesem Grund ebenfalls nicht geeignet. Um auf Modelle einzugehen, die speziell für mobile und embedded Geräte geeignet sind (A.4.4-05), wird ein Vergleich[6] herangezogen, in dem unterschiedlicher Modelle in Verbindung mit dem Jetson Nano getestet werden. Darin werden unter anderem die Modelle SSD-MobileNet v2 und YOLOv3-Tiny aufgeführt. Da diese Modelle rückblickend betrachtet alle bereits aufgeführten Anforderungen erfüllen, werden diese als favorisierte Lösungen ausgewählt und in Tabelle 5.7 bewertet. Entsprechend den Ergebnissen aus Tabelle 5.7 wird YOLOv3-Tiny mit dem Darknet-19 als Modell für das Embedded Vision System ausgewählt. Ausschlaggebend hierfür ist die geringere Anzahl der Layer. Diesbezüglich handelt es sich um 19 Layer beim YOLOv3-Tiny Modell im Vergleich zu 20 Layern beim SSD-Modell. Für YOLOv3-Tiny spricht zudem der größere Support, der im Zusammenhang mit diesem Modell geboten wird (A.4.4-08). Da SSD MobileNet v2 hinsichtlich der Anzahl der Parameter (A.4.4-03) und in Bezug auf die Geschwindigkeit (A.4.4-07) besser abschließt, sind beide Lösungen annähernd gleichwertig.

Neben der Beschreibung des Auswahlverfahrens des Modells beinhaltet dieses Unterkapitel eine kurze Beschreibung der Auswahl weiterer Softwareanteile, die in dieser Arbeit eingesetzt werden. Hierbei handelt es sich zum einen um die Programmiersprache und zum anderen um die Auswahl des Frameworks.

Als Programmiersprache wird Python (Version 3.6.9) gewählt. Auf Grund der steigenden Beliebtheit dieser Programmiersprachen, ist auch die Verfügbarkeit nutzbarer Bibliotheken in den Bereichen Data Science und Machine Learning angestiegen. Exemplarisch seien hier *scikit-learn* oder Bibliotheken wie *NumPy*, *SciPy*, *matplotlib* oder *pandas* genannt, die sich ebenso für das wissenschaftliche Arbeiten eignen. Überdies bietet Python den Vorteil einer einfachen Syntax, die zur Vermeidung von Syntaxfehlern beiträgt und sich so der Fokus des Programmierens auf die jeweilige Problemstellung richten kann.

In Bezug auf die Auswahl des Frameworks werden TensorFlow in Verbindung mit Keras und PyTorch in Betracht gezogen. Der Hintergrund dieser Auswahl liegt in der Popularität beider Frameworks. Während es sich bei TensorFlow (inklusive Keras) um das populärste Framework handelt, ist PyTorch ein Framework, welches stetig an Popularität gewinnt (vgl. [Sinhal and Sachan, 2018]). Beide Frameworks können demnach als besonders stark konkurrierende Lösungen betrachtet werden.

[6] https://developer.nvidia.com/blog/wp-content/uploads/2019/03/imageLikeEmbed.png

Tabelle 5.8.: Frameworks und deren wesentliche Eigenschaften

Bewertung	Beschreibung
TensorFlow inkl. Keras	TensorFlow - Umfangreiches Framework - Große Anzahl an vortrainierten Modellen und Codebeispielen - Umfangreicher Support durch Tutorials, Literatur und Dokumentation - Große Präsenz auf Webseiten wie GitHub oder Stack Overflow - Geeignet für einfache Modelle und auch große CNNs Keras - Intuitive und einfach anwendbare Schnittstelle für das Kodieren - Gut geeignet für das Rapid Prototyping - Reduziert die Anzahl erforderlicher Codezeilen
PyTorch	- Flexible und einfache Anwendbarkeit - Großer Python Support - Populär in Forschung und akademischen Kreisen - Gut geeignet für kleine / akademische Projekte - Gut geeignet für das Rapid Prototyping - Verbesserte Eignung für die Produktion durch die Integration des Frameworks *Caffe2* in 2018

In Tabelle 5.8 finden sich einige wesentliche Eigenschaften dieser Frameworks (vgl. [Koul et al., 2019], Abschnitt *Frameworks*).

Beide Frameworks können in Python implementiert werden, sind als Open Source verfügbar und überdies für den Einsatz in Verbindung mit GPUs optimiert. Dadurch kann mit beiden Frameworks die parallele Datenverarbeitung von GPUs genutzt werden, um Trainingsprozesse und das Treffen von Vorhersagen zu beschleunigen. Darüber hinaus ist zu beachten, dass es sich um Frameworks handelt, die von dem in Unterkapitel 5.1 ausgewählten Development-Board unterstützt werden.
Zur Entscheidungsfindung werden zum einen ein umfangreicher Support mittels Tutorials, Fachliteratur, Dokumentation und einschlägiger Webseiten wie GitHub und Stack Overflow und zum anderen eine anwenderfreundliche Nutzbarkeit die höchste Priorität beigemessen. PyTorch wird hinsichtlich der Anwenderfreundlichkeit tendenziell als vorteilhafter betrachtet. Dieser Vorteil wird durch Keras jedoch relativiert. In Bezug auf den vorhandenen Support liegt der Vorteil deutlich bei TensorFlow und Keras. Basierend auf dem verhältnismäßig geringen Unterschied bei der Anwenderfreundlichkeit und dem deutlicheren Vorteil TensorFlows hinsichtlich des Supports, wird TensorFlow inklusive Keras als Framework verwendet.

5.4. Methodik der Laufweganalyse

In diesem Unterkapitel wird die Methodik oder auch die Herangehensweise beschrieben, mit der die Laufweganalyse durchgeführt wird. Diese kann in drei Phasen unterteilt werden.

1. Definition / Festlegung der Regions of Interest
2. Datenakquisition
3. Analyse und Visualisierung

In der ersten Phase werden durch eine Betrachtung des gesamten Aufnahmebereichs die für die Laufweganalyse relevanten Teilbereiche, sinngemäß handelt es sich hierbei um die Regions of Interest (ROIs), identifiziert. Anschließend liegt der Fokus darauf, sinnvolle Teilbereiche zu bestimmen, die zu einer Laufweganalyse beitragen, welche für die Einsatzplanung von fahrerlosen Transportsystem einen Mehrwert liefert. Dieser wird erreicht, wenn Laufweganalyse und Einsatzplanung zu weniger Personenbehinderungen führen. Entscheidend ist es demnach, Teilbereiche zu bestimmen, mit denen die voraussichtlich gängigen Bereiche für Laufwege abgedeckt sind. Ein engmaschiges Raster an Teilbereichen zu definieren ist nicht erforderlich. Die Bestimmung der Teilbereiche wird in Kapitel 6.3 beschrieben.
Während der zweiten Phase wird die Personenerkennung ausgeführt. Dabei werden die Positionsdaten der erkannten Personen respektive der entsprechenden Bounding Boxes kontinuierlich erfasst und lokal auf dem Development-Board gespeichert. Das Abspeichern erfolgt lokal, da das Embedded Vision System ein möglichst autarkes Stand-alone-System darstellen soll. Überdies werden zusätzliche Latenzen, die bei einem Datentransfer auftreten vermieden.
Die Verarbeitung samt Visualisierung der akquirierten Daten wird in der dritten Phase umgesetzt. Anhand der gesicherten Positionsdaten erfolgt die Zuordnung der erkannten Personen zu den entsprechenden Teilbereichen. Dabei wird festgestellt, wie viele Personen sich während einzelner Zeitintervalle in einem Teilbereich befinden, wie stark dieser also von Personen frequentiert bzw. genutzt wird. Personen außerhalb der Teilbereiche müssen nicht berücksichtigt werden. Zur Veranschaulichung wird die Nutzungshäufigkeit der einzelnen Teilbereiche in Abhängigkeit von der Zeit in einem gemeinsamen Diagramm dargestellt. Das Diagramm wird zusammen mit den akquirierten Daten als Grundlage für die Interpretation und Auswertung der genutzten Laufwege verwendet. Die entsprechende Ausarbeitung erfolgt im Rahmen der Evaluation in Kapitel 7.

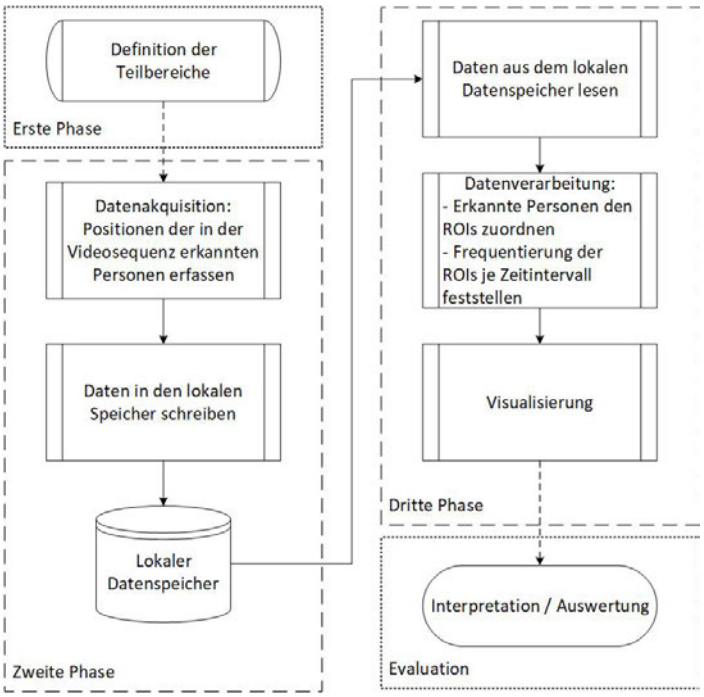


Abbildung 5.2.: Ablaufdiagramm für die zweite und dritte Phase der Analyse von Laufwegen

6. Systemdesign und Laufweganalyse

Im Rahmen dieses Kapitels werden die Realisierung des Embedded Vision Systems, die wesentlichen Aspekte des Trainings des Modells sowie die Laufweganalyse beschrieben.
Das Unterkapitel zur Realisierung des Embedded Vision System beruht im Wesentlichen auf den erarbeiteten Inhalten des konzeptionellen Teils zur Auswahl des Development-Boards (Unterkapitel 5.1) sowie des Modells (Unterkapitel 5.3). Im anschließenden Unterkapitel zum Training des Modells, mit dem in Unterkapitel 5.2 ausgewählten Datensatz, werden die Vorverarbeitung der Trainings- und Testdaten sowie die vorgenommene Parametrisierung und das Resultat des Trainingsprozesses erläutert. Innerhalb des letzten Unterkapitels werden, entsprechend der Vorarbeit in Unterkapitel 5.4, die Umsetzung und der Prozess der Laufweganalyse geschildert und anhand eines beispielhaften Szenarios verdeutlicht.

6.1. Realisierung des Embedded Vision Systems

Die in diesem Unterkapitel dargestellte Realisierung des Embedded Vision Systems gliedert sich in zwei Abschnitte. Im ersten Abschnitt werden die einzelnen Hardwarekomponenten beschrieben, aus denen sich das Gesamtsystem zusammensetzt. Der zweite Abschnitt enthält eine Beschreibung der Softwareanteile des Systems.

6.1.1. Hardwareanteile des Embedded Vision Systems

Das realisierte Embedded Vision System besteht seitens der Hardware aus dem im konzeptionellen Teil ausgewählten Jetson Nano sowie dem Raspberry Pi Kameramodul V2. Der Jetson Nano ist mit einer 64 GB MicroSD-Speicherkarte ausgestattet. Laut Hersteller wird eine Speicherkarte von mindestens 32 GB empfohlen (vgl. [NVIDIA, 2019b]). Im Kontext dieser Arbeit wird die doppelte Speicherkapazität gewählt, damit während der Entwicklungs- und Realisierungsphase ausreichende Speicherreserven für den Test und die Erprobung möglicher Lösungswege zur Verfügung stehen. Als Spannungsquelle dient ein handelsübliches 5 VDC Steckernetzteil. Die wesentlichen Bestandteile des Jetson Nanos selbst sind ein Trägerboard, das *Jetson Nano Modul* mit Kühlkörper sowie verschiedene Hardwareschnittstellen für den Anschluss von Peripherie-Geräten. Das Development-Kit besteht aus dem

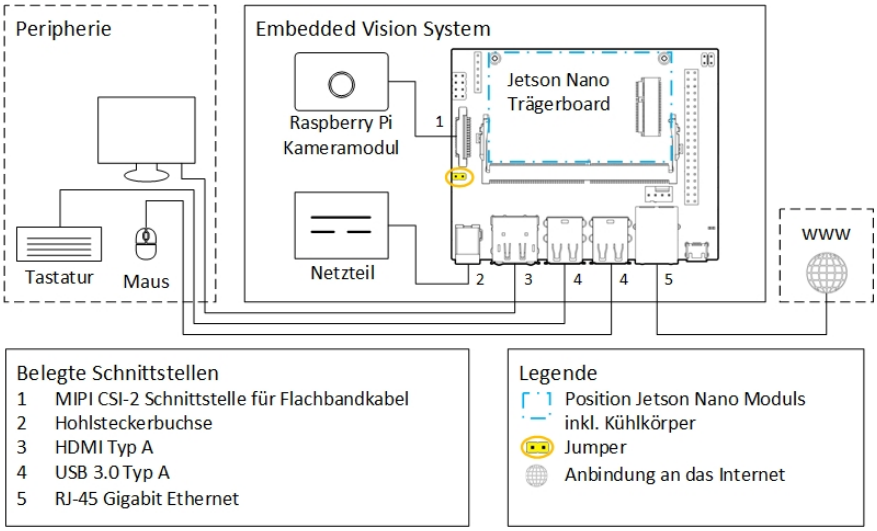


Abbildung 6.1.: Verkabelung des Embedded Vision Systems ohne Darstellung des Jetson Nano Moduls und des Kühlkörpers

Trägerboard mitsamt den Hardwareschnittstellen sowie dem Jetson Nano Modul inklusive Kühlkörper als fertigen Zusammenbau. Eine Übersicht der technischen Daten findet sich in Anhang A.6.

Im Rahmen dieser Arbeit werden die MIPI CSI-2 Schnittstelle für den Anschluss des Raspberry Pi Kameramoduls mittels Flachbandkabel, zwei USB 3.0 Typ A Anschlüsse für Tastatur und Maus, der HDMI 2.0 Typ A Anschluss für einen Monitor und die Anschlussbuchse für Hohlstecker für die Spannungsversorgung verwendet. Die während der Realisierungsphase erforderliche Anbindung an das Internet erfolgt über die RJ-45 Gigabit Ethernet Schnittstelle. Diese Verkabelung ist in Abbildung 6.1 dargestellt.

In der Grundkonfiguration erfolgt die Spannungsversorgung des Nanos über einen Micro-USB 2.0 Anschluss. Über diesen kann der Nano mit einem 5 VDC, 2 A Micro-USB Netzteil betrieben werden. In der vorliegenden Arbeit erfolgt die Spannungsversorgung des Jetson Nanos bzw. des Embedded Vision Systems über die Anschlussbuchse für Hohlstecker. Dazu sind zwei Pins des Nanos mit einem *Jumper* verbunden, wodurch die Hohlsteckerbuchse als Anschluss für die Versorgungsspannung aktiviert wird. Auf diese Weise kann der Jetson Nano mit einem 5 VDC, 4 A Steckernetzteil betrieben werden. Der Micro-USB 2.0 Anschluss ist in dieser Konfiguration nicht mehr für die Spannungsversorgung einsetzbar, kann aber als gewöhnliche Peripherie-Schnittstelle genutzt werden.

Die höhere verfügbare Stromstärke dient dazu, Spannungsabfällen vorzubeugen, die beispielsweise aus dem Betrieb angeschlossener Peripherie resultieren können. Auch wenn dies im Rahmen der vorliegenden Arbeit prinzipiell nicht zu erwarten ist, erfolgt die Spannungsversorgung über die Hohlsteckerbuchse, da durch diese Maßnahme mit einem geringen Aufwand die Robustheit des Systems gegenüber Fehlern erhöht wird.

Abbildung 6.2.: Beschriftung des Bildes

6.1.2. Softwareanteile des Embedded Vision Systems

Basierend auf den im konzeptionellen Teil durchgeführten Auswahlverfahren, werden entsprechend des Unterkapitels 5.3 das Modell YOLOv3-Tiny, das Framework TensorFlow inklusive Keras und die Programmiersprache Python für das Embedded Vision System eingesetzt.
Das eingesetzte YOLOv3-Tiny Modell nutzt als Backbone-Netzwerk eine aus sieben Convolutional-Layern des Typs *Conv2D* und sechs Max-Pooling-Layer des Typs *MaxPooling2D* bestehende Architektur (siehe Tabelle 6.1). Nach einer Convolution werden jeweils eine *Batch Normalization* sowie die Aktivierungsfunktion ausgeführt. Die Batch Normalization dient dem Zweck, den Lernprozess des Modells zu vereinfachen bzw. zu beschleunigen und darüber hinaus die Verallgemeinerungsfähigkeit zu verbessern. Bei dem eingesetzten Optimierer handelt es sich um den *Adam* Algorithmus. In der vorliegenden Arbeit ist diese Architektur in Anlehnung an das *Darknet-19* Backbone-Netzwerk, welches mit YOLOv2 eingeführt wurde (vgl. [Redmon and Farhadi, 2016]), in der Funktion mit der Bezeichnung darknet19_tiny implementiert.
Die gewählte Größe für den Input-Layers beträgt 416 x 416 x 3. Aus dem nachfolgenden Backbone-Netzwerk werden zwei Ausgaben herausgeführt. Die erste Ausgabe erfolgt nach dem fünften Convolutional-Layer und die zweite Ausgabe nach dem siebten Convolutional-Layer am Ende des Backbone-Netzwerks. Außerhalb des Backbone-Netz werden diese Ausgaben über die TensorFlow Operation *tf.concat()* verkettet. Die nach dem fünften Convolutional-Layer herausgeführte Ausgabe wird direkt an die Verkettung übergeben, wäh-

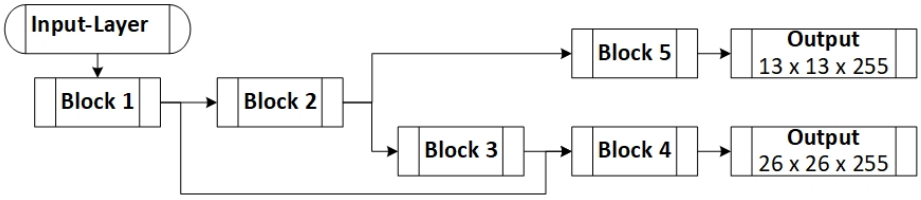
Tabelle 6.1.: YOLOv3-Tiny Backbone-Netzwerk mit sieben Convolutional-Layern und sechs Max-Pooling-Layern

Layer Type	Filter	Size/Stride	Output
Convolutional	16	3 x 3 / 2	416 x 416 x 16
Max-Pooling		2 x 2 / 2	208 x 208 x 16
Convolutional	32	3 x 3 / 2	208 x 208 x 32
Max-Pooling		2 x 2 / 2	104 x 104 x 32
Convolutional	64	3 x 3 / 2	104 x 104 x 64
Max-Pooling		2 x 2 / 2	52 x 52 x 64
Convolutional	128	3 x 3 / 2	52 x 52 x 128
Max-Pooling		2 x 2 / 2	26 x 26 x 128
Convolutional	256	3 x 3 / 2	26 x 26 x 256
Max-Pooling		2 x 2 / 2	13 x 13 x 256
Convolutional	512	3 x 3 / 2	13 x 13 x 512
Max-Pooling		2 x 2 / 1	13 x 13 x 512
Convolutional	1024	3 x 3 / 2	13 x 13 x 1024

rend die Ausgabe am Ende des Backbone-Netzwerks zunächst über weitere Layer geführt wird, um dann mit der concat-Operation verkettet zu werden. Bei den weiteren Layern handelt es sich um zwei Convolutional-Layer, auf welche anschließend die TensorFlow Operation *tf.image.resize()* folgt, welche die Nearest-Neighbor Interpolation ausgeführt, um die Größe der Ausgabe für die nachfolgende Verkettung anzupassen. Nach der ausgeführten Verkettung folgen abschließend noch zwei Convolutional-Layer, von denen der zweite eine Ausgabe der Form 26 x 26 x 255 generiert, um damit kleinere Objekte zu erkennen.
Die Ausgabe am Ende des Backbone-Netzwerks führt zudem über zwei andere Convolutional-Layer zu einem dritten Convolutional-Layer, der mit der Form 13 x 13 x 255 das Erkennen größerer Objekte ermöglicht.
Die Form dieses und auch der übrigen Layer kann der Tabelle 6.1 sowie der Abbildung 6.3 entnommen werden. Die Convolutional-Layer in dieser Struktur verwenden die *same-Padding*-Strategie und bewirken auf diese Weise, dass die Größe der einzelnen Feature Maps hinsichtlich Breite unverändert bleibt, während sich die Anzahl der Feature Maps und damit die Tiefe der einzelnen Convolutional-Layer jeweils verdoppelt. Als Aktivierungsfunktion wird die *Leaky-ReLu*-Funktion verwendet, welche im Vergleich zur klassischen ReLu Funktion nicht für jeden negativen Eingang den Wert Null ausgibt, sondern eine Ausgabe entsprechend

$$f(z) = az \tag{6.1}$$

erzeugt. Der Parameter a hat dabei einen festen Wert von 0,01 (vgl. [Goodfellow et al., 2016], S. 193).



