Personenerkennung durch schwache FPGAs in autonomen Fahrzeugen mittels Neuronaler Netze

Projektbericht

Philipp Altnickel
Ferhat Cansu
Ismail Sastim
Johannes Steffen
Mattia Uhlenbrock
Prof. Dr. Jan Bredereke

Bremen University of Applied Sciences
Fakultät 4: Elektrotechnik und Informatik
25. Oktober 2021 – 01. März 2022

Zusammenfassung

Geschrieben von Mattia Uhlenbrock

Künstliche Neuronale Netze erweisen sich zunehmend als geeigneter um Aufgaben zu bewältigen, die bei einer Lösung durch herkömmliche Software mit großen Aufwand verbunden sind. Um Objekterkennung umzusetzen erweisen sich gefaltete Netze als besonders geeigneter Ansatz. Diese verarbeiten Informationen hoch parallel, was auf schwacher aber auf parallele Ausführung spezialisierter Hardware genutzt werden kann. Um von diese Eigenschaft zu profitieren, können Neuronale Netze mit Hilfe des FINN-Frameworks auf FPGAs einprogrammiert werden. In dieser Projektarbeit soll Personenerkennung mit einem Neuronalen Netz auf einem Digilent PYNQ-Z1 mit Zynq SoC implementiert werden. Die Erkennung wird für ein Verfolgungsszenario mit einem autonomen Fahrzeug verwendet. Dabei wird die Person in einem von einer im Fahrzeug integrierten Kamera aufgenommenen Bild in Echtzeit lokalisiert. Für das Training des Neuronalen Netzes wird ein auf dem COCO-Datensatz basierender Subdatensatz erstellt. Das trainierte Netz kann mittels des FINN-Frameworks auf dem FPGA des autonomen Fahrzeugs installiert werden. Die Fahrzeugsteuerung zum Verfolgen einer Person anhand der Lokalisierung der Person im Bild ist ebenfalls im Rahmen dieses Projektes umgesetzt worden. Die Ergebnisse der Evaluation zeigen, dass sich die Personenerkennung und -verfolgung mit den gewählten Mitteln umsetzen lässt. Allerdings kann auch die Steuerung des Fahrzeugs durch eine Gestenerkennung erweitert werden und die Perfomance des Systems noch weiter verbessert werden.

Inhaltsverzeichnis

1 Einleitung **5**
- 1.1 Projektkontext . . . 5
- 1.2 Projektziele . . . 5
- 1.3 Methodik . . . 6

2 Grundlagen **7**
- 2.1 Neuronale Netze . . . 7
  - 2.1.1 Convolutional Neural Network (CNN) . . . 7
  - 2.1.2 Pooling . . . 7
  - 2.1.3 Convolutional Layer . . . 8
  - 2.1.4 Aktivierungsfunktionen . . . 11
  - 2.1.5 Fully Connected Layer . . . 11
  - 2.1.6 Dropout Layer . . . 12
  - 2.1.7 Overfitting . . . 12
- 2.2 Copy number variation (CNV) . . . 13
  - 2.2.1 You Only Look Once (YOLO) . . . 15
  - 2.2.2 Single Shot Detector (SSD) . . . 17
- 2.3 Verfügbare Datensätze . . . 19
- 2.4 Tiling . . . 19
- 2.5 Entwicklungsumgebung . . . 20
  - 2.5.1 PyTorch . . . 20
  - 2.5.2 Brevitas . . . 20
  - 2.5.3 FINN Framework und HLS . . . 20

3 Konzeption des Gesamtsystems **21**
- 3.1 Aktueller Stand als Grundlage der Weiterentwicklung . . . 21
- 3.2 Auswahl des Neuronalen Netzes . . . 21
  - 3.2.1 YOLO . . . 21
  - 3.2.2 SSD . . . 22
  - 3.2.3 One-Class Convolutional Neural Network . . . 23
  - 3.2.4 CNV mit zusätzlichen Tiling . . . 23
- 3.3 Auftrennung von Personen- und Gestenerkennung . . . 23
- 3.4 Datenfluss auf dem autonomen Fahrzeug . . . 23

4 Neuronales Netz zur Personenerkennung **26**
- 4.1 Trainingsdaten . . . 26
  - 4.1.1 Ausgangsbilder . . . 26
  - 4.1.2 Auswahl geeigneter Bilder . . . 27
  - 4.1.3 Generierung des Trainingsdatensatzes . . . 28
  - 4.1.4 Tiling der Trainingsdaten . . . 28
- 4.2 Training . . . 31
  - 4.2.1 Vorbereitung . . . 31
  - 4.2.2 Durchführung . . . 31
- 4.3 Übersetzung für das FPGA . . . 31

5 Einbindung in das System **32**
- 5.1 Übertragung des Netzes . . . 32
- 5.2 Anpassungen an der Base Station . . . 32
- 5.3 Anpassungen am Tiling . . . 32
- 5.4 Anpassung der Fahrzeug Steuerung . . . 33
- 5.5 Optimierung des Fahrzeugs . . . 33

6 Evaluation **35**

6.1 Genauigkeit des Personennetzes . . . 35
6.2 Ressourcenverbrauch . . . 35
6.3 Systemtest . . . 36
6.3.1 High FPS Test . . . 36
6.3.2 Low FPS Test . . . 37
6.3.3 Systemtest . . . 37
6.3.4 Vergleich der Frames per Second . . . 37
6.4 Verifikation der Projektziele . . . 38

7 Ergebnisse und Ausblick **39**
7.1 Zusammenfassung der Ergebnisse . . . 39
7.2 Ausblick . . . 39
7.2.1 Verbesserung der Personenerkennung . . . 39
7.2.2 Überarbeitung der Skripte . . . 39
7.2.3 Einbinden der Gestenerkennung . . . 40
7.2.4 Optimierung CPU Inferenz . . . 40
7.2.5 Weiterführende Konzeption eines selbst entworfenen Neuronalen Netzes . . . 41
7.2.6 Verbesserung am Fahrzeug . . . 41
7.2.7 Sonstiges . . . 42

Literatur **43**

A Abbildungsverzeichnis **45**

B Tabellenverzeichnis **46**

C Anleitung: Training auf Google Cloud Platform **47**

D Anleitung: Vorgehen mit dem COCO Datensatz **51**

E Anleitung: Inbetriebnahme Pynq-Z1 via ad-hoc Netz **53**

F Testszenario zur Überprüfung der Projektziele **56**

G Testszenario zur Überprüfung der Teilprojektziele der Personenerkennung **57**

1 Einleitung

1.1 Projektkontext

Geschrieben von Mattia Uhlenbrock

Die Verwendung Neuronaler Netze für bspw. Bilderkennung erfordert in der Regel viel Rechenleistung. Es gibt allerdings ein berechtigtes Interesse Aufgaben wie Bild- und Audioerkennung auf autonomer leistungsschwacher Hardware auszuführen.

Ein möglicher Anwendungsbereich ist die Raumfahrt, in welcher Systeme aufgrund der Weltraumstrahlung strahlensicher oder zumindest strahlungstolerant ausgelegt werden. Die strahlentolerante und -sichere Hardware ist aufgrund struktureller Eigenschaften nicht sonderlich leistungsstark. Oftmals werden deshalb bestimmte FPGAs (Field Programmable Gate Array) für On-Board-Rechner verwendet. Außerdem kann keine ständige Verbindung zu einem Bodensystem gewährleistet werden, welches die Rechenleistung bereitstellen könnte. Da das betreiben von Neuronalen Netzen eine hoch parallele Aufgabe ist und sich FPGAs durch besonders gute Eignung für hoch parallele Aufgaben auszeichnen, beschäftigt sich dieses Projekt mit dem Lösen von Bilderkennungsaufgaben auf FPGAs.

Dieses Projekt versteht sich als Fortsetzung des Vorgängerprojekts. „Gesten- und Objekterkennung durch schwache FPGAs in autonomen Fahrzeugen mittels neuronaler Netze“ [Alt+21]. Bereits im Vorgängerprojekt wurde ein Neuronales Netz trainiert und auf einem FPGA eines autonomen Fahrzeug installiert. Der Fokus lag auf dem Training eines Neuronalen Netzes mit eigenen Daten und dem Übertragen eines selbst trainierten Netzes auf das System. Außerdem wurde die Wiederverwendbarkeit der Ergebnisse in Vorbereitung auf folgende Projekte gesteigert. Das autonome Fahrzeug, welches bereits in einem weiteren studentischen Projekt [Mül+21] gebaut wurde, dient auch in diesem Projekt als Basis. Es verfügt über eine Kamera, welches für den im letzten Projekt definierten Anwendungsfall der Bilderkennung in Form von Personen bzw. Gesten essentiell ist.

Das Ziel des direkten Vorgängerprojektes zu diesem Thema war es "die Leistung des neuronalen Netzes unter den vorhandenen Hardware-Randbedingungen massiv zu steigern und damit entsprechend auch die Leistung des Vehikels beim autonomen Fahren zu verbessern. Projektergebnis soll ein besseres Verständnis für die vielen Optimierungsmöglichkeiten sein, sowohl auf Seiten der Digitaltechnik als auch auf Seiten der neuronalen Netze."[Alt+21, S. 8].

1.2 Projektziele

Geschrieben von Johannes Steffen

Unser selbst gesetztes Ziel dieses Semesters ist die Umsetzung von uns als relevant eingestufter Optimierungsmöglichkeiten, welche im letzten Semester bereits teilweise in Angriff genommen wurden. Unser gesetztes Hauptziel ist die Lauffähigkeit des autonom fahrenden Fahrzeugs, welches wir durch die Umsetzung einiger der Optimierungmöglichkeiten umsetzen wollen. Die Wissensaquise im Bereich der Neuronalen Netze und deren Optimierung für FPGAs soll aber auch in diesem Semester weiterhin voran gebracht werden und als Nebenziel des Projektes gelten. Um diese Aufgabe zu erreichen, soll als erstes Teilziel die Implementierung einer Personenerkennung erfolgen. Diese Personenerkennung und die damit zusammenhängende Ortung einer Person innerhalb des Kamerabildes soll mit einem FPGA Board beschleunigt werden und autonom auf dem Fahrzeug funktionieren. Das Fahrzeug soll, wenn es eine Person erkennt diese verfolgen und ansonsten nach einer Person suchen die verfolgt werden kann. In einem nachfolgenden Teilziel soll das Ergebnis der Personenerkennung, in Form von Bildausschnitten, an die Gestenerkennung, welche ebenfalls autonom auf dem Fahrzeug funktionieren soll, weitergeleitet werden und das Fahrzeug auf den erkannten Gesten basierend steuern.

Weiteres Projektziel *Geschrieben von Ferhat Cansu*

Zu Forschungsarbeiten gehört neben der Implementierung und Entwicklung eines Systems auch die Abgrenzung und der Vergleich mit ähnlichen bereits entstandenen Forschungsarbeiten. Deswegen besteht ein weiteres Ziel in diesem Semester darin zu verstehen, wie ein Neuronales Netz aufgebaut ist, um daraus ein Konzept für ein geeignetes Neuronales Netz für die Personenerkennung zu erstellen. Dazu werden die unterschiedlichen Schichten in einem Neuronalen Netz genauer betrachtet, um daraus schlussfolgern zu können wie ein geeignetes Neuronale Netz aufgebaut werden muss um Personen zu detektieren.

1.3 Methodik

Geschrieben von Johannes Steffen

Um die Erfüllung unserer Ziele verifizieren zu können benötigen wir Testszenarien, welche wiederholbar sind und welche die Funktion unseres Fahrzeuges nachweisen können. Diese können im Anhang F gefunden werden.
Das System hat auf Grund des Kamerabildes einen sehr großen Eingabevektor. Eine vollständige Überprüfung aller möglichen Eingabevektoren entfällt deshalb. Alternativ dazu soll eine auf Erfahrung basierte Einschätzung über die Lauffähigkeit des Fahrzeuges nach mehrmaliger Durchführung eines Testszenarios gegeben werden. Bei dieser Einschätzung sollen die Reaktionszeit des Fahrzeugs und die Performance der Neuronalen Netze mit eingebunden werden. Die in unserem Projekt unterschiedlich trainierten Neuronalen Netze sollen mittels Skripten zur Überprüfung der Genauigkeit verglichen und damit die Funktionsweise der Neuronalen Netze verifiziert werden. Dabei ist wegen mangelnder Zeit nur ein Vergleich innerhalb des Projektes vorgesehen und es wird von einem qualitativen Vergleich mit anderen Forschungsprojekten abgesehen.

Ergebnis des Projektes soll die Durchführung eines Systemtests sein, welcher dokumentiert und in Form eines Videos festgehalten werden soll. Eine auf Erfahrungen im Projektkontext basierte Einschätzung und Evaluation der in diesem Test erzielten Ergebnisse soll den Erfolg dieses Projektes einordnen.

2 Grundlagen

2.1 Neuronale Netze

Geschrieben von Philipp Altnickel

Neuronale Netze sind grundsätzlich eine Idee, die auf den Neuronen im Nervensystem von Lebewesen basiert. Die Neuronen können in diversen Formationen angeordnet werden und bilden eine Reihe von Verarbeitungsschritten von Informationen. Mit Hilfe von mathematischen Zusammenhängen können Ausgabewerte aus Eingabewerten ermittelt werden. Das besondere dabei ist, dass diese Zusammenhänge trainiert werden können, d.h. mit einer Reihe von bekannten Kombinationen aus Eingabe- und Ausgabewerten kann das Netz so verändert werden, dass es auf unbekannte Eingaben möglichst passende Ausgaben errechnen kann. [RW08]

2.1.1 Convolutional Neural Network (CNN)

Geschrieben von Mattia Uhlenbrock, Ferhat Cansu

Convolutional Neural Networks (CNNs) sind aufgrund der architekturellen Besonderheiten unter anderem besonders für die Objekterkennung in Bilder geeignet. Die Struktur eines klassischen Convolutional Neural Networks besteht aus einem oder mehreren Convolutional Layern, gefolgt von einem Pooling Layer, diese Kombination ist prinzipiell beliebig oft wiederholbar. Bei ausreichenden Wiederholungen spricht man dann von Deep Convolutional Neural Networks, welches zu einem Deep Learning Verfahren führt. Durch die Verwendung von Filtern in den Convolutional-Layern werden sogenannte Feature-Maps erzeugt, deren Größe durch Pooling-Layer für die Auswertung in weiteren Convolutional-Layern zum Erkennen komplexerer Muster oder zur Kategorisierung der Muster in den Fully-Connected-Layern reduziert werden.

2.1.2 Pooling

Geschrieben von Ferhat Cansu

Pooling wird verwendet, um überflüssige Informationen herauszufiltern, wie z.B. Position der Kanten. Es ist ein Diskretisierungsprozess, in dem die Eingabedarstellung abgetastet wird, um die Dimensionalität zu reduzieren. Dabei werden Annahmen über die in den Unterregionen erhaltenen Features erhalten. Es werden aber die zu erlernenden Parameter reduziert und somit auch der Rechenaufwand. Beim Pooling werden mathematische Funktionen verwendet, wodurch die Pixel zusammengefasst und nur die wichtigsten Informationen beibehalten werden. Es gibt unterschiedliche Arten von Pooling, die bekanntesten sind Max-Pooling und Average-Pooling. Beim Max-Pooling werden aus einem 2×2 Quadrat aus Neuronen des Convolutional Layers nur die Aktivität des aktivsten Neurons für die weiteren Berechnungsschritte beibehalten, siehe Abbildung 1.

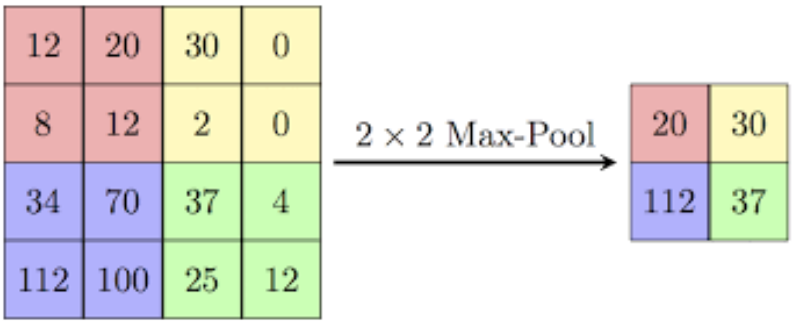


Abbildung 1: Berechnung Max-Pooling [Roo21]

Average-Pooling ist unter anderem eines der am meisten verwendeten Pooling Layer. Es wird ebenfalls für die Filterung von überflüssigen Informationen verwendet. Wie der Name schon sagt, wird beim Average-Pooling der Durchschnittswert für Patches einer Feature-Map berechnet, siehe Abbildung 2.

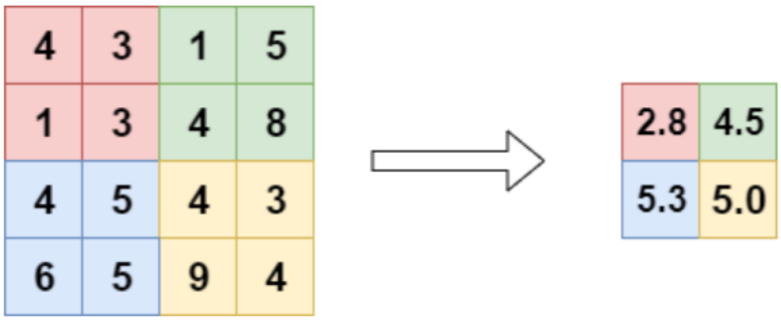


Abbildung 2: Berechnung Average-Pooling [Alt+21]

2.1.3 Convolutional Layer

Geschrieben von Ferhat Cansu

Im Convolutional Layer werden die Eingabedaten mit Hilfe von Filtern auf bestimmte Formen untersucht. Die Eingabe in den Convolutional Layer liegt als zwei- oder dreidimensionale Matrix vor z.B. ein Farbbild (RGB = 3-Dimensional). Die Aktivität jedes Neurons wird über eine diskrete Faltung berechnet, daher der Name Convolutional. Es wird schrittweise eine kleine Faltungsmatrix (Filter) über die Eingabe gelegt und somit wird die Eingabe eines Neurons im Convolutional Layer durch das Skalarprodukt der Faltungsmatrix berechnet. Dementsprechend reagieren benachbarte Neuronen im Convolutional Layer auf sich überlappende Bereiche, z.B. gleicher Hintergrund. Die Gewichte aller Neuronen eines Convolutional Layers sind identisch, jedes Neuron im ersten Convolutional Layer wird codiert, zu welcher Intensität eine Kante in einem bestimmten lokalen Bereich der Eingabe zum Beispiel bei einer Kantenerkennung vorliegt. Kantenerkennung als erster Schritt der Erkennung besitzt hohe biologische Plausibilität, dass bedeutet das auch unser Gehirn zunächst das was wir sehen nach Kanten filtert. Der mittels Faltung ermittelte Input eines jeden Neurons wird anschließend von einer Aktivierungsfunktion in den Output des Convolutional Layers gesetzt. Je tiefer ein Convolutional Layer verwendet wird, desto komplexer die zu erkennenden Features z.B. Teile eines Gesichts.

