HSB
Hochschule Bremen
City University of Applied Sciences

Hochschule Bremen

Fakultät 4: Elektrotechnik und Informatik

Konzeption und Entwicklung eines autonomen Vehikels mit Bild- und Sprachkommandoerkennung durch neuronale Netze auf einem weltraumtauglichen FPGA

Keno Albers, Milan Becker, Luca Bianchi, Julius Böttger, Jan Bredereke, Jonas Broda, Arian Dannemann, Frederic Goretzky, Jannik Hennicke, Philipp Hennken, Alexander Hoegen-Jupp, Tim Jaeschke, Tuanna Karadas, Nils Karsten, Jonas Kleimann, Simon Koger, Simon Köhler, Jonas Landwehr, Kevin Lingk, Matthias Löwen, Marvin Lünswilken, Kilian Müller, Niklas Otten, Enrico Ramm, Tim Ranft, Max Tepe, Nils van Rijsinge

Dozent: Prof. Dr. Jan Bredereke

14. Oktober 2025

Wir untersuchen, wie man neuronale Netze auf feldprogrammierbaren Gate-Arrays (FPGAs) am besten nutzen kann, wenn diese so leistungsbeschränkt wie strahlungsfeste FPGAs in der Raumfahrt sein müssen (und etwas allgemeiner entsprechend neuronale Netze auf FPGAs für Edge-Computing). Ein autonomes Modellauto als konkretes Vehikel dient uns dabei als Stellvertreter für ein autonomes Raumfahrzeug. An der Architektur, Hardware und Software dafür haben wir vieles verbessert. Darüber hinaus gibt eine Literaturrecherche zu Hardware-bedingten Rechenfehlern in neuronalen Netzen, motiviert durch die Problematik der Weltraumstrahlung, einen Überblick über die aktuelle Forschung zu schaltungstechnischen Maßnahmen zur Eindämmung der Fehlerwirkungen sowie zu Meßansätzen, um den Erfolg solcher Maßnahmen nachzuweisen. Und die experimentelle Untersuchung systematischer Fehler beim Erkennen von Sprachkommandos ergab Einsichten, worauf beim Training eines neuronalen Netzes dafür zu achten ist. Schließlich identifizieren wir interessante verbliebene Aufgaben und Herausforderungen im Kontext unserer Forschungsfrage.

Inhaltsverzeichnis

1. Einleitung

geschrieben von Jan Bredereke

Dies ist der Projektbericht zum Mini-Projekt in der Lehrveranstaltung „Eingebettete Systeme in der Praxis“ (ESYSP) an der Hochschule Bremen im Sommersemester 2025. Dieses Kapitel führt zunächst in den Kontext ein und präsentiert dann die Forschungsfrage. Anschließend stellt es die Beispielanwendung und einschlägige vorhergehende Arbeiten an der Hochschule Bremen vor. Das Kapitel schließt mit einem Überblick über die in der aktuellen Runde des Projekts behandelten Teilthemen.

1.1. Kontext

Die Motivation für das Projekt entstammt der Raumfahrt. Auf Bordrechnern ist besonders wenig Rechenleistung verfügbar, und eine Verbindung zu einer Bodenstation ist meist nur zeitweise gegeben. Aufgrund der hohen Belastung durch Weltraumstrahlung würden übliche heutige Prozessoren sehr schnell ausfallen. Daher verwendet man Spezialrechner, deren Chips Strukturbreiten von mindestens 65 nm aufweisen. Diese Spezialrechner sind robust genug, aber entsprechend weniger leistungsfähig als solche, die mit aktuellen 5 nm hergestellt sind.

Aufgrund der äußerst geringen Stückzahlen dieser Art von Rechnern werden sie oft nicht mit speziell entwickelten Chips (ASICs), sondern mit programmierbarer Standard-Hardware (FPGAs) realisiert. Strahlungsfeste Versionen bestimmter FPGAs mit entsprechend großer Strukturbreite sind hierfür verfügbar.

Zunehmend besteht Bedarf an mehr Onboard-Rechenleistung, zum Beispiel für Bildverarbeitung an Bord, etwa für autonome Rover auf anderen Himmelskörpern oder für Schwärme von Kleinsatelliten mit jeweils nur wenig Bandbreite zu einer Bodenstation.

Insbesondere möchte man gerne neuronale Netze nutzen, die üblicherweise in Rechenzentren mit leistungstarker und stromhungriger Spezialhardware eingesetzt werden. Sie können allerdings nicht einfach am Rande der Cloud („Edge“), d.h. nahe bei den Sensoren und Aktoren, oder gar autonom von Daten- und Stromversorgungsverbindungen, auf einem Mikrocontroller eingesetzt werden, denn die CPU eines Mikrocontrollers ist dafür zu rechenschwach.

Mehr Leistung, sowohl absolut als auch pro Stromverbrauch, verspricht der Einsatz eines feldprogrammierbaren Gate-Arrays (FPGA). Grundsätzlich ist ein FPGA für die hochparallele Struktur eines neuronalen Netzes sehr gut geeignet. In der Praxis ergeben sich aber viele Optimierungsaufgaben, die erst gelöst werden müssen, bevor das FPGA diesen Vorteil voll ausspielen kann.

1.2. Forschungsfrage

Unsere Forschungsfrage ist, wie man neuronale Netze auf FPGAs am besten nutzen kann, die so leistungsbeschränkt wie strahlungsfeste FPGAs sind. Etwas allgemeiner stellt sich diese Frage entsprechend für neuronale Netze auf FPGAs für Edge-Computing.

1.3. Die Beispielanwendung

Ein autonomes Modellauto als konkretes Vehikel (siehe Abb. 1.1) dient uns als Stellvertreter für ein autonomes Raumfahrzeug. Es ist mit einer Kamera, einem Mikrofon und einem SoC Zync-7020 ausgerüstet. Das SoC enthält außer einer Arm-CPU auch ein FPGA Artix-7, dessen Leistungsfähigkeit recht gut einem weltraumtauglichen FPGA entspricht. Das SoC ist auf einem PYNQ-Z2 Board verbaut.

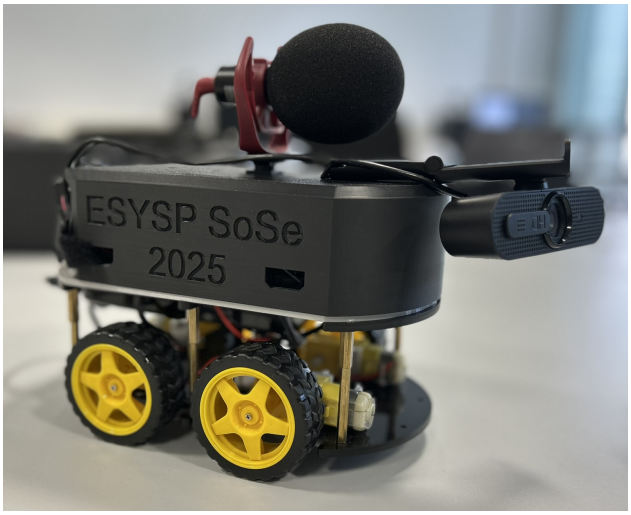


Abbildung 1.1.: Das Fahrzeug der Lehrveranstaltung ESYSP im SoSe 2025.
Foto: Frederic Goretzky

1.4. Vorhergehende Arbeiten an der Hochschule Bremen

1.4.1. Bilderkennung

Die Lehrveranstaltung „Projekt: SoC-NN – FPGAs für neuronale Netze: Edge Computing auf autonomen Vehikeln“ im Wintersemester 2021/22 sowie einige Lehrveranstaltungen davor [ACB+22; ACB+21; Mül21; MKB+21; HSB+20] realisierten eine Bilderkennung: Das Fahrzeug soll autonom und in Echtzeit eine Person in ihrem Blickfeld erkennen, dieser Person folgen, wenn sie sich bewegt, und Fahrkommandos durch einfache Gesten gehorchen.

Der Konferenzbeitrag [Bre23] fasst die Ergebnisse dieser Arbeiten zusammen. Das Erkennen und Verfolgen einer Person durch das Fahrzeug wurde erfolgreich realisiert. Die Gestenerkennung wurde ebenfalls realisiert, konnte aus Zeitgründen allerdings nicht mehr in das Fahrzeug integriert werden. Das Zerlegen eines Bildes eines Videostroms in einzelne Kacheln, die klein genug für eine Analyse durch das neuronale Netz auf einem FPGA sind, wurde von Müller in seiner Abschlussarbeit auf einem FPGA [Mül21] effizient und flexibel implementiert; die Integration in das Gesamtsystem steht jedoch noch aus.