INPUT-LAYER		
Layer (Type)	**Output Shape**	**Parameter #**
input_86 (Input-Layer)	[(None, 416, 416, 3)]	0
BLOCK 1		
conv2d_1113 (Conv2D)	(None, 416, 416, 16)	432
max_pooling2d_510 (MaxPooling2D)	(None, 208, 208, 16)	0
conv2d_1114 (Conv2D)	(None, 208, 208, 32)	4608
max_pooling2d_511 (MaxPooling2D)	(None, 104, 104, 32)	0
conv2d_1115 (Conv2D)	(None, 104, 104, 64)	18432
max_pooling2d_512 (MaxPooling2D)	(None, 52, 52, 64)	0
conv2d_1116 (Conv2D)	(None, 52, 52, 128)	73728
max_pooling2d_513 (MaxPooling2D)	(None, 26, 26, 128)	0
conv2d_1117 (Conv2D)	(None, 26, 26, 256)	294912
BLOCK 2		
max_pooling2d_514 (MaxPooling2D)	(None, 13, 13, 256)	0
conv2d_1118 (Conv2D)	(None, 13, 13, 512)	1179648
max_pooling2d_515 (MaxPooling2D)	(None, 13, 13, 512)	0
conv2d_1119 (Conv2D)	(None, 13, 13, 1024)	4718592
conv2d_1120 (Conv2D)	(None, 13, 13, 256)	262144
BLOCK 3		
conv2d_1123 (Conv2D)	(None, 13, 13, 128)	32768
tf_op_layer_ResizeNearestNeighbor_85 (TensorFlowOpLayer)	[(None, 26, 26, 128)]	0
BLOCK 4		
tf_op_layer_concat_449 (TensorFlowOpLayer)	[(None, 26, 26, 384)]	0
conv2d_1124 (Conv2D)	(None, 26, 26, 256)	884736
conv2d_1125 (Conv2D)	(None, 26, 26, 255)	65535
BLOCK 5		
conv2d_1121 (Conv2D)	(None, 13, 13, 512)	1179648
conv2d_1122 (Conv2D)	(None, 13, 13, 255)	130815

Abbildung 6.3.: Blockdiagramm zur Verarbeitung von Eingangsdaten durch das YOLOv3-Tiny Modell und Darstellung der einzelnen Layer mitsamt deren Eigenschaften und insgesamt 8.858.734 Parametern (inkl. den nicht aufgeführten Parametern der Batch Normalization). Blau hervorgehoben: Layer des YOLOv3-Tiny Backbone-Netzwerks.

Die Reduzierung der Bildgröße, hier eine Halbierung der Breite und Höhe, erfolgt entsprechend der genutzten Pooling-Layer nach dem Max-Pooling-Prinzip. Davon ausgenommen ist der letzte Pooling-Layer, der keine Größenänderung bewirkt.
Eine ausführlichere Auflistung der in Abbildung 6.3 gezeigten Layer-Struktur, inklusive der hier nicht aufgelisteten Batch Normalization und der Leaky-ReLu-Funktion, findet sich in Anhang A.7. Der Quellcode zur Implementierung des YOLOv3-Tiny Modells respektive der Personenerkennung und der entsprechenden Datenakquisition ist dem GitHub Repository [pythonlessons, 2020] entnommen. In Bezug auf die an das Modell gestellten Anforderungen ist der Quellcode dieses Repositories für die vorliegende Aufgabe geeignet und wird aus diesem Grund im Rahmen dieser Arbeit eingesetzt. Entsprechend des Verwendungszwecks ist dieser Quellcode an den relevanten Stellen modifiziert und erweitert. Die entsprechenden Python-Dateien finden sich im digitalen Anhang.

Für den Gebrauch des Jetson Nano Development-Kits und folglich den Einsatz des beschriebenen Modells, wird ein Image benötigt, welches als Open Source im NVIDIA Download Center verfügbar ist. Im realisierten Embedded Vision System ist das Image *jetson-nano-4gb-p441-sd-card-image* mit dem NVIDIA JetPack™ Software Development Kit (SDK), Version JP 4.4.1, auf der MicroSD-Speicherkarte installiert. Das JetPack Software Development Kit enthält das Betriebssystem Ubuntu 18.04 LTS in der 64-bit Version. Nach der Installation des Images, für welche ein zusätzlicher Rechner erforderlich ist, wird der Jetson Nano und dementsprechend das Embedded Vision System autark betrieben. Davon ausgenommen ist das Training des Modells, welches über einen kostenlosen Cloud-Service erfolgt, der GPUs bzw. die entsprechende Rechenleistung bereitstellt.
Neben dem Betriebssystem beinhaltet das JetPack verschiedene KI-, Deep Learning und Computer Vision Bibliotheken sowie APIs (Application Programming Interfaces) und unterstützt in Verbindung mit dem Jetson Nano unterschiedliche Deep Learning Netze bzw. Modelle sowie Frameworks (vgl. [Franklin, 2019]). Auf diese Weise ist eine gewisse Unterstützung bei der Einrichtung sowie dem ersten Gebrauch des Jetson Nano Development-Kits gegeben. Im Kontext dieser Arbeit ist dabei die Unterstützung des YOLOv3-Tiny Modells und des Frameworks TensorFlow inklusive Keras von Relevanz. Der Einsatz des Raspberry Pi Kameramoduls V2 ist nach dem Plug-and-Play Prinzip möglich, da das JetPack ebenfalls eine Unterstützung von IMX219 Sensoren bietet, welche in den Raspberry Pi Kameramodulen eingesetzt werden.
Eine grundlegende Konfiguration innerhalb des Betriebssystems ist die freigegebene Leistung, die für Machine Learning Anwendungen und den Betrieb des Jetson Nanos abrufbar ist. Der Jetson Nano wird im 10 Watt Modus betrieben, in welchem die Leistungsfähigkeit maximal ist. Auf Grund der Spannungsversorgung mittels Netzteil, ist dies uneingeschränkt möglich. Im Falle eines Akku-Betriebs kann ein Wechsel in den energiesparenderen 5 Watt Modus erfolgen. Die Leistungsfähigkeit wäre infolgedessen niedriger.

6.2. Training des Modells

In diesem Unterkapitel wird das Trainingsverfahren des eingesetzten Modells beschrieben. Im Kontext dieser Arbeit umfasst das Trainingsverfahren die Vorverarbeitung und Vorbereitung des eingesetzten Datensatzes sowie das Training an sich. Die nachfolgenden Beschreibungen umfassen dementsprechend Erläuterungen zur Vorverarbeitung und Vorbereitung der Trainingsdaten, dem durchgeführten Training und der genutzten Trainingsplattform.
Der verwendete gesamte Datensatz setzt sich entsprechend des konzeptionellen Teils (Kapitel 5.2) zum einen aus Bilddaten zusammen, die auf der Webseite[1] der Multi Object Tracking (MOT) Challenge verfügbar sind und zum anderen aus eigens für die vorliegende Arbeit generierten Bilddaten, die Personen im Einsatzbereich des Embedded Vision Systems zeigen. Der Datensatz der MOT Webseite trägt die Bezeichnung PETS09-S2L1 und umfasst 795 (Trainings-) Bilder. Die entsprechenden Ground Truth Daten liegen als einfache CSV-Textdatei vor und können mit den Bildern heruntergeladen werden. In diesen Textdateien sind originär mehr Informationen enthalten, als es für die vorliegende Arbeit erforderlich ist (siehe Anhang A.5). Darüber hinaus liegen die Koordinaten der Bounding Boxes noch nicht in der benötigten Form vor.
Um ein Training auszuführen, wird im Rahmen dieser Arbeit eine CSV-Textdatei benötigt, in der für jedes Bild des Datensatzes genau eine Zeile vorliegt. Dabei enthält jede Zeile den absoluten Dateipfad des entsprechenden Bildes, die Bounding Box Koordinaten X_{min}, Y_{min}, X_{max} und Y_{max} sowie die Klassenbezeichnung des Objektes. Hier handelt es sich bei X_{min} um die linke und X_{max} um die rechte Seite einer Bounding Box. Y_{min} markiert die Oberseite einer Bounding Box und Y_{max} folglich die Unterseite. Befinden sich mehrere Objekte, in der vorliegenden Arbeit Personen, innerhalb eines Bildes, werden die Bounding Box Koordinaten und die Klassenbezeichnungen aller Personen in die Zeile des zugehörigen Bildes geschrieben. Die Daten unterschiedlicher Personen sind durch Leerzeichen getrennt.

Bei dem eigens generierten Datensatz handelt es sich wie in Kapitel 5.2 beschrieben um einzelne Bilder aus aufgezeichneten Videosequenzen. Nach dem Markieren der in diesen Bildern dargestellten Personen mittels Bounding Boxes und den (Klassen-)Bezeichnungen der Personen durch Labels, erzeugt das Tool LabelImg Ground Truth Daten, die im YOLO-Format vorliegen. Bei diesem Format liegt für jedes Bild eine einzelne Textdatei vor, in der für jedes Objekt eine separate Zeile angelegt ist. Jede Zeile enthält die Bezeichnung der Objekt-Klasse, den x-Wert und den y-Wertes des Mittelpunktes der Bounding Box sowie die Breite und Höhe der Bounding Box. Diese Daten sind jeweils durch Leerzeichen voneinander getrennt. Abgesehen von der Objekt-Klasse handelt es sich um normierte Daten, so das diese einen Wert zwischen 0 und 1 annehmen. Dementsprechend müssen auch

[1] https://motchallenge.net/data/MOT15/

diese Daten für die Verwendung in dieser Arbeit in das hier erforderliche Format konvertiert werden.
Zunächst werden dazu die Daten der einzelnen Textdateien in einer gemeinsamen Textdatei zusammengefasst. Dies kann beispielsweise mit dem Eingabebefehl *Copy/b* .txt NameDerNeuenDatei.txt* in einer Kommandozeile ausgeführt werden. Der Eingabebefehl muss in dem Verzeichnis ausgeführt werden, in dem die einzelnen Textdateien gespeichert sind. Die noch vorhandenen Leerzeichen zwischen den einzelnen Daten jeder Zeilen können durch einen einfachen *finden und ersetzen* Befehl (Tastenkombination STRG + H) durch Kommata ersetzt werden. Abschließend wird jede Zeile am Zeilenanfang durch eine manuelle Eintragung um die Nummer des korrespondierenden Bildes, welche zudem dem Dateinamen des jeweiligen Bildes entsprechen muss, erweitert. Die so vorverarbeitete Datei kann nun mittels Skript[2] in das zuvor beschriebene erforderliche Format konvertiert werden.
Für die Aufteilung des gesamten Datensatzes in Trainings- und Testdaten dienen die Angaben in [Gerón, 2019], Seite 31, als Richtwert. Dort wird eine Aufteilung in 80 % Trainings- und 20 % Testdaten angegeben. Die Bilder des PETS Datensatzes sind genau in 80 % Trainings- und 20 % Testdaten aufgeteilt. Für die am Einsatzort aufgenommenen Bilder gilt die gleiche Aufteilung, so dass demnach auch der gesamte Datensatz 80 % Trainings- und 20 % Testdaten umfasst. Bei den verwendeten vortrainierten Gewichten handelt es sich um die *yolov3-tiny.weights* Gewichte der YOLO-Webseite[3].

Für das Training mit den präparierten Ground Truth Daten, dem Datensatz sowie den vortrainierten Gewichten wird die Plattform *Google Colaboratory* (kurz Colab) verwendet. Hierbei handelt es sich um eine Virtuelle Maschine (VM), die kostenlos GPUs und TPUs bzw. die entsprechende Rechenleistung zur Verfügung stellt. Colab bietet eine interaktive Umgebung für das Erstellen von Python-Notebooks. Bei diesen handelt es sich um *Jupyter*-Notebooks, die im persönlichen Google Drive gesichert werden können. Die Verwendung von Colab erfolgt über das Starten einer Laufzeit und die Auswahl eines Hardware-Beschleunigers respektive einer der wählbaren Processing Units. Im Rahmen dieser Arbeit ist das Training unter Verwendung einer *Tesla V100-SXM2-16 GB* Grafikkarte mit einer GPU als Hardware-Beschleuniger erfolgt.
Vor dem Start des Trainings ist es erforderlich die lokal auf dem Jetson Nano gesicherten Dateien, welche für das Training erforderlich sind, in Google Drive zu laden. Es empfiehlt sich, die gesamte lokale Ordnerstruktur[4] inklusive der enthaltenen Unterordner und Dateien ebenfalls in Google Drive zu hinterlegen, da sich Unterschiede in den Ordnerstrukturen respektive den darin enthaltene Dateien als mögliche Fehlerquelle bei der späteren lokalen

[2] Die Skripte zur Umsetzung der Konvertierungen finden sich im digitalen Anhang dieser Arbeit. Ferner findet sich dort zur Veranschaulichung ein Auszug einer CSV-Textdatei (Dateiname *Auszug_Dataset_train.txt*) im geforderten Format.
[3] https://pjreddie.com/darknet/yolo/
[4] Diese Aussage bezieht sich auf Ordner und Dateien, welche für die vorliegende Arbeit von Relevanz sind.

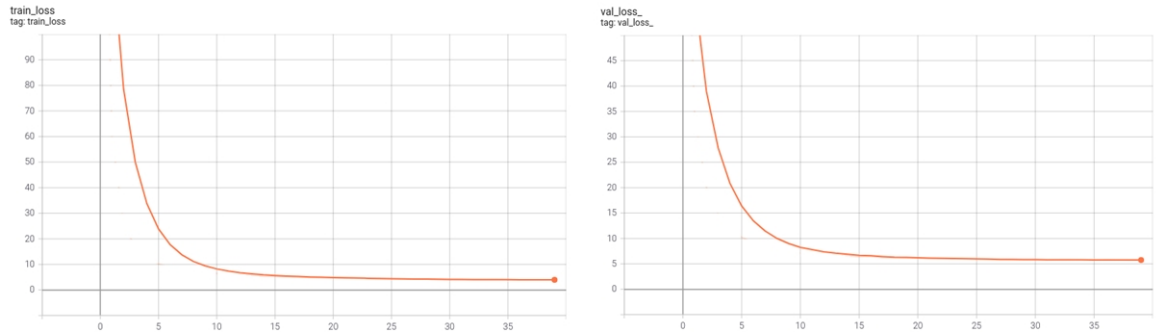


Abbildung 6.4.: Werteverlauf der Verlustfunktion für den Trainings- (links) und Test-Datensatz (rechts)

Ausführung des trainierten Modells herausstellen können.
Nach dem Start einer Laufzeit ist eine Verbindung zwischen Colab und dem verwendeten Google Drive erforderlich, damit die für das Training erforderlichen Daten sowie das Trainingsskript verfügbar bzw. ausführbar sind. Hier ist zu beachten, dass die Ground Truth Daten für das Training des YOLO-Modells die absoluten Dateipfade zu den Trainings- und Testbildern benötigen. Da der Trainingsprozess innerhalb von Colab ausgeführt wird, müssen nach der Verbindung von Colab und Google Drive die noch auf den Jetson Nano angepassten lokalen Dateipfade entsprechend aktualisiert werden. Anschließend wird das Trainingsskript inklusive den erforderlichen Funktionen über eine Codezelle in Colab importiert und der Trainingsprozess ausgeführt.
Für den Trainingsprozess werden Batches der Größe 8 verwendet. Dieser Wert orientiert sich an der Größe des eingesetzten Datensatz und an dem bereitgestellten Grafikkartenspeicher der NVIDIA Tesla V100-SXM2. Die Grafikkarte bietet mit 16 GB einen ausreichend großen Speicher für die gewählte Batch-Größe. Diese wird nicht größer gewählt, da es sich bei 824 Trainingsbildern um eine verhältnismäßig kleine Trainingsmenge handelt. Die Lernrate entwickelt sich während des Trainings dynamisch und aktualisiert ihren Wert dementsprechend automatisch. Die Größe für den Input-Layer ist auf 416 x 416 Pixel eingestellt. Die Wahl dieser Bildgröße ermöglicht eine Einflussnahme auf die Genauigkeit sowie die Geschwindigkeit des Modells. Mit steigender Auflösung verbessert sich die Genauigkeit, wohingegen sich die Geschwindigkeit reduziert. Um die Geschwindigkeit des Modells zu erhöhen, kann die Auflösung reduziert werden. Dabei reduziert sich auch die Genauigkeit des Modells.
Das Modell wird über 40 Epochen trainiert und erreicht nach dieser Trainingszeit eine mean Average Precision (mAP) von 88,566 %. In der vorliegenden Arbeit entspricht die mAP der Average Precision (AP), da ausschließlich Personen als zu erkennende Objekte definiert sind. Nach 40 Epochen liegt der Wert der Verlustfunktion für den Trainingsdatensatz bei 3,915, der entsprechende Wert für den Testdatensatz beträgt 5,74. In Abbildung 6.4 zeigt sich, dass die Werte der Verlustfunktion für den Trainings- und Testdatensatz eine gleichmä-

ßige Entwicklung nehmen und nach den gewählten 40 Trainingsepochen ein relativ konstantes Werteniveau erreicht haben.

6.3. Laufweganalyse

Zum Abschluss der Beschreibung des realisierten Embedded Vision Systems wird in diesem Unterkapitel mit der Laufweganalyse die Systemfunktion respektive der Verwendungszweck des Systems beschrieben. In der ersten Phase der Laufweganalyse erfolgt vorbereitend für den weiteren Analyseprozess eine Bestimmung der Regions of Interest, welche vereinfacht als Teilbereiche bezeichnet werden. Die zweite Phase umfasst das Aufzeichnen einer Videosequenz und die Personenerkennung unter Verwendung des ausgewählten YOLO-Modells. Im Kontext dieser Arbeit wird die Personenerkennung als die Datenakquisition betrachtet, da diese anhand von Bounding Boxes die Positionsdaten erkannter Personen liefert, auf deren Basis die Laufweganalyse weitergeführt wird. Zum Abschluss der zweiten Phase werden die relevanten Positionsdaten lokal gespeichert und für die dritte Phase bereitgestellt. In der dritten Phase werden die Positionsdaten verarbeitet, um sie für die Laufweganalyse nutzbar zu machen bzw. für eine Einsatzplanung von Transportsystemen einsetzen zu können. Diese Positionsdaten werden den einzelnen Teilbereichen zugewiesen und eine dementsprechende Visualisierung generiert. Die Visualisierung bildet die Grundlage für die Interpretation der Ergebnisse im Rahmen der Evaluation in Kapitel 7.