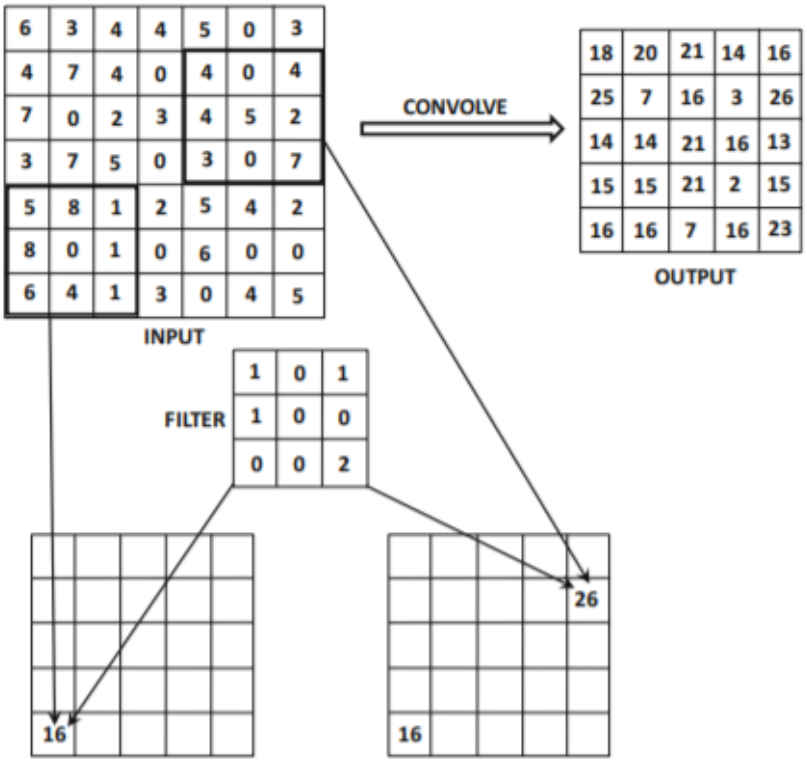


Abbildung 3: Berechnung des Convolutional Layers [Roo21]

$$5 * 1 + 8 * 1 + 1 * 1 + 1 * 2 = 16 \tag{1}$$

$$4 * 1 + 4 * 1 + 4 * 1 + 7 * 2 = 26 \tag{2}$$

Durch das Training werden Filterformen selbst erlernt. Es besteht auch die Möglichkeit eigene Filter zu konstruieren, jedoch wird hiervon abgeraten. Für eine Personen Erkennung werden viele unterschiedliche Filter in 32 x 32 oder 64 x 64 verwendet. Ein Beispiel für unterschiedliche Filtertypen stellt die Abbildung 4 Filtertypen sehr gut dar.

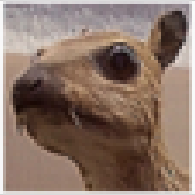
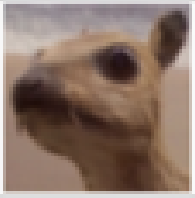
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Abbildung 4: Filtertypen Convolutional Layer [Aus18]

2.1.4 Aktivierungsfunktionen

Geschrieben von Ferhat Cansu

Aktivierungsfunktionen stellen den Zusammenhang zwischen dem Input und der Aktivierung eines Neurons dar. Es existieren viele unterschiedliche Aktivierungsfunktionen. Die inder Praxis am meisten verwendeten sind Lineare, Lineare mit Schwelle, Binäre und Sigmoide Aktivierungsfunktionen. Aktivierungsfunktionen sind Mathematische Schwellwertfunktionen, welche ab einer bestimmten Schwelle einen High- oder Low-Pegel ausgeben, siehe Abbildung 5. Das Aktivitätslevel wird durch eine sogenannte Ausgabefunktion in den Output transformiert. Häufig wird als Ausgabefunktion die Identitätsfunktion verwendet.

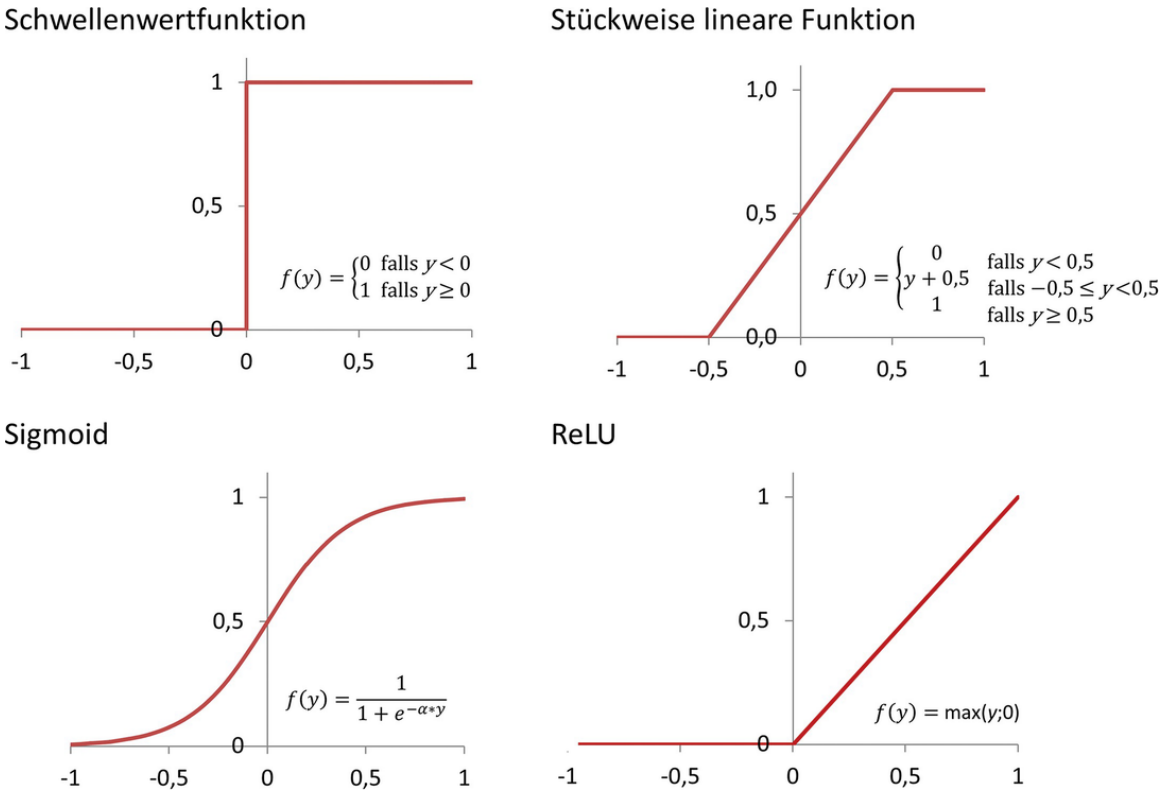


Abbildung 5: Aktivierungsfunktionen [Aus18]

2.1.5 Fully Connected Layer

Geschrieben von Ferhat Cansu

Fully Conected Layer sind meistens die letzten Layer eines Netzes. Sie klassifizieren die Ergebnisse. Die Eingabe ist der letzte Polling- oder Convolutional-Layer. Dazu werden die erkannten Features in einer eindimensionalen Menge zusammengeführt. Jedes Feature ist ein Eingangsneuron und jedes zu erkennende Objekt ein Ausgangsneuron. Dazwischen befinden sich Hiddenlayer die nach außen nicht sichtbar sind.[Wu16]

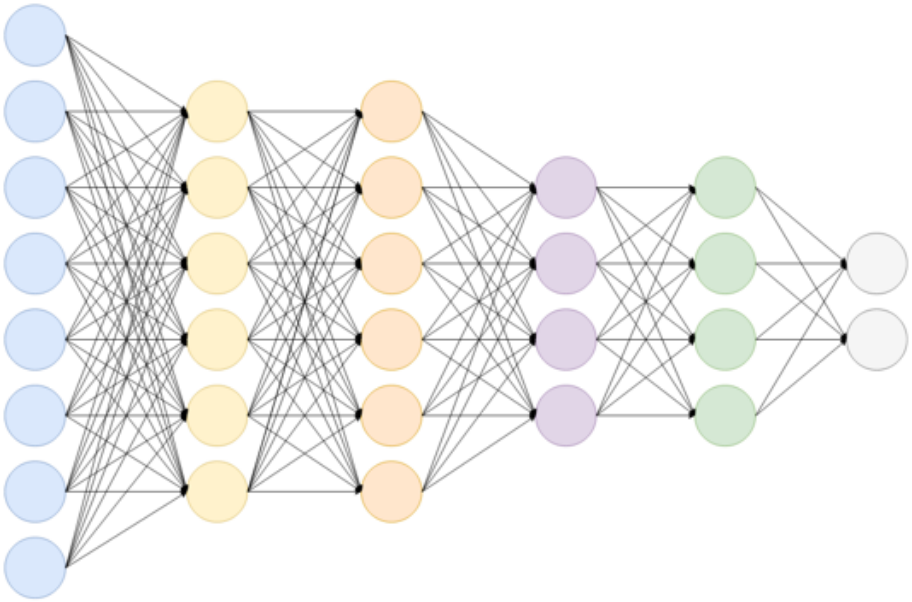


Abbildung 6: Fully Connected Layer [Wu16]

2.1.6 Dropout Layer

Geschrieben von Ferhat Cansu

Der sogenannte Dropout Layer ist eine effektive Methode um Overfitting zu vermeiden, indem Neuronen während dem Trainingsprozess kurzzeitig ausgeschaltet werden. Somit werden deren Ergebnisse in den nachfolgenden Berechnungen nicht berücksichtigt. Dies führt dazu, dass das NN dazu gezwungen wird Muster in den Eingabedaten zu erkennen und damit zu verallgemeinern.[Wu16]

2.1.7 Overfitting

Geschrieben von Ferhat Cansu

Bei dem trainieren eines Netzes kann Overfitting entstehen. Unter Overfitting versteht man, dass das System dazu neigt die Trainingsdaten auswendig zu lernen. Bei dem trainieren von Netzen versucht man die Funktion zwischen den Input- und Outputlayern zu optimieren. Durch das auswendig lernen erzielt man zwar sehr hohe Genauigkeiten bei den Trainingsdaten, jedoch fällt die Genauigkeit bei den Validierungsdaten sehr gering aus. Der richtige Zeitpunkt zum beenden des Trainings ist dann , wenn die Loss-Werte wieder zunehmen, die Abbildung 7 verdeutlicht den zusammenhang .[Alt+21]

Abbildung 7: Training Iteration
[Boh21]

2.2 Copy number variation (CNV)

Geschrieben von Ferhat Cansu

Copy number variation (CNV) ist ein Derivat der VGG-16- und BinaryNet Topologie. Die VGG16-Topologie wird für tiefe Lernbildklassifizierungen verwendet. Die CNV-Variante ist trainiert auf dem CIFAR-10-Datensatz, der 32x32 RGB-Bilder klassifiziert. Dieses Netzwerk enthält eine Abfolge von 3x3 Faltung, 3x3 Faltung, 2x2 Maxpool Schichten, die dreimal wiederholt werden mit 64-128-256 Kanälen. Die Eingabe wird zunächst mit 64 unterschiedlichen 3 x 3 großen Filtern auf verschiedene Formen untersucht. Dieser Prozess wird wiederholt und anschließend eine 2 x 2 Maxpooling-Layer angewendet, um unwichtige Informationen zu verwerfen. Dieser Prozess wird zweimal wiederholt. Bei mehreren Wiederholungen spricht man von einem Deep Learning verfahren. Bei der letzten Wiederholung werden nach den Convolutional Layern, wenn das Bild seine kleinste Matrix erreicht hat, Fully Connected Layer mit 512 Neuronen verwendet, um die gefilterten Informationen einem Ausgabeneuron zuzuschreiben.

CNV akzeptiert Bilder in der Größe 32x32 mit 24 Bit/Pixel und gibt eine 10-Element-Vektor von 16-Bit-Werten als Ergebnis aus. CNV bietet 3 Möglichkeiten für die Erkennung bzw. Gewichtung und der Aktivierung. In der CNVw1a1-Variante, welche in diesem Projekt verwendet wird, wird die Gewichtung und die Aktivierungen auf bipolare Werte, entweder -1 oder +1 quantisiert. Die Ausnahme liegt im Eingang, welche einen RGB mit 8 Bit pro Kanal besitzt. Des Weiteren gibt es die CNVw1a2- und CNVw2a2-Variante. Wie w1a2 Variante besitzt die bipolare Gewichtung und die 2 Bit Aktivierung. Bei der w2a2 Variante werden 2 Bit für die Gewichtung und die Aktivierung verwendet.[Roo21]

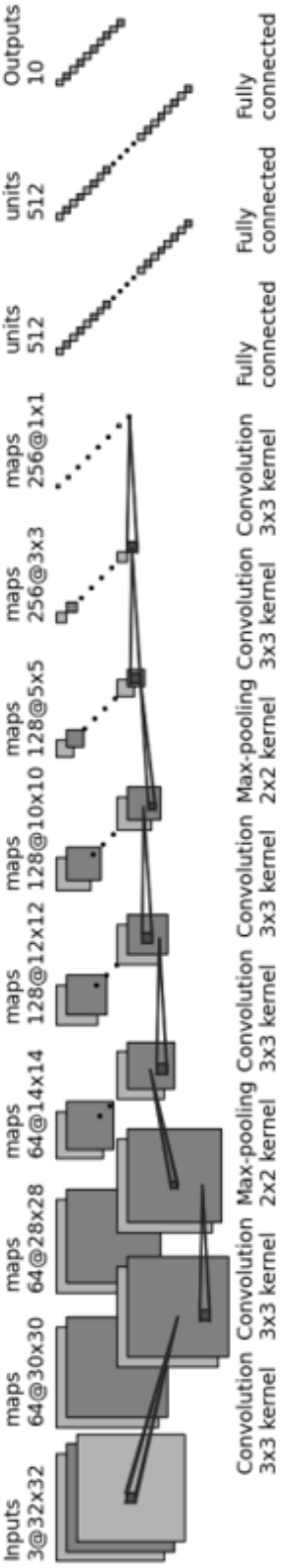


Abbildung 8: Aufbau CNV [Roo21]

2.2.1 You Only Look Once (YOLO)

Geschrieben von Philipp Altnickel

YOLO ist ein Modell auf Basis von CNNs, welches auf die Echtzeiterkennung von Objekten in Bildern ausgelegt ist. YOLO wurde seit seiner ersten Version V1 aus dem Jahr 2016 in verschiedenen Versionen weiterentwickelt. Im Gegensatz zu vielen anderen Ansätzen zur Objekterkennung, kann YOLO ganze Eingangsbilder in höheren Auflösungen verarbeiten. Das Eingangsbild wird in ein gleichmäßiges Gitterfeld zerteilt und die Wahrscheinlichkeit jedes Feldes ermittelt. Das Besondere ist, dass sowohl Bounding Boxes von Objekten, als auch deren Klassenwahrscheinlichkeiten anhand der Wahrscheinlichkeiten dieser einzelnen Felder ermittelt können. [Red16]

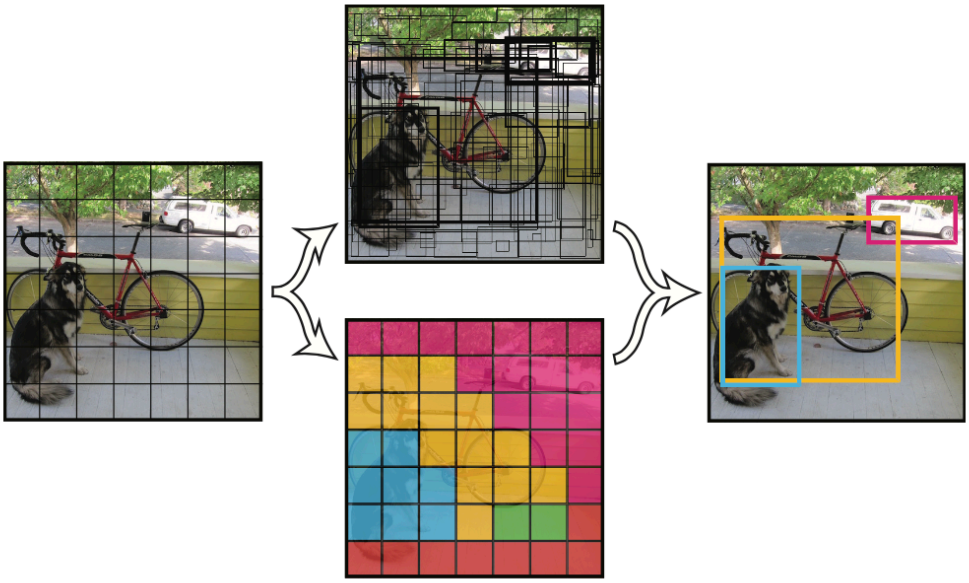


Abbildung 9: Die Verarbeitungsschritte von YOLO [Red16]

YOLO V1 wurde darauf ausgelegt, die 20 Klassen des PASCAL VOC 2012 [EGW07] Datensatzes zu erkennen. Die Architektur besteht aus 24 Convolutional Layers, um die Features zu erkennen. Anschließend folgen 2 Fully Connected Layers zur Erkennung der Wahrscheinlichkeiten und Koordinaten. Das Netz wird mit Bildern der Größe 224x224 Pixel trainiert, die Erkennung passiert dann anhand von doppelt so großen Bildern (448x448 Pixel). [Red+16]

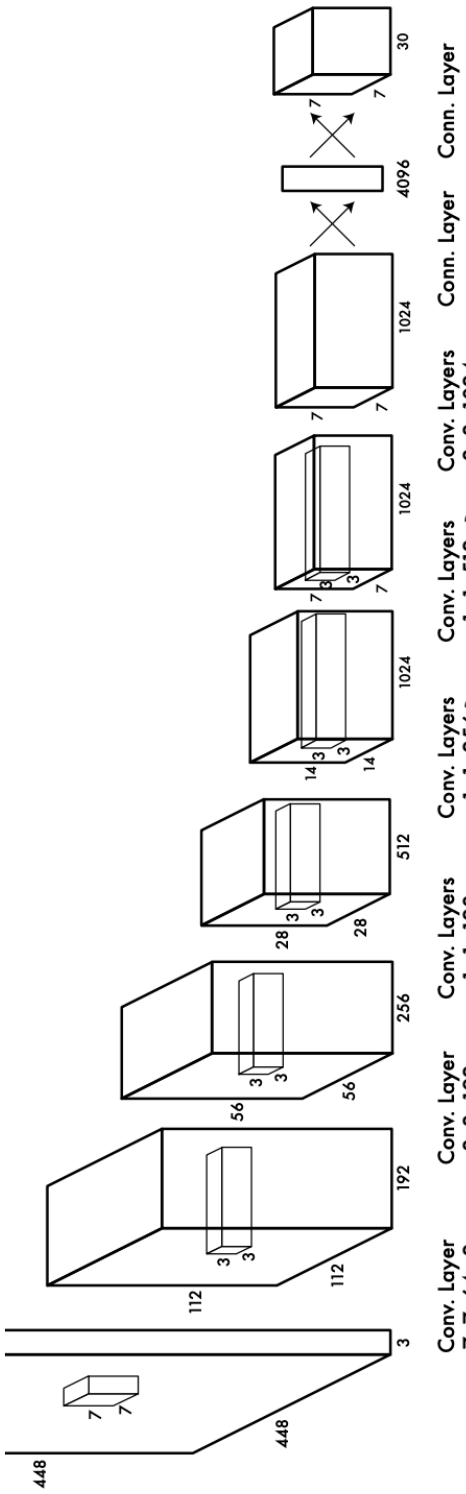


Abbildung 10: Aufbau YOLO V1 [Red+16]

YOLO auf schwacher Hardware *Geschrieben von Johannes Steffen*

Es gibt viele Projekte mit dem YOLO Ansatz: YOLO V1, V2 und V3 basieren alle auf dem darknet, einem Open Source Deep Neural Network, entwickelt von den Erfindern des YOLO Ansatzes, Joseph Redmon und weiteren Beteiligten [Red14]. Bei der ersten Veröffentlichung von YOLO V1, wurde bereits ein kleineres Modell namens „Fast YOLO“ entwickelt und vorgestellt. Anstatt 24 Convolutional Layer verwendet dieses System nur 9 Convolutional Layers [Red+16, S. 2]. Dementsprechend ist das System schneller aber weniger akkurat. Ziel eines kleinen Modells ist die Ausführung auf schwacher und meist autonomer, also batteriebetriebener Hardware.

