

BACHELORTHESIS
Niklas Ostermann

Echtzeitklassifikation von Objekten mittels Faltender Neuronaler Netze auf einer Eingebetteten GPU-Plattform für einen Elektronischen Blindenführhund

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

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

Niklas Ostermann

Echtzeitklassifikation von Objekten mittels Faltender
Neuronaler Netze auf einer Eingebetteten
GPU-Plattform für einen Elektronischen
Blindenführhund

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Lutz Leutelt
Zweitgutachter: Prof. Dr. Klaus Jünemann

Eingereicht am: 04.03.2021

Niklas Ostermann

Thema der Arbeit

Echtzeitklassifikation von Objekten mittels Faltender Neuronaler Netze auf einer Eingebetteten GPU-Plattform für einen Elektronischen Blindenführhund

Stichworte

EBF, Künstliche Intelligenz, Neuronale Netze, GPU, Kamera, FNN, Jetson

Kurzzusammenfassung

Diese Bachelorthesis beschäftigt sich mit dem Projekt *Elektronischer Blindenführhund (EBF)*. Es wird eine leistungsfähige Rechenplattform sowie eine Kamera ausgewählt und evaluiert auf Grundlage derer eine Objektdetektion in Echtzeit erfolgen und die zukünftig im EBF verwendet werden soll. Zu diesem Zweck wird auch ein auf künstlichen neuronalen Netzen basierendes Objektdetektionsnetzwerk ausgewählt und evaluiert.

Niklas Ostermann

Title of Thesis

Real-time classification of objects using convolutional neural networks oan an embedded GPU platform for an electronic guide dog

Keywords

EGD, Artificial Intelligence, Neural Networks, GPU, Camera, CNN, Jetson

Abstract

This bachelor thesis deals with the project *Electronic Guide Dog (EGD)*. A powerful computing platform and a camera will be selected and evaluated. With this components a real time object detection will take place and the components will be used in the EBF in the future. For this purpose, an object detection network based on artificial neural networks is also selected and evaluated.

Inhaltsverzeichnis

Abbildungsverzeichnis	vii
Tabellenverzeichnis	ix
1 Einleitung	1
1.1 Motivation	1
1.2 Anwendungsmöglichkeiten des elektronischen Blindenführhundes	3
1.3 Aufbau der Arbeit	5
2 Grundlagen	6
2.1 Maschinelles Lernen	6
2.1.1 Künstliche Neuronale Netze	7
2.1.2 Faltende Neuronale Netze	9
2.2 Objektdetektor-Netzwerke	12
2.2.1 Single Stage Detektor	12
2.2.2 Two Stage Detektor	13
2.2.3 Average Precision	14
2.3 Eingebettete Systeme und Computer	15
2.4 GPU-Computing	17
2.5 Serielle Kommunikationsstandards	19
2.5.1 UART	20
2.5.2 I ² C-Bus	21
2.5.3 SPI-Bus	22
3 Anforderungsanalyse	23
3.1 Hardware	25
3.1.1 Eingebettete Recheneinheit	25
3.1.2 Eingebettete Kamera	26
3.1.3 Spezifische Anforderungen an Recheneinheit und Kamera	27

3.2	Klassifikationsalgorithmus	28
3.3	Evaluation der ausgewählten Komponenten	30
4	Vergleich und Auswahl der Hard- und Softwarekomponenten	31
4.1	Eingebettete Recheneinheit	31
4.1.1	Vorstellung und Bewertung der zur Wahl stehenden Recheneinheiten	33
4.1.2	Auswahl einer Recheneinheit	43
4.2	Eingebettete Kamera	44
4.2.1	Vorstellung und Bewertung der zur Wahl stehenden Kameras . . .	44
4.2.2	Auswahl einer Kamera	45
4.3	Detektionsalgorithmus	47
4.3.1	Bestandteile und Funktionsweise eines Single Stage Detektors . . .	47
4.3.2	Auswahl eines Algorithmus	50
4.3.3	Funktionsweise und Bestandteile des Bilddetektors YOLO	50
4.4	Die <i>Video-Detector Plattform</i> als <i>Black Box</i>	54
4.5	Serielle Kommunikationsschnittstelle	55
4.6	Zusammenfassung der ausgewählten Komponenten	56
5	Praktische Realisierung und Implementierung der Komponenten	57
5.1	Eingebettete Recheneinheit und Kamera	57
5.1.1	Inbetriebnahme	57
5.1.2	Funktionstest der Recheneinheit	60
5.2	Objektdetektion mit YOLO	62
5.2.1	Darknet	62
5.2.2	YOLO-Anwendung	64
5.3	Implementierung der UART-Schnittstelle	66
5.3.1	Implementierung auf der Recheneinheit	68
5.3.2	Implementierung auf der <i>Motion&Sensor-Plattform</i>	71
6	Evaluierung der Hard- und Softwarekomponenten	74
6.1	Vorstellung der Evaluationsbasis	74
6.1.1	YOLO-Modell	74
6.1.2	Evaluationskriterien	75
6.1.3	Funktionen des Jetson Xavier NX zur Leistungsoptimierung	75
6.2	Vergleich verschiedener Rechnersystem zur Evaluierung der Echtzeitfähigkeit bei einer Objektdetektion	77
6.2.1	Objektdetektion in einem Einzelbild	78

6.2.2	Objektdetektion in einem Video	80
6.2.3	Objektdetektion in einem Livestream	82
6.2.4	Ergebnisse des Vergleichs	84
6.3	Spezifische Evaluierung der ausgewählten Hardwarekomponenten	86
6.3.1	Auswirkung verschiedener Kameraauflösungen auf die Detektions- ergebnisse	87
6.3.2	Auswirkung der PowerModi auf die Detektionsergebnisse	89
6.3.3	Auswirkung verschiedener Bildraten auf die Leistungsaufnahme	90
6.4	Bewertung der Implementierung der UART-Schnittstelle	91
7	Zusammenfassung und Ausblick	93
	Literaturverzeichnis	96
A	Anhang	103
	Selbstständigkeitserklärung	109

Abbildungsverzeichnis

2.1	Darstellung eines künstliches Neurons, modifiziert nach [69].	7
2.2	Qualitative Darstellung eines künstliches neuronales Netzes.	8
2.3	Extraktion von Features aus Trainingsdaten, modifiziert nach [70].	9
2.4	Beispiel einer Faltungsschicht, modifiziert nach [70].	10
2.5	Beispiel einer Anwendung des Max-Pooling, modifiziert nach [70].	10
2.6	Beispiel einer Normalisierung mittels ReLu-Funktion, modifiziert nach [70].	11
2.7	Beispiel eines fully connected Layer, modifiziert nach [70].	11
2.8	Schematische Darstellung eines SSD [25].	12
2.9	Schematische Darstellung eines TSD [25]	14
2.10	Aufgabenverteilung zwischen GPU und CPU [12].	18
2.11	Datenpaket der UART-Kommunikation.	20
2.12	Blockschaltbild eines I ² C-Bussystems.	21
2.13	Blockschaltbild eines SPI-Bussystems. Rot: Taktleitung, blau: Datenlei- tung, schwarz: Chip Select-Signal.	22
3.1	Blockschaltbild des inneren Aufbau des EBF.	23
4.1	Beispiellilder aus dem PASCAL VOC Datensatz [24].	47
4.2	Aufbau von Detektor-Netzwerken, am Beispiel von YOLOv4 [9].	48
4.3	Schematische Darstellung des YOLO-Verfahren [49].	51
4.4	Vergleich verschiedener Bilddetektoren im Hinblick auf die Vorhersagege- nauigkeit (<i>AP</i>) und die Verarbeitungsgeschwindigkeit (<i>FPS</i>) [9].	53
4.5	Blockschaltbild der UART-Verbindung im EBF	56
5.1	Der NVIDIA Jetson Xavier NX im Überblick [35].	58
5.2	Die Kamera mit angeschlossenem Flachbandkabel.	59
5.3	Der Xavier NX und die Kamera im System.	59
5.4	Ergebnis der Einzelbilddetektion zum Test von Darknet.	65

5.5	Ausgabe der Detektionsergebnisse auf der Konsole (oben) und im JSON-Format (unten).	66
5.6	Ablauf der UART-Kommunikation zwischen <i>Motion&Sensor Plattform</i> und <i>Video-Detector Plattform</i>	67
5.7	Beispiel einer Textdatei, in der die Detektionsergebnisse eines Livestream-Frames gespeichert sind.	68
5.8	Ablauf der Speicherung der Detektionsergebnisse in eine Datei.	69
5.9	Ablauf der Kommunikation zwischen der <i>Video-Detector Plattform</i> und der <i>Motion&Sensor-Plattform</i> zur Implementierung der UART-Kommunikation auf der <i>Video-Detector Plattform</i>	70
5.10	Ablauf der Kommunikation zwischen der <i>Video-Detector Plattform</i> und der <i>Motion&Sensor-Plattform</i> zur Implementierung der UART-Kommunikation auf der <i>Motion&Sensor Plattform</i>	72
5.11	Ausgabe der Detektionsergebnisse auf der Konsole der Recheneinheit (oben) und auf der Konsole der IDE (unten).	73
6.1	Ergebnisse der Einzelbilddetektion auf dem Xavier NX mit YOLOv3.	78
6.2	Ergebnisse der Einzelbilddetektion auf dem Xavier NX mit YOLOv4.	78
6.3	Ergebnisse der Einzelbilddetektion auf Server/Laptop mit YOLOv3/v4.	79
6.4	Ergebnisse der Videodetektion auf dem Xavier NX mit YOLOv3.	80
6.5	Ergebnisse der Videodetektion auf dem Xavier NX mit YOLOv4.	80
6.6	Ergebnisse der Videodetektion auf dem Server, CPU oben, GPU unten.	81
6.7	Ergebnisse der Livestreamdetektion auf dem Xavier NX mit YOLOv3.	82
6.8	Ergebnisse der Livestreamdetektion auf dem Xavier NX mit YOLOv4.	83
6.9	Oberfläche des Programmes <i>jtop</i>	86
6.10	Ergebnisse der Detektion mit einer Auflösung von $320 \times 480 \text{ Pixel}$	87
6.11	Ergebnisse der Detektion mit einer Auflösung von $3264 \times 2646 \text{ Pixel}$	88
6.12	Ergebnisse der Objektdetektion mit PowerMode 0.	89
6.13	Zusammenhang von Bildrate und Leistungsaufnahme.	90
A.1	Vergleich verschiedener Objektdetektoren [9].	103
A.2	Tabelle des Plattformvergleichs, erster Teil.	104
A.3	Tabelle des Plattformvergleichs, zweiter Teil.	105
A.4	Tabelle des Plattformvergleichs, dritter Teil.	106
A.5	Tabelle des Plattformvergleichs, vierter Teil.	107
A.6	Tabelle des Plattformvergleichs, fünfter Teil.	108

Tabellenverzeichnis

4.1	Aufstellung der zur Wahl stehenden eingebetteten Recheneinheiten.	33
4.2	Übersicht verschiedener MIPI-CSI Kameras, Tabelle angepasst nach [61]. .	45
5.1	Vergleich der Ergebnisse des Benchmark-Test.	60
5.2	Optionale Parameter beim Aufruf der Objektdetektion.	65
6.1	Übersicht der PowerModi des Jetson Xavier NX.	76

1 Einleitung

1.1 Motivation

In Deutschland gibt es 349.036 seheingeschränkte Menschen [11], die als schwerbehindert eingestuft werden. Darüber hinaus gibt es zusätzlich ca. 8,519 Millionen Bundesbürger [58], die an verschiedenen Augenkrankheiten leiden, die die Sehkraft der betroffenen Personen mehr oder weniger stark einschränkt.

Trotz der hohen Zahlen sind seheingeschränkte Personen im Alltag für nicht-seheingeschränkte Menschen nur bedingt wahrnehmbar. Dies liegt einerseits daran, dass sich nur die wenigstens Blinden äußerlich als solche kennzeichnen, aber auch daran, dass sich viele seheingeschränkte Personen, auf Grund von fehlenden Hilfsmitteln, nicht trauen am hektischen und schnelllebigen Alltag teilzunehmen. Meist haben die betroffenen Personen sehr unterschiedliche Anforderungen an die Hilfsmittel, sodass es bisher nicht möglich ist ein Gerät auf dem Markt anzubieten, mit dem alle Anforderungen gleichermaßen erfüllt werden. Auch stand bis vor einigen Jahren die Technologie nicht zur Verfügung, um zum Beispiel eine Objekterkennung in Echtzeit durchzuführen, auf Basis derer der Nutzer vor Hindernissen gewarnt werden könnte. Klassische Hilfsmittel, wie die Braille-Blindenschrift, gibt es zwar schon seit dem Anfang des 18. Jahrhunderts [13], allerdings hilft diese im Alltag nur bedingt und ist nur in wenigen Bereichen zu finden (z.B. an Ampelanlagen).

Typische Herausforderungen im Alltag seheingeschränkter Menschen sind zum Beispiel: die Orientierung im Supermarkt und Straßenverkehr, Identifikation von Fahrtrichtungen von Verkehrsmitteln im öffentlichen Personennahverkehr (ÖPNV) und Reaktion auf spontan auftretende Hindernisse in der eigenen Bewegungsrichtung. Für einige der Herausforderungen gibt es heute Lösungsansätze. So kann das Smartphone mit den passenden Apps zum Barcode-Leser werden oder Objekte erkennen [32]. Es gibt allerdings kein Gerät, welches die Funktionalitäten der vielen Teillösungen zusammenfasst und speziell auf den Benutzer zugeschnitten ist.

An diesem Punkt setzt das Projekt *Elektronischer Blindenführhund (EBF)* an, dass das Ziel ist verfolgt ein kompaktes, multifunktionales Gerät zu entwickeln mit dem sehingeschränkten Menschen das alltägliche Leben erleichtert wird. Im Rahmen des Projektes, welches von der *HAW Hamburg* initiiert wird, haben sich bereits mehrere Studierende im Rahmen von Bachelorarbeiten und -projekten an der Entwicklung des EBF beteiligt. Jede der Arbeiten hat sich mit verschiedenen Teilaufgaben beschäftigt, die bei der Entwicklung des EBF zu lösen sind. Die bisher entstandenen Ergebnisse sind für sich allein stehend aussagekräftig. So wurde zum Beispiel in einem Bachelorprojekt das Umfeld des EBF mit einem Lidar-Sensor erfasst um mögliche Hindernisse zu detektieren [60]. Dies ist mit einem Mikrocontroller realisiert worden. Allerdings besteht das Problem, dass alle Arbeiten auf unterschiedlichen Plattformen umgesetzt wurden. So wird zum Beispiel für die Berechnung eines neuronalen Netzes, das die Buslinien auf der Anzeigetafel eines Busses erkennen soll, ein Onlineserver verwendet, mit dem die Ergebnisse zwar sehr schnell berechnet werden, jedoch nicht zeitnah auf einem benutzergeführten mobilen Gerät zur Verfügung stehen. Auch lässt sich das verwendete neuronale Netz nicht direkt auf ein mobiles, bzw. eingebettetes System übertragen, da bei Onlineservern ein neuronales Netz sehr komplex aufgebaut sein kann und dadurch eine Vielzahl von Berechnungen durchgeführt werden müssen. Durch die hohe Rechenleistung des Servers werden trotzdem sehr schnell Ergebnisse generiert, die Ausführung des selben neuronalen Netzes auf einem eingebetteten System würde hingegen deutlich länger dauern und zusätzlich möglicherweise zu viel Speicherplatz benötigen. Aus diesem Grund ist es bisher nicht möglich gewesen die entwickelten Teillösungen zu einem Gerät zusammenzuführen. Dazu kommt, dass bei den bisherigen Arbeiten meist mit aufgenommen Bild/Videomaterial gearbeitet wurde, in dem zum Beispiel Busse erkannt werden. Da der EBF allerdings im schnelllebigen Alltag verwendet werden soll, muss die Erkennung von Objekten zwingend auf Live-Daten basieren, die in Echtzeit verarbeitet werden. Nur durch die Verarbeitung in Echtzeit ist es möglich dem Nutzer in spontan auftretenden Situationen die richtigen Anweisungen zu geben.

Um dies zu ermöglichen, wird im Rahmen dieser Arbeit eine Recheneinheit ausgewählt und evaluiert, die zukünftig im EBF verwendet wird und auf der eine Vielzahl von unterschiedlichen Anwendungen implementiert werden können. Für die Live-Verarbeitung der Daten wird eine Kamera benötigt. Auch diese soll im Rahmen dieser Arbeit ausgewählt und evaluiert werden. Neben den Hardwarekomponenten wird auch ein Algorithmus, in Form eines neuronalen Netzes, zur Objekterkennung ausgewählt, implementiert und evaluiert. Dabei soll der Algorithmus nicht optimiert werden, sondern lediglich verwendet und auf die weitere Verwendung im EBF untersucht werden. Dieser dient dazu, die

Funktionalität von Kamera und Recheneinheit zu evaluieren und zeigt so die Stärken, Schwächen und Merkmale der Hardware auf, die bei der zukünftigen Entwicklung des EBF beachtet werden sollten. Außerdem stellt der Algorithmus in Verbindung mit Recheneinheit und Kamera eine erste Grundlage dar, mit der der EBF in seinem Umfeld Objekte erkennen kann. Damit ist es möglich erste Tests mit einem Prototypen des EBF im Alltag durchzuführen und Daten für die weitere Entwicklung zu sammeln.

1.2 Anwendungsmöglichkeiten des elektronischen Blindenführhundes

Wie zuvor kurz beschrieben, kämpfen seheingeschränkte Personen mit einer Vielzahl von Problemen. Der Blinden- und Sehbehinderten Verband Niedersachsen e.V. hat einen typischen Tagesablauf einer betroffenen Person verschriftlicht [8]. Aus diesem Tagesablauf wird deutlich, wie viele ungeahnten Herausforderungen im Alltag für Seheingeschränkte auftreten, für die es meist keinerlei Hilfsmittel gibt.

Durch das Einprägen markanter Merkmale ist die Orientierung in der häuslicher Umgebung gut möglich, aber schon auf dem Weg zur Arbeitsstelle offenbaren sich Probleme. Das Erkennen und Ausweichen von statischen Hindernissen wie Straßenbeleuchtungen, Verkehrsschildern und Parkuhren ist bereits nur schwer möglich. Noch schwieriger ist es auf spontan auftretende Hindernisse oder Situationen wie Gegenverkehr durch Personen oder Radfahrer, Fahrzeuge, die aus Seitenstraßen/Ausfahrten kommen oder eine neu eingerichtete Baustelle zu reagieren. In genau solchen Situationen ist der Einsatz des EBF sinnvoll und im schlimmsten kann Fall sogar das Leben des Nutzers durch den EBF gerettet werden. All diese Hindernisse würden mit der Kamera aufgenommen und durch einen Algorithmus erkannt werden und der Nutzer wird mit eine entsprechende Nachricht gewarnt. Dabei ist es natürlich einfacher auf statische Hindernisse zu reagieren, aber auch dynamische Situationen können durch den richtigen Einsatz von neuronalen Netzen frühzeitig erkannt und umgesetzt werden.

Das Warnen vor Hindernissen ist ein wichtiger, aber nicht der einzige denkbare Anwendungszweck. Im Straßenverkehr haben Bus- und Bahnverbindung meist einen hohen Stellenwert, da dies die einzigen motorisierten Fortbewegungsmittel für seheingeschränkte Personen sind. Dort besteht die Schwierigkeit die gewünschte Linie und Fahrtrichtung zu erkennen. Auch hier kann der EBF behilflich sein und die Linie mit Fahrtrichtung erkennen und ansagen.

An einer solchen Lösung arbeitet eine parallel laufende Bachelorarbeit.

Ein Anwendungszweck außerhalb des Straßenverkehrs ist das Einkaufen. Auch im Supermarkt ist der Einsatz des EBF möglich. Solange der Nutzer immer den selben Markt benutzt, kann auch hier die Orientierung durch das Einprägen markanter Dinge erfolgen. Wird jedoch ein neuer/anderer Markt genutzt oder werden die Produkte im bekannten Markt neu angeordnet, fällt die Orientierung schwer. Hier kann der EBF, durch Erkennen von Warengruppen, zum Beispiel die aktuelle Abteilung ansagen oder den Inhalt eines Regales auflisten, vor dem der Nutzer mit dem EBF steht. Auch ist es denkbar, dass eine zuvor erstellte Einkaufsliste an den EBF übermittelt wird und er anschließend auf Knopfdruck den Nutzer im Supermarkt gezielt zu den einzelnen Produkten navigiert. Dadurch wird das Einkaufen auch in gänzlich unbekanntem Supermärkten möglich.

Dies sind nur drei Beispiele von vielen Anwendungsmöglichkeiten des EBF im Alltag, die durch die in dieser Arbeit auszuwählenden und zu untersuchenden Software- und Hardwarekomponenten ermöglicht werden sollen.

1.3 Aufbau der Arbeit

Nach der zuvor erfolgten Einführung in das Projekt EBF und der Vorstellung möglicher Anwendungszwecke, werden in Abschnitt 2 zunächst grundlegende Inhalte vermittelt, die für das weitere Verständnis der Arbeit wichtig sind. Dort werden die Themen *maschinelles Lernen*, *neuronale Netze*, *Bild-Detektoren*, *eingebettete Systeme*, *GPU-Computing* und *serielle Kommunikationsschnittstellen* beleuchtet. Anschließend werden in der Anforderungsanalyse (Abschnitt 3) Überlegungen für die sinnvolle Aufgabenverteilung innerhalb des EBF getroffen und Anforderungen an die auszuwählenden Hardware- und Softwarekomponenten, sowie an die auszuführende Evaluation abgeleitet und festgelegt. Auf Basis dieser wird im Konzept-Kapitel (Abschnitt 4) die aktuelle Marktverfügbarkeit von eingebetteten Rechnersystemen, mit der Spezialisierung auf hochperformante Berechnung, und eingebetteten Kameras analysiert und schlussendlich die benötigten Hardwarekomponenten ausgewählt. Des Weiteren werden Algorithmen zur **Objektklassifikation** näher betrachtet und ein Algorithmus für die weitere Verwendung im EBF ausgewählt. Im darauf folgenden Abschnitt 5 geht es um die praktische Realisierung der Hardware und Software. Die Hardwarekomponenten werden in Betrieb genommen und getestet, gleiches gilt für den ausgewählten **Objekt-Klassifikationsalgorithmus**. Außerdem wird eine serielle Kommunikationsschnittstelle implementiert. Nachdem die Funktionsfähigkeit aller Komponenten festgestellt werden konnte, wird die Hard- und Software im Hinblick auf die Verwendung im EBF evaluiert (Abschnitt 6). Dazu werden verschiedene Untersuchungen durchgeführt, die zeigen sollen, ob die ausgewählten Komponenten für einen **Bildklassifikation** in Echtzeit geeignet sind. Am Ende der Arbeit (Abschnitt 7) werden die Ergebnisse und Inhalte zusammengefasst und ein Ausblick für die weitere Entwicklung des EBF gegeben.

2 Grundlagen

In diesem Kapitel werden einige wichtige Grundlagen beleuchtet, die bei der Ausarbeitung und für das Verständnis der Arbeit eine wichtige Rolle spielen.

2.1 Maschinelles Lernen

Das maschinelle Lernen (ML) ist ein Teilbereich der künstlichen Intelligenz [64] und bildet die Grundlage intelligenter Maschinen [51]. Die Intelligenz kann auf fest programmierten Abläufen basieren oder durch maschinelles Lernen erzeugt werden [29].

Beim ML lernen Maschinen, bzw. Computer anhand großer Datenmengen anpassungsfähig und autonom zu agieren. Die Aufgabe, die die Maschine ausführen soll, ist dabei klar definiert. Das Besondere beim maschinellen Lernen ist, dass nicht wie bisher ein konkreter Lösungsweg, für die jeweilige Aufgabe fest programmiert ist, sondern ein Algorithmus *trainiert* wird, der selbstständig einen Lösungsweg entwickelt, bzw. erlernt. Dabei werden Strukturen und Muster in Trainingsdaten erkannt und diese Erkenntnisse in Form von Zahlen, in verschiedenen Modellen gespeichert, sodass Wissen generiert wird. Dieses Wissen kann bei einer erneuten Aufgabe des selben Typs angewendet werden. Ziel ist es, dass das Modell nach dem Training zu bisher vollkommen unbekanntem Daten richtige Aussagen treffen kann [65] und damit eine Aufgabe korrekt ausführt.

Eine Form der Speicherung des erlangten Wissen sind *künstliche neuronale Netze* (KNN). sobald neuronale Netze zum Einsatz kommen spricht man nicht mehr von maschinellem Lernen, sondern von *Deep Learning* [6].

2.1.1 Künstliche Neuronale Netze

Künstliche neuronale Netze (KNN) bilden die Verbindungen innerhalb des Gehirns eines Lebewesens und damit natürlich neuronale Netze nach [69]. Dabei geht es jedoch nicht um die exakte Nachbildung der natürlichen Verbindungen, sondern um die Abstraktion des Modells für eine Informationsverarbeitung.

In der Technik werden KNN meist nur *neuronale Netze* (NN) genannt, da klar ist, dass diese künstlich sind. Fortan wird dies in dieser Arbeit ebenfalls so gemacht. Wie in einem Gehirn, können mit einem NN sehr unterschiedliche Aufgaben realisiert werden. Typische Anwendungen sind Objekterkennung oder die Vorhersage von Aktienkursen und Verkehrsaufkommen.

Die Grundlage eines jeden neuronalen Netzes sind *Neuronen*. Neuronen sind mathematische Funktionen, die gleichzeitig mehrere Operationen durchführen. Die an das Neuron angelegten, gewichteten Werte werden summiert und anschließend mit einer *Aktivierungsfunktion* verarbeitet (siehe Abbildung 2.1). Am Ausgang eines jeden Neurons steht ein neuer Zahlenwert.

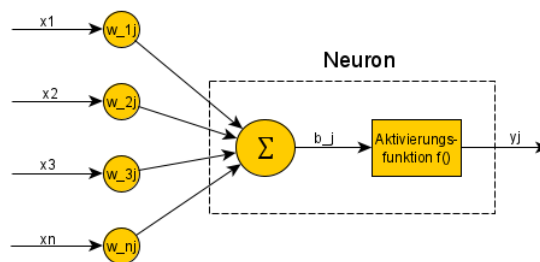


Abbildung 2.1: Darstellung eines künstliches Neurons, modifiziert nach [69].

Die mathematische Beschreibung des Neurons in Abbildung 2.1 lautet

$$b_j = \sum_{j=1}^n w_{ij} \cdot x_j \quad y_j = f(b_j). \quad (2.1)$$

Eine wichtige Rolle spielen dabei die Gewichtungen w_{ij} und die Aktivierungsfunktion. Mit den Gewichtungen wird festgelegt wie stark der jeweilige Wert in der Berechnung berücksichtigt wird. Sie werden beim oben erwähnten Training so lange variiert, bis das gewünschte Ergebnis erzielt wird. Ebenso kann durch Variation der Aktivierungsfunktion

das Verhalten des neuronalen Netzes verändert werden. Aktivierungsfunktionen können verschiedene Formen haben, sowohl lineare als auch nicht-lineare Funktionen sind möglich.

Ein neuronales Netz besteht aus mehreren Neuronen, die in Schichten (engl. *Layer*) angeordnet sind. Dabei sind die einzelnen Neuronen zwischen zwei Schichten vollständig miteinander verbunden. Das heißt, jedes Neuron hat Einfluss auf die Neuronen der nächsten Schicht. Dies visualisiert die Abbildung 2.2 anschaulich.

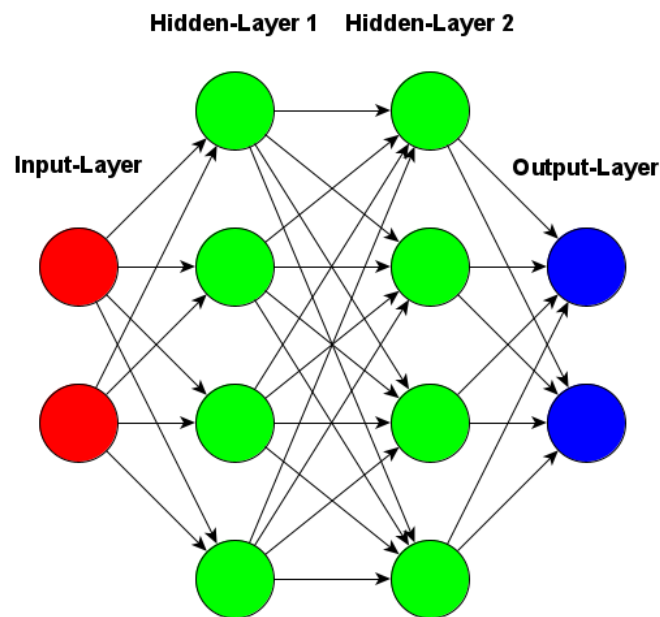


Abbildung 2.2: Qualitative Darstellung eines künstliches neuronales Netzes.

Die erste Schicht eines NN (siehe Abb. 2.2, rot markiert) nennt sich *Eingabeschicht*, engl. *input layer*. In dieser werden die zu verarbeitenden Daten übernommen und Neuronen zugeordnet. Die letzte Schicht (siehe Abb. 2.2, blau markiert) eines NN nennt sich *Ausgabeschicht*, engl. *output layer*. Hier steht das generierte Ergebnis zur Verfügung. Zwischen der Ein- und Ausgabeschicht befinden sich eine oder mehrere weitere Schichten (siehe Abb. 2.2, grün markiert), man spricht von *versteckten Schichten*, engl. *hidden layer*. Sobald das NN mehrere versteckte Schichten hat, wird von einem *tiefen neuronalen Netz*, engl. *deep neuronal network*, gesprochen. Die einzelnen Schichten können unterschiedliche Anzahlen von Neuronen enthalten und auch die Aktivierungsfunktionen der Schichten können variieren.

2.1.2 Faltende Neuronale Netze

Faltende neuronale Netze (FNN), engl. *convolutional neural networks* (CNN) sind eine spezielle Form von neuronalen Netzen und eignen sich durch die verwendeten, faltenden Schichten besonders gut für Bild-, bzw. bildatenförmige Datenverarbeitung [70].

Das zentrale Element eines FNN sind eine, bzw. mehrere faltende Schichten. Diese Schichten vergleichen sogenannte *Features* mit dem Eingangsbild. Features sind kleine Bildausschnitte mit einer Größe von wenigen Pixel. Gewöhnlich sind 3×3 *Pixel* (siehe Abb. 2.3, oben). In den Features sind Merkmale gespeichert, die im Eingangsbild zu erkennen werden sollen. Die Abbildung 2.3 zeigt das Eingangsbild (unten) und drei daraus extrahierte Features (oben). Das CNN soll in diesem Beispiel den Buchstaben *X* erkennen [70].

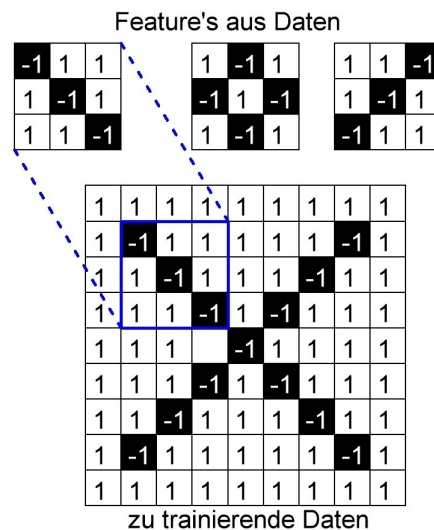


Abbildung 2.3: Extraktion von Features aus Trainingsdaten, modifiziert nach [70].

Das Feature wird pixelweise über das Eingangsbild geschoben, was einer Faltung entspricht. Bei jedem Schritt werden die Pixelwerte des, durch das Feature überdeckten Bildbereichs mit den Pixelwerten des Features multipliziert. Anschließend werden die Ergebnisse aufsummiert und durch die Anzahl der Pixel des Features geteilt. Am Ende eines Schrittes entsteht ein einziges, neues Ergebnis, welches in die sogenannte *Feature-Map* eingetragen wird. Dieser Vorgang ist in Abbildung 2.4 dargestellt.

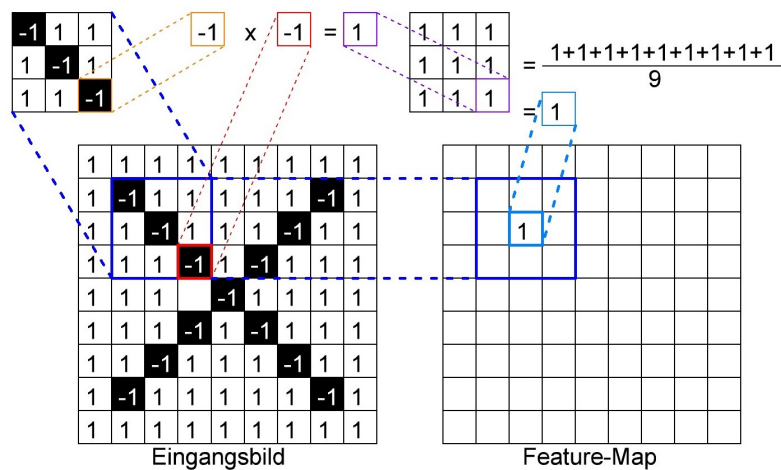


Abbildung 2.4: Beispiel einer Faltungsschicht, modifiziert nach [70].

Dieser Vorgang wird für jedes einzelne Pixel des Eingangsbilds mit einer variablen Anzahl von Featurs durchgeführt, sodass am Ende einer faltenden Schicht eine variable Anzahl von Feature-Maps entsteht. Jede der Maps beinhaltet die Information darüber, an welcher Stelle des Eingangsbildes das jeweilige Feature wie stark auftritt.

Nach der faltenden Schicht wird meist eine sogenannte *pooling*-Schicht angewendet. Diese Schicht verarbeitet die einzelnen Feature-Maps mit einem festgelegten Pixelrahmen von zum Beispiel 2×2 Pixeln. Aus dem ausgewählten Pixelrahmen der Feature-Map wird der Maximalwert extrahiert und dieser erneut in eine, dieses mal aber deutlich reduzierte, Feature-Map gespeichert. Das sogenannte *max-pooling* wird angewendet, um die Datengröße der Feature-Maps zu reduzieren, damit diese einfacher weiterverarbeitet werden können. Außerdem werden die einzelnen Merkmale des ursprünglichen Eingangsbildes weiter konzentriert. Das Vorgehen ist in Abbildung 2.5 dargestellt.

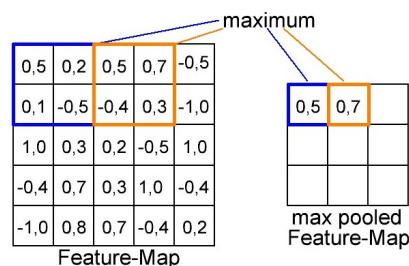


Abbildung 2.5: Beispiel einer Anwendung des Max-Pooling, modifiziert nach [70].

Als letzte versteckte Schicht eines FNN wird häufig eine Normalisierung durchgeführt. Auch die Normalisierung dient zur Datenreduktion und erfolgt durch die Anwendung der *Rectified Linera Units (ReLU)* Funktion auf jedes einzelne Pixel der Feature-Map. Die Funktion setzt alle negativen Werte auf Null und behält die positiven Werte unverändert bei. Die Anwendung der ReLu-Funktion ist in Abbildung 2.6 dargestellt.

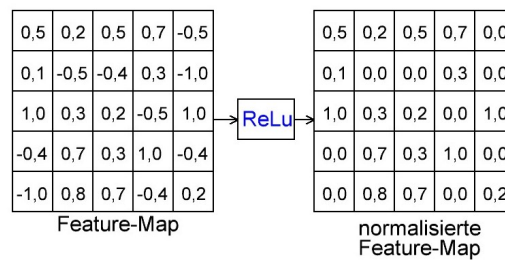


Abbildung 2.6: Beispiel einer Normalisierung mittels ReLu-Funktion, modifiziert nach [70].

Die einzelnen Schichten können beliebig angeordnet und wiederholt werden, sodass die gewünschte Funktion des CNN erreicht wird. Am Ende eines CNN wird eine sogenannte *fully connected* Schicht verwendet. Diese weitet die letztlich extrahierten Feature-Maps in einzelne Neuronen auf, die das Ergebnis der Datenverarbeitung beinhalten.