6.3.1. Bestimmung der Teilbereiche

Die Grundlage der Laufweganalyse bildet die Bestimmung der Teilbereiche, für die eine Untersuchung der Nutzungshäufigkeit durch Personen erfolgen soll. Im Kontext dieser Arbeit sind die Bereiche relevant, die sowohl von Personen als Laufwege als auch von Transportsystemen als Fahrwege genutzt werden können. Bei dem zum Praxistest des Embedded Vision Systems genutzten Einsatzort handelt es sich um einen verkehrsberuhigten Bereich eines Betriebsgeländes mit Pkw-Stellplätzen. Dieser verbindet unterschiedliche Teile des Geländes, woraus ein regelmäßiger Personenverkehr resultiert. Darüber hinaus wird dieser Bereich in unregelmäßigen Zeitabständen von Transportfahrzeugen befahren. Diese können stellvertretend als die in der Zielsetzung (Unterkapitel 1.1) genannten mobilen Transportsysteme betrachtet werden. Ausschließlich in begrenzten Zeiträumen tritt ein erhöhter Pkw-Verkehr auf, der in der vorliegenden Arbeit jedoch keine weitere Berücksichtigung findet.
Zur Veranschaulichung der Laufweganalyse und einer begründbaren Auswahl der Teilbereiche soll für den praktischen Einsatz des Embedded Vision Systems ein beispielhaftes Szenario angenommen werden. In Abhängigkeit definierter Zeitintervalle soll analysiert werden, wie viele Personen sich durchschnittlich innerhalb der einzelnen Intervalle im Bereich der

Abbildung 6.5.: Aufnahmebereich aus der Perspektive des Embedded Vision Systems inklusive des Analysebereichs und der einzelnen Teilbereiche

Pkw-Stellplätze bewegen und welche der zwei Stellplatzbuchten häufiger aufgesucht wird. Ferner soll innerhalb derselben Zeitintervalle analysiert werden, in welchem Bereich vor dem Gebäude sich Personen, die dieses betreten oder verlassen, häufiger bewegen. In diesem Zusammenhang werden zwei Teilbereiche mit den laufenden Nummern *1* und *2* vor den Pkw-Stellplätzen sowie zwei weitere Teilbereiche mit den laufenden Nummern *3* und *4* links und rechts des Gebäudeeingangs bestimmt.
Die hierbei akquirierten Analysedaten liefern im Rahmen des angenommenen Szenarios für die Einsatzplanung der Transportsysteme Informationen darüber, welcher Laufweg im Bereich der Pkw-Stellplätze stärker frequentiert ist. Aus dieser Kenntnis kann überdies geschlossen werden, wann ein erhöhter Pkw-Verkehr zu erwarten ist, sofern dies bei einer Untersuchung berücksichtigt werden soll. Des Weiteren liefern diese Daten Informationen darüber, welche Laufwege vorzugsweise zwischen einzelnen Gebäuden und Bereichen des Betriebsgeländes genutzt werden.
Der Bereich für den die Laufweganalyse ausgeführt wird, ist in den Abbildungen 6.5 und 6.6 dargestellt. Abbildung 6.5 zeigt diesen Bereich aus der Perspektive des Embedded Vision Systems. Bei Abbildung 6.6 handelt es sich um zwei von *Google Maps* abgerufene Satellitenbilder[5] des Einsatzortes.

Der relevante Analysebereich und die einzelnen Teilbereiche sind in beiden Abbildungen mit farbigen Rahmen skizziert. Die etwaige Position des Embedded Vision Systems ist im

[5] Abruf am 19.04.2021
https://www.google.de/maps/@53.1026445,8.8574001,32m/data=!3m1!1e3?hl=de
https://www.google.de/maps/@53.102694,8.8573928,51a,35y,17.06h/data=!3m1!1e3?hl=de

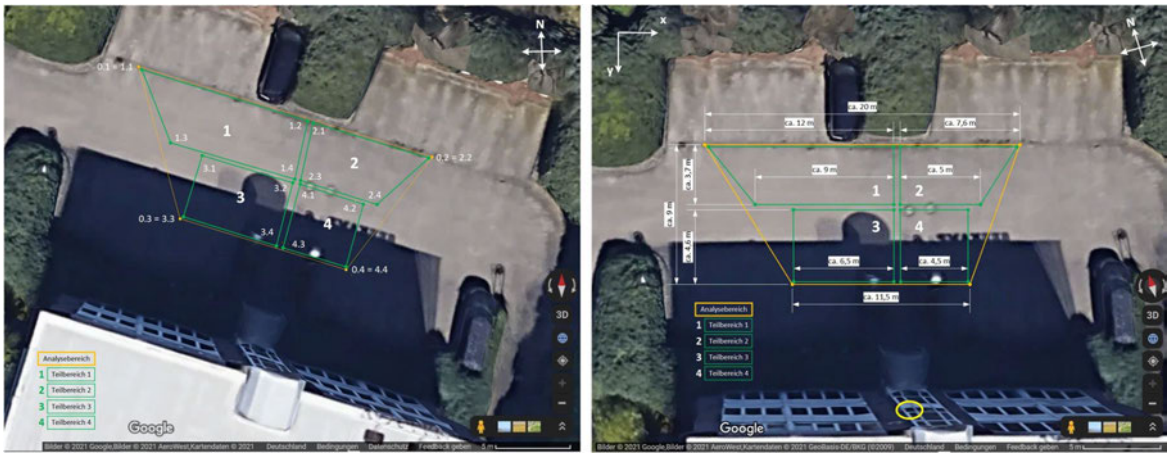


Abbildung 6.6.: Darstellung des Analysebereichs und der einzelnen Teilbereiche aus unterschiedlichen Perspektiven. Links: Satellitenbild des Einsatzortes in Nord-Süd-Ausrichtung mit den Bezeichnungen der Eckpunkte. Rechts: Gedrehtes Satellitenbild des Einsatzortes entsprechend der Ausrichtung des Embedded Vision Systems

rechten Bild der Abbildung 6.6 mit einer gelben Umrundung gekennzeichnet. Diese Position befindet sich in der 2. Etage des Bürogebäudes in einer Höhe h von etwa $8\ m$ über dem Grund. Bei einem maximalen Abstand von $l_{max} \approx 14,7\ m$ zwischen Embedded Vision System und dem äußeren Rand des relevanten Analysebereichs am Grund und einem minimalen Abstand von $l_{min} \approx 6,7\ m$, ergibt sich nach den Gleichungen 6.2, 6.3 und 6.4 ein mittlerer Aufnahmewinkel α_{mittel} von ca. 50°. Dieser entspricht in ausreichender Näherung dem entsprechend A.4.1-01 vorgegebenen Richtwert von 45° (Toleranz $\pm 15°$) in Bezug auf die Vertikale.

$$\alpha_1 = \arctan \frac{l_{max}}{h} = \arctan \frac{15,2\ m}{8\ m} \approx 62,2° \tag{6.2}$$

$$\alpha_2 = \arctan \frac{l_{min}}{h} = \arctan \frac{6,2\ m}{8\ m} \approx 37,8° \tag{6.3}$$

$$\alpha_{mittel} = \frac{\alpha_1 + \alpha_2}{2} = 50° \tag{6.4}$$

Aus der beschriebenen Lage und dem genannten Winkel erfasst das Kameramodul des Embedded Vision Systems eine für die Laufweganalyse relevante Fläche von etwa $142\ m^2$, welche bzgl. A.4.1-02 einer ausreichenden Fläche entspricht und als der relevante Analysebereich angenommen wird. Die einzelnen Abmessungen sind im rechten Bild der Abbil-

dung 6.6 aufgeführt.
Die genaue Festlegung der Teilbereiche erfolgt mittels GPS-Dezimalkoordinaten. Dies erlaubt eine genaue Definition der Teilbereiche hinsichtlich deren realer Größe und der Positionen der entsprechenden Eckpunkte, da hierbei, anders als im Kamerabild des Embedded Vision Systems, keine perspektivischen Verzerrungen auftreten. Entsprechend der Anforderung A.4.1-05 werden auf diese Weise perspektivische Verzerrungen bereits berücksichtigt. Zudem können GPS-Koordinaten Informationen liefern, die ohne eine Konvertierung in ein anderes Datenformat unmittelbar für die Einsatzplanung von Transportsystemen nutzbar sind. Bezüglich der GPS-Koordinaten bietet das Format *Dezimalgrad* den Vorteil, dass Längen- und Breitengrade hierbei mit jeweils einem einzelnen Wert angegeben werden. Dies ermöglicht eine praktikable Weiterverwendung der entsprechenden Daten.
Um die Bestimmung der Teilbereiche zu realisieren und in diesem Zusammenhang eine Vorarbeit für spätere Bearbeitungsschritte im Rahmen des Unterkapitels zur Visualisierung zu treffen, werden vier Eckpunkte für den Analysebereich in der Aufnahme des Embedded Vision Systems sowie die korrespondierenden Punkte im gedrehten Satellitenbild (siehe Abbildung 6.6, rechtes Bild) gesetzt. Dazu werden Marker verwendet, die am Einsatzort direkt am Rand des erfassbaren Aufnahmebereichs des Embedded Vision Systems platziert sind und so die Eckpunkte des relevanten Analysebereichs kennzeichnen. Die Positionen der entsprechenden Bildpunkte in der Aufnahme des Embedded Vision Systems werden für eine spätere Verwendung festgehalten. Die dazu korrespondierenden Eckpunkte im Satellitenbild werden anhand markanter Orientierungspunkte und den vermessenen Abständen zwischen den Markern ermittelt und die entsprechenden GPS-Koordinaten erfasst. Durch das Platzieren der Eckpunkte direkt am Rand des Aufnahmebereichs kann eine möglichst große, relevante Fläche für die Laufweganalyse berücksichtigt werden. Für jeden einzelnen Teilbereich sind jeweils vier weitere Eckpunkte im Satellitenbild gesetzt und die entsprechenden GPS-Koordinaten erfasst. Jeder der Eckpunkte des Analysebereichs entspricht gleichzeitig einem der Eckpunkte der Teilbereiche. Zum einen wird hierdurch der Analysebereich in Richtung der Ecken vollständig ausgenutzt und zum anderen handelt es sich um bereits bekannte Punkte, deren Verwendung pragmatisch begründet ist. Hier ist zu beachten, dass die Eckpunkte des Analysebereichs und die entsprechenden Eckpunkte der einzelnen Teilbereiche nicht exakt übereinander liegen, sonder einen leichten Versatz aufweisen. Dieser Versatz dient lediglich dazu, alle Eckpunkte ohne eine gegenseitige Überdeckung darstellen zu können.
Die Teilbereiche *3* und *4* grenzen bzgl. der x-Achse[6] (Bildbreite) in Richtung der Bildmitte etwa zentral vor der Eingangstür des Bürogebäudes aneinander. Auf diese Weise können

[6] Obwohl die Definition der Teilbereiche auf GPS-Daten basiert, wird bei den Erläuterungen zu deren Bestimmung kein Bezug auf Längen- und Breitengrade genommen, sondern auf die Achsen in x- und y-Richtung. Da es sich um ein gedrehtes Satellitenbild handel, verlaufen Längen- und Breitengrade nicht mehr vertikal respektive horizontal durch das Bild. Da dies im vorliegenden Kontext für Erläuterungen ungeeignet ist, wird auf ein eingeführtes Koordinatensystem mit x- und y-Achse Bezug genommen.

Personen mit der direkt dem Teilbereich zugewiesen werden, durch welchen diese Personen sich tatsächlich bewegen werden. Wäre vor der Eingangstür lediglich ein einzelner Teilbereich angeordnet, würden alle Personen die das Bürogebäude betreten oder verlassen, zunächst diesem Bereich zugeordnet werden, unabhängig davon, welchen Laufweg diese Personen anschließend wählen. Die Teilbereiche *3* und *4* stellen sich als Rechtecke dar, da diese Geometrie eine einfache Aufteilung eines Gesamtbereichs in definierte Teilbereiche ermöglicht. Ferner sollen beide Teilbereiche eine möglichst gleichgroße Fläche aufweisen. Auf Grund der Position des Embedded Vision Systems, welche sich auf Grund örtlicher Gegebenheiten nicht zentral über dem Gebäudeeingang befindet, ist dies jedoch nicht vollständig realisierbar.
Die Teilbereiche *1* und *2* sind, analog zu den Teilbereichen *3* und *4*, nach dem selben Prinzip bestimmt. Da die Eckpunkte *0.1* und *0.2* des Analysebereich gleichzeitig jeweils einen Eckpunkt dieser Teilbereiche darstellen, weisen diese jedoch keine rechteckige Geometrie auf. Im Falle einer rechteckigen Form, befänden sich die Eckpunkte *1.3* und *2.4* der Teilbereiche *1* und *2* außerhalb des Analyse- als auch des erfassbaren Aufnahmebereichs. Dementsprechend sind die letztgenannten Punkte hinsichtlich der x-Achse in Richtung der Bildmitte und damit in den Analysebereich verschoben. Alle Teilbereiche grenzen hinsichtlich der y-Achse (Bildhöhe) ungefähr mittig innerhalb des Analysebereichs aneinander, um eine annähernd symmetrische und gleichgroße Aufteilung zu erreichen. Die Abmessungen der Teilbereiche *3* und *4* sind ca. einen Meter größer als die der Teilbereiche *1* und *2* gewählt, da diese wiederum hinsichtlich der x-Achse größere Abmessungen aufweisen und so die flächenmäßigen Unterschiede reduziert werden. Um zu gewährleisten, dass Personen eindeutig einem Teilbereich zuzuordnen sind, befinden sich zwischen allen Teilbereichen Lücken, in denen erkannte Personen für die Analyse nicht berücksichtigt werden.
Eine Auflistung der GPS-Koordinaten aller Eckpunkte findet sich in Anhang A.8.

6.3.2. Datenakquisition für die Laufweganalyse

Die Datenakquisition wird als die Phase von dem Aufzeichnen einer Videosequenz bis zum Sichern der für die Laufweganalyse relevanten und erforderlichen Daten definiert. Die Übersicht dieser Phase ist in Abbildung 6.7 dargestellt.

Die mit dem Embedded Vision System aufgezeichnete Videosequenz hat eine Dauer von 130 Sekunden, eine Einzelbildrate von 30 Bildern pro Sekunde und eine Bildauflösung von 1280 x 720 Pixeln. Nachdem eine aufgezeichnete Videosequenz vorliegt, stellt die Personenerkennung mit dem YOLO-Tiny Modell, inklusive dem Markieren der erkannten Personen mittels Bounding Boxes, den grundlegenden und wesentlichen Schritt der Datenakquisition dar. Für die weiteren Schritte der Datenakquisition sowie der Laufweganalyse insgesamt, wird je erkannter Person ein bestimmter Bildpunkt respektive die entsprechende Position des Bildpunktes in x- und y-Richtung weiter verarbeitet. Bei diesem Punkt handelt es sich

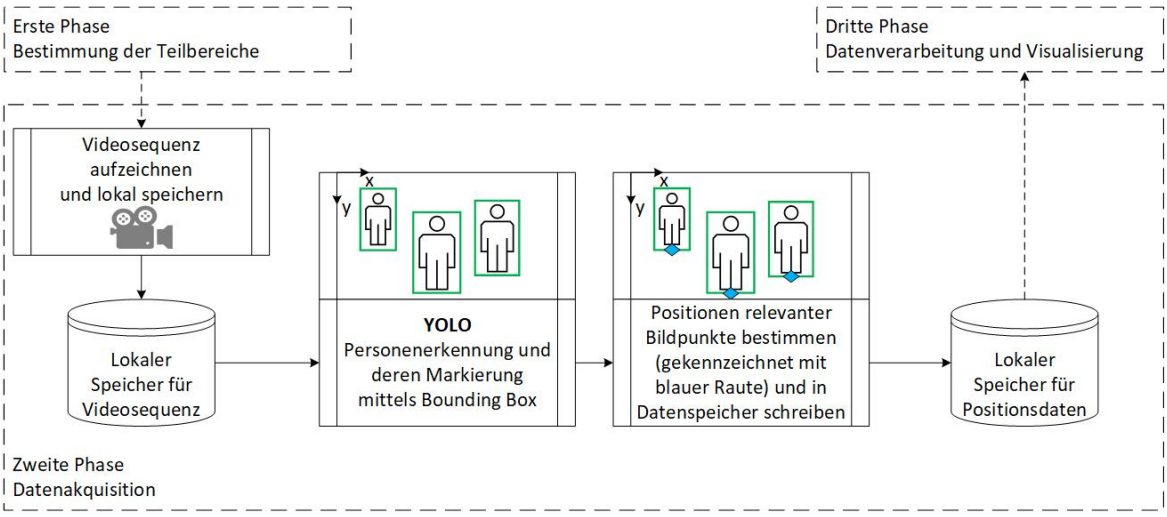


Abbildung 6.7.: Datenakquisition von der Aufnahme der Videosequenz bis zum Sichern der Positionsdaten

um die etwaige Position der Füße einer Person[7], da diese in größter Näherung der realen Position einer Person entspricht. In Abbildung 6.7 ist diese für die einzelnen Personen jeweils mit einer blauen Raute gekennzeichnet. Der Nullpunkt des Bezugssystems befindet sich in der oberen linken Ecke eines Bildes.
Die konkreten Werte der Fußpositionen jeder erkannten Person ergeben sich basierend auf den ausgegebenen Bounding Box Daten aus

$$x_{fuss} = \frac{x_2 - x_1}{2} + x_1 \tag{6.5}$$

und

$$y_{fuss} = y_2 \tag{6.6}$$

mit x_{fuss} als Fußposition in x-Richtung, y_{fuss} als Fußposition in y-Richtung, x_1 als linke und x_2 als rechte Begrenzung jeder Bounding Box sowie y_1 als die obere und y_2 als die untere Begrenzung der einzelnen Bounding Boxes. Die Daten, die neben den berechneten Werten für x_{fuss} und y_{fuss} in eine als Datenspeicher fungierende CSV-Datei geschrieben werden, sind in Tabelle 6.2 aufgeführt. Für jede erkannte Person werden die zugehörigen Daten in eine separate Zeile geschrieben. Werden innerhalb eines Frames mehrere Personen erkannt, wiederholt sich die Nummer des Frames in Spalte 0 der CSV-Datei so oft, bis alle Personendaten in separaten Zeilen erfasst sind.

[7] Aus Gründen der Lesbarkeit wird vereinfacht von der *Fußposition* gesprochen.

Tabelle 6.2.: Daten in der als lokaler Speicher fungierenden CSV-Datei

Spalte	Inhalt	Beschreibung
0	Frame	Nummer des Frames in dem eine Person erkannt wird
1	x_{fuss}	Fußposition einer erkannten Person in x-Richtung
2	y_{fuss}	Fußposition einer erkannten Person in y-Richtung
3	Klassen-ID	Klasse des erkannten Objekts; hier nur Person = 0
4	Konfidenzwert	Die *Zuversicht* / Sicherheit des Systems, ein Objekt erkannt zu haben

Die Klassen-ID wird in der vorliegenden Arbeit nicht benötigt, da nur die Klasse *Person* existiert. Da diese zusätzliche Angabe bzw. die entsprechenden Daten keinen wesentlichen Einfluss auf den erforderlichen Speicherbedarf haben, wird diese Information mit Ausblick auf mögliche Ergänzungen weiterer Klassen neben den bereits relevanten Daten gesichert. Darüber hinaus findet sich in der vierten Spalte der Konfidenzwert.