In der Open Source Community gibt es weitere interessante Entwicklungsarbeiten im Bereich YOLO: YOLO V4 [Ale16] und YOLO V5 [Gle21] oder YOLO X [Ge+21] sind nur einige von zahlreichen Implementierungen und Anpassungen des YOLO Konzepts. Auch diese besitzen kleinere Implementierungen zur Ausführung auf schwacher Hardware und besitzen deswegen weniger Layer.

Wie schon die älteste YOLO Implementierung V1 bieten viele der bereits genannten Implementierungen kleinere Versionen, welche auf schwächerer Hardware und mit weniger Leistung laufen können. Bei YOLO V3 heißt diese Implementierung Tiny YOLO:

Model	Train	Test	mAP	FLOPS	FPS
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220

Tabelle 1: Vergleich von verschiedenen YOLO V3 Versionen [RF18]

Wie in Tabelle 1 zu sehen ist hat Tiny YOLO eine geringere Genauigkeit (mean Average Precision; mAP) dafür aber einen im Vergleich viel höheren Umsatz an Eingangsbilder (Frames Per Second; FPS) bei der Objekterkennung. Die anderen in der Tabelle verglichenen Netzwerke besitzen verschieden große Eingangsbilder. Die Größen sind bei den verglichenen Netzwerken von oben nach unten 320x320p, 416p und 608p. Es ist im allgemeinen Fall anzunehmen, dass ein größeres Eingabebild eine bessere Genauigkeit bietet aber dafür auch eine höhere Ausführungszeit und Speicherplatz (FLOPS) benötigt.

2.2.2 Single Shot Detector (SSD)

Geschrieben von Mattia Uhlenbrock

Der Single Shot Multibox Detector (SSD) ist eine Architektur die eine einstufige Objekterkennung in Bildern erlaubt. Dieser Ansatz unterscheidet sich konzeptionell zu zweistufigen Ansätzen (bspw. Fast R-CNN), in denen zunächst sogenannte Regions of Interest (RoI) gesucht werden und in einem zweiten Schritt die RoIs durch ein CNN klassifiziert werden. Der SSD kommt dabei ohne das bestimmen von Regional Proposals und die daraus folgende Aufbereitung des Regional Proposals für die anschließende Klassifizierung aus. [Liu+15]

Die Architektur des SSDs nutzt ein beliebiges Netz für Bildklassifizieren, genannt Base Network oder Backbone. Ursprünglich wurde das VGG-16 als Base Network verwendet. Aktuelle Implementierungen verwenden allerdings auch andere Netze, wie bspw. VGG-19, Resnet, Inception und weitere [SV20]. Das Base Network erzeugt dabei die üblichen Feature-Maps.

Gegenüber anderen einstufigen Detektoren (z.B. YOLO) nutzt der SSD mehrere Feature-Maps unterschiedlicher Skalierung. Dazu werden zusätzliche Convolutional-Layer, die kontinuierlich kleiner skaliert werden, an den Backbone angehängt, wie in Abb. 11 dargestellt wird. Durch die zusätzlichen Convolutional-Layer wird eine definierte Anzahl (abhängig von der Dimensionierung der Filter) an Erken-

nungsergebnisse pro Layer erzeugt. Ein Ergebnis kann entweder eine Punktzahl für eine Kategorie oder einen Offset zu einer „default box“ sein. Für jeden Eintrag in der Feature-Map gibt es eine korrespondierende default box. Die default boxes sind die resultierenden Zellen eines abstrakten Gitters, welches über das Ausgangsbild gelegt wird. [Liu+15]

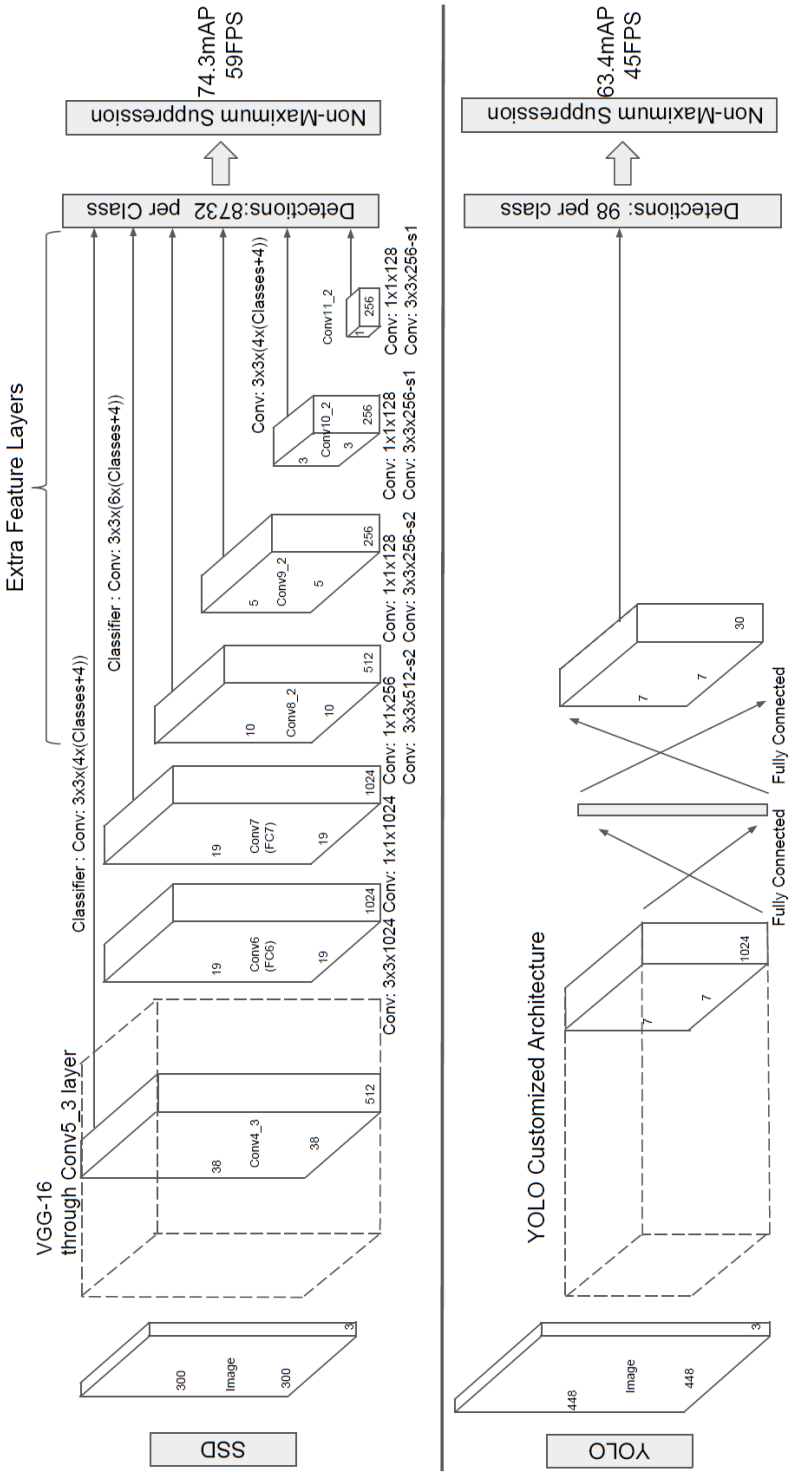


Abbildung 11: Vergleich des Aufbaus von SSD und YOLO V1 [Liu+15]

2.3 Verfügbare Datensätze

Geschrieben von Ferhat Cansu

Bei der Auswahl der öffentlich verfügbaren Datensätzen müssen bestimmte Punkte berücksichtigt werden. Bei dem Datensatz muss es sich um Bilder von Personen handeln, welche deutlich und aufrecht abgebildet sind. Ideal wäre es, wenn die abgebildeten Personen von unten nach oben aufgenommen wären, weil die Kamera auf dem autonomen Fahrzeug sich auf einer Höhe von circa 15cm befindet und eine aufrechtstehende Person erkennen soll. Außerdem werden die Daten der Kamera im RGB Format übermittelt, was dazu folgt, dass sich ein Datensatz in Graustufen nicht eignet.

Öffentliche Datensätze:

AT&T Ein Datensatz ist die *„Database of Faces“* der AT&T Laboratories Cambridge. Die Datenbank beinhaltet Bilder von 40 Personen, welche innerhalb von 2 Jahren in jeweils 10 Terminen aufgenommen wurden. In diesem Datensatz sind die Gesichter der Personen immer frontal zur Kamera. Außerdem liegen die Bilder in Graustufen mit einer große von 92x112 Pixel vor, wodurch sie sich für unsere Anwendung im RGB bereich nicht eignet. [Wir21]

Max Planck Institut Informatik Human Pose Dataset Ein weiterer Datensatz ist der sogenannte MPii Human Pose Dataset. In diesem sind Personen abgebildet, welche eine Sportart ausüben, indem sie nicht eindeutig zu erkennen sind, wodurch es eher dazu verwendet wird, Sportarten zu erkennen als Personen. Deshalb ist dieser Datensatz für unseren Anwendungsfall nicht geeignet.[MPi21]

COCO Bei dem Datensatz COCO handelt es sich um einen Datensatz mit 330K Bildern mit 91 Objektklassen und 1,5 Millionen Objektinstanzen, welcher zur Erkennung von Personen, Tieren und Objekten dient. Außerdem sind in den Daten die Bounding Boxen mit integriert, welche erlauben, Objekte aus den Bildern herauszuschneiden und somit hervorzuheben. [Lin+15]

2.4 Tiling

Geschrieben von Philipp Altnickel

Sollen hochauflösende Bilder mittels eines neuronalen Netzes analysiert werden, gibt es oft die Problematik, dass aufgrund von z.B. begrenzten Hardware Ressourcen, nur Bilder von relativ geringer Auflösung verarbeitet werden können. Solche Bilder einfach herunter zu skalieren führt oft dazu, dass diese unbrauchbar zur Analyse werden, da die interessanten Details meist nicht Bild ausfüllend und damit nach der Skalierung nicht mehr erkennbar sind. Das Konzept des „Kachelns“ (auf Englisch im folgenden Dokument auch „Tiling“ genannt) verfolgt dabei die Idee, das Bild in kleine Teile zu zerschneiden, um so den interessanten Bereich möglichst Ausschnitt füllend einzufangen. Um dies allerdings zu erzielen, muss das ganze Bild in überlappende Kacheln zerschnitten werden, indem ein sogenanntes „Sliding Window“ in regelmäßigen Abständen verschoben wird. Die Auslastung des Neuronalen Netzes wird so erheblich steigen, da je nach Größe des interessanten Objekts sehr viele, unterschiedlich große Ausschnitte erzeugt werden müssen, allerdings ist die Chance sehr hoch, dass das Netz in einem Tile das gesuchte Objekt mit sehr hoher Wahrscheinlichkeit erkennt. Außerdem lässt sich so eine Lokalisierung des Objekts erzielen, da die Position des erkannten Tiles bekannt ist. [Mül21]

Weitere Details zu dieser Thematik lassen sich der Bachelorarbeit von Felix Müller [Mül21] entnehmen, die sich umfassend mit dem Thema Tiling auf schwacher Hardware auseinandersetzt.

2.5 Entwicklungsumgebung

2.5.1 PyTorch

Geschrieben von Ismail Sastim

PyTorch ist ein Framework und besitzt zwei Anwendungszwecke. Zum einen kann es lediglich für effiziente Tensorberechnungen mit GPU-Beschleunigung genutzt werden ähnlich zur bekannten numpy Bibliothek. Zum anderen kann es auch gleich für die Entwicklung und Inferenz Neuronaler Netze verwendet werden als Alternative zu beispielsweise TensorFlow. [Pas+19]

2.5.2 Brevitas

Geschrieben von Mattia Uhlenbrock

Brevitas ist eine Software Bibliothek für PyTorch, um quantisierte Neuronale Netze für die Verwendung im FINN Framework 2.5.3 erstellen zu können. Dabei bietet Brevitas spezielle Netzwerk-Layer für die Definition eines Modells und die Möglichkeit ein trainiertes Netz in einem für das FINN Framework spezifische ONNX-Format zu exportieren. Für das Training hingegen werden die gängigen PyTorch Funktionen verwendet. [Alt+21, S. 39]

2.5.3 FINN Framework und HLS

Geschrieben von Ismail Sastim

FINN ist ein experimentelles Framework einer Forschungsgruppe von Xilinx zur Übersetzung Neuronaler Netze in FPGA Hardware der PYNQ Familie.

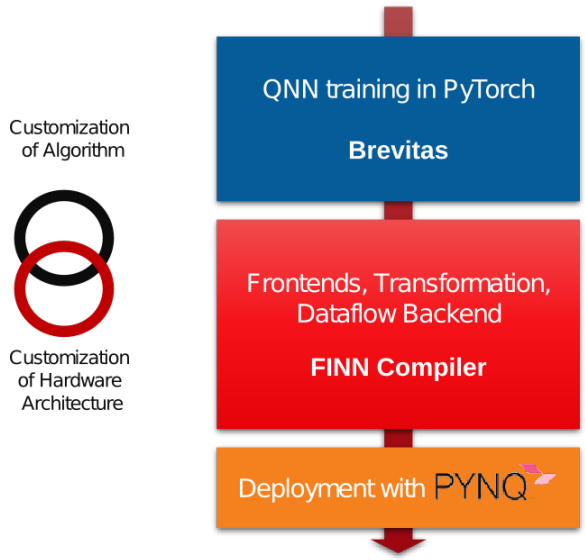


Abbildung 12: FINN Stack

Die in Kap. 2.5.2 erwähnte Bibliothek Brevitas ist Teil des FINN Frameworks. Die Ausgabe des mit Brevitas und PyTorch trainierten Netzes ist im „Open Neural Network Exchange"-Format (ONNX). Darin sind die einzelnen quantisierten Layer enthalten, welche durch vorgefertigte C++-Funktionen ersetzt werden, um sie anschließend mit Vivado High-Level Sythesis (HLS) in ein Bistream für das FPGA Board umgewandeln zu können. [Alt+21, S. 40 f.]

3 Konzeption des Gesamtsystems

3.1 Aktueller Stand als Grundlage der Weiterentwicklung

Geschrieben von Mattia Uhlenbrock

Im Vorgängerprojekt wurde ein Neuronales Netz zur Personen- und Gestenerkennung auf dem autonomen Fahrzeug betrieben. Das Fahrzeug sollte anhand ausgewählter Signalgesten die Verfolgung einer Person starten oder stoppen. Dazu wurde ein Datensatz mit den zu erkennenden Gesten angefertigt. Dieser besteht aus Bildern auf denen die gezeigte Geste isoliert (zentral im Bild und mit wenig Kontext) dargestellt sind. Mit diesem Datensatz wurde ein Netz trainiert, mit dem versucht wurde zwei Probleme gleichzeitig zu lösen, zum einen die Personen- und zum anderen die Gestenerkennung. Vermutlich führte unter anderem dieser Versuch und die Spezialisierung des Datensatzes auf Gesten zu einer schlechten Erkennungsrate des Neuronalen Netzes im Gesamtsystem.

Für das aktuelle Projekt wird das Verfolgungsszenario mit dem autonomen Fahrzeug beibehalten. Die Hardware des Fahrzeugs bleibt unverändert, während der Fokus auf der Weiterentwicklung der Software-Komponenten liegt. Das zuvor trainierte Netz eignet sich nicht, um die Personen- und Gestenerkennung in einem Schritt durchzuführen [Alt+21, S. 88]. Deshalb werden in diesem Projekt weitere Ansätze evaluiert. In Betracht kommen dabei zum einen der Ansatz ein einzelnes aber besser geeignetes Netz zu finden und zum Anderen die Gesten- und Personenerkennung in zwei speziell für die jeweilige Aufgabe trainierte Netze zu trennen.

3.2 Auswahl des Neuronalen Netzes

Geschrieben von Johannes Steffen

Es gibt viele Bilderkennungsalgorithmen zur Klassifizierung von Objekten. Bereits im letzten Semester wurden einige davon betrachtet [Alt+21]. YOLO war dabei ein betrachteter Ansatz und überzeugte durch seine relativ hohe Genauigkeit aber vor allem Schnelligkeit im Vergleich zu andern Neuronalen Netzen, welcher in diesem Paper [Red+16, S. 6] gefunden werden kann. Single Shot Detektoren (siehe 2.2.2) sind eine Alternative zu YOLO. Sie besitzen eine andere Architektur um eine Objekterkennung durchzuführen. Die Erkennung und Klassifizierung nur eines einzigen Typens ist für unser erstes Teilziel, der Personenerkennung, notwendig. Es gibt Neuronale Netze die sich auf solche Aufgaben spezialisieren. Neben diesen drei möglichen Konzepten soll in diesem Kapitel auch das in Vorgänger Projekten verwendete Netz (CNV) betrachtet und die Nutzbarkeit aller Netze in diesem Projektkontext betrachtet werden. Da der zeitliche Aufwand um sich mit mehr Konzepten zu beschäftigen dem Projektziel nicht dienlich ist, wurden nur diese vier Konzepte genauer betrachtet. Andere abgelehnte Ideen sind zum Beispiel im letzten Projektbericht zu finden [Alt+21]. Dazu gehören R-CNN, Fast R-CNN oder gar eine Eigenentwicklung eines binarisierten Neuronalen Netzwerkes nach eigenem Konzept. Der Aufwand von vor allem letzterem ist von unserer Projektgruppe aber nicht tragbar und wurde deswegen verworfen.

3.2.1 YOLO

Geschrieben von Johannes Steffen, Ismail Sastim

YOLO konnte nach ersten Tests auf einem x86 Prozessor sehr schnell gute Ergebnisse und eine präzise Erkennung von Personen aufweisen. Nach so einem Algorithmus wird in diesem Projekt gesucht. Es gibt eine gute Dokumentation für die Programmierung mit zum Beispiel OpenCV in der Programmiersprache Python. Code und Anwendungsbeispiele wie [Ros18] gibt es zahlreich im Netz und konnten schnell auf den Laborrechnern reproduziert werden. Eine erneute Analyse über den Gebrauch von nach dem YOLO Ansatz entwickelte Neuronale Netze (Details siehe Kapitel 2.2.1) erschien der Projektgruppe deshalb sinnvoll.

YOLO auf FPGA YOLO ist ein sehr komplexes System, welches in darknet, einem Deep Neuronal Network implementiert wurde. Darknet ist in C++ geschrieben. Eine Implementierung von YOLO auf einem FPGA ist schwierig, da ein FPGA ganz anders funktioniert als zum Beispiel ein x86 oder ARM Prozessor. Ein FPGA kann zwar einen Prozessor simulieren aber um C Code auf einem FPGA auszuführen muss das Programm komplett auf das vorhandene FPGA Framework designed und neu implementiert werden. Trotz dieses hohen Aufwands haben bereits Forschungsgruppen den YOLO Ansatz auf ein FPGA gebracht [YB20, S. 343]. Dafür wurde YOLO von Grund auf neu entwickelt und in HLS überführt um auf schwachen FPGAs mit eingeschränkten Ressourcen zu funktionieren.