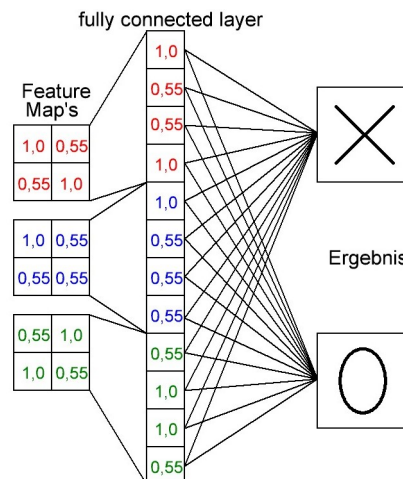


Abbildung 2.7: Beispiel eines fully connected Layer, modifiziert nach [70].

Auf diesen grundlegenden Schichten basieren intelligente Bilddetektor-Netzwerke.

2.2 Objektdetektor-Netzwerke

Detektor-Netzwerke sind auf neuronalen Netzen basierende Algorithmen, die speziell zur Objektklassifizierung und -erkennung optimiert sind. Hierbei geht es darum Objekte in einem Bild zu klassifizieren und den Ort des Objektes in dem Bild zu bestimmen. Grundsätzlich kann ein solcher Detektor als *single stage detector (SSD)* oder als *two stage detector (TSD)* aufgebaut sein. Beide Konzepte basieren auf faltenden neuronalen Netzen, bieten aber unterschiedliche Vor- und Nachteile, woraus unterschiedliche Anwendungszwecke resultieren. Nachfolgend werden die Grundlagen der beiden Detektorarten vorgestellt. Nähere Angaben zur Funktionsweise spezieller Detektoren sind in Abschnitt 4.3 zu finden.

2.2.1 Single Stage Detektor

Single stage Detektoren führen die Objekterkennung und Objektlokalisierung direkt vom Eingangsbild, in einem Schritt durch [24]. Das heißt, es wird lediglich ein neuronales Netz benötigt, das alle benötigten Arbeitsschritte beinhaltet. Die Abbildung 2.8 stellt ein SSD schematische Darstellung dar.

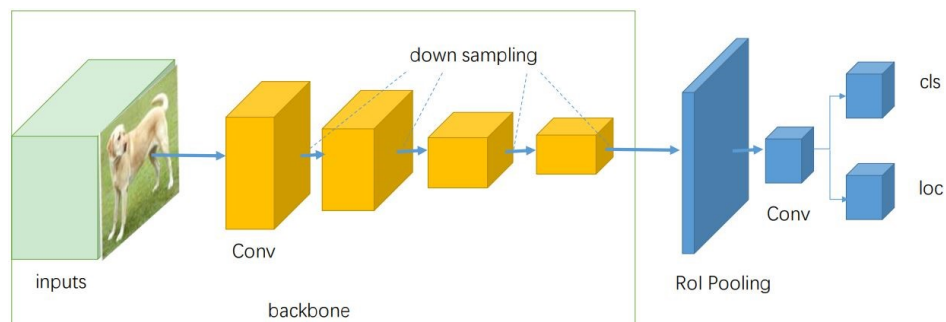


Abbildung 2.8: Schematische Darstellung eines SSD [25].

Gut zu erkennen sind die faltenden Schichten, die die Merkmale eines Bildes am Eingang extrahieren. Dieses Modul wird *backbone* genannt und stellt die Grundlage der Objekterkennung und Lokalisierung dar [25]. Am Ende des backbone ist eine feste Anzahl von Feature-Maps entstanden, die jeweils Merkmale spezifischer Klassen (zum Beispiel Auto, Vogel und Person) beinhalten und dadurch bereits erste Aussagen über die enthaltenen Objekte in einem Bild möglich machen. An dieser Stelle der Verarbeitung sind pro Bild,

viele hundert Vorhersagen von Objekten in den Feature-Maps gespeichert. Um aus dieser Vielzahl die genauen Objekte und vor allem deren Lokalisierung zu generieren, müssen die Daten in weiteren Schichten verarbeitet und gefiltert werden. So wird bei Merkmalen, die das selbe Objekt vorhersagen zum Beispiel berechnet, welche der Vorhersagen die genaueste ist. Dabei kann bei einem SSD schlussendlich pro Zelle nur eine einzige Vorhersage über ein Objekt getroffen werden. Wie dies im Speziellen erfolgt, wird in Abschnitt 4.3 beleuchtet. Am Ende des SSD stehen die Ergebnisse in Form von Objekten mit Wahrscheinlichkeit und dazugehörigen Koordinaten, mit denen das Objekt im Bild lokalisiert worden ist, zur Verfügung.

Der große Vorteil eines SSD gegenüber eines TSD ist die Verarbeitung in einem Schritt. Bei der Ausführung einer Objekterkennung spart dies Zeit und es sind höhere Verarbeitungsraten möglich. Genau für diesen Zweck der hochperformanten Objekterkennung in Bildern sind SSD entwickelt worden und ermöglichen Objekterkennungen in Echtzeit. Neben der besseren Performance, hat ein SSD beim Training den Vorteil, dass nur ein Netz trainiert werden muss. Beim TSD ist das Training von zwei separaten Netzen nötig.

2.2.2 Two Stage Detektor

Im Gegensatz zu Single Stage Detektoren, kommen bei einem *Two Stage* Detektor zwei separate neuronale Netze zum Einsatz. Zunächst wird ein Netzwerk zur verteilten Objektvorhersage verwendet. Hier werden Regionen innerhalb des Eingangsbildes ausfindig gemacht, die ein Objekt enthalten könnten. Diese möglicherweise interessanten Regionen werden in einem weiteren Netz optimiert und verfeinert [24]. Der schematische Aufbau eines TSD ist in Abbildung 2.9 abgebildet.

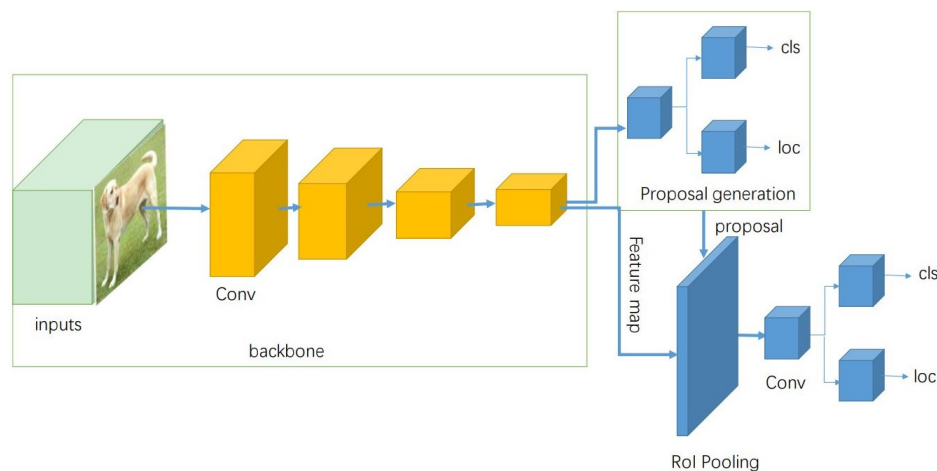


Abbildung 2.9: Schematische Darstellung eines TSD [25]

Der Aufbau ähnelt stark dem eines SSD, mit dem Unterschied des *proposal* Netzwerk, welches oben rechts in Abbildung 2.9 abgebildet ist. Wie dargestellt trifft dieses Netzwerk bereits Vorhersagen über Objekte und Lokalisierung, die jedoch noch sehr ungenau sind. Erst durch die Anwendung eines zweiten Netzwerks werden diese Vorhersagen, unter Zuhilfenahme der generierten Feature-Maps verfeinert, sodass am Ende des TSD eine möglichst genaue Vorhersage über Objekte und deren Lokalisierung in einem Bild gemacht werden kann. Für eine vollständige Objekterkennung müssen immer zwei Schritten nacheinander durchgeführt werden. Durch die Verarbeitung in zwei Schritten, dauert die Verarbeitung zwar länger, ist aber in den meisten Fällen genauer als ein SSD [25]. Dadurch eignet sich diese Methode für stationäre Anwendungen, bei denen die Verarbeitungszeit nur eine untergeordnete Rolle spielt.

2.2.3 Average Precision

Die *Average Precision (AP)*, deutsch *durchschnittliche Genauigkeit* ist ein Wert zwischen Null und Eins, der häufig für die Bewertung von Bild-Detektoren verwendet wird [70]. Der Wert gibt an mit welcher Genauigkeit ein Objekt erkannt wird, wenn es erkannt wird. Der Wert gibt keine Auskunft darüber wie gut der Detektor über alle Objekte arbeitet. Ein Wert von z.B. $AP = 76\%$ sagt aus, dass die erkannten Objekte mit einer durchschnittlichen Genauigkeit von 76 % erkannt worden sind. Es bedeutet nicht, dass 76 % aller Objekte erkannt werden.

2.3 Eingebettete Systeme und Computer

Ein eingebettetes System, engl. *embedded system*, ist eine Zusammenstellung verschiedener technischer Komponenten, die zusammen eine oder mehrere wohl definierte Aufgaben abarbeiten [53]. Eingebettete Systeme können, je nach zu erfüllenden Aufgaben, sehr komplex sein und finden in unterschiedlichsten Bereichen Anwendung. Das äußere Erscheinungsbild eingebetteter System kann sehr unterschiedlich sein. Oft ist für den Anwender nicht erkennbar, ob es sich um ein eingebettetes System handelt oder nicht. In diesen Fällen erledigt das eingebettete System im Hintergrund Arbeiten und der Mensch bzw. Nutzer sieht nur die Ergebnisse von diesen. Ein Beispiel einer solchen Anwendung stellt ein Smartphone dar. Hier wird eine Vielzahl verschiedener eingebetteter Systeme eingesetzt, die einzeln betrachtet kleine Teilaufgaben verrichten, aber im Zusammenspiel die komplexen Funktionalitäten eines Smartphones realisieren.

Typischerweise sind eingebettete Systeme programmierbar. Dabei kommen entweder Programmierungen zum Einsatz, die fest auf der Recheneinheit gespeichert sind und nicht verändert werden können, oder die Programmierungen lassen sich vom Nutzer anpassen. Die Recheneinheit wird dabei meist mit Mikrocontrollern oder Mikroprozessoren realisiert. In diesem Zusammenhang kann auch von eingebetteten Computern gesprochen werden.

Eingebettete Computer bieten ähnliche Funktionen wie ein normaler *Personal Computer (PC)*, unterscheiden sich aber in weiten Teilen von einem solchen [16]. Ein eingebetteter Computer wird meist nicht über Maus und Tastatur gesteuert, sondern verrichtet seine Arbeit eigenständig im Verborgenen und wird nicht direkt durch den Nutzer gesteuert. Meist findet gar keine Interaktion mit einem Nutzer statt, sondern ausschließlich mit anderen eingebetteten Computern und Systemen. Aus diesem Grund fehlt meist auch eine direkte, grafische Ausgabe auf einem Bildschirm. Eine solche Anwendung wird *kopflos*, engl. *headless*-Betrieb genannt. Die Aufgaben und Anforderungen an einen eingebetteten Computer variieren sehr stark, sodass für jeden Anwendungszweck ein spezielles Gerät verwendet wird, das die gewünschten Aufgaben erledigt. Ein Motorsteuergerät, als Beispiel für ein eingebettetes System, ist darauf ausgelegt alle benötigten Daten für den Betrieb eines Verbrennungsmotors bereitzustellen, erledigt dies perfekt, aber kann keine anderen Aufgaben, wie zum Beispiel das Abspielen von Musik, durchführen. Für solche Aufgaben werden andere Computer und Systeme benötigt, die mit dem Motorsteuergerät vernetzt sind, aber eigenständig Aufgaben erledigen. So ist es möglich ein komplexes System mit einzelnen, relativ einfach aufgebauten, aber stark spezialisierten Computern und Systemen zu realisieren.

Gemäß dem Anwendungszweck unterliegen eingebettete Computer gewöhnlicherweise unter Anderem Reglementierungen bei der Bauform und dem Energieverbrauch. Beide Parameter werden meist möglichst gering gehalten, um den Verbau in oft kleinen Bauräumen zu ermöglichen und die Energiekosten gering zu halten, bzw. bei akkubasierten Geräten wie dem EBF, die Einsatzdauer zu erhöhen. Die Software, die auf eingebetteten Computern zum Einsatz kommt ist meist auf Effizienz getrimmt. So kommen, wenn nicht unbedingt nötig, oft Recheneinheiten ohne Betriebssystem (BS) zum Einsatz, auf denen nur genau die Software implementiert wird, die für die Anwendung benötigt wird. Wenn ein BS benötigt wird, kommen abgespeckte Varianten bekannter Workstation-BS zum Einsatz. Oft werden verschiedenen Distributionen von *Linux* [56] verwendet, aber auch *Windows*-Versionen sind möglich [33].

2.4 GPU-Computing

Das GPU-Computing, oder auch *General Purpose Computation on Graphics Processing Unit* (GPGPU) ist die Verwendung von Grafikprozessoren über ihren eigentlichen Aufgabenbereich hinweg [63].

Grafikprozessoren, engl. *Graphical Processing Unit (GPU)*, werden gewöhnlicherweise auf Grafikkarten verbaut, die in einem PC für die Ausgabe von Grafiken auf einem Bildschirm sorgen. Dabei muss für jedes Pixel des Bildschirms der korrekte Farbwert berechnet werden. Bei Standard-Bildschirmen mit einer Auflösung von 1920×1080 Pixel bedeutet dies, dass gut zwei Millionen Pixel berechnet werden müssen, um das Bild auf dem Bildschirm zu aktualisieren. Die Zeit, die eine *Central Processing Unit (CPU)* [12] für diese Menge von Berechnungen brauchen würde, wäre so groß, dass keine flüssige Darstellung auf dem Bildschirm möglich wäre. Durch die spezielle Architektur einer GPU, wird dieses Problem behoben. Der Unterschied zwischen CPU und GPU besteht in der Anzahl der Rechenkerne. Normale CPU's besitzen heute üblicherweise vier bis acht Kerne, die sehr universell nutzbar sind und mit denen vier bis acht Aufgaben parallel ausgeführt werden können. Dagegen haben heutige GPU's mehrere Tausend Rechenkerne die speziell für die Berechnung von gleichförmigen Matrixoperationen konstruiert sind, mit denen die GPU die entsprechend hohe Anzahl von Aufgaben gleichzeitig ausführen kann. Dadurch lassen sich deutlich mehr Berechnungen pro Zeiteinheit durchführen, als mit einer CPU, wodurch die flüssige Darstellung von Bildern auch auf sehr hochauflösenden Bildschirm möglich ist. Der Vollständigkeit halber ist zu erwähnen, dass CPU's mit integrierter GPU am Markt verfügbar sind. Mit diesen ist ebenfalls eine flüssige Darstellung auf einem Bildschirm möglich, jedoch ist die Leistung einer integrierten Grafikeinheit deutlich geringer, als die einer dedizierten GPU.

Beim GPGPU wird, wie bei der gewöhnlichen Nutzung einer GPU, die GPU parallel zur CPU verwendet. Dabei steuert die CPU die Abläufe und vergibt Aufgaben an die GPU. Die GPU kann demnach als eine Art Co-Prozessor der CPU angesehen werden [28]. Die Abbildung 2.10 zeigt die Aufgabenverteilung zwischen GPU und CPU. Die CPU führt sequenziell auftretende Codeabschnitte, wie das Laden von Variablen oder die Ausgabe von Daten an Peripheriegeräte aus. Die GPU übernimmt rechenintensive Operationen, wie die Berechnung von Matrizen.

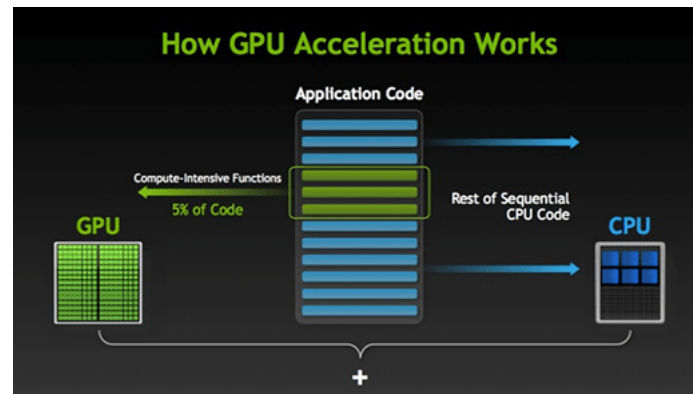


Abbildung 2.10: Aufgabenverteilung zwischen GPU und CPU [12].

Die hohe, parallele Rechenleistung der GPU wird genutzt und auf viele verschiedene Arten von Berechnungen übertragen. So wird GPGPU eingesetzt um in neuronalen Netzen sehr häufig vorkommende, große Matrizen sehr schnell und effizient zu verarbeiten, was die Verarbeitungszeit zwischen Eingabe der Rohdaten in das neuronale Netz und Ausgabe eines Ergebnisses deutlich reduziert.

Speziell für diese allgemeine Nutzung der GPU hat der Computer-Zubehör-Hersteller *NVIDIA* [36], exklusiv für die eigenen Grafikkarten und GPU's, die Programmiersprache *CUDA* [40] entwickelt. Diese ermöglicht die Nutzung der GPU als allgemeine Rechen-einheit in Hochsprachen wie C/C++, Python oder Fortran [10]. Dadurch erfreuen sich die Grafikkarten des Herstellers nicht nur großer Beliebtheit im Computerspielbereich, sondern auch in der Industrie und Wissenschaft zur Lösung aufwändiger, mathematischer Berechnungen. Zu erwähnen ist, dass sich auch Grafikkarten und GPU's anderer Hersteller (zum Beispiel *AMD* [2]) für das GPGPU-Computing nutzen lassen.

2.5 Serielle Kommunikationsstandards

Damit Computer, Prozessoren und Mikrocontroller miteinander Daten austauschen können, wird eine Kommunikationsmöglichkeit benötigt. Dabei gibt es heute viele verschiedene Kommunikationsprotokolle, die für verschiedene Anwendungszwecke mehr oder weniger gut geeignet sind [27]. Grob aufgeteilt werden diese sogenannten *Schnittstellen*, engl. *interface*, in *serielle* und *parallele Schnittstellen*.

Bei einer parallelen Schnittstelle werden, pro Takt, mehrere Bits parallel, also gleichzeitig zu dem/den Empfänger/Empfängern übertragen. Dies ermöglicht, bei entsprechend hoher Taktfrequenz, eine hohe Datenrate, allerdings werden für diese Verbindungsart eine hohe Anzahl von Leitungen und aufwändig gestaltete Steckverbinder benötigt, die kostenintensiv sein können. Diese Technik war in der Anfangszeit der PC-Technik, bis in die 2000er Jahre weit verbreitet.

Heute erfolgt der Datenaustausch meist über eine serielle Schnittstelle. Dabei werden die Bit's seriell, also nacheinander zu dem/den Empfänger/Empfängern gesendet. Diese Art der Übertragung erspart aufwändige, mehradrige Leitungen samt Stecker und ist, je nach Realisierung, deutlich störunempfindlicher [17] als die parallele Datenübertragung.

Die im Konsumermarkt am Weitesten verbreitete serielle Schnittstelle ist *USB* [68]. USB wird gewöhnlicherweise für den Datenaustausch zwischen einem PC und Peripheriegeräten, wie Smartphone, Drucker oder einem Messgerät, genutzt [68]. Für den Datenaustausch innerhalb eines eingebetteten Systems über kurze Distanzen bieten sich die seriellen Schnittstellen *UART*, *I2C* und *SPI* an. Was diese drei Schnittstellen auszeichnet und an welchen Stellen sich die technischen Spezifikationen voneinander unterscheiden, wird nachfolgend vorgestellt.

2.5.1 UART

Die Abkürzung *UART* steht für *Universal Asynchronous Receiver Transmitter* und beschreibt bereits die Besonderheit dieser seriellen Schnittstelle: Die Kommunikation erfolgt asynchron [66]. Entwickelt wurde UART für die Kommunikation zwischen einem PC und Peripheriegeräten. Allerdings wird die Schnittstelle auch für die Kommunikation zwischen verschiedenen eingebetteten Mikrocontroller- und Prozessoren verwendet. UART beschreibt das Grundprinzip der Kommunikation, aus der verschiedene Protokolle, wie RS-232 und RS-485 [17], mit unterschiedlichen technischen Spezifikationen entwickelt worden sind.

UART ist nicht adressorientiert, sondern es handelt sich um eine Punkt-zu-Punkt Verbindung zwischen Sender und Empfänger. Diese sind lediglich mit drei Leitungen miteinander verbunden: *Rx*, *Tx* und *GND*. Für das Empfangen und Senden von Daten steht jeweils eine Leitung zur Verfügung, weshalb eine Vollduplex-Kommunikation, also paralleles Senden und Empfangen, möglich ist. Die Besonderheit ist, wie bereits kurz erwähnt, dass der Datenaustausch asynchron erfolgt. Es ist keine Taktleitung nötig, mit der sich Sender und Empfänger synchronisieren, sondern der Empfänger reagiert dynamisch auf den Beginn des zu empfangenen Datenpaketes und synchronisiert sich dadurch. Das übertragene Datenpaket hat eine festgelegte Struktur, kann aber je nach System leicht variieren, was die Abbildung 2.11 zeigt.

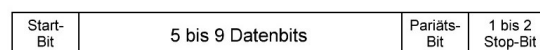


Abbildung 2.11: Datenpaket der UART-Kommunikation.

Mit dieser Struktur ist eine maximale Datenrate von bis 3 *Mbit/s* möglich, die jedoch nur bei sehr kurzen Leitungslängen erreicht werden kann [66]. An die verwendeten Leitungen und Steckverbinder werden kein besonderen Ansprüche gestellt. Der Vollständigkeit halber ist zu erwähnen, dass es auch UART-Varianten mit aktivem Handshake-Verfahren gibt. Bei diesen werden neben den drei erwähnten Leitungen, noch zwei bis drei zusätzliche Leitungen verwendet. Bei der Verwendung des UART muss beachtet werden, dass je nach Hardware, bzw. Controller unterschiedliche Signalpegel verwendet werden. Um diese aufeinander anzupassen, werden Pegelwandler verwendet. Die Programmierung der Schnittstelle ist in verschiedenen Programmiersprachen, durch die Verwendung von Bibliotheken, unkompliziert möglich.

2.5.2 I²C-Bus

Die Abkürzung *I²C* steht für *Inter Integrated Circuit* und ist ein BUS, der zum Austausch von Daten zwischen verschiedenen integrierten Schaltkreisen (IC), in einer Schaltung bzw. auf einer Leiterplatte dient [27]. Heute haben nicht nur IC's eine I²C-Schnittstelle, sondern auch Mikrocontroller und -prozessoren, womit der Einsatz in einem eingebetteten System möglich ist.

Der I²C-Bus ist als Master-Slave-Bus aufgebaut und der Datenaustausch erfolgt synchron. Das heißt, der Takt, in dem die Daten übertragen werden, wird vom Bus-Master bereitgestellt und an alle Bus-Slave weitergegeben. Neben der Taktleitung (SCL) wird auch die Datenleitung (SDA) an alle Komponenten des Bus angelegt. Der Aufbau eines solchen Systems ist in Abbildung 2.12 dargestellt.

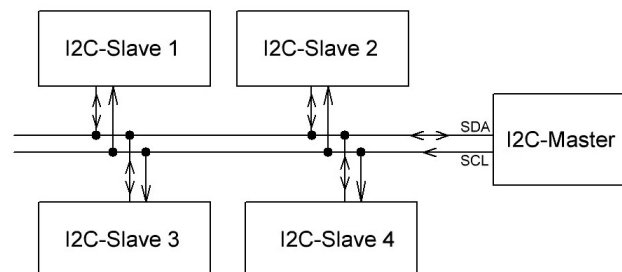


Abbildung 2.12: Blockschaltbild eines I²C-Bussystems.

Der Datenaustausch erfolgt nicht in einem festen Rahmen, sondern mit einer dynamischen Kommunikation zwischen Master und Slave. Welche Bus-Komponente Daten senden, bzw. empfangen soll wird durch den Master bestimmt, indem die Adresse der entsprechenden Komponente auf die Datenleitung gelegt wird, nachdem mit einer Start-Bedingung der Anfang einer Kommunikation signalisiert worden ist. Der angesprochene Bus-Slave bestätigt die Ansprache mit einem *Acknowledge-Bit (ACK-Bit)*, anschließend sendet der Master eine variable Anzahl von Daten-Bytes, wobei der Slave jedes empfangene Byte erneut mit einem ACK-Bit bestätigt. Das Ende des Datenaustauschs wird durch eine Stop-Bedingung signalisiert. Mit dieser Art der Kommunikation ist beim I²C-Bus eine maximale Datenrate von 1 *Mbit/s* möglich. Wie beim UART bestehen beim I²C-Bus keine besonderen Anforderungen an Leitungen und Steckverbinder. Für die Programmierung stehen verschiedene Möglichkeiten zur Verfügung, um die Schnittstelle in diversen Programmiersprachen zu implementieren.

2.5.3 SPI-Bus

Die Abkürzung *SPI* steht für *Serial Peripheral Interface* und ist, wie der I²C-Bus, ein BUS, der zum Datenaustausch innerhalb eines Schaltkreises entwickelt worden ist [27]. Auch der SPI-Bus ist nach dem Master-Slave-Prinzip aufgebaut und der Datenaustausch erfolgt synchron. Auch hier wird der Takt (SCLK) vom Master erzeugt und an alle Slave's weitergegeben, allerdings erfolgt die Kommunikation über zwei Leitungen (MOSI und MISO). Mit der MISO-Leitung empfängt der Master Daten von den Slave-Geräten, mit der MOSI-Leitung sendet der Master Daten an die Slave-geräte. Der Aufbau eines SPI-Bussystems ist in Abbildung 2.13 dargestellt.

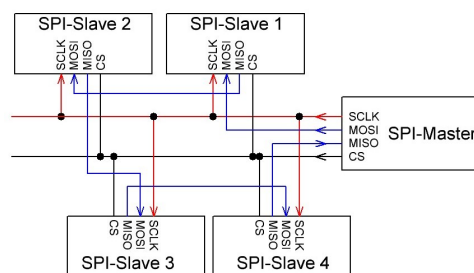


Abbildung 2.13: Blockschaltbild eines SPI-Bussystems. Rot: Taktleitung, blau: Datenleitung, schwarz: Chip Select-Signal.

Mit welchem Bus-Slave eine Kommunikation erfolgen soll wird mit dem *Chip Select (CS)*-Signal festgelegt. Das Signal besteht aus mehreren parallelen Bit, sodass mehrere unterschiedliche Slave-Komponenten angesprochen werden können. Mit dem Setzen des CS-Signal wird der Datenaustausch gestartet. Anschließend legt der Master die zu sendenden Daten, im Takt von SCLK, auf die MOSI-Leitung. Gleichzeitig werden Daten von der MISO-Leitung gelesen. Dadurch ist ein Vollduplex-Kommunikation möglich, wobei der Master immer die "Antwort" der vorherigen "Frage" bekommt. Es ist kein Stop-Signal nötig. Durch die rahmenlose Datenübertragung ist für den SPI-Bus keine maximale Datenrate festgelegt und es ist kein Handshake-Verfahren vorhanden. Wie auch bei den zuvor genannten Schnittstellen, bestehen keine besonderen Anforderungen an die Leitungen und Steckverbinder. Zu beachten ist jedoch, dass ohne weitere elektronische Schaltungen die Leitungslänge nur wenige Zentimeter betragen darf. Die Implementierung der Schnittstelle ist unkompliziert möglich, da nur einfache Ein- und Ausgabefunktionen benötigt werden, sodass eine direkte Implementierung, in zum Beispiel einem Linux-System, möglich ist.

3 Anforderungsanalyse

In diesem Kapitel werden die Anforderungen an die auszuwählenden Hard- und Softwarekomponenten festgelegt. Außerdem werden Kriterien für die Evaluierung der ausgewählten Komponenten festgelegt.

Grundlage für die Anforderungen an die auszuwählende Hardware ist das in Abbildung 3.1 dargestellte Blockschaltbild aus einer Vorbesprechung einer Masterarbeit, die sich parallel zu dieser Arbeit mit der Entwicklung eines Navigationssystem für den EBF beschäftigt.

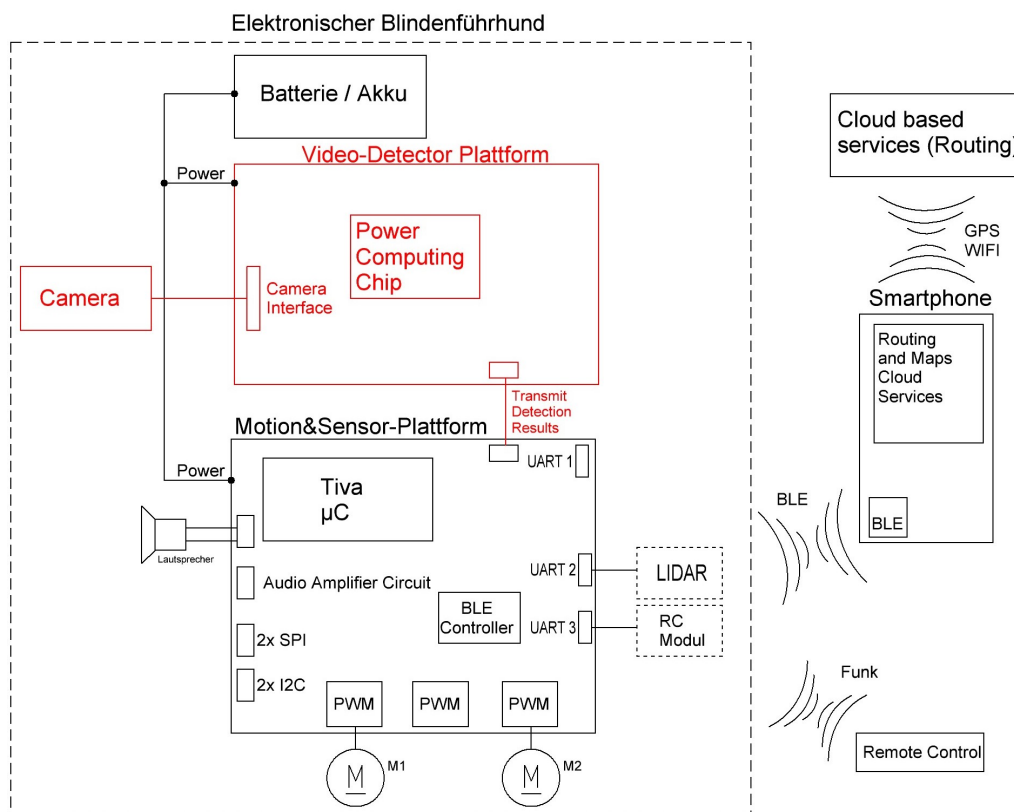


Abbildung 3.1: Blockschaltbild des inneren Aufbau des EBF.

Das Blockschaltbild (siehe Abbildung 3.1) zeigt, wie der innere Aufbau des EBF in Zukunft aussehen soll. Die genaue Konzeption und Konstruktion des inneren Aufbaus des EBF ist nicht Bestandteil dieser Arbeit, allerdings ist es für den weiteren Verlauf der Arbeit sinnvoll, dass das Blockschaltbild kurz erklärt wird. Am oberen Rand ist die Energieversorgung des Systems abgebildet. Hier bietet sich ein wiederaufladbarer Akku an, der genug Kapazität haben muss um alle Komponenten über eine ausreichende Zeit mit Energie zu versorgen. Darunter befindet sich die, *Video-Detector Plattform* genannte Recheneinheit (rot). Auf dieser sollen komplexe Vorgänge, wie die Berechnung neuronaler Netze zur Objekterkennung ausgeführt werden, für die eine hohe Rechenleistung benötigt wird. An die Recheneinheit wird eine Kamera angeschlossen. Diese dient zur Aufnahme der Bilder, auf denen eine Live-Objekterkennung basieren soll. Kamera und Recheneinheit sind die Hardwarekomponenten, mit denen sich in den folgenden Kapitel umfassend beschäftigt wird.

Unter der *Video-Detector Plattform* befindet sich die mikrocontrollerbasierte *Motion&Sensor Plattform*. An diese werden alle weiteren Ein- und Ausgabegeräte, wie Lautsprecher, Tasten und Sensoren angeschlossen. Mit dem Mikrocontroller sollen, unter anderem die Ergebnisse der Objekterkennung in passende elektrische Signale umgesetzt werden, die den späteren Nutzer zum Beispiel vor einer Gefahr warnt. Generell wird festgestellt, dass die *Motion&Sensor Plattform* für hardwarenahe Operationen und Anwendungen zuständig ist. Für die *Motion&Sensor Plattform* wird ein Mikrocontroller des Typs *Tiva™ TM4C1294NCPDT* [55] verwendet. Der Mikrocontroller ist sehr weit verbreitet, kostengünstig und wird in vielen Laborveranstaltungen an der HAW Hamburg verwendet. Daher ist der Controller vielen Studierenden bekannt, was die weitere Entwicklung des EBF vereinfacht. Außerdem steht zu dem Controller ein kostengünstiges Evaluation-Kit zur Verfügung, mit dem schnell und einfach Implementierungen vorgenommen werden können.

Zwischen der *Video-Detector Plattform* und *Motion&Sensor Plattform* soll eine Datenverbindung bestehen, um Daten zwischen den beiden Plattformen austauschen zu können. Die letzte wichtige Komponente des Systems ist ein Smartphone, das über Bluetooth an die *Motion&Sensor Plattform* angeschlossen wird. Das Smartphone soll als Schnittstelle zwischen Nutzer und EBF dienen. Außerdem soll es für die Navigation genutzt werden, damit keine zusätzlichen GPS- und WLAN-Controller im EBF verbaut werden. Auch ist der EBF darüber ständig, per WLAN oder über mobile Netzwerke an das Internet angeschlossen.

3.1 Hardware

Nachfolgend werden die Anforderungen an die auszuwählende Hardware, das heißt Recheneinheit (*Video-Detector Plattform*) und Kamera, analysiert und festgelegt. Dabei fließen sowohl äußerer Vorgaben als auch Aspekte, die bei der Recherche und Erarbeitung der Arbeit gesammelt worden sind ein.

3.1.1 Eingebettete Recheneinheit

Die Recheneinheit soll primär in der Lage sein, komplexe neuronale Netze in Echtzeit zu berechnen und die Ergebnisse in passender Form auszugeben. Dabei spielt die Echtzeitfähigkeit eine zentrale Rolle, denn der spätere Einsatzzweck des EBF ist das alltägliche Leben, in dem spontan auftretende Situationen sehr schnell verarbeitet werden müssen, um einen optimalen Schutz und Komfort des Nutzers zu gewährleisten. Der EBF wird ein mobiles, kompaktes und leicht handhabbares Gerät, das den Nutzer durch die äußeren Abmessungen nicht behindern darf. Deshalb ist es zwingend erforderlich, dass die Bauform der Recheneinheit möglichst kompakt ist. Hier bietet sich die Verwendung eines eingebetteten Computers an

Um den gezeigten Aufbau des EBF (Abbildung 3.1) möglichst effizient zu nutzen, ist es sinnvoll Überlegungen darüber anzustellen welche grundlegenden Aufgaben der Objekterkennung- und auswertung auf welcher der beiden genannten Plattformen ausgeführt werden sollten. Die Aufgaben definieren sich wie folgt:

- 1) Anschluss einer Kamera, als Quelle der Bilddaten für die Objekterkennung.
- 2) Objekterkennung mit Bildung und Ausgabe der Ergebnisse in Textform.
- 3) Auswertung der Ergebnisse der Objekterkennung.

Theoretisch ist es möglich, alle drei Aufgaben mit der *Video-Detector Plattform* zu erfüllen. Allerdings wird die Weiterentwicklung des EBF vereinfacht, wenn die Auswertung der Ergebnisse der Objekterkennung auf der *Motion&Sensor Plattform* geschieht und der Kameranschluss und die Objekterkennung mit der *Video-Detector Plattform* erfolgt.

Die Objekterkennung benötigt große Rechenleistung, die ein eingebetteter Computer mutmaßlich bieten kann und ein Mikrocontroller nur bedingt bereitstellen kann. Daher sollte die Objekterkennung auf der *Video-Detector Plattform* implementiert werden.