6.3.3. Datenverarbeitung und Visualisierung

In der dritten Phase werden zunächst zeilenweise die Daten aus der lokalen Datenbank respektive der CSV-Datei ausgelesen. Dabei werden die ausgelesenen Daten einer Zeile zuerst verarbeitet, bevor die Daten der Folgezeile ausgelesen werden. Nachdem die Daten der letzten Zeile verarbeitet sind, erfolgt die Visualisierung. Der entsprechende Programmablauf ist Abbildung 6.8 dargestellt und wird nachfolgend näher erläutert.

Nach dem Auslesen der Daten erfolgt eine perspektivische Transformation der Positionsdaten der Fußpositionen. Diese liegen noch als x- und y-Werte des entsprechenden Bildpunktes vor und werden in GPS-Dezimalkoordinaten transformiert. Dazu werden die in Unterkapitel 6.3.2 bestimmten GPS-Koordinaten der Eckpunkte des Analysebereichs sowie die korrespondierenden Positionsdaten der Eckpunkte im aufgezeichneten Bild des Kameramoduls herangezogen. Diese Punkte sind erforderlich, um eine 3×3 Transformationsmatrix zu bilden, die für das Ausführen der Transformation erforderlich ist. OpenCV bietet für die Berechnung dieser Matrix die Funktion *cv2.getPerspectiveTransform()* (siehe [Mordvintsev, 2013]), die als Parameter die Werte der zuvor genannten Eckpunkte benötigt. Die entsprechenden Werte sind ohne Angabe der Einheit so in der mittleren und rechten Spalte der Tabelle 6.3 aufgeführt, wie sie an die Funktion übergeben werden. Bei den Werten zu den Positionen der Bildpunkte ist folgendes zu beachten: Die angegebenen Werte, die um einen Faktor ergänzt sind, beziehen sich auf ein Einzelbild der aufgezeichneten Videosequenz, welches in der vorliegenden Arbeit eine andere Auflösung als die Videosequenz selbst aufweist. Da das

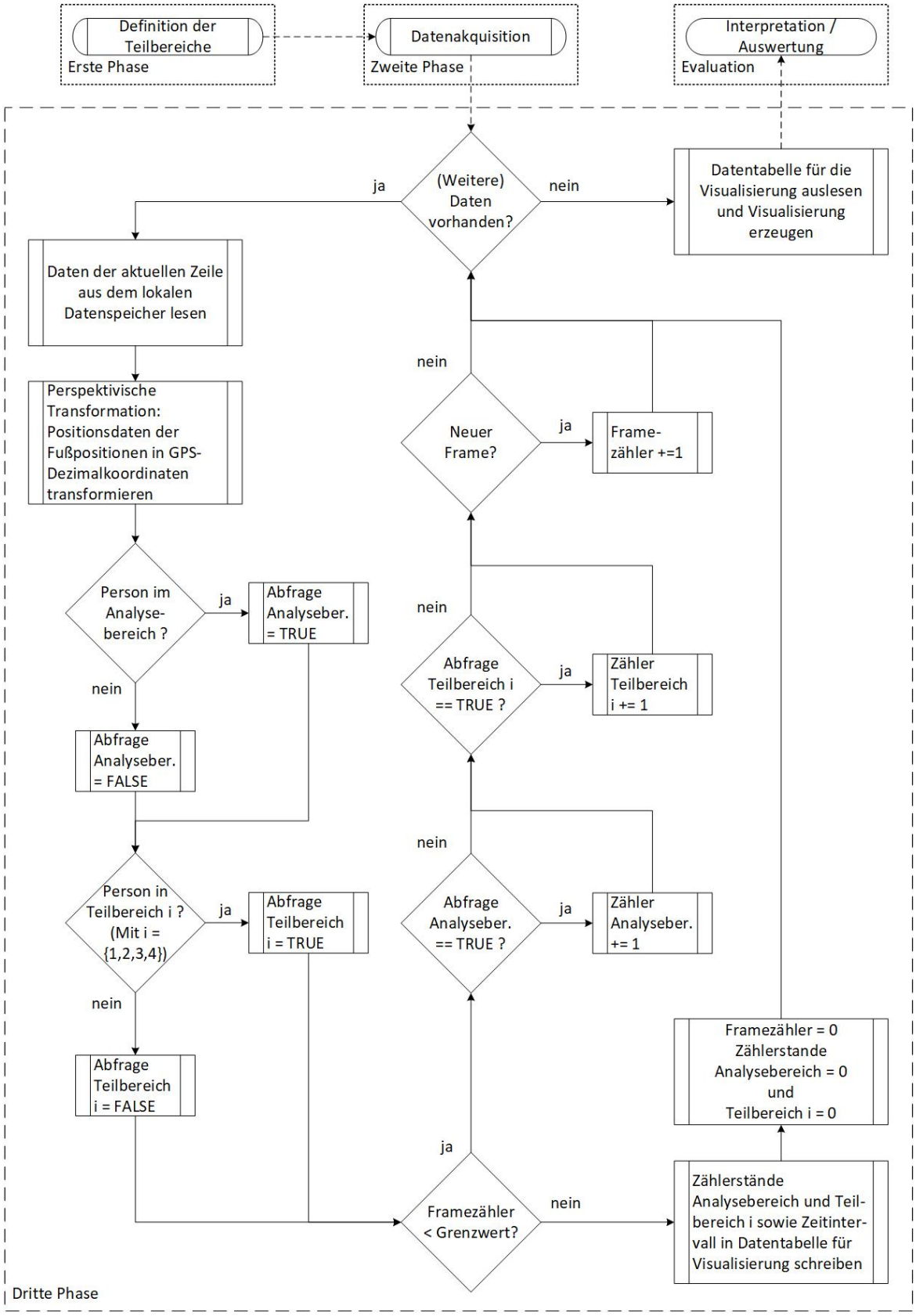


Abbildung 6.8.: Programmablauf der Datenverarbeitung bis zur Visualisierung

Tabelle 6.3.: Gegenüberstellung von GPS-Dezimalkoordinaten und den korrespondierenden Positionen der Bildpunkte in der Aufnahme des Embedded Vision Systems

Eckpunkt	**GPS-Dezimalkoordinaten (Längengrad, Breitengrad)**	**Position Bildpunkt (x, y $\cdot a$)**
0.1	(8.857385, 53.102731)	(1, 206 $\cdot a$)
0.2	(8.857684, 53.102676)	(1280, 186 $\cdot a$)
0.3	(8.857429, 53.102637)	(1,426 $\cdot a$)
0.4	(8.857603, 53.102606)	(1280,404 $\cdot a$)

YOLO Modell auf die Videosequenz angewendet wird, beziehen sich die Positionen erkannter Personen demnach auf die Auflösung der Videosequenz. Um dies zu berücksichtigen, wird bezüglich der y-Achse respektive der Vertikalen des Videobildes, der Skalierungsfaktor a einbezogen, welcher sich aus

$$a = \frac{vertikale\_Videoauflösung}{vertikale\_Bildauflösung} = \frac{720\,Pixel}{616\,Pixel} \tag{6.7}$$

ergibt.

Damit hinsichtlich der x-Achse respektive der Horizontalen des Videobildes, wie erforderlich, weiterhin die Bildpunkte an den äußersten Rändern des Aufnahmebereichs einbezogen werden, wird hier kein Skalierungsfaktor eingesetzt.

Unter Anwendung der Transformationsmatrix wird die perspektivische Transformation mittels der Funktion *cv2.PerspectiveTransform()* auf die Menge der einzelnen Fußpositionen respektive den entsprechenden Bildpunkten angewendet.

Es handelt sich um eine 3 x 3 Matrix, da die perspektivische Transformation Punkte einer zweidimensionalen Ebene, die in einen dreidimensionalen Raum eingebettet ist, auf eine andere zweidimensionale Ebene abbildet. Der wesentliche Aspekt ist es, die Punkte auf den zweidimensionalen Ebenen in homogenen Koordinaten zu repräsentieren, also mit drei Zahlenwerten. Hier besteht der Vorteil darin, dass in homogenen Abbildungen die projektiven Abbildungen linear sind. Trotz der drei Zahlenwerte handelt es sich um zweidimensionale Ebenen, da die Länge der Koordinatenvektoren für den Punkt, den sie beschreiben, nicht relevant ist. Auf diese Weise werden Punkte der Eingabe in die zweidimensionale Ebene der Ausgabe projiziert (vgl. [Bradski and Kaehler, 2008], S. 171f.). Durch diese Transformation wird entsprechend der Anforderung A.4.1-05 die Problemstellung der perspektivischen Verzerrung berücksichtigt.
Nach der perspektivischen Transformation liegen die Positionen erkannter Personen in Form von GPS-Dezimalkoordinaten vor. Diese werden zunächst für die Abfrage benötigt, ob sich Personen innerhalb des Analysebereichs befinden. Für die Laufweganalyse ist dies nicht von

unmittelbarer Relevanz, die Ergebnisse liefern jedoch eine zusätzliche Information über die Frequentierung des gesamten Analysebereichs.
Anschließend erfolgt die für die Laufweganalyse relevante Abfrage der Teilbereiche. Dazu wird überprüft, ob sich die GPS-Koordinaten einer Person innerhalb eines der Teilbereiche befinden. Hier bietet die Programmbibliothek Matplotlib eine entsprechende Lösung. Über das Modul *matplotlib.Path()* und die GPS-Koordinaten, mit denen die Teilbereiche definiert sind, werden Vielecke erzeugt und diese einer Variablen zugewiesen. Exemplarisch wird hier der Variablenname „teilbereich“ verwendet. Mittels *teilbereich.contains_point()* wird daraufhin abgefragt, ob sich die GPS-Koordinaten einer Person innerhalb eines Teilbereichs befinden. x Das Ergebnis der Abfrage wird entweder auf *TRUE* oder auf *FALSE* gesetzt. Die erste Abfrage, ob sich Personen innerhalb des gesamten Analysebereichs befinden, wird analog ausgeführt.
Im nachfolgenden Schritt erfolgt die Abfrage, ob der Wert des Framezählers unter dem angegebene Grenzwert von 162 liegt. Für das vorliegende Szenario wird angenommen, dass der Analysebereich 12 h aufgezeichnet wird und ein einzelnes Zeitintervall 0,5 h beträgt. In Bezug auf die mit dem Embedded Vision System aufgezeichnete Videosequenz entsprechen die 12 h folglich 130 s (Länge der Videosequenz). Daraus ergibt sich für 0,5 h eine Videolaufzeit von ca. 5,42 s. Bei einer Einzelbildrate von 30 Bildern pro Sekunde entspricht ein Zeitintervall damit ca. 162 Frames.
Solange der Grenzwert nicht erreicht ist, werden die Zählerstände des Analysebereichs sowie des Teilbereichs, dessen Abfrageergebnis *TRUE* lautet, erhöht. Sobald der Grenzwert erreicht wird, werden die Zählerstände in die Datentabelle für die Visualisierung geschrieben und anschließend inklusive dem Framezähler auf Null zurückgesetzt. Dieser iterative Prozess wird fortgesetzt, bis alle akquirierten Daten ausgelesen und verarbeitet sind.
Zum Abschluss der dritten Phase erfolgt, wie in den Abbildungen 6.9, 6.10 und 6.11 dargestellt, die Visualisierung der Zählerstände in Bezug auf die Zeitintervalle in Form von Balkendiagrammen[8]. Die zu den Diagrammen korrespondierenden Daten finden sich in der Tabelle, die hier in der Abbildung 6.12 dargestellt ist. Darin ist ersichtlich, dass es sich nicht um ganze Zahlen, sondern um Kommazahlen handelt. Im diesem Zusammenhang ist zu beachten, dass die angezeigten Zahlenwerte nicht wie im Sinne eines klassischen Personenzählers die absoluten Personenzahlen während eines Zeitintervalls angeben, sondern die Anzahl der Personen, die sich durchschnittlich innerhalb eines Zeitintervalls in den einzelnen Teilbereichen bewegt haben. Die Kommazahlen resultieren demnach aus der Mittelwertbildung, mit der die Anzahl der Personen berechnet wird, die sich innerhalb der einzelnen Zeitintervalle in den Teilbereichen bewegen. Im Kontext dieser Arbeit ist diese Zahlendarstellung unkritisch, da es für die Untersuchung der Nutzungsintensität bestimmter Bereiche nicht re-

[8] Während des Bearbeitungszeitraums der vorliegenden Arbeit war lediglich einer begrenzten Personenzahl der Zutritt zum Betriebsgelände gewährt. Dementsprechend niedrig fällt die Anzahl der erkannten Personen aus. Da das Funktionsprinzip der Laufweganalyse unberührt bleibt, ist die Anzahl der erkannten Personen in den Diagrammen zur Verbesserung der Anschaulichkeit um den Faktor 10 erhöht.

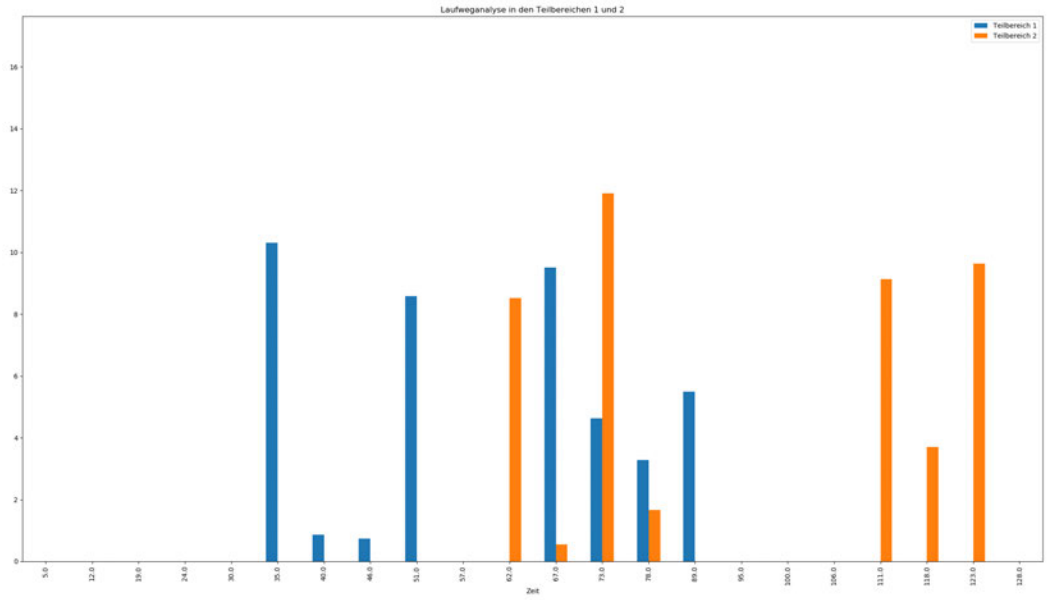


Abbildung 6.9.: Durchschnittliche Personenanzahl je Zeitintervall in den Teilbereichen 1 und 2. Teilbereich 1 - blauer Balken, Teilbereich 2 - orangener Balken

levant ist, ob es sich um ganze Zahlen oder eine Kommazahlen handelt. Eine Unterstützung der Einsatzplanung von Transportsystemen ist mit beiden Zahlenformaten gleichermaßen möglich.

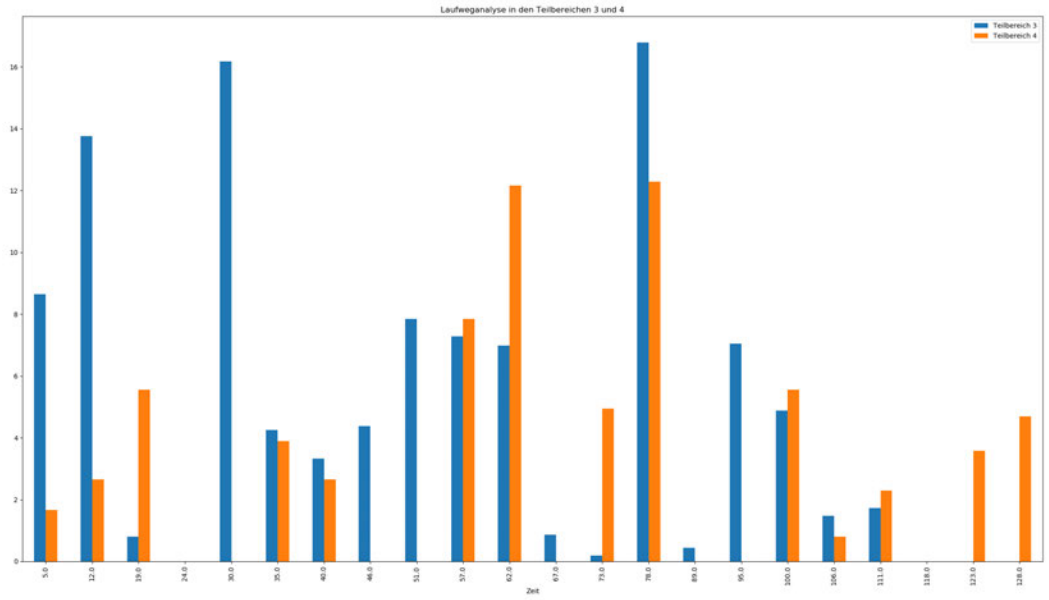


Abbildung 6.10.: Durchschnittliche Personenanzahl je Zeitintervall in den Teilbereichen 3 und 4. Teilbereich 3 - blauer Balken, Teilbereich 4 - orangener Balken

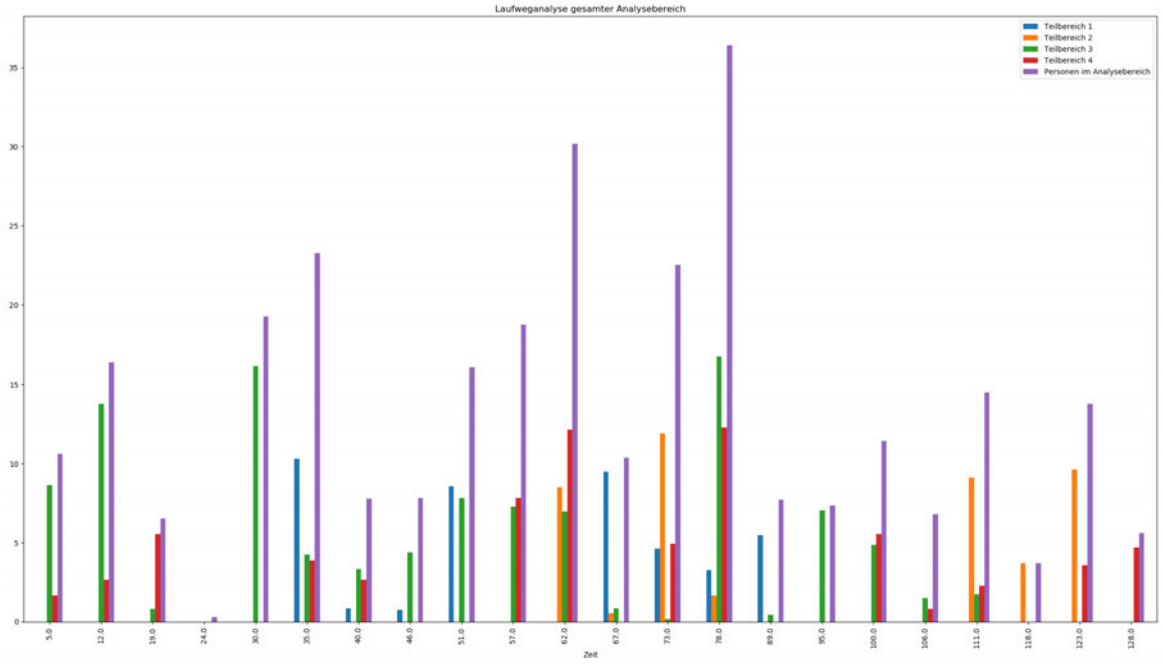


Abbildung 6.11.: Durchschnittliche Personenanzahl je Zeitintervall in den einzelnen Teilbereichen und dem gesamten Analysebereich. Teilbereich 1 - blauer Balken, Teilbereich 2 - orangener Balken, Teilbereich 3 - grüner Balken, Teilbereich 4 - roter Balken, gesamter Analysebereich - lilaner Balken