1.4.2. Sprachkommandoerkennung

Die Lehrveranstaltung „Embedded Systems“ im Wintersemester 2021/22 [HHB+22] begann im Rahmen des zugehörigen Mini-Projekts, auf dem selben Vehikel eine Fahrsteuerung durch Sprachkommandos statt durch Gesten zu realisieren, ebenfalls autonom und in Echtzeit. Damit stiegen wir neu in den Bereich der akustischen Signale ein. In diesem Gebiet gibt es weniger fertige Lösungen für neuronale Netze als für die Bilderkennung. Dieses Mini-Projekt erstellte ein Konzept, und es produzierte Trainingsdaten, es kam aber auf Grund der beschränkten Zeit noch nicht weiter.

Die Lehrveranstaltung „Embedded Systems“ im Wintersemester 2022/23 [BJB+23] begann mit der Umsetzung. Hardware-Grundlage war weiterhin das FPGA-Board PYNQ-Z1. Die Systemarchitektur wurde, aufbauend auf dem obigen Vorgängerprojekt, detailliert ausgearbeitet. Ein Teil der Audiodatenverarbeitung wurde auf dem FPGA realisiert. Die Qualität von Trainingsdaten, die aus dem Vorgängerprojekt übernommen wurden, wurde verbessert, und es wurde ein systematisches Problem beim Aufnehmen von Trainingsdaten identifiziert und behoben. Es wurde die Möglichkeit geschaffen, einfacher verschiedene Strukturen für das neuronale Netz zu erstellen, um damit zu experimentieren. Es wurden bereits viele Schritte für die Implementierung des neuronalen Netzes auf dem FPGA ausgearbeitet; die Integration mit der Audiodatenverarbeitung konnte allerdings nicht mehr realisiert werden. Erste Zeitmessungen zeigten, dass der Einsatz eines quantisierten neuronalen Netzes die Verwendung von Fließkommazahlen bei der Inferenz überflüssig macht, welche ohne eine leistungshungrige Hardware nicht verfügbar sind. Zur Erkennungsrate für Sprachkommandos auf dem FPGA lagen bei Projektende noch keine aussagekräftigen Daten vor.

Die Lehrveranstaltung „Embedded Systems“ im Wintersemester 2024/25 [ABB+25] setzte die Arbeit fort. Es wurden die nötigen Verarbeitungsschritte an neue Audio-Hardware angepasst sowie detaillierter identifiziert und festgelegt. Aus einem aufgetretenen Schnittstellenproblem heraus ergab sich eine neue Einsicht zu einer Systemarchitektur für eine effiziente Verarbeitung auf einem FPGA.

1.5. Behandelte Teilthemen in der aktuellen Runde des Projekts

In der aktuellen Runde des Projekts werden die oben beschriebenen angefangenen Arbeiten weiter vorangetrieben. Dabei werden sowohl die Bild- als auch die Audioverarbeitung unter den Randbedingungen der schwachen Hardware verbessert, um das Verständnis für die Optimierungsmöglichkeiten zu vertiefen. Unterstützend wird die verwendete Plattform ebenfalls verbessert. Schließlich wird als neues Teilthema ein Verständnis für die Eigenschaften von neuronalen Netzen auf Hardware unter Weltraumstrahlung aufgebaut.

Im einzelnen werden in diesem Bericht die folgenden Teilthemen behandelt:

- Es wird ein Überblick über die geplante Gesamt-Systemarchitektur erarbeitet, um sowohl die Komponenten als auch die Schnittstellen zwischen ihnen zu definieren.
- Bei der Personenerkennung wird analysiert, warum die bisherigen Ergebnisse des Testskripts nicht zum Ergebnis des End-to-End Tests passen. Zusätzlich wird das Trainings- und Testskript verbessert, sowie ein objektiv beurteilbarer Test für die Personenerkennung erstellt.
- Die Toolchain für die hardwarebeschleunigte Gestenerkennung auf dem PYNQ-Z2 Board wird reevaluiert.

- Beim neuronale Netz für die Spracherkennung werden mögliche systematische Fehler analysiert und ausgeschlossen bzw. behoben.
- Die Fahrsteuerung wird analysiert, modularisiert und genauer gemacht.
- Die Elektronik und Mechanik der Fahrzeuge wird verbessert.
- Es wird Forschungsliteratur recherchiert, die die Auswirkungen von Rechenfehlern in einem neuronalen Netz abschätzt. Darüber hinaus werden sowohl Härtungsmaßnahmen zum Schutz vor Rechenfehlern als auch systematische Evaluationsmethoden zur Bewertung der Fehlertoleranz behandelt. Dies ist der Einstieg in ein weitergehendes Thema im selben Kontext von FPGAs für Weltraumanwendungen.

2. Überblick über die Systemarchitektur

geschrieben von Enrico Ramm und Luca Bianchi

Ziel dieses Kapitels ist es, einen einheitlichen und sinnvollen Gesamtüberblick über die Systemarchitektur des Projekts zu geben. Gerade in einem technisch breit gefächerten Projekt wie diesem ist eine übergreifende Systembeschreibung essenziell, um die Funktionsweise des Gesamtsystems zu verstehen und die Zusammenarbeit zwischen den beteiligten Gruppen zu koordinieren. Daher wird an zukünftige Gruppen appelliert, dieses Kapitel auch in den Folgeberichten fortzuführen und ggf. um detailliertere Schnittstellendetails zu erweitern.

In den letzten Projekten aus den Wintersemestern 2023/2024 [BJB+23] 2022/2023 [HHB+22] 2021/2022 [MKB+21] lag der Fokus stark auf Funktionen wie die Sprachkommandoerkennung, jedoch sind Aspekte wie die gemeinsame Nutzung bzw. Integration dieser innerhalb des Gesamtsystems nicht näher behandelt worden.

In diesem Kapitel wird zuerst ein Überblick über die bislang erreichten Ergebnisse im Kontext einer definierten Gesamtarchitektur erarbeitet und beschrieben (siehe Abschnitt 2.1).

Anschließend wird im Abschnitt 2.3 das in diesem Projekt geplante Architekturkonzept erläutert, indem die Hauptfunktionen der wichtigsten Teilsysteme zusammengefasst und deren zentrale Schnittstellen definiert werden. Die Komponenten des Projekts werden aus unterschiedlichen Perspektiven beschrieben, sowohl aus logischer Sicht als auch unter datenflussorientierten Gesichtspunkten. Außerdem wird erwähnt, welche Teile der geplanten Architektur bereits vorhanden sind.

2.1. Vorherige Ergebnisse

Wie zu Beginn dieses Berichtes erwähnt, baut das vorherige autonome System auf den PYNQ Z1 Board auf. Im Kern des PYNQ Z1 Boards arbeitet ein SoC (System on a chip) vom Modell Zynq-7020, welcher in PS (Processing System) und PL (Programmable Logic) unterteilt werden kann. Zum PS gehören zum einen ein ARM Cortex A9 Prozessor, der via Schnittstellen Zugriff auf den Arbeitsspeicher, Peripherie und I/O, sowie den FPGA (PL) Teil des SoCs besitzt. Der PL ist der FPGA-Teil des SoCs, wodurch dieser sich individuell programmieren lässt, um schnelle Hardware-Logik ausführen zu können.

Die Gesamtarchitektur der vorherigen Projekte lässt sich als mehrstufige, eng gekoppelte Signal- und Datenverarbeitungskette beschreiben, die von der Kamera-Ebene bis zur Motoransteuerung reicht (vgl. Abbildung 2.1). Um die Hardware-Beschleunigung auf dem Board zu nutzen, werden unterschiedliche Module entweder auf dem PS (hier blau markiert) oder dem PL (hier orange markiert) ausgeführt.

Im Folgenden werden die wesentlichen Verarbeitungsschritte erläutert.

Kamera-Akquisition und Übertragung Das Fahrzeug führt eine USB-Kamera, deren Rohdaten in der Kamera-Verbindung zunächst mittels Video4Linux-Treibern auf dem ARM-Teil des Zynq-7020 SoC empfangen werden. Das resultierende RGB-Bild dient als einziger Sensoreingang für die Folgeketten.