Die Bilder, in denen Objekte erkannt werden sollen, müssen möglichst schnell und konstant zur auswertenden Einheit gelangen. Aus der zuvor festgelegten Aufgabenverteilung folgt, dass die Kamera an die *Video-Detector Plattform* angeschlossen wird und diese eine entsprechende Schnittstelle aufweisen muss. Neben der Kameraschnittstelle sollte die Recheneinheit weitere, universell nutzbare Datenschnittstellen haben, um mit anderen Geräten kommunizieren zu können. Ein Netzwerkanschluss ist für die Entwicklung sinnvoll, für den späteren Betrieb des EBF jedoch nicht zwingend erforderlich.

3.1.2 Eingebettete Kamera

Bei der auszuwählenden Kamera handelt es sich um das zentrale Bauteile des EBF, dass für die Bilddatenbeschaffung zuständig ist. Die aufgenommen Bildern sollen als Live-Stream zur Recheneinheit weitergeleitet werden und dort passende neuronale Netze durchlaufen, die Objekte in den Bildern detektieren. Da die Kamera direkt an die Recheneinheit angeschlossen wird, muss eine passende Schnittstelle an der Kamera zur Verfügung stehen. Neben der passenden Schnittstelle, sollten Kamera und Recheneinheit auch softwareseitig zueinander passen, das heißt zur Kamera sollten passende Treiber für die Recheneinheit zur Verfügung stehen. Wie schon bei der Recheneinheit spielt auch hier die Bauform eine große Rolle. Sie sollte möglichst klein sein, sodass die Integration der Kamera in ein kompaktes Gehäuse möglich ist. Außerdem sollte sie eine zeitgemäße Auflösung von mindestens *Full High Definition* (FHD) [19] haben, damit die Objekterkennung zuverlässig funktionieren kann.

3.1.3 Spezifische Anforderungen an Recheneinheit und Kamera

Neben den technischen Anforderungen gibt es planerische Aspekte, die Beachtung finden müssen. So darf das Budget nicht überschritten werden und die ausgewählten Komponenten müssen kurzfristig beschaffbar sein, um im Rahmen dieser Arbeit damit arbeiten zu können. Die planerischen und technischen Anforderungen aus den Abschnitten 3.1.1 und 3.1.2 sind nachfolgend zusammengefasst:

- Eingebetteter Computer mit Betriebssystem und kompakter Bauform.
- Hohe Rechenleistung.
- Datenschnittstellen müssen vorhanden sein.
- Möglichste umfangreiche Unterstützung von KI/ML-Software-Frameworks.
- Software- und Hardwareschnittstelle für eine Kamera.
- Kamera mit mindestens FHD-Auflösung (1920x1080p).
- Möglichst kompakte Bauform.
- Möglichst geringer Energiebedarf.
- Maximales Budget von 500€ muss eingehalten werden.
- Schnelle Verfügbarkeit der Hardware.

3.2 Klassifikationsalgorithmus

Neben der Auswahl der Hardwarekomponenten ist auch die Auswahl eines Algorithmus zur Objektklassifizierung Teil dieser Arbeit. Eine Objektklassifizierung auf Grundlage eines Bildes trifft eine Aussage darüber welche Objekte in dem Bild enthalten sind. Im Hinblick auf den Anwendungszweck des EBF ist jedoch nicht nur festzustellen, dass bestimmte Objekte im Bild enthalten sind, sondern auch wo sich diese Objekte befinden. In diesem Fall spricht man von einer Objekterkennung oder -detektion.

Eine Objekterkennung kann mit verschiedenen Methoden erfolgen. Eine erste Methode ist die klassische Bildverarbeitung, bei der Algorithmen und mathematische Funktionen verwendet werden um bekannte Regelmäßigkeiten in den Quelldaten zu finden und diese entsprechend zu markieren. Diese Methode eignet sich besonders für statische Anwendungen (Kamera bewegt sich nicht, Umwelt ist konstant) und häufig wiederkehrende gleichförmige Bilddaten, wie sie zum Beispiel bei der optischen Kontrolle von Leiterplatten auftreten.

Deutlich besser für dynamische Anwendungen (Kamera bewegt sich, Umwelt ändert sich) geeignet ist die Verwendung von neuronalen Netzen als Objektdetektor. Bei dieser Variante wird zunächst ein scheinbar intelligentes neuronales Netz (Modell) mittels maschinellem Lernen auf die zu erkennenden Objekte trainiert. Das trainierte Modell wird auf die Bilddaten, die in diesem Fall von einer Kamera kommen, was eine Objekterkennung möglich macht. Die besondere Eignung für dynamische Anwendungen wird durch das Training des neuronalen Netzes erreicht, bei dem typischerweise eine hohe sechsstellige Zahl [14] von unterschiedlichen Bildern, in denen jeweils die zu erkennen Objekte enthalten sind, verwendet wird. Durch diese Fülle an Daten lernt das neuronale Netz eine große Menge an Merkmalen für die einzelnen zu erkennenden Objekte und ist dadurch in der Lage, zum Beispiel zwei Autos zu erkennen, obwohl Form, Farbe und Größe unterschiedlich sind. Außerdem kann das neuronale Netz relativ unkompliziert auf neue oder andere zu erkennende Objekte trainiert werden, indem ein entsprechend angepasster Trainingsdatensatz verwendet wird. Aus diesen Gründen soll beim EBF die Objekterkennung mittels neuronaler Netze zum Einsatz kommen.

Wie bereits in Abschnitt 3.1.1 erwähnt, liegt der Fokus bei der Objekterkennung auf der Echtzeitfähigkeit. Dies wird zum einen durch eine hohe Rechenleistung der Recheneinheit und zum Anderen durch einen entsprechend optimierten Detektionsalgorithmus erreicht. Daraus folgt, dass nur Algorithmen in Frage kommen, die in der Lage sind in kurzer Zeit eine große Menge von Bildern verarbeiten zu können. Auch sollte die mittlere Vorhersagegenauigkeit (AP), mit der die Objekte erkannt werden, mindestens 60 % betragen .

Beachtet werden muss auch der Umfang des neuronalen Netzes, da auf dem auszuwählenden, eingebetteten System nur begrenzter Speicherplatz zur Verfügung steht. Neben der Ausführung auf der eingebetteten Recheneinheit sollte es auch möglich sein, den Algorithmus auf Desktop-Computern und, oder Servern zu verwenden. Dies ist wichtig, da ein neuronales Netz typischerweise nicht auf der eingebetteten Recheneinheit trainiert wird, sondern auf einem extern Computer, der deutlich mehr Rechenleistung hat. Um das Training verifizieren zu können, muss das Modell auf diesem externen PC ausgeführt werden. Ein solches Vorgehen spart Zeit, da zum effizienten Trainieren neuronaler Netze, im Vergleich zur Ausführung von diesen, eine deutlich größere Rechenleistung benötigt wird. Diese Anforderung wird für die zukünftige Entwicklung des EBF festgestellt. Im Rahmen dieser Arbeit soll das neuronale Netz nicht trainiert werden!

Neben den technischen Anforderungen sind auch konzeptionelle Anforderungen vorhanden. Der verwendete Algorithmus sollte Open-Source verfügbar sein, sodass keine Kosten bei der Verwendung entstehen. Zudem hilft eine breite Community/Distributor-Unterstützung hilft bei der Lösung von möglicherweise auftretenden Problemen. Zusammengefasst bestehen folgende Anforderung an den Klassifikationsalgorithmus:

- Hauptzweck: Objekterkennung.
- Echtzeitfähig.
- Genauigkeit der Vorhersage soll im Bereich von $AP = 60\%$ bis 70% liegen.
- Speicherbedarf muss im Rahmen liegen.
- Ausführung auf eingebetteten Systemen und Workstation/Server-PC muss möglich sein.
- Open-Source.
- Breite Community/Distributor-Unterstützung.

3.3 Evaluation der ausgewählten Komponenten

Mit der Evaluation der ausgewählten Hardware- und Softwarekomponenten soll zum einen der Funktionsumfang von diesen analysiert, aber auch Schwächen, Stärken und spezifische Merkmale herausgearbeitet werden, die bei der zukünftige Verwendung im EBF Beachtung finden sollten.

Dabei sollen zunächst die Hardwarekomponenten als eigenständiges eingebettetes System analysiert und getestet werden, um die Eignung für die Verwendung im EBF festzustellen. Faktoren wie Leistungsaufnahme, Baugröße und Bedienbarkeit spielen dabei eine Rolle. Das Hauptaugenmerk liegt allerdings in der Verwendung der ausgewählten Komponenten zur Objekterkennung. Dabei wird der auszuwählende Detektionsalgorithmus auf den auszuwählenden Hardwarekomponenten ausgeführt. Ziel ist es, dass Objekte in den Bilddaten in Echtzeit erkannt werden und die Ergebnisse sowohl grafisch als auch in Textform ausgegeben werden. Zu untersuchen ist, wie die Objekterkennung auf verschiedene Parameter wie Bildauflösung, Bildrate und Rechenleistung reagiert. Auch sollen die hardwarespezifischen Funktionen der auszuwählenden Recheneinheit analysiert und getestet werden, sodass eine Aussage über die Auswirkungen der verschiedenen Einstellungen getroffen werden kann. Das eingebettete System wird zu diesem Zweck mit zwei weiteren, unterschiedlichen Rechnersystemen verglichen, auf denen jeweils die selbe Objekterkennung durchgeführt wird. Zusammengefasst bestehen folgende Anforderung an die Evaluation:

- Test der Hardwarekomponenten als eingebettetes System.
- Bewertung von Bauform, Leistungsaufnahme und allgemeiner Bedienbarkeit.
- Test der Hardware- und Softwarekomponenten zur Objekterkennung.
- Vergleich mit zwei weiteren, unterschiedlichen Rechner-Systemen.
- Untersuchung hardwarespezifischer Funktionen.
- Untersuchung der Einflussnahme verschiedener Parameter auf die Ergebnisse der Objekterkennung.
- Untersuchung der zu implementierenden Kommunikation zwischen *Video-Detector Plattform* und *Motion&Sensor Plattform*.

4 Vergleich und Auswahl der Hard- und Softwarekomponenten

Dieses Kapitel beschäftigt sich tiefgreifend mit den auszuwählenden Hardware- und Softwarekomponenten. Dabei werden die in der Anforderungsanalyse festgelegten Anforderungen durch Recherche ergänzt und konkretisiert, sodass abschließend eine Auswahl für die einzelnen Komponenten getroffen werden kann.

4.1 Eingebettete Recheneinheit

Durch die in Abschnitt 3.1.1 festgelegten Anforderungen wird die *Video-Detector Plattform* genau beschrieben, sodass einige Gerätetypen von Anfang an ausgeschlossen werden können. So kommen große Workstation/Server-PC's genau so wenig in Frage wie kleine Mikrocontroller-Lösungen.

Um eine passende Recheneinheit zu finden ist eine umfassende Internetrecherche unumgänglich. Es sind einige eingebettete Computer auf dem Markt erhältlich, allerdings handelt es sich bei diesen meist um Kompakt-PC's, die für den Gebrauch im industriellen Umfeld gedacht sind. Die Bauform dieser PC's ist deutlich kleiner, als die Bauform von herkömmlichen Workstation-PC's. Für die Integration in ein kompaktes Gerät, bzw. das Gehäuse eines kompakten Gerätes ist sie jedoch immer noch zu groß. Außerdem liegen die Geräte deutlich über dem maximalen Budget. Allerdings lassen sich bei diesen PC's bereits einige Merkmale finden, mit denen man eingebettete Computer miteinander vergleichen kann, wie zum Beispiel die Speicherkapazität, die Spannungsversorgung und die vorhandenen Anschlüsse. Nach der Ausweitung der Recherche mittels englischer Suchbegriffe, lassen sich mehrere Hersteller und Produkte finden, die dem Anforderungsprofil eher entsprechen. Die meisten der in Frage kommenden Produkte werden als *Single-Board-Computer* (SBC) bezeichnet. Dabei handelt es sich um vollständige Computer, die

auf nur einer einzigen, kleinen Leiterplatte realisiert werden. Alle handelsüblichen Anschlüsse wie USB, HDMI und Ethernet sind bei einem SBC meist enthalten, sodass sich die Geräte wie ein normaler PC benutzen lassen. Meist zielen die SBC auf Entwickler ab, die schnell und einfach Produkte oder Software entwickeln wollen. Man spricht in diesem Zusammenhang auch von einem *developer-kit* oder *developer-board*. Bekannt geworden ist dieser Typ von Computer mit dem *Raspberry Pi* [48], der mittlerweile in der vierten Version erschienen ist und sich großer Beliebtheit erfreut. Es zeigt sich, dass es neben vielen Konsumer-SBC, wie dem Raspberry Pi, auch einige spezielle Geräte in diesem Format am Markt verfügbar sind. Diese Geräte sind auf die eingebettet Anwendung von künstlicher Intelligenz und neuronalen Netzen ausgerichtet und haben dafür spezielle Chips/-Controller verbaut. Die Recherche liefert neben den einzelnen SBC's auch viele Kriterien, mit denen man die Geräte vergleichen kann. Folgende Kriterien werden verwendet: Betriebssystem, CPU-Typ, CPU-Takt, GPU, KI-Technik, KI-Performance, Arbeitsspeichergröße, Massenspeichergröße, Power-Computing-Technik, Displayanschluss, Kameraanschluss, Netzwerkanschluss, Datenschnittstellen, spezieller Anwendungsbereich, unterstützte KI/ML-Frameworks, Leistungsaufnahme, mechanische Abmessungen, Erscheinungsjahr und Preis.

Aus diesen Kriterien ist eine umfangreiche Vergleichstabelle entstanden (siehe Abb. A.2, A.3, A.4, A.5 und A.6) in die die einzelnen SBC eingetragen werden. Die Tabelle in Originalform ist auf der CD zu finden. Eines der wichtigsten Kriterien ist die sogenannte KI-Performance. Bei der Berechnung von neuronalen Netzen muss eine sehr große Anzahl von mathematischen Operationen durchgeführt werden. Je schneller diese erfolgen, desto schneller steht ein Ergebnis zur Verfügung. Die KI-Performance wird in *Floating Point Operations Per Second (FLOPS)* angegeben und ist keine physikalische Größe, sondern setzt sich aus der CPU/GPU-Frequenz, der Anzahl der CPU/GPU-Kerne, den Instruktionen pro Takt und der Anzahl von vorhandenen CPU/GPU zusammen. Die Hersteller werben meist mit diesem Wert. Dabei gilt: je höher der Wert ist, desto schneller können Berechnungen durchgeführt werden. Teilweise ist jedoch fraglich auf welche Bestandteile der Geräte sich der angegebene Wert bezieht. Da die Hersteller allerdings nur einen Wert für den ganzen Computer/SBC angeben, wird angenommen, dass bei der Berechnung alle verbauten Komponenten berücksichtigt worden sind, die in irgendeiner Form Rechenleistung zur Verfügung stellen. Neben der CPU sind dies die möglicherweise verbaut GPU und spezielle KI-Controller.

Um einen groben Überblick über die zur Wahl stehenden Geräte zu geben, werden diese zusätzlich in der nachfolgenden Tabelle 4.1 aufgeführt.

Tabelle 4.1: Aufstellung der zur Wahl stehenden eingebetteten Recheneinheiten.

Lfd. Nr.	Hersteller	Bezeichnung	KI-Compute-Technik
1	Google	Coral Dev Board	Google Edge TPU
1.1	Google	Coral USB Accelerator	Google Edge TPU
2	Intel	UP Squared Pentium Quad Core 08/64	Nicht vorhanden, Basis-Board
2.1	Intel	UP AI Core X Series (Myriad™X)	Movidius™ Myriad™ X VPU 2485
2.2	Intel	Neural Compute Stick 2	Movidius™ Myriad™ X VPU 2485
3	Asus	Tinker Edge R	Rockchip NPU
4	Asus	Tinker Edge T	Google Edge TPU
5	NVIDIA	Jetson Nano Dev-Kit	128-CUDA-core Maxwell GPU
6	NVIDIA	Jetson TX2 Dev-Kit	256-CUDA-core Pascal GPU
6.1	NVIDIA	Jetson TX2i SoM	256-CUDA-core Pascal GPU
7	NVIDIA	Jetson Xavier NX Dev-Kit	384-CUDA-core Volta GPU, 45 Tensor Cores
8	NVIDIA	Jetson AGX Xavier Dev-Kit	512-CUDA-core Volta GPU, 64 Tensor Cores
9	BeagleBoard	BeagleBone AI	embedded-vision-engine (EVE)
10	Raspberry Pi	Raspberry Pi 4 B	Nicht vorhanden

4.1.1 Vorstellung und Bewertung der zur Wahl stehenden Recheneinheiten

Nachfolgend werden die in Tabelle 4.1 aufgeführten, eingebetteten Computer vorgestellt und die grundlegendsten Vor- und Nachteile dargestellt. Umfassende Angaben zu Details

und den technischen Daten der einzelnen Geräte sind der Tabelle auf der CD zu entnehmen. Die in der Tabelle aufgezählten Rechen-USB-Stick's von *Google* und *Intel* sind zwar in der Vergleichstabelle auf der CD enthalten, allerdings werden diese nachfolgend nicht berücksichtigt, da sie eher für die Verwendung an einem Workstation-PC gedacht sind und nicht für den Einsatz in einem eingebetteten System. Auch der *NVIDIA Jetson AGX Xavier* wird nicht näher beleuchtet, da dieser außerhalb des Budgets liegt.

1. *Google Coral Dev Board* [18]

Das *Google Coral Dev Board* ist seit März 2019 verfügbar und speziell auf die Verarbeitung von neuronalen Netzen ausgelegt. Dafür hat es die *Google Tensor Processing Unit (TPU)* verbaut. Dies ist ein spezieller Controller, der für die Verarbeitung von Tensoren optimiert ist, die häufig in neuronalen Netzen auftreten.

Vorteile:

- 4 *TFLOPS* durch TPU.
- Ethernet, WLAN, Bluetooth integriert.
- Geringe Leistungsaufnahme von max. 10 *W*.
- Geringe mechanische Abmessungen (85 x 56 *mm*).
- Im unteren Preissegment angesiedelt (119,22 €).

Nachteile:

- Nur eingeschränkte Framework-Unterstützung (TF Lite, AutoML Vision Edge).
- Nur 1 *GB* Arbeitsspeicher vorhanden.
- Nur ein MIPI-CSI Kameraanschluss vorhanden.
- Einzig spezielle Debian-Version (Mendel) wird unterstützt.
- Wenig Community-Unterstützung, da noch sehr neu auf dem Markt.

Zusammenfassend bietet das Dev-Board eine gut Grundlage für den Einstieg in das Thema maschinelles Lernen und die Anwendung von einfachen neuronalen Netzen. Mit der TPU und dem vergleichsweise geringen Preis wird ein breites Publikum angesprochen. Die eher schlechte Framework-Unterstützung und der Arbeitsspeicher sind jedoch große Einschränkungen, die die weitere Entwicklung de EBF behindern könnten.

2. und 2.1 Intel® UP Squared UP AI Core X Series (Myriad™ X) [22] [21]

Die *Up Squared* Baureihe von Intel umfasst mehrere, mehr oder weniger eingebettet Computer, die für viele unterschiedliche Anwendungszwecke konfiguriert werden können. Die Grundlage bildet ein SBC, der mit unterschiedlichen Prozessoren ausgestattet sein kann. Um das Gerät auf künstliche Intelligenz auszurichten, wird eine Aufsteckplatine des Typs *UP AI Core X Series (Myriad™ X)* benötigt. Diese Platine beinhaltet eine speziell für die Berechnung von neuronalen Netzen entwickelte *Vision Processing Unit (VPU)*. Die Aufsteckplatinen sind in unterschiedlichen Bauformen und mit unterschiedlicher Anzahl von VPU's verfügbar. Beim eingebetteten Computer ist jedoch nur die niedrigste Ausbaustufe mit einer VPU verwendbar.

Vorteile:

- Bis zu 4 *TFLOPS* durch VPU.
- Gute Framework-Unterstützung.
- Alle gängigen Betriebssysteme lassen sich verwenden.
- Geringe mechanische Abmessungen (85 x 56 mm)
- Optional großer Massenspeicher (bis zu 128 GB) erhältlich.

Nachteile:

- Separates VPU-Modul wird benötigt.
- I2C- und SPI-Schnittstellen fehlen.
- Hohe Leistungsaufnahme von ca. 25 W.
- Im mittleren Preissegment angesiedelt (323,01 €).
- Wenig bis gar keine Community-Unterstützung, da noch sehr neu auf dem Markt.

Die Produkte von Intel sind durch die Verwendung von Aufsteckplatinen sehr variabel. Die Rechenleistung bewegt sich auf dem selben Niveau wie beim Coral-Dev-Board, die Größe des an Massen- und Arbeitsspeichers ist bei der Bestellung stufenweise wählbar. Auch lassen sich alle gängigen Betriebssysteme verwenden. Diese Punkte begründen den hohen Preis. Der eingebettete Intel-Computer mit passender KI-Aufsteckplatine ist mehr als doppelt so teuer wie das Coral-Dev-Board. Allerdings ist das Preis-Leistungsverhältnis im Vergleich zum Coral-Dev-Board eher schlecht, da der Preis deutlich steigt, die Rechenleistung jedoch ebenfalls 4 *TFLOPS* beträgt.

3. und 4. *Asus Tinker Edge T und R* [4] [5]

Der Hersteller Asus hat mit der Serie *Tinker Edge* zwei verschiedene SBC im Angebot: das Modell *T* und *R*. Das T-Modell ist von den technischen Daten baugleich mit dem Google Coral-Board. Deshalb wird es nicht näher vorgestellt. Es hat ebenso die Google-TPU verbaut, kostet allerdings etwas mehr (154,99 €) als das Vorbild.

Das R-Modell hingegen ist eine Eigenentwicklung von *Asus* und ist erst im Mai 2020 erschienen. Der SBC hat neben schnellen Dual-Core und Quad-Core ARM Prozessoren (zwei Prozessoren in einem Gehäuse) eine *Neural-Network Processing Unit (NPU)* verbaut, die speziell für die schnelle Berechnung von neuronalen Netzen konzipiert ist. Von der Rechenleistung ist die NPU mit 3 *TFLOPS* der Google-TPU leicht unterlegen.

Vorteile:

- Debian (Linux) und Android als Betriebssystem nutzbar.
- Alle gängigen Daten-Schnittstellen vorhanden.
- Zwei MIPI-CSI Kameranschlüsse vorhanden.
- 4 GB Arbeitsspeicher + 2 GB NPU-Speicher vorhanden.
- Geringe mechanische Abmessungen (85 x 56 mm)
- Im mittleren Preissegment angesiedelt (232,97 €)

Nachteile:

- Mit bis zu 3 *TFLOPS* Rechenleistung im Mittelfeld des Vergleichs.
- Breite Framework-Unterstützung nur durch Umwandlungen der Modelle nutzbar.
- Wenig bis gar keine Community-Unterstützung, da noch sehr neu auf dem Markt.

Die *Tinker Edge* Serie lässt sich in die gleiche Leistungsklasse wie das Coral-Dev-Board und Intel Up-Squared einordnen und liegt im Vergleich von diesen drei Geräten sowohl hinsichtlich der Rechenleistung, als auch der Kosten im Mittelfeld. Das R-Modell hebt sich durch die leistungsstärkere CPU und die verbaute NPU deutlich vom T-Modell ab. Trotzdem hat dieser SBC eine 25% geringere Rechenleistung (3 *TFLOPS*) als der Google-Coral SBC und eine noch schlechtere Community-Unterstützung als die Mitbewerber.

5. bis 8. NVIDIA Jetson-Familie

NVIDIA ist einer der größten Hersteller von Grafikkarten und GPU's [36]. Bis vor ca. 10 Jahren sind GPU's hauptsächlich für Anwendungen mit hohem grafischen Anspruch (PC-Spiele) verwendet worden. Heute werden diese zunehmend zum allgemeinen, massiven parallelen Rechnen verwendet. Man spricht vom im Abschnitt 2.4 vorgestellten *GPGPU*-Computing. Von dieser Technik profitiert unter Anderem der Forschungsbe- reich der künstliche Intelligenz mit den dort verwendeten neuronalen Netzen. In die- sen muss möglichst schnell eine hohe Zahl von gleichförmigen Berechnungen durchge- führt werden. Dabei gilt allgemein: Je schneller, desto besser. Neben den, eigentlich für die Computerspiele-Industrie entwickelten, handelsüblichen Grafikkarten bietet NVIDIA auch Produkte an, die speziell für die Verwendung als eingebettetes System und die Be- rechnung neuronaler Netze konzipiert sind. Diese Produktserie trägt den Namen *Jetson* [34]. Die Jetson Recheneinheiten sind SBC, die auf einem *System on Module (SoM)* auf- gebaut und GPU-basiert sind und dafür konzipiert in vielen verschiedenen Endprodukten eingesetzt zu werden. In diesen sollen die einzelnen SoM die benötigte Rechenleistung zur Verfügung stellen.

Ein SoM ist ein komplett funktionsfähiges und eigenständiges Computersystem, beste- hend aus einer Platine mit diversen Controllern und Bauteilen, dass von der Funktionali- tät vergleichbar mit einem SBC ist. Bei einem SBC können direkt Ein- und Ausgabegeräte wie Tastatur, Maus und Bildschirm angeschlossen werden. Dies ist bei einem SoM nicht möglich. Wie der Name schon impliziert, handelt es sich um ein *Modul*, dass gewöhnlicher- weise dafür konstruiert ist in eine Trägersystem (Carrier-Board) eingesteckt zu werden. Das SoM dient bei vielen System daher als zentrales Bauteil, dass die Rechenleistung zur Verfügung stellt. Um das SoM herum wird das Carrier-Board konstruiert, dass ty- pischerweise Anschlüsse für die zuvor genannten Ein- und Ausgabegeräte sowie weitere elektronische Schaltungen, wie zum Beispiel ADC, DAC oder Verstärker aufweist, die spezifisch für den jeweiligen Anwendungszweck benötigt werden. Durch die Verwendung mehrerer, auch unterschiedlicher, SoM in einem System ist es möglich ein leistungsstarkes und dennoch kompaktes eingebettetes System zu konstruieren.

Bei den in Frage kommenden Entwicklung-Kits der Jetson-Familie sind je nach Mo- dell verschiedene SoM von NVIDIA auf Carrier-Boards aufgesteckt, die unterschiedli- che äußere Maße und technische Spezifikationen aufweisen. Die Kits dienen häufig als Entwicklungsplattform, mit denen Entwickler Software entwickeln. Sobald die Software- entwicklung abgeschlossen ist, wird das SoM entnommen und im eigens konstruierten Carrier-Board oder eingebetteten System eingesteckt. Dadurch können sowohl bei der

Entwicklung als auch bei der Produktion eines eingebetteten Systems Kosten gespart werden. Auf allen SoM kommt Linux als Betriebssystem zum Einsatz. Eine spezielle Linux-Version (*Jetpack*) wird von NVIDIA bereitgestellt. Diese enthält bereits viele, für KI-Anwendungen, relevante Programme und Frameworks, wodurch das oftmals sehr aufwändige und langwierige Installieren von spezieller Software in weiten Teilen entfällt. Nachfolgend werden die Vor- und Nachteile der verschiedenen Jetson-Modelle vorgestellt.

5. Jetson Nano Developer Kit [41]

Der Jetson-Nano ist das leistungsschwächste, aber auch günstigste Modell der Jetson-Familie und zielt besonders auf Einsteiger beim den Themen maschinelles Lernen und neuronale Netze ab.

Vorteile:

- 128-CUDA-core Maxwell GPU.
- Alle gängigen Datenschnittstellen vorhanden.
- Geringe Leistungsaufnahme von max. 10 W.
- Im unteren Preissegment angesiedelt (109,00 € Dev-Kit, 84,50 € SoM).
- Alle üblichen KI/ML-Frameworks werden unterstützt.
- Sehr große Community- und Hersteller-Unterstützung.

Nachteile:

- Kein interner Massenspeicher, M2-SSD oder MicroSD-Karte muss verwendet werden.
- Mit bis zu 472 GFLOPS Rechenleistung im unteren Bereich des Vergleichs.
- Mechanische Abmessungen von 100 x 80 x 29 mm, die im Mittelfeld des Vergleichs liegen.

Zusammenfassend ist der Nano eine gute Plattform um günstig erste Schritte mit dem Thema GPGPU-Computing zu machen. Allerdings hat er eine eher geringe Rechenleistung, weshalb er für die Ausführung mehrerer paralleler neuronaler Netze vermutlich weniger gut geeignet ist und deswegen beim EBF nicht verwendet werden sollte.

6. und 6.1 Jetson TX2 Developer Kit [38]

Das Jetson-TX2 Developer-Kit ist mit drei verschiedenen SoM verfügbar (TX2 4GB, TX2, TX2i). Von diesen drei SoM empfiehlt NVIDIA bei Neuentwicklungen die Verwendung des TX2i-Moduls, da dies die längste zugesicherte Verfügbarkeit hat (EOL April 2028). Die drei Module unterscheiden sich nur geringfügig, z.B. in der Größe des Massen- und Arbeitsspeichers.

Vorteile:

- 256-CUDA-core Pascal GPU.
- Alle gängigen Datenschnittstellen vorhanden.
- Einstellbare Leistungsaufnahme von max. 20 W.
- Je nach Modell bis zu 32 GB Massen- und 8 GB Arbeitsspeicher.
- Alle üblichen KI/ML-Frameworks werden unterstützt.
- Sehr große Community- und Hersteller-Unterstützung.
- Kamera ist im Dev-Kit inkludiert.

Nachteile:

- Im oberen Preissegment angesiedelt (429,00 € Dev-Kit, 255,02 € bis 639,33 € SoM).
- Mit 1,26 – 1,33 TFLOPS Rechenleistung im unteren Bereich des Vergleichs.
- Vergleichsweise große mechanische Abmessungen (170 x 170 x 50 mm).

Das Jetson-TX2 Kit bietet die knapp dreifache Rechenleistung des Jetson-Nano, ist dabei allerdings gut viermal so teuer. Das Dev-Kit bietet viele Anschlussmöglichkeiten für externe Geräte und Sensoren, ist aber für die Verwendung in einem kompakten Gerät deutlich zu groß. Die Rechenleistung von 1,26 – 1,33 TFLOPS ist vergleichsweise gering und wird durch einige der zuvor genannten, deutlich günstigeren Plattformen übertroffen. Die Kosten für das empfohlene TX2i-Modul deutlich über dem Budget (639,33 €), weshalb nur die älteren SoM TX2 4GB und TX2 in Frage kommen würden.

7. Jetson Xavier NX Developer Kit [39]

Das Jetson Xavier NX Developer Kit ist äußerlich, bis auf den verwendeten Kühlkörper, nicht vom Jetson-Nano unterscheidbar. Beide nutzen das gleiche kompakte Carrier-Board, weshalb die Anschlussmöglichkeiten für Zubehör und die vorhandenen Schnittstellen gleich sind. Der große Unterschied liegt im verwendeten SoM. Beim Xavier NX kommt die neuere GPU-Architektur *NVIDIA Volta* [42] zum Einsatz und zusätzlich sind spezielle KI-Controller verbaut, wodurch bei konstanter Bauform eine deutlich größere Rechenleistung erzielt wird. Neben der deutlich stärkeren GPU ist auch die verwendete CPU deutlich leistungsstärker.

Vorteile:

- 384-CUDA-core Volta GPU + 48 TensorCores.
- Zwei NVDLA Engine, 7-Way VLIW Vision Processor.
- Alle gängigen Datenschnittstellen sind vorhanden.
- Einstellbare Leistungsaufnahme von max. 15 W.
- 8 GB Arbeitsspeicher.
- Ethernet, WLAN und Bluetooth integriert.
- Alle üblichen KI/ML-Frameworks werden unterstützt.
- Sehr große Community- und Hersteller-Unterstützung.
- Mit 21 *TFLOPS* sehr hohe Rechenleistung.

Nachteile:

- Kein interner Massenspeicher, M2-SSD oder MicroSD-Karte muss verwendet werden.
- Betriebsspannung 19 V.
- Lüfter im Kühlkörper, eventuell anfällig gegen Erschütterungen.
- Im oberen Preissegment angesiedelt (489,00 € Dev-Kit, 476,00 € SoM).
- Mechanische Abmessungen von 100 x 80 x 40 mm, liegen im Mittelfeld des Vergleichs.

Mit dem Xavier NX werden Leistungsdaten erreicht, die keine der zuvor genannten Plattformen erreicht. Durch die Volta GPU-Architektur wird bei deutlich höherer Rechenleistung weniger Energie verbraucht, was die Verwendung in einem batteriebetriebenen Gerät begünstigt. Die mechanischen Abmessungen des Dev-Kits liegen in einem annehmbaren Bereich. Der Einbau in ein mobiles, kompaktes Geräte sollte damit möglich sein. Durch die sehr hohe Rechenleistung von 21 *TFLOPS* ist das Preis-Leistungsverhältnis des NX gut, jedoch würde das Budget, bei einer Entscheidung für dieses Gerät, ausgeschöpft werden!

9. BeagleBone AI [7]

Der BeagleBone AI ist von der Baugröße vergleichbar mit dem Google Coral. Speziell für KI-Anwendungen nutzbar wird der SBC durch die verbauten *Embedded Vision Engines (EVEs)*, die auch hier für die Beschleunigung von KI-Berechnungen dienen. Um die EVE's nutzen zu können ist allerdings die Verwendung einer speziellen Software-Bibliothek unumgänglich, wodurch die Nutzung der gängigen KI/ML-Frameworks eingeschränkt wird.

Vorteile:

- Alle gängigen Datenschnittstellen vorhanden.
- Ethernet, WLAN und Bluetooth integriert.
- 16 GB Massenspeicher vorhanden, durch MicroSD-Karte erweiterbar.
- Ausführliche Dokumentation vorhanden.
- Im unteren Preissegment angesiedelt (115,80 €).
- Geringe mechanische Abmessungen (89 x 54 x 15 mm).

Nachteile:

- Keine MIPI-CSI Schnittstelle vorhanden.
- Nur 1 GB Arbeitsspeicher vorhanden.
- Keine Angaben über Rechenleistung.
- Nur eingeschränkte Nutzung von ML/KI-Frameworks.
- So gut wie keine Community-Unterstützung, da noch sehr neu am Markt.

Der BeagleBone AI ist eine interessante und eher unbekannt Alternative zu den zuvor vorgestellten Geräten. Besonders der geringe Preis lässt den SBC interessant werden. Leider lassen sich kaum Referenzen und technische Daten zu dem Gerät finden, sodass eine objektiver Vergleich nicht möglich ist. Die zwangsweise einzusetzende KI-Bibliotheken und der fehlende MIPI-CSI Kameranschluss schränkt die Funktion des SBC zusätzlich stark ein, sodass es für den Einsatz im EBF eher nicht geeignet ist.

10. Raspberry Pi 4B [48]

Das letzte Gerät im Vergleich dient mehr als Referenz für die zuvor vorgestellten Plattformen und eher weniger als mögliche Plattform für den EBF. Es handelt sich dabei um den Raspberry Pi 4B, die jüngste Version des sehr beliebten und weit verbreiteten SBC. Der Raspberry Pi ist ein linux-basierter handelsüblicher SBC ohne spezielle Technik zur Berechnung von neuronalen Netzen oder anderen Anwendungen der künstlichen Intelligenz. Er besitzt alle Bestandteile eines normalen PC's und wird oft als Ersatz für einen solchen eingesetzt.

Vorteile:

- Alle gängigen Datenschnittstellen vorhanden.
- Ethernet, WLAN und Bluetooth integriert.
- Leistungsaufnahme von max. 15 W.
- Bis zu 8 GB Arbeitsspeicher.
- Ausführliche Dokumentation verfügbar.
- Im unteren Preissegment angesiedelt (107,13 €).
- Geringe mechanische Abmessungen (85 x 56 x 15 mm).
- Sehr gute Community-Unterstützung.

Nachteile:

- Keine spezielle KI-Technik integriert.
- Geringe Rechenleistung von 13,5 GFLOPS.
- Nur eingeschränkte Nutzung von ML/KI-Frameworks möglich.
- Kein Massenspeicher integriert, MicroSD-Karte muss verwendet werden.
- Sehr gute Community-Unterstützung.