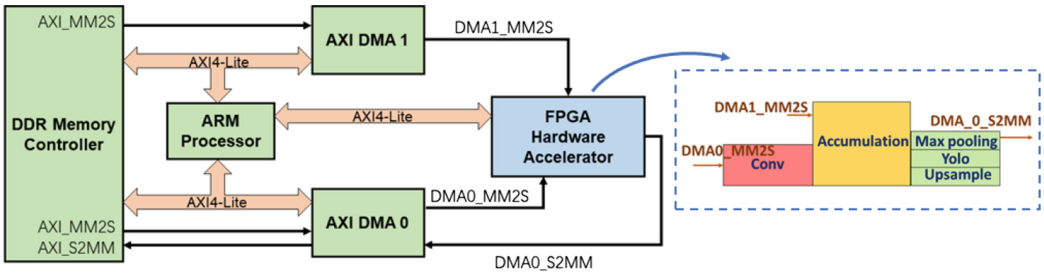


Abbildung 13: System Architektur eines tinyYolo auf FPGA Ansatzes [YB20, S. 335]

In Abbildung 13 ist ein solcher Entwurf zum Beispiel zu erkennen. Er ähnelt der im letzten Projekt verwendeten Architektur, dahin gehend, dass Daten mittels des AXI an das FPGA übertragen werden können. Mehr Informationen zum AXI4 Bus kann im letzten Projektbericht gefunden werden [Alt+21, 42f]. Leider verwendete keine bei der Recherche gefundene Forschungsgruppe das von uns verwendete Board und auch der Source Code von vielen Forschungsteams konnte nicht für unseren Einsatzbereich verwendet werden.
Die eigene Implementierung des YOLO Ansatzes für das von uns verwendete Pynq Z1 Board ist unserer Einschätzung nach die einzige Möglichkeit den YOLO Ansatz in diesem gegebenen Projektumfeld umzusetzen. Diese Möglichkeit wurde als zu schwierig und risikoreich eingeschätzt und deswegen nicht weiter verfolgt.
Ein weiterer Ansatz wäre die Verwendung von OpenCV und Vivado HLS. OpenCV besitzt Module um vortrainierte Models zu laden und Bilddaten zu inferieren. In [Mül21] wurde OpenCV verwendet, um Bilder auf dem FPGA des PYNQ-Board zu skalieren. Erste Beispiele mit OpenCV und tiny-Yolo-v3 wurden erfolgreich auf einem Windowssystem getestet. Jedoch fielen zwei Problematiken auf. Zum einen war das verhältnismäßig kleine Model immer noch zu groß für das FPGA mit 34MB. Zum anderen braucht OpenCV Dateisystemzugriff, der auf einem FPGA nicht existiert, um die Modeldatei einzulesen. Daher wurde der Ansatz ebenfalls nicht weiter verfolgt.

3.2.2 SSD

Geschrieben von Mattia Uhlenbrock

Da es keine mit Brevitas für das FINN-Framework umgesetzte Implementierung des SSDs gibt, müsste für die Verwendung des SSDs dies zunächst umgesetzt werden. Dazu wäre es zwingend notwendig, bisher in diesem und in den Vorgängerprojekten nicht verwendete, Trainingsmethoden zu implementiert und ein Übersetzungsskript für das SSD-Modell zu erzeugen.

Es ist nicht gelungen ein beliebiges in Brevitas erstelltes trainiertes Netz mit einem der beiden Übersetzungsworkflows des FINN-Frameworks in Hardware zu übersetzten. Aus zeitlichen Gründen und wegen der schlechten Dokumentation des Frameworks wurde keine weitere Arbeit für die Implementierung des SSDs aufgewendet.

3.2.3 One-Class Convolutional Neural Network

Geschrieben von Mattia Uhlenbrock

CNN-Modelle die in ihrer Architektur und durch geeignete Trainingsmethoden auf das Erkennen von Objekten einer einzelnen Klasse optimiert sind, eignen sich potentiell auch für das Erkennen von Objekten der Klasse „Mensch“ oder „Person“ in Bildern. Dieser Ansatz wurde aufgrund der knappen Zeit für die nötige Einarbeitung und Durchführung des Projekts nicht weiter verfolgt.

3.2.4 CNV mit zusätzlichen Tiling

Geschrieben von Ferhat Cansu

CNV ist eines der bekanntesten Neuronale Netze die zur Erkennung von Objekten verwendet werden. Außerdem bieten CNVs aufgrund ihrer Zusammensetzung mit der VGG-16 Topologie eine hervorragende Eigenschaft zur Erkennung von Objekten, da die VGG-16 Topologie für tiefe Lernbildklassifizierungen verwendet wird. Außerdem eignet sich CNV sehr gut für unsere Anwendung, weil die am Fahrzeug montierte Kamera Bilder im RGB Bereich aufnimmt und weitergibt. Desweiteren ist ein funktionierendes Übersetzungsskript für dieses Netz verfügbar.

Deswegen haben wir uns entschieden wie im Vorgängerprojekt das CNV zu verwenden, um das Verfolgungsszenario unseres autonomen Roboters umzusetzen.

3.3 Auftrennung von Personen- und Gestenerkennung

Geschrieben von Johannes Steffen

Nach gemeinsamer Diskussion sind die Trennung der Gesten- und Personenerkennung in zwei verschiedene Systeme und die Umsetzung dieser beiden Systeme auf FPGA Hardware unserer Einschätzung nach die relevantesten Themen für dieses Projekt, weshalb wir uns dieses Semester auf diese fokussieren wollen.
Die Trennung der Personen- und Gestenerkennung wurde bereits im Ausblick vom letzten Projektbericht grob beschrieben: "Hierzu könnte für die Klassifizierung, ob es sich um eine Person handelt, ein CNN verwendet werden. Im Anschluss wird dieses CNN an das vorhandene Netz [...] verknüpft"[Alt+21, S. 89]. Dieses Konzept wollen wir verfeinern um damit unser Verfolgungszenario und damit die Lauffähigkeit des Roboters erfolgreich umzusetzen.
Die Idee ist es den Bereich, in dem eine Person vorhanden ist mittels des ersten Netzes zu bestimmen und den zu generierenden Bildausschnitt auf dem die Person zu erkennen ist an das bereits bestehende System der Gestenerkennung aus dem letzten Semester zu senden. Es sollen nur relevante Bereiche des Kamerabildes durch die Gestenerkennung überprüft werden. Zu bestimmen welche Bereiche relevant sind ist Aufgabe des Neuronalen Netz zur Personenerkennung.

3.4 Datenfluss auf dem autonomen Fahrzeug

Geschrieben von Philipp Altnickel

Um das Konzept für dieses Semester genauer zu veranschaulichen, wurde ein Datenfluss entwickelt der die Verarbeitung der Daten auf dem Gesamtsystem durch beide Neuronale Netze beschreibt. Dieser basiert auf dem im Vorgängerprojekt verwendeten System. Zur bisherigen Verarbeitung der Daten sollen nun allerdings diverse Änderungen eingearbeitet werden, die die Erkennung verbessern und schlussendlich zu einer akzeptablen Funktion des Systems führen sollen. Die alte Verarbeitung lässt sich anhand des Projektberichtes des vorherigen Semesters [Alt+21] nachvollziehen. Grundsätzlich gab es dort verschiedene Controller, die als eine Art Kette (mit Zwischenverzweigungen) gearbeitet haben, um das Kamerabild schlussendlich in Bewegungen des Fahrzeugs zu übersetzen.

Die Ziele für dieses Projekt lassen sich in der Konzeptgrafik (Abb. 14) wiederfinden. Der obere Fluss zeigt die Verarbeitung, die im ersten Schritt implementiert werden soll. Darunter ist eine Erweiterung darge-

stellt, die später ohne allzu große Änderungen hinzugefügt werden kann. Der Beginn des Flusses, d.h. das Einlesen des Kamerabildes und das Tiling kann im wesentlichen wie gehabt beibehalten werden. Lediglich die Größe des Ausgangsbildes, sowie die Art und Größe der Tiles müssen rekonfiguriert werden. Bisher war es nötig sehr viele, große und auch in verschiedenen Stufen kleiner werdende Tiles zu erzeugen, um eine Person möglichst Tile-füllend zu treffen. Die Idee, nun ein Neuronales Netz zur Personenerkennung zu verwenden basiert darauf, dass Personen nun kleiner in einem Tile sein können um trotzdem als Person erkannt zu werden. Es können also weniger, gröbere Tiles erzeugt werden. Bisher wurde mit Anzahlen in der Größenordung 1000 Tiles genutzt, die Hoffnung ist nun, wesentlich weniger, möglicherweise weit unter 100 Tiles nutzen zu können. So könnten mehr als die bisherigen 1-2 Bilder pro Sekunde verarbeitet werden, bzw. Kapazität für ein weiteres Neuronales Netz geschaffen werden. Der *InferenceController* muss vermutlich auch nicht umfassend verändert werden. Das Übergeben und Abholen der Daten an und von dem Neuronalen Netz auf dem FPGA sollte kaum unterschiedlich sein. Im *InferenceResultController* wird es lediglich darauf ankommen, die Position der Person aus den Confidences der groben Tiles zu ermittelt. Zuletzt kann die hier ermittelte Position an die Fahrzeugsteuerung übergeben werden. Eine Position der Person im Bild wurde hier bereits ermittelt, verschiedene Zustände, welche durch Gesten gewechselt werden sollten, können hier ausgelassen werden, indem immer im Fahrzustand verweilt wird.

Im zweiten Schritt des Projekts ist es dann möglich, den grob funktionierenden Stand zu erweitern. Der *InferenceResultController* kann so angepasst werden, dass er aus den Confidences des ersten Netzes eine Region of Interest bestimmt und zusammen mit den Eingangsbildern der Kamera in einem weiteren *TilingController*, weniger aber höherauflösende Tiles des interessanten Bereichs erstellt. Diese können dann mit Hilfe eines zweiten *InterfenceControllers* an das Neuronale Netz zur Personenerkennung weitergeleitet werden. Sollte noch Platz auf dem FPGA sein, kann dies weiterhin dort betrieben werden, was weniger Arbeitsaufwand bedeuten würde. Anderenfalls kann dies auf der CPU laufen und seine Ergebnisse an einen zweiten *InferenceResultController* weiterleiten, der aus den Ergebnissen sowohl eine genauere Position der erkannten Person, als auch deren Geste erkennt. Diese Daten können an die Fahrzeugsteuerung weitergeleitet werden, die hier ebenfalls nicht wirklich angepasst werden muss.

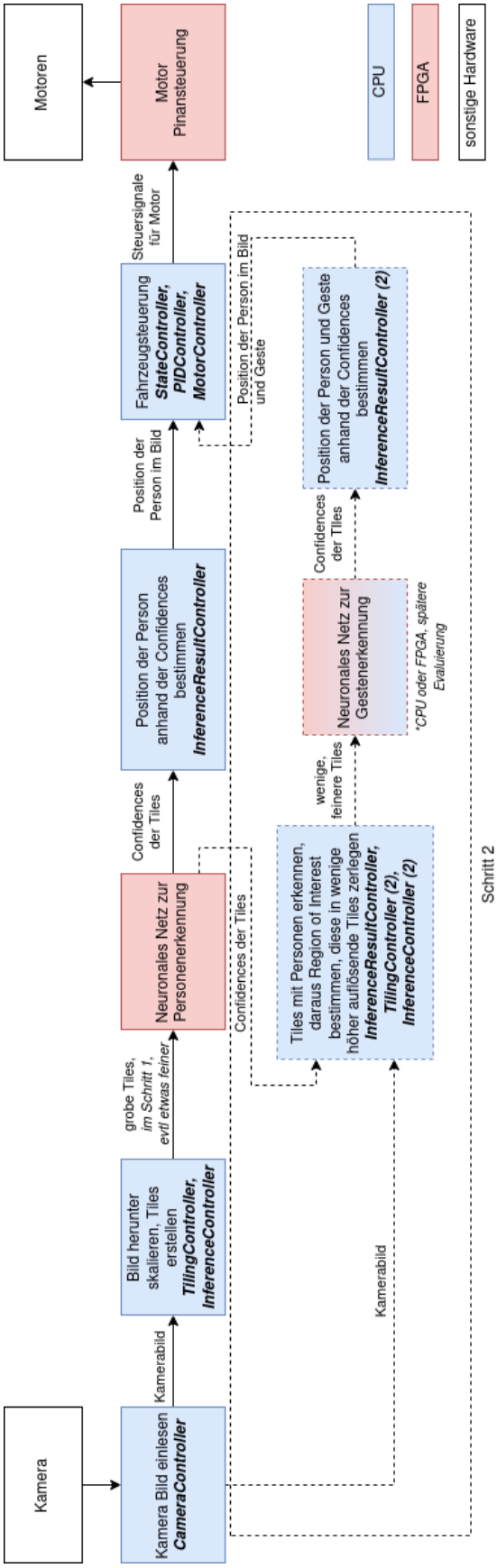


Abbildung 14: Konzept zur Verarbeitung innerhalb des Systems

4 Neuronales Netz zur Personenerkennung

4.1 Trainingsdaten

Geschrieben von Johannes Steffen

Um ein Neuronales Netz zu implementieren werden Trainingsdaten benötigt. Wir haben unsere Ausgangsbilder mittels Python Skripten gefiltert und unseren Bedürfnissen angepasst. Dies wird in folgenden Kapiteln genauer beschrieben.

Eine Übersicht über das Vorgehen mit dem Trainingsdatensatz ist im Anhang D mit entsprechenden Workflow Schritten zu finden.

4.1.1 Ausgangsbilder

Geschrieben von Ismail Sastim

Durch die Evaluation des mit dem COCO Datensatz trainieretem YOLO-Netzes ergab sich, dass eine zuverlässige und präzise Personenerkennung mit diesem Datensatz grundsätzlich möglich ist. Das ist nicht zu Letzt dem Umfang zu verdanken mit $\approx 330K$ Bildern und 91 Objektklassen darunter auch Personen. Folglich bietet der COCO Datensatz zusätzlich auch Vielfalt und Rauschen. Ferner haben die Autoren die Koordinaten der Bounding Boxen mit inkludiert, die es bei Bedarf erlauben, die Objekte aus den Bildern gezielt und automatisiert auszuschneiden. [Lin+15]

Ein merklicher Nachteil ist jedoch, dass viele Personenbilder existieren, wo die Personen nur sehr klein, teilweise oder unkenntlich abgebildet sind. Das ist für unseren Anwendungsfall unbrauchbar und zudem technisch schwer realisierbar, da die Bilder für das CNV Netz auf 32x32 herunterskaliert werden müssen.

Abbildung 15: Klein abgebildete Personen bei schlechten Lichtverhältnissen

Abbildung 16: Arme von Person aus Egoperspektive geschossen

Eine erste Evaluation mit dem COCO Datensatz aufgeteilt und balanciert auf Personen- und nicht-Personenbilder ergab mit den vorhandenen Skripten eine Genaugikeit von $53,125\%$. Dies lies vermuten, dass das CNV Netz vermeintlich nicht richtig gelernt hätte, da die statistische Wahrscheinlichkeit beim Raten bei $\approx 50\%$ liegt, weswegen entschieden wurde, den Datensatz zunächst händisch zu filtern und auf diese, ähnlich wie es auf dem Processing System durchgeführt wird, Softwaretiling anzuwenden, damit die Trainingsbilder so nah wie möglich an die Eingangsbilder kommt. Auch diese Maßnahmen führten zu schlechten Genauigkeitswerten von $\approx 50\%$. Später wurde festgestellt, dass das Validierungsskript erhebliche Bugs aufweist. Bei Systemtests zeigte sich jedoch eine zufriedenstellende Erkennungsrate von $> 80\%$. (siehe auch Kapitel 6.1)

4.1.2 Auswahl geeigneter Bilder

Geschrieben von Johannes Steffen und Ferhat Cansu

Bei der händischen Filterung der Bilder des COCO Datensatzes wurden in einem ersten Schritt für unser Szenario geeignete Bilder von Personen bestimmt.
Das autonome Fahrzeug soll aufrecht stehende Personen erkennen und verfolgen. Es ist bei der Auswahl der Trainingsdaten deswegen wichtig, dass die Personen auf den Bildern fast vollständig in einer aufrechten Position zu sehen sind. Die Personen sollten dabei freigestellt und nicht in einer Menge zusammen stehen, da wie im Testszenario (Anhang F) definiert, nur eine Person im Blickfeld des Fahrzeuges zu sehen seien soll. Dies entspricht dem ursprünglichen Verfolgungsszenario.
Die Personen sollen vollständig auf den auszuwählenden Bildern zu sehen sein, damit das spätere Tiling der Trainingsdaten (beschrieben in Kapitel 4.1.4) auch effektiv ist und einen ausgeglichenen Datensatz erzeugt und nicht zum Beispiel überproportional viele Beine im finalen Trainingsdatensatz vorhanden sind.

Die Auswahl der zu verwendenden Bilder ist mit der Software Lightroom Classic in der Version 10.0 vom Oktober 2020 erfolgt [Ado21] und wurde dementsprechend in einem proprietären Format in einem so genannten Lightroom Katalog festgehalten und später exportiert.

Für unser Trainingsdatensatz zur Personenerkennung wurden durch händische Auswahl in der Lightroom Software aus dem COCO Datensatz Bilder in denen Personen, welche von vorne, hinten und seitlich abgebildet sind manuell entnommen. Dabei wurde darauf geachtet, dass Personen deutlich und möglichst vollständig abgebildet sind und aufrecht stehen.

Mittels eines Skripts, welches im Git Repository des Personennetzes unter „scripts/filter_skript.py“ zu finden ist, werden daraufhin alle ausgewählten Dateinamen in einer Textdatei übertragen.
Bilder der Klassifizierung „noPerson“ wurden zufällig aus den restlichen Bildern auf denen laut COCO Annotationen keine Personen zu erkennen sind gewonnen. Dabei bietet der COCO Datensatz ein gutes Hintergrundrauschen mit sehr diversen Bildern von denen sich die händisch ausgewählten Personen Bilder abheben und deswegen eine gute Erkennungsrate zu erwarten ist.

4.1.3 Generierung des Trainingsdatensatzes

Geschrieben von Mattia Uhlenbrock

Der Trainings- und Validierungsdatensatz werden anhand der ausgewählten Bilder erzeugt und balanciert. Dabei ist das Verhältnis der Bilder etwa zehn zu eins.

Zunächst werden die IDs der ausgewählten Bilder aus dem COCO-Datensatz anhand der in einer Liste gespeicherten Dateinamen extrahiert. Anschließend werden die Annotationen des COCO-Datensatzes anhand der IDs gefiltert. Die gefilterten Annotationen enthalten die Information über die Bounding-Box des Objekts und dessen Klassifikation.

Für alle Annotation die die Klassifikation „person“ enthalten werden die Personen aus den Bildern des COCO-Datensatzes anhand der Bounding-Boxes und falls möglich mit einem zusätzlichen Bildauschnitt von 20% ausgeschnitten und in den entsprechenden Teildatensatz (Training oder Validierung) abgelegt. Die Bilder, die keine Person enthalten werden ohne weitere Bearbeitung in den entsprechenden Teildatensatz kopiert.

Nachdem alle ausgewählten Bilder kopiert sind, werden automatisch weitere Bilder ohne Person in den Datensatz kopiert, bis die Anzahl der Bilder von Personen und ohne Person balanciert ist. Anschließend wird das Verhältnis der Anzahl der Bilder in Trainings- und Validierungsdatensatz pro Klassifikation überprüft und gegebenenfalls solange Bilder zwischen den Teildatensätzen einer Klassifikation verschoben, bis das Verhältnis von zehn zu eins erreicht ist.