Personspezifische Bildvorverarbeitung Innerhalb des Personenerkennungs-Moduls wird das Kamerabild zuerst skaliert und in überlappende Tiles zerlegt. Das Sliding-Window-Verfahren reduziert dabei den Eingaberaum so weit, dass jedes Tile einer Person potenziell objektfüllend entspricht, ohne die hohe globale Auflösung beibehalten zu müssen. Auf dem PYNQ-Z1 führt ein dedizierter TilingController diese Aufgabe aus.

Positionsbestimmung Die Konfidenz-Map aller Tiles wird softwareseitig aggregiert. Ein Maximum-Suche-Algorithmus bestimmt das Tile-Zentrum mit der höchsten Wahrscheinlichkeit und gibt dessen Pixelkoordinaten als Person Position aus. Diese Verdichtung reduziert den Datenstrom auf wenige Bytes pro Bild und übergibt ein abstraktes Weltmodell an die nachfolgende Regelungsschicht. Im Abschnitt 3 wird genauer auf die Personenerkennung eingegangen.

Fahrzeugregelung und Motorsteuerung Die Fahrzeugsteuerung interpretiert die Position relativ zum Bildmittelpunkt als Regelgröße. Ein diskreter PID-Regler erzeugt Soll-Geschwindigkeiten für Lenk- und Fahrmotor, um die Person mittig und in konstanter Distanz zu halten. Alle Reglerparameter sind im Python-Teil des SoC adaptiv konfigurierbar und wurden empirisch auf das Chassis abgestimmt.

Das Motorsteuerungs-Modul setzt die Reglerausgaben in PWM-Signale für Heck- und Lenkservomotor um. Hierzu nutzt es ein leichtgewichtiges AXI-Lite-Interface, sodass die Steuerzyklen deterministisch innerhalb der Video-Frame-Zeit erfolgen können.

Telemetrie- und Debug-Pfad Parallel zum Echtzeitpfad überträgt das System jedes verarbeitete Frame-Resultat als DataFrame via WLAN an eine Basistation. Die Debug-Schnittstelle visualisiert Konfidenzkarten, Tiles und Reglergrößen in Jupyter-Notebooks, wodurch Parameter-Tuning und Fehlersuche ohne zusätzliche Logik auf dem Fahrzeug erfolgen können.

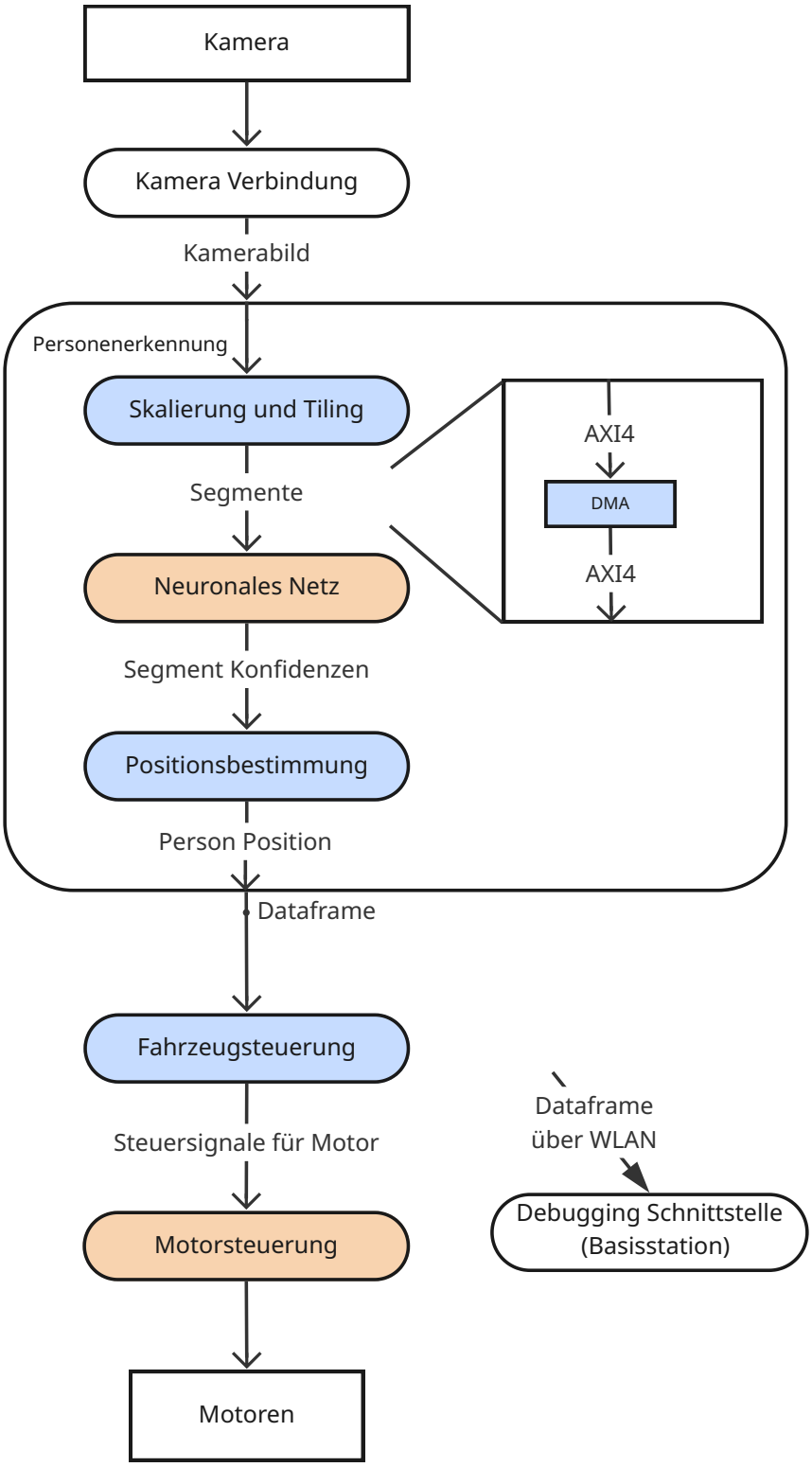


Abbildung 2.1.: Vorherige Architektur (Hauptmodule und Datenpfad)

2.2. Datenaustausch zwischen PS und PL

Aufgrund der Aufteilung der funktionalen Module zwischen PS und PL, stellt sich die Übertragung von Daten von der Software zur programmierten Hardware als sehr wichtig dar. Um den Kommunikationsweg genauer zu verstehen, wird der Datentransfer zwischen PS und PL (genauer

zwischen CPU und neuronalen Netz) in diesem Abschnitt beschrieben. Zudem ist es anzumerken, dass diese Themen bereits ausführlicher in den vorherigen Projekt-Berichten [HSB+20] und [ACB+21] erläutert wurden, weshalb hier Nötigste zusammengefasst wird.

2.2.1. AXI (Advanced Extensible Interface)

Das Advanced Extensible Interface oder kurz AXI, ist ein Protokoll-Standard, welcher Teil der von ARM gepflegten Protokoll-Familie AMBA (Advanced Microcontroller Bus Architecture) ist. Hierbei besitzt AXI4 die Untervarianten AXI4, AXI4-Lite und AXI4-Stream, welche leistungsstark sind und ein breites Spektrum an Datentransfer-Bedürfnissen abdecken (vgl. [Arm25]).

Im Kern des AXI-Protokolls kommunizieren Komponenten als Master (Manager) und Slave (Subordinate) miteinander, wobei der Master mittels Adresse Daten von einem Slave lesen oder an einen Slave senden kann. AXI definiert hierbei nur eine 1 zu 1 Verbindung, weshalb bei gleichzeitiger Kommunikation zwischen mehreren Master- und Slave-Komponenten eine AXI-Interconnect-Komponente die Verbindungen dieser steuert bzw. verwaltet. Eine genauere Beschreibung der AXI-Schnittstellen (AXI-Lite, AXI-Stream) kann im Kapitel 2.10 des Berichts [ACB+21] gefunden werden.