Herausstechend ist die geringe Rechenleistung von 13,5 GFLOPS. Im Vergleich belegt der Raspberry Pi in dieser Kategorie den letzten Platz und zeigt damit deutlich, dass der Einsatz von speziellen KI-Controllern und -Techniken unumgänglich ist, wenn eine hohe Rechenleistung mit einer eingebetteten Recheneinheit realisiert werden soll. Hoch optimierte neuronale Netze und Netze, die nur geringe Funktionalität bieten, funktionieren auch auf dem Raspberry Pi. Dies ist eine persönliche Erfahrung von mir, die ich im Bachelorprojekt gemacht habe. Doch für die Anwendung im EBF, bei dem der Umfang der verwendeten neuronalen Netze zum jetzigen Zeitpunkt nicht abgeschätzt werden kann, ist der Raspberry Pi eher nicht geeignet.

4.1.2 Auswahl einer Recheneinheit

Nachfolgend wird einer der in Abschnitt 4.1.1 vorgestellten Recheneinheiten ausgewählt, die im EBF als *Video-Detektor Plattform* eingesetzt werden soll. Maßgeblich bei der Auswahl ist der Vergleich der verschiedenen SBC anhand der in Abschnitt 4.1 vorgestellten Leistungskriterien. Ergebnis dieses Vergleichs ist, dass der *NVIDIA Jetson Xavier NX* für die Verwendung als eingebettete Recheneinheit im EBF ausgewählt wird.

Der Xavier NX bringt ein gutes Preis-Leistungsverhältnis und herausstechende Leistungsdaten mit sich. Mit 21 *TFLOPS* Rechenleistung ist der Xavier NX mit Abstand der leistungsstärkste SBC im Vergleich, der noch innerhalb des Budgets liegt. Dies wird vor allem durch die verbaute GPU mit 384 *CUDA* Kernen und 48 TensorKernen erreicht, aber auch die *Carmel*-CPU trägt dazu bei. Neben der Rechenleistung werden alle in der Anforderungsanalyse (Abschnitt 3.1.1) festgelegten Anforderungen erfüllt oder sogar übertroffen, wobei das Budget so gut wie komplett ausgenutzt wird. Die Integration in den inneren Aufbau des EBF (Abbildung 3.1) sollte durch die hohe Anzahl von vorhandenen Datenschnittstellen problemlos erfolgen können und das vom Hersteller bereitgestellte, linux-basierte Betriebssystem sorgt für die Nutzbarkeit von allen gängigen ML/KI-Software-Frameworks. Die Geräte der Jetson-Familie und damit auch der Xavier NX, bieten die Möglichkeit zwischen verschiedenen *PowerMode (PM)* umzuschalten. Die PM variieren die Anzahl der verwendeten CPU-Kerne und begrenzen in verschiedenen Stufen die Leistungsaufnahme des SBC. Dies ist eine interessante Funktion, die möglicherweise für die Anwendung in einem mobilen Gerät gut geeignet ist um Energie zu sparen. Dies zu untersuchen ist Bestandteil der Evaluation der Plattform. Weitere positive Aspekte sind die kleine Bauform und die Ausführung des NX als SoM. Dadurch ist es möglich, kurzfristig direkt mit dem Developer-Kit und dem darin enthaltenen Carrier-Board zu arbeiten, wodurch verhältnismäßig schnell Ergebnisse in der Softwareentwicklung zu erwarten sind.

4.2 Eingebettete Kamera

Nachdem die Entscheidung für eine Recheneinheit getroffen ist, erfolgt eine ähnlich umfassende Recherche für die auszuwählende eingebettete Kamera. Wieder werden Merkmale für den Vergleich solcher Kameras zusammengetragen und verschiedene Kameramodelle verglichen aus denen anschließend eine für die weitere Verwendung im EBF ausgewählt wird..

4.2.1 Vorstellung und Bewertung der zur Wahl stehenden Kameras

Aus Abschnitt 3.1.3 ist bekannt, dass die Auflösung der Kamera mindestens Full HD-Qualität ($1920 \times 1080 \text{ Pixel}$) betragen muss und die Bauform zu einem kompakten Gerät passen soll. Durch Recherche wird herausgefunden, dass der Öffnungswinkel und die Anschlussart der Kamera weitere Kriterien sind, anhand derer eingebettete Kameras miteinander verglichen werden können.

Gängige Anschlussarten von Kameras sind *USB*, *Ethernet* und *MIPI-CSI*, wobei *USB* und *Ethernet* meist bei Industrie-Kameras zum Einsatz kommen, die typischerweise nicht direkt an der auswertenden Einheit platziert sind. Bei diesen Systemen werden aufwändige Signalaufbereitungs- und Fehlerkorrekturverfahren angewendet, wodurch die Übertragung großer Datenmengen über eine große Distanz möglich wird. Für Kameras die nah an der auswertenden Einheit platziert werden können, wie dies beim EBF der Fall ist, wird häufig die *MIPI-CSI* Schnittstelle verwendet. Bei dieser werden Recheneinheit und Kamera über ein 15-poliges Flachbandkabel miteinander verbunden. Über dieses Kabel erfolgt sowohl die Spannungsversorgung der Kamera, als auch der Datentransfer zwischen Kamera und Recheneinheit. Durch die Vielzahl paralleler Leitungen können relative hohe Datenraten von $1,25 \text{ Gbit/s}$ bis $5,8 \text{ Gbit/s}$ [62] erzielt werden. Durch diese hohen Datenraten ist es problemlos möglich, Bilder mit hoher Auflösung und Wiederholrate zu transferieren. Die *MIPI-CSI*-Schnittstelle bietet sich für die Verwendung im EBF ebenfalls an, da am Jetson Xavier NX zwei *MIPI-CSI* Anschlüsse verfügbar sind.

Durch Analyse der Dokumentation des Xavier NX konnte festgestellt werden, dass die Produkte der Jetson-Serie standardmäßig Softwaretreiber für den Foto-Sensor *IMX291* von *Sony* [52] installiert haben. Da dieser Sensor bei vielen eingebetteten Kameras mit *MIPI-CSI*-Schnittstelle verwendet wird, stellt dies eine große Arbeitserleichterung dar, denn die Kompatibilität zwischen Kamera und Recheneinheit ist gewährleistet. Auch erfüllt der Sensor die Anforderung an die Mindestauflösung von $1920 \times 1080 \text{ Pixel}$. Aus

diesem Grund wird sich bei der Auswahl einer Kamera auf Modelle mit diesem Sensor beschränkt. Eine Aufstellung entsprechender Kameras ist Tabelle 4.2 zu entnehmen. Alle aufgeführten Kamera haben den IMX219-Sensor mit 8 *Megapixel* verbaut und werden von der Firma *Waveshare* [61] hergestellt.

Tabelle 4.2: Übersicht verschiedener MIPI-CSI Kameras, Tabelle angepasst nach [61].

Bezeichnung	Infrarot	Öffnungswinkel in <i>Grad</i>	Blende <i>F</i>	Brennweite <i>f</i> in <i>mm</i>
RPi Camera V2	Nein	62,2	2,00	3,03
RPi NoIR Camera V2	Ja	62,2	2,00	3,04
IMX219-77 Camera	Nein	77,0	2,00	2,96
IMX219-77IR Camera	Ja	77,0	2,00	2,96
IMX219-120 Camera	Nein	120,0	2,20	1,88
IMX219-160 Camera	Nein	160,0	2,35	3,15
IMX219-160IR Camera	Ja	160,0	2,35	3,15
IMX219-160 IR-CUT Camera	Ja	162,0	2,70	3,62
IMX219-170 Camera	Nein	170,0	2,00	2,2
IMX219-200 Camera	Nein	200,0	2,00	0,87
IMX219-D160	Nein	160,0	2,35	3,15

4.2.2 Auswahl einer Kamera

Einige der in Tabelle 4.2 aufgeführten Kameras arbeiten im Infrarotbereich. Dies kann beim Betrieb der Kamera in dunklen Umgebungen und bei allgemein schlechten Lichtverhältnissen hilfreich sein, da eine Infrarotkamera Licht in einem anderen Frequenzband wahrnimmt als Normalbildkameras. In dieser Arbeit soll eine Normalbild Kamera verwendet werden, allerdings ist die Option einer parallel vorhandenen Infrarotkamera für die weitere Entwicklung des EBF möglicherweise vorteilhaft. Die Integration einer zweiten Kamera wäre möglich, da die Recheneinheit zwei MIPI-CSI nschlüsse hat. Das für diese Arbeit relevanteste Unterscheidungsmerkmal der aufgeführten Kameras ist der Öffnungswinkel. Natürlich müssen die Blende F und die Brennweite f ebenfalls Beachtung finden, allerdings unterscheiden sich diese bei den einzelnen Kameras nicht so stark wie der Öffnungswinkel. Je größer der Öffnungswinkel ist, desto größer ist der Aufnahmebereich

der Kamera. Für eine Objekterkennung, die Objekte in Richtung der aktuellen Bewegungsrichtung des Nutzers erkennen soll, ist ein großer Aufnahmebereich nicht sinnvoll. Dadurch werden möglicherweise Objekte wahrgenommen, die den Nutzer nicht betreffen. Auf der anderen Seite ist auch ein zu geringer Öffnungswinkel zu vermeiden, da der Aufnahmebereich zu klein werden könnte und so relevante Objekte nicht erkannt werden könnten. Aus diesem Grund ist die Wahl auf die *IMX219-77* Kamera gefallen. Diese besitzt einen Öffnungswinkel von 77° und liegt damit im mittleren Bereich der möglichen Öffnungswinkel.

Die Kamera erfüllt alle Anforderungen aus Abschnitt 3.1.3. Die maximale Auflösung liegt bei $3280 \times 2464 \text{ Pixel}$, was einer *4K*-Auflösung entspricht [19]. Durch die geringen Abmessungen von $25 \times 24 \text{ mm}$ lässt sich die Kamera in ein kompaktes Gehäuse integrieren. Ob der gewählte Öffnungswinkel zur Anwendung im EBF passt, muss durch Feldtests valisiert werden. Mit knapp 16 € ist die Kamera jedoch so preiswert, dass sie problemlos gegen eine Kamera mit einem anderem Öffnungswinkel ausgetauscht werden kann, wenn sich herausstellen sollte, dass der aktuell gewählte Öffnungswinkel nicht passend ist.

4.3 Detektionsalgorithmus

Neben den Hardwarekomponenten, soll auch ein Algorithmus zur Objektdetektion ausgewählt werden, mit dem die Hardware evaluiert, sowie getestet wird und der später im EBF eingesetzt werden kann.

Wie in den Abschnitten 2.2.1 und 2.2.2 beschrieben, kommen für eine solche Anwendung grundsätzlich einstufige- (SSD) oder zweistufige Detektoren (TSD) in Frage. Auf Grund der Tatsache, dass der EBF im dynamischen Alltag eingesetzt werden soll, indem spontan Situationen auftreten können, die in Echtzeit richtig eingeschätzt werden müssen, ist die Verwendung eines SSD unumgänglich. Nur mit dem einstufigen Detektor lassen sich Bilder in Echtzeit verarbeiten und mögliche Gefahrensituationen erfolgreich erkennen. Nachfolgend wird zunächst genauer die Funktionsweise von einstufigen Detektoren beleuchtet. Anschließend wird ein Algorithmus ausgewählt und die Funktionsweise sowie Besonderheiten von diesem vorgestellt.

4.3.1 Bestandteile und Funktionsweise eines Single Stage Detektors

Ein SSD hat die Aufgabe, Objekte in Bildern zu erkennen und diese im Bild zu lokalisieren und zu markieren. Die Zusammenfassung dieser Aufgaben wird *Detektion* bzw. *detektieren* genannt. Es müssen sowohl einzelne Objekte, als auch Objekte, die sich gegenseitig überlappen detektiert werden [24]. Zwei Beispiele von zu detektierenden Objekten sind in Abbildung 4.1 dargestellt.

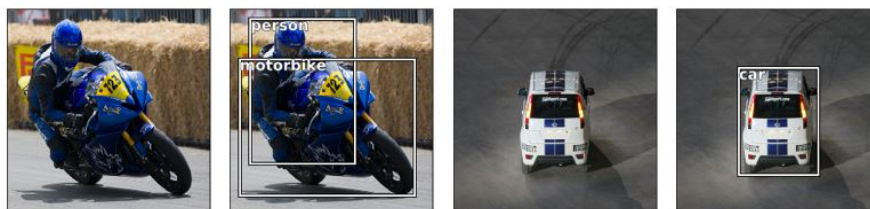


Abbildung 4.1: Beispiellbilder aus dem PASCAL VOC Datensatz [24].

Die Objekte werden durch Rechtecke, sogenannte *bounding boxes*, im Bild markiert. Die Rechtecke, die in den Bildern zu sehen sind, werden *ground truth box* genannt und kennzeichnen die *wahren* Objekte eines Bildes. Beim Training eines SSD dienen diese als Referenz, an Hand derer das neuronale Netz des Detektors optimiert wird. Dabei gilt: Je

näher die vorhergesagten Koordinaten der *bounding boxes* an die Koordinaten der textitground truth box heran reichen, desto besser ist die Optimierung des Netzes.

Der Aufbau eines SSD ist in Abschnitt 2.2.1 bereits grob beschrieben worden. Um einen genaueren Einblick zu erhalten, wird der in Abbildung 4.2 dargestellte Aufbau eingeführt, der sowohl für SSD als auch für TSD gültig ist [9]. Ein TSD hat, im Vergleich zum SSD einen zusätzlichen Funktionsblock, der hier ganz rechts außen dargestellt ist. Im Folgenden wird sich auf die Funktionsblöcke innerhalb des grün gestrichelten Rechteckes konzentriert, mit denen die Funktion eines SSD erfüllt wird.

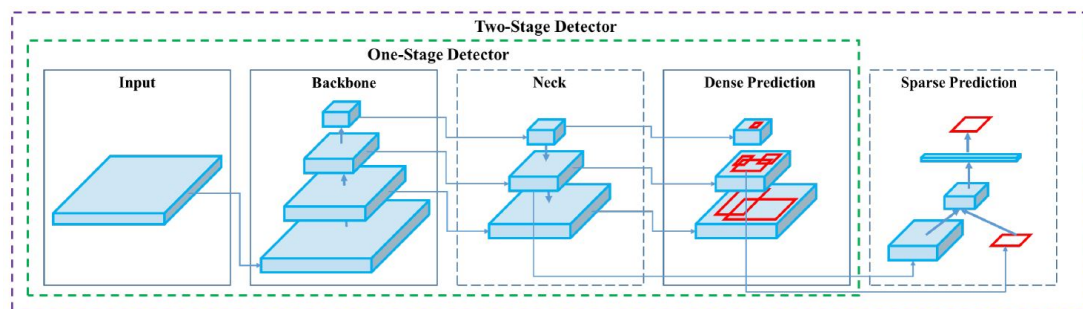


Abbildung 4.2: Aufbau von Detektor-Netzwerken, am Beispiel von YOLOv4 [9].

Der erste Funktionsblock heißt *Input* und stellt den Eingang des Bilddetektors dar. Hier werden die Bilddaten an den Detektor übergeben. Dies kann als Einzelbild erfolgen, aber auch ganze Bildsätze oder Bildpyramiden sind möglich. Der darauf folgende Block *Backbone* ist das zentrale Element für die Extraktion der Features aus den Bilddaten. Wie in Abschnitt 2.1.2 beschrieben, wird dazu ein CNN verwendet. Bei den meisten Bilddetektoren wird für den Backbone ein eigenständiger Bild-Klassifikationsalgorithmus verwendet, an dessen Ende normalerweise ein oder mehrere fully connected layer zu finden sind. Dadurch stehen am Ende des CNN die Klassifikationsergebnisse der Bilder zur Verfügung. Da der Algorithmus hier jedoch nur als Feature-Extraktor verwendet werden soll, an dessen Ende Feature-Maps stehen sollen, werden die fully connected layer weggelassen, bzw. entfernt. Anschaulich gesprochen, wäre es am Ende des Backbones möglich mehrere Bounding Boxes um das gesamte Bild zu zeichnen, die jeweils für ein klassifiziertes Objekt im Bild steht [23]. Da beim SSD jedoch auch die Position des Objektes bestimmt werden soll, sind weitere Verarbeitungsschritte nötig. Entgegen der Darstellungen in Abschnitt 2.2.1 und 2.2.2, folgt auf den Backbone der *Neck*.

Der Neck ist ein beim Detektornetzwerk *YOLOv4* neu eingeführter Funktionsblock [9],

der als Eingangsgröße die Feature-Maps, die vom Backbone generiert werden zugeführt bekommt. Nicht jeder Bilddetektor hat einen expliziten Neck, sondern meist tragen die Schichten zwischen Backbone und *Dense Prediction* keinen besonderen Namen. So zum Beispiele beim *FCOS* Detektor [57]. Da *YOLO* jedoch ein weit verbreiteter Bilddetektor ist, darf die Erwähnung dieses Funktionsblockes nicht fehlen. *YOLO* wird in Abschnitt 4.3.3 näher beschrieben.

Der innere Aufbau des Neck kann stark variieren, Ziel ist es die Ergebnisse der einzelnen Backbone-Schichten auf bestimmte Art und Weise miteinander zu vermischen und zu kombinieren, sodass die in den einzelnen Feature-Maps enthaltenen Features komprimiert und zentral angehäuft werden [45]. Dies bereitet die eigentliche Detektion der Objekte vor. Realisiert wird dies durch die geschickte Verwendung verschiedenster neuronaler Netze.

Den Abschluss eines SSD bildet der Funktionsblock *Head*, der in Abbildung 4.2 *Dense Prediction* genannt wird. Im Head erfolgt die eigentliche Detektion der Objekte, das heißt die erkannten Objekte werden einer Klasse zugeordnet (z.B. Auto, Mensch, Ball) und die Position wird ermittelt. Für diese beiden Aufgaben wird, wie in Abschnitt 2.2.1 erwähnt, ein einziges Netzwerk verwendet, das die eigentlichen Vorhersagen trifft. Alle vorigen Schritte sind vorbereitenden Schritte, die nach belieben variiert werden können und, wie zuvor beschrieben, meist auf fertig trainierten neuronalen Netzen basieren. Bei den meisten Bilddetektoren wird jedoch eine favorisierte Anordnung von Backbone, Neck und Head angegeben, mit dem der Detektor getestet worden ist.

Zur abschließenden Detektion erzeugt der Head für jedes Objekt einen Vektor, der die Koordinaten der Bounding Box, die Vorhersagegenauigkeit (*AP*) der Klasse und die Klasse des jeweiligen Objektes selbst beinhaltet. Die genaue Berechnung dieser Werte ist sehr komplex und basiert auf mathematischen Funktionen und Algorithmen, die je nach SSD variieren. Da diese Arbeit nicht das Ziel verfolgt einen Bilddetektor zu optimieren, werden die teilweise sehr umfangreichen und komplexen Berechnungen innerhalb der einzelnen Funktionsblöcke nicht weiter beleuchtet.

Zuvor sind die Funktionsblöcke als eigenständige neuronale Netze bezeichnet worden. Für einen Bilddetektor werden jedoch schlussendlich alle Netzwerke bzw. Schichten aneinandergereiht, sodass ein neues, großes Netzwerk entsteht, dass als SSD bezeichnet wird.

Die Recherche zum Thema SSD hat gezeigt, dass aktuell viele verschiedene Implementierungen eines SSD verfügbar sind. Der Aufbau dieser Detektoren entspricht in weiten Teilen dem in Abbildung 4.2 dargestellten Aufbau. Allerdings weisen die SSD in der genauen Implementierung teilweise deutliche Unterschiede auf, sodass sich deren Ergebnisse

in Bezug auf die Verarbeitungsgeschwindigkeit und Vorhersagegenauigkeit unterscheiden. Die Autoren von *YOLO* in der vierten Version haben eine aussagekräftige Tabelle (siehe Abbildung A.1) erstellt, in der verschiedene SSD auf genau diese beiden Parameter hin untersucht worden sind. Dabei sind die Modelle jeweils mit dem *MS COCO* [14] Datensatz trainiert und getestet worden, mit dem die Detektion von 80 verschiedenen Objektklassen möglich ist. Als Recheneinheit wird eine NVIDIA Grafikkarte mit Volta-Architektur verwendet, die der ausgewählte Jetson Xavier NX ebenfalls aufweist. Deshalb lassen sich die Ergebnisse dieser Tabelle gut als Referenz nutzen.

4.3.2 Auswahl eines Algorithmus

Aus der erwähnten Tabelle 10 in [9] geht hervor, dass der Bilddetektor *You Only Look Once* (*YOLO*) die im Vergleich besten Ergebnisse liefert. Speziell die Version vier (*YOLOv4*) liegt sowohl bei der Verarbeitungsgeschwindigkeit (max. 96 *FPS*) als auch bei der Vorhersagegenauigkeit (max. $AP_{50} = 65,7\%$) vor allen anderen untersuchten Detektoren. Diese beiden Werte bilden einen Kompromiss aus Geschwindigkeit und Genauigkeit, der für die zukünftige Entwicklung und Verwendung des EBF mutmaßlich passend ist. Auch ist aus vorigen Arbeiten und persönlichen Vorkenntnissen bekannt, dass für *YOLO* eine sehr breite Community-Unterstützung besteht und, dass die Modelle sowohl auf eingebetteten Recheneinheiten als auch auf Servern und PC's ausgeführt werden können. Damit erfüllt *YOLO* alle Anforderungen, die in der Anforderungsanalyse festgelegt wurden, weshalb *YOLO* als Objektdetektor für die nachfolgende Evaluierung der Hardware und die zukünftige Verwendung im EBF ausgewählt.

Nachfolgend wird die Funktionsweise und Bestandteile von *YOLO* am Beispiel von *YOLOv4* näher erläutert. Es wird betrachtet welche Netze für Backbone, Neck und Head verwendet werden und wie *YOLO* aus einem Eingangsbild die Detektionsergebnisse mit den sogenannten *Bounding Boxes* generiert.

4.3.3 Funktionsweise und Bestandteile des Bilddetektors YOLO

Erstmals ist *YOLO* in 2015 [49] von vier US-amerikanischen Wissenschaftlern vorgestellt worden. Zum damaligen Zeitpunkt stellte *YOLO* eine grundlegende Neuerung in der auf faltenden neuronalen Netzen basierenden, schnellen Bildverarbeitung dar. Bis zu diesem Zeitpunkt ist vor allem die Entwicklung von TSD vorangetrieben worden, die zwar meist eine hohe Vorhersagegenauigkeit haben, aber vergleichsweise langsam sind. Im April 2020

ist die bisher (offiziell) letzte Version *YOLOv4* [9] erschienen, die stark auf der dritten Version (*YOLOv3*) basiert und eine spezielle Architektur mit sich bringt. Anhand von *YOLOv4* werden die grundlegende Funktionsweise und die verschiedenen Netze dargestellt, die für die Detektion von Objekten verwendet werden.

Der Aufbau von *YOLOv4* ist bereits mit der Abbildung 4.2 dargestellt worden. Neu bei *YOLOv4* ist die explizite Erwähnung des Neck, der bei vorigen Versionen in Form verschiedener neuronaler Schichten bereits vorhanden war, allerdings nicht eindeutig einem Funktionsblock zugeordnet wurde.

YOLO ist ein auf *anchor boxes* (Anker-Boxen) basierender Bilddetektor. Dies bedeutet, dass bei einer Detektion, nachdem das Eingangsbild mit dem Input-Block an den Backbone übergeben wurde, ein Gitter über das Eingangsbild gelegt wird, wodurch es in einzelne Zellen eingeteilt wird (siehe Abb. 4.3 links). Pro Zelle wird eine bestimmte Anzahl von Anker-Boxen mit unterschiedlichen Größen erzeugt (siehe Abb. 4.3 mitte, oben), wobei jede einzelne Box ein Wert mit sich trägt, der angibt wie stark die Anker-Box ein Objekt repräsentiert. Dieser Wert wird *objectness p_{Obj}* genannt. Ein objectness-Wert von $p_{Obj} = 0$ bedeutet, dass die Box kein Objekt überdeckt, der Wert $p_{Obj} = 1$ bedeutet, dass die Box eindeutig ein Objekt, oder ein Teil eines Objektes überdeckt. Zwischen 0 und 1 kann jeder Wert angenommen werden. Im gleichen Zuge wird jede Zelle des Bildes einer Klasse (Objekt) zugeordnet (siehe Abb. 4.3 mitte, unten). Dieser Vorgang wird an drei verschiedenen Stellen innerhalb des Detektor-Netzwerkes durchgeführt. An diesen drei Stellen hat das Eingangsbild, bzw. die daraus generierte Feature-Map's verschiedene Auflösungen, sodass auch sehr große und sehr kleine Objekte möglichst gut detektiert werden können.

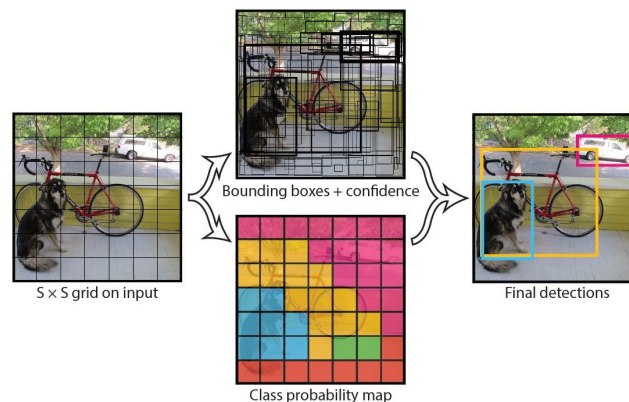


Abbildung 4.3: Schematische Darstellung des YOLO-Verfahrens [49].

Neben dem objectness-Wert werden für jede Anker-Box die relativen Koordinaten bestimmt, die sich auf die linke obere Ecke des Eingangsbildes beziehen. Beim Training des Detektors ist es dadurch möglich die Bestimmung der schlussendlich optimalen *Bounding Box* (umgebende Box) als Regressionsproblem zu behandeln. Hierzu werden in den Trainingsdaten ground truth boxen eingezeichnet, die die tatsächlich enthaltenen Objekte innerhalb eines Bildes markieren. Während des Trainings nähert sich das Netzwerk durch verschiedene mathematische Funktionen an diese ground truth boxen an. Die Abbildung 4.1 zeigt Bilder mit markierten ground truth Boxen.

Am Ende des YOLOv4-Netzwerkes hat der Algorithmus die mehreren Tausend Anker-Boxen pro Bild so weit verarbeitet, dass nur noch die tatsächlich im Bild enthaltenen Objekte mit Klasse, Klassenwahrscheinlichkeit und Koordinaten beschrieben und markiert werden. Ein solches Ergebnis ist in Abbildung 4.3 rechts dargestellt. Da YOLO in dieser Arbeit angewendet und nicht optimiert werden soll, ist es nicht nötig die mathematischen Funktionen, die innerhalb des Modells verwendet werden näher zu beleuchten. Kurz erläutert wird jedoch, welche Netze bei YOLOv4 für die in Abschnitt 4.3.1 dargestellten Funktionsblöcke verwendet werden. Als Backbone kommt das Netzwerk *CPS-Darknet53* zum Einsatz, welches eine Weiterentwicklung des in *YOLOv3* [50] verwendeten Backbone *Darknet53* ist. Das Netzwerk hat 53 Schichten, die größtenteils faltend sind und für die Extraktion der Feature's aus dem Eingangsbild zuständig sind. Neu ist, dass die im generierten Feature-Map's an mehreren Stufen des Backbones in zwei Teile aufgespalten werden, von denen ein Teil direkt zur nächsten faltenden Schicht geführt wird und der andere Teil einen sogenannten *Dense Block* durchläuft und erst dann wieder dem ursprünglichen Pfad zugeführt wird. Diese Art der Verbindung innerhalb eines NN wird *Cross-Stage-Partialconnections (CSP)* genannt und steigerte die Modell-Robustheit und reduziert den Rechenaufwand [59].

Im Neck wird das *Path Aggregation Network (PAN)* [31] in Verbindung mit *Spartial Pyramid Pooling Layer* [20] verwendet. Das PAN verarbeitet die Feature-Map's vom Backbone in verschiedenen Auflösungen. Dabei werden die Map's zunächst schrittweise pyramidenförmig verkleinert und anschließend wieder vergrößert. An bestimmten Punkten dieses Vorgangs werden bestimmte Feature-Map's entnommen und miteinander kombiniert, bevor sie weiterverarbeitet werden. Dies steigert die Robustheit des Modells gegenüber Eingangsbildern mit verschiedenen Auflösungen. Durch das häufige skalieren der Feature-Map's gehen zwangsläufig Informationen verloren. Diesem Effekt wird durch die Verwendung der SPP Layer entgegen gewirkt, indem Abkürzungen innerhalb des Netzes eingefügt werden, die zum Beispiel eine hochauflösende Feature-Map vom Anfang des Netzes zu einer weiter hinten im Netz liegenden Schicht weiterleitet. Dadurch wird auch

an Stelle des Netzes Informationen verarbeitet, die normalerweise nur zu Beginn des Netzes zur Verfügung stehen.

Den Abschluss des Detektors bildet der Head. Hier wird die selbe Anordnung von Schichten verwendet wie bei YOLOv3 [50], die für die abschließende Detektion der Objekte zuständig sind. Die Grafik 4.4 ist im Zuge der YOLOv4-Entwicklung entstanden und zeigt die Überlegenheit hinsichtlich Vorhersagegenauigkeit und Verarbeitungsgeschwindigkeit, die hier in *FPS* ausgedrückt wird, von YOLOv4 gegenüber anderen Algorithmen zur Objektdetektion.

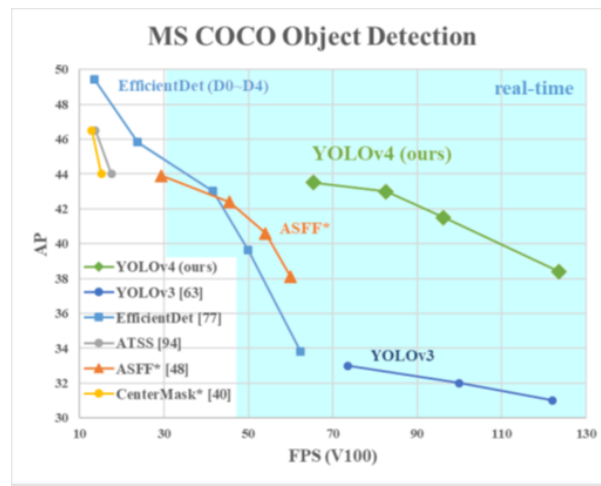


Abbildung 4.4: Vergleich verschiedener Bilddetektoren im Hinblick auf die Vorhersagegenauigkeit (*AP*) und die Verarbeitungsgeschwindigkeit (*FPS*) [9].

Deutlich zu erkennen ist die hohe Verarbeitungsgeschwindigkeit der YOLO-Modelle. Sowohl YOLOv3 als auch YOLOv4 heben sich deutlich ab. Der Zugewinn bei YOLOv4 liegt in der Vorhersagegenauigkeit. Diese liegt nun in einem zeitgemäßen Bereich und kann sich mit großen TSD-Netzwerken messen. Durch die Kombination aus der hohen Geschwindigkeit und Vorhersagegenauigkeit eignet sich YOLOv4 mutmaßlich sehr gut für die Verwendung im EBF. In wie weit diese Auswahl bestätigt werden kann, wird in Abschnitt 6.1.1 untersucht.

4.4 Die *Video-Detector Plattform* als *Black Box*

Wie in Abschnitt 3.1.1 beschrieben, ist es sinnvoll die Auswertung der Ergebnisse der Objektdetektion auf der *Motion&Sensor Plattform* durchzuführen. Der große Vorteil der dadurch entsteht ist, dass die *Video-Detector Plattform* für die Weiterentwicklung als *Black Box* angesehen werden kann. Eine *Black Box* ist in technischem Kontext ein Gerät oder System, dessen Inhalt und genaue Funktionsweise für Außenstehende unbekannt ist [67]. Nur die Ein- und Ausgabe-Schnittstellen sind bekannt. Eine *Black Box* arbeitet also eigenständig wohldefinierte Aufgaben ab, ohne, dass ein technisches Verständnis vorhanden sein muss. Bei dieser Definition handelt es sich um eine eigens für diese Anwendung getroffene Aussage.

Im Zusammenhang mit dem EBF kann die *Video-Detector Plattform* als *Black Box* bezeichnet werden, wenn auf ihr, unabhängig von anderen Systemkomponenten, die Objektdetektion mit Ausgabe der Ergebnisse auf einer passenden Schnittstelle ausgeführt werden kann. Die *Video-Detector Plattform* kann dann in näherer Zukunft als eigenständiges System betrieben und verwendet werden, sodass sich auf die Auswertung der übermittelten Ergebnisse und die Weiterverarbeitung von diesen konzentriert werden kann. Die Auswertung der Ergebnisse erfolgt dann ausschließlich auf der *Motion&Sensor Plattform*. Es muss lediglich eine Schnittstelle beschrieben und implementiert werden, um mit der *Video-Detector Plattform* Daten austauschen zu können. Dabei müssen alle zur Verfügung stehenden Informationen zu den Detektionsergebnissen an die *Motion&Sensor Plattform* übermittelt werden, um größt mögliche Flexibilität für weitere Verarbeitungsschritte auf der *Motion&Sensor Plattform* zu garantieren. Denkbar wäre zum Beispiel eine akustische Ausgabe auf einen Lautsprecher oder Kopfhörer, in der die Umgebung in Form von Objekten mit Angabe der Richtungen, in der die Objekte detektiert worden sind, beschrieben wird.

Mit Fortschritt der Entwicklung des EBF muss der Zustand der *Black Box* jedoch verlassen werden, da auf der *Video-Detector Plattform* sehr wahrscheinlich weitere komplexe Netze und Algorithmen implementiert werden, um alle Funktionen des EBF zu realisieren. Ab diesem Zeitpunkt muss wieder aktiv mit der *Video-Detector Plattform* gearbeitet werden.

4.5 Serielle Kommunikationsschnittstelle

Wie in Abschnitt 3.1.1 festgelegt, werden die Aufgaben der Objektdetektion auf die *Motion&Sensor Plattform* und *Video-Detector Plattform* aufgeteilt. Für den Datenaustausch zwischen den beiden Plattformen bietet sich die Verwendung einer seriellen Schnittstelle an. Durch die geschickte Nutzung von vorgefertigten und ausgiebig getesteten Software-Bibliotheken, ist die Implementierung einfach. Der Xavier NX stellt dazu folgende Schnittstellen zur Verfügung: SPI, UART und I²C. Der Mikrocontroller der *Motion&Sensor Plattform* lässt ebenfalls die Kommunikation über diese drei Schnittstellen zu, sodass aus den drei genannten Schnittstellen frei gewählt werden kann. Alle Schnittstellen stellen keine besonderen Anforderungen an die verwendeten Leitungen und Steckverbinder, allerdings unterscheidet sich die Art der Datenübertragung voneinander. Da lediglich eine einzige Punkt-zu-Punkt Verbindung zwischen *Motion&Sensor Plattform* und *Video-Detector Plattform* innerhalb des EBF realsiert werden soll, wird die UART-Schnittstelle gewählt.

Wie in Abschnitt 2.5 beschrieben, ist UART genau für eine solche Verbindung zwischen zwei definierten Komponenten konzipiert. Durch die zwei festgelegten Systemkomponenten entfällt bei der Initialisierung einer Datenübertragung die Übermittlung der Empfängeradresse, wodurch der maximale Datendurchsatz auf bis zu 3 *Mbit/s* erhöht wird. Mit einer solchen Datenrate ist auch die Übertragung umfangreicher Detektionsergebnisse möglich. UART bietet den, im Vergleich der drei Schnittstellen, geringsten Verdrahtungsaufwand von nur drei Leitungen: Rx, Tx und GND. Auf Grund der asynchronen Datenübertragung ist es nicht nötig einen Takt zwischen Sender und Empfänger zu übertragen. Die Synchronisierung der beiden Komponenten erfolgt dynamisch anhand des übertragenen Start-Bits, welches am Anfang eines jeden übertragenen Datenpaketes steht. Das Format des Datenpaketes wird, genau wie die Übertragungsgeschwindigkeit, in beiden Komponenten fest implementiert, wodurch eine systematische Auswertung der Detektion-Ergebnisse auf der *Motion&Sensor Plattform* möglich ist.