	Zeit	Teilbereich 1	Teilbereich 2	Teilbereich 3	Teilbereich 4	Personen im Analysebereich
1	5.0	0.000000	0.000000	8.641975	1.666667	10.617284
2	12.0	0.000000	0.000000	13.765432	2.654321	16.419753
3	19.0	0.000000	0.000000	0.802469	5.555556	6.543210
4	24.0	0.000000	0.000000	0.000000	0.000000	0.308642
5	30.0	0.000000	0.000000	16.172840	0.000000	19.259259
6	35.0	10.308642	0.000000	4.259259	3.888889	23.271605
7	40.0	0.864198	0.000000	3.333333	2.654321	7.777778
8	46.0	0.740741	0.000000	4.382716	0.000000	7.839506
9	51.0	8.580247	0.000000	7.839506	0.000000	16.111111
10	57.0	0.000000	0.000000	7.283951	7.839506	18.765432
11	62.0	0.000000	8.518519	6.975309	12.160494	30.185185
12	67.0	9.506173	0.555556	0.864198	0.000000	10.370370
13	73.0	4.629630	11.913580	0.185185	4.938272	22.530864
14	78.0	3.271605	1.666667	16.790123	12.283951	36.419753
15	89.0	5.493827	0.000000	0.432099	0.000000	7.716049
16	95.0	0.000000	0.000000	7.037037	0.000000	7.345679
17	100.0	0.000000	0.000000	4.876543	5.555556	11.419753
18	106.0	0.000000	0.000000	1.481481	0.802469	6.790123
19	111.0	0.000000	9.135802	1.728395	2.283951	14.506173
20	118.0	0.000000	3.703704	0.000000	0.000000	3.703704
21	123.0	0.000000	9.629630	0.000000	3.580247	13.765432
22	128.0	0.000000	0.000000	0.000000	4.691358	5.617284

Abbildung 6.12.: Daten zu den Balkendiagrammen in den Abbildungen 6.9, 6.10 und 6.11

7. Evaluation

Dieses Kapitel behandelt die Bewertung und Evaluation des realisierten Embedded Vision Systems. Die Bewertung und Evaluation orientieren sich an der Anforderungsanalyse in Kapitel 4 und nehmen Bezug auf den konzeptionellen Teil in Kapitels 5 sowie die in Kapitel 6 beschriebene Realisierung. Zunächst werden in Unterkapitel 7.1 die einzelnen Hauptbestandteile, also das Development-Board, der Datensatz und das Modell behandelt. Darauf folgend werden in Unterkapitel 7.2 das Gesamtsystem sowie die Laufweganalyse betrachtet.

7.1. Evaluation des Development-Boards, Datensatzes und Modells

In Rahmen dieses Unterkapitels erfolgt zunächst die Bewertung des Jetson Nanos. Anschließend folgen die Bewertungen des Datensatzes und des Modells. Die entsprechenden Beschreibungen innerhalb dieses Unterkapitels sind kurz gehalten und sollen hauptsächlich als eine Art Checkliste betrachtet werden.
Die Bewertung des Jetson Nanos erfolgt aus pragmatischen Gründen in tabellarischer Form (siehe Tabelle 7.1). Dazu sind neben der jeweiligen Anforderung die entsprechenden Daten des Jetson Nanos eingetragen. Bei den nicht quantifizierbaren Anforderungen findet sich jeweils ein Verweis zu dem Kapitel oder Abschnitt, in dem sich die entsprechenden Angaben finden.

Wie auch der Jetson Nano erfüllt der Datensatz die aufgestellten Anforderungen. Bei den Anforderungen A.4.3-01 (Datensatz für Objekterkennung oder -verfolgung) und A.4.3-02 (spezieller Datensatz für Personen oder Personengruppen) handelt es sich um Grundvoraussetzungen für die Berücksichtigung eines Datensatzes, womit diese Anforderungen folglich erfüllt sind. Die Anforderungen nach wenig verdeckten Personen (Anforderung A.4.3-03: Dichte Personengruppen in maximal 20 % der Bilder) wird ebenfalls erfüllt, da dichte Personengruppen in dem verwendeten Datensatz eine Ausnahme bilden und dies in nicht mehr als 20 % der Bilder vorkommt. Anforderung A.4.3-04 (Ground Truth Daten bzw. Bounding Boxes) wird ebenfalls als erfüllt betrachtet, da alle Bilder des PETS09-S2L1 vollständig annotiert sind und diese im Vergleich zu den eigens aufgezeichneten Bilder, für welche zunächst Ground

Tabelle 7.1.: Bewertung der an den Jetson Nano gestellten Anforderungen

ID	Anforderung	Jetson Nano	erfüllt
A.4.2-01	Spezielle KI-Hardware in Form eines Development-Boards	[Franklin, 2019]	ja
A.4.2-02	Verfügbar als Bestandteil eines Development-Kits	Kapitel 6.1.1 [Franklin, 2019]	ja
A.4.2-03	Eignung für den Einsatz neuronaler Netze	[Franklin, 2019]	ja
A.4.2-04	Verfügbarkeit einer kompatiblen Kamera	Raspberry Pi Kameramodul V2	ja
A.4.2-05	Geringer Energiebedarf (max. 10 W)	5 W / 10 W	ja
A.4.2-06	Geringe Versorgungsspannung (max. 5 VDC)	5 VDC	ja
A.4.2-07	Geringe Abmessungen (max. 100 mm x 100 mm, Länge x Breite)	80 mm x 100 mm (Länge x Breite)	ja
A.4.2-08	Niedriges Gewicht ($<$ 150 g)	138 g inkl. Kühlkörper	ja
A.4.2-09	Großer Arbeitsspeicher ($\geq$ 1 GB)	4 GB	ja
A.4.2-10	Hohe Rechen- bzw. Prozessorleistung ($\geq$ 0,5 TFLOPS)	0,5 TFLOPS	ja
A.4.2-11	Große Anzahl unterstützter / nutzbarer Frameworks	Kapitel 6.1.2 Anhang A.2	ja
A.4.2-12	Open Source / keine Bindung an kostenpflichtige, proprietäre Software	Anhang A.2 NVIDIA Download Center	ja
A.4.2-13	Investitionskosten unterhalb des Richtwertes von €500,00	Anhang A.2	ja

Truth Daten zu erzeugen waren, den größeren Anteil des gesamten Datensatzes ausmachen. Der Darstellungswinkel von Personen liegt bei dem PETS09 Datensatz bei ca. 30° und bei dem eigens aufgezeichneten Datensatz bei 50°, womit der Richtwert von 45° ± 15° (Anforderung A.4.3-05) eingehalten wird. Hinsichtlich der Anforderungen A.4.3-06 (Praxisnahe Personengröße), A.4.3-07 (Variation der Personengröße und -orientierung) und A.4.3-08 (Aufnahmeumfeld mit Bezug zu etwaigen Einsatzorten und wechselnden Lichtverhältnissen) kann auf die Abbildung 5.1 verwiesen werden. In dieser Abbildung sind ausgewählte Bilder des Datensatzes dargestellt, die als Nachweis für die vorab aufgeführten Anforderungen dienen. Mit einem gesamten Speicherbedarf von 125 MB wird der in Anforderung A.4.3-09 genannte Richtwert von 1 GB eingehalten und damit auch diese Anforderung erfüllt.
Das YOLOv3-Tiny Modell konnte ebenfalls alle gestellten Anforderungen erfüllen. Bei YOLOv3-Tiny handelt es sich um einen einstufigen Detektor, womit die Anforderung A.4.4-01 erfüllt wird. Darüber hinaus handelt es sich bei YOLO generell um ein speziell für die Objekterkennung eingesetztes Modell, wodurch auch die Anforderung A.4.4-04 erfüllt ist. Auf Grund seiner Architektur mit lediglich 19 Layern (Darknet-19 Backbone) erfüllt YOLOv3-Tiny auch Anforderung A.4.4-02, die einen Richtwert von 20 Layern vorgegeben hat. Mit und einer Parameteranzahl unter 10 Millionen wird auch Anforderung A.4.4-03 erfüllt. Basierend auf diesen Eigenschaften stellt YOLOv3-Tiny ein Modell dar, welches speziell für mobile und embedded Geräte wie den Jetson Nano geeignet ist (A.4.4-05). Die Anforderung nach einer Geschwindigkeit von 3 fps wird mit durchschnittlich 3,5 fps ebenfalls erfüllt (A.4.4-07). Die zuletzt gestellte Anforderung hinsichtlich des Supports (A.4.4-08) wird ebenfalls erfüllt.

7.2. Evaluation des Gesamtsystems und der Laufweganalyse

Innerhalb dieses Unterkapitels werden die Bewertung und Evaluation des Gesamtsystems sowie der Laufweganalyse behandelt. In Bezug auf das Gesamtsystem werden die Anforderungen aus dem Unterkapitel 4.1 herangezogen. Bei der Evaluation der Laufweganalyse werden zum einen Aspekte betrachtet, die mit der Personenerkennung im Zusammenhang stehen und zum anderen erfolgt eine Interpretation der Ergebnisse mit einem Bezug zu dem angenommenen beispielhaften Szenario.
Wie in Unterkapitel 6.3.1 beschrieben, ist die Position des Embedded Vision Systems mit einem mittleren Aufnahmewinkel von 50° so gewählt, dass dieser innerhalb der angegebenen Toleranz (siehe Anforderung A.4.1-01: Richtwert: 45°, ±15°) des geforderten Aufnahmewinkels liegt. Aus dieser Position wird eine relevante Fläche von ca. $142\,m^2$ und einer kürzeren Seitenlänge von ca. 9 m erfasst und die Anforderung A.4.1-02 (Fläche mindestens $100\,m^2$, keine Seite kürzer als 8 m) wie in Anforderung A.4.1-03 gefordert erfüllt.Überdies erfolgt ebenfalls in Unterkapitel 6.3.1 die Identifikation, Festlegung und Visualisierung der

Teilbereiche. Dazu sei hier auf die Abbildungen 6.5 und 6.6 verwiesen. Der Abbildung 6.6 (rechtes Bild) können darüber hinaus die Abmessungen der einzelnen Teilbereiche entnommen werden. Alle vier Teilbereichen haben eine Fläche von mindestens $20\,m^2$, wobei keine Seitenlänge kürzer als 3,5 m ist. Damit sind die Anforderungen entsprechend A.4.1-04 erfüllt.
Um die Problemstellung perspektivischer Verzerrungen zu berücksichtigen (Anforderung A.4.1-05), werden entsprechend den Schilderungen in Unterkapitel 6.3.3 die OpenCV-Funktionen *cv2.getPerspectiveTransform()* und *cv2.PerspectiveTransform()* eingesetzt. Darüber hinaus findet sich in diesem Unterkapitel mit dem Verweis auf das eingesetzte Modul *matplotlib.Path()* die Erläuterung, wie im Rahmen dieser Arbeit erfasst wird, in welchen Teilbereichen sich Personen befinden (Anforderung A.4.1-06). Die Anforderung nach zeitlichen Intervallen (A.4.1-07) wird durch die Verwendung eines Frame-Zählers umgesetzt (siehe Abbildung 6.8).

Im Nachfolgenden wird auf die Aspekte eingegangen, die mit der Laufweganalyse sowie der Personenerkennung zusammenhängen. Zunächst soll der Aspekt der Größe des Eingangsbildes in das YOLOv3-Tiny Modell betrachtet werden und wie sich dies für das realisierte Embedded Vision System auf die Geschwindigkeit des Modells auswirkt. Die Abbildungen 7.1 und 7.2 zeigen Einzelbilder, die aus drei unterschiedlichen Videosequenz stammen. Jede dieser Videosequenzen zeigt das Resultat der Personenerkennung durch das verwendete YOLOv3-Tiny Modell in Abhängigkeit von der Größe des Eingangsbildes, welches über den Input-Layer in das Modell eingegeben wird. Generell erhöht sich die Geschwindigkeit bei einer niedrigeren Auflösung des Eingangsbildes, während die Geschwindigkeit sinkt, wenn eine höhere Auflösung gewählt wird. Im vorliegenden Fall zeigt sich, dass sich die Geschwindigkeit des Modells bei Eingangsgrößen von 320 x 320 Pixeln, 416 x 416 Pixeln und 512 x 512 Pixeln nur geringfügig unterscheidet. Die entsprechenden Werte sind in den genannten Abbildungen jeweils in der oberen linken Bildecke aufgeführt und liegen bei 3,7 fps, 3,5 fps und 3,3 fps. Hieran zeigt sich, dass die Geschwindigkeit des Modells im vorliegenden Fall nicht wesentlich durch die Wahl der Eingangsgröße optimiert werden kann.

Hinsichtlich des Aspekts der angezeigten Klassenwahrscheinlichkeiten zeigt sich, dass diese bei einer geringeren Auflösung (320 x 320 Pixel) deutlich absinken, Personen also weniger sicher erkannt werden. Zudem steigt die Anzahl der fälschlicherweise erkannten Personen (False Positives), wodurch eine Verschlechterung der *Precision* eintritt. Im Vergleich dazu tritt bei einer Erhöhung der Auflösung von 416 x 416 Pixel auf 512 x 512 Pixel keine eindeutige Verbesserung auf. Vielmehr bewegen sich die ausgegebenen Werte in ähnlichen Größenordnungen.

Zur Evaluation der bei der Laufweganalyse erzielten Ergebnisse, ist eine manuelle Zählung

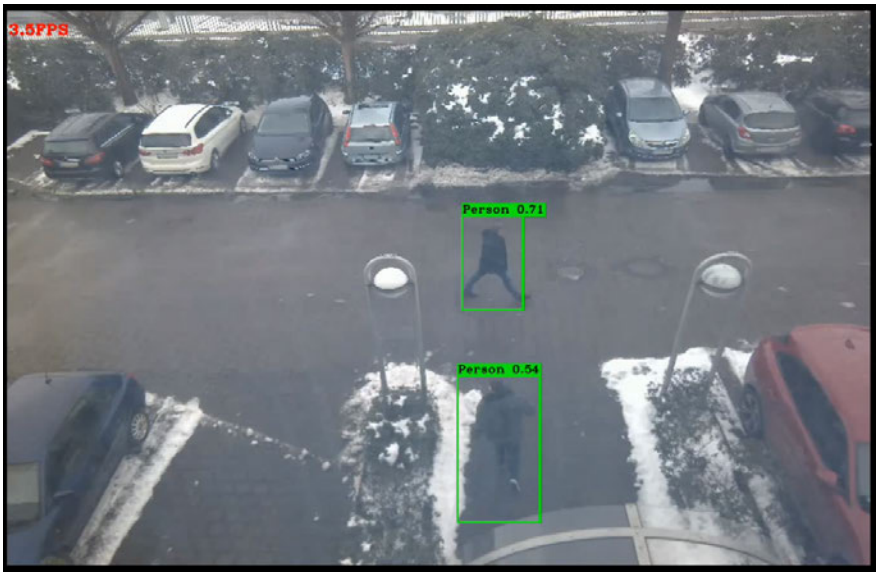


Abbildung 7.1.: Personenerkennung bei der für das Embedded Vision System genutzten Größe der Modelleingaben von 416 x 416 x 3 und einer Verarbeitungsgeschwindigkeit von 3,5 fps. Die Klassenwahrscheinlichkeiten liegen bei 71 % für die obere und 54 % für die untere Person.

Abbildung 7.2.: Personenerkennung bei einer Größe der Modelleingaben von 320 x 320 x 3 und einer Verarbeitungsgeschwindigkeit von 3,7 fps (links) sowie einer Modelleingabe von 512 x 512 x 3 und einer Verarbeitungsgeschwindigkeit von 3,3 fps (rechts). Die Klassenwahrscheinlichkeiten liegen im linken Bild (nur die True Positives) bei 52 % für die obere und 50 % für die untere Person. Im rechten Bild liegen die Werte bei 72 % für die obere und 53 % für die untere Person.

	Zeit	Teilbereich 1	Teilbereich 2	Teilbereich 3	Teilbereich 4	Personen im Analysebereich
1	5,0	0,00	0,00	10,00	0,80	11,46
2	12,0	0,00	0,00	11,43	3,14	14,96
3	19,0	0,00	0,00	3,43	5,43	10,62
4	24,0	0,00	0,00	0,00	0,00	0,00
5	30,0	0,00	0,00	15,33	0,00	16,66
6	35,0	8,00	0,00	4,60	3,60	20,60
7	40,0	1,00	0,00	6,00	3,40	10,40
8	46,0	1,33	0,00	2,00	0,00	4,33
9	51,0	7,40	0,00	8,33	0,00	15,73
10	57,0	0,00	0,00	8,33	4,80	13,13
11	62,0	0,00	7,20	7,40	6,60	25,78
12	67,0	6,40	2,00	1,20	0,00	9,60
13	73,0	7,17	9,00	0,00	5,00	21,84
14	78,0	4,80	3,20	15,20	13,40	38,00
15	89,0	4,36	0,00	0,90	0,90	6,16
16	95,0	1,00	0,00	7,33	0,00	8,33
17	100,0	0,00	0,00	6,80	5,00	11,80
18	106,0	0,00	0,00	2,30	1,00	8,66
19	111,0	0,00	6,60	3,80	0,80	11,80
20	118,0	0,00	0,00	0,00	0,00	0,00
21	123,0	0,00	3,00	0,00	1,80	4,80
22	128,0	0,00	0,00	0,00	5,80	5,80

Abbildung 7.3.: Ergebnis der manuell ermittelten Personenzahlen, die sich durchschnittlich je Zeitintervall in den einzelnen Teilbereichen sowie dem gesamten Analysebereich aufgehalten haben.

der in der aufgezeichneten Videosequenz gezeigten Personen erfolgt. Dazu ist ein schrittweiser Durchlauf der Videosequenz erfolgt, um jeweils die zeitliche Dauer zu erfassen, während derer sich die Personen in den einzelnen Teilbereichen sowie dem gesamten Analysebereich befunden haben. Unter Beachtung der einzelnen Zeitintervalle ist abschließend für jeden der vier Teilbereiche sowie den gesamten Analysebereich eine Berechnung erfolgt, um zu ermitteln, wie viele Personen sich durchschnittlich während der einzelnen Zeitintervalle in den einzelnen Bereichen respektive dem gesamten Analysebereich befunden haben. Das Ergebnis dieser Zählung und Berechnung ist als Auflistung in Abbildung 7.3 dargestellt. Im Vergleich[1] zu den während der Laufweganalyse aufgenommenen Daten (Abbildung 6.12) zeigt sich, dass das Embedded Vision System in der Regel mehr Personen in den einzelnen Teilbereichen sowie dem gesamten Analysebereich erkennt. In Tabelle 7.2 ist gegenübergestellt, welche Gesamtzahlen die manuelle Zählung und die Zählung des Embedded Vision Systems ergeben haben.

Die Anzahl der manuell gezählten Personen ist generell niedriger, da hier keine Zählfehler durch False Positives auftreten. In den oberen Teilbereichen 1 und 2 ist die von dem Embed-

[1] Die Zählergebnisse der manuellen Zählung und die durch das Embedded Vision System erzielten Zählergebnisse finden sich in einer gemeinsamen Darstellung in Anhang A.9. Darüber hinaus ist auch hier zu beachten, dass die vorliegenden Zählergebnisse zur besseren Veranschaulichung um den Faktor 10 erhöht sind.