2.2.2. Datentransfer zwischen Processing System und neuronalen Netz

Der Datentransfer zwischen CPU und neuronalen Netz wird im Kapitel 9.1.3 des Projektberichts [ACB+21] als Zusammenspiel aus dem DMA (Direct Memory Access)-Speicher, AXI-Lite Bus zur Konfiguration und AXI-Stream zur schnellen Datenübertragung beschrieben. Nachdem die vorverarbeiteten Bild-Tiles von der CPU im externen DDR-RAM abgelegt wurden, werden diese über den DMA-Speicher abgegriffen, um anschließend an den IP-Block des neuronalen Netzes zur Analyse weitergeleitet zu werden. Genauer wird für das Einlesen der Bild-Tiles aus dem RAM ein AXI-Lite Bus verwendet, welcher die bestimmten Speicheradressen und die Größe der zu übertragenen Bilder über Konfigurationsregister setzt. Somit benötigt die CPU nur einige Taktzyklen zum Konfigurieren der Übertragung und überlässt die eigentliche Datenübertragung anschließend dem DMA-Controller, wodurch die CPU während des Transfers entlastet wird.

Nachdem das neuronale Netz die vorverarbeiteten Bilder analysiert hat, werden die aus den Tiles ergebenen Konfidenzergebnisse in einem weiteren DMA-Block (ODMA genannt) direkt geschrieben. Auch hierfür wird die Datenübertragung mittels AXI-Lite konfiguriert, indem die Ziel-Speicheradresse im RAM für den DMA-Block angegeben wird. Ein Statusregister teilt der CPU mit, wann ein Schreibvorgang (also der Durchlauf des NNs) fertig ist, um weiter mit den Konfidenzwerten arbeiten zu können.

2.2.3. Verwendete Hilfsquellen zum Verständnis der Projektarchitektur

Wie bereits erwähnt, wurde vor allem in den letzten Projektberichten der Überblick der Gesamtarchitektur als Konsequenz der spezialisierten Betrachtung von Teilaspekten eher vernachlässigt. Eine Folge dessen ist, dass viele Informationen im Kontext der Systembeschreibungen über alle Projektberichte verteilt sind. Um diesen entgegenzuwirken, wurde eine Auflistung der wichtigsten Systembeschreibungen aus den vorherigen Projekten zusammengefasst:

Folgende Tabelle listet alle für die Dokumentation der Gesamtbeschreibung verwendeten Hilfsquellen mit ihrer Herkunft und kurzer Beschreibung auf:

Zweck der Quelle	Quelle	Anmerkung/Kommentar
Architektur		
Internes Konzept für die interne Verarbeitung der Kamerasignale	[ACB+22, S. 25]	Dieser Abschnitt beschreibt die interne Bildverarbeitung mit neuronalen Netzen.
Überblick der Klassenbeziehungen zwischen Controllerklassen und Fahrzeugsteuerung	[ACB+21, S. 67]	Dieser Abschnitt beschreibt Anpassungen und das bis jetzt aktuell-laufende Design für die Fahrzeugsteuerung
Architekturkonzept für Sprachkommandoerkennung	[ABB+25, S. 17]	Dieser Abschnitt beschreibt ein Gesamtsystemkonzept für eine Sprachkommandoerkennung
Schnittstelle PS und PL		
AXI-Stream IP-Core & DMA-Loop-Back	[HSB+20, S. 39-41]	Zeigt ein exemplarisches AXI-Stream Projekt
Datenpfad (AXI-DMA) & Register-Mapping	[ACB+21, S. 67-70]	Gibt Aufschluss über den Datentransfer zwischen Processing System und neuronalem Netz

Tabelle 2.1.: Verwendete Quellen zur Dokumentation der vorherigen Projektarchitektur

2.3. Geplante Architektur

In Abbildung 2.2 auf Seite 12 ist die geplante Architektur des Projektes dargestellt. Die Architektur basiert sowohl auf Ansätzen vorheriger Gruppen, als auch neuen Ideen. Dies ist in den folgenden Unterabschnitten entsprechend markiert.

2.3.1. Personen- und Gestenerkennung

Die Personen- und Gestenerkennung sind getrennte Module, welche sich jedoch aufgrund ihrer ähnlichen Aufgabe das Untermodul zuständig für die Skalierung und das Tiling teilen. Dieses Untermodul erhält die Bilddaten von einer Kamera, welche über USB und der Kameraanbindung an das System angeschlossen sind. Das Skalierung und Tiling-Untermodul erstellt aus den Bilddaten Kacheln für der Personenerkennung und für die Gestenerkennung.

Die Kacheln für die Personenerkennung werden sequenziell in ein neuronales Netz gegeben. Dieses gibt einen Ausgabevektor mit Segment Konfidenzen für die Eingabe-Tiles aus. Die Segment-Konfidenzen werden genutzt, um mit einem weiteren Untermodul eine Region of Interest (ROI) zu bestimmen. Die ROI wird als Feedback verwendet, um sowohl für die Personen- als auch für die Gestenerkennung die Skalierung und das Tiling zu optimieren. Außerdem wird die ROI verwendet, um die Position der Person im Bild zu bestimmen. Die Position wird an das Modul der Fahrzeugsteuerung weitergegeben.

Die Kacheln für die Gestenerkennung werden in ein weiteres neuronales Netz gegeben. Dieses gibt einen Vektor mit Konfidenzen für einige vordefinierte Gesten aus. Anhand der Konfidenzen werden die verschiedenen Gesten bestimmt. Die jeweilige Geste wird an die Fahrzeugsteuerung weitergegeben.

Rechenintensive Aufgaben, wie die beiden neuronalen Netze und die Skalierung und das Tiling sollen auf dem FPGA (PL) laufen. Alle weiteren Aufgaben, die weniger rechenintensiv sind, sollten aufgrund der einfacheren Implementierung in Software zum Laufen auf der CPU realisiert werden.

Für das Tiling existiert ein TilingController aus [ACB+21] (siehe Abschnitt 3.3 und Abschnitt 4.2). Auch die Bestimmung der ROI wird in Abschnitt 4.2 beschrieben, jedoch konnte keine abschließende Integration auf dem PYNQ-Z2-Board erreicht werden. Ebenfalls ist die beschriebene Feedback-Schleife für die ROI noch nicht implementiert.

2.3.2. Spracherkennung

Die Spracherkennung erhält das Eingabesignal über ein Mikrofon und die zugehörige Anbindung. Zusätzliches wird das analoge Audiosignal vom AD-Wandler des PYNQ-Z2 digitalisiert. Der AD-Wandler muss vorher in Software konfiguriert werden.

Das digitalisierte Audiosignal wird an das Spracherkennungsmodul gegeben. Der Aufbau des Moduls wurde [ABB+25, S. 17] entnommen. Das kontinuierlich aufgenommene Audiosignal wird in einzelne Fenster aufgeteilt. Diese werden genutzt, um Mel Frequency Cepstral Coefficients (MFCCs) zu erstellen. MFCCs dienen zur kompakten Darstellung eines Frequenzspektrums und werden häufig zur Spracherkennung genutzt [Log+00]. Die MFCCs werden erneut in zeitliche Fenster aufgeteilt, bzw. systematisch zu größeren Fenstern zusammengefügt (z. B., um Fenster zu sammeln, bis ein Bereich von 2 Sekunden abgedeckt ist). Diese Fenster werden anschließend in ein weiteres neuronales Netz gegeben, welches das Audiosignal klassifiziert. Das Ergebnis der Klassifizierung wird zur Verarbeitung weitergegeben, um die gesprochenen Worte daraus zu identifizieren. Die erkannten Worte werden an die Fahrzeugsteuerung weiter gegeben.

Implementiert wurde in dieser Runde des Projekts eine verbesserte Erkennung einzelner Kommandos, jedoch bisher nur auf dem Laptop (siehe Kapitel 7). Ausstehend ist außerdem die Vervollständigung der beschriebenen Audio-Pipeline.

2.3.3. Fahrzeug- und Motorsteuerung

Die Fahrzeugsteuerung erhält folgende Eingabesignale:

- Von der Personenerkennung die Position der Person im Bild
- Von der Gestenerkennung die erkannte Geste
- Von der Spracherkennung die erkannten gesprochenen Worte

Anhand der Eingabesignale entscheidet die Fahzeugsteuerung welches Kommando an die Motorsteuerung gesendet wird. Die Umsetzung soll dabei einfach und intuitiv gestaltet sein. Z. B. wenn die Personenerkennung die Position der Person im Bild innerhalb eines gewissen Bereiches auf der linken Seite des Bildes angibt, soll sich das Fahrzeug nach links drehen. Oder wenn der Befehl „Vorwärts“ gesprochen wird, soll das Fahrzeug nach vorne fahren. Der Wechsel zwischen Personen-, Gesten- und Spracherkennung soll über Sprachbefehle geschehen.