Die Verbindung mittels der UART-Schnittstelle ist, wie in Abbildung 4.5 dargestellt, bidirektional. Dabei bestehen getrennte Leitungen für das Senden (Tx) und Empfangen (Rx) von Daten. Die Kommunikation in beide Richtungen ist bei der Verwendung im EBF notwendig, damit der spätere Nutzer per *Knopfdruck* Daten über die aktuelle Situation, in der er sich befindet, anfordern kann. Der *Knopfdruck* kann dabei auf verschiedene Arten erfolgen, denkbar wäre ein Sprachbefehl oder ein physisches Betätigen eines Tasters. Sowohl das benötigte Mikrofon, als auch der Taster werden dabei an die *Motion&Sensor*

Plattform angeschlossen, die die Anforderung über die serielle Schnittstelle an die *Video-Detektor Plattform* übermittelt, welche anschließend die aktuellen Detektionsergebnisse wiederum an die *Motion&Sensor Plattform* sendet.

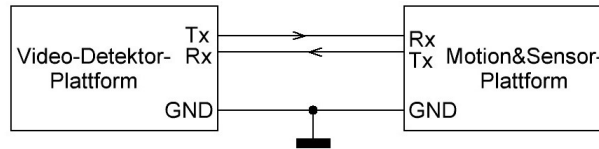


Abbildung 4.5: Blockschaltbild der UART-Verbindung im EBF

4.6 Zusammenfassung der ausgewählten Komponenten

Abschließend werden noch einmal alle ausgewählten Komponenten aufgeführt:

- Recheneinheit: NVIDIA Jetson Xavier NX.
- Kamera: Waveshare IMX219-77.
- Detektornetzwer: YOLO.
- Serielle Schnittstelle: YOLO.

5 Praktische Realisierung und Implementierung der Komponenten

In diesem Abschnitt werden die ausgewählten Komponenten näher vorgestellt, in Betrieb genommen und ein Funktionstest durchgeführt. Außerdem wird die Schnittstelle zum Datenaustausch zwischen *Motion&Sensor Plattform* und *Video-Detector Plattform* beschrieben, implementiert und in Betrieb genommen.

5.1 Eingebettete Recheneinheit und Kamera

Zunächst wird die in Abschnitt 4.1.2 und 4.2.2 ausgewählte Recheneinheit und Kamera vorgestellt, eingerichtet und in Betrieb genommen. Um die Funktionsfähigkeit der Komponenten festzustellen, wird jeweils ein Funktionstest durchgeführt.

5.1.1 Inbetriebnahme

Der ausgewählte Jetson Xavier NX wird fertig montiert, samt Netzteil, geliefert. Das Netzteil hat eine Ausgangsspannung von $U_{Out} = 19,0 V$. Das Carrier-Board ist so konstruiert, dass die Versorgungsspannung U_B zwischen $9 V$ und $19 V$ liegen kann. Die Spannung von $19 V$ ist typisch bei der Versorgung von Laptops, die oft eine hohe Leistungsaufnahme haben. Durch die relativ hohe Spannung reduziert sich, nach dem Ohmschen Gesetz, die Stromaufnahme, wodurch geringere Kabelquerschnitte, bzw. dünnere Leiterbahnen verwendet werden können. Das Xavier-NX-SoM hingegen benötigt nur eine für PC-Hardware typische Versorgungsspannung von $U_B = 5,0 V$ [15]. Daraus folgt, dass auf dem Carrier-Board ein Spannungsreglerschaltung zur Generierung der benötigten Spannungen implementiert ist. Dies schafft Spielraum für die Verwendung des Developer-Kits im EBF, da die Versorgungsspannung, die höchstwahrscheinlich von einem Akku bereitgestellt wird, frei zwischen $9 V$ und $19 V$ liegen kann.

Die nachfolgende Abbildung 5.1 zeigt das Carrier-Board mit montiertem Xavier-NX-SoM und allen Anschlüssen.

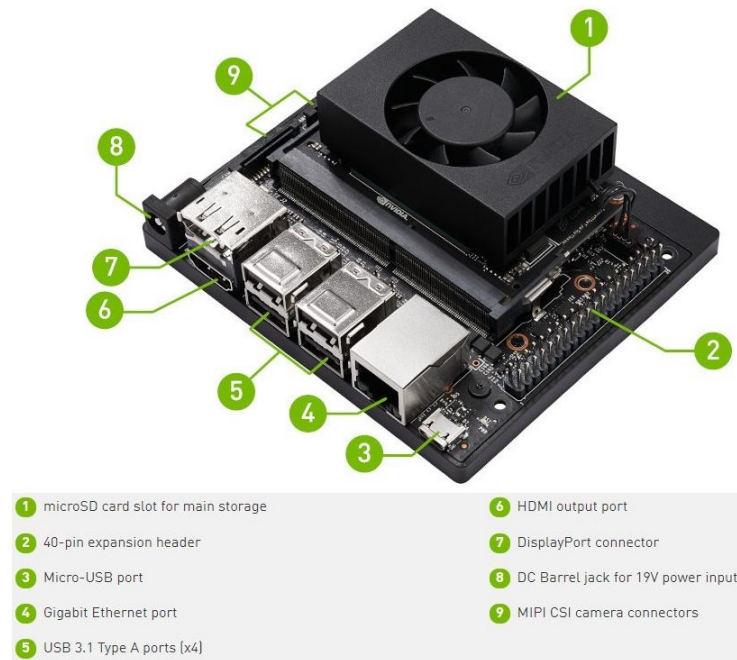


Abbildung 5.1: Der NVIDIA Jetson Xavier NX im Überblick [35].

Nicht direkt ersichtlich ist der Steckplatz für die microSD-Karte, der auf der Unterseite des SoM platziert ist. Wie in Abschnitt 4.1.1 beschrieben, wird die microSD-Karte nur bei der Entwickler-Version des Xavier-NX-SoM als Boot- und Speichermedium benötigt. Bei der Fabrikation-Version sind standardmäßig 16 GB Flash-Speicher integriert, sodass die Speicherkarte entfallen kann. Um den Xavier-NX in Betrieb zu nehmen, wird das von NVIDIA bereitgestellte Betriebssystem *JetPack* in der Version 4.4.1 als Image auf die Speicherkarte geschrieben und diese anschließend in den genannten Steckplatz eingesteckt. Gestartet wird die Recheneinheit durch Anschließen der Betriebsspannung. Dabei sollten Maus, Tastatur und Bildschirm über USB und HDMI angeschlossen sein, um mit der grafischen Oberfläche arbeiten zu können. Dies ist jedoch nicht zwingend notwendig, der Xavier NX kann auch *headless*, also konsolenbasiert, verwendet werden. Nachfolgend wird aber im normalen Betrieb mit grafischer Ausgabe gearbeitet. Nach einigen Einstellungen und Konfigurationen im Betriebssystem ist die Recheneinheit einsatzbereit. Um die ausgewählte Kamera IMX219-77 in Betrieb zu nehmen, muss diese lediglich mit

dem Flachkabel an einen der MIPI-CSI Anschlüsse (Abbildung 5.1, Nr. 9) der Recheneinheit angeschlossen werden. Da keine aktive Kamera verwendet, können an ihr direkt keine Einstellungen vorgenommen werden. Etwaige Softwareeinstellungen werden als Parameter an entsprechende Kamera-Dienstprogramme übergeben, die auf der Recheneinheit installiert sind. Die Abbildung 5.2 zeigt die Kamera mit angeschlossenem Flachbandkabel.

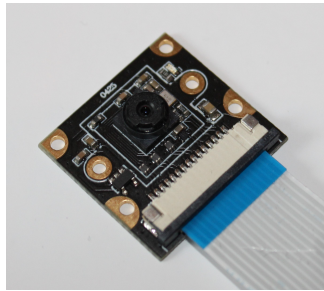


Abbildung 5.2: Die Kamera mit angeschlossenem Flachbandkabel.

Der Aufbau der Kamera ist einfach. Es handelt sich dabei um eine Platine, auf der der Fotosensor und die benötigte Beschaltung verlötet ist. Die Abbildung 5.3 zeigt die Recheneinheit mit angeschlossener Kamera. Damit ist der Hardwareaufbau und die Inbetriebnahme abgeschlossen. Mit diesem eingebetteten System wird nachfolgend gearbeitet.

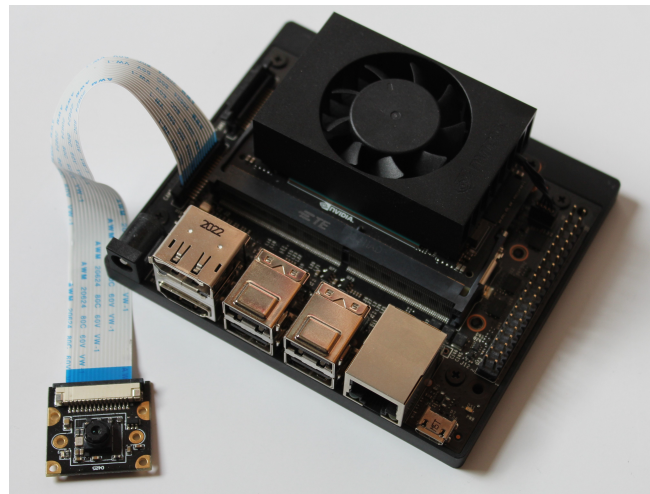


Abbildung 5.3: Der Xavier NX und die Kamera im System.

5.1.2 Funktionstest der Recheneinheit

Um die Funktion der Recheneinheit zu testen wird der Xavier-NX zunächst als einfacher PC verwendet und typische Anwendungen wie ein Internetbrowser, Textverarbeitungsprogramm und Bildbetrachtungsprogramm ausgeführt. Alle Programme funktionieren wie an einem normalen PC, womit die allgemeine Funktionsfähigkeit gegeben ist. Um nun die Funktion im Hinblick auf die Berechnung von neuronalen Netzen zu testen, wird ein von NVIDIA bereitgestelltes Benchmark-Programm [37] ausgeführt. Das Programm berechnet verschiedene neuronale Netze. Als Ergebnisse wird ausgegeben wie viele *Frame per second (FPS)* im Durchschnitt berechnet werden konnten. Die Frames haben je nach Modell unterschiedliche Größen (siehe Tabelle 5.1). Die Ergebnisse des Tests werden in der nachfolgenden Tabelle mit den Angaben des Herstellers [37] verglichen.

Tabelle 5.1: Vergleich der Ergebnisse des Benchmark-Test.

Modell	$FPS_{Original}$	FPS_{Test}	Abweichungen in %
Inception V4 (299x299)	320	320,24	0,08
VGG-19 (224x224)	67	66,09	-01,36
Super Resolution (481x321)	164	164,88	0,54
Unet (256x256)	166	145,66	-12,25
OpenPose (256x456)	238	238,14	0,06
Tiny YOLO V3 (416x416)	607	602,89	-0,68
ResNet-50 (224x224)	824	874,75	05,80
SSD Mobilenet-V1 (300x300)	909	906,54	-0,27

Der Vergleich zeigt, dass die meisten Ergebnisse den Angaben des Herstellers, mit minimalen Abweichungen, entsprechen. Auffällig sind jedoch die relativ großen Abweichungen bei den Modellen Unet ($-12,25\%$) und ResNet-50 ($+5,80\%$). Der Grund dafür konnte nicht eindeutig bestimmt werden. Möglicherweise wurde der Test beim Hersteller mit anderen Umgebungsbedingungen durchgeführt. Dazu sind allerdings leider keine Informationen vorhanden. Weitere Testläufe haben gezeigt, dass die Ergebnisse nicht konstant sind, sondern immer ein wenig variieren.

Aus diesem Grund wird festgestellt, dass die Funktion der Recheneinheit vollumfänglich gegeben ist. Zu erwähnen ist, dass bei der Ausführung des Benchmark-Programms automatisch der *Power Mode 0* eingeschaltet wird. Nähere Erklärungen dazu, werden in Abschnitt 6.1.3 getätigt.

Der Funktionstest der Kamera ist simpel und erfolgt durch den Aufruf des Dienst-Programmes *gst-launch*. Der Aufruf ist nachfolgend abgedruckt.

```
1  gst-launch-1.0 nvarguscamerasrc sensor_mode=0 ! 'video/x-raw(memory:NVMM),...
2      width=3820, height=2464, framerate=21/1, format=NV12' !...
3      nvvidconv flip-method=2 ! 'video/x-raw,width=960, height=616' !...
4      nvvidconv ! nvegltransform ! nveglglessink -e
```

Listing 1: Funktionsaufruf für einen Kamera-Livestream.

Das Programm baut eine sogenannte *Pipeline* zur Kamera auf, wodurch die Bilder als Livestream in einem Fenster angezeigt werden. Der Livestream funktioniert problemlos, womit die Funktionsfähigkeit der Kamera bewiesen ist.

Damit ist der Funktionstest des Gesamtsystems erfolgreich abgeschlossen und dieses bereit für die Verwendung.

5.2 Objektdetektion mit YOLO

Für die nachfolgende Evaluierung der ausgewählten Komponenten und erste Funktionstests des EBF soll eine Objektdetektion realisiert werden. Dazu wird, wie in der Konzeptdiskussion 4.3.2 beschrieben, der einstufige Bilddetektor YOLO verwendet. Mit dem reinen YOLO-Modell kann keine Objektdetektion durchgeführt werden. Zusätzlich wird ein Dienstprogramm, auch Framework genannt, benötigt, das die Daten eines YOLO-Modells auf variable Bilddaten anwendet und die Ergebnisse in Textform generiert.

Durch Recherche zeigt sich, dass YOLO-Modelle mit zwei Methoden angewendet werden können. Bei der Ersten wird die Bildverarbeitungs-Bibliothek *OpenCV* [44] verwendet. OpenCV ist eine Bibliothek, die universell auf verschiedensten Rechnersystemen verwendet werden kann und speziell für Bildverarbeitung im Bereich des maschinellen Lernens und der künstlichen Intelligenz entwickelt worden ist. Für OpenCV sind diverse Application Programming Interface's (API's) verfügbar, sodass die Bibliothek mit vielen verschiedenen Programmiersprachen angewendet werden kann. Dadurch können vortrainierte YOLO-Modelle relativ einfach angewendet werden.

Bei der zweiten Methode wird das ML/KI-Framework *Darknet* [26] verwendet. Darknet ist ein einfach zu installierendes und anzuwendendes Framework, das speziell für das YOLO-Netz optimiert ist. Mit diesem ist sowohl die Ausführung einer Objektdetektion als auch das Training eines neuronalen Netzes möglich. In Darknet kann OpenCV integriert werden, wodurch eine Kombination aus einem sehr universellen und einem spezialisierten Framework entsteht. Auch unterstützt Darknet die Programmiersprache *Compute Unified Device Architecture (CUDA)* und damit die Nutzung der GPU als Rechenbeschleuniger. Dies ist ein sehr wichtiges Merkmal, da, wie in Abschnitt 4.1.2 beschrieben, die hohe Rechenleistung des Jetson Xavier NX größtenteils durch die verbaute GPU erreicht wird. Aus diesen Gründen wird nachfolgend diese Methode verwendet.

5.2.1 Darknet

Für die nachfolgende Evaluierung wird Darknet verwendet. Die Installation dieses Frameworks ist auf Geräten mit Linux Betriebssystem problemlos möglich und erfolgt indem die benötigten Dateien aus dem entsprechenden GitHub Verzeichnis [26] heruntergeladen und an einem passenden Ort gespeichert werden. Die Datengröße des kompletten Frameworks beträgt ca. 150 MB, womit es sich sehr gut auf einem eingebetteten System, das nur begrenzt Speicherplatz bietet, verwenden lässt. Für die Evaluierung ist es sinnvoll,

dass die Detektionsergebnisse grafisch auf einem Bildschirm ausgegeben werden können. Aus diesem Grund wird zusätzlich OpenCV installiert. Dies ist für die spätere Verwendung im EBF nicht zwingend erforderlich.

Wie die meisten Frameworks, hat auch Darknet keine grafische Benutzeroberfläche, sondern wird durch passende Befehle auf der Konsole gesteuert. Um Optionen wie die Verwendung von OpenCV und GPU-Unterstützung zu steuern, wird ein sogenanntes *Make-File* verwendet. Ein Ausschnitt mit den relevanten Einstellungen für OpenCV (Zeile 4) und die GPU-Nutzung (Zeile 1) ist in Listing 2 abgebildet.

```
1 GPU=1
2 CUDNN=1
3 CUDNN_HALF=1
4 OPENCV=1
5 AVX=0
6 OPENMP=1
7 LIBSO=1
8 ZED_CAMERA=0
9 ZED_CAMERA_v2_8=0
10
11 # set GPU=1 and CUDNN=1 to speedup on GPU
12 # set CUDNN_HALF=1 to further speedup 3 x times (Mixed-precision on Tensor
13 #Cores) GPU: Volta, Xavier, Turing and higher
14 # set AVX=1 and OPENMP=1 to speedup on CPU (if error occurs then set AVX=0)
15 # set ZED_CAMERA=1 to enable ZED SDK 3.0 and above
16 # set ZED_CAMERA_v2_8=1 to enable ZED SDK 2.X
17
18 USE_CPP=0
19 DEBUG=0
20
21 ARCH= -gencode arch=compute_30,code=sm_30 \
22       -gencode arch=compute_35,code=sm_35 \
23       -gencode arch=compute_50,code=[sm_50,compute_50] \
24       -gencode arch=compute_52,code=[sm_52,compute_52] \
25       -gencode arch=compute_61,code=[sm_61,compute_61]
```

Listing 2: Ausschnitt des verwendeten Make-Files für Darknet.

Durch setzen, bzw. rücksetzen der in Zeile eins bis neun dargestellten Variablen werden die einzelnen Optionen aktiviert, bzw. deaktiviert. Damit die Einstellungen wirksam werden, muss das Make-File ausgeführt werden. Dabei wird das komplette Darknet-Framework neu kompiliert und die eingestellten Änderungen vorgenommen.

Der Aufruf von Darknet erfolgt mit der Konsoleneingabe `./darknet detector ...`, wobei sich die Konsole in dem passenden Verzeichnis befinden muss. Damit ist das passende Framework zur Ausführung einer Objekterkennung gefunden und einsatzbereit.

5.2.2 YOLO-Anwendung

Mit Darknet ist die Verwendung eines vortrainierten YOLO-Modells unkompliziert. Es muss lediglich die Detektorfunktion aufgerufen und die entsprechenden Parameter übergeben werden. Das Listing 3 zeigt die Darknet-Aufrufe, die für die Objektdetektion benötigt werden.

```
1 //Bild-Objekterkennung
2 ./darknet detector test cfg/XXXX.data cfg/XXXX.cfg XXXX.weights...
3     data/dog.jpg -dont_show -ext_output -out result.json
4
5 //Video-Objekterkennung
6 ./darknet detector demo cfg/XXXX.data cfg/XXXX.cfg XXXX.weights...
7     data/traffic.mp4 -dont_show -ext_output -out traffic_detect.mp4
8
9 //Live-Stream-Bilderkennung
10 ./darknet detector demo cfg/XXXX.data cfg/XXXX.cfg XXXX.weights
11     'nvarguscamerasrc ! video/x-raw(memory:NVMM), width=(int)1280,
12     height=(int)720,format=(string)NV12, framerate=(fraction)15/1 !
13     nvvidconv flip-method=2 ! video/x-raw, format=(string)BGRx !
14     videoconvert ! video/x-raw, format=(string)BGR ! appsink'
15     -ext_output -save_textresult -dont_show
```

Listing 3: Aufruf zur Objekterkennung Zeile 1-2: Bild, 4-5: Video, 7-12: Livestream.

Für die Objekterkennung in Einzelbildern wird die Funktion `detector test` (Zeile 1-3) verwendet. Für die Objektdetektion auf Basis eines Videos oder Livestreams wird die Funktion `detector demo` (Zeile 5-15) verwendet. An die Funktionen werden zunächst die Konfigurationsdaten für den zu verwendenden Datensatz (`XXXX.data`) und das verwendete Netzwerk (`XXXX.cfg`) sowie das in Abschnitt 4.3 erwähnte weights-file (`XXXX.weights`) des YOLO-Modells übergeben. Anschließend folgt die Daten/Bildquelle und optionale Parameter, deren Funktionen der Tabelle 5.2 zu entnehmen sind.

Tabelle 5.2: Optionale Parameter beim Aufruf der Objektdetektion.

Parameter	Funktion
-dont_show	Kein grafische Anzeige der Ergebnisse im Bild/Video.
-ext_output	Ausgabe der Detektionsergebnisse auf der Konsole.
-out beispiel.datei	Ergebnisse werden in separater Datei (beispiel.datei) gespeichert.
-save_textresult	Ergebnisse werden in .txt-Datei gespeichert, selber implementiert, mehr dazu in Abschnitt 5.3.

Um den, bzw. die Aufrufe aus Listing 3 zu testen, wird ein vortrainiertes YOLO-Modell verwendet und eine Objektdetektion in einem Einzelbild durchgeführt. Auf das verwendete YOLO-Modell wird in Abschnitt 6.1.1 näher eingegangen.

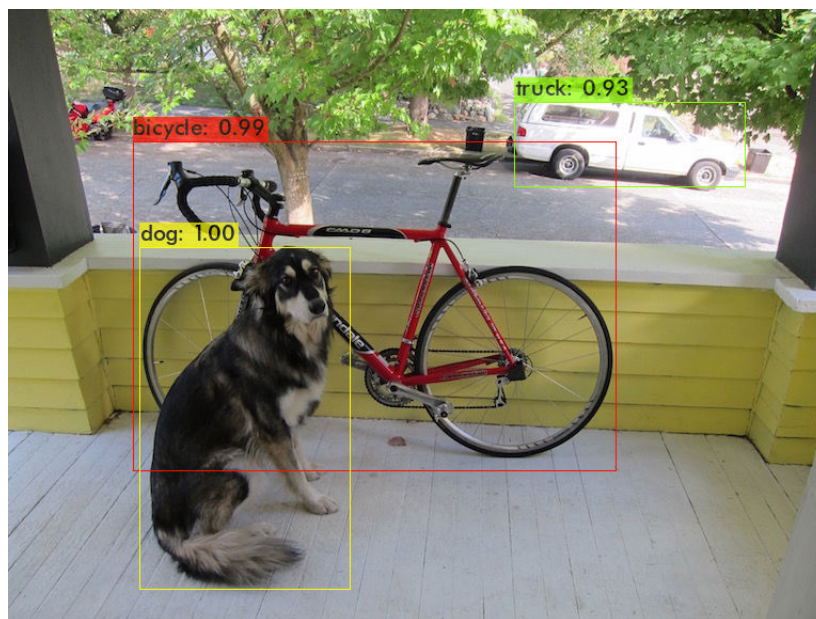


Abbildung 5.4: Ergebnis der Einzelbilddetektion zum Test von Darknet.

Wie in Abbildung 5.4 zu sehen ist, werden verschiedene Objekte in dem Bild erkannt und markiert. Damit ist die Funktion von Darknet in Verbindung mit YOLO erfolgreich getestet worden diese Kombination kann für die Evaluierung des Gesamtsystems verwendet werden.

5.3 Implementierung der UART-Schnittstelle

Wie in Abschnitt 4.5 beschrieben, soll für die Kommunikation zwischen der *Motion&Sensor Plattform* und der *Video-Detector Plattform* die UART-Schnittstelle verwendet werden. In den nachfolgenden Abschnitten wird sowohl die Implementierung der Schnittstelle auf der *Video-Detector Plattform* als auch auf der *Motion&Sensor Plattform* dargestellt. Um eine möglichst flexible Auswertung der Detektionsergebnisse zu ermöglichen, werden möglichst umfangreiche Informationen über die detektierten Objekten an die *Motion&Sensor Plattform* weitergegeben. In Darknet implementiert ist eine Ausgabe der Detektionsergebnisse auf die Konsole. Dies ist sowohl für die Objektdetektion in Einzelbildern als auch für die Detektion in Videos und Livestreams möglich. Die Ausgabe, bzw. Speicherung der Ergebnisse in Textform in eine Datei ist jedoch lediglich für die Objektdetektion in Einzelbildern implementiert. Hier wird das JSON-Format [1] verwendet. Anhand dieser Implementierung zeigt sich sehr gut, welche Informationen über die detektierten Objekte zur Verfügung stehen. Die Abbildung 5.5 zeigt exemplarisch die Detektionsergebnisse eines Bildes, das vergleichbar mit der Abbildung 5.4 ist.

```
bicycle: 99% (left_x: 128 top_y: 130 width: 440 height: 294)
dog: 99% (left_x: 133 top_y: 233 width: 177 height: 303)
car: 27% (left_x: 465 top_y: 74 width: 219 height: 98)
truck: 91% (left_x: 465 top_y: 75 width: 222 height: 97)
[
  {
    "frame_id":1,
    "filename":"data/dog.jpg",
    "objects": [
      {"class_id":16, "name":"dog", "relative_coordinates":{"center_x":
0.288533, "center_y":0.668102, "width":0.231042, "height":0.525917},
"confidence":0.991277},
      {"class_id":7, "name":"truck", "relative_coordinates":{"center_x":
0.749755, "center_y":0.214364, "width":0.289598, "height":0.168809},
"confidence":0.912496},
      {"class_id":2, "name":"car", "relative_coordinates":{"center_x":
0.747692, "center_y":0.212608, "width":0.285524, "height":0.169940},
"confidence":0.267310},
      {"class_id":1, "name":"bicycle", "relative_coordinates":
{"center_x":0.453275, "center_y":0.480780, "width":0.573530,
"height":0.510200}, "confidence":0.988831}
    ]
  }
]
```

Abbildung 5.5: Ausgabe der Detektionsergebnisse auf der Konsole (oben) und im JSON-Format (unten).

Neben den eigentlichen Objekten mit den dazugehörigen Vorhersagewahrscheinlichkeiten sind auch die relativen Koordinaten der Objekte verfügbar. Diese können im EBF verwendet werden, um zum Beispiel zu differenzieren, ob ein Objekt in der jeweiligen Situation relevant für den Nutzer ist oder nicht.

Die Speicherung der Daten ist bisher nur für die Einzelbilddetektion möglich. Der EBF wird in Zukunft jedoch einen Livestream als Detektionsgrundlage verwenden müssen. Deshalb wird nachfolgend eine Speicherung der Detektionsergebnisse und Ausgabe von diesen auf die serielle Schnittstelle implementiert. Dazu wird zunächst der Ablauf einer Kommunikation zwischen der *Motion&Sensor Plattform* und der *Video-Detector Plattform* mit einem Ablaufdiagramm (siehe Abbildung 5.6) entworfen.

Ablauf der UART-Kommunikation

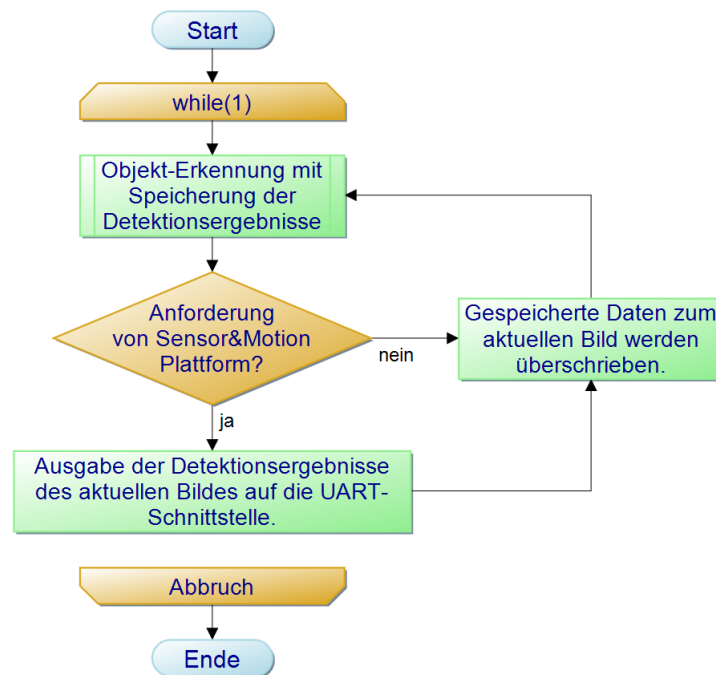


Abbildung 5.6: Ablauf der UART-Kommunikation zwischen *Motion&Sensor Plattform* und *Video-Detector Plattform*.

Die *Video-Detector Plattform* soll auf Anforderung der *Motion&Sensor Plattform* die Detektionsergebnisse des aktuell verfügbaren Bildes, auch *Frame* genannt, des Livestreams auf die serielle Schnittstelle ausgeben. Von dort kann anschließend die *Motion&Sensor Plattform* die Daten lesen und verarbeiten. Um dies zu realisieren, wird nachfolgend zunächst die Speicherung der Detektionsergebnisse auf der Recheneinheit und anschließend die UART-Schnittstelle sowohl auf der Recheneinheit als auch auf der *Motion&Sensor Plattform* implementiert.

5.3.1 Implementierung auf der Recheneinheit

Die Implementierung auf der Recheneinheit erfolgt in zwei Schritten. Zunächst wird in Darknet eine Möglichkeit geschaffen, um die Detektionsergebnisse der Livestreamdetektion in eine Datei zu speichern. Dies erfolgt nach dem Vorbild der Speicherung der Ergebnisse einer Einzelbilddetektion. Allerdings wird nicht das JSON-Format verwendet, sondern die Daten werden als Strings in eine Textdatei (*.txt*) gespeichert.

Die in Abbildung 5.5 gezeigten Daten können gut in der vorhandenen Funktion *draw_detections_cv_v3()* von Darknet abgegriffen werden. Diese Funktion dient in eigentlich zur Ausgabe der Detektionsergebnisse auf der Konsole. Ein Beispiel dieser Ausgabe ist in der Abbildung 5.5 im oberen Bereich dargestellt. Durch die Übergabe des Parameters *-ext_output* an die Darknet-Funktion *detector_demo* beim Start der Objektdetektion, wird die Funktion *draw_detections_cv_v3()* verwendet. Damit weiterhin die Ausgabe der Ergebnisse auf der Konsole auch ohne die Speicherung der Detektionsergebnisse möglich ist, wird der Übergabeparameter *-save_textresult* eingeführt. Nur wenn dieser Parameter an die Darknet-Funktion *detector_demo* übergeben wird, werden die Ergebnisse gespeichert. Die Abbildung 5.7 zeigt exemplarisch die Form in der die Detektionsergebnisse in die Textdatei gespeichert werden.

```
62
cell phone (41%) left_x: 920 top_y: 0 width: 258 height: 130
mouse (93%) left_x: 68 top_y: 586 width: 138 height: 87
pottedplant (49%) left_x: 2 top_y: 3 width: 523 height: 606
person (99%) left_x: 10 top_y: 0 width:1280 height: 720
```

Abbildung 5.7: Beispiel einer Textdatei, in der die Detektionsergebnisse eines Livestream-Frames gespeichert sind.

Die Funktion *draw_detections_cv_v3()* mit der integrierten Speicherfunktion wird nur aufgerufen, wenn Objekte im aktuellen Livestream-Frame detektiert werden. Wenn dies der Fall ist, werden die zu diesem Zeitpunkt als numerische Vektoren vorhandenen Detektionsergebnisse in lesbaren Text umgeformt. Nach der Umwandlung setzt die neue Implementierung zur Speicherung der Ergebnisse an. Zunächst wird überprüft, ob der Parameter *-save_textresult* gesetzt ist. Wenn nicht, werden die Daten nur auf die Konsole ausgegeben, wenn ja wird die Speicherung fortgeführt, indem eine neue Datei, in die die Ergebnisse gespeichert werden angelegt wird. Für die Zuordnung der Ergebnisse zum entsprechenden Livestream-Frame wird als erstes die Framenummer in die Datei geschrieben. Anschließend werden alle zu dem Frame gehörigen Detektionsergebnisse als String

in die Datei geschrieben. Anschließend wird die Datei geschlossen und die Funktion ist bereit um die Ergebnisse eines neuen Frames zu speichern.

Das nachfolgende Ablaufdiagramm 5.8 visualisiert den zuvor beschriebenen Ablauf der Speicherung. Der gezeigte Ablauf ist in die vorhandenen Funktion implementiert worden und funktioniert.

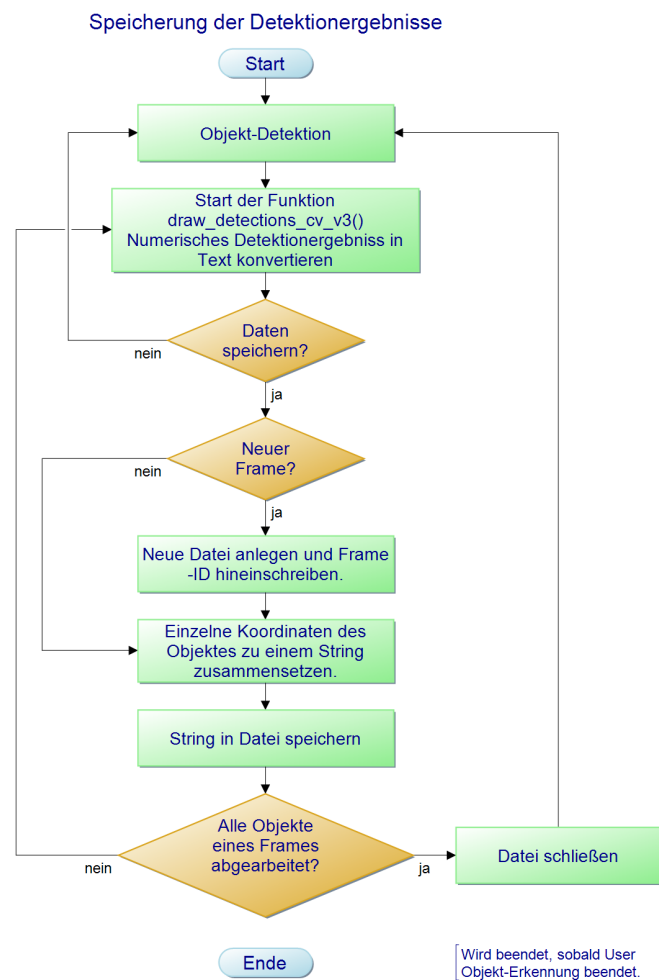


Abbildung 5.8: Ablauf der Speicherung der Detektionsergebnisse in eine Datei.

Im zweiten Schritt wird die UART-Schnittstelle auf der Recheneinheit konfiguriert und anschließend die Ausgabe der Detektionsergebnisse auf dieser implementiert. Dazu wird ein separates Skript in der Programmiersprache *Python* [46] geschrieben. In diesem Skript wird zunächst die Schnittstelle mit den Standardeinstellungen des Xavier NX konfigu-

riert. Für die Erste von drei verfügbaren UART-Schnittstellen der Recheneinheit handelt es sich dabei um die Konfigurationsparameter: Port = `/dev/ttyTHS0`, Datenrate= 115200 *Bit/s*, Datenbits = 1 *Byte*, kein Paritätsbit und ein Stop-Bit. Mit diesen Einstellungen ist die Schnittstelle konfiguriert und einsatzbereit.

Für die Ausgabe der Daten auf die UART-Schnittstelle wird vor der eigentlichen Implementierung, der Ablauf der Kommunikation zwischen der *Video-Detector Plattform* und der *Motion&Sensor-Plattform* entworfen. Das Ergebnis ist im Ablaufdiagramm 5.9 dargestellt.

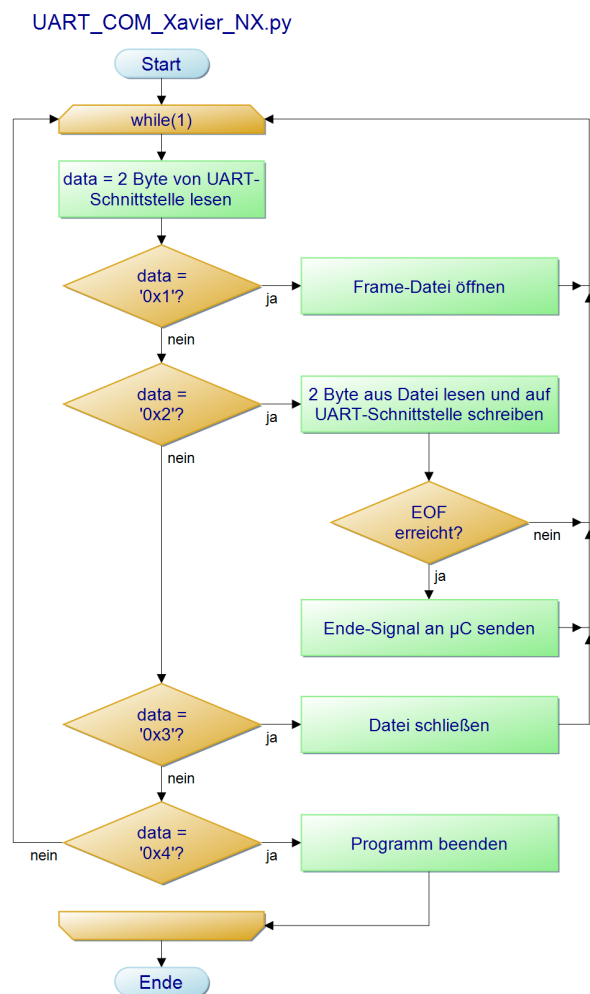


Abbildung 5.9: Ablauf der Kommunikation zwischen der *Video-Detector Plattform* und der *Motion&Sensor-Plattform* zur Implementierung der UART-Kommunikation auf der *Video-Detector Plattform*.