Tabelle 7.2.: Vergleich der Zählergebnisse (Personenzahl) in manuell durchgeführter Form und automatisch durch das Embedded Vision System (TB - Teilbereich, AB - Analysebereich)

Zählung	**TB 1**	**TB 2**	**TB 3**	**TB 4**	**Ges. AB**
manuell	41,46	31	114,38	61,47	270,46
automatisch	43,39	45,13	106,85	70,65	297,3

ded Vision System ausgegebene Personenzahl tendenziell höher, da die unteren Begrenzungen der Bounding Boxes (diese gibt in der vorliegenden Arbeit die Position einer Person an) häufig in diesen Teilbereichen liegen, während sich die zugehörigen Personen noch nahe an den abgestellten Fahrzeugen befinden und damit noch außerhalb der Teilbereiche liegen. Die Ausnahme bildet der Teilbereich 3, bei dem die manuelle Zählung ein höheres Ergebnis geliefert hat. Dies liegt zum einen daran, dass es hier zu Personenüberdeckungen kommt und die verdeckten Personen dabei nicht erkannt werden. Zum anderen bewegen sich die Personen mehrfach am unteren Rand des Teilbereichs 3, wodurch jeweils die unteren Teile der Bounding Boxes außerhalb dieses Teilbereichs liegen. Prozentual betrachtet erkennt das System mehr Personen in den Teilbereichen 1 (4,66 %), 2 (45,58 %) und 4 (14,93 %) sowie dem gesamten Analysebereich (9,92 %) mehr Personen und in Teilbereich 3 (-6,58 %) weniger Personen. Im Schnitt liegt die prozentuale Differenz der durch das System mehr erkannten Personen bei 13,7 %.

In Bezug auf das angenommene beispielhafte Szenario kann eine weitere Evaluation und Interpretation der erzielten Ergebnisse anhand der Abbildung 7.4 erfolgen. Darin wird die Entwicklung der Laufwege der erkannten Personen dargestellt. Sobald eine Person innerhalb der Videosequenz erkannt wird, wird ein Punkt an der entsprechenden Stelle in das Videobild appliziert. Schrittweise ergibt sich so das dargestellte Muster. Je größer die Punktdichte, desto häufiger wird der entsprechende Bereich als Laufweg genutzt.

Es zeigt sich, dass der Bereich in Gebäudenähe, also die Teilbereiche 3 und 4 am stärksten durch Personen frequentiert sind. Für die Einsatzplanung von Transportsystem kann diese Erkenntnis dazu genutzt werden, die Teilbereiche 1 und 2, also die Bereiche vor den oberen Parkplatzbuchten, als Fahrwege zu priorisieren. Unter Einbeziehung des Balkendiagramms in Abbildung 6.9 kann für die priorisierten Teilbereiche ausgesagt werden, dass Transportfahrten speziell im ersten Viertel des Aufnahmezeitraums ausgeführt werden sollten.

Abbildung 7.4.: Entwicklung der Laufwegmarkierungen während der Laufweganalyse. Die hier nicht dargestellten Teilbereiche befinden sich

8. Fazit

8.1. Zusammenfassung

Aufgabe der vorliegenden Abschlussarbeit war es, ein Deep Learning basiertes Embedded Vision System zur Analyse der Laufwege von Personen anhand von Videosequenzen zu konzipieren und zu entwickeln. Durch diese Analyse kann festgestellt werden wie häufig einzelne Teilbereiche innerhalb eines bestimmten Aufnahmefeldes von Personen als Laufwege genutzt werden. Die Ergebnisse sollen die Einsatzplanung mobiler Transportsysteme unterstützen und damit die Behinderung von Personen bei Transportfahrten verhindern. Eine entsprechende Visualisierung sollte die Ergebnisse veranschaulichen.
Um ein für diese Aufgabe geeignetes System zu realisieren, mussten ein Development-Kit mitsamt Kamera, ein neuronales Netz bzw. Deep Learning Modell sowie ein Datensatz zum Training der Personenerkennung bestimmt werden. Dazu wurden im Rahmen einer Analyse die Anforderungen an diese Systembestandteile erarbeitet und basierend auf deren Relevanz eine Gewichtung vergeben. Darauf aufbauend wurde das Embedded Vision System konzipiert, realisiert und abschließend bewertet.
Im Rahmen der Konzeption wurde der Jetson Nano als Development-Kit bzw. Development-Board ausgewählt. Der Jetson Nano ist speziell für den Einsatz in Verbindung mit KI-Anwendung und dem Deep Learning konzipiert, bietet einen Arbeitsspeicher von 4 GB und eine Rechenleistung von 0,5 TFLOPS. Zudem ist der Jetson Nano auf Grund der großen Auswahl an nutzbaren Frameworks ein flexibel einsetzbares Development-Board. Als Kamera kam das Raspberry Pi Kameramodul V2 mit MIPI CSI-2 Schnittstelle zum Einsatz. Die für das Kameramodul erforderlichen Treiber sind Bestandteil des Software Development Kits, so das ein direkter Betrieb des Kameramoduls möglich war.
Bei dem eingesetzten Datensatz handelt es sich um den PETS09-S2L1 Datensatz, der 795 annotierte Bilder speziell für die Personenerkennung und -verfolgung enthält. Dieser Datensatz wurde um 235 am späteren Einsatzort des Embedded Vision Systems aufgenommene Bilder erweitert. Die erforderlichen Markierungen der dargestellten Personen mittels Bounding Boxes wurden mit dem Tool LabelIMG ergänzt. Für das Training des eingesetzten YOLOv3-Tiny Modells wurden vortrainierte YOLOv3-Tiny Gewichte genutzt. Das Training des Modells wurde auf der kostenlosen Plattform *Google Colaboratory* durchgeführt. Auf Google Colaboratory kam eine Tesla V100-SXM2-16GB Grafikkarte als GPU Hardware- Beschleu-

niger zum Einsatz. Das Training des Modells über 40 Epochen erreichte eine mean Average Precision von 88,566 %.
Das entwickelte System zur Laufweganalyse unterteilt einen Aufnahmebereich in mehrere Teilbereiche. Den Teilbereichen sind GPS-Koordinaten hinterlegt. Dadurch können die Ergebnisse der Laufweganalyse die Einsatzplanung der mobilen Transportsysteme direkt unterstützen, da Transportsysteme so genaue Ortsangaben der Teilbereiche erhalten können. Bei dem ausgewählten Einsatzort handelte es sich um einen verkehrsberuhigten Bereich, in dem sich Personen zwischen einzelnen Gebäuden bewegten und ein unregelmäßiger Verkehr von Transportfahrzeugen auftrat.
Die Grundlage der Laufweganalyse stellt die Personenerkennung durch das YOLOv3-Tiny Modell dar. Während der Ausführung des Modells werden die Positionsdaten erkannter Personen in eine Datentabelle geschrieben. Nach dem Ende der Personenerkennung werden die akquirierten Daten in eine Laufweganalyse-Software eingelesen und verarbeitet. Das Skript wandelt die Pixelpositionen in GPS-Koordinaten um und überprüft, ob sich die ermittelten Positionen innerhalb einer der Teilbereiche befinden. Auf diese Weise wird erfasst, wie häufig sich Personen in den unterschiedlichen Bereichen aufhalten. Die finale Auswertung gibt an, wie viele Personen sich durchschnittlich innerhalb bestimmter Zeitintervalle in den einzelnen Teilbereichen aufgehalten haben. Die so ermittelten Werte werden anschließend für die Visualisierung der Nutzungshäufigkeit genutzt.
Für den Test und die Evaluierung des realisierten Embedded Vision Systems wurde eine am Einsatzort aufgezeichnete Videosequenz verwendet. In diesem Zusammenhang haben sich auf Grund der pandemischen Umstände vor und während der ausgeführten Test einige Herausforderungen ergeben. Zum einen war es nicht möglich Videosequenzen mit größeren Personengruppen aufzuzeichnen und zum anderen konnten Videosequenzen lediglich in einem begrenzten Zeitraum erstellt werden.
Das Embedded Vision Systems konnte alle direkten Anforderungen an das Development-Board sowie den Datensatz erfüllen. Hinsichtlich der mit dem YOLOv3-Tiny Modell erzielten durchschnittlichen Verarbeitungsgeschwindigkeit von 3,5 fps hat sich ein für die Erfüllung der Aufgabe ausreichend schneller, aber immer noch niedrigerer Wert als ursprünglich angenommen ergeben. Für die Bewertung des durch das Embedded Vision System erreichte Zählergebnis für die durchschnittliche Anzahl der Personen in den einzelnen Teilbereichen sowie dem gesamten Analysebereich, wurde dieses mit dem Ergebnis einer manuellen Zählung und Berechnung verglichen. Dabei hat sich eine durchschnittliche prozentuale Differenz von 13,7 % ergeben. Die Auswertung der Sicherheit des Systems bei der Vorhersage von Klassenwahrscheinlichkeiten hat einen Wert von 59 % ergeben.
Abschließend soll an dieser Stelle auf aufgetretene Problematiken bei der Inbetriebnahme und Verwendung des Jetson Nanos eingegangen werden. Diese hat sich im Laufe der Arbeit als sehr herausfordern dargestellt. Speziell bei der Installation erforderlicher Software-Pakete und Bibliotheken sind häufig Kompatibilitätsprobleme aufgetreten, so dass diese entweder nicht eingesetzt werden konnten oder aber Aktualisierungen und Neuinstallationen anderer

Pakete durchgeführt werden mussten. Um einen nachvollziehbaren Zustand des Systems sicherzustellen, musste das System zwischenzeitlich mehrfach neu aufgesetzt werden. Über diese Probleme hinaus ist deutlich häufiger als erwartet ein Überauslastung des Arbeitsspeichers aufgetreten, so dass vorgesehene Erprobungen verschiedener Anwendungen mit der Meldung *out of memory* abgebrochen wurden oder erwartete Geschwindigkeiten hinsichtlich der fps nicht erreicht werden konnten.
Trotz der geschilderten Herausforderungen wurde im Rahmen letztendlich ein Embedded Vision System realisiert, mit dem die aufgestellte Zielsetzung erfüllt werden konnten.

8.2. Ausblick

Das in dieser Arbeit realisierte Embedded Vision System zur Analyse von Laufwegen hat erste vielversprechende Ergebnisse geliefert und bietet eine sehr gute Grundlage für zukünftige Lösungsansätze, Versuchsreihen und Erweiterungen. Eine Reihe von möglichen Ansatzpunkten zur Erweiterung des bisherigen Systems sind denkbar. In Bezug auf die Hardware kann die Mobilität und Robustheit des Systems verbessert werden, um einen autarken Einsatz im Außenbereich mittels Outdoor-Gehäuse und Akkupacks zu ermöglichen. Dazu müsste der Jetson Nano voraussichtlich mit reduzierter Leistung betrieben werden, was ebenfalls Modifikationen an der Software erfordern würde. Bei einem weiterhin ortsfesten Einsatz des Embedded Vision Systems könnte der Ansatz auf unterschiedliche Blickwinkel ausgeweitet werden. Dazu müssten weitere Datensätze akquiriert sowie der Personendetektor mit weiteren Trainingsdaten angelernt werden.
In Hinblick auf den gewählten Machine Learning Ansatz können noch weitergehende Auswertungen und Verbesserungen zur Steigerung der bisherigen Leistungsfähigkeit des Embedded Vision Systems durchgeführt werden Mögliche Ansatzpunkte beinhalten den Einsatz eines alternativen Frameworks wie beispielsweise *TensorFlow Lite* und weitere Experimente mit anderen Modellen. Hier böten sich unter anderem neuere YOLO-Versionen oder eine aktuelle Version des Modells *SSD Mobilenet* an. Darüber hinaus ließen sich Versuchsreihen mit alternativen Datensätzen vornehmen, um weitere Objektklassen mit dem System detektieren zu können. Rein funktional könnte die hier realisierte Ermittlung der durchschnittlich in den einzelnen Teilbereichen anwesenden Personen in einen klassischen Personenzähler umgewandelt werden, der Personen beispielsweise beim Überschreiten bestimmter Linien zählt. Dazu wäre es notwendig, die hier eingesetzte Objekterkennung zu einer Objektverfolgung zu erweitern. Weiterführende Aufgaben könnten die Laufwegprädiktion von Personen sein oder der Einsatz mehrerer Embedded Vision Systeme und deren Synchronisation zur Generierung von Tiefenbildinformationen.

Literaturverzeichnis

BaslerAG. Embedded vision für NVIDIA | basler, o. J. URL https://www.baslerweb.com/de/embedded-vision/embedded-vision-portfolio/embedded-vision-fuer-nvidia/.

Bosch. The 'brains' of your security system, o. J. URL https://www.boschsecurity.com/xc/en/solutions/video-systems/video-analytics/video-analytics-at-the-edge/.

Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. Ö'Reilly Media, Inc.", 2008. ISBN 978-0-596-51613-0.

Francois Chollet. *Deep Learning with Python*. Manning Publications Co., 2018a. ISBN 978-1-61729-443-3.

Francois Chollet. *Deep Learning mit Python und Keras das Praxis-Handbuch : vom Entwickler der Keras-Bibliothek*. mitp Verlags GmbH & Co. KG, 1. auflage edition, 2018b. ISBN 978-3-95845-838-3.

Abhinav Dadhich. *Practical Computer Vision: Extract Insightful Information from Images Using TensorFlow, Keras, and OpenCV*. Packt Publishing Ltd, 2018.

A. Forero and F. Calderon. Vehicle and pedestrian video-tracking with classification based on deep convolutional neural networks. In *2019 XXII Symposium on Image, Signal Processing and Artificial Vision (STSIVA)*, pages 1–5, April 2019. doi: 10.1109/STSIVA.2019.8730234.

Dustin Franklin. Jetson nano brings AI computing to everyone, 2019. URL https://developer.nvidia.com/blog/jetson-nano-ai-computing/.

Auerélien Gerón. *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*. O'Reilly, 2. auflage edition, 2019. ISBN 978-1-492-03264-9.

Auerélien Gerón. *Praxiseinstieg Machine Learning mit Scikit-Learn, Keras und TensorFlow*. O'Reilly, 2. auflage edition, 2020. ISBN 978-3-96009-124-0.

Ross Girshick. Fast r-CNN. 2015. URL http://arxiv.org/abs/1504.08083.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. pages 580–587, 2014. URL http://openaccess.thecvf.com/content_cvpr_2014/html/Girshick_Rich_Feature_Hierarchies_2014_CVPR_paper.html.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Google. Frequently asked questions, o. J.a. URL https://coral.ai/docs/edgetpu/faq/.

Google. Products, o. J.b. URL https://coral.ai/products/.

Google. Dev board, o. J.c. URL https://coral.ai/products/dev-board.

Google. About coral, o. J.d. URL https://coral.ai/about-coral/.

Ramyad Hadidi, Jiashen Cao, Yilun Xie, Bahar Asgari, Tushar Krishna, and Hyesoon Kim. Characterizing the deployment of deep neural networks on commercial edge devices. In *IEEE International Symposium on Workload Characterization (IISWC), Orlando, Florida*, 2019.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-CNN. 2018. URL http://arxiv.org/abs/1703.06870.

Sabir Hossain and Deok-jin Lee. Deep learning-based real-time multiple-object detection and tracking from aerial imagery via a flying robot with gpu-based embedded devices. *Sensors*, 19(15):3371, 2019.

Intel. Intel movidius neural compute stick, o. J. URL https://movidius.github.io/ncsdk/ncs.html.

L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu. A survey of deep learning-based object detection. *IEEE Access*, 7:128837–128868, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2939201.

Anirudh Koul, Siddha Ganju, and Meher Kasam. *Practical Deep Learning for Cloud, Mobile, and Edge*. O'Reilly Media, Inc., 2019. ISBN 978-1-492-03486-5.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, 2017. ISSN 0001-0782. URL https://doi.org/10.1145/3065386.

Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2:396–404, 1989.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. 9905:21–37, 2016. doi: 10.1007/978-3-319-46448-0_2. URL http://arxiv.org/abs/1512.02325.

Qi-Chao Mao, Hong-Mei Sun, Yan-Bo Liu, and Rui-Sheng Jia. Mini-YOLOv3: Real-time object detector for embedded applications. PP:1–1. doi: 10.1109/ACCESS.2019.2941547.

Abid Mordvintsev, Alexander und K. Geometric transformations of images - opencv-python tutorials 1 documentation, 2013. URL https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html.

MOT. MOT challenge - data, o. J.a. URL https://motchallenge.net/data/MOT15/.

MOT. MOT challenge - data, o. J.b. URL https://motchallenge.net/data/MOT17Det/.

MOT. MOT challenge - data, o. J.c. URL https://motchallenge.net/data/MOT20/.

MOT. MOT challenge - visualize, o. J.d. URL https://motchallenge.net/vis/MOT20-03.

S. Nagaraj, B. Muthiyan, S. Ravi, V. Menezes, K. Kapoor, and H. Jeon. Edge-based street object detection. In *2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 1–4, Aug 2017. doi: 10.1109/UIC-ATC.2017.8397675.

Neuralet. Smart social distancing, o. J. URL https://neuralet.com/smart-social-distancing/.

Michael A. Nielsen. Neural networks and deep learning. 2015. URL http://neuralnetworksanddeeplearning.com.

NVIDIA. Jetson nano: Deep learning inference benchmarks, 2019a. URL https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks. Library Catalog: developer.nvidia.com.

NVIDIA. Getting started with jetson nano developer kit, 2019b. URL https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit.

NVIDIA. Jetson developer kits, 2020a. URL https://developer.nvidia.com/EMBEDDED/jetson-developer-kits.

NVIDIA. Jetson modules, 2020b. URL https://developer.nvidia.com/embedded/Jetson-modules.

pythonlessons. pythonlessons/TensorFlow-2.x-YOLOv3, 2020. URL https://github.com/pythonlessons/TensorFlow-2.x-YOLOv3/tree/d1e4402aa32d879e03dbd0b9e339af6a315930a9.

Sebastian Raschka. rasbt/python-machine-learning-book-3rd-edition, 2019. URL https://github.com/rasbt/python-machine-learning-book-3rd-edition.

Vahid Raschka, Sebastian und Mirjalili. *Machine Learning mit Python und Scikit-learn und TensorFlow*. mitp Verlags GmbH & Co. KG, 2. auflage edition, 2018. ISBN 978-3-95845-733-1.

RaspberryPi. Camera module - raspberry pi documentation, o. J. URL https://www.raspberrypi.org/documentation/hardware/camera/.

Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. pages 7263–7271, 2016. URL http://openaccess.thecvf.com/content_cvpr_2017/html/Redmon_YOLO9000_Better_Faster_CVPR_2017_paper.html.

Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. 2018. URL http://arxiv.org/abs/1804.02767.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. 2016. URL http://arxiv.org/abs/1506.02640.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-CNN: Towards real-time object detection with region proposal networks. 39(6):1137–1149, 2017. ISSN 1939-3539. doi: 10.1109/TPAMI.2016.2577031. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union. June 2019.

W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016. ISSN 2327-4662. doi: 10.1109/JIOT.2016.2579198.

Koustubh Sinhal and Ankit Sachan. Choosing a deep learning framework: Tensorflow or pytorch?, 2018. URL https://cv-tricks.com/deep-learning-2/tensorflow-or-pytorch/. Section: Deep Learning.

Herbert Süße and Erik Rodner. *Bildverarbeitung und Objekterkennung: Computer Vision in Industrie und Medizin*. Springer Fachmedien Wiesbaden, 2014. ISBN 978-3-8348-2606-0.

Andre Tao, Jon Barker, and Sriya Sarathy. DetectNet: Deep neural network for object detection in DIGITS, 2016. URL https://developer.nvidia.com/blog/detectnet-deep-neural-network-object-detection-digits/.