Die von der Fahrzeugsteuerung bestimmten Befehle werden von der Motorsteuerung durch die Ausgabe von einer entsprechenden Spannung an den Motor-Pins umgesetzt. Die Motorsteuerung läuft in Software. Da jedoch die GPIO Pins nicht von der Software aus angesteuert werden können, muss auf dem FPGA eine Hardwarebrücke zu den Pins aufgesetzt werden.

Die Motorsteuerung enthält ebenfalls einen Webserver. Über eine Steuerungsapp können so direkt Befehle an die Motorsteuerung gegeben werden. Dies dient zum Testen und Debuggen der Motorsteuerung, solange die Personen-, Gesten- und Spracherkennungsmodule noch nicht zuverlässig funktionieren.

In dieser Runde des Projekts wurde die alte Motorsteuerung (geschrieben in C++) verworfen und vollständig neu in Python implementiert (siehe Unterabschnitt 5.3.6). Ebenfalls wurde eine Hardwarebrücke zu den Pins eingerichtet (siehe Unterunterabschnitt 5.1.2) und der Webserver samt Steuerungsapp implementiert (siehe Kapitel 5). Noch nicht implementiert ist die Umsetzung der Eingabesignale der Personen-, Gesten- und Spracherkennung in Befehle für die Motorsteuerung, sowie Sprachbefehle zum Wechsel den drei Eingabemethoden.

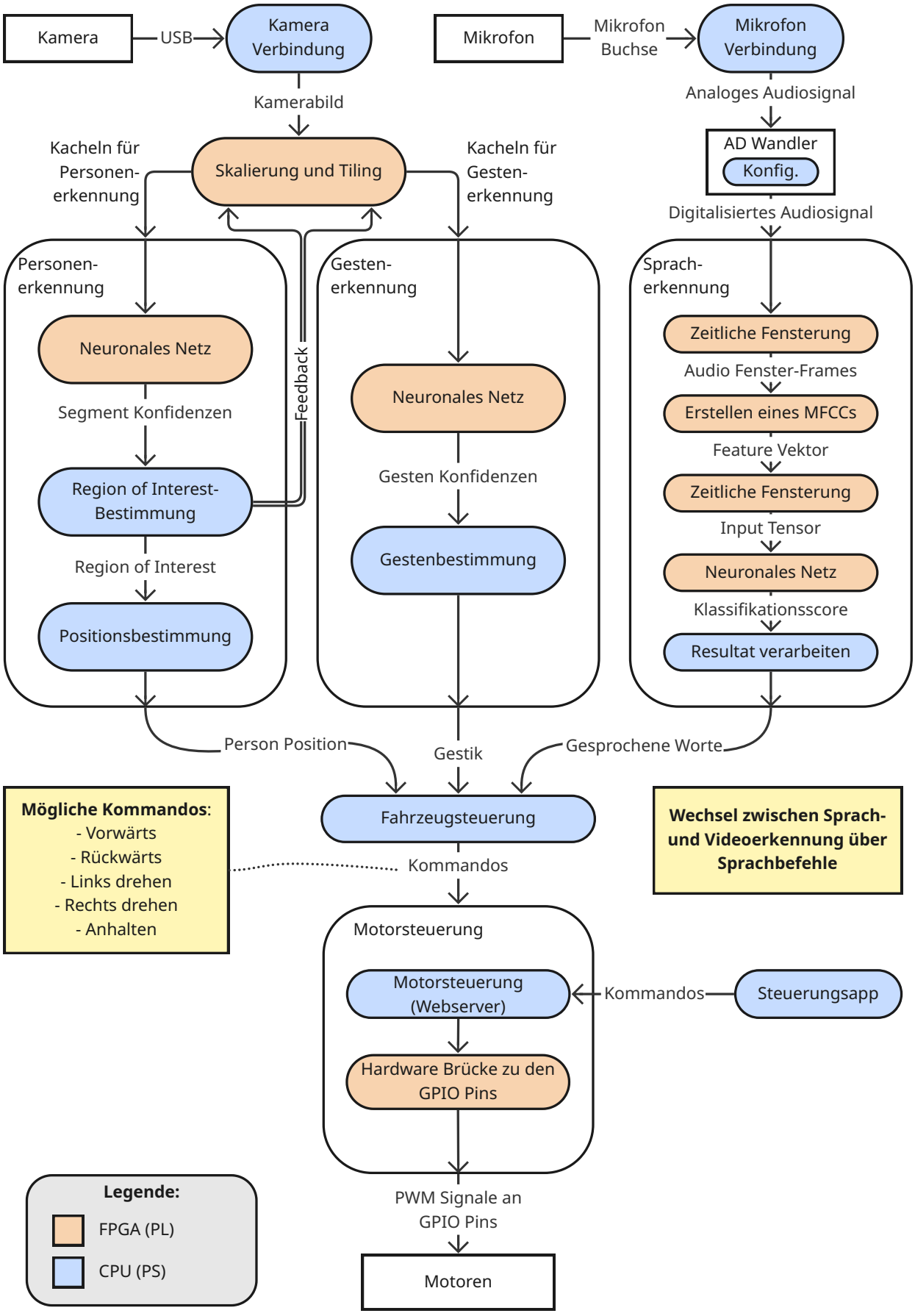


Abbildung 2.2.: Geplante Architektur

3. Personenerkennung

geschrieben von Keno Albers, Jonas Broda, Julius Böttger, Jannik Hennicke, Nils van Rijsinge

3.1. Behandelte Themen

Die ursprüngliche Aufgabenstellung lautete: „Analyse, warum der Ende-zu-Ende-Systemtest nicht ganz so gute Erkennungsergebnisse lieferte wie das Validierungsskipt“. Während der Recherche im Bericht aus dem Wintersemester 2021/22 [ACB+22] stellte sich jedoch heraus, dass der Systemtest bessere Ergebnisse liefert und das „Validierungsskript“ fehlerhaft ist. Zudem ist der Systemtest als nicht aussagekräftig und reproduzierbar zu bewerten und auch das Trainingsskript bedarf noch Verbesserungen. Daraus ergaben sich drei Aufgabenstellungen bzw. -bereiche. Auf der einen Seite sollen sowohl der Trainingsprozess als auch das „Validierungsskript“ verbessert werden (Abschnitt 3.4 und Abschnitt 3.5). Auf der anderen Seite ist das Ziel, eine neue Testmethode für den Ende-zu-Ende-Systemtest zu entwickeln (Abschnitt 3.6). Diese soll einen zuverlässigen, aussagekräftigen und nachvollziehbaren Wert liefern. Bestenfalls liefert die Überprüfung der Genauigkeit des Modells via „Validierungsskript“ dann einen deutlich besseren Wert und der Unterschied der Resultate der beiden Testverfahren ist nicht mehr so groß.

3.2. Grundlagen

3.2.1. Overfitting

Zum Trainieren von neuronalen Netzen, die in der Lage sind, auf neue, ungesehene Daten generalisiert zu reagieren, ist eine Modellbewertung ein entscheidender Schritt [WBM23]. Ein bekanntes Problem hierbei ist Overfitting, bei dem ein Model nicht gut von bekannten Daten auf neue Daten generalisieren kann. [Yin19]. Einfacher ausgedrückt beschreibt Overfitting den Vorgang, bei welchem ein neuronales Netz, oder ein KI-Lerner im Allgemeinen, die Daten so sehr annähert, dass dieser diese im Grunde „auswendig“ lernt, aber sein Gelerntes nicht auf unbekannte Daten anwenden kann. Dies kann auch mit einer begrenzten Anzahl von Daten und bereits nach wenigen Trainingsphasen auftreten [CT10].