Das Skript *UART_COM_Xavier_NX.py* (Abbildung 5.9) arbeitet mit einer Endlosschleife und wartet auf verschiedene Befehle, in Form von bestimmten Datenbytes. Diese werden von der *Motion&Sensor-Plattform* über die UART-Schnittstelle an die *Video-Detector Plattform* gesendet. Mit dem Byte *0x1* wird die aktuell verfügbare Datei mit den Detektionsergebnissen geöffnet. Die Ausgabe der Ergebnisse auf die Schnittstelle erfolgt in 16 *Byte* großen Datenpaketen und wird mit dem Byte *0x2* gestartet. Die Begrenzung auf 16*Byte* erfolgt durch die Kapazität des UART-FIFO der *Motion&Sensor-Plattform*. Die *Motion&Sensor-Plattform* fordert so lange Daten an, bis das Ende der Datei erreicht ist. Ist das Ende erreicht, schreibt die Recheneinheit ein Ende-Befehl auf die Schnittstelle, was der *Motion&Sensor-Plattform* signalisiert, dass die Datei geschlossen werden muss. Dazu dient das Byte *0x3*. Mit dem Byte *0x4* kann die *Motion&Sensor-Plattform* die Übertragung der Detektionsergebnisse beenden. Wie beschrieben erfolgt die Implementierung in dem Skript *UART_COM_Xavier_NX.py*.

Damit ist die serielle Schnittstelle auf der Seite der Recheneinheit fertig implementiert und bereit für die Kommunikation mit dem Mikrocontroller der *Motion&Sensor-Plattform*.

5.3.2 Implementierung auf der *Motion&Sensor-Plattform*

Die Realisierung der UART-Schnittstelle auf der *Motion&Sensor Plattform* erfolgt exemplarisch, da die Auswertung der Detektionsergebnisse auf der *Motion&Sensor Plattform* nicht Teil dieser Arbeit ist. Es soll lediglich gezeigt werden, dass die Kommunikation mit der *Video-Detector Plattform* funktioniert und damit das Konzept der *Black Box* erfüllt wird. Eine spätere Implementierung kann effizienter und für die jeweilige Anwendung passender gestaltet werden. Auch hier wird zunächst ein Ablaufdiagramm (siehe Abbildung 5.10) erstellt, das den Ablauf des Programms auf der *Motion&Sensor Plattform* visualisiert. Das Programm der *Motion&Sensor Plattform* ist das Gegenstück zu dem in Abbildung 5.9 vorgestellten Ablauf auf der Recheneinheit.

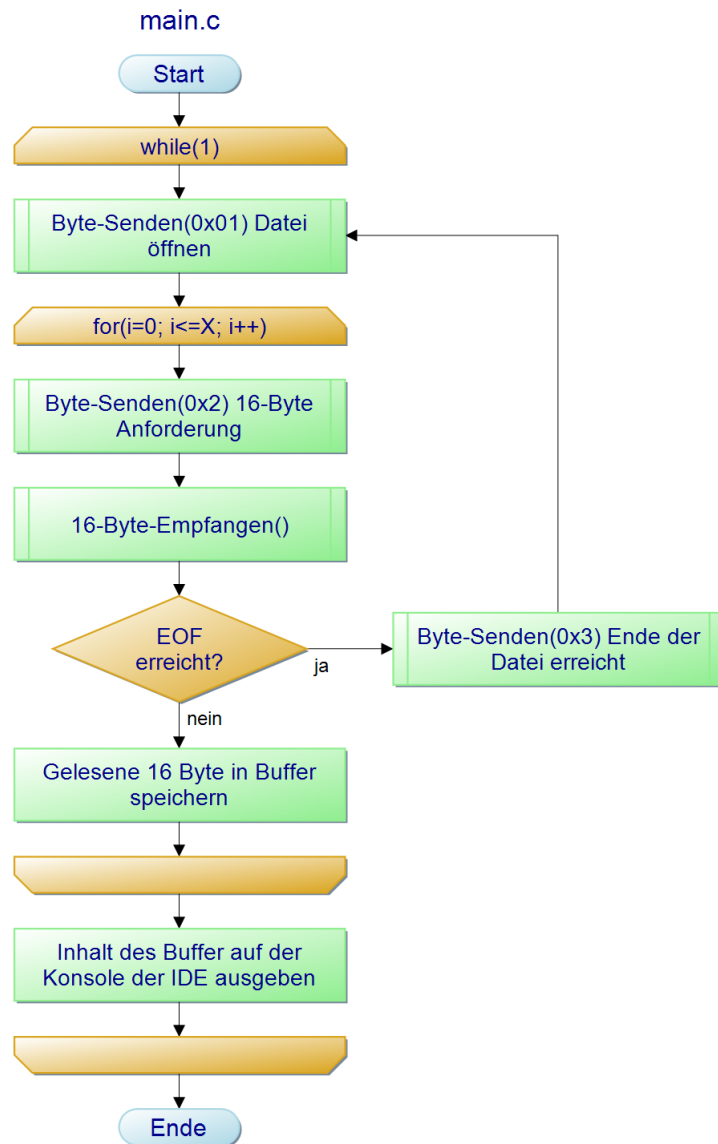


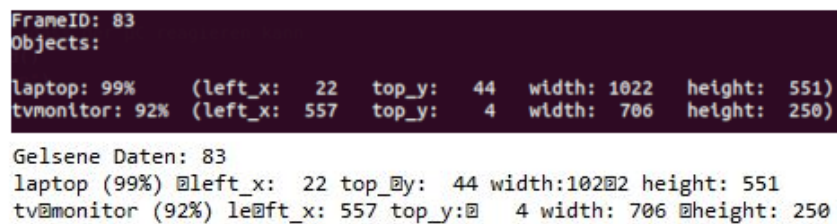
Abbildung 5.10: Ablauf der Kommunikation zwischen der *Video-Detector Plattform* und der *Motion&Sensor-Plattform* zur Implementierung der UART-Kommunikation auf der *Motion&Sensor Plattform*.

Das Programm der *Motion&Sensor Plattform* basiert auf dem Senden der in Abschnitt 5.3.1 genannten Daten-Bytes. Um einen reibungslosen Ablauf zu gewährleisten muss dies zu passenden Zeitpunkten erfolgen. So darf nach dem Senden des Bytes 0x01 zur Anfor-

derung von Daten mit einer Größe von 16 *Byte* nicht direkt von der Schnittstelle gelesen werden, sondern die *Video-Detector Plattform* muss genug Zeit zum Senden der Daten haben. Da hier kein aktiver Handshake zwischen der *Video-Detector Plattform* und *Motion&Sensor Plattform* stattfindet, muss dies durch Verzögerungszeiten im Programmcode realisiert werden.

In dieser beispielhaften Implementierung werden maximal 96 mal 16 *Byte* angefordert, sodass ca. 24 Objekte von der *Video-Detector Plattform* zur *Motion&Sensor Plattform* übertragen werden können. Die Zahl von 24 Objekten ist ein Mittelwert von detektierten Objekten innerhalb eines Livestream-Frames, der durch praktisch Test's festgelegt wurde. In einer späteren, spezifischeren Implementierung kann die Anzahl den Anforderungen entsprechend angepasst werden. Damit ist auch die UART-Kommunikation auf der *Motion&Sensor-Plattform* implementiert und einsatzbereit.

Für die Ausführung der Objektdetektion und Ausgabe der Ergebnisse auf die UART-Schnittstellen müssen, wie zuvor beschrieben, zwei verschiedene Programme, bzw. Skripte gestartet werden. Damit dies nicht manuell durchgeführt werden muss, wird ein Skript (*Start_Detection_and_Communication.py*) geschrieben, dass die Programme automatisch startet. Bei der Ausführung muss beachtet werden, dass als erstes das Programm auf der *Motion&Sensor Plattform* gestartet wird, da ansonsten Fehler durch die nicht parametrisierte Schnittstelle auftreten. Die Abbildung 5.11 zeigt beispielhaft die über die UART-Schnittstelle an die *Motion&Sensor-Plattform* übermittelten Detektionsergebnisse eines Livestream-Frames in der Konsole der verwendeten *Integrated Development Environment (IDE) Code Composer Studio* [54].



```
FrameID: 83
Objects:
laptop: 99% (left_x: 22 top_y: 44 width: 1022 height: 551)
tvmonitor: 92% (left_x: 557 top_y: 4 width: 706 height: 250)

Gelsene Daten: 83
laptop (99%) @left_x: 22 top_y: 44 width:1022 height: 551
tv@monitor (92%) le@ft_x: 557 top_y:@ 4 width: 706 @height: 250
```

Abbildung 5.11: Ausgabe der Detektionsergebnisse auf der Konsole der Recheneinheit (oben) und auf der Konsole der IDE (unten).

Wie zu erkennen werden die Ergebnisse inhaltlich unverändert an die *Motion&Sensor-Plattform* übertragen. Nähere Angaben zur Funktionsweise der Implementierung werden in Abschnitt 6.4 getätigt.

6 Evaluierung der Hard- und Softwarekomponenten

Um die Funktion und die Eignung der ausgewählten Komponenten bestehend aus Recheneinheit, Kamera und Klassifikationsalgorithmus, für die Verwendung im EBF zu evaluieren, werden in diesem Kapitel verschiedene Untersuchungen durchgeführt. Diese sind alle auf die zukünftige Verwendung der Komponenten im EBF ausgerichtet und sollen Stärken und Schwächen und besondere Merkmale aufzeigen, die für die Entwicklung des EBF wichtig sein könnten. Besondere Beachtung finden die Echtzeitfähigkeit und die Leistungsaufnahme des Systems während der Ausführung einer Objektdetektion auf Basis eines Livestreams.

6.1 Vorstellung der Evaluationsbasis

Bevor die eigentliche Evaluierung durchgeführt wird, werden zunächst die verwendeten Bestandteile vorgestellt und Kriterien festgelegt, anhand derer die Ergebnisse bewertet werden.

6.1.1 YOLO-Modell

Für die Evaluierung werden zwei verschiedene YOLO-Modelle verwendet. Dabei handelt es sich um jeweils ein fertig trainiertes YOLOv3- und YOLOv4-Modell, die mit dem COCO-Datensatz [14] trainiert und verifiziert worden sind. Die Nutzung von vor-trainierten Modellen bietet sich für die Evaluierung an, da das sehr zeitaufwändige Training entfällt. Durch das Training mit dem COCO-Datensatz sind die Modelle in der Lage 80 verschiedene Alltagsobjekte, wie Autos, Menschen, oder Bücher zu erkennen. Die Modelle werden durch eine sogenannte *weights*-Datei und *cfg*-Datei repräsentiert. In der *weights*-Datei sind die beim Training erlernten Merkmale gespeichert, die durch Zahlen (Gewichte)

dargestellt sind. In der *cfg*-Datei ist der Aufbau des YOLO-Modells mit den einzelnen Schichten gespeichert. Beide Dateien können mit Darknet direkt verarbeitet werden. Bei der Analyse der Dateien zeigt sich, dass die Dateien sehr umfangreich sind. Alleine die *weights*-Datei des YOLOv3-Modells hat eine Dateigröße von 236 MB. Dies zeigt, dass die Anzahl von 80 detektierbaren Objekten sehr hoch ist und nur möglich ist, da das Training des Modells mit mehreren hunderttausend Bildern erfolgt ist.

6.1.2 Evaluationskriterien

In den nachfolgenden Untersuchungen sollen die ausgewählten Komponenten hauptsächlich auf Echtzeitfähigkeit und allgemeine Leistungsfähigkeit untersucht werden. *Echtzeit* hat im technischen Kontext keine feste Definition, sondern ist immer anwendungsbezogen definiert. So kann ein System, das nach mehreren Sekunden ein Ergebnis liefert genau so echtzeitfähig sein, wie eines, das nach Millisekunden ein Ergebnis liefert [3]. Aus diesem Grund wird für diese Anwendung folgende Definition von Echtzeit eingeführt: Das System zur Objektdetektion ist echtzeitfähig, wenn mindestens fünf Bilder innerhalb einer Sekunde verarbeitet werden können. Es wird angenommen, dass mit fünf Bildern pro Sekunde ein zuverlässiger und sinnvoller Betrieb des EBF möglich ist. *Verarbeitet* heißt in diesem Zusammenhang: Aufnahme, Auswertung und Bildung eines Ergebnisses auf Grundlage eines Bildes, bzw. Frames eines Livestreams.

Die Leistungsfähigkeit definiert sich teilweise durch die Echtzeitfähigkeit: Wenn das System echtzeitfähig ist, ist es auch leistungsfähig. Um die Leistung des Systems einordnen zu können wird nachfolgend eine Untersuchung durchgeführt, bei der die Recheneinheit mit zwei anderen Computersystemen verglichen wird. Durch den Vergleich der technischen Daten der drei Computersysteme wird festgelegt, dass wenn das Ergebnis der Untersuchung ist, dass sich der Jetson Xavier NX in Bezug auf die Verarbeitungsraten zwischen diesen beiden Computersystemen einordnet die Leistungsfähigkeit festgestellt ist.

6.1.3 Funktionen des Jetson Xavier NX zur Leistungsoptimierung

Für die Evaluierung bietet der Jetson Xavier NX zwei Funktionen, mit denen möglicherweise die Leistungsfähigkeit und der Energiebedarf der Recheneinheit beeinflusst wird. Die erste Funktion ist der sogenannte *PowerMode (PM)*. Der Xavier NX hat fünf verschiedene PM mit denen sich einstellen lässt, wie viele CPU-Kerne verwendet werden

sollen und wo das *power target* liegt. Mit dem *power target* wird festgelegt, wie viel elektrische Leistung das System benutzen darf.

Die zweite Funktion heißt *JetsonClocks (JC)*. Bei Aktivierung dieser Funktion, werden die Arbeitsfrequenzen von CPU und GPU auf das Maximum eingestellt, wobei sich diese je nach PM unterscheiden. Außerdem wird der PWM-gesteuerte Lüfter des Kühlkörpers mit $\frac{t_D}{T} = 100\%$ angesteuert. Im Normalbetrieb regelt das Betriebssystem dynamisch die Arbeitsfrequenzen. Durch das dauerhafte Aktivieren der maximalen Frequenzen soll die maximale Leistung in jedem PM erzielt werden. In wie weit die Funktion für die Anwendung der Objektdetektion effizient genutzt werden kann, sollen die nachfolgenden Untersuchungen zeigen. Die verschiedenen Einstellungen, die durch die Wahl der Power Modes und der JetsonClocks-Funktion getroffen werden, sind in Tabelle 6.1 aufgeführt.

Tabelle 6.1: Übersicht der PowerModi des Jetson Xavier NX.

PM	CPU-Kerne	Power Target in W	JC f_{CPU}	JC f_{GPU}
0	2	15	1,9 GHz	1,1 GHz
1	4	15	1,5 GHz	1,1 GHz
2	6	15	1,4 GHz	1,1 GHz
3	2	10	1,5 GHz	803 MHz
4	4	10	1,2 GHz	803 MHz

Festzustellen ist, dass im *PM 0* die höchsten Arbeitsfrequenzen an CPU und GPU angelegt werden und, dass im *PM 2* die maximale Anzahl von sechs CPU-Kernen genutzt wird. Wie in Abschnitt 5.1.2 festgestellt, wird für das Benchmark-Programm von NVIDIA der *PM 0* verwendet. Dies deutet darauf hin, dass der *PM 0* eine hohe Leistungsfähigkeit des Systems ermöglichen soll. Ob diese Einstellungen zur höchsten Leistungsfähigkeit, bzw. den besten Ergebnissen bei der Livestreamobjektdetektion führt, wird sich durch die nachfolgenden Untersuchungen zeigen.

6.2 Vergleich verschiedener Rechnersystem zur Evaluierung der Echtzeitfähigkeit bei einer Objektdetektion

Um die Echzeit- und Leistungsfähigkeit des ausgewählten eingebetteten Computers einzuordnen, wird eine Untersuchung durchgeführt, bei der auf drei verschiedenen Computern die gleiche Objektdetektion durchgeführt wird.

Der erste Computer ist der in dieser Arbeit ausgewählte Jetson Xavier NX mit eingebetteter GPU. Der Zweite ist ein handelsüblicher, relativ leistungsstarker Laptop von *Lenovo* Typ *E585* [30], der keine dedizierte GPU hat und das letzte Gerät ist ein GPU-Server auf Linux-Basis, der an der HAW Hamburg steht, und speziell für die Berechnung neuronaler Netze vier sehr leistungsstarke Grafikkarten vom Typ *NVIDIA RTX 2080 Ti* [43] bereitstellt. Durch diese drei Geräte besteht ein guter Vergleich zwischen einem Computer ohne dedizierte GPU, einer eingebetteten GPU und mehreren extrem leistungsstarken GPU's. Auf allen drei Computern wird eine Einzelbildobjektdetektion, Videoobjektdetektion und Livestreamobjektdetektion durchgeführt. Die Livestreamobjektdetektion kann nur auf dem Jetson Xavier NX ausgeführt werden, da die ausgewählte eingebettete Kamera nur an diesen angeschlossen werden kann. Folgende Parameter werden für die einzelnen Tests verwendet:

- Detektions-Grundlage:
 - Einzelbild: dog.jpg (siehe Abbildung 5.4).
 - Video: traffic.mp4.
 - Livestream: eingebettete Kamera.
- Eingangsschicht-Größe / Bildgröße: 416 x 416 *Pixel*.
- Weights-File: yolov3.weights und yolov4.weights.
- Cfg-File: yolov3.cfg und yolov4.cfg.
- Datensatz: coco.data.

Außerdem werden bei allen Tests die verschiedenen Power Modes und die JetsonClocks-Funktion verwendet.

6.2.1 Objektdetektion in einem Einzelbild

Als Erstes wird die Objektdetektion in einem Einzelbild auf allen zuvor genannten Plattformen durchgeführt. Die bei den Tests entstandenen Ergebnisse sind grafisch aufbereitet in den Abbildung 6.1, 6.2 und 6.3 zu sehen. Dabei werden die verschiedenen PowerModes gegen die Detektioszeit in Sekunden aufgetragen. Es gilt: Je niedriger die Detektioszeit, desto leistungsfähiger ist der Computer.

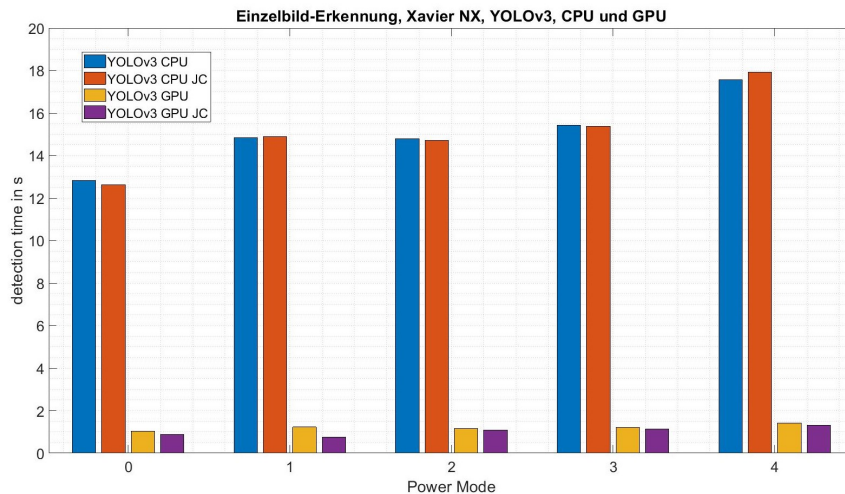


Abbildung 6.1: Ergebnisse der Einzelbilddetektion auf dem Xavier NX mit YOLOv3.

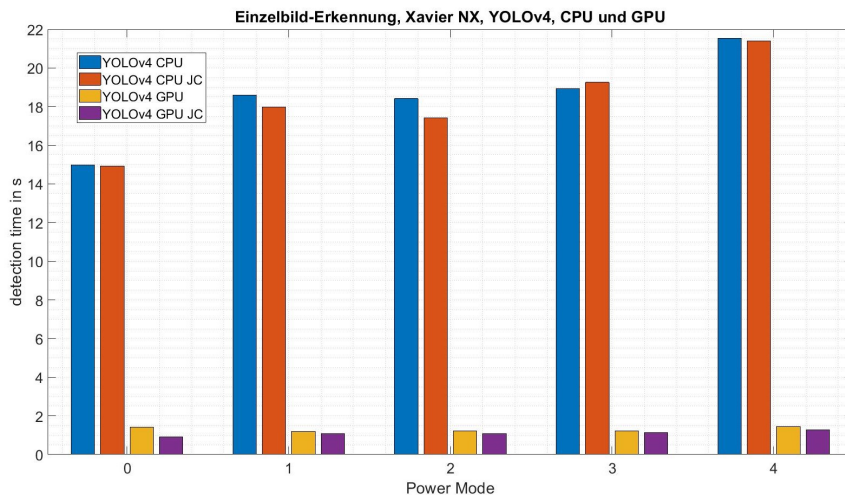


Abbildung 6.2: Ergebnisse der Einzelbilddetektion auf dem Xavier NX mit YOLOv4.

Gut zu erkennen ist, dass bei beiden YOLO-Modellen die durchschnittliche Detektion-Zeit im CPU-Betrieb deutlich größer ist ($\overline{t_{CPU}} = 16,07 s$), als im GPU-Betrieb ($\overline{t_{GPU}} = 1,21 s$). Dies gilt für alle PowerModes und unabhängig davon, ob die JetsonClocks-Funktion verwendet wird oder nicht. Festzustellen ist auch, dass die JetsonClocks-Funktion die Ergebnisse nur in geringem Maße beeinflusst. So wird in den meisten Fällen eine leichte Verbesserung der Ergebnisse erzielt. Bei zwei PowerModi (PM3 und PM0) haben sich die Ergebnisse mit aktivierter JC-Funktion allerdings leicht verschlechtert. Die Abbildung 6.3 stellt die Ergebnisse des Tests auf dem Laptop und dem Server dar.

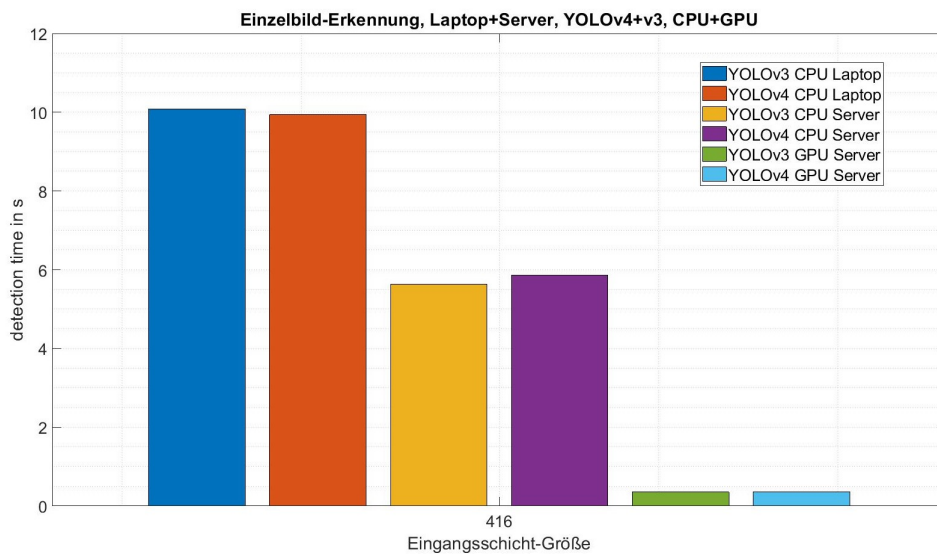


Abbildung 6.3: Ergebnisse der Einzelbilddetektion auf Server/Laptop mit YOLOv3/v4.

Es ist zu erkennen, dass die CPU-Ergebnisse des Laptops mit den CPU-Ergebnissen des Xavier NX vergleichbar sind (vergleiche Abb. 6.1 und 6.2). Die CPU-Ergebnisse des Servers sind hingegen deutlich besser als die CPU-Ergebnisse von Xavier NX und Laptop. Hier wird nur gut die Hälfte der Zeit benötigt. Auch zeigt sich beim Server, dass die GPU-Nutzung deutliche Vorteile bietet. Sowohl im Vergleich zu den CPU-Ergebnissen des Servers, als auch zu den GPU-Ergebnissen des Xavier NX führt die Nutzung der Server-GPU zu deutlich schnelleren Ergebnissen.

6.2.2 Objektdetektion in einem Video

Als Annäherung an die Livestreamobjektdetektion wird als Zweites eine Detektion basierend auf einem aufgenommenen Video durchgeführt. Die Abbildungen 6.4, 6.5 und 6.6 zeigen die Ergebnisse des Vergleichs der drei Computer. In den Abbildungen werden nun die PowerModi gegen *Frames per Second (FPS)* aufgetragen, wobei gilt: Je mehr FPS erreicht werden, desto leistungsfähiger ist der Computer.

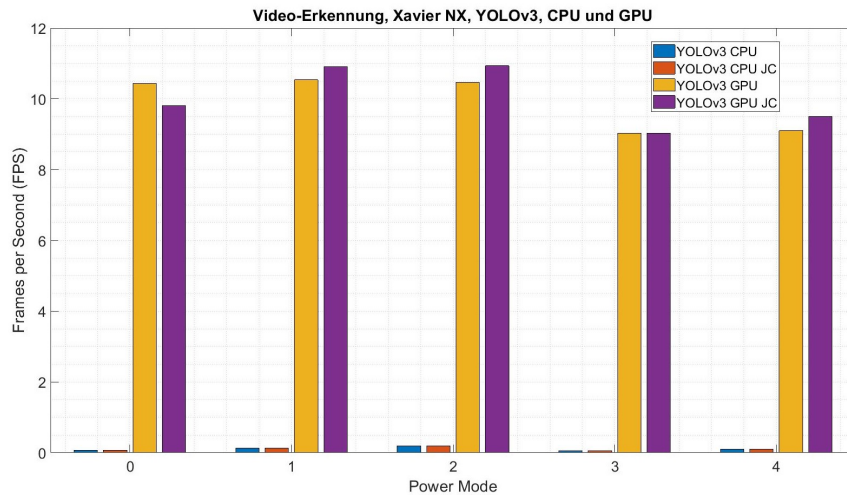


Abbildung 6.4: Ergebnisse der Videodetektion auf dem Xavier NX mit YOLOv3.

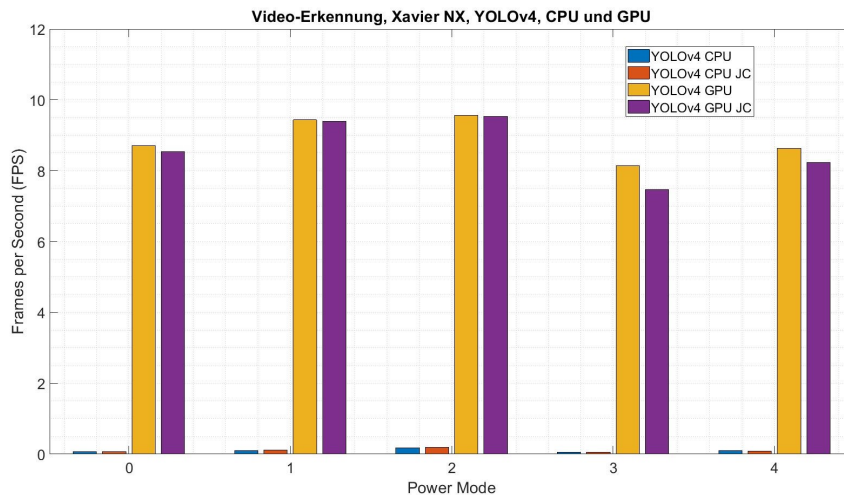


Abbildung 6.5: Ergebnisse der Videodetektion auf dem Xavier NX mit YOLOv4.

Die Ergebnisse ähneln stark den Ergebnissen aus Abschnitt 6.2.1, was zu erwarten ist, denn die Videoobjektdetektion ist eine Bildobjektdetektion, die möglichst schnell möglichst viele Einzelbilder und damit ein Video verarbeitet. Durch die Nutzung der GPU liegt die Bildrate des Xavier NX bei durchschnittlich $9,37 \text{ FPS}$. Bei der CPU werden nur zwischen $0,1 \text{ FPS}$ und $0,2 \text{ FPS}$ erreicht. Durch die Nutzung der JetsonClocks-Funktion werden, beim YOLOv3-Modell, die Ergebnisse größtenteils leicht verbessert. Beim YOLOv4-Modell wirkt sich die Funktion eher negativ aus.

Bei der Videodetektion auf dem Server und dem Laptop (Abbildung 6.6) zeigen sich große Unterschiede. Der Laptop hat bei diesem Test keine Ergebnisse liefern können, da er an seine Leistungsgrenze gestoßen ist. Beim Server setzen sich die großen Unterschiede zwischen CPU- und GPU-Ergebnisse fort. Bei der reinen CPU-Nutzung, dauert die Berechnung eines Frames bei beiden YOLO-Modellen gut 6 s (siehe Abbildung 6.6 oben). Deutlich schneller werden die Ergebnisse erneut mit der GPU berechnet. Die durchschnittliche Bildrate des Servers bei GPU-Nutzung liegt bei $79,7 \text{ FPS}$, was einen deutlichen Unterschied zu den Ergebnissen des Xavier NX darstellt. Die GPU-Ergebnisse sind im unteren Plot der Abbildung 6.6 ersichtlich. Durch die sehr leistungsstarken GPU's des Servers, bestätigt das Ergebnis die Erwartung.

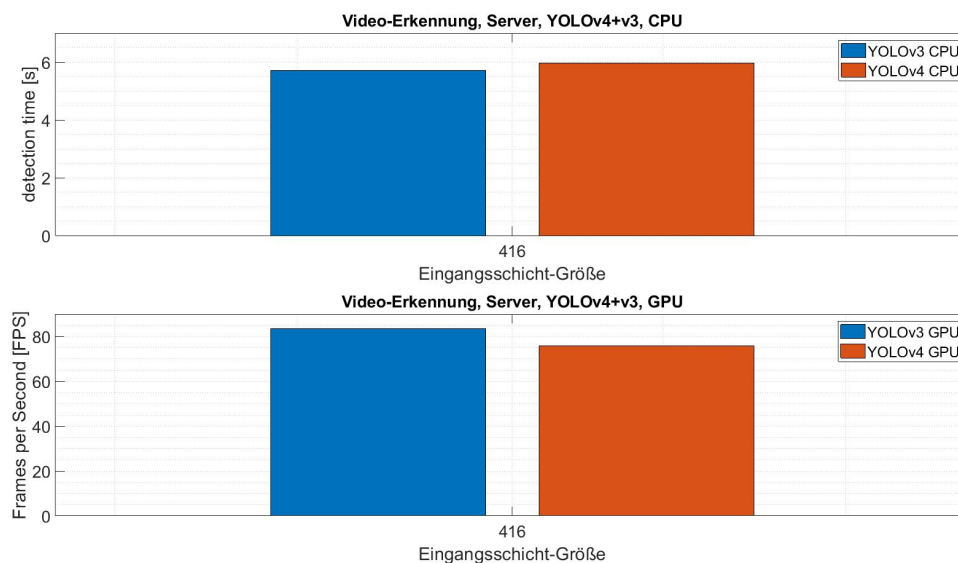


Abbildung 6.6: Ergebnisse der Videodetektion auf dem Server, CPU oben, GPU unten.

6.2.3 Objektdetektion in einem Livestream

Der nachfolgende Test und die Ergebnisse spiegeln die spätere Anwendung des EBF wieder. Die Kamera liefert einen Livestream an die Recheneinheit, welche die Objektdetektion durchführt. Da der Laptop bereits beim vorigen Test mit einem Video keine Ergebnisse liefern konnte und an den Server keine Kamera angeschlossen werden kann, wird die Livestream-Untersuchung nur auf dem Xavier NX durchgeführt. Auch werden die Tests nur mit der GPU durchgeführt, die sich in den vorigen Untersuchungen gezeigt hat, dass durch die CPU keine vernünftigen Ergebnisse zu erwarten sind. Die Kamera wird auf eine möglichst hohe Bildrate eingestellt. Dadurch wird sichtbar, wo die Leistungsgrenze der Recheneinheit liegt. Bei dieser Untersuchung wird die Abkürzung *DS* eingeführt. Diese steht für *dont show* und ist ein Parameter des Darknet-Aufrufs (siehe Tabelle 5.2), der die grafische Darstellung der Detektionsergebnisse auf einem Bildschirm deaktiviert. DS wird in dieser Untersuchung verwendet, da bei der späteren Verwendung des EBF ebenfalls keine grafische Ausgabe der Detektionsergebnisse benötigt wird. Mit diesen Einstellungen wird also der spätere Einsatz der Recheneinheit im EBF möglichst gut nachgebildet.

Die Ergebnisse der Tests sind in den Abbildungen 6.7 und 6.8 dargestellt.

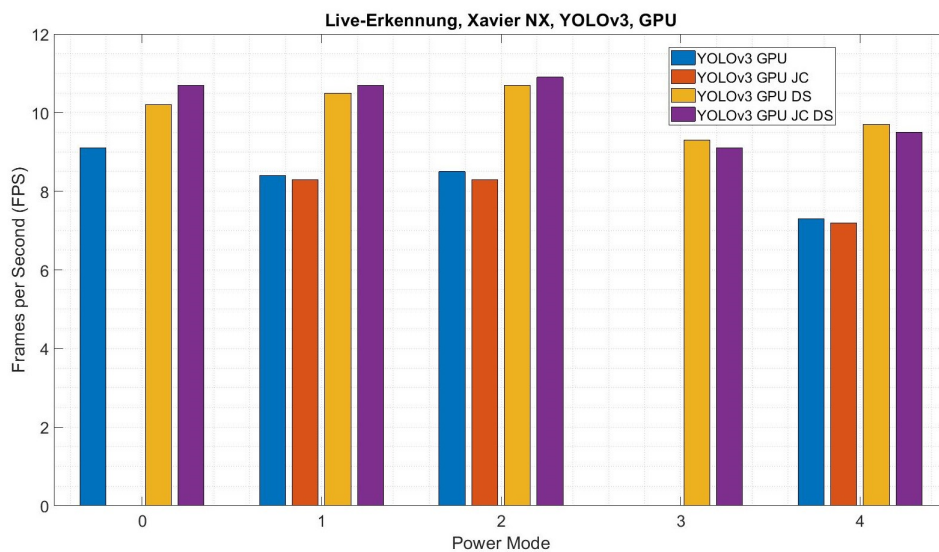


Abbildung 6.7: Ergebnisse der Livestreamdetektion auf dem Xavier NX mit YOLOv3.