Der generierte Datensatz folgt der Verzeichnisstruktur in Abbildung 17.

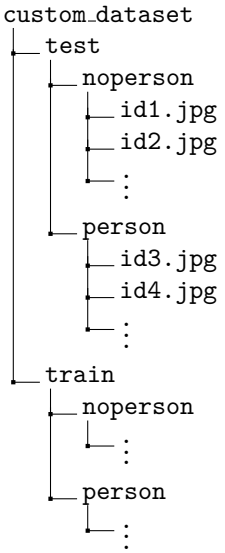


Abbildung 17: Verzeichnisstruktur

4.1.4 Tiling der Trainingsdaten

Geschrieben von Johannes Steffen, Philipp Altnickel

Das CNV Netz, welches von uns verwendet wird, kann wie in Kapitel 2.2 beschrieben nur Bilder mit 32x32 Pixeln Größe verarbeiten. Um eine gute Erkennungsrate zu erreichen ist es nicht sinnvoll das Kamerabild mit FullHD (1080p) Auflösung an unser Neuronales Netzwerk zu senden und als ganzes zu skalieren, sondern mittels Tiling (Kapitel 2.4) kleine Bildausschnitte an unser Netzwerk weiterzugeben.

Dieses sollte auch beim Training beachtet werden.

Die zugeschnittenen Personen, bzw. die ausgewählten nicht-Personen Bilder werden deswegen im nächsten Schritt in Tiles zerlegt, um so eine größere Genauigkeit bei der Erkennung im späteren Netz erzielen zu können.

Tiling der Personenbilder *Geschrieben von Philipp Altnickel*

Die Personenbilder werden dabei nach einem speziellen Algorithmus zerteilt, sodass möglichst optimale Ausschnitte entstehen. Es sollen quadratische Tiles entstehen, die zum einen nicht zu klein werden dürfen, da sonst nur einzelne, kleine Teile oder gar kein Teil einer Person abgebildet wären. Die Überlappung der Tiles sollte nicht zu klein sein, da es sonst möglicherweise unglückliche Teilungen gibt und Personen evtl. nur noch zerschnitten vorhanden sind. Aus diesem Grund wird der nachfolgend beschriebene Algorithmus verwendet. Die Implementierung ist im Git Repository des Personendatensatzes im Verzeichnis „scripts/cutouttiles.py“ zu finden.

Die Tilesgrößen werden immer anhand der kürzeren Seite (KS) ermittelt. Die Tiles werden in zwei Schritten erzeugt. Im ersten Schritt entspricht die Größe der Tiles der kürzeren Seite des Bildes, somit entstehen möglichst große Tiles. In zweiten Schritt entspricht die Tilehöhe 60% der Länge der kürzeren Seite, es entstehen drei Tiles mit 66% Überlappung mit dem Nachbartile.

In Richtung der längeren Seite (LS) wird die Anzahl der Tiles jeweils mit folgendem Befehl ermittelt:

$$math.ceil(2 * (imageLength/tileSize) - 1) \tag{3}$$

Diese Anzahl der Tiles wird gleichmäßig über die Bildlänge verteilt. Es entstehen somit Tiles, die immer mindestens 50% Überlappung zum Nachbartile besitzen (siehe Skizzen in Abbildung 18). Der Grund für diese relativ komplexe Berechnung der Anzahl der Tiles ist, dass diese quadratisch und gleichmäßig verteilt sein sollen. Da die Bilder in der Regel nicht extrem viel höher als breit sind ist somit die Anzahl der zu verteilenden Tiles selten deutlich größer als 5. Bei fester Überlappung würde das letzte Tile nie richtig passen und man würde entweder einen Teil weg lassen oder zwei sehr ähnliche Tiles haben, was bei dieser kleinen Anzahl einen großen Fehleranteil ausmacht.

Für alle Schritte gilt: Ist das Bild so klein, dass die entstehenden Tiles kleiner als 32x32 Pixel groß werden, werden diese ignoriert, da sie nicht gut für das spätere Training geeignet sind.

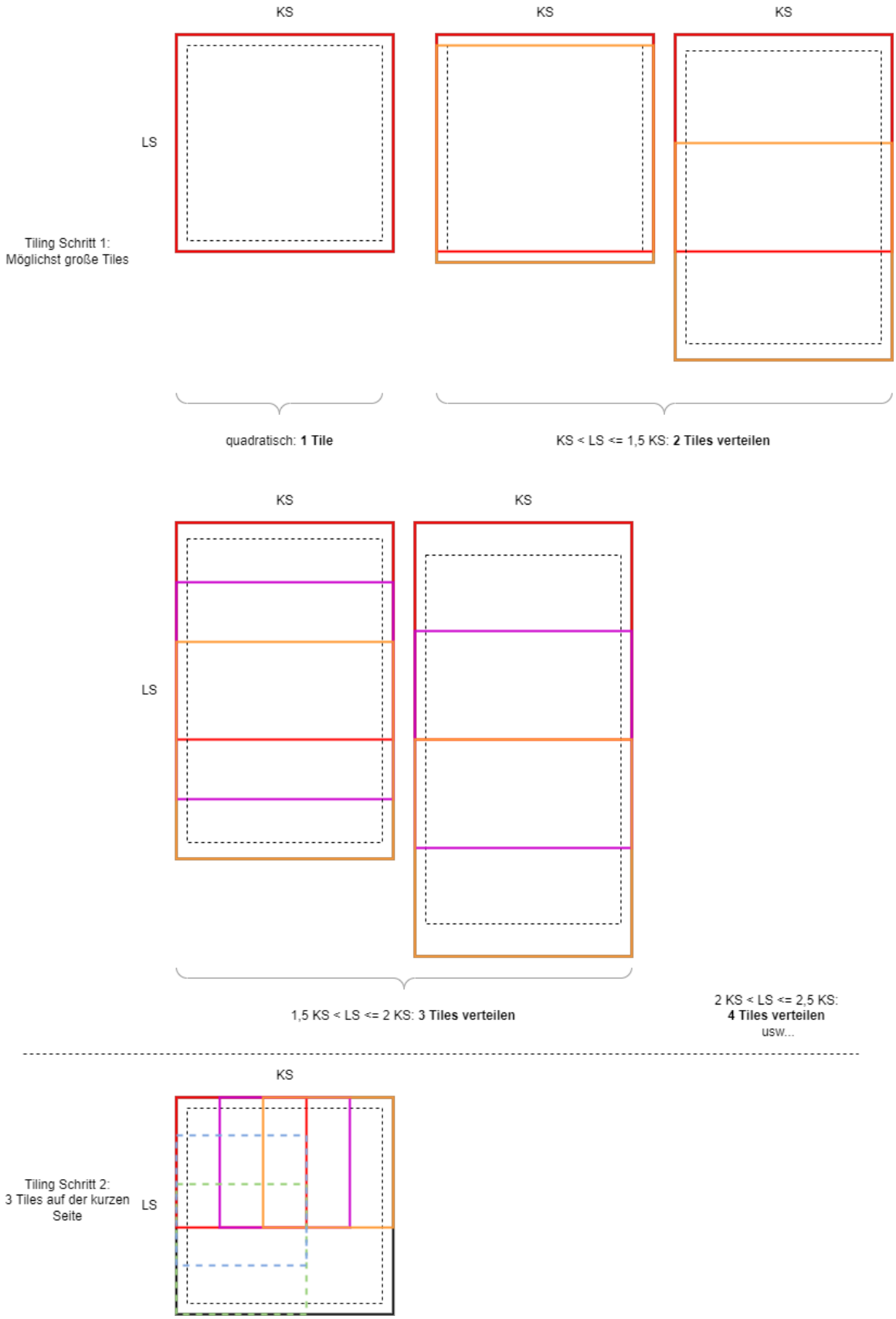


Abbildung 18: Prinzip des Tilings der Person-Tiles (KS = kürzere Seite, LS = längere Seite)

Tiling der nicht-Personenbilder Das Tiling der nicht-Personenbilder erfolgt nach einem wesentlich einfacheren Prinzip. Da es hier nicht auf den konkreten Inhalt ankommt, sind Überlappungen irrelevant. Hier wird einfach die Länge der kürzeren Seite durch eine Anzahl von Tiles n geteilt. In Richtung der längeren Seite werden dann so viele Tiles der Größe n, wie komplett darauf passen, ausgeschnitten und der Rest verworfen. Auch dies kann gemacht werden, da der konkrete Inhalt nicht relevant ist. Der Parameter n kann variiert werden, um am Ende eine einigermaßen ausbalancierte Anzahl von Personen und nicht-Personenbildern zu erhalten.

4.2 Training

Geschrieben von Mattia Uhlenbrock

Um das CNV-Modell auf Personenerkennung zu trainieren, wurden Anpassungen an der bestehenden Trainingsstruktur vorgenommen. Dabei wurde sowohl die Trainingsumgebung als auch die Trainingsskripte gegenüber dem Vorgängerprojekt verändert.

4.2.1 Vorbereitung

Bereits im Vorgängerprojekt wurde das Trainingsskript des FINN-Frameworks für das CNV-Modell modifiziert und auf den verwendeten Datensatz und die Anzahl der zu Klassifizierenden Objekte angepasst [Alt+21]. Diese modifizierten Trainingsskripte wurden auf die beiden Klassen „person“ und „noperson“ umgestellt und der Personendatensatz ohne Tiling 4.1.3 bzw. mit Tiling 4.1.4 eingebunden.

4.2.2 Durchführung

Da das Training von Neuronalen Netzen sehr rechenintensiv ist, wurde bereits im Vorgängerprojekt auf Möglichkeiten des Trainings in Cloud-Infrastruktur zurückgegriffen [Alt+21, S. 61ff]. Auch für die Durchführung des Trainings zur Personenerkennung wurde auf Cloud-Ressourcen zurückgegriffen, allerdings unter Verwendung der Google Cloud Plattform. Für eine Anleitung zum Einrichten einer Trainings-Instanz für Studenten siehe Anhang C.

4.3 Übersetzung für das FPGA

Geschrieben von Philipp Altnickel, Ismail Sastim

Das trainieren des Neuronalen Netzes erzeugt eine Ausgabedatei im onnx-Format. Damit diese auf dem FPGA laufen kann, muss sie im folgenden Schritt übersetzt werden. Es wird ein Bitstream erzeugt, der dann auf das FPGA geladen werden kann. Der Ablauf wurde bereits im vorherigen Projekt genutzt, daher ist dieser mit allen Einzelschritten im entsprechenden Bericht im Kapitel 8 [Alt+21, S. 64ff] und Anhang J [Alt+21, S. 133ff] beschrieben.

Die Ausführung des Synthetisierungsskriptes im aktuellen Semester hat jedoch zu Fehlern geführt. Dies war auf ein Bug in der genutzten Vivado 2021.1 Version zurückzuführen, welcher das Datum in ein *INT32* speichert und ab dem Jahr 2022 zu einem Überlauf führt. Als Workaround wurde nach dem Starten des Notebooks die Systemuhrzeit auf einen beliebigen Tag im Jahr 2021 zurückgestellt.[Pit22]

5 Einbindung in das System

Geschrieben von Philipp Altnickel

Der folgende Abschnitt beschreibt alle Änderungen, die nötig waren, um das vorbereitete Netz auf das System zu übertragen und lauffähig zu machen.

5.1 Übertragung des Netzes

Geschrieben von Philipp Altnickel

Der erzeugte Bitstream liegt dann in Form einer *.bit Datei mit zusätzlicher Beschreibung der Hardwarekonfiguration als *.hwh Datei vor. Anhang H des Berichts des Vorgängerprojektes [Alt+21, S. 128ff] beschreibt neben der Neugenerierung dieser Dateien auch das kopieren in das Processing System. Mit jedem Start des Fahrzeugs werden diese Dateien auf das FPGA geladen.

5.2 Anpassungen an der Base Station

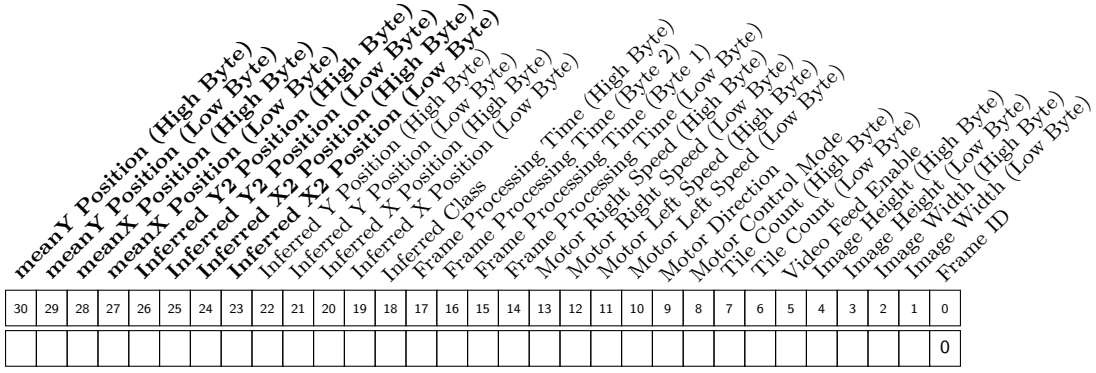
Geschrieben von Philipp Altnickel

Die nötigen Änderungen an der Base Station sind minimal. Einzig das Label für die erkannte Klasse musste angepasst werden. Wobei in dem aktuellen Schritt sowieso nur die Klasse „person“ erkannt werden kann. Somit kann das Label aktuell nur dazu genutzt werden, anzueigen, ob eine Person erkann wurde („person“) oder ob keine Person im Bild ist („noperson“).

Damit in folgenden Schritten die Gestenerkennung besser umgesetzt werden kann, das Processing System so erweitert, dass es auch zwei XY-Koodinaten für die obere linke und die untere rechte Ecke des Bereichs, in dem sich die Person befindet übermittelt. Vorher wurde lediglich die Mitte errechnet, übermittelt und mit einem Kreuz markiert. Nun zeigt die Base Station auch ein Rechteck um die Person an. So kann der an das spätere Gestennetz übermittelte Bereich besser analysiert werden.

Der neue Datensatz, der für jeden verarbeiteten Frame vom Fahrzeug an die Base Station gesendet wird, sieht nun folgendermaßen aus (Die neuen Daten sind fett markiert):

Register 5.1: Control Dataframe (Vehicle to Base Station)



5.3 Anpassungen am Tiling

Geschrieben von Philipp Altnickel

Das Tiling der mit der Kamera aufgenommenen Bilder hat sich im Vergleich zum Vorgängerprojekt nicht geändert. Details zur genauen Funktionsweise können im alten Bericht nachgelesen werden [Alt+21].

Nur die Konfiguration der Anzahl und Größe der zu erstellenden Tiles wurde über die Konfigurationsdatei „vehicle.conf“ so verändert, dass sie besser mit den veränderten Netz funktioniert. Das genaue Vorgehen, also welche Parameter schlussendlich gewählt wurden und wie diese bestimmt wurden, wird zusammen mit anderen Parametern in einem folgenden Abschnitt zur Optimierung näher erläutert.

5.4 Anpassung der Fahrzeug Steuerung

Geschrieben von Philipp Altnickel

Da die Speicheradressen zum Übertragen der Daten zu und von dem Neuronalen Netz auf dem FPGA identisch zum bisher verwendeten Netz geblieben sind, mussten an dieser Stelle keine Änderungen am Code vorgenommen werden. Lediglich die Anzahl der Klassen hat sich verringert, sodass der InferenceResultController nun weniger Confidences pro Tile zurück erhält. Aus den Confidences wird ähnlich wie bisher auch die Position der Person errechnet, in dem der Mittelwert aus allen Tilepositionen gebildet wird, in denen die Wahrscheinlichkeit für eine erkannte Person über einem bestimmten Grenzwert liegt. In der folgenden Verarbeitung ist davon hauptsächlich der X-Wert von Bedeutung, da aus ihm die nötigen Lenkbewegungen zum verfolgen der Person ermittelt werden müssen.

Die Fahrzeugsteuerung selbst wurde bisher als Zustandsautomat implementiert. Da das Projekt bisher nie so weit war, dass dieser sinnvoll getestet werden konnte, stellte sich an dieser Stelle heraus, dass seine Funktion nicht einwandfrei war. Da bisher sowieso keine funktionierende Gestenerkennung implementiert ist, und somit noch immer kein sinnvoller Test stattfinden kann, wurde der Zustandsautomat in seinen wesentlichen Teilen auskommentiert und durch einen einfachen Wechsel zwischen Search-Mode und Approach-Mode ersetzt, je nachdem ob eine Person erkannt wird oder nicht. (Weitere Arbeit am Zustandsautomaten sollte dann stattfinden, wenn die Gestenerkennung lauffähig gemacht wird.)

5.5 Optimierung des Fahrzeugs

Geschrieben von Johannes Steffen, Philipp Altnickel

Die wesentlichen Parameter aus der Datei „vehicle.conf“, die eine Verbesserung der Erkennung möglich machen, sind „PRESENCE_CONFIDENCE_THRESHOLD“, „PRESENCE_CLASS_THRESHOLD“ und „TILE_SPEC“. Aus diesem Grund wurden bei Testläufen auch hauptsächlich an diesen Parametern Änderungen vorgenommen.

„PRESENCE_CONFIDENCE_THRESHOLD“ ist dabei so anzupassen, dass die Confidence von nichtpersonen Objekten im Bild unterhalb diesen Grenzwertes liegt und die aller Personen über diesem. Als guter Kompromiss wurde dabei 370 identifiziert. Es gibt leider Objekte - in den Tests waren dies z.B. oft Stühle - bei denen sich die Wahrscheinlichkeit der person-Klasse wenig von der einer echten Person unterscheidet. „PRESENCE_CLASS_THRESHOLD“ ist die Anzahl von korrekt erkannten Tiles, die es geben muss, damit das Fahrzeug annimmt, es isdt wirklich eine Person im Bild. So können einzelne, falsch erkannte Tiles kompensiert werden. Da Problem dabei ist, dass diese Anzahl natürlich stark von der Anzahl der insgesamt erzeugten Tiles abhängt. Gibt es sehr viele Tiles, entstehen auch mehr fehlerhaft korrekt erkannte. Für die im folgenden als am Besten identifizierte Tile Spezifikation hat sich ein „PRESENCE_CLASS_THRESHOLD“ von 5 als sinnvoll herausgestellt.

Der wohl entschiedenste Parameter dafür, wie gut die Steuerung des Fahrzeugs am Ende funktioniert ist die „TILE_SPEC“. Sie legt die Größe und Anzahl der erzeugten Tiles fest. Bei Testläufen mit verschiedenen Einstellungen hat sich gezeigt, je kleiner die Tiles sind, die erzeugt werden, desto größer wird die Distanz, in der eine Erkennung möglich ist. Eine größere Distanz der Person zum Fahrzeug sorgt auch dafür, dass die Steuerung besser funktioniert. Ist die Person sehr bildfüllend, springt die im Mittel erkannte Position sehr viel hin und her. Das Fahrzeug weiß dann nicht genau wo es hinfahren soll. Der limitierende Faktor, der einen Kompromiss bei der Anzahl der Tiles nötig macht, ist allerdings die Geschwindigkeit der Verarbeitung. Je kleiner die Tiles werden, desto größer wird logischerweise auch die Anzahl. Es hat sich gezeigt, dass eine Verarbeitungsgeschwindigkeit von 5-6 Bildern pro Sekunde notwendig ist, um die

Lenkung gut zu steuern. Bei zu langsamer Verarbeitung ist die Verzögerung bei der Reaktion zu groß und das Fahrzeug fährt zu lange Kurven, obwohl die Person schon wieder woanders steht. Ein Ansatz war hier noch, die Fahrzeuggeschwindigkeit insgesamt zu senken, allerdings sind die Motoren nicht geeignet um noch langsamer zu fahren. Als besten Kompromiss zwischen möglicher Distanz und Verarbeitungsgeschwindigkeit hat sich die folgende „TILE_SPEC“ ergeben: „32,32,1,1 64,64,.75,.75, 128,128,1,1“. Mit ihr waren Distanzen von ca. 4-5 Metern und eine Verarbeitung von ca. 6 Bildern pro Sekunde möglich.