Für Overfitting kann es unterschiedliche Gründe geben. Diese können in drei Kategorien unterteilt werden. Zum einen das Auswendiglernen von Rauschen in den Daten, ohne das dahinterliegende Konzept zu verstehen. Dies passiert oftmals, wenn es zu wenig Daten oder Daten gibt, die die wichtigen Informationen nicht gut abbilden. Eine weitere Ursache liegt in der Balance zwischen „accuracy“ (Genauigkeit) und „consistency“ (Konsistenz). Wenn die Genauigkeit erhöht werden soll, kann die Komplexität bzw. die Anzahl der Eingänge erhöht werden, dies verringert jedoch die Konsistenz. Somit wie gut das Modell auf andere Daten angewendet werden kann. Die dritte Kategorie beschreibt das Problem bei der Auswahl von guten Modellen. Diese werden basierend auf bestimmten „scores“ und Bewertungsfunktionen ausgewählt. Da jedoch immer das Modell

mit dem besten score gewählt wird, wird ein Modell, welches nur garantiert gut an die gegebenen Daten passt, ausgewählt und nicht garantiert gut generalisiert [Yin19, S. 1-2].

Ein einfaches und bekanntes Verfahren, um Overfitting zu vermeiden ist „Early-Stopping“. Wenn der Accuracy-Score auf Trainingsdaten immer besser wird, wird das Modell aufgrund von Noise-Lernen ab einem gewissen Zeitpunkt schlechter. Zur Überprüfung wird ein Validierungs-Score verwendet, wenn dieser sich nicht verbessert oder sogar schlechter wird, dann ist dies ein direkter Indikator dafür, dass der Lerner gerade auf dem Weg zum Overfitting ist. Das Training sollte abgebrochen bzw. frühzeitig gestoppt werden [Yin19].

3.2.2. Stratified Sampling

Beim zufälligen Aufteilen des Datensatzes für Training und Validation gibt es ein Problem. Es kann passieren, dass durch das zufällige Aufsplitten des Datensatzes eine Klasse in Training- oder Validate-Subdatensatz über- beziehungsweise unterrepräsentiert ist. Dies hat zur Folge, dass die berechneten Scores aus Unterabschnitt 3.2.1 nicht mehr aussagekräftig sind, da diese nach unten oder oben verschoben sein können.

Dieses Problem wird mit der Methode des Stratified Sampling gelöst. Die Idee kommt aus der Statistik in der Wirtschafts- und Sozial-Forschung und wurde im KI-Kontext wiederverwendet. Hier wird die "[...] Grundgesamtheit in disjunkte Klassen (Schichten) [aufgeteilt]"[Mos12, S. 4]. Dabei sollen die diese Schichten so gewählt werden, dass sich die Schichten in Bezug auf die Klassen gleich verhalten, also die Verhältnisse erhalten bleiben. Dazu werden n Schichten gebildet, in welchen dann zufällige Datensätze verschiedener Klassen proportional zu ihrem Vorkommen ausgewählt und dieser Schicht zugeordnet werden. So erhält man n Schichten in welchen die Klassenverhältnisse annähernd gleich sind.

3.2.3. Trainieren von neuronalen Netzen und Modellbewertung

Um die Genauigkeit eines Modells zu bestimmen, werden für verlässliche Aussagen nicht die gleichen Daten, wie beim Training verwendet. Diese könnten aufgrund von Overfitting nicht die tatsächliche Güte des Modells, sondern nur die Anpassung an die Daten beschreiben. Aus diesem Grund werden die Daten aufgeteilt in einen Trainings- und einen Testdatensatz. Zusätzlich können die Trainingsdaten erneut aufgeteilt werden in einen Training- und einen Validierungsdatensatz. [WBM23]

Der Trainingsdatensatz wird dazu verwendet das Netz zu trainieren und die Gewichte so anzupassen, dass sie den Trainingsdatensatz gut annähern. Der Validierungsdatensatz oder auch Cross-Validierungsdatensatz wird noch im Trainingsprozess verwendet, um das Modell optimal zu trainieren. Dieser Datensatz kann mit Cross-Validierung genutzt werden, um verschiedene Algorithmen zu vergleichen und den besten auszuwählen. Auch können die besten Hyperparameter eines Algorithmus ermittelt werden oder ein Overfitting beim Trainieren von neuronalen Netzen festgestellt werden.
Das Testskript wird nach Abschluss des Trainings verwendet, um die Güte des Netzes mit Daten, die es noch nie vorher gesehen hat, zu bestimmen. Damit wird geprüft, ob das Netz wirklich die Eigenschaft der gelernten Klassifikation abbilden kann. [WBM23; LLC22, S. 115ff.]

Die bisher verwendete Metrik für die Güte des Modells (Accuracy) kann z. B. bei stark unausgewogenen Klassen irreführend hoch sein, auch wenn das Modell die Minderheitsklasse schlecht

vorhergesagt wurde. Die Accuracy ist insgesamt nicht gut geeignet, wenn nur eine einzelne Metrik als Maß der Güte eines Modells gesucht ist. Matthews Correlation Coefficient (MCC) ist hingegen beliebt für diesen Anwendungsfall. Bei seiner Berechnung werden alle vier Felder der Konfusionsmatrix (TP, TN, FP, FN) mit einbezogen:

$$\text{MCC} = \frac{(TP \cdot TN) - (FP \cdot FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Der MCC bietet damit einen aussagekräftigeren und vergleichbaren Wert für die Güte verschiedener Modelle. [LLC22, S. 136]

3.3. Erkenntnisse aus vorherigen Projektberichten

Im Sommersemester 2021 [ACB+21] wurde sich unter anderem mit der Gestenerkennung beschäftigt. Um die drei Gesten *START*, *STOP* und *NOPOSE* erkennen zu können, wurde ein neuronales Netz mithilfe von Python-Skripten entwickelt (Die Implementation des Trainingsprozesses ist in Abschnitt 3.4 beschrieben). Zum Trainieren und Testen wurde ein Datensatz aus selbst aufgenommenen Bildern erstellt. Als Modellframework wurde Brevitas verwendet, was vor allem durch die Möglichkeit des Exports eines trainierten Modells im ONNX-Format überzeugen konnte. Der Trainingsprozess wurde in einer Cloud-Umgebung ausgeführt, um schneller neuronale Netze trainieren zu können ohne entsprechende eigene Hardware zu benötigen (vgl. [ACB+21, S. 61ff.]). Ein Training auf einer Cloud Graphics Processing Unit (GPU) war dabei jedoch nicht erfolgreich und es wurde auf der CPU trainiert (vgl. [ACB+21, S. 63]).

Zum Ausführen des neuronalen Netzes wurden aus den Bilder Kacheln von 32×32 Pixel erzeugt, da das verwendete Modell nicht mit größeren Bildern arbeiten kann. Diese Kacheln werden im *TilingController* auf dem Auto erzeugt. In [ACB+21, S. 73] wird angegeben, dass der *TilingController* aus dem Vorgängerprojekt übernommen wurde. Die genaue Umsetzung des Tilings ist in [MKB+21, S. 23ff.] beschrieben.

Im Wintersemester 2021/22 [ACB+22] wurde an der Personenerkennung und nicht mehr an der Gestenerkennung gearbeitet. Das Ziel war, ein neuronales Netz zu trainieren, dass erkennen kann, ob sich eine Person auf einem gegebenen Bild befindet oder nicht. Die Skripte dafür wurden aus dem vorherigen Semester [ACB+21] übernommen. Der Datensatz basiert auf dem COCO-Datensatz [COC], welcher unter der Creative Commons Attribution 4.0 License [Cre] veröffentlicht worden ist, wurde aber zudem noch manuell gefiltert. So wurden nur Bilder in den Datensatz übernommen, auf denen Personen relativ groß im Bild zu sehen sind. Aufgeteilt wurde der Datensatz in etwa 90 % Trainigs- und 10 % Testdaten. Das Projektteam hat am Ende festgestellt, dass sich die Ergebnisse des „Validierungsskripts“ mit den Beobachtungen aus dem Systemtest widersprechen.