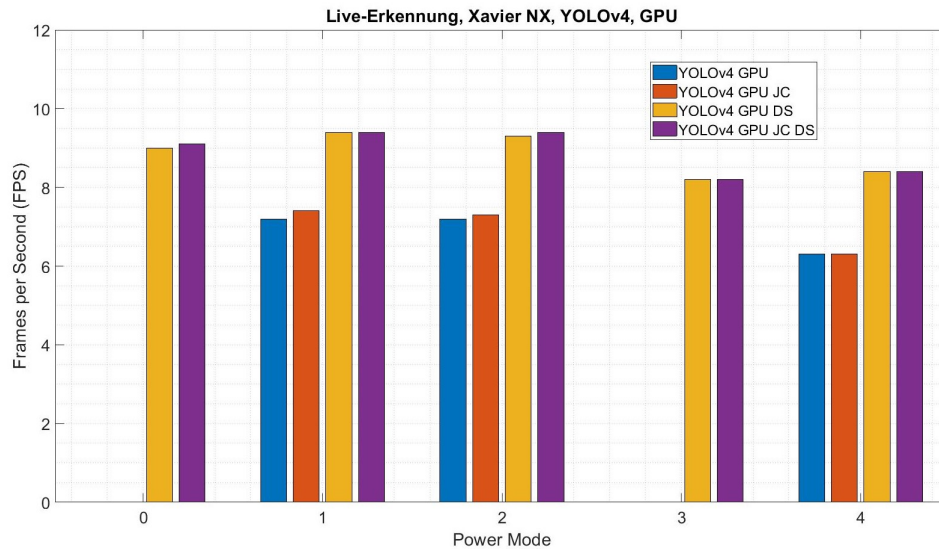


Abbildung 6.8: Ergebnisse der Livestreamdetektion auf dem Xavier NX mit YOLOv4.

Auffällig ist, dass in beiden Abbildung an mehreren Stellen Datenlücken bei Tests ohne DS entstanden sind. Diese werden durch sehr stark schwankende Ergebnisse einzelner Messreihen hervorgerufen, wodurch die Ergebnisse nicht repräsentativ sind. Um die Grafik trotzdem aussagekräftig darzustellen, werden die Ergebnisse dieser Messreihen auf Null gesetzt. Bereits an diesem Punkt zeigt sich, dass die Power Modi 0 und 3 möglicherweise nicht für eine Liveobjektdetektion geeignet sind.

Ein anderes Bild entsteht jedoch durch das Entfernen der grafischen Ausgabe der Detektionsausgabe auf dem Bildschirm mit der Funktion DS. Hier sind sowohl mit dem YOLOv3 als auch mit dem YOLOv4-Modell deutlich höhere Bildraten möglich als zuvor. Ohne grafische Ausgabe werden durchschnittlich 9,5 *FPS* erreicht, mit grafischer Ausgabe nur durchschnittlich 7,6 *FPS*. Außerdem ist festzustellen, dass durch die fehlende grafische Darstellung die starken Schwankungen bei Power Mode 0 und 3 nicht mehr auftreten und die beiden zuvor ausgeschlossenen Power Modi wieder mit in die Betrachtung einfließen können. Daraus wird geschlossen, dass die grafische Ausgabe verhältnismäßig viel Rechenleistung benötigt. Die Beeinflussung der Ergebnisse durch die JetsonClocks-Funktion ist, analog zur Videoobjektdetektion, gering.

6.2.4 Ergebnisse des Vergleichs

Ein wichtiger Aspekt bei den zuvor getätigten Beschreibung der Ergebnisse, der bewusst bisher nicht beleuchtet wurde, ist, dass bei allen Tests die Ergebnisse mit dem YOLOv3-Modell besser sind, als mit dem YOLOv4-Modell. *Besser* heißt in diesem Zusammenhang, dass eine höhere Bildrate, bzw. niedrigere Verarbeitungszeit erzielt werden konnte. Wie in Abschnitt 4.3 beschrieben, sollte eigentlich mit YOLOv4 eine höhere Bildrate möglich sein, als mit YOLOv3. Da allerdings YOLOv4 größtenteils auf YOLOv3 basiert und gut 60 Schichten zu dem Modell hinzugekommen sind ist es denkbar, dass sich dies auf der eingebetteten Recheneinheit negativ auswirkt. Zwar steht für eine eingebettete Lösung eine vergleichsweise hohe Rechenleistung zur Verfügung, allerdings ist diese im Vergleich mit einer modernen Grafikkarte, mit der die Ergebnisse aus Abbildung 4.4 erzielt worden sind, immer noch eher gering. Für Geräte mit sehr hohe Rechenleistung ist YOLOv4 demnach eine Verbesserung, für Geräte mit einer kleineren Rechenleistung eher eine Verschlechterung im Bezug auf die Verarbeitungsgeschwindigkeit! Abgesehen davon liegen die Ergebnisse des Xavier NX, in Relation zu den anderen beiden Computern, im erwarteten Bereich. Es ist eine deutlich höhere Datenrate als mit dem Laptop möglich. Genau so deutlich in negativer Richtung, ist jedoch auch der Unterschied zum GPU-Server.

Der Unterschied zum Laptop entsteht durch die Verwendung der dedizierten, eingebetteten GPU auf dem Xavier NX. Da der Laptop keine dedizierte GPU hat, kommt das Gerät sehr schnell an seine Leistungsgrenze und ist für eine Einzelbildobjektdetektion nur sehr bedingt und für eine Video/Livestreamobjektdetektion nicht einsetzbar.

Mit dem GPU-Server konnte eine gut 8-fach höhere Bildrate bei der Videoobjektdetektion erreicht werden, als mit dem Xavier NX (Vergleich der Abbildungen 6.4, 6.5 und 6.6). Wie in den vorigen Tests zu sehen, wird dies nicht durch die im Server verbaute CPU erreicht, sondern durch die GPU's der Grafikkarten des Servers. Für die Tests wird immer nur eine der vier verbauten Grafikkarten verwendet. Eine GPU einer Grafikkarte des Servers weist 4352 CUDA-Kerne [43] auf, was gut dem 11-fachen des Xavier NX entspricht, der 384 CUDA-Kerne hat [39]. Daher ist zu erwarten, dass mit der Grafikkarte eine gut elf-fach höhere Bildrate möglich ist. Der Xavier NX hat allerdings zusätzlich 48 Tensor-Kerne integriert, wodurch die tatsächliche, achtfach höhere, Bildrate plausibel erklärt ist.

Bei den PowerModi zeigt sich, dass sich für die Einzelbildobjektdetektion der PowerMode 1 am Besten eignet (siehe Abb. 6.1 und 6.2). Bei der Video- und auch Livestreamobjektdetektion werden mit dem PowerMode 2 die schnellsten Ergebnisse, bzw. höchsten Bildraten erzielt (siehe Abb. 6.4, 6.5 und 6.7,6.8). Diese beiden PowerModi nutzen zwar

eine unterschiedliche Anzahl von CPU-Kernen (PM0 = 2 Kerne, PM2 = 6 Kerne), aber beide lassen eine maximale Leistungsaufnahme von $P_{TD} = 15\text{ W}$ zu. Daraus folgt, dass ein direkter Zusammenhang zwischen Leistungsaufnahme und Rechenleistung besteht: Je mehr elektrische Leistung zur Verfügung steht, desto höher ist die Rechenleistung. Dabei muss beachtet werden, dass auch die Abwärmeleistung steigt, wodurch die maximal verwendbare elektrische Leistung in der Praxis limitiert ist. Auch muss das Carrier-Board für die Leistungsaufnahme konzipiert sein und ein entsprechend starkes Netzteil, bzw. starke Spannungsversorgung zur Verfügung stehen.

In den Tests hat sich ebenso gezeigt, dass die JetsonClocks-Funktion nur einen geringfügigen Einfluss auf die Ergebnisse und damit auf die Rechenleistung hat. Die dynamische Regelung der Arbeitsfrequenzen von CPU, GPU und Speicher scheint effizient gelöst zu sein, sodass auch ohne die Nutzung der Funktion gute Ergebnisse erzielt werden. Für die Nutzung im EBF empfiehlt sich die JetsonClocks-Funktion nicht zu verwenden.

Durch die maximal erreichte Bildrate von $10,9\text{ FPS}$ bei der Livestreamobjektdetektion im PowerMode 2 ohne grafische Ausgabe ist bewiesen worden, dass die ausgewählten Komponenten für eine Objektdetektion in Echtzeit geeignet sind.

6.3 Spezifische Evaluierung der ausgewählten Hardwarekomponenten

Im vorigen Vergleich der verschiedenen Computer lag der Fokus auf der Echtzeit- und allgemeine Leistungsfähigkeit der eingebetteten Kamera und Recheneinheit. In den nachfolgenden Untersuchungen liegt der Fokus auf der Qualität der Ergebnisse der Objektdetektion in Verbindung mit der Leistungsaufnahme des Systems. Es wird untersucht in wie weit die Ergebnisse und die Leistungsaufnahme durch verschiedene Parameter beeinflusst werden. Als Werkzeug für die Untersuchungen wird das Programm *jtop* [47] verwendet. Dieses Programm lässt ein Monitoring verschiedenster Systemkomponenten, wie die Auslastung von CPU, GPU und Speicher zu und auch die aktuelle Leistungsaufnahme und Temperaturentwicklung des Xavier NX wird dargestellt. Außerdem lassen sich die verschiedenen PowerModi und die JetsonClock-Funktion einstellen, bzw. aktivieren. Die Abbildung 6.9 zeigt die Oberfläche von *jtop*.

```

jtop Xavier NX (Developer Kit Version) - JC: Inactive - 15W 6CORE
NVIDIA Jetson Xavier NX (Developer Kit Version) - Jetpack UNKNOWN [L4T 32.4.4]
CPU1 [||||| Schedutil - 22%] 1.2GHz CPU4 [||||| Schedutil - 18%] 1.2GHz
CPU2 [||||| Schedutil - 19%] 1.2GHz CPU5 [||||| Schedutil - 12%] 1.2GHz
CPU3 [||||| Schedutil - 12%] 1.2GHz CPU6 [||||| Schedutil - 18%] 1.3GHz

HTS FG [|||||] 0% BG [|||||] 0%
Mem [|||||] 2.2G/7.9GB (lfb 1113x4MB)
Swp [|||||] 0.0GB/3.9GB (cached 0MB)
EMC [|||||] 3%] 1.6GHz

GPU [|||||] 0%] 114MHz
Dsk [|||||] 32.1GB/58.1GB

[Info] [Sensor] [Temp] [Power/mW] [Cur] [Avr]
UpT: 0 days 1:27:29 AO 25.50C CPU GPU CV 657 835
FAN [|||||] 65% Ta= 0% AUX 24.50C SOC 1107 1181
Jetson Clocks: inactive CPU 27.00C ALL 3451 3754
NV Power[2]: 15W 6CORE GPU 25.00C
[HW engines] thermal 25.40C
APE: 150MHz
NVENC: [OFF] NVDEC: [OFF]
NVJPG: [OFF]

1ALL 2GPU 3CPU 4MEM 5CTRL 6INFO Quit Raffaello Bonghi

```

Abbildung 6.9: Oberfläche des Programmes *jtop*.

6.3.1 Auswirkung verschiedener Kameraauflösungen auf die Detektionsergebnisse

In der Anforderungsanalyse (Abschnitt 3.1.3) ist festgelegt worden, dass die Kamera eine minimale Auflösung von 1920×1080 *Pixel* haben muss. Nach dieser frei getroffenen Festlegung ist, wie in Abschnitt 4.2.2 beschrieben, eine Kamera ausgewählt worden. Unbekannt ist jedoch, in wie weit die Auflösung der Kamera, bzw. der Bilder die Ergebnisse der Objektdetektion und die Leistungsaufnahme der Recheneinheit beeinflusst. Dies wird nachfolgend untersucht.

Mit dem in Listing 1 dargestellten Kameraaufruf kann die Auflösung der Kamera variabel eingestellt werden. Dabei werden die Werte der Variablen *width* und *height*, mit denen die Kameraauflösung eingestellt wird, in folgenden Schritten variiert: 320×240 ; 640×480 ; 1280×720 ; 1920×1090 ; 2048×1536 und 3264×2464 *Pixel*. Für die Objektdetektion wird das schon zuvor genutzte YOLOv3-Modell verwendet. Diese Untersuchung wird qualitativ durchgeführt, in dem die Livestreamobjektdetektion für ein konstantes Bild-Motiv erfolgt und die Detektionsergebnisse manuell beobachtet werden. Die Abbildung 6.10 zeigt die Objektdetektion mit einer Auflösung von 320×480 *Pixel*. Im Realbild (Abb. 6.10 unten links) sind eine Vielzahl von detektierten Objekten (Fenster unten links). Es werden sowohl räumlich näher, als auch ferner gelegene Objekte erkannt und markiert.

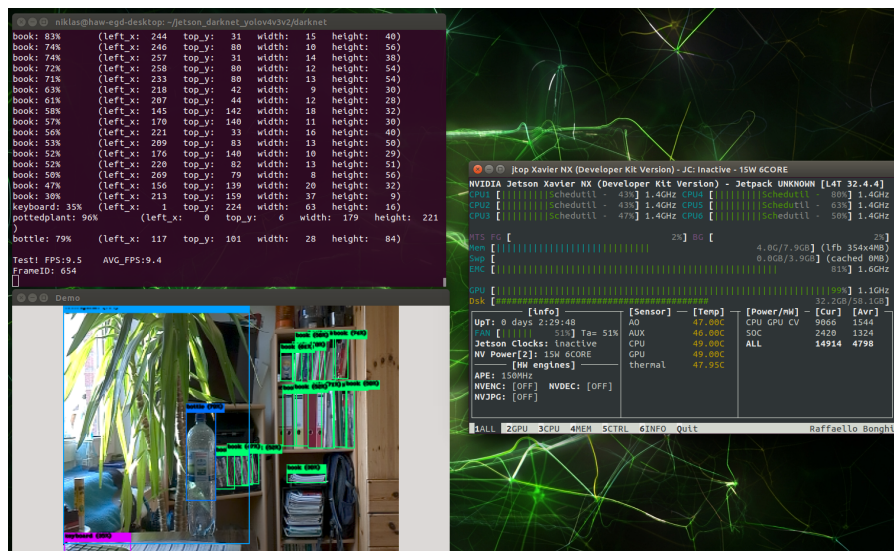


Abbildung 6.10: Ergebnisse der Detektion mit einer Auflösung von 320×480 *Pixel*.

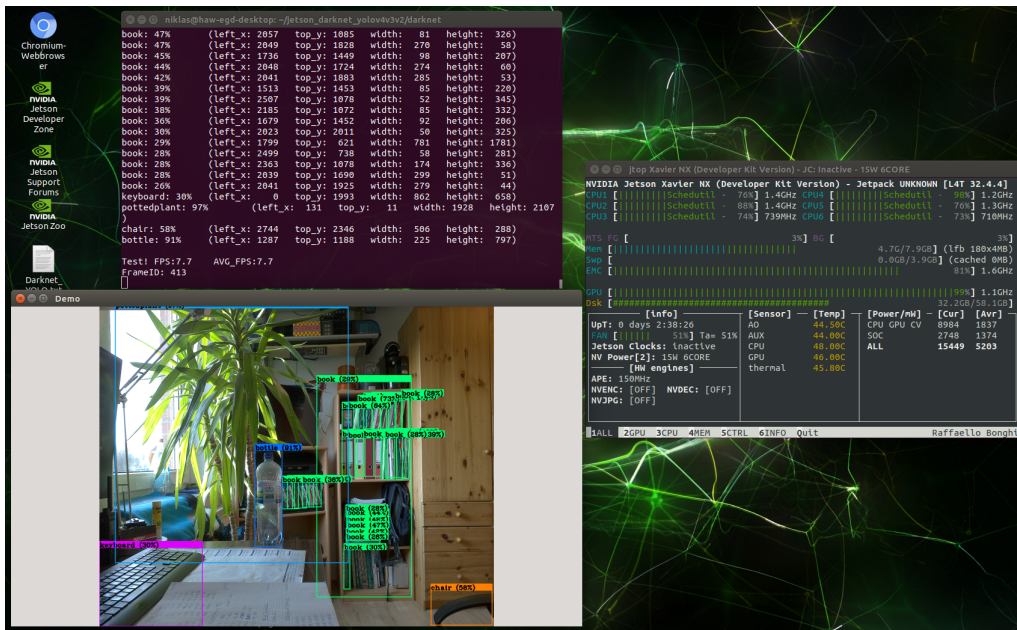


Abbildung 6.11: Ergebnisse der Detektion mit einer Auflösung von $3264 \times 2646 \text{ Pixel}$.

Die Abbildung 6.11 zeigt die Objektdetektion mit einer Auflösung von $3264 \times 2646 \text{ Pixel}$. In dem Realbild (Abb. 6.11 unten links) ist die deutlich bessere Bildqualität erkennbar. Die detektierten Objekte unterschieden sich nur minimal zu denen aus Abbildung 6.10. Die geringen Abweichungen sind auf "flackern" zurückzuführen. Die detektierten Ergebnisse sind nie vollständig stabil, sodass zu den unterschiedlichen Aufnahmezeiten der Bilder leicht unterschiedliche Ergebnisse angezeigt werden. Ebenfalls ist kein Unterschied bei der Leistungsaufnahme des Xavier NX zu erkennen. In beiden Fällen liegt diese bei 15 W , was der maximal abrufbaren elektrischen Leistung entspricht. Das heißt zwischen der Auflösung und der Detektionsqualität besteht nur ein geringer und zwischen der Auflösung und der Leistungsaufnahme besteht kein Zusammenhang.

Ein deutlicher Unterschied ist jedoch bei den Bildausschnitten zu erkennen, die durch die Auflösung entstehen. Bei der Auflösung $3264 \times 2646 \text{ Pixel}$ ist der erfasste Bildausschnitt deutlich größer, als bei der Auflösung $320 \times 480 \text{ Pixel}$. Diese Eigenschaft kann genutzt werden, um einen passenden Bildausschnitt, in dem Objekte erkannt werden, einzustellen, der für den späteren Nutzer des EBF sinnvoll ist.

6.3.2 Auswirkung der PowerModi auf die Detektionsergebnisse

Wie in der Untersuchung 6.2.4 festgestellt, lässt sich durch die geschickte Verwendung der PowerModi die Leistungsaufnahme des Xavier NX senken. Nicht beleuchtet wurde bisher, ob die PowerModi die Ergebnisse der Objektdetektion beeinflussen. Dies wird in diesem Test untersucht. Dazu wird für jeden PM die Objektdetektion mit dem YOLOv3-Modell und der eingebettete Kamera mit einer Auflösung von $1280 \times 720 \text{ Pixel}$ durchgeführt und diese erneut qualitativ bewertet.

Die Ergebnisse für den PowerMode 0 sind in der Abbildung 6.12 exemplarisches abgebildet. Die Bewertung der Ergebnisse ist nur schwer möglich, da diese, wie zuvor erwähnt, schwanken. Trotzdem zeigt der Vergleich der Ergebnisse der Objektdetektionen mit den unterschiedlichen PM, dass es keinen Unterschied in der Qualität der Ergebnisse gibt. Die Detektionsergebnisse werden nicht durch die PowerModi beeinflusst. Dies ist ein sehr wichtiges Ergebnis, da dadurch der Verwendung der verschiedenen PowerModi zur Senkung der Leistungsaufnahme problemlos möglich ist.

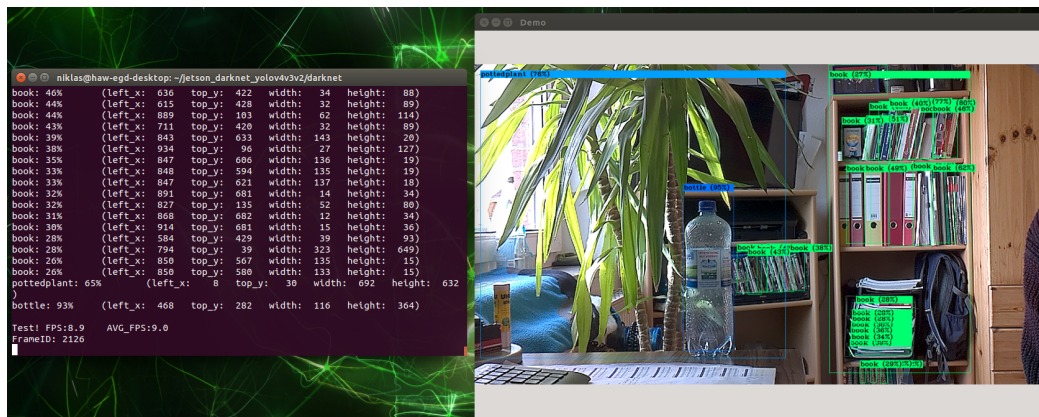


Abbildung 6.12: Ergebnisse der Objektdetektion mit PowerMode 0.

6.3.3 Auswirkung verschiedener Bildraten auf die Leistungsaufnahme

Wie in Abschnitt 6.2.4 festgestellt, liegt die maximale Bildrate mit der die Recheneinheit einzelne Bilder eines Videos verarbeiten kann bei $10,9 \text{ FPS}$. Dies wurde ermittelt, indem die Kamera auf eine hohe Bildrate von 21 FPS eingestellt wurde. Das heißt die Kamera hat 21 Bilder pro Sekunde zur Recheneinheit geliefert, von denen nur $10,9$ durch diese verarbeitet werden können.

In diesem letzten Test soll untersucht werden, in welchem Maß die Bildrate die Leistungsaufnahme der Recheneinheit beeinflusst. Dafür werden mehrere verschiedene Bildraten an der Kamera (2 ; 4 ; 6 ; 8 und 10 FPS) eingestellt, die unter der ermittelten maximalen Bildrate liegen und jeweils eine Livestreamobjektdetektion durchgeführt. Während der Objektdetektion wird die Leistungsaufnahme beobachtet und der jeweilige Mittelwert notiert. Als PowerMode wird der PM 2 ohne JC verwendet.

Wie in Grafik 6.13 erkennbar ist, sinkt die Leistungsaufnahme des Xavier NX mit sinkender Bildrate der Kamera. Der Wert bei 0 FPS repräsentiert die Grundlast des Xavier NX. Diese stellt ein Offset für die weiteren Werte dar. Aus diesem Ergebnis ist zu schlussfolgern, dass es vorteilhaft ist, mit einer geringen Bildrate zu arbeiten, wenn nicht zwingend eine höhere Bildrate benötigt wird. Dies spart Energie und im Hinblick auf die Verwendung der Komponenten im EBF kann durch diese Maßnahme die Laufzeit des akkubetriebenen Geräts deutlich verlängert werden.

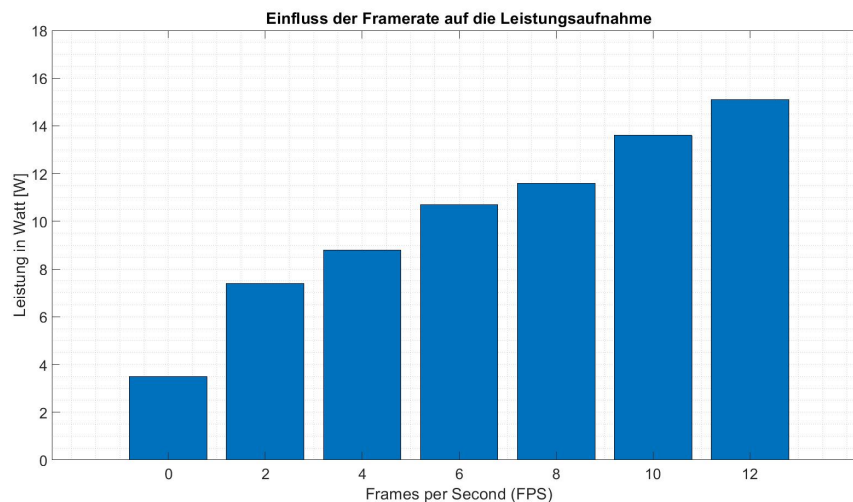


Abbildung 6.13: Zusammenhang von Bildrate und Leistungsaufnahme.

6.4 Bewertung der Implementierung der UART-Schnittstelle

Wie in Abschnitt 5.3.2 mit der Abbildung 5.11 gezeigt, funktioniert die Übertragung der Detektionsergebnisse von der *Video-Detektor Plattform* zur *Motion&Sensor-Plattform* wie konzipiert. In der Abbildung 5.11 ist zu erkennen, dass innerhalb der übertragenen Daten einige Fehlzeichen abgedruckt werden. Diese stammen aus einer nicht zu einhundert Prozent sauber ausgeführten Implementierung des Speichermanagements auf der *Motion&Sensor-Plattform*. Da hier jedoch nur gezeigt werden soll, dass die Datenübertragung generell funktioniert, ist dies zu vernachlässigen. In einer späteren, auf eine bestimmte Anwendung zugeschnittenen Implementierung, kann dies behoben werden. Bei der Datenübertragung muss gewährleistet sein, dass die Daten konsistent sind. Es wird angenommen, dass pro detektiertem Objekt $64 \text{ Byte} = 576 \text{ Bit}$ Daten entstehen. Dies ist ein Mittelwert, der in Tests ermittelt wurde. Mit der eingestellten Datenrate der UART-Schnittstelle von 115200 Bit/s lässt sich berechnen, dass

$$N = \frac{\text{Datenrate}}{\text{Daten pro Objekt}} = \frac{115200 \text{ Bit/s}}{576 \text{ Bit/Objekt}} = 200 \frac{\text{Objekte}}{\text{s}} \quad (6.1)$$

die Übertragung von 200 Objekten pro Sekunde möglich ist. Ebenfalls durch Tests wurde ermittelt, dass im Mittel 25 Objekte pro Bild detektiert werden. Für die Übertragung der 25 Objekte wird folglich eine Zeit von

$$t_{25\text{Objekte}} = \frac{25 \text{ Objekte}}{200 \text{ Objekte/s}} = 125 \text{ ms} \quad (6.2)$$

benötigt. Dies bedeutet, dass die Detektionsergebnisse eines Bildes mindestens für 125 ms verfügbar sein müssen, um die Daten konsistent übertragen zu können. Um die fehlerfreie Übertragung zu gewährleisten, wird ein zusätzlicher Sicherheitspuffer vorgesehen. Es werden immer vier Textdateien und damit vier Detektionsergebnisse mit dem Namen *Frame-A.txt* bis *Frame-D.txt* gespeichert. Nachdem das *D-Frame* gespeichert wurde, wird bei den nächsten zu speichernden Ergebnissen die *A-Frame* Datei wieder überschrieben. Bei einer Bildrate von $10,9 \text{ FPS}$ sind so die Dateien jeweils $t_{\text{Frame}} = \frac{4}{10,9 \text{ FPS}} = 366,9 \text{ ms}$ verfügbar, bevor sie überschrieben werden. Entsprechend der obigen Rechnung ist dies

ausreichend Zeit, um den Inhalt der Datei auszulesen und auf die UART-Schnittstelle zu schreiben. Bei einer Anforderung der *Motion&Sensor-Plattform* wird immer die *A-Frame* Datei ausgelesen und an die *Video-Detektor Plattform* übertragen. Durch den zuvor beschriebenen Mechanismus ist der Inhalt dieser Datei maximal $t_{Frame} = 366,9 \text{ ms}$ alt. Das heißt der Nutzer bekommt im schlechtesten Fall Anweisungen, die auf $366,9 \text{ ms}$ alten Daten basieren. Ob dieser Wert für den praktischen Einsatz des EBF akzeptabel ist oder ob dies zu Problemen führt, muss durch Praxistests validiert werden.

7 Zusammenfassung und Ausblick

Diese Arbeit beschäftigt sich mit dem Projekt *Elektronischen Blindenführhund (EBF)* und dessen zukünftige Hardware- und Softwarekomponenten. Der elektronische Blindenführhund soll sehingeschränkten Personen das alltägliche Leben erleichtern und den Nutzern vor möglichen Gefahrensituationen schützen. Dafür sollen Objekte, die sich in der Umgebung des Nutzers befinden, erkannt und lokalisiert werden. Grundlage dieser Detektion soll ein, auf künstlichen neuronalen Netzen basierender, Algorithmus sein, der mit den Komponenten des EBF berechnet werden soll.

Inhalt dieser Arbeit ist es, eine leistungsstarke Recheneinheit und eine passende Kamera auszuwählen und zu evaluieren, die für die Aufnahme der Bilder und für die Detektion der Objekte in diesen zuständig sind. Außerdem soll ein passender Algorithmus zur Objektdetektion ausgewählt und evaluiert werden. Es ist festgestellt worden, dass die Objektdetektion und die Auswertung der Detektionsergebnisse auf zwei getrennten Geräten erfolgen sollte. Dies vereinfacht die Entwicklung des EBF. Die Detektion erfolgt auf der auszuwählenden Recheneinheit und die Auswertung der Ergebnisse geschieht auf Basis eines hardwarenahen Mikrocontrollers. Mit diesem beschäftigt sich zeitgleich eine andere Arbeit. Zum Datenaustausch zwischen den beiden Komponenten wird eine Kommunikationsschnittstelle benötigt. Auch diese wird innerhalb dieser Arbeit ausgewählt, implementiert und evaluiert.

Das Hauptaugenmerk bei der Auswahl der Recheneinheit ist eine hohe parallele Rechenleistung. Unter anderem deswegen ist der eingebettete Computer *NVIDIA Jetson Xavier NX* ausgewählt worden. Dieser erzielt durch die verbaute GPU eine vergleichsweise hohe Rechenleistung und erfüllt weiter alle Anforderungen, die zusätzlich an die Recheneinheit gestellt worden sind. Der Xavier NX hat ein Linux Betriebssystem, was eine breite Anwendung von Frameworks ermöglicht, die für die Entwicklung und Anwendung von neuronalen Netzen nötig sind. Bei der Kamera, die direkt an die Recheneinheit angeschlossen wird, ist die Wahl auf die *Waveshare IMX219-77* gefallen. Diese Kamera besitzt einen weit verbreiteten und zum Xavier NX kompatiblen Fotosensor und weist eine kompakte Bauform auf. Der Anschluss an die Recheneinheit erfolgt mit der MIPI-

CSI Schnittstelle, mit der hohe Datenraten und damit die Übermittlung von qualitativ hochauflösenden Bildern möglich ist. Als Kommunikationsschnittstelle zur Übermittlung der Detektionsergebnisse wird UART verwendet. Die UART-Schnittstelle ermöglicht bei passender Parametrierung mittlere bis hohe Datenraten, wodurch die Übermittlung umfangreicher Daten zeitnah möglich ist. Außerdem lässt sich diese Schnittstelle sowohl auf der Recheneinheit als auch auf dem Mikrocontroller unkompliziert implementieren. Für die Objektdetektion ist das Detektornetzwerk *You Only Look Once (YOLO)* ausgewählt worden. Dieses Netzwerk ist speziell auf die Objektdetektion in Echtzeit ausgerichtet und eignet sich dadurch gut für die Anwendung im EBF.

Die Evaluierung der Software- und Hardwarekomponenten zeigt, dass eine Objektdetektion mit dem Detektionsnetzwerk YOLO auf der Recheneinheit Xavier NX möglich ist. Die Tests haben ergeben, dass mit dem PowerMode 2 und dem YOLOv3-Netzwerk eine maximale Verarbeitungsrate von 10,9 *FPS* erreicht wird. Das heißt, es können pro Sekunde knapp elf Bilder analysiert und in diesen Bildern Objekte erkannt werden. Dadurch ist es möglich, auch auf sehr spontan auftretende Situationen und Objekte schnell zu reagieren. Die Detektionsergebnisse zeigen, dass auch kleine und räumlich weit entfernte Objekte zuverlässig detektiert werden. Weiter hat die Evaluierung gezeigt, dass die sinnvolle Verwendung der PowerModi des Xavier NX sich positiv auf die Leistungsaufnahme der Recheneinheit auswirkt. Ebenfalls einen Einfluss auf die Leistungsaufnahme hat die Bildrate der Kamera. Maximal sind mit der Recheneinheit 10,9 *FPS* möglich. Wird die Bildrate der Kamera unter diesen Wert eingestellt, sinkt folglich die Verarbeitungsrate in der Recheneinheit, aber gleichzeitig sinkt auch ihre Leistungsaufnahme. Durch die Kombination einer definierten Bildrate der Kamera und den PowerModi der Recheneinheit können die Hardwarekomponenten effizient auf die jeweilige Anwendung eingestellt werden. Dies wirkt sich positiv auf die Einsatzdauer des EBF aus. Für die Übertragung der Detektionsergebnisse zum Mikrocontroller ist zunächst eine Speicherung der Ergebnisse innerhalb des verwendeten Frameworks *Darknet* konzipiert und implementiert worden. Die temporär gespeicherten Detektionsergebnisse werden mit Hilfe eines Python-Skriptes auf Anforderung des Mikrocontrollers blockweise auf die UART-Schnittstelle geschrieben. Von dieser kann der Mikrocontroller die Daten lesen und anschließend passend weiterverarbeiten. Zur Evaluierung der Implementierung der UART- und Speicherfunktionen auf der Recheneinheit ist zusätzlich eine Implementierung auf dem Mikrocontroller erfolgt. Es zeigt sich, dass die Datenübertragung zuverlässig funktioniert und die auf der Recheneinheit entstandenen Detektionsergebnisse dem Mikrocontroller zeitnah zur Verfügung gestellt werden können.

Alle Ergebnisse sind unter Laborbedingungen generiert worden. Ein nächster Schritt

in der Entwicklung des EBF ist der Test unter realen Bedingungen. Dazu müssen die Hardwarekomponenten in ein Gehäuse verbaut werden und dann verschiedene alltägliche Situationen analysiert werden. Dadurch zeigt sich an welchen Stellen weiterer Entwicklungsbedarf besteht. Denkbar ist, dass neben der ausgewählten Normalbildkamera eine zusätzliche Infrarotkamera verwendet wird. Eine Infrarotkamera eignet sich gut, um auch bei schlechten Lichtverhältnissen oder auch bei Nacht, qualitativ hochwertige Bilder zu erhalten. Dadurch wird es möglich, dass die Objektdetektion bei jeder Tageszeit möglich ist. Durch die Konstruktion des Xavier NX als System-on-Module (SoM) ist es ebenfalls denkbar, dass eine eigene Leiterplatte konstruiert wird, die speziell auf die Bedürfnisse des EBF zugeschnitten wird. Auf dieser Leiterplatte muss lediglich die passende Beschaltung und ein passender Steckplatz für das SoM vorgesehen werden. So könnten die aktuell jeweils als eigenständiges Gerät ausgeführte Recheneinheit und Mikrocontroller auf einer gemeinsamen Leiterplatte angeordnet werden, wodurch die fehleranfällige Verdrahtung größtenteils entfallen könnte.

Im Bereich der Software wäre ein möglicher nächster Schritt die Optimierung des ausgewählten Objektdetektornetzwerkes YOLO. Dazu kann einerseits YOLO auf einen eigenen Datensatz trainiert werden, andererseits kann das YOLO-Modell an sich optimiert werden. Dazu bietet sich das Programm *TensorRT* von NVIDIA an. Dieses optimiert ein vorhandenes neuronales Netz so weit wie möglich, was dazu führen soll, dass weniger Berechnungen durchgeführt werden müssen. Dies wiederum würde bedeuten, dass mutmaßlich eine höhere Verarbeitungsrate mit dem optimierten YOLO-Modell möglich wäre. Auch ist die Verwendung der in Abschnitt 4.1 erwähnten USB-Sticks zur Beschleunigung der Berechnungen von neuronalen Netzen ein möglicherweise aussichtsreiches Weiterentwicklungsgebiet. Die Verwendung eines solchen Stick könnte die Rechenleistung weiter beschleunigen. In wie weit die unterschiedlichen Sticks zu dem Jetson Xavier NX passen, muss untersucht werden.