Im Search Mode, in welchem sich das Fahrzeug dreht um eine Person zu finden, ist dem Neuronalen Netz eine eindeutige Personen Erkennung nicht möglich. Das Fahrzeug drehte sich in den meisten Tests fortwährend im Kreis und es ensteht ein verwischtes Kamerabild. Dies ist leider nicht lösbar, da die Ansteuerung der verwendeten Motoren nicht noch kleinschrittiger passieren kann. Deswegen wurde das Feature des Drehens im Search Mode entfernt. Das Fahrzeug muss also von einem Operateur abgeholt werden indem dieser sich in das Sichtfeld des Roboters bewegt und damit der Approach Mode und die damit verbundene Drehung hervorgerufen wird.

6 Evaluation

Um die Fertigstellung zu verifizieren und das Resultat zu validieren, wurden verschiedene Merkmale evaluiert.

Genauigkeit des Personennetzes Das Neuronale Netz erfüllt die Kernfunktionalität des Projektes. Entsprechend ist die Evaluation der Genauigkeit ausschlaggebend. Die Überprüfung wurde automatisiert anhand Pythonskripte ermittelt.

Ressourcenverbrauch Die erzeugten IP-Blöcke aus dem trainierten Netz müssen auf das verwendete FPGA-Board passen damit die Inferenz der Bilder überhaupt funktioniert. Zudem kann so auch verifiziert werden, ob ein ähnlich ausgestattetes strahlungstolerantes FPGA-Board ebenfalls genügend Kapazitäten hätte. Neben der Frage ob, ist auch von Interesse, wie viel Puffer noch übrig ist. Überprüft werden kann das mit Vivado 2021.1.

Systemtest Da zu diesem Semester vorgenommen wurde ein *"lauffähiges"* Endresultat zu erzielen und das Projekt aus verschiedenen Komponenten besteht, ist ein Systemtest essentiell. Um dies zu überprüfen wurde das Fahrzeug in einem Raum des ZIMT platziert und sichergestellt, dass eine Person im Sichtfeld der Kamera zu sehen ist. Die Person hat in Schrittgeschwindigkeit in verschiedene Richtungen bewegt. Es wurde dabei beobachtet, ob das Fahrzeug grundsätzlich einer Person mit einer angemessenen Reaktionsgeschwindigkeit verfolgen kann.

6.1 Genauigkeit des Personennetzes

Geschrieben von Ismail Sastim

Um die Genauigkeit des Neuronalen Netzes zu messen, wurden die vorhandenen Python Skripte genutzt. Da erst spät gegen Ende des Semesters einige Bugs auffielen, durch welche die Ergebnisse zumindest für unseren Personendatensatz unbrauchbar wurden, konnte einiges nicht nachgewiesen werden. Beispielsweise dass das manuelle Aussortieren der Bilder oder das Tiling vor dem Training zu einer Verbesserung führen. Genauso fehlte die Zeit, um die Variation der Parameter Batch Size und Epochen genau zu untersuchen. Es wurde im laufenden Endergebnis auf Basis der Erkenntnisse aus dem letzten Semester mit 250 Epochen und einer Batch Size von 64 trainiert und auf dem Board aufgespielt. Darauf folgend wurde eine **subjektive Einschätzung** gemacht, die zeigte, dass Personen zuverlässig in $> 80\%$ der Fälle erkannt werden. Auffällig war jedoch, dass Bürostühle häufig vom trainierten Netz ebenfalls als Personen erkannt wurden. Dies könnte durch hinzufügen einiger Bilder in die *noperson*-Klasse ggfs. behoben werden.

6.2 Ressourcenverbrauch

Geschrieben von Ismail Sastim

Für die Ermittlung der verbleibenden Kapazitäten des FPGA wurde Vivado genutzt.

Ressource	Benutzung	Verfügbar	Auslastung in %
LUT	23'119	53'200	43,46
LUTRAM	1'554	17'400	8,93
FF	31'003	106'400	29,14
BRAM	98	140	70,00
IO	6	125	4,80
BUFG	1	32	3,13

Tabelle 2: Ressourcenverbrauch laut Vivado Summary für Neuronales Netz zur Personenerkennung

Es zeigt sich, dass noch einiges an Ressourcen übrig sind, um ggfs. weitere Funktionalitäten auf das FPGA auszulagern wie das Hardware-Tiling nach [Mül21]. Dieser benötigt je nach Parameter folgende Ressourcen:

Ressource	**MIN**	**MAX**
LUT	9'498	21'717
FF	11'889	38'331
BRAM	6,5	6,5
IO	14	14

Tabelle 3: Ressourcenverbrauch für Hardware-Tiling nach [Mül21]

Es werden nur die Angaben *LUT*, *FF*, *BRAM* betrachtet, da die anderen Ressourcen in [Mül21] nicht angegeben sind. Diese sollten schätzungsweise aber genügen um abzuschätzen, dass das Hardware-Tiling noch auf das FPGA passen könnte selbst unter Betrachtung der Maximalwerte, da die anderen Ressourcen kaum benötigt werden.

Ressource	**Benutzung**	**Verfügbar**	**Auslastung in %**
LUT	44'836	53'200	84,28
FF	69'334	106'400	65,16
BRAM	104,5	140	74,64
IO	20	125	16

Tabelle 4: Theoretischer Ressourcenverbrauch für Personenerkennung inklusive Hardware-Tiling nach [Mül21]

6.3 Systemtest

Geschrieben von Johannes Steffen

Einige der bereits erwähnten Tests und der Systemtest wurden in einem Video festgehalten. Dieses Video ist im Main Git Repository zu finden.

In diesem Video sind 2 Tests und der Systemtest zu finden.

Da es zeitlich nicht möglich war auch die Gestenerkennung abzuschließen, wurde ein Testszenario nur zur Überprüfung der Personenerkennung angefertigt und im Anhang G angehangen. Im Video ist zu erkennen, dass mitten im Test die zu verfolgende Person gewechselt wurde. Dies wurde getan, um die Diversität des Videos, welches mit einer GoPro Hero 5 geschossen wurde zu erhöhen.

Generell ist in allen Tests eine Lauffähigkeit des Fahrzeugs zu sehen. Das Fahrzeug ist jedoch nur in kontrollierten Umgebung lauffähig und noch keineswegs für einen komplett autonomen Betrieb bereit. Es wurde eine Art Arena mit umgekippten Tischen gebaut um Stühle und andere sich im Umkreis befindlichen Objekte aus dem Blickfeld der Kamera zu entfernen. Dies erhöhte die Erkennungsrate von Personen enorm. Stühle und andere dunklen Objekte wurden sonst häufig auch als Personen erkannt und entsprechend verfolgt.

6.3.1 High FPS Test

Bei diesem Test wurden folgende Parameter für das Tiling verwendet: „TILE_SPEC“: „64,64,1,1, 128,128,1,1“. Dies entspricht, wie man auch der BaseStation Software im Video entnehmen kann, 24 Tiles insgesamt.

Wie bereits in dem Video zu sehen ist das Fahrzeug relativ agil und kann sich aus schwierigen Situationen befreien. Das bedeutet, dass auch wenn das Fahrzeug sich nah an einer Wand befindet eine Person in engster Nähe erkennt und verfolgt wird. Dies ist bei geringeren FPS zum Beispiel nicht der Fall.

Es dauert jedoch bei einer schrittweisen Näherung der zu verfolgenden Person relativ lange bis das Fahrzeug diese erkennt. Die Entfernung in dem eine Person überhaupt erkannt wird ist geringer als bei anderen Einstellungen des Tilings. Eine Verfolgung bei einer Distanz von 1-2 Meter ist mit diesen Einstellungen möglich. Für eine flüssige Verfolgung durch das Fahrzeug muss die Person ständig erkannt werden. Dies ist, wie im Video zu sehen leider nicht der Fall. Die Verfolgung bricht häufig ab und die Testperson muss sich dem Fahrzeug nähern, damit dieses die Verfolgung wieder aufnimmt.

Beim High FPS Test wurden ca. 10-12 FPS erreicht.

6.3.2 Low FPS Test

Bei diesem Test wurden folgende Parameter für das Tiling verwendet: „TILE_SPEC“: „32,32,0.5,0.5, 64,64,0.75,0.75, 128,128,1,1“. Dies entspricht, wie man auch der BaseStation Software im Video entnehmen kann, 396 Tiles insgesamt.

Bei geringen FPS und mehr Tiles reagiert der Roboter träge. Bei einem sehr geringen Abstand der zu verfolgenden Person steuert der Roboter zu schnell und verliert schnell die Person aus dem Blickfeld der Kamera. Leider ist es nicht möglich die Räder feiner anzusteuern, da bei geringeren Spannungen die Räder gar nicht mehr drehen würden. Auf Distanz funktioniert die Verfolgung jedoch gut. Wegen der vielen Tiles kann auch eine Verfolgung einer Person im Abstand von knapp 6 Metern erfolgen.

Beim Low FPS Test wurden ca. 2-3 FPS erreicht.

6.3.3 Systemtest

Aus diesen beiden Tests haben wir gelernt, dass bei höheren FPS das Fahrzeug schneller reagiert, was bei der Verfolgung einer Person in naher Distanz notwendig ist. Viele Tiles und deswegen wenig FPS sind jedoch notwendig um eine Verfolgung mit größerer Distanz zur Person durchzuführen. Als Optimum haben sich deswegen folgende Einstellungen des Tilings ergeben. „TILE_SPEC“: „32,32,1,1, 64,64,.75,.75, 128,128,1,1“. Dies entspricht, wie man auch der BaseStation Software im Video entnehmen kann, 138 Tiles insgesamt.

Beim Systemtest wurden ca. 6 FPS erreicht.

6.3.4 Vergleich der Frames per Second

Bereits in Vorgängerprojekten [Mül+21] in denen es um die Umsetzung eines Neuronalen Netzwerkes zur Erkennung und Verfolgung von Verkehrszeichen ging wurde in einem Video gezeigt, dass auch bei geringen FPS im Bereich 2 bis 3 das Fahrzeug relativ zeitnah und damit zufriedenstellend reagiert. Wir behaupten, dass unsere Fahrzeugsteuerung mit ca. 6 FPS flüssiger als mit 2-3 FPS Tests funktioniert, aber unerheblich schlechter als mit 10-12.

Für folgende Projekte und Tests empfehlen wir deswegen für eine gute Lauffähigkeit mindestens 5 FPS.

Die Berechnung der FPS (Frames Per Second) erfolgte bei allen Tests innerhalb der Software auf dem autonomen Fahrzeug selber und wurde dem Benutzer via der BaseStation auf dem Bildschirm mitgeteilt.

6.4 Verifikation der Projektziele

Geschrieben von Johannes Steffen

Unser Projektziel war es ein lauffähiges autonomes Fahrzeug zu entwickeln. Dies ist wie im Systemtest zu erkennen mit entsprechenden Einschränkungen geglückt. Das Testszenario im Anhang F wurde den Umständen zur Verfolgung einer Person ohne Gestenerkennung angepasst und durchgeführt. Eine relativ flüssige Verfolgung einer einzelnen Person war bei der Durchführung des Tests nachweisbar. Die zu verfolgende Person muss sich aktiv in das Blickfeld der Kamera bewegen um eine Verfolgung zu initiieren. Auch bei langsamen Schritttempo verliert der Roboter manchmal die zu verfolgende Person. Eine Bewegung der Person zurück in das Blickfeld der Kamera ist meistens notwendig um die Verfolgung dann wieder aufzunehmen.

7 Ergebnisse und Ausblick

7.1 Zusammenfassung der Ergebnisse

Geschrieben von Philipp Altnickel

Zusammenfassend lässt sich also sagen: Der zu Anfang im Konzept ausgearbeitete Ansatz, das System in zwei separate Neuronale Netze aufzutrennen wurde prinzipiell erfolgreich in Angriff genommen. Allerdings wurde aus Zeitgründen lediglich der erste Teil, also die Implementierung eines Neuronalen Netzes rein zur Personen durchgeführt. Das separate Gestennetz konnte nur in Ansätzen vorbereitet werden.

Bei der Auswahl eines Ansatzes für die Personenerkennung wurden verschiedene Möglichkeiten betrachtet, schlussendlich haben sich aber viele als relativ komplex und schlecht dokumentiert herausgestellt, sodass eine Implementierung mit viel Zeitaufwand und nicht garantiertem Erfolg bei der schlussendlichen Brauchbarkeit verbunden gewesen wäre. Aus diesen Gründen wurde lediglich an einem neuen Datensatz gearbeitet und mit der im wesentlichen unveränderten Architektur des Vorgängerprojektes trainiert.

An der Evaluation der Arbeitsergebnisse lässt sich erkennen, dass das implementierte Neuronale Netz durchaus sichtbare Erfolge erzielt und auf jeden Fall ansatzweise eine Personenverfolgung möglich ist. Somit konnte zumindest das erste Projektziel, eine funktionierende und lauffähige Personenverfolgung auf dem Fahrzeug zu implementieren erreicht werden. Des Weiteren konnte das Nebenziel der Wissensquise vorangetrieben werden. Neben vielen immer noch offenen Verbesserungs- und Erweiterungsvorschlägen aus dem letzten Projekt, haben sich dadurch noch weitere Optimierungsmöglichkeiten ergeben, insbesondere bei der Verbesserung der Erkennung durch eine eigens erstellte Neuronale Netzarchitektur.

Es sind also durchaus Erfolge zu verzeichnen, allerdings bleiben neben der Umsetzung des zweiten Projektzielen - des Gestennetzes - noch diverse andere Ansätze zur weiteren Arbeit.

7.2 Ausblick

Im Folgenden sind einige Ideen und Ansätze aufgelistet, wie an diesem Projekt weiter gearbeitet werden könnte.

7.2.1 Verbesserung der Personenerkennung

Geschrieben von Mattia Uhlenbrock

Die Personenerkennung funktioniert grundlegend. Allerdings sind aus aktueller Sicht weitere Verbesserungsarbeiten möglich, die in folgenden Punkten aufgelistet werden:

- Der Personendatensatz kann erweitert werden. Dabei können sowohl mehr Bilder mit Personen allgemein aber auch mehr szenariospezifische Bilder ausgesucht werden. Ebenfalls denkbar wäre es weitere Bilder für ein szenariospezifisches Hintergrundrauschen zu selektieren.
- Da nicht klar ist, wie gut das verwendete neuronale Netz geeignet ist, wäre auch die Entwicklung eines speziell für die Personenerkennung geeignete Architektur denkbar.
- Denkbar ist auch die Verwendung eines Netzes, dass speziell für Single Class Detection entwickelt ist. Somit würden die Negativbeispiele (noperson) beim Training entfallen.

7.2.2 Überarbeitung der Skripte

Geschrieben von Ismail Sastim

Die Ergebnisse des Validierungsskripts widersprechen sich mit den Beobachtungen aus dem Systemtest. Dies sollte genauer untersucht werden, da ansonsten nicht nachweislich bewiesen werden kann, ob eine Veränderung zu einer Verbesserung oder Verschlechterung führt.

Auch das Trainingsskript kann eine Überarbeitung benötigen, da nicht genau vorausgesagt werden kann, wie viele Epochen optimal sind für das Training. Hierzu muss momentan jedes Mal neu Trainiert werden mit veränderter Epochenzahl. Sinnvoller wäre ein Checkpoint-Feature. Also dass nach jeder Epoche das trainierte Model validiert und gespeichert wird bis zu einer gesetzten Obergrenze, sodass man am Ende den Stand mit der höchsten Genauigkeit weiternutzen kann. Dies würde zum einen den Evaluationsaufwand wesentlich vereinfachen und letztendlich die Genauigkeit erhöhen. Evtl. kann sich auch an den Beispielskripten von Brevitas [Pap21] orientiert werden, da diese eine ähnliche Funktion bieten.

7.2.3 Einbinden der Gestenerkennung

Geschrieben von Philipp Altnickel

Wie in der Konzeption dieses Berichtes bereits erwähnt, sollte das Fahrzeug um ein weiteres Neuronales Netz zur Erkennung von Gesten erweitert werden. Dies konnte aus Zeitmangel in diesem Semester nicht mehr umgesetzt werden. In einem extra Branch des Git-Repositories „vehicleps“ wurde allerdings schon angefangen, die in der Konzeption skizzierte Umsetzung (Abbildung 14) zu implementieren. Die Arbeit könnte fortgesetzt werden.

Konkrete Punkte an denen gearbeitet werden müsste wären:

- Integration des Gestennetzes aus dem letzten Semester. Dies kann, wenn es auf das FPGA passt entweder auch in Hardware oder alternativ in Software passieren (dann müsste auch evaluiert werden, ob die Leistung ausreichend ist).
- Alternativ kann auch ein neues Neuronalen Netzes für diesen Zweck entwickelt werden. Es ist nicht ganz sicher, wie gut das alte Netz geeignet ist. Dies müsste ebenfalls evaluiert werden.
- Die bereits begonnene Implementierung der Controller fortsetzen und die Ansteuerung des Gestennetzes umsetzen.
- Anschließend den StateController überarbeiten, sodass die erkannten Gesten in entsprechendes Verhalten des Fahrzeugs umgesetzt werden können. (Der Zustandsautomat funktioniert vermutlich nicht richtig und wurde auch nie richtig getestet.)

7.2.4 Optimierung CPU Inferenz

Geschrieben von Ismail Sastim

Zu Anfang während der Zielfindung wurde bei Recherchen festgestellt, dass die Inferenz auf weit verbreiteten, leistungsschwachen ARM-Systemen ein populäres Thema zu sein scheint. Aufgrund der beschränkten Ressourcen und der schlechten Parallelität gibt es verschiedene Tools und Frameworks die Inferenz auf solchen Systemen weiter zu optimieren. Falls der Ansatz verfolgt werden sollte, das Fahrzeug um weitere Neuronale Netze zu erweitern, die nicht mehr auf das FPGA passen und auf die CPU ausgelagert werden sollen, lohnt es sich hier nochmals gründlich den aktuellen Stand zu recherchieren. Mögliche Softwarelösungen könnten sein:

- XNNPack [Goo21b]
- TensorFlow Lite [Goo21a]
- ARM NN SDK [ARM21b]
- Tengine [LAB21]
- OpenVINO Toolkit [Int21]
- ARM Compute Library [ARM21a]
- Die meisten Frameworks sind auch als Bibliothek für C++ oder andere Compilersprachen verfügbar.

Diese sollten für die Inferent in Erwägung gezogen werden, statt auf die Bequemlichkeit von Python zu setzen.

- Es könnte sein, dass ein in beispielsweise PyTorch oder TensorFlow trainiertes Model in einem anderen Framework wie OpenCV [Ope21] importiert werden kann und schneller inferiert.