TheImagingSource. NVIDIA jetson nano development kits, o. J. URL https://www.theimagingsource.de/embedded-vision/development-kits/nvidia-jetson-nano/.

VisualTrackerBenchmark. Visual tracker benchmark, o. J. URL http://cvlab.hanyang.ac.kr/tracker_benchmark/datasets.html.

Sebastian Weiss. Jetson nano and google coral edge TPU - a comparison, 2019. URL https://3dvisionlabs.com/2019/09/18/jetson-nano-and-google-coral-edge-tpu-a-comparison/.

Shifeng Zhang, Yiliang Xie, Jun Wan, Hansheng Xia, Stan Z. Li, and Guodong Guo. Widerperson: A diverse dataset for dense pedestrian detection in the wild. *IEEE Transactions on Multimedia (TMM)*, 2019.

]

A. Anhang

Der Anhang zur Arbeit befindet sich auf CD und kann beim Erstgutachter Prof. Dr.-Ing. Dipl.-Kfm. Jörg Dahlkemper eingesehen werden.

Darüber hinaus finden sich einige dieser Anhänge in den nachfolgenden Abschnitten.

A.1. Leistungsvergleich zwischen dem Jetson Nano und dem Google Coral Dev Board

Model	Application	Framework	NVIDIA Jetson Nano	Raspberry Pi 3	Raspberry Pi 3 + Intel Neural Compute Stick 2	Google Edge TPU Dev Board
ResNet-50 (224×224)	Classification	TensorFlow	36 FPS	1.4 FPS	16 FPS	DNR
MobileNet-v2 (300×300)	Classification	TensorFlow	64 FPS	2.5 FPS	30 FPS	130 FPS
SSD ResNet-18 (960×544)	Object Detection	TensorFlow	5 FPS	DNR	DNR	DNR
SSD ResNet-18 (480×272)	Object Detection	TensorFlow	16 FPS	DNR	DNR	DNR
SSD ResNet-18 (300×300)	Object Detection	TensorFlow	18 FPS	DNR	DNR	DNR
SSD Mobilenet-V2 (960×544)	Object Detection	TensorFlow	8 FPS	DNR	1.8 FPS	DNR
SSD Mobilenet-V2 (480×272)	Object Detection	TensorFlow	27 FPS	DNR	7 FPS	DNR
SSD Mobilenet-V2 (300×300)	Object Detection	TensorFlow	39 FPS	1 FPS	11 FPS	48 FPS
Inception V4 (299×299)	Classification	PyTorch	11 FPS	DNR	DNR	9 FPS
Tiny YOLO V3 (416×416)	Object Detection	Darknet	25 FPS	0.5 FPS	DNR	DNR
OpenPose (256×256)	Pose Estimation	Caffe	14 FPS	DNR	5 FPS	DNR
VGG-19 (224×224)	Classification	MXNet	10 FPS	0.5 FPS	5 FPS	DNR
Super Resolution (481×321)	Image Processing	PyTorch	15 FPS	DNR	0.6 FPS	DNR
Unet (1x512x512)	Segmentation	Caffe	18 FPS	DNR	5 FPS	DNR

Table 1. Inference performance results from Jetson Nano, Raspberry Pi 3, Intel Neural Compute Stick 2, and Google Edge TPU Coral Dev Board

Der Bildausschnitt findet sich unter https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks. (Abruf am 24.05.2020)

A.2. Vorauswahl Development-Kit

Hinsichtlich der Anforderungs-Nummern in den nachfolgenden Tabellen ist zu beachten, dass diese nicht vollständig mit den Anforderungs-IDs der vorangegangenen Kapitel übereinstimmen.

Hinweise	
[/]	Recherche auf Grund eines negativen Recherche-Prozesses vorzeitig beendet.
[X]	Recherche beendet, da dieses Development-Board eine odere mehrere A-Kriterien nicht erfüllt hat.
Rot markiert	Diese Dev.-Boards / Dev.-Kits haben mindestens eine Anforderung der Kategorie A nicht erfüllt. Kenndaten und Eigenschaften zu den Anforderungen der Kategorien B und C sind aus diesem Grund nicht vollständig aufgeführt.

HW-Vergleich - Gesamtübersicht Vorauswahl							
Anforder-ungs-Nr. (Afo)	Kriterium	NVIDIA					
		Jetson Nano	Jetson TX2 4GB	Jetson TX2	Jetson TX2i	AGX Xavier 8GB	AGX Xavier
Afo.A01	Spezielle AI- bzw. ML-HW in Form eines Development-Boards [*1]	√					
Afo.A02	Eignung des HW-Boards für den Einsatz Neuronaler Netze	√					
Afo.A03	HW-Board in Verbindung mit einem Development-Kit erhältlich	√	X	√	X	X	√
Afo.A04	Verfügbarkeit einer kompatiblen / unterstützten Kamera	√					
Afo.A05	Beschaffungskosten im Rahmen der HAW-Budgetvorgabe (Richtwert 500,00 €)	√ ca. 109 €	√ $249	√ 419,00 €	X $749; 730 €	X ab ca. 729,00 €	
Afo.B01	Geringer Energiebedarf	5 - 10 W	7,5 - 15 W		10 - 20 W	10 - 20 W	
Afo.B02	Geringe Versorgungsspannung	5 V DC (typical)	9 - 19,6 V DC	5,5 - 19,6 V DC	9 - 19,6 V DC	[X]	[X]
Afo.B03	Großer Arbeitsspeicher	4 GB, 64-Bit-LPDDR4 (16 GB eMMC 5.1) (theoretische Bandbreite bei 1600 MHz ist 25 GB/s)	4 GB, 128-Bit-LPDDR4	8 GB, 128-Bit-LPDDR4		8 GB,256-bit-LPDDR4x 1333 MHz - 85GB/s	16 GB,256-bit-LPDDR4x 2133 MHz - 137GB/s
Afo.B04	Umfangreicher Nutzer-Support / Anwender-Freundlichkeit	Z. B.: Jeston Download Center, Beispielprojekte, Tutorials, Jetson-Forum, Jetson-Wiki, Software Development Kit (SDK), Intuitive Suche und Navigation durch die angebotenen Support-Möglichkeiten;					

HW-Vergleich - Gesamtübersicht Vorauswahl							
Afo.B05	Hohe Rechenleistung (Leistungsfähige(r) Prozessor(en) (CPU, GPU, FPU, ASIC, TPU))	0,5 TFLOPS (FP16) *(512 GFLOPs)* CPU: Quad-core ARM Cortex-57 (@1,43 GHz) GPU: NVIDIA Maxwell-Architektur mit 128 NVIDIA CUDA-Recheneinheiten	1,33 TFLOPS CPU: ARMv8 (64-bit) Multi-Processor CPU Complex - ARM Cortex-A57 MPCore (quad-core) (max. Operating Frequency per Core: 2 GHz) - Denver 2 (dual-core) (max. Operating Frequency per Core: 2 GHz) GPU: NVIDIA Pascal-Architektur mit 256 NVIDIA CUDA-Recheneinheiten		1,26 TFLOPS CPU: ARMv8 (64-bit) Multi-Processor CPU Complex – ARM Cortex-A57 MPCore (quad-core) (max. Operating Frequency per Core: 2 GHz) – Denver 2 (dual-core) (max. Operating Frequency per Core: 2 GHz) GPU: NVIDIA Pascal-Architektur mit 256 NVIDIA CUDA-Recheneinheiten	5,5 TFLOPS (FP16) 11,1 TFLOPS (INT8) GPU: NVIDIA Volta Architektur mit 384 NVIDIA CUDA-Recheneinheiten und 48 Tensor-Recheneinheiten Zusätzlicher DLA (Deep Learning Accelerator): 4,1 TFLOPS (FP16) 8,2 TOPS (INT8)	11 TFLOPS (FP16) 22 TFLOPS (INT8) GPU: NVIDIA Volta Architektur mit 512 NVIDIA CUDA-Recheneinheiten und 64 Tensor-Recheneinheiten Zusätzlicher DLA (Deep Learning Accelerator): 5 TFLOPS (FP16) 10 TOPS (INT8)
Afo.B06	Unterstützte Betriebssysteme: Windows und / oder Linux	Windows, Linux	[x]	Linux	[x]	[x]	[x]
Afo.B07	Geringe Abmessungen	69,6 mm x 45 mm (Modul) 100 mm x 80 mm (Carrier Board)	87 mm x 50 mm (Modul) 165,1 mm x 170,18 mm (Carrier Board)			100 mm x 87 mm	
Afo.B08	Geringes Gewicht	0,138 kg (Dev.-Board mit Kühlkörper)	[X]	0,085 kg (Modul) 0,041 kg (Carrier Board -> z. B. "Orbitty Carrier) 0,126 kg (Dev.-Kit) 0,195 kg (Dev.-Kit + Kühlkörper vom Orbitty Carrier)	[x]	[x]	[x]
Afo.B09	Hohe Anzahl unterstützter / nutzbarer DL-Frameworks (Libraries)	Z. B. TensorFlow, PyTorch, Caffe/Caffe2, Keras, MXNet, Darknet	[x]	Z. B. TensorFlow, PyTorch, Caffe/Caffe2, Keras, MXNet	[x]	[x]	[x]
Afo.B10	Open Source (bzgl. SW, Frameworks, Libraries, Dev.- + Toolkits, APIs, etc.)	√	[x]	√	[x]	[x]	[x]

Afo.C01	Einrichtung bzw. Erstinbetriebname mit Windows möglich *(pers. Präferenz)*	J	[x]	N (Ubuntu Linux x64 v16.04 or v18.04 Host PC)	[x]	[x]	[x]
Afo.C02	Dev.-Kit nach der Einrichtung bzw. Erstinbetriebname ohne zusätzlichen Host-PC nutzbar (j/n)	J	[X]	[/]	[X]	[X]	[X]

HW-Vergleich - Gesamtübersicht Vorauswahl					
Anforder-ungs-Nr. (Afo)	Google	Intrinsyc	SolidRun	Huawei	Qualcomm
	Coral Dev Board	Open-Q 605 SBC Development Kit	HummingBoard Pulse -Quad 1.8GHz i.MX8M Mini	Atlas 200 Developer Kit	DragonBoard 820c Development Board
Afo.A01	√	√	√	√	√
Afo.A02	√	√	√	√	[/]
Afo.A03	√	√	√	√	√
Afo.A04	√	√	√	√	[/]
Afo.A05	√ ca. 150 €	√ ca. $429	√ ca. $242	X $840	√ ca. $414
Afo.B01	2 W (bei 4 TOPS) + max. 0,65 W (Kühlkörper mit Lüfter)	4,85 W (SoM; Durchschnitt, wenn alle Kerne u. Kühlung aktiv sind) 1 W (bei 2,1 TFLOPS)	0,7 W (bei 16.8 TOPS) 1 W (bei 24 TOPS) (nur der AI-Accelerator)	11 W (nur das Modul) 20 W (Dev. Kit)	[X]
Afo.B02	5 V DC	5 - 15 V DC (12 V DC Adapter im Dev. Kit)	12 V DC	3,5 - 4,5 V DC (Module) 5 - 28 V DC (Dev. Kit; 12 V recommended)	[X]
Afo.B03	1GB LPDDR4, 4-Channel 32-Bit (8 GB eMMC) (max. 1600 MHz)	4GB LPDDR4X SDRAM (32 GB eMMC)	2GB LPDDR4 RAM (8 GB eMMC)	8 GB/4 GB,128-bit-LPDDR4x; 3200 Mbit/s (interface rate)	LPDDR4 SDRAM
Afo.B04	Z. B.: Download-Center; Beispielprojekte; Get startet Guides; Intuitive Suche und Navigation durch die angebotenen Support-Möglichkeiten; Verweis auf vorhandene (geeignete) Datensätze	Z. B.: Zugang zur vollständigen Dokumentation mit dem Erwerb des Dev.-Kits; Basic Dev.-Kit Support (im Dev.-Kit enthalten); Weitere SDKs (z. B. Connected Camera SDK und Neural Processing SDK); Umfangreicheren Support (inkl. Doku. + Downloads) nach der Registrierung des Dev.-Boards	Z. B.: Documentation-Downloads; Get startet Guide; Weiterer Support über die Website des AI-Accelerator Herstellers (z. B. Software Dev.-Kits)	MindStudio SW Environment mit Simulations Umgebung; Get Startet Guide; Developer Community;	Download-Center, Beispielprojekte; Qualcomm-Foren, Qualcomm Developer Network

HW-Vergleich - Gesamtübersicht Vorauswahl					
Afo.B05	TPU: 4 TOPS @ 2W GPU: 32 GFLOPs 32-bit or 64 GFLOPs 16-bit GPU: Vivante GC7000Lite (32 GFLOPs 32-bit or 64 GFLOPs 16-bit) Edge TPU SoM: Edge TPU Coprocessor (von Google designter ASIC, dient als ML-Accelerator); Verbindung zum iMX8M SoC -> Quad-Core Cortex-A53 (support über 64-bit Armv8-A architecture mit max. Frequenz 1,5 GHz) + Cortex-M4F) Weitere: VPU	2,1 TFLOPS @ 1W CPU: QCS605 Kyro 300: 64-bit octa-core (2x Gold (2.5GHz) + 6x Silver (1.7GHz)) GPU: Qualcomm Adreno 615 (mit 64-bit Addressierung bis zu 780 MHz) DSP: AI Engine/'Qualcomm Hexagon 685 ("dual hexagon vector extension for running DNN models and advanced") ISP: Dual 14-bit Qualcomm Spectra 270 (unterstützt bis zu dual 16MP sensoren)	24 TOPS/W; - bis 16,8 TOPS @ 300 MHz - 16,8 TOPS @ 0,7W (nur das SoM) CPU: 4 x Arm Cortex A53 + M4 (1.8 GHz) (NXP i.MX8M Mini) GPU: GC NanoUltra 3D + GC320 2D Zusatz: Gyrfalcon Technology Lightspeeur® 2803 AI Accelerator	8 TFLOPS (FP16) 16 TOPS (INT8) Zwei DaVinci AI cores CPU: 8-core A55, max. 1.6 GHz	[X] CPU: Snapdragon 820E 64-bit kryo quad core CPU (2.35GHz) GPU: Qualcomm Adreno 530 DSP: Qualcomm Hexagon 68
Afo.B06	Linux, (Mac)	Android	Linux	[X]	[X]
Afo.B07	40 mm x 48 mm (SoM) 85 mm x 56 mm (Carrier Board) 88,1 mm x 59,9 mm (Carrier Board + Anschlüsse)	68 mm x 50 mm	102 mm x 69 mm	52,6 mm x 38,5 mm (Modul) 137,8 mm x 93 mm x 32,9 mm	[X]
Afo.B08	0,013 kg (SoM)	0,018 kg (Single Board computer)	0,45 kg	0,03 kg (Module) 0,234 kg (Dev. Kit)	[X]
Afo.B09	TensorFlow Light (only) (Modelle müssen 8-bit quantisiert sein und speziell für die Edge TPU kompiliert werden)	Tensorflow, Caffe/Caffe2, ONNX	TensorFlow, Caffe, PyTorch	[X]	[X]
Afo.B10	V	O	V	[X]	[X]

Afo.C01	N (Linux or Mac Host PC)	[/]	[/]	[X]	[X]
Afo.C02	[/]	On-Device Machine Learning	[/]	[X]	[X]

A.3. Bewertung ausgewählter Development-Boards in Bezug auf Anforderungen der Kategorie I

Die nachfolgenden Bewertungen beruhen auf den Angaben in Anhang A.2. Hinsichtlich der nachfolgend aufgeführten Anforderungs-Nummern ist zu beachten, dass diese nicht vollständig mit den Anforderungs-IDs der vorangegangenen Kapitel übereinstimmen. Die hier und in Anhang A.2 verwendeten Anforderungs-Nummern stimmen jedoch überein.

Anforderungen an das Hardware Board

Afo.A01	Spezielle AI- bzw. ML-Hardware in Form eines Development-Boards
Afo.A02	Eignung für den Einsatz neuronaler Netze
Afo.A03	Verfügbarkeit in Verbindung mit einem Development-Kit
Afo.A04	Verfügbarkeit einer kompatiblen / unterstützten Kamera
Afo.A05	Beschaffungskosten im Rahmen der HAW-Budgetvorgabe (Richtwert 500,00 €)

Bewertung und Resultat:

Je höher die Summe der Bewertungspunkte, desto besser ist das Resultat des Development-Boards.

++	Zwei Bewertungspunkte werden addiert
+	Ein Bewertungspunkt wird addiert
o	Kein Bewertungspunkt wird addiert oder abgezogen
-	Ein halber Bewertungspunkt wird abgezogen
--	Ein Bewertungspunkt wird abgezogen

Anforderungs-Nr.	**Development-Board**				
	Jetson Nano	Jetson TX2	Google Coral Dev Board	Open-Q 605 SBC Dev. Kit	HummingBoard Pulse
Afo.A01	++	++	++	+	++
Afo.A02	++	++	++	++	++
Afo.A03	++	++	++	++	+
Afo.A04	++	++	++	++	++
Afo.A05	++	--	++	--	o
Σ Afo.A	**10**	**7**	**10**	**6**	**7**

A.4. Vergleich von Datensätzen im Rahmen einer Voruntersuchung

Datensatz	Klasse(n) (sinngemäß)	Kameraperspektive, Aufnahmeumfeld	Anwendungszweck (vordergründig)	Vorverarbeitung	Speicher-bedarf	Erste Bewertung
MOT15	Personen	- Erhöhte Position, Blickwinkel ca. 30° - Outdoor, verkehrsberuhigter Bereich	Objekt-Erkennung und -Tracking	- Ground Truth - Bounding Boxes - Annotations	1,8 GB	+ Speicherbedarf: Sehr gering; problemlos + Perspektive: Den Vorstellungen annähernd vollständig entsprechend (geringfügig zu niedrig) + Personendichte: Den Vorstellungen entsprechend + Vorverarbeitung: Ok
MOT17DET	Menschen(mengen)	- Erhöhte Position, Blickwinkel ca. 40° - Fußgängerpassage	Objekt-Erkennung und -Tracking	- Ground Truth - Bounding Boxes - Annotations	1,9 GB	+ Speicherbedarf: Gering bis mäßig; problemlos + Vorverarbeitung: Ok + Perspektive: Den Vorstellungen tendenziell entsprechend + Vorverarbeitung: Ok
(Crowd Counting) MALL Dataset	Pedestrian (hier im Sinne von Passanten)	- Erhöhte Position, Blickwinkel grob $30° < \alpha < 60°$ - Indoor, Einkaufspassage	Object Counting	- Ground Truth - Label	1,8 GB	+ Speicherbedarf: Sehr gering; problemlos + Perspektive: Den Vorstellungen annähernd vollständig entsprechend (geringfügig zu hoch) + Personendichte: Den Vorstellungen entsprechend + Vorverarbeitung: Ok Sonstiges: + Support-Information bei kaggle.com
Collective Motion Database	Menschen(mengen)	- erhöhte Position, sehr unterschiedliche Winkel - Straßenverkehr - öffentliche Plätze - Flughafen	Nicht explizit benannt	- Ground Truth - Label	4,3 GB	+ Speicherbedarf: Gering bis mäßig; problemlos + Vorverarbeitung: Ok o Perspektive: Zwar eine erhöhte Position, aber tendenziell zu hoch o Anwendungszweck: Nicht explizit benannt - Personendichte: Fast ausschließlich deutlich zu hoch Sonstiges: + Begleitmaterial auf github.com verfügbar o Enthält Aufnahmeumfelder, die weniger relevant sind - Wenig Informationen zum Datensatz vor dem Download verfügbar - Unübersichtliche Menge an Videodaten