Grundsätzlich sind Tests wichtig, um die Erfüllung von Anforderungen anhand von korrektem Verhalten zu überprüfen. So auch für die Personenerkennung auf dem Fahrzeug. Aus dem Bericht [ACB+22] lässt sich bereits ein Testszenario entnehmen. Das „Testszenario zur Überprüfung der Teilprojektziele der Personenerkennung“ [ACB+22, Anhang G] des Berichts zeigt eine angepasste Version des „Testszenarios zur Überprüfung der Projektziele“ [ACB+22, Anhang F]. Die Version wurde aufgrund eines Zeitmangels bis zur Projektdeadline auf den damaligen Projektfortschritt angepasst. Entsprechend der Testszenarien wurde ein Systemtest durchgeführt, der mithilfe eines Videos festgehalten wurde (zu finden im Repository des WiSe 2021/22 unter *soc-nn_main_repo/abgabe/WiSe2122*) und einen Eindruck der Performance in einem Systemtest gibt. Ziel war es

das Testszenario auszuführen, im Video wurden jedoch nicht vollständig die Rahmenbedingungen des Szenarios eingehalten. Beispielsweise gibt es Zeitpunkte, zu denen zwei Personen im Bild zu sehen sind. Dort wurde die Personenerkennung auf dem Auto selbst mit der gesamten Hardware in der Praxis getestet. Im Projektbericht wird als Ergebnis von einer „subjektiven Einschätzung“ geschrieben, welche angibt, dass in mehr als 80 % der Fälle eine Person korrekterweise erkannt wird (vgl. [ACB+22, S. 35]). Die Methodik oder Grundlage dieser Einschätzung wird nicht genannt. Dieser Wert ist somit als nicht aussagekräftig zu bewerten und kann auch nicht reproduziert werden.

Als weiteren Test wurde zur Evaluation ein „Validierungsskript“ verwendet. Dieses prüft für einen Testdatensatz, wie viele Bilder von einem trainierten neuronalen Netz korrekt klassifiziert werden. Die Ausführung des Skripts findet in einer Python-Entwicklungsumgebung auf einem Rechner statt, Genaueres zum bisherigen Stand des Codes ist in Abschnitt 3.4 aufgeführt. Für den verwendeten Datensatz gab das Skript eine Genauigkeit von etwa 50 % für das Modell an (vgl. [ACB+22, S. 27]). Allerdings ist dem Projektteam aufgefallen, dass das „Validierungsskript“ erhebliche Bugs aufwies, die im Wintersemester 2021/22 jedoch nicht mehr behoben wurden. Es wird davon ausgegangen, dass die schlechte Genauigkeit von 50 % und starke Abweichung vom Resultat des Systemtests auf das fehlerhafte Validierungsskript zurückzuführen sei.

Als Ausblick und mögliche Verbesserungen wird im Bericht aus dem Wintersemester 2021/22 genannt, dass die große Diskrepanz zwischen den ermittelten Werten der Genuigkeit des Modells untersucht werden soll, da eine aussagekräftige Evaluation essenziell für das Bewerten von Änderungen am Trainingsprozess ist. Weiterhin wird erwähnt, dass eine Überarbeitung des Trainingsskripts sinnvoll sein kann, da bisher nicht beim Training erkannt werden kann, wie viele Epochen optimalerweise trainiert werden sollte (vgl. [ACB+22, S. 39f.]).

3.4. Bisheriger Stand des Codes

Der gesamte Ablauf des Trainings- und Testverfahrens sowie der FINN Export geschehen in der Jupyter-Notebook-Datei *src/training/Train.ipynb* des *soc-nn_main_repo*. Dies wurde im Rahmen des Sommersemesters 2021 erarbeitet. Im Folgenden wird der gesamte Prozess beschrieben. Genauere Informationen zum Ablauf sind im Projektbericht des Sommersemesters 2021 [ACB+21, S. 107f.] nachzulesen. Der Code und die dortigen Erläuterungen sind zwar primär für die Gestenerkennung geschrieben, das Prinzip kann aber genauso auf die Personenerkennung angewendet werden, wie es auch bereits im Wintersemester 2021/22 getan wurde.

Das Jupyter Notebook ist in die Sektionen *Prepare Environment*, *Load Dataset*, *Show Class Ids*, *Training*, *Measure accuracy* und *FINN Export* aufgeteilt. Im ersten Teil werden die Pfade zu den Datensätzen angegeben und wo das Modell gespeichert bzw. abgerufen werden soll. Zudem werden die Konfigurationsparameter für das quantisierte neuronale Netz festgelegt und das Gerät bestimmt, auf dem das Training durchgeführt werden soll (Central Processing Unit (CPU) oder GPU). Unter *Load Dataset* wird der Datensatz erstellt. Dazu wird unter anderem die Batch Size bestimmt, die in diesem Fall 32 beträgt. Außerdem wird der Name des Datensatzes angegeben. Hier wird der COCO-Datensatz verwendet, der schon aufgeteilt in Trainigs- und Testdaten vorliegt. Dementsprechend gibt es einen Ordner *train* und einen Ordner *test*, in denen es jeweils einen Ordner für die beiden Klassen *person* und *no person* gibt. Die Aufteilung zwischen Trainings- und Testdaten beträgt etwa 90/10. Im nächsten Schritt werden in der Konsole die aus dem Datensatz identifizierten Klassen ausgegeben.

Der Abschnitt *Training* beinhaltet das Training des neuronalen Netzes. Dazu wird das ausgelagerte

Trainingsskript (*src/training/trainers/basic.py*) verwendet. Zuerst wird eine Instanz der Klasse *Trainer* erstellt, die das neuronale Netz, den Pfad, wo die Gewichtungen gespeichert werden sollen, den Trainingsdatensatz, das „Trainingsgerät“ und die Anzahl der zu trainierenden Epochen übergeben bekommt. Anschließend wird die *train()*-Methode aufgerufen.

Das Testen des trainierten Modells erfolgt im anschließenden Abschnitt, ebenfalls mit einem ausgelagerten Skript. Hier gibt es die Klasse *Validator*, die das neuronale Netz, den Pfad zu den gespeicherten Gewichtungen, den Testdatensatz und auch das „Trainingsgerät“ übergeben bekommt. Um den Test auszuführen, wird die *validate()*-Methode aufgerufen. In dieser werden die Testdaten mithilfe des Modells klassifiziert, die vorhergesagten Labels mit den tatsächlichen verglichen und eine Genauigkeit des Modells bestimmt.

Unter dem Abschnitt *FINN Export* wird das trainierte, quantisierte Modell mit FINN, einem Framework für neuronale Netze auf FPGAs, so zu exportieren, dass es auf ein FPGA geladen werden kann. Nach erfolgreicher Ausführung des Trainings mit den festgelegten Python-Paket-Versionen (siehe Unterabschnitt 3.5.2) produzierte der FINN Export immer noch eine Vielzahl kryptischer Fehlermeldungen.

Der Code ist generell nicht einfach ausführbar, da weder die benötigten Python-Pakete noch ihre Versionen dokumentiert sind. Die einfache Installation der neusten Versionen aller im Code verwendeten Pakete führt zu Import-Fehlern, weil die installierten, neusten Versionen einiger Pakete nicht kompatibel mit den unbekannten, zuletzt verwendeten Versionen sind.

3.4.1. Bisheriger Stand und Probleme des Trainingsskripts

Das Trainingsskript trainiert ein Modell für eine vorgegebene Anzahl an Epochen. Dabei erfolgt keine Laufzeit-Überprüfung für Overfitting und anschließendes Early Stopping oder ähnliches. Das Modell, was aus der letzten Trainings-Epoche resultiert ist, wird gespeichert und als „bestes Modell“ weiter verwendet/getestet.
Das ist problematisch, da bei einer bestimmten (unbekannten) Anzahl an Trainings-Epochen das Modell anfängt zu overfitten, was nicht überprüft wird. Es sollte schon während des Trainings eine Validierung des Modells erfolgen, was (unter anderem) die Erkennung und Vermeidung von Overfitting erlaubt, aber auch ermöglicht, begründet das „beste Modell“ auszuwählen, anstatt blind das Letzte weiterzuverwenden.

3.4.2. Bisheriger Stand und Probleme des Testskripts

Das Testskript war zunächst nicht ausführbar, da die Variable *self.total* nicht definiert war, bevor sie das erste Mal ausgelesen wurde. Zudem wurde für jeden Batch das gesamte Modell des neuronalen Netzes neu geladen, was unnötig Rechenzeit beansprucht. Ein weiteres Problem war, dass die Batch Size nicht richtig bestimmt wurde. Dadurch sind immer nur zwei Bilder statt 32 pro Batch überprüft worden. Dementsprechend wurde nur ein kleiner Teil der Testdaten wirklich verwendet. Außerdem waren auch die Textausgabe und das Plotting der Bilder fehlerhaft. Dies ist in Unterabschnitt 3.4.3 genauer aufgezeigt.