Literaturverzeichnis

- [1] : *Introducing JSON*. – URL <https://www.json.org/json-de.html>. – [Online; Zugriff 17. Januar 2021]
- [2] ADVANCED MICRO DEVICES, INCL: *Homepage*. – URL <https://www.amd.com/>. – [Online; Zugriff 01. Februar 2021]
- [3] ARIANE RÜDIGER: *Schneller als sofort - Was ist Echtzeit und wo braucht man sie?* Juli 2019. – URL <https://www.datacenter-insider.de/was-ist-echtzeit-und-wo-braucht-man-sie-a-845434/>. – [Online; Zugriff 23. Januar 2021]
- [4] ASUSTeK COMPUTER INC: *Tinker Edge R*. – URL <https://tinker-board.asus.com/product/tinker-edge-r.html>. – [Online; Zugriff 13. Januar 2021]
- [5] ASUSTeK COMPUTER INC: *Tinker Edge T*. – URL <https://tinker-board.asus.com/product/tinker-edge-t.html>. – [Online; Zugriff 13. Januar 2021]
- [6] AUNKOFER, Benjamin: *Machine Learning vs Deep Learning – Wo liegt der Unterschied?* Mai 2018. – URL <https://data-science-blog.com/blog/2018/05/14/machine-learning-vs-deep-learning-wo-liegt-der-unterschied/>. – [Online; Zugriff 16. Februar 2021]
- [7] BEAGLEBOARD.ORG FOUNDATION: *BeagleBone® AI*. – URL <http://beagleboard.org/ai>. – [Online; Zugriff 13. Januar 2021]
- [8] BLINDEN- UND SEHBEHINDERTENVERBAND NIEDERSACHSEN E.V.: *Der Tag eines Blinden*. – URL <https://www.blindenverband.org/sie-machen-mit/tag-eines-blinden/>. – [Online; Zugriff 13. Januar 2021]
- [9] BOCHKOVSKIY, Alexey ; WANG, Chien-Yao ; LIAO, Hong-Yuan M.: *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020

- [10] BOSTON.CO.UK: *WHAT IS GPU COMPUTING?*. – URL <https://www.boston.co.uk/info/nvidia-kepler/what-is-gpu-computing.aspx>. – [Online; Zugriff 22. Dezember 2020]
- [11] BUNDESAMT, Statistisches: *Statistik der schwerbehinderten Menschen 2019*. 09 2020
- [12] B.V., Chip I.: *GPU Computing, the basics*:. Juli 2017. – URL <https://www.chipict.com/gpu-computing-the-basics/>. – [Online; Zugriff 22. Dezember 2020]
- [13] CHRISTOFFEL-BLINDENMISSION CHRISTIAN BLIND MISSION E.V.: *Braille – eine Blindenschrift aus Punkten erobert die Welt sehbehinderter Menschen*. – URL <https://www.cbm.de/behinderung-und-sprache/blindenschrift-braille.html>. – [Online; Zugriff 10. Januar 2021]
- [14] COCO COMMON OBJECTS IN CONTEXT: *Overview*. – URL <https://cocodataset.org/#overview>. – [Online; Zugriff 9. Januar 2021]
- [15] CORPORATION, NVIDIA: *NVIDIA Jetson Xavier NX Design Guide*. 08 2020. – Seite 21
- [16] FANTON, Darek: *Was ist Ein Embedded-computer?* Juli 2020. – URL <https://www.onlogic.com/company/io-hub/de/was-ist-ein-embedded-computer/>. – [Online; Zugriff 21. Dezember 2020]
- [17] GMBH, KUNBUS. – URL <https://www.kunbus.de/rs-485.html>. – [Online; Zugriff 02. Februar 2021]
- [18] GOOGLE: *Dev Board*. – URL <https://coral.ai/products/dev-board>. – [Online; Zugriff 13. Januar 2021]
- [19] HARALD KARCHER: *Praxistest: 4K-Auflösung auf dem Desktop*. 2014. – URL <https://www.tecchannel.de/a/praxistest-4k-aufloesung-auf-dem-desktop,2072873>. – [Online; Zugriff 16. Januar 2021]
- [20] HE, Kaiming ; ZHANG, Xiangyu ; REN, Shaoqing ; SUN, Jian: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In: *Lecture Notes in Computer Science* (2014), S. 346–361. – URL http://dx.doi.org/10.1007/978-3-319-10578-9_23. – ISBN 9783319105789
- [21] INTEL: *UP AI Core X Series (Myriad™X)*. – URL <https://up-shop.org/up-ai-core-x-series-myriadtmx.html>. – [Online; Zugriff 13. Januar 2021]

- [22] INTEL: *UP Squared Series*. – URL <https://up-shop.org/up-squared-series.html>. – [Online; Zugriff 13. Januar 2021]
- [23] JACOB SOLAWETZ: *Breaking Down YOLOv4*. Juni 2020. – URL <https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/>. – [Online; Zugriff 09. Februar 2021]
- [24] JEREMY JORDAN: *An overview of object detection: one-stage methods*. Juli 2018. – URL <https://www.jeremyjordan.me/object-detection-one-stage/>. – [Online; Zugriff 29. Januar 2021]
- [25] JIAO, Licheng ; ZHANG, Fan ; LIU, Fang ; YANG, Shuyuan ; LI, Lingling ; FENG, Zhixi ; QU, Rong: A Survey of Deep Learning-Based Object Detection. In: *IEEE Access* 7 (2019), S. 128837–128868. – URL <http://dx.doi.org/10.1109/ACCESS.2019.2939201>. – ISSN 2169-3536
- [26] JOSEPH REDMON: *repository "darknet"*. – URL <https://github.com/pjreddie/darknet>. – [Online; Zugriff 9. Januar 2021]
- [27] JÜRGEN PLATE: *Schnittstellen - Serielle Schnittstelle, USB, Feldbusse, SPI, I2C, 1-Wire etc.* November 2017. – URL <http://www.netzmafia.de/skripten/hardware/Control/schnittstellen.pdf>. – [Online; Zugriff 02. Februar 2021]
- [28] KNOLL, Andreas: *Moderne Bildverarbeitungs-Software verteilt die Aufgaben zwischen CPU und GPU - Die GPU als Co-Prozessor der CPU*. Februar 2011. – URL <https://www.elektroniknet.de/automation/die-gpu-als-co-prozessor-der-cpu.75167.html>. – [Online; Zugriff 01. Februar 2021]
- [29] KOGNITIVE SYSTEME IKS, Fraunhofer-Institut für: *Künstliche Intelligenz (KI) und maschinelles Lernen*. – URL <https://www.iks.fraunhofer.de/de/themen/kuenstliche-intelligenz.html>. – [Online; Zugriff 27. Dezember 2020]
- [30] LENOVO: *ThinkPad E585*. 2018. – URL <https://www.lenovo.com/de/de/laptops/thinkpad/edge-series/ThinkPad-E585/p/22TP2TEE585>. – [Online; Zugriff 20. Januar 2021]
- [31] LIU, Shu ; QI, Lu ; QIN, Haifang ; SHI, Jianping ; JIA, Jiaya: *Path Aggregation Network for Instance Segmentation*. 2018

- [32] MICROSOFT: *Complete multiple tasks with one app.* – URL <https://www.microsoft.com/en-us/ai/seeing-ai>. – [Online; Zugriff 12. Januar 2021]
- [33] MICROSOFT: *Homepage.* – URL <https://www.microsoft.com/de-de/windows>. – [Online; Zugriff 01. Februar 2021]
- [34] NVIDIA CORPORATION: *EINGEBETTETE SYSTEME FÜR AUTONOME MASCHINEN DER NÄCHSTEN GENERATION.* – URL <https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/>. – [Online; Zugriff 13. Februar 2021]
- [35] NVIDIA CORPORATION: *Getting Started With Jetson Xavier NX Developer Kit.* – URL <https://developer.nvidia.com/embedded/learn/get-started-jetson-xavier-nx-devkit>. – [Online; Zugriff 8. Januar 2021]
- [36] NVIDIA CORPORATION: *Homepage.* – URL <https://www.nvidia.com/>. – [Online; Zugriff 01. Februar 2021]
- [37] NVIDIA CORPORATION: *Jetson Benchmarks.* – URL <https://developer.nvidia.com/embedded/jetson-benchmarks>. – [Online; Zugriff 8. Januar 2021]
- [38] NVIDIA CORPORATION: *JETSON TX2.* – URL <https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-tx2/>. – [Online; Zugriff 13. Januar 2021]
- [39] NVIDIA CORPORATION: *JETSON XAVIER NX.* – URL <https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-xavier-nx/>. – [Online; Zugriff 13. Januar 2021]
- [40] NVIDIA CORPORATION: *NVIDIA Developer CUDA Zone.* – URL <https://developer.nvidia.com/cuda-zone>. – [Online; Zugriff 01. Februar 2021]
- [41] NVIDIA CORPORATION: *NVIDIA JETSON NANO.* – URL <https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-nano/>. – [Online; Zugriff 13. Januar 2021]
- [42] NVIDIA CORPORATION: *NVIDIA VOLTA.* – URL <https://www.nvidia.com/de-de/data-center/volta-gpu-architecture/>. – [Online; Zugriff 13. Februar 2021]

- [43] NVIDIA CORPORATION: *RTX. IT'S ON. GEFORCE RTX 2080 Ti*. 2018. – URL <https://www.nvidia.com/de-de/geforce/graphics-cards/rtx-2080-ti>. – [Online; Zugriff 20. Januar 2021]
- [44] OPENCV TEAM: *About*. – URL <https://staging.opencv.org/about/>. – [Online; Zugriff 20. Januar 2021]
- [45] PIERRICK RUGERY: *Explanation of YOLO V4 a one stage detector*. September 2020. – URL <https://becominghuman.ai/explaining-yolov4-a-one-stage-detector-cdac0826cbd7>. – [Online; Zugriff 09. Februar 2021]
- [46] PYTHON SOFTWARE FOUNDATION: *Python Website*. – URL <https://www.python.org/>. – [Online; Zugriff 18. Januar 2021]
- [47] RAFFAELLO BONGHI: *jtop*. August 2018. – URL https://github.com/rbonghi/jetson_stats/wiki/jtop. – [Online; Zugriff 22. Januar 2021]
- [48] RASPBERRY PI FOUNDATION: *Raspberry Pi 4*. – URL <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/>. – [Online; Zugriff 13. Januar 2021]
- [49] REDMON, Joseph ; DIVVALA, Santosh ; GIRSHICK, Ross ; FARHADI, Ali: *You Only Look Once: Unified, Real-Time Object Detection*. 2016
- [50] REDMON, Joseph ; FARHADI, Ali: *YOLOv3: An Incremental Improvement*. 2018
- [51] SCHAARSCHMIDT, Miriam: *Was ist der Unterschied zwischen KI und maschinellem Lernen?* Juni 2020. – URL https://www.crowdguru.de/blog/unterschied_ki_maschinelles_lernen-was-ist-maschinelles_lernen/. – [Online; Zugriff 27. Dezember 2020]
- [52] SONY: *IMX219PQH5-C*. – URL https://publiclab.org/system/images/photos/000/023/294/original/RASPBERRY_PI_CAMERA_V2_DATASHEET_IMX219PQH5_7.0.0_Datasheet_XXX.PDF. – [Online; Zugriff 14. Februar 2021]
- [53] TECHTARGET- REDAKTION: *Embedded System (Eingebettetes System)*. – URL <https://whatis.techtarget.com/de/definition/Eingebettetes-System>. – [Online; Zugriff 01. Februar 2021]

- [54] TEXAS INSTRUMENTS INCORPORATED: *Code Composer Studio (CCS) Integrated Development Environment (IDE)*. – URL <https://www.ti.com/tool/CCSTUDIO>. – [Online; Zugriff 18. Januar 2021]
- [55] TEXAS INSTRUMENTS INCORPORATED: *TM4C1294NCPDT - IoT enabled High performance 32-bit ARM® Cortex®-M4F based MCU*. – URL <https://www.ti.com/product/TM4C1294NCPDT>. – [Online; Zugriff 12. Januar 2021]
- [56] THE LINUX FOUNDATION®: *Homepage*. – URL <https://www.linuxfoundation.org/>. – [Online; Zugriff 01. Februar 2021]
- [57] TIAN, Zhi ; SHEN, Chunhua ; CHEN, Hao ; HE, Tong: *FCOS: Fully Convolutional One-Stage Object Detection*. 2019
- [58] VOLKER LENK, Prof. Dr. med. Alexander Schuster M.: *Augenkrankheiten – Zahlen für Deutschland*. Oktober 2018. – URL <https://www.woche-des-sehens.de/infothek/zahlen-und-fakten/augenkrankheiten-zahlen-fuer-deutschland/>. – [Online; Zugriff 20. Dezember 2020]
- [59] WANG, Chien-Yao ; LIAO, Hong-Yuan M. ; YEH, I-Hau ; WU, Yueh-Hua ; CHEN, Ping-Yang ; HSIEH, Jun-Wei: *CSPNet: A New Backbone that can Enhance Learning Capability of CNN*. 2019
- [60] WARKENTIN, Lukas ; FREWER, Niklas: *Bachelor Projekt - Der elektronische Blindenführhund*. April 2020. – [Unpubliziertes Dokument, Hochschule für angewandte Wissenschaften Hamburg (HAW)]
- [61] WAVESHARE.COM: *IMX219-77 Camera, 77° FOV, Applicable for Jetson Nano*. – URL <https://www.waveshare.com/imx219-77-camera.htm>. – [Online; Zugriff 6. Januar 2021]
- [62] WIKIPEDIA: *Camera Serial Interface* — *Wikipedia, Die freie Enzyklopädie*. 2020. – URL https://en.wikipedia.org/w/index.php?title=Camera_Serial_Interface&oldid=977963951. – [Online; Zugriff 6. Januar 2021]
- [63] WIKIPEDIA: *General Purpose Computation on Graphics Processing Unit* — *Wikipedia, Die freie Enzyklopädie*. 2020. – URL https://de.wikipedia.org/w/index.php?title=General_Purpose_Computation_on_Graphics_Processing_Unit&oldid=197051755. – [Online; Zugriff 22. Dezember 2020]

- [64] WIKIPEDIA: *Künstliche Intelligenz* — *Wikipedia, Die freie Enzyklopädie*. 2020. – URL https://de.wikipedia.org/w/index.php?title=K%C3%BCnstliche_Intelligenz&oldid=206473463. – [Online; Zugriff 23. Dezember 2020]
- [65] WIKIPEDIA: *Maschinelles Lernen* — *Wikipedia, Die freie Enzyklopädie*. 2020. – URL https://de.wikipedia.org/w/index.php?title=Maschinelles_Lernen&oldid=204714424. – [Online; Zugriff 23. Dezember 2020]
- [66] WIKIPEDIA: *Universal Asynchronous Receiver Transmitter* — *Wikipedia, Die freie Enzyklopädie*. 2020. – URL https://de.wikipedia.org/w/index.php?title=Universal_Asynchronous_Receiver_Transmitter&oldid=206516581. – [Online; Zugriff 21. Dezember 2020]
- [67] WIKIPEDIA: *Black Box (Systemtheorie)* — *Wikipedia, Die freie Enzyklopädie*. 2021. – URL [https://de.wikipedia.org/w/index.php?title=Black_Box_\(Systemtheorie\)&oldid=208445104](https://de.wikipedia.org/w/index.php?title=Black_Box_(Systemtheorie)&oldid=208445104). – [Online; Zugriff 14. Februar 2021]
- [68] WIKIPEDIA: *Universal Serial Bus* — *Wikipedia, Die freie Enzyklopädie*. 2021. – URL https://de.wikipedia.org/w/index.php?title=Universal_Serial_Bus&oldid=208308842. – [Online; Zugriff 02. Februar 2021]
- [69] WUTTKE, Laurenz: *Künstliche Neuronale Netzwerke: Definition, Einführung, Arten und Funktion*. – URL <https://datasolut.com/neuronale-netzwerke-einfuehrung/>. – [Online-Video; Zugriff 01. Februar 2021]
- [70] WÜRTZ, Udo: *CNNs Convolutional Neural Networks Basiswissen*. Dezember 2019. – URL <https://www.youtube.com/watch?v=OV0KXyYpEZY&t=2190s>. – [Online-Video; Zugriff 01. Februar 2021]

A Anhang

Table 10: Comparison of the speed and accuracy of different object detectors on the MS COCO dataset (test-dev 2017). (Real-time detectors with FPS 30 or higher are highlighted here. We compare the results with batch=1 without using tensorRT.)

Method	Backbone	Size	FPS	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
YOLOv4: Optimal Speed and Accuracy of Object Detection									
YOLOv4	CSPDarknet-53	416	96 (V)	41.2%	62.8%	44.3%	20.4%	44.4%	56.0%
YOLOv4	CSPDarknet-53	512	83 (V)	43.0%	64.9%	46.5%	24.3%	46.1%	55.2%
YOLOv4	CSPDarknet-53	608	62 (V)	43.5%	65.7%	47.3%	26.7%	46.7%	53.3%
EfficientDet: Scalable and Efficient Object Detection [77]									
EfficientDet-D0	Efficient-B0	512	62.5 (V)	33.8%	52.2%	35.8%	12.0%	38.3%	51.2%
EfficientDet-D1	Efficient-B1	640	50.0 (V)	39.6%	58.6%	42.3%	17.9%	44.3%	56.0%
EfficientDet-D2	Efficient-B2	768	41.7 (V)	43.0%	62.3%	46.2%	22.5%	47.0%	58.4%
EfficientDet-D3	Efficient-B3	896	23.8 (V)	45.8%	65.0%	49.3%	26.6%	49.4%	59.8%
Learning Spatial Fusion for Single-Shot Object Detection [48]									
YOLOv3 + ASFF*	Darknet-53	320	60 (V)	38.1%	57.4%	42.1%	16.1%	41.6%	53.6%
YOLOv3 + ASFF*	Darknet-53	416	54 (V)	40.6%	60.6%	45.1%	20.3%	44.2%	54.1%
YOLOv3 + ASFF*	Darknet-53	608 ×	45.5 (V)	42.4%	63.0%	47.4%	25.5%	45.7%	52.3%
YOLOv3 + ASFF*	Darknet-53	800 ×	29.4 (V)	43.9%	64.1%	49.2%	27.0%	46.6%	53.4%
HardNet: A Low Memory Traffic Network [4]									
RFBNet	HardNet68	512	41.5 (V)	33.9%	54.3%	36.2%	14.7%	36.6%	50.5%
RFBNet	HardNet85	512	37.1 (V)	36.8%	57.1%	39.5%	16.9%	40.5%	52.9%
Focal Loss for Dense Object Detection [45]									
RetinaNet	ResNet-50	640	37 (V)	37.0%	-	-	-	-	-
RetinaNet	ResNet-101	640	29.4 (V)	37.9%	-	-	-	-	-
RetinaNet	ResNet-50	1024	19.6 (V)	40.1%	-	-	-	-	-
RetinaNet	ResNet-101	1024	15.4 (V)	41.1%	-	-	-	-	-
SM-NAS: Structural-to-Modular Neural Architecture Search for Object Detection [88]									
SM-NAS: E2	-	800 × 600	25.3 (V)	40.0%	58.2%	43.4%	21.1%	42.4%	51.7%
SM-NAS: E3	-	800 × 600	19.7 (V)	42.8%	61.2%	46.5%	23.5%	45.5%	55.6%
SM-NAS: E5	-	1333 × 800	9.3 (V)	45.9%	64.6%	49.6%	27.1%	49.0%	58.0%
NAS-FPN: Learning scalable feature pyramid architecture for object detection [17]									
NAS-FPN	ResNet-50	640	24.4 (V)	39.9%	-	-	-	-	-
NAS-FPN	ResNet-50	1024	12.7 (V)	44.2%	-	-	-	-	-
Bridging the Gap Between Anchor-based and Anchor-free Detection via Adaptive Training Sample Selection [94]									
ATSS	ResNet-101	800 ×	17.5 (V)	43.6%	62.1%	47.4%	26.1%	47.0%	53.6%
ATSS	ResNet-101-DCN	800 ×	13.7 (V)	46.3%	64.7%	50.4%	27.7%	49.8%	58.4%
RDSNet: A New Deep Architecture for Reciprocal Object Detection and Instance Segmentation [83]									
RDSNet	ResNet-101	600	16.8 (V)	36.0%	55.2%	38.7%	17.4%	39.6%	49.7%
RDSNet	ResNet-101	800	10.9 (V)	38.1%	58.5%	40.8%	21.2%	41.5%	48.2%
CenterMask: Real-Time Anchor-Free Instance Segmentation [40]									
CenterMask	ResNet-101-FPN	800 ×	15.2 (V)	44.0%	-	-	25.8%	46.8%	54.9%
CenterMask	VoVNet-99-FPN	800 ×	12.9 (V)	46.5%	-	-	28.7%	48.9%	57.2%

Abbildung A.1: Vergleich verschiedener Objektdetektoren [9].

Pos.	Hersteller	Bezeichnung	Teilenummer	Betriebssystem	CPU	CPU Takt-Frequenz
1	Google	Coral Dev Board	G950-01455-01	Mendei (Debian Linux)	NXP i.MX 8M SoC (quad Cortex-A53, Cortex-M4F)	max 1,5 GHz
	Google	Coral USB Accelerator	G950-01456-01	setzt Linux, MacOS oder Windows 10 voraus	-	
2	Intel	UP Squared Pentium Quad Core 08164	UPS-APLP4-A20-0864	Microsoft Windows 10 full version, Linux, Android	Intel® Pentium™ M4200 (up to 2.5 GHz)	typical 1,1 GHz
	Intel	UP AI Core X Series (Myriad™ X)	PER-TAIX2-A10-001	nicht notwendig	nicht notwendig	700 MHz
	Intel	Intel® Neural Compute Stick 2		setzt Windows®, 10, Ubuntu®, or macOS® voraus		
3	Asus	Tinker Edge R	A52-2000003	Debian 9 / Android 9	Dual-core ARM® Cortex®-A72 @ 1,8 GHz	1,8 GHz
4	Asus	Tinker Edge T	A52-2000002	Debian 9	Quad-core ARM® Cortex®-A53 @ 1,4 GHz	1,4 GHz
5	NVIDIA	Jetson Nano Developer Kit		Linux, spezielle Version "JetPack" extra für die Jetson Platform	Quad-core ARM A57	@ 1,43 GHz
	NVIDIA	Jetson Nano SoM		siehe Position 5	siehe Position 5	siehe Position 5
6	NVIDIA	Jetson TX2 Developer Kit		siehe Position 5	Dual-Core NVIDIA Denver 2 64-Bit CPU and Quad-Core ARM® Cortex®-A57 complex	1,2 GHz
	NVIDIA	Jetson TX2 4GB SoM	900-83469-0080-000	siehe Position 5	siehe Position 6	siehe Position 6
	NVIDIA	Jetson TX2 SoM	900-83310-0001-000	siehe Position 5	siehe Position 6	siehe Position 6
	NVIDIA	Jetson TX2i SoM		siehe Position 5	siehe Position 6	siehe Position 6
7	NVIDIA	Jetson Xavier NX Develop Kit	945-83516-0000-000	siehe Position 5	NVIDIA Carmel ARM v8.2 64-Bit-CPU, 6 Cores 6 MB L2 + 4 MB L3	1,4/1,6 GHz, je nach CPU-Mode
	NVIDIA	Jetson Xavier NX SoM	900-83668-0000-000	siehe Position 5	siehe Position 7	siehe Position 7
8	NVIDIA	Jetson AGX Xavier Develop Kit		siehe Position 5	8-core NVIDIA Carmel Arm v8.2 64-bit CPU 8MB L2 + 4MB L3	2,265 GHz
	NVIDIA	Jetson AGX Xavier SoM (8GB Version)	900-82868-0060-000	siehe Position 5	siehe Position 8	siehe Position 8
9	BeagleBoard	BeagleBone AI	358-BBONE-AI	Debian GNU/Linux	Texas Instruments Sitara AM5729	
10	Raspberry	Raspberry Pi 4 Model B Set	RP4 BBDL 8GB	Linux	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC	1,5 GHz

Abbildung A.2: Tabelle des Plattformvergleichs, erster Teil.

Pos.	Hersteller	Bezeichnung	GPU	AI Core	Performance	Arbeitsspeicher	Speicher
1	Google	Coral Dev Board	Integrated GC7000 Lite Graphics	Google Edge TPU coprocessor	4 TOPS (int8)	1GB LPDDR4	8 GB eMMC, microSD slot
	Google	Coral USB Accelerator	-	Google Edge TPU coprocessor	4 TOPS (int8)	-	-
2	Intel	UP Squared Pentium Quad Core 08164	Intel® HD Graphics 500	Intel® Movidius™ Myriad™ X VPU 2485	keine Infos zu finden	8GB	64GB (mit 128GB erhältlich)
	Intel	UP AI Core X Series (Myriad™ X)	-	-	-	-	4GB
	Intel	Intel® Neural Compute Stick 2	-	Intel® Movidius™ Myriad™ X	-	-	-
3	Asus	Tinker Edge R	ARM® Mali™-T860 MP4 GPU @ 800 MHz	Rockchip MPU	3 TOPS	4GB LPDDR4	16GB eMMC
4	Asus	Tinker Edge T	GC7000 Lite	Google Edge TPU ML accelerator coprocessor	4 TOPS	LPDDR4 1.5B	8GB eMMC, microSD slot
5	NVIDIA	Jetson Nano Developer Kit	128-core Maxwell	GPU	472 GFLOPS	4 GB LPDDR4	microSD-Card slot
	NVIDIA	Jetson Nano SoM	siehe Position 5	siehe Position 5	siehe Position 5	siehe Position 5	16 GB eMMC 5.1
	NVIDIA	Jetson TX2 Developer Kit	NVIDIA Pascal™ Architektur mit 256 NVIDIA CUDA Recheneinheiten	GPU	1,33/1,26 TFLOPS je nach TX2-SoM	LPDDR4	32 GB eMMC 5.1
	NVIDIA	Jetson TX2 4GB SoM	siehe Position 6	siehe Position 6	1,33 TFLOPS	4 GB 128-bit LPDDR4 51,2GB/s	16 GB eMMC 5.1
	NVIDIA	Jetson TX2 SoM	siehe Position 6	siehe Position 6	1,33 TFLOPS	8 GB 128-bit LPDDR4 59,7GB/s	32 GB eMMC 5.1
	NVIDIA	Jetson TX2i SoM	siehe Position 6	siehe Position 6	1,26 TFLOPS	8 GB 128-bit LPDDR4 51,2GB/s	32 GB eMMC 5.1
7	NVIDIA	Jetson Xavier NX Develop Kit	384-core NVIDIA Volta™ GPU with 48 Tensor Cores	2x NVIDIA Engines 7-Way VLIW Vision Processor	21 TOPS	8 GB 128-bit LPDDR4x 51,2GB/s	microSD-Card slot
	NVIDIA	Jetson Xavier NX SoM	siehe Position 7	siehe Position 7	siehe Position 7	siehe Position 7	16 GB eMMC 5.1
	NVIDIA	Jetson AGX Xavier Develop Kit	512-core NVIDIA Volta™ GPU with 64 Tensor Cores	2x NVIDIA Engines 7-Way VLIW Vision Processor	32 TOPS	LPDDR4x 136,5GB/s / 8 GB	32 GB eMMC 5.1
	NVIDIA	Jetson AGX Xavier SoM (8GB Version)	siehe Position 8	siehe Position 8	siehe Position 8	8 GB	siehe Position 8
9	BeagleBoard	BeagleBone AI	integriert in CPU	4x embedded-vision-engine (EVE) cores in CPU integriert	13,5 GFLOPS	1GB DDR3	16GB eMMC, microSD slot
10	Raspberrypi	Raspberrypi 4 Model B Set	integriert in CPU	fehlt	13,5 GFLOPS	8GB LPDDR4	microSD-Card slot

Abbildung A.3: Tabelle des Plattformvergleichs, zweiter Teil.

Pos.	Hersteller	Bezeichnung	Power-Computing Technik	spezieller Anwendungsbereich	Display-Anschluss	Kamera-Anschluss
1	Google	Coral Dev Board	Google TPU	maschinelles lernen, IoT Security	HDMI, MIPI-DSI	MIPICSI2
	Google	Coral USB Accelerator	Google TPU	maschinelles lernen	nicht notwendig	nicht notwendig
2	Intel	UP Squared Pentium Quad Core 08164	Intel AI Boost notwendig		HDMI, DP, eDP	MIPICSI2 2 lane, 4 lane
	Intel	UP AI Core X Series (MyriadMX)	1x VPU	AI Inference	nicht notwendig	nicht notwendig
	Intel	Intel® Neural Compute Stick 2	VPU	deep learning AI Inference	nicht notwendig	nicht notwendig
3	Asus	Tinker Edge R			HDMI, USB-C, MIPI-DSI	1x 22-pin MIPICSI-2 (4 lane)
4	Asus	Tinker Edge T	Google TPU		HDMI, MIPI-DSI	1x 22-pin MIPICSI-2/DSI (4 lane) 2x 24-pin MIPICSI-2
5	NVIDIA	Jetson Nano Developer Kit	CUDA	run multiple neural networks in parallel, parallel image processing, maximum Power through GPU usage	HDMI, DP	2x MIPICSI-2 DPHY lanes up to 4 cameras
	NVIDIA	Jetson Nano SoM	siehe Position 5	siehe Position 5	2 multi-mode DP 1,2/eDP 1,4/HDMI 2,0 1x2 DSI (1,5Gbps/lane)	12 lanes MIPICSI-2 D-PHY 1,1 (up to 18 Gbps)
6	NVIDIA	Jetson TX2 Developer Kit	siehe Position 5	siehe Position 5	HDMI	Bis zu 6 Kameras (2 Lanes) CSI2 D-PHY 1,2 (2,5 Gbps/Lane)
	NVIDIA	Jetson TX2 4GB SoM	siehe Position 5	siehe Position 5	2 multi-mode DP 1,2/eDP 1,4/HDMI 2,0	siehe Position 6
	NVIDIA	Jetson TX2 SoM	siehe Position 5	siehe Position 5	2 multi-mode DP 1,2/eDP 1,4/HDMI 2,0	siehe Position 6
	NVIDIA	Jetson TX2i SoM	siehe Position 5	siehe Position 5	2 multi-mode DP 1,2/eDP 1,4/HDMI 2,0	siehe Position 6
7	NVIDIA	Jetson Xavier NX Develop Kit	CUDA + Vision Engines	siehe Position 5	HDMI und DP	2x MIPICSI-2 D-PHY Spuren
	NVIDIA	Jetson Xavier NX SoM	siehe Position 7	siehe Position 5	2 multi-mode DP 1,4/eDP 1,4/HDMI 2,0	Up to 6 cameras (24 via virtual channels) 12 lanes MIPICSI-2 D-PHY 1,2 (up to 30 Gbps)
8	NVIDIA	Jetson AGX Xavier Develop Kit	siehe Position 7	siehe Position 5	HDMI 2.0	(16x) CSI-2 Lanes
	NVIDIA	Jetson AGX Xavier SoM (8GB Version)	siehe Position 7	siehe Position 5	3 multi-mode DP 1,4/eDP 1,4/HDMI 2,0	Up to 6 cameras (36 via virtual channels) 16 lanes MIPICSI-2) 8 lanes SLVS-EC D-PHY 1,2 (up to 40 Gbps) C-PHY 1,1 (up to 31 Gbps)
9	BeagleBoard	BeagleBone AI			microHDMI	
10	Raspberrypi	Raspberrypi 4 Model B Set	fehlt	fehlt	2x micro-HDMI ports 2-lane MIPI DSI display port	2-lane MIPICSI camera port

Abbildung A.4: Tabelle des Plattformvergleichs , dritter Teil.

Pos.	Hersteller	Bezeichnung	Netzwerk	Kommunikationsschnittstellen	unterstützte AI/ML-Frameworks	Leistungsaufnahme
1	Google	Coral Dev Board	Gb LAN, Wifi, BL nicht notwendig	USB, 2x UART, 2x I2C, 2x SPI, 16 GPIO, 4x PWM USB 3.0	TensorFlow Lite, AutoML, Vision Edge	ca. 10W bei 5V ca. 4W
2	Intel	UP Squared Pentium Quad Core 08164	2x Gb LAN nicht notwendig	USB 2.0 und 3.0, 2x UART, 1x SATA 3.0, 40-pin GP-Bus mPCIe	TensorFlow, Caffe, MXNet, Kaldi, ONNX TensorFlow, Caffe, MXNet, Kaldi, ONNX and PaddlePaddle via an ONNX conversion conversion from Caffe, TensorFlow, TensorFlow Lite, ONNX, Darknet and more.	18W -
3	Asus	Intel® Neural Compute Stick 2	nicht notwendig	4x USB 3.2 Gen1, 2x SPI, 2x UART, 2x I2C, 2x PWM uvm.		up to 65W
4	Asus	Tinker Edge R Tinker Edge T	Gb LAN Gb LAN	3x USB 3.2, SPI, I2C, UART, PWM		up to 45W
5	NVIDIA	Jetson Nano Developer Kit	Gb LAN	4x USB 3.0, 1x USB 2.0, GPIO, I2C, I2S, SPI, UART	PyTorch, MXNet, TensorFlow, Matlab, Caffe, Chainer, PaddlePaddle, JetPack SDK	max. 10W
	NVIDIA	Jetson Nano SoM	Gb LAN		siehe Position 5	
6	NVIDIA	Jetson TX2 Developer Kit	Gb LAN, wLAN, BL	GPIO, I2C, I2S, SPI, CAN, UART, USB 3.0 + USB 2.0	siehe Position 5	max. 20W
	NVIDIA	Jetson TX2 4GB SoM	Gb LAN	siehe Position 6	siehe Position 5	max. 15W
	NVIDIA	Jetson TX2 SoM	Gb LAN, wLAN	siehe Position 6	siehe Position 5	max. 20W
	NVIDIA	Jetson TX2i SoM	Gb LAN	siehe Position 6	siehe Position 5	max. 20W
7	NVIDIA	Jetson Xavier NX Develop Kit	Gb LAN, wLAN, BL	4x USB 3.1, USB 2.0 Micro-B, GPIOs, I2C, I2S, SPI, UART	siehe Position 5	max. 15W
	NVIDIA	Jetson Xavier NX SoM	Gb LAN	siehe Position 7	siehe Position 5	siehe Position 7
8	NVIDIA	Jetson AGX Xavier Develop Kit	Gb LAN	2x USB 3.1, UART, SPI, CAN, I2C, I2S, DMIC, GPIOs	siehe Position 5	max. 30W
	NVIDIA	Jetson AGX Xavier SoM (8GB Version)	Gb LAN	siehe Position 8	siehe Position 5	siehe Position 8
9	BeagleBoard	BeagleBone AI	Gb LAN, BL, wLAN	2x USB, 16-bit LCD interfaces, 4x UART, 2x I2C, 2x I2C, GPIOs	spezielle IDL machine learning library nötig für EVE, alle gängigen Frameworks, nichts speziell darauf zugeschnitten	
10	Raspberrypi	Raspberrypi 4 Model B Set	Gb LAN, BL, wLAN	2x USB 3.0, 2x USB 2.0		min. 15W

Abbildung A.5: Tabelle des Plattformvergleichs, vierter Teil.

Pos.	Hersteller	Bezeichnung	mechanische Abmessungen	Erscheinung	Preis	Verfügbarkeit	EDL	Händler
1	Google	Coral Dev Board	85x56 mm	Miz 19	119,221	Ab Lager		Mouser
	Google	Coral USB Accelerator	65x30 mm	Miz 19	53,731	Ab Lager		Mouser
2	Intel	UP Squared Pentium Quad Core 08164	86,5x30 mm	-	255,221	7 Tage		Up-Squared Shop
	Intel	UP AI Core X Series (Myriad™) X	51x30 mm	-	67,791	7 Tage		Up-Squared Shop
	Intel	Intel® Neural Compute Stick 2	72,5x27x14 mm	-	66,911	Ab Lager		Mouser
3	Asus	Tinker Edge R	100x72 mm	Mai 20	232,971	Ab Lager		Welectron
4	Asus	Tinker Edge T	85,5x54 mm	Jan 20	154,991	Ab Lager		Welectron
5	NVIDIA	Jetson Nano Developer Kit	100x80x29 mm		109,001	Ab Lager		NVIDIA
	NVIDIA	Jetson Nano SoM	69x45 mm		84,501	Ab Lager	Jan 25	Arrow
6	NVIDIA	Jetson TX2 Developer Kit	100x100 mm	Miz 17	429,001	Ab Lager		Arrow
	NVIDIA	Jetson TX2 4GB SoM	87x50 mm		255,221	Ab Lager	Jan 25	Arrow
	NVIDIA	Jetson TX2 SoM			385,051	Ab Lager	Jan 25	Arrow
	NVIDIA	Jetson TX2i SoM			639,331	Ab Lager	Apr 28	Arrow
7	NVIDIA	Jetson Xavier NX Develop Kit	103x90,5x34 mm	Mai 20	429,001	Ab Lager		NVIDIA/Arrow
	NVIDIA	Jetson Xavier NX SoM	69,6x45 mm		397,771	Ab Lager	Jan 26	NVIDIA
8	NVIDIA	Jetson AGX Xavier Develop Kit			749,001	Ab Lager		NVIDIA
	NVIDIA	Jetson AGX Xavier SoM (8GB Version)	100x87 mm		579,731	Ab Lager	Jan 25	Arrow
9	BeagleBoard	BeagleBone AI	89x54x15 mm		115,181	Ab Lager		Mouser
10	Raspberrypi	Raspberrypi 4 Model B Set	85x56 mm		107,131	Ab Lager	Jan 26	Reichelt

Abbildung A.6: Tabelle des Plattformvergleichs, fünfter Teil.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort

Datum

Unterschrift im Original