Data-Driven Crowd Dataset	Menschen(mengen)	- Augenhöhe und erhöhte Position, sehr unterschiedliche Winkel - Straßenverkehr - öffentliche Plätze - Veranstaltungsgelände	- Tracking von Einzelpersonen in Menschenmengen - Untersuchen von Bewegungsmustern in Menschenmengen	Information vor dem Download nicht verfügbar.	11 GB	+ Anwendungszweck: Passend o Speicherbedarf: Verhältnismäßig groß o Perspektive: Entspricht teilweise den Vorstellungen - Personendichte: Großer Anteil an Videosequenzen mit zu hoher Personendichte Sonstiges: o Enthält Aufnahmeumfelder, die weniger relevant sind
MOT20DET	Menschen(mengen)	- Erhöhte Position, Blickwinkel grob 30° < α < 70° - Bahnhofshalle - Stadionausgang - öffentlicher Platz	Objekt-Erkennung und -Tracking	- Ground Truth - Bounding Boxes - Annotations	4,7 GB	+ Speicherbedarf: Gering bis mäßig; problemlos + Vorverarbeitung: Ok + Perspektive: Den Vorstellungen tendenziell entsprechend (teilweise geringfügig zu hoch, teilweise eindeutig zu hoch) + Vorverarbeitung: Ok + Anwendungszweck: Passend o Personendichte: In der Bahnhofshalle den Vorstellungen entsprechend; auf dem öffentlichen Platz deutlich zu hoch Sonstiges: - Zwei Videosequenzen sind bei Nacht aufgenommen, wobei der Aufnahmebereich in einer der
MOT20	Menschen(mengen)	- Erhöhte Position, Blickwinkel grob 30° < α < 70° - Indoor: Bahnhofshalle - Outdoor: Stadion-ausgang, öffentlicher Platz	Objekt-Erkennung und -Tracking	- Ground Truth - Bounding Boxes - Annotations	4,7 GB	+ Speicherbedarf: Gering bis mäßig; problemlos + Vorverarbeitung: Ok + Perspektive: Den Vorstellungen tendenziell entsprechend (teilweise geringfügig zu hoch, teilweise eindeutig zu hoch) + Vorverarbeitung: Ok + Anwendungszweck: Passend o Personendichte: In der Bahnhofshalle den Vorstellungen entsprechend; auf dem öffentlichen Platz deutlich zu hoch Sonstiges: - Zwei Videosequenzen sind bei Nacht aufgenommen, wobei der Aufnahmebereich in einer der

SAIVT-QUT Crowd Counting Database (PETS2006 und PETS2009)	Menschen(mengen)	- Erhöhte Position, Blickwinkel grob 45° < α < 70° - Indoor: Empfangs-bereich, Ein- und Ausgangsbereich - Bahnstation - Outdoor: Gehweg, Straße	Crowd Counting	- Ground Truth - wenige Annotations	< 1 GB	+ Speicherbedarf: Sehr gering; problemlos + Perspektive: Den Vorstellungen annähernd vollständig entsprechend (Aufnahmebereich in einer Videosequenz etwas klein) + Personendichte: Ok o Vorverarbeitung: Ok o Anwendungszweck: Ok Sonstiges: + Videosequenzen innerhalb eines Universitätsgeländes
SAIVT-SoftBio database (multi-camera surveillance dataset)	Personen	- Erhöhte Position, Blickwinkel grob 45° < α < 80° - Indoor: Ein- und Ausgangsbereiche, Snackbar mit Tisch-gruppe, PC-	Person recognition + re-identification + detection	- unklar	2,14 GB	+ Speicherbedarf: Sehr gering; problemlos + Perspektive: Den Vorstellungen annähernd vollständig entsprechend o Vorverarbeitung: unklar Sonstiges: + Aufnahmeumfeld Universität
SAIVT-BuildingMoni tor-ing Dataset	Personen(gruppen)	- Erhöhte Position, Blickwinkel grob 45° < α < 80° - Universitätscampus Gebäude	Crowd Counting	- Annotations (teilweise)	> 16 GB	+ Perspektive: Den Vorstellungen annähernd vollständig entsprechend o Vorverarbeitung: Nur teilweise - Speicherbedarf: Deutlich zu groß, bei vollständigem Download Sonstiges: + Aufnahmeumfeld Universität

CVPR 2020 MOTS Challenge Datensatz	Menschen(mengen)	- Augenhöhe - Aufnahmen statisch und von fahrender Plattform - Indoor: Einkaufscenter - Outdoor: Einkaufsstraße,	Objekt Tracking	- Annotations	4,7 GB	+ Speicherbedarf: Gering bis mäßig; problemlos + Vorverarbeitung: Ok + Anwendungszweck: Passend o Personendichte: / - Perspektive: Zu niedrig; Aufnahmen von fahrender Plattform nicht notwendig Sonstiges: o Enthält Aufnahmeumfelder, die weniger relevant sind
Train Station Dataset	Personen(gruppen)	- (leicht) erhöhte Position; Blickwinkel grob $20° < \alpha < 45°$ - Bahnsteig	Multi-Person Tracking	- Ground Truth - Annotations - Label	> 8,1 GB	+ Vorverarbeitung: Ok + Anwendungszweck: Passend o Speicherbedarf: Tendenz zu "hoch" o Perspektive: Tendenziell zu niedrig Sonstiges: - Zu großer Fokus auf die Aktivitäten der aufgenommenen Personen - Einziges Aufnahmeumfeld weniger relevant
WiderPerson: A Diverse Dataset for Dense Pedestrian Detection in the Wild	Personen(gruppen), Menschen(mengen)	- I. d. R. Augenhöhe; teilweise erhöhte Positionen - diverse unterschiedliche Umfelder	Fußgänger / Personen Detektion	- Annotations - Label	< 1 GB	+ Speicherbedarf: Sehr gering; problemlos o Perspektive: Tendenziell zu niedrig o Personendichte: Teilweise passend - Perspektive: Tendenziell zu niedrig Sonstiges: + Trainings-, Test- und Validation-Subsets sowie Evaluation Codes o Enthält Aufnahmeumfelder, die weniger relevant sind
Visual Tracker Benchmark	Personen	- Erhöhte Position, Blickwinkel grob $30° < \alpha < 80°$ - Indoor: Einkaufszentrum - Outdoor: Gehweg, Straße, Bahnstation	Personen Tracking	- Bounding Box	< 100 MB	+ Vorverarbeitung: Ok + Anwendungszweck: Passend o Speicherbedarf: Sehr gering o Perspektive: Teilweise entsprechend den Vorstellungen, teilweise zu hoch o Personendichte: Tendenziell zu gering, zu Teilen den Vorstellungen entsprechend Sonstiges: o Enthält Aufnahmeumfelder, die weniger relevant sind

A.5. Original Dateiformat der Ground Truth Daten des PETS09-S2L1 Datensatzes

File Format

Please submit your results as a *single* ***.zip*** *file*. The results for each sequence must be stored in a separate .txt file in the archive's root folder. The file name must be exactly like the sequence name (case sensitive).

The file format should be the same as the ground truth file, which is a CSV text-file containing one object instance per line. Each line must contain 10 values:

```
<frame>, <id>, <bb_left>, <bb_top>, <bb_width>, <bb_height>, <conf>, <x>, <y>, <z>
```

The `conf` value contains the detection confidence in the `det.txt` files. For the ground truth, it acts as a flag whether the entry is to be considered. A value of 0 means that this particular instance is ignored in the evaluation, while any other value can be used to mark it as active. For submitted results, *all* lines in the .txt file are considered. The world coordinates `x,y,z` are ignored for the 2D challenge and can be filled with -1. Similarly, the bounding boxes are ignored for the 3D challenge. However, each line is still required to contain 10 values.

Der Bildausschnitt findet sich unter https://motchallenge.net/instructions/. (Abruf am 09.05.2021)

A.6. Technische Daten des Jetson Nano Development-Kits

Technische Daten des Jetson Nano Development-Kits	
Grafikprozessor	NVIDIA Maxwell™-Architektur mit 128 NVIDIA CUDA-Recheneinheiten 0,5 TFLOPS (FP16)
CPU	Quad-core ARM® Cortex® A57 MPCore mit 1,43 GHz
Arbeitsspeicher	4 GB, 64 Bit LPDDR4 1600 MHz - 25,6 GB/s
Datenspeicher	microSD (nicht enthalten)
Datenspeicher (Jetson Nano Modul)	16 GB eMMC 5.1-Flashspeicher
Schnittstellen	4x USB 3.0 Typ A USB 2.0 Micro-B HDMI Typ A Display Port (DP) Hohlsteckerbuchse (für 2,1 mm x 5,5 mm x 9,5 mm Stecker, positive Polarität) MIPI-CSI Camera Connector RJ-45 Gigabit Ethernet M.2 Key E 40-pin header (GPIO, I^2C, I^2S, SPI, UART) 12-pin header (Power and related signals, UART) 4-pin fan header
Abmessungen	100 mm x 80 mm x 29 mm (B x L x H)
Abmessungen (Jetson Nano Modul)	69,6 mm x 45 mm (B x L)
Gewicht	0,138 kg inkl. Kühlkörper
Power Supply	Micro USB (5 VDC, 2,5 A) Hohlsteckerbuchse (5 VDC, 4 A)
Energiebedarf	5 W, bei reduzierter Leistungsfähigkeit 10 W, bei maximaler Leistungsfähigkeit

A.7. YOLOv3-Tiny-Layer von dem Input-Layer bis zu den Layer für die Personenerkennung

Layer (type)	Output Shape	Param #	Connected to
input_86 (InputLayer)	[(None, 416, 416, 3)]	0	
conv2d_1113 (Conv2D)	(None, 416, 416, 16)	432	input_86[0][0]
batch_normalization_943 (BatchNormalization)	(None, 416, 416, 16)	64	conv2d_1113[0][0]
leaky_re_lu_943 (LeakyReLU)	(None, 416, 416, 16)	0	batch_normalization_943[0][0]
max_pooling2d_510 (MaxPooling2D)	(None, 208, 208, 16)	0	leaky_re_lu_943[0][0]
conv2d_1114 (Conv2D)	(None, 208, 208, 32)	4608	max_pooling2d_510[0][0]
batch_normalization_944 (BatchNormalization)	(None, 208, 208, 32)	128	conv2d_1114[0][0]
leaky_re_lu_944 (LeakyReLU)	(None, 208, 208, 32)	0	batch_normalization_944[0][0]
max_pooling2d_511 (MaxPooling2D)	(None, 104, 104, 32)	0	leaky_re_lu_944[0][0]
conv2d_1115 (Conv2D)	(None, 104, 104, 64)	18432	max_pooling2d_511[0][0]
batch_normalization_945 (BatchNormalization)	(None, 104, 104, 64)	256	conv2d_1115[0][0]
leaky_re_lu_945 (LeakyReLU)	(None, 104, 104, 64)	0	batch_normalization_945[0][0]
max_pooling2d_512 (MaxPooling2D)	(None, 52, 52, 64)	0	leaky_re_lu_945[0][0]
conv2d_1116 (Conv2D)	(None, 52, 52, 128)	73728	max_pooling2d_512[0][0]
batch_normalization_946 (BatchNormalization)	(None, 52, 52, 128)	512	conv2d_1116[0][0]
leaky_re_lu_946 (LeakyReLU)	(None, 52, 52, 128)	0	batch_normalization_946[0][0]
max_pooling2d_513 (MaxPooling2D)	(None, 26, 26, 128)	0	leaky_re_lu_946[0][0]
conv2d_1117 (Conv2D)	(None, 26, 26, 256)	294912	max_pooling2d_513[0][0]
batch_normalization_947 (BatchNormalization)	(None, 26, 26, 256)	1024	conv2d_1117[0][0]
leaky_re_lu_947 (LeakyReLU)	(None, 26, 26, 256)	0	batch_normalization_947[0][0]
max_pooling2d_514 (MaxPooling2D)	(None, 13, 13, 256)	0	leaky_re_lu_947[0][0]

conv2d_1118 (Conv2D)	(None, 13, 13, 512)	1179648	max_pooling2d_514[0][0]
batch_normalization_948 (BatchNormalization)	(None, 13, 13, 512)	2048	conv2d_1118[0][0]
leaky_re_lu_948 (LeakyReLU)	(None, 13, 13, 512)	0	batch_normalization_948[0][0]
max_pooling2d_515 (MaxPooling2D)	(None, 13, 13, 512)	0	leaky_re_lu_948[0][0]
conv2d_1119 (Conv2D)	(None, 13, 13, 1024)	4718592	max_pooling2d_515[0][0]
batch_normalization_949 (BatchNormalization)	(None, 13, 13, 1024)	4096	conv2d_1119[0][0]
leaky_re_lu_949 (LeakyReLU)	(None, 13, 13, 1024)	0	batch_normalization_949[0][0]
conv2d_1120 (Conv2D)	(None, 13, 13, 256)	262144	leaky_re_lu_949[0][0]
batch_normalization_950 (BatchNormalization)	(None, 13, 13, 256)	1024	conv2d_1120[0][0]
leaky_re_lu_950 (LeakyReLU)	(None, 13, 13, 256)	0	batch_normalization_950[0][0]
conv2d_1123 (Conv2D)	(None, 13, 13, 128)	32768	leaky_re_lu_950[0][0]
batch_normalization_952 (BatchNormalization)	(None, 13, 13, 128)	512	conv2d_1123[0][0]
leaky_re_lu_952 (LeakyReLU)	(None, 13, 13, 128)	0	batch_normalization_952[0][0]
tf_op_layer_ResizeNearestNeighbor_85 (TensorFlowOpLayer)	[(None, 26, 26, 128)]	0	leaky_re_lu_952[0][0]
tf_op_layer_concat_449 (TensorFlowOpLayer)	[(None, 26, 26, 384)]	0	tf_op_layer_ResizeNearestNeighbor_85 leaky_re_lu_947[0][0]
conv2d_1124 (Conv2D)	(None, 26, 26, 256)	884736	tf_op_layer_concat_449[0][0]
conv2d_1121 (Conv2D)	(None, 13, 13, 512)	1179648	leaky_re_lu_950[0][0]
batch_normalization_953 (BatchNormalization)	(None, 26, 26, 256)	1024	conv2d_1124[0][0]
batch_normalization_951 (BatchNormalization)	(None, 13, 13, 512)	2048	conv2d_1121[0][0]
leaky_re_lu_953 (LeakyReLU)	(None, 26, 26, 256)	0	batch_normalization_953[0][0]
leaky_re_lu_951 (LeakyReLU)	(None, 13, 13, 512)	0	batch_normalization_951[0][0]
conv2d_1125 (Conv2D)	(None, 26, 26, 255)	65535	leaky_re_lu_953[0][0]
conv2d_1122 (Conv2D)	(None, 13, 13, 255)	130815	leaky_re_lu_951[0][0]

A.8. Analysebereich GPS-Dezimalkoordinaten

Tabelle mit den GPS-Dezimalkoordinaten der Eckpunkte des Analysebereichs sowie der einzelnen Teilbereiche.

Bereich	**Eckpunkt**	**Breiten- und Längengrad**
Analysebereich	0.1	N 53,102731° E 8,857385°
	0.2	N 53,102676° E 8,857684°
	0.3	N 53,102606° E 8,857603°
	0.4	N 53,102637° E 8,857429°
Teilbereich 1	1.1	N 53,102731° E 8,857385°
	1.2	N 53,102699° E 8,857560°
	1.3	N 53,102665° E 8,857544°
	1.4	N 53,102687° E 8,857417°
Teilbereich 2	2.1	N 53,102699° E 8,857563°
	2.2	N 53,102676° E 8,857684°
	2.3	N 53,102650° E 8,857635°
	2.4	N 53,102665° E 8,857550°
Teilbereich 3	3.1	N 53,102677° E 8,857442°
	3.2	N 53,102662° E 8,857544°
	3.3	N 53,102623° E 8,857532°
	3.4	N 53,102637° E 8,857429°
Teilbereich 4	4.1	N 53,102662° E 8,857547°
	4.2	N 53,102647° E 8,857611°
	4.3	N 53,102606° E 8,857603°
	4.4	N 53,102622° E 8,857535°

A.9. Übersicht der manuell und durch das Embedded Vision System erfassten Personenzahlen

Ergebnis der manuell ermittelten Anzahlen an Personen, die sich durchschnittlich je Zeitintervall in den einzelnen Teilbereichen sowie dem gesamten Analysebereich aufgehalten haben.

	Zeit	Teilbereich 1	Teilbereich 2	Teilbereich 3	Teilbereich 4	Personen im Analysebereich
1	5,0	0,00	0,00	10,00	0,80	11,46
2	12,0	0,00	0,00	11,43	3,14	14,96
3	19,0	0,00	0,00	3,43	5,43	10,62
4	24,0	0,00	0,00	0,00	0,00	0,00
5	30,0	0,00	0,00	15,33	0,00	16,66
6	35,0	8,00	0,00	4,60	3,60	20,60
7	40,0	1,00	0,00	6,00	3,40	10,40
8	46,0	1,33	0,00	2,00	0,00	4,33
9	51,0	7,40	0,00	8,33	0,00	15,73
10	57,0	0,00	0,00	8,33	4,80	13,13
11	62,0	0,00	7,20	7,40	6,60	25,78
12	67,0	6,40	2,00	1,20	0,00	9,60
13	73,0	7,17	9,00	0,00	5,00	21,84
14	78,0	4,80	3,20	15,20	13,40	38,00
15	89,0	4,36	0,00	0,90	0,90	6,16
16	95,0	1,00	0,00	7,33	0,00	8,33
17	100,0	0,00	0,00	6,80	5,00	11,80
18	106,0	0,00	0,00	2,30	1,00	8,66
19	111,0	0,00	6,60	3,80	0,80	11,80
20	118,0	0,00	0,00	0,00	0,00	0,00
21	123,0	0,00	3,00	0,00	1,80	4,80
22	128,0	0,00	0,00	0,00	5,80	5,80

Ergebnis der durch das Embedded Vision System ermittelten Anzahlen an Personen, die sich durchschnittlich je Zeitintervall in den einzelnen Teilbereichen sowie dem gesamten Analysebereich aufgehalten haben.

	Zeit	Teilbereich 1	Teilbereich 2	Teilbereich 3	Teilbereich 4	Personen im Analysebereich
1	5.0	0.000000	0.000000	8.641975	1.666667	10.617284
2	12.0	0.000000	0.000000	13.765432	2.654321	16.419753
3	19.0	0.000000	0.000000	0.802469	5.555556	6.543210
4	24.0	0.000000	0.000000	0.000000	0.000000	0.308642
5	30.0	0.000000	0.000000	16.172840	0.000000	19.259259
6	35.0	10.308642	0.000000	4.259259	3.888889	23.271605
7	40.0	0.864198	0.000000	3.333333	2.654321	7.777778
8	46.0	0.740741	0.000000	4.382716	0.000000	7.839506
9	51.0	8.580247	0.000000	7.839506	0.000000	16.111111
10	57.0	0.000000	0.000000	7.283951	7.839506	18.765432
11	62.0	0.000000	8.518519	6.975309	12.160494	30.185185
12	67.0	9.506173	0.555556	0.864198	0.000000	10.370370
13	73.0	4.629630	11.913580	0.185185	4.938272	22.530864
14	78.0	3.271605	1.666667	16.790123	12.283951	36.419753
15	89.0	5.493827	0.000000	0.432099	0.000000	7.716049
16	95.0	0.000000	0.000000	7.037037	0.000000	7.345679
17	100.0	0.000000	0.000000	4.876543	5.555556	11.419753
18	106.0	0.000000	0.000000	1.481481	0.802469	6.790123
19	111.0	0.000000	9.135802	1.728395	2.283951	14.506173
20	118.0	0.000000	3.703704	0.000000	0.000000	3.703704
21	123.0	0.000000	9.629630	0.000000	3.580247	13.765432
22	128.0	0.000000	0.000000	0.000000	4.691358	5.617284

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 20. Mai 2021
Ort, Datum

[redacted]
Unterschrift