7.2.5 Weiterführende Konzeption eines selbst entworfenen Neuronalen Netzes

Geschrieben von Ferhat Cansu

Es gibt viele Möglichkeiten ein Neuronales Netz für eine Personenerkennung oder allgemein zur Erkennung zu erstellen. Das CNV ist durch seinen Aufbau speziell für die Erkennung von Objekten erstellt worden, daher eignet es sich besonders gut. Wichtige Elemente für das entwerfen eines eigenen Netzes wurde hierfür in den vorherigen Seiten des Berichts erläutert. Das wichtigste jedoch ist die Zusammenstellung der unterschiedlichen Operationen (Layer). Vorher sollte man sich jedoch über die Eingabedaten Gedanken machen, ob es sich um einen Farbkanal (Graustufenbild) oder um mehr Farbkanäle z.B. drei Farbkanäle (RGB) handelt. Daraufhin sollten die Eingabedaten durch einen Convolutional Layer mit mehreren Filtern auf bestimmte Formen untersucht werden. Es ist empfehlenswert, diesen Schritt mit unterschiedlichen Filtern zweimal zu wiederholen, bevor man durch einen Pooling Operator nur die Wichtigen Informationen Filtert. In erster Linie sollte für eine Personenerkennung zum Beispiel die Kantenerkennung stehen, da das menschliche Gehirn dazu ausgelegt ist Kanten zuerst zu analysieren. Die Filter werden durch das Learning verfahren des Netzes selbst erlernt. Die Filterung der Eingabedaten kann mehrmals wiederholt werden, je nach Komplexität der zu erkennenden Objekte. Hierzu kann man sich am Aufbau des CNV orientieren. Zum Schluss müssen die gefilterten Daten klassifiziert werden, dazu werden Fully Connected Layer verwendet. Es wäre möglich ein Netz selbst zu entwerfen, welches dem CNV ähnelt, eventuell mit mehreren Wiederholungen der Convolutional- und der Pooling-Layer. Es wäre durchaus sehr interessant die Genauigkeit der Erkennung anhand der während des Learning Verfahrens selbst erlernt und der selbst entworfenen Filter zu analysieren und darzustellen.

7.2.6 Verbesserung am Fahrzeug

Geschrieben von Johannes Steffen, Philipp Altnickel

An der Hardware des Fahrzeugs und auch den Software Komponenten der Steuerung des Fahrzeugs könnten Verbesserungen vorgenommen werden.

- Bessere Motoren, die eine genauere Steuerung erlauben. Aktuelle Motoren sind ungenau, links ist z.B. etwas schneller als rechts, sodass er immer eine leichte Kurve fährt. Sie können auch nicht sehr langsam fahren. Dies führt zu verwischten Kamerabildern.
- Search Mode überarbeiten. Das Fahrzeug dreht sich einfach nur für eine Zeit x und sucht dann wieder eine Person. Das ist nicht so gut dafür geeignet sinnvoll eine Person wieder zu finden.
- Durch die Verwendung von mehreren Kameras mit größerem Blickwinkel könnte ein solcher Search-mode auch ohne die Bewegung des Fahrzeugs funktionieren.
- Mittels eines Gimbals oder einer anders angebrachten Kamera könnten Bewegungen des Fahrzeugs kompensiert werden, was zu einer besseren Kameraqualität und damit wahrscheinlich zu höheren Erkennungsraten auch während der Bewegung des Fahrzeugs führen könnte.
- Es könnte ein anderes FPGA verwendet werden um eine Implementierung von YOLO auf jenem FPGA durchführen zu können.

7.2.7 Sonstiges

Weitere Punkte, die bearbeitet werden könnten sind:

- Tiling der Bilder in Hardware auf dem FPGA durchführen. Dazu kann der Ansatz aus der Bachelorarbeit von Felix Müller [Mül21] eingebunden werden.
- Einen Crosscompiler implementieren, sodass der Code des Processing System nicht auf dem doch relativ langsamen Prozessor des Pynq Boards durchgeführt werden muss. Dies würde den Entwicklungsprozess der Software deutlich vereinfachen und beschleunigen.

Literatur

[Ado21] Adobe. *Desktop photo editor | Adobe Photoshop Lightroom Classic*. en-US. 2021. URL: https://www.adobe.com/products/photoshop-lightroom-classic.html (besucht am 13. 12. 2021).

[Ale16] Alexey. *Yolo v4, v3 and v2 for Windows and Linux*. Dez. 2016. URL: https://github.com/AlexeyAB/darknet (besucht am 13. 12. 2021).

[Alt+21] Philipp Altnickel u. a. *Gesten- und Objekterkennung durch schwache FPGAs in autonomen Fahrzeugen mittels neuronaler Netze*. de. Techn. Ber. Hochschule Bremen, 1. Sep. 2021, S. 153. URL: http://homepages.hs-bremen.de/~jbredereke/downloads/gestenerkennung-fpga-neuronale-netze-projekt-21.pdf.

[ARM21a] ARM. *ARM Compute Library*. 2021. URL: https://github.com/ARM-software/ComputeLibrary (besucht am 2021-11-28).

[ARM21b] ARM. *ARM-NN*. 2021. URL: https://github.com/ARM-software/armnn (besucht am 2021-11-28).

[Aus18] Holger Aust. *Das Gehirn kopieren? – Künstliche neuronale Netze*. Apr. 2018. URL: https://link.springer.com/chapter/10.1007/978-3-662-62336-7_6 (besucht am 13. 12. 2021).

[Boh21] Prof. Dr. Uta Bohnebeck. *Big Data and Machine Learning Neural Networks*. Englisch. Techn. Ber. Hochschule Bremen, 1. Okt. 2021. 56 S. URL: https://aulis.hs-bremen.de/goto.php?target=file_1533796_download&client_id=hsbremen (besucht am 01. 01. 2022).

[EGW07] Mark Everingham, Luc van Gool und Chris Williams. *The PASCAL Visual Object Classes Challenge 2007*. 2007. URL: http://host.robots.ox.ac.uk/pascal/VOC/ (besucht am 08. 12. 2021).

[Ge+21] Zheng Ge u. a. *YOLOX: Exceeding YOLO Series in 2021*. Dez. 2021. URL: https://github.com/Megvii-BaseDetection/YOLOX (besucht am 13. 12. 2021).

[Gle21] Ultralytics Glenn Jocher. *ultralytics/yolov5*. Dez. 2021. URL: https://github.com/ultralytics/yolov5 (besucht am 13. 12. 2021).

[Goo21a] Google. *TensorFlow Lite*. 2021. URL: https://www.tensorflow.org/lite (besucht am 2021-11-28).

[Goo21b] Google. *XNNPACK*. 2021. URL: https://github.com/google/XNNPACK (besucht am 2021-11-28).

[Int21] Intel. *OpenVINO Toolkit*. 2021. URL: https://github.com/openvinotoolkit/openvino (besucht am 2021-11-28).

[LAB21] OPEN AI LAB. *Tengine*. 2021. URL: https://github.com/OAID/Tengine (besucht am 2021-11-28).

[Lin+15] Tsung-Yi Lin u. a. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].

[Liu+15] Wei Liu u. a. „SSD: Single Shot MultiBox Detector“. In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[MPi21] MPii. *MPII Human Pose Dataset*. Apr. 2021. URL: http://human-pose.mpi-inf.mpg.de/ (besucht am 08. 01. 2022).

[Mül+21] Felix Müller u. a. *Applying Binarized Neural Networks on FPGAs to an Autonomous Driving Problem*. Englisch. Techn. Ber. Hochschule Bremen, 31. März 2021. 50 S. URL: http://homepages.hs-bremen.de/~jbredereke/de/forschung/veroeffentlichungen/bnns-on-fpgas-driving-projekt-2021.html (besucht am 21. 06. 2021).

[Mül21] Felix Müller. „Dynamisches Tiling auf schwachen FPGAs zur Objekterkennung mithilfe kleiner neuronaler Netze“. BSc-Thesis. Hochschule Bremen, 21. Juni 2021. 87 S. URL: http://homepages.hs-bremen.de/~jbredereke/downloads/mueller-bsc-thesis-2021.pdf (besucht am 06. 12. 2021).

[Ope21] OpenCV. *OpenCV*. 2021. URL: https://opencv.org/ (besucht am 2021-11-28).

[Pap21] Alessandro Pappalardo. *Xilinx/brevitas*. Version latest. 2021. URL: https://github.com/Xilinx/brevitas/tree/master/src/brevitas_examples/bnn_pynq (besucht am 2021-07-04).

[Pas+19] Adam Paszke u. a. „PyTorch: An Imperative Style, High-Performance Deep Learning Library“. In: *Advances in Neural Information Processing Systems 32*. Hrsg. von H. Wallach u. a. Curran Associates, Inc., 2019, S. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[Pit22] Radoslav Pitoňák. *Y2K22 BUG: ERROR: [BD 5-390] IP definition not found for VLNV*. 2022. URL: https://github.com/Xilinx/finn/discussions/483 (besucht am 06. 02. 2022).

[Red+16] Joseph Redmon u. a. „You Only Look Once: Unified, Real-Time Object Detection“. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Juni 2016, S. 779–788. DOI: 10.1109/CVPR.2016.91.

[Red14] Joseph Redmon. *Darknet*. Dez. 2014. URL: https://github.com/pjreddie/darknet (besucht am 13. 12. 2021).

[Red16] Joseph Redmon. *YOLO: Real Time Object Detection*. 2016. URL: https://github.com/pjreddie/darknet/wiki/YOLO:-Real-Time-Object-Detection (besucht am 08. 12. 2021).

[RF18] Joseph Redmon und Ali Farhadi. „YOLOv3: An Incremental Improvement“. In: *arXiv* (Apr. 2018). URL: https://arxiv.org/abs/1804.02767v1 (besucht am 13. 12. 2021).

[Roo21] Martijn Cornelis Bernardus de Rooij. *Ultra low latency deep neural network inference for gravitational waves interferometer Improvement*. März 2021. URL: https://repository.tudelft.nl/islandora/object/uuid:0496ebc1-d295-4c89-858a-06f84209c0da?collection=education (besucht am 13. 12. 2021).

[Ros18] Adrian Rosebrock. *YOLO object detection with OpenCV*. en-US. Nov. 2018. URL: https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/ (besucht am 08. 11. 2021).

[RW08] Günter Daniel Rey und Karl F. Wender. *Neuronale Netze Eine Einführung in die Grundlagen, Anwendungen und Datenauswertung*. 2008. ISBN: 978-3-456-84513-5.

[SV20] Purvesh Sharma und Damian Valles. „Backbone Neural Network Design of Single Shot Detector from RGB-D Images for Object Detection“. In: *2020 11th IEEE Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*. 2020, S. 0112–0117. DOI: 10.1109/UEMCON51285.2020.9298175.

[Wir21] Isabella Wireoky. *Data scines and all*. Apr. 2021. URL: https://about.att.com/sites/labs_research/ai (besucht am 08. 01. 2022).

[Wu16] Jia-Nan Wu. *Compression of fully-connected layer in neural network*. 2016. ISBN: 978-1-4673-7782-9.

[YB20] Zhewen Yu und Christos-Savvas Bouganis. „A Parameterisable FPGA-Tailored Architecture for YOLOv3-Tiny“. en. In: *Applied Reconfigurable Computing. Architectures, Tools, and Applications*. Hrsg. von Fernando Rincón u. a. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, S. 330–344. ISBN: 978-3-030-44534-8. DOI: 10.1007/978-3-030-44534-8_25.

A Abbildungsverzeichnis

1 Berechnung Max-Pooling [Roo21] . . . 7
2 Berechnung Average-Pooling [Alt+21] . . . 8
3 Berechnung des Convolutional Layers [Roo21] . . . 9
4 Filtertypen Convolutional Layer [Aus18] . . . 10
5 Aktivierungsfunktionen [Aus18] . . . 11
6 Fully Connected Layer [Wu16] . . . 12
7 Training Iteration . . . 13
8 Aufbau CNV [Roo21] . . . 14
9 Die Verarbeitungsschritte von YOLO [Red16] . . . 15
10 Aufbau YOLO V1 [Red+16] . . . 16
11 Vergleich des Aufbaus von SSD und YOLO V1 [Liu+15] . . . 18
12 FINN Stack . . . 20
13 System Architektur eines tinyYolo auf FPGA Ansatzes [YB20, S. 335] . . . 22
14 Konzept zur Verarbeitung innerhalb des Systems . . . 25
15 Klein abgebildete Personen bei schlechten Lichtverhältnissen . . . 26
16 Arme von Person aus Egoperspektive geschossen . . . 27
17 Verzeichnisstruktur . . . 28
18 Prinzip des Tilings der Person-Tiles (KS = kürzere Seite, LS = längere Seite) . . . 30
19 GCP Dashboard . . . 47
20 Billing-Eintrag . . . 47
21 Billing-Übersicht . . . 47
22 AI Platform Übersicht . . . 48
23 Neues Notebook erstellen . . . 48
24 Vereinfachte Konfigurationsübersicht . . . 49
25 Fortgeschrittene Konfigurationsübersicht . . . 49
26 Notebook übersicht . . . 50
27 Jupyterlab Übersicht . . . 50

B Tabellenverzeichnis

1 Vergleich von verschiedenen YOLO V3 Versionen [RF18] 17
2 Ressourcenverbrauch laut Vivado Summary für Neuronales Netz zur Personenerkennung 35
3 Ressourcenverbrauch für Hardware-Tiling nach [Mül21] 36
4 Theoretischer Ressourcenverbrauch für Personenerkennung inklusive Hardware-Tiling nach [Mül21] 36

C Anleitung: Training auf Google Cloud Platform

Geschrieben von Ismail Sastim

Das Training eines neuronalen Netzes erweist sich als sehr zeit- und ressourcenaufwändig. Abhilfe kann Cloud Computing schaffen. Hierfür stehen Studenten der Hochschule Bremen ein 50€ Guthaben für Google Cloud Platform zur Verfügung.
Über den Link `https://gcp.secure.force.com/GCPEDU?cid=ESFLrgVcZWg1S4rZvLPn9epvYN2cX14qVSjVUtXe2purxQpA95eb4URO61xczXB3` kann der Gutschein eingelöst werden. Es muss lediglich dem selbsterklärendem Anmeldeprozess gefolgt werden. Es sei angemerkt, dass vorher sich mit einem persönlichem Google Account angemeldet werden muss, damit der Gutschein diesem gutgeschrieben werden kann. Das Dashboard auf `https://console.cloud.google.com/` sollte am Ende folgendermaßen aussehen:

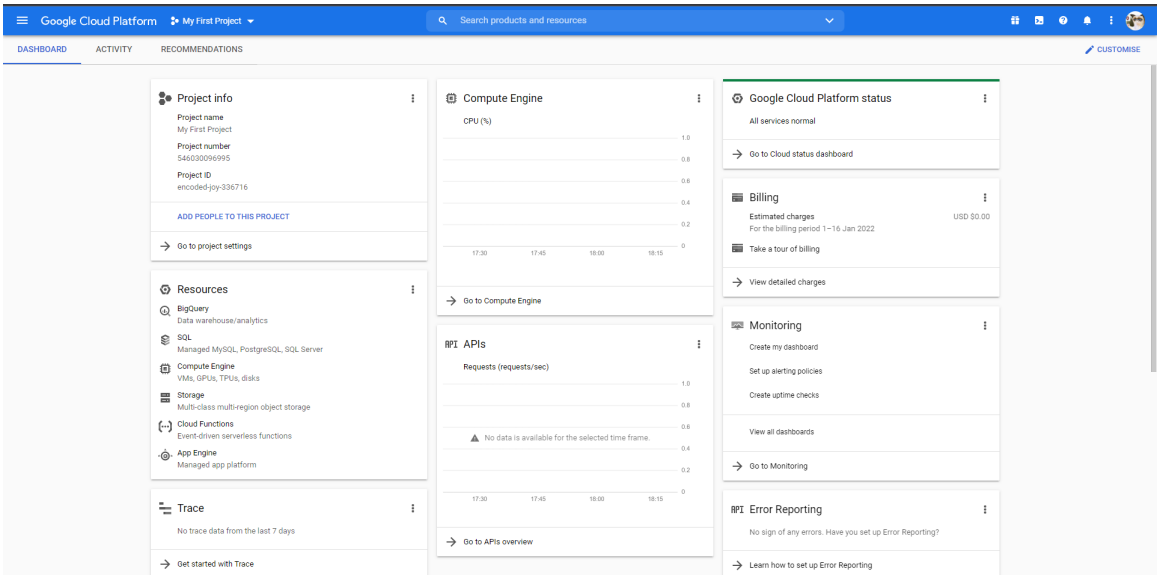


Abbildung 19: GCP Dashboard

Das verbleibende Guthaben kann durchs Klicken des *"Billing"*-Eintrags im Menü links eingesehen werden.

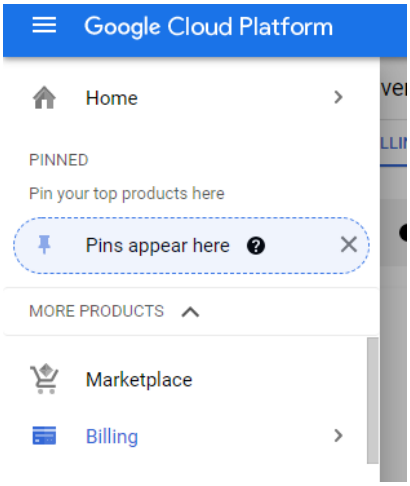


Abbildung 20: Billing-Eintrag

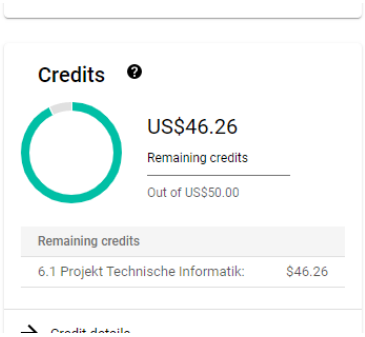


Abbildung 21: Billing-Übersicht

Um das Netz zu trainieren kann über die Suchfunktion nach *"AI Platform"* gesucht werden. Nach Bestätigung das Eintrags gelangt man auf folgende Oberfläche:

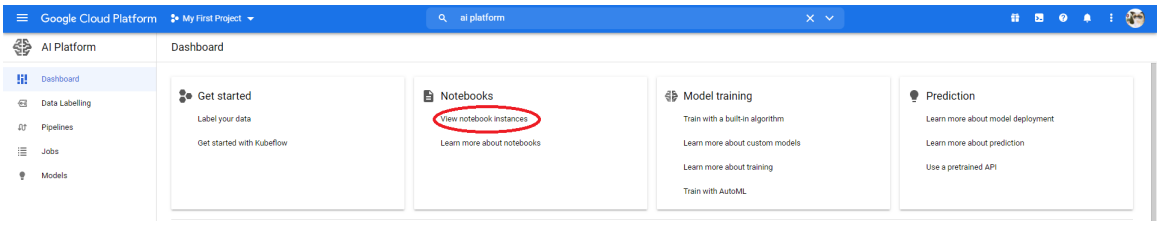


Abbildung 22: AI Platform Übersicht

Hinweis: Falls im Laufe der Anleitung eine Aufforderungen erscheinen, eine API zu aktivieren, sollte dem befolgt werden.

Hier muss auf *"View notebook instances"* angeklickt werden. Als nächstes muss ein neues Notebook erstellt werden. Hierfür auf den entsprechenden Button klicken und *"Python 3"* als Vorlage auswählen.

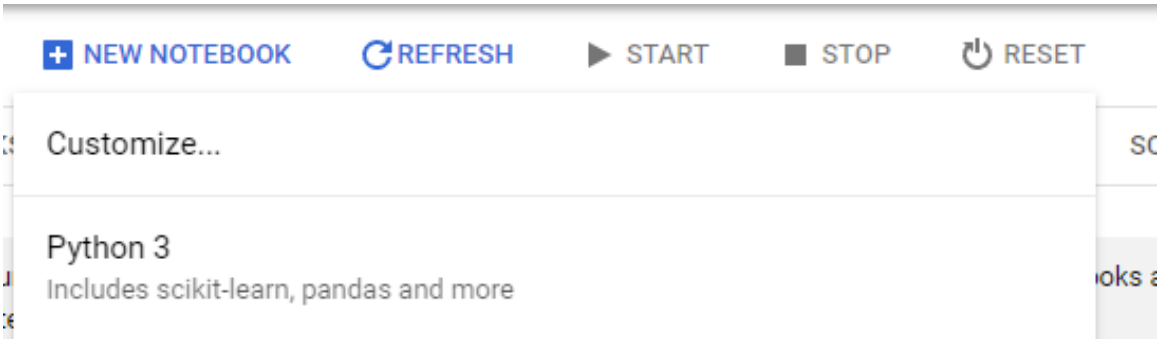


Abbildung 23: Neues Notebook erstellen

Im folgendem Fenster kann man Grobeinstellungen setzen. Es ist jedoch ratsam auf *"Advanced Options"* zu klicken.

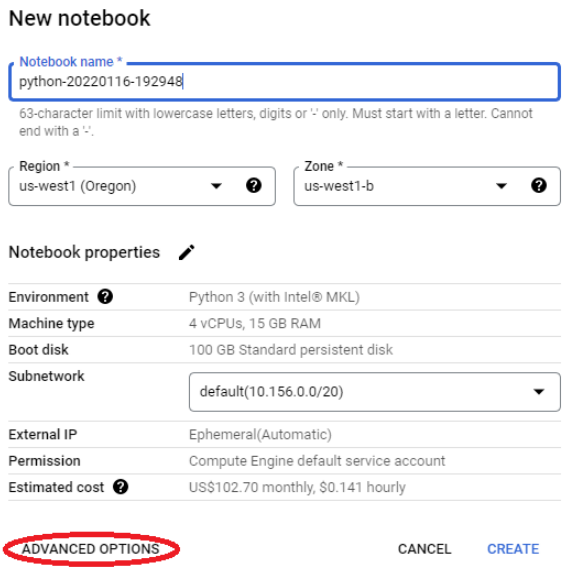


Abbildung 24: Vereinfachte Konfigurationsübersicht

Die Einstellungen sollten folgendermaßen aussehen:

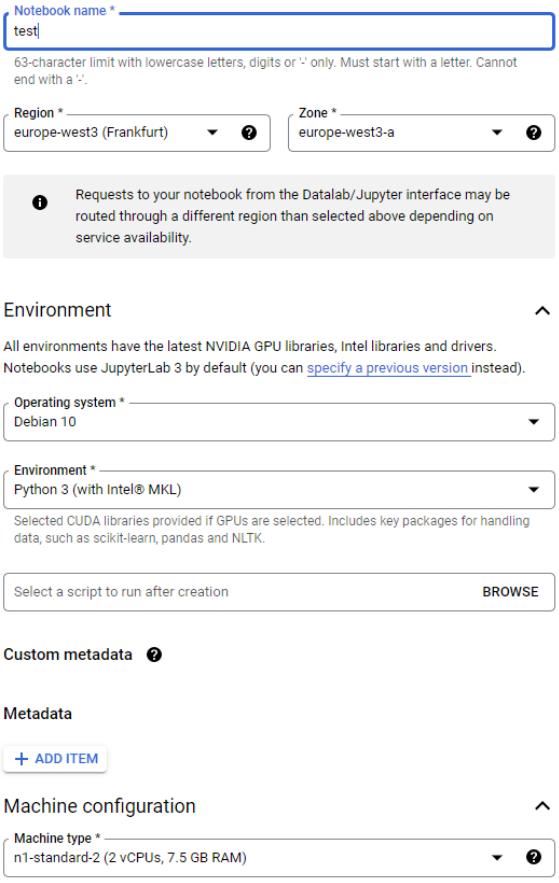


Abbildung 25: Fortgeschrittene Konfigurationsübersicht

Nach Bestätigung sollte ein Debian-System in der Übersicht zur Verfügung stehen.

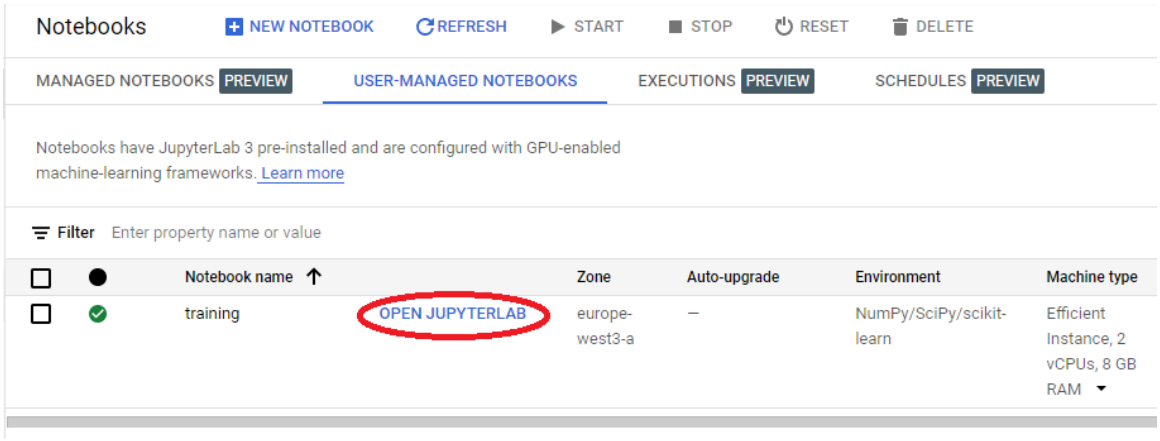


Abbildung 26: Notebook übersicht

Die Instanzen können durch die Auswahlkästchen angewählt und durch die obrigen Buttons gestartet und gestoppt werden. Insbesondere sollten die Instanzen gestoppt werden, wenn sie nicht mehr benötigt werden, damit keine unnötigen Kosten anfallen.
Um auf die Cloud-Maschine zugreifen zu können muss lediglich auf *"OPEN JUPYTERLAB"* angeklickt werden. Dies führt zu einer neuen Oberfläche:

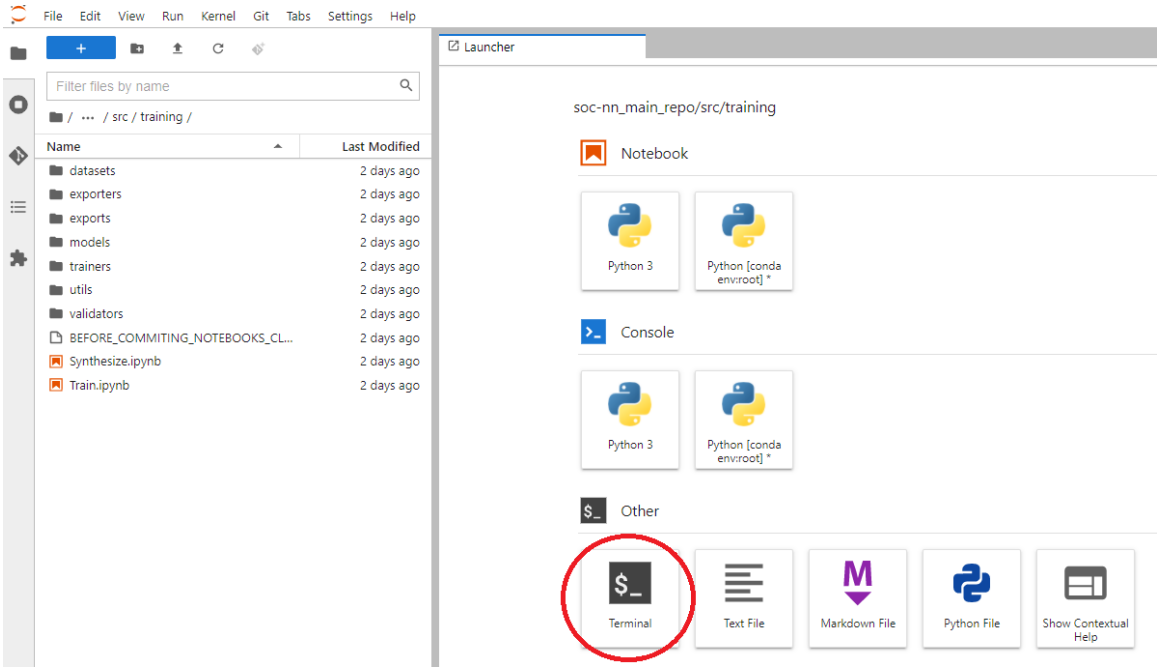


Abbildung 27: Jupyterlab Übersicht

Links ist ein File Browser zu finden, worüber man Notebooks, also Dateien mit *".ipynb"* Endung, öffnen kann. Zuerst müssen jedoch unsere Repositorys geklont und diverse Terminalkommandos ausgeführt werden. Daher sollte zunächst das entsprechende Button angeklickt werden um ein Terminalfenster zu öffnen. Ab hier kann der normale Workflow durchgeführt werden.

D Anleitung: Vorgehen mit dem COCO Datensatz

Geschrieben von Johannes Steffen

Vorgehen und Anleitung zum Trainieren

Dieses Dokument soll beschreiben wie ein Datensatz zum Trainieren eines Neuronalen Netzwerkes zur Erkennung von Personen gefiltert und aufbereitet wird.

Auswahl der Bilder (Johannes)

Eingabe:

- COCO Datensatz
- COCO Annotations

1. Lightroom benutzen zum Auswählen (gespeichert im .lrcat)
2. Ausgewählte Bilder mit Lightroom exportieren

Zwischenergebnis:

- Ein Ordner mit den ausgewählten Bildern (1000124.jpg ...)

3. Python Script zum generieren einer TXT Datei mit zu verwendenden Bildern

Ausgabe:

- TXT Datei
- Ordner mit ausgewählten Bildern (nur lokal)

Generierung eines Trainingsdatensatz (Mattia)

Eingabe:

- TXT Datei mit allen ausgewählten Bildern (1000124.jpg ...)
- Einen Ordner mit den ausgewählten Bilder (nur lokal)
- COCO Annotations

1. Exportiere alle nicht Personen in einen Ordner und sortiere sie in Training und Validierungs ordner (filter die txt datei)
2. Schneide Personen aus (+10-20% margin)
3. Exportiere Personen Bilder in entsprechender Ordner Struktur
4. Bei mehreren Personen wird jede einzeln exportiert

Ausgabe (soll als ZIP ins GitLab):

- Ordner mit:
    - noperson
    - person

Tiling der Dateien (Phillip & Ismail)

- Trainingsdatensatz (siehe Format oben)

1. Alle nicht Personen Bilder werden auf 6*4 (änderbar) Tiles runtergebrochen

2. Alle Personen Bilder werden je nach Seitenverhältniss in Tiles aufgeteilt.

Ausgang:

- Ordner mit quadratischen Tiles von person und noperson

E Anleitung: Inbetriebnahme Pynq-Z1 via ad-hoc Netz

Geschrieben von Philipp Altnickel

Erweiterung der Anleitung zur Inbetriebnahme aus dem Projektbericht de SoSe2021 (Anleitung I, S.130 ff.). Einige Details können dieser alten Anleitung entnommen werden.

Die folgenden Schritte wurden unter Linux Mint 20.2 Cinnamon getestet. Bei anderen Versionen sind möglicherweise noch einige Pakete zusätzlich zu installieren.

Download und Installation der Projektumgebung

Folgende Pakete müssen installiert werden, damit sowohl die Basestation, als auch das Processign System lauffähig sind:

```
sudo apt install docker.io
sudo apt install rsync
sudo apt install build-essential
sudo apt install libsdl2-ttf-dev
sudo apt install libsdl2-dev
```

Die beiden Repositories vehicleps und vehiclebasestation werden benötigt, um das Fahrzeug zum Laufen zu bringen:

```
git clone https://gitlab.com/soc-nn/vehicleps
git clone https://gitlab.com/soc-nn/vehiclebasestation
```

Ad-hoc Netz erstellen

Um ein Netz zu erstellen, zu welchem sich das PYNQ Board verbinden, kann folgender Befehl aus dem Verzeichnis der Basestation verwendet werden:

```
./network.sh start [interface-name]
```

Möglicherweise muss der Name des zu verwendenden WiFi Interfaces angepasst werden. Die Interfaces können via folgendem Befehl aufgelistet werden:

```
ifconfig
```

Zu beachten ist, dass nun über dieses Interface keine Verbindung zu einem anderen WLAN-Netz hergestellt werden kann. Sollte dies gewünscht sein, muss entweder eine kabelgebundene Verbindung oder ein zweiter WLAN Adapter genutzt werden.

Beendet kann der Acess-Point wie folgt werden:

```
./network.sh stop [interface-name]
```

PYNQ-Z1 verbinden

War das Board zuletzt bereits mit einem ad-hoc Netzwerk verbunden, können die folgenden Schritte übersprungen werden. Das Board sollte nun bereits eine Verbindung zum Rechner aufbauen.

Um die Netzwerkeinstellungen des Boards via ssh zu ändern, muss zunächst eine kabelgebundene Verbindung in ein bestehendes Netzwerk hergestellt werden. Um nun ein WLAN zu konfigurieren, sollte nach Eingabe von

```
ifconfig
```

ein Eintrag mit wlan0 erscheinen. Falls dem nicht so ist, muss das WLAN gestartet werden:

```
sudo ifconfig wlan0 up
```

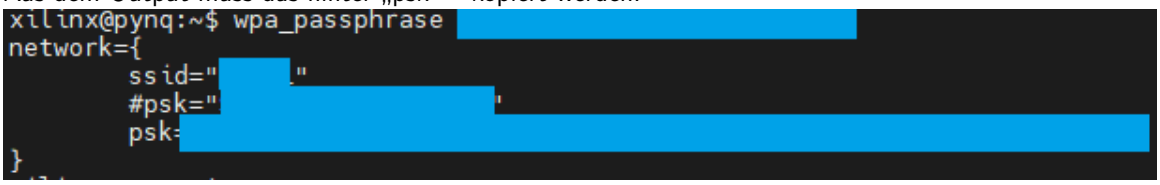
Anschließend kann man zwecks Verifizierung nach den vorhandenen WLAN-Netzen suchen:

```
sudo iwlist wlan0 s | grep socnn-network
```

Zur Automatischen Verbindung wird der WPA-Passkey benötigt:

```
wpa_passphrase socnn-network socnn2021
```

Aus dem Output muss das hinter „psk=“ kopiert werden:



Anschließend wird die Interface-Konfiguration geöffnet:

```
sudo nano /etc/network/interfaces.d/wlan0
```

Dort wird folgendes eingetragen, wobei die SSID und der psk ergänzt werden muss:

```
auto wlan0
iface wlan0 inet dhcp
wpa-ssid
wpa-psk
```

Nach einem Neustart sollte das PYNQ sich mit dem ad-hoc Netz verbinden:

```
sudo reboot
```

Basestation und Fahrzeug starten

Um die Basestation zu starten, kann folgender Befehl auf den entsprechenden vehiclebasestation-Verzeichnis genutzt werden:

```
make run
```

Mittels folgendem Befehl kann das kompilierte Programm auf dem Fahrzeug gestartet werden. In der *vehicle.conf* sollte die *BASE_STATION_IP* auf die IP des Rechners im ad-hoc Netz gesetzt werden.

```
./run.sh run *IP des PYNQ im ad-hoc NDer neue Datensatz, der für jeden
↪  verarbeiteten Frame vom Fahrzeug an die Base Station gesendet wird, sieht nun
↪  folgendermaßen aus (Die neuen Daten sind fett markiert):

etz*
```

Die IP Adressen, sowohl der Basestation, als auch des PYNQ-Boards, können wie folgt aus dem Verzeichnis der Basestation ermittelt werden:

```
./network.sh list [interface-name]
```

Genauere Details zum Cross-Compiler können aus der alten Anleitung entnommen werden. Diese gelten weiterhin.

F Testszenario zur Überprüfung der Projektziele

Zu jedem Augenblick soll sich maximal eine Person im Sichtfeld des autonomen Fahrzeug befinden.

- Das Fahrzeug wird auf einer ebenen Fläche (zum Beispiel Parkplatz) platziert
- Das Fahrzeug wird angeschaltet und mit der Base Station auf einem Laptop via Wifi verbunden
- Das Fahrzeug soll nach der Initialisierung mit dem Suchen beginnen und dieses an die Base Station melden (zu diesem Zeitpunkt soll keine Person im Blickfeld des autonomen Fahrzeugs sein)
- Eine Person soll sich außerhalb des Blickfeldes des autonomen Autos aufstellen (erreichbar mit der Kamera; also nicht hinter Gegenständen verstecken)
- Das Autonome Fahrzeug (im Suchmodus) soll die Person innerhalb von 30 Sekunden erkennen (Person befindet sich in „NOPOSE“ - siehe Datensatz [Alt+21, S. 108])
- Das Fahrzeug soll, wenn die Person im Blickfeld ist, dieses der Bodenstation mitteilen.
- Das Fahrzeug soll so lange sich orientieren, sodass die erkannte Person mittig im Bild zu sehen ist.
- Die Person soll sich schnell aus dem Blickfeld des autonomen Fahrzeugs bewegen.
- Das Fahrzeug soll wieder in den Suchmodus übergehen und dieses der Bodenstation mitteilen
- Nachdem das Fahrzeug die Person („NOPOSE“) wieder gefunden hat soll die Person die SSTART"Pose einnehmen
- das Fahrzeug soll diese „START“ Pose erkennen
- das Fahrzeug soll nun die Person verfolgen, dies kann durch Bewegungen in Richtung der Person verifiziert werden. Dabei soll die Person immer einen Abstand von mindestens 5 Metern und maximal 10 Metern einhalten und sich im Blickfeld des Fahrzeugs aufhalten.
- Die Person soll nach 30 Sekunden Verfolgung die „STOP“ Pose einnehmen
- Das Fahrzeug soll diese „STOP“ Pose erkennen und dementsprechend stoppen
- Die Person soll wieder die „START“ Pose einnehmen
- Das Fahrzeug soll daraufhin die Person verfolgen
- Die Person soll das Sichtfeld der Kamera verlassen
- Das Fahrzeug soll ab dem Moment wo die Person sich außerhalb des Sichtbereiches befindet stoppen
- Das Fahrzeug soll in den Suchmodus übergehen

G Testszenario zur Überprüfung der Teilprojektziele der Personenerkennung

Zu jedem Augenblick soll sich maximal eine Person im Sichtfeld des autonomen Fahrzeug befinden.

- Das Fahrzeug wird auf einer ebenen Fläche (zum Beispiel Parkplatz) platziert
- Das Fahrzeug wird angeschaltet und mit der Base Station auf einem Laptop via WiFi verbunden
- Das Fahrzeug soll nach der Initialisierung mit dem Erkennen von Personen beginnen und dieses an die Base Station melden (zu diesem Zeitpunkt soll keine Person im Blickfeld des autonomen Fahrzeugs sein)
- Die Person soll sich dem Fahrzeug nähern.
- Das Fahrzeug (im Suchmodus) soll die Person erkennen
- Das Fahrzeug soll nun die Person verfolgen, dies kann durch Bewegungen in Richtung der Person verifiziert werden. Dabei soll die Person immer einen Abstand von mindestens 1 Metern und maximal 5 Metern einhalten und sich im Blickfeld des Fahrzeugs aufhalten.
- Die Person soll sich nach eigenem ermessen in verschiedene Richtungen relativ zum Fahrzeug bewegen und das Fahrzeug zum Fahren von engen Kurven bewegen.