3.4.3. Durchführen des Trainings- und Testverfahrens mit bestehendem Modell und Skript

Um den aktuellen Stand und die bisherigen Ergebnisse besser beurteilen zu können, wurde das gesamte Trainings- und Testverfahren mit den bestehenden Skripten durchgeführt. Dazu waren zunächst einige Änderungen notwendig, damit die Skripte ausführbar waren und eine beurteilbare Ausgabe lieferten. Diese Änderungen wurden in den Dateien *src/training/validators/basic.py* und *src/training/Train.ipynb* des *soc-nn_main_repo* vorgenommen. Erstere enthält das Testskript und musste dahingehend angepasst werden, dass die Variable *self.total* am Anfang initialisiert werden musste. Im Jupyter Notebook mussten die Pfade an die eignene Dateistruktur angepasst werden. Zudem war eine Namensänderung des Datensatzes nötig. Für die richtige Ausgabe der Ergebnisse wurde beim Training *print_every* auf 10 gesetzt, damit es während des Trainings Statusupdates gibt. Außerdem wurde *print_test_information* auf *True* gesetzt, um einen ausführlichen Output der Testergebnisse zu ermöglichen. Der FINN-Export bleibt unverändert und damit nicht funktionsfähig, da er das Ziel unserer Aufgabenstellung nicht relevant ist. Der Code, mit dem das Training- und Testverfahren durchgeführt wurde, befindet sich in einer separat exportierten zip im Repository *baseline-running-version.zip*. Wie dort auch in der REAMDE beschrieben, wurde der Datensatz selbst nicht exportiert. Es wurde jedoch der gleiche Datensatz, wie in der finalen Version auf dem Repo, verwendet.

Bei einer Durchführung des alten Testskripts nach Trainieren eines Modells über 10 Epochen lieferte das Skript folgende Ergebnisse. Von den eigentlich knapp 160 Testbildern wurden letztendlich nur 10 getestet. Davon wurden 6 Bilder falsch vom Modell klassifiziert, was einer unbrauchbaren Accuracy von 40 % entspricht. Auf unterschiedlichen Geräten wurden mit gleichem Code zudem unterschiedliche Werte ermittelt (40 %, 50 % und 70 %), was untermalt, dass das „Validierungsskript“ unbrauchbar ist. Außerdem gibt es allgemein Fehler in der Ausgabe der Ergebnisse. Diese sind in Abbildung 3.1 zu erkennen. Zum einen kommen in der Textausgabe Bilder mehrfach vor, gleiche Bilder werden teilweise unterschiedlich klassifiziert laut dem Output und Label und Prediction stimmen teilweise überein, obwohl die jeweiligen Bilder angeblich falsch klassifiziert wurden. Auch die Ausgabe der Bilder weist erhebliche Fehler auf. Die Bilder passen nicht zu den angegebenen Pfaden, da alle Bilder eine Person zeigen, obwohl in allen Pfaden „noperson“ steht.

Abbildung 3.1.: Fehlerhafter Output des Validierungsskriptes; Lizenz Personenbilder: CC BY 4.0 [Cre]

3.5. Verbesserungen

3.5.1. Allgemeine Verbesserungen

In vorgänger Projekten wurde auf ein Cloud-Training zurückgegriffen. Die Einrichtung und der nötige Import und Export für jeden Trainingsdurchlauf wurde als großer Nachteil und zusätzlicher Aufwand gesehen. Die Vorteile für eine einheitliche Umgebung konnten mit Verbesserungen für eine reproduzierbare Entwicklungsumgebung erreicht werden und ein schnelles Training konnte durch die Verwendung einer GPU (siehe Unterabschnitt 3.5.3) erreicht werden. Aus diesen Gründen wird das Training lokal und nicht in der Cloud ausgeführt, womit eine deutliche schnellere Entwicklung und Verbesserung der Skripte möglich ist.

Bei der initialen Erstellung des Trainingsprozesses in [ACB+21] wurde Beispielcode für das ausgewählte Model *CNV* von Brevitas [FPF25] übernommen (vgl. [ACB+21, S. 55]). Dies wurde zwar im Bericht dokumentiert, aber im Code nicht angegeben. Der Code in den entsprechenden drei Dateien (*src/models/cnv.py*, *src/models/common.py* und *src/models/tensor_norm.py*) wurde in diesem Jahr auf den aktuellsten Beispielcode von Brevitas aktualisiert. Dabei wurde die Originalquelle[1], die Lizenz und jegliche Änderungen im Code angegeben. Der Code ist unter der „3-Clause BSD“ Lizenz [Reg] verfügbar. Nicht verwendeter Beispielcode wurde entfernt.

Die bisherige Aufteilung in Trainings- und Testdaten wurde in einen *Train-Validate-Test-Split*

[1] https://github.com/Xilinx/brevitas/blob/41b78bbee5c3d5b7514906048c19ebd766339b53/src/brevitas_examples/bnn_pynq/models/

geändert. Somit besteht nun die Möglichkeit, einen Validierungsdatensatz zur Vermeidung von Overfitting zu verwenden. Die Daten liegen zunächst weiterhin in der bisherigen Aufteilung vor. Vor dem Training wird zusätzlich der Trainingsdatensatz in *Train* und *Validate* aufgeteilt. Dazu wurde der Parameter *validate_ratio* eingeführt, um das Verhältnis zu bestimmen. In diesem Fall wurde sich für eine 80/20-Aufteilung entschieden.

In diesem Zuge ist auch aufgefallen, dass das „Validierungsskript“ eigentlich ein Testskript ist. Die Überprüfung der Accuracy nach dem abgeschlossenen Training wird als Test bezeichnet, während die Validierung in jeder Epoche des Trainings zur Bestimmung des Validation Losses stattfindet. Daher wurde die Klasse *Validator* in *Tester* umbenannt und entsprechend auch weitere notwendige Namensänderungen durchgeführt.

Des Weiteren wurde im Teil *Prepare Environment* des Jupyter Notebooks der Parameter *num_classes* von 3 auf 2 gesetzt, da bei der Personenerkennung nur die Klassen PERSON und NOPERSON existieren. Zuvor wurde bei der Gestenerkennung zwischen 3 Klassen unterschieden.

Zuletzt wurde versucht, Determinismus für das Trainings-Notebook zu konfigurieren, was aus unbekannten Gründen nicht erfolgreich war. Alle verwendeten Python-Pakete, die es unterstützen, wurden entsprechend konfiguriert (z. B. durch die Angabe eines festen Seeds für Zufalls-Operationen), jedoch ist trotzdem ein nicht-determinismus des Codes feststellbar, z. B. durch unterschiedlich gute Modelle die nach dem Training mit dem selben Code auf dem selben Gerät entstehen.

3.5.2. Verbesserungen für eine reproduzierbare Entwicklungsumgebung

Da bei den meisten verwendeten Python-Paketen nicht bekannt war, welche Version zuletzt verwendet wurde, wurde nun initial von der jeweils neusten Version ausgegangen, mit der Vermutung, dass für einige Packages ältere Versionen benötigt werden. Bis auf `dependencies` (für Dependency Injection) und `brevitas` (für quantisierte neuronale Netze) haben die aktuell neusten Paket-Versionen problemlos funktioniert.

Bezüglich Brevitas war aus vorheriger Projektdokumentation bekannt, dass zuletzt die Version 0.2.0a (Alpha, Januar 2021) eingesetzt wurde. Diese Version war erforderlich, da der benötigte FINN-ONNX-Export in späteren Versionen verändert, und schließlich in Version v0.10.0 (Dezember 2023)[2] in seiner bisherigen Form sogar ganz entfernt wurde. Die Installation der Alpha-Version gestaltete sich jedoch problematisch, da sie lokal aus dem Quellcode gebaut werden musste, was aus unklaren Gründen nur auf einem unserer Geräte erfolgreich war.
Mit der nächst-besten Version (0.2.0), die in dem offiziellen Python Package Repository PyPI verfügbar war, traten wiederholt Probleme beim Import des FINN-ONNX-Exporters auf, die teils nur durch einen Neustart der Entwicklungsumgebung behoben werden konnten.
Als Nächstes wurde die Version 0.9.1 (April 2023) untersucht, da sie die neuste Version ist, die den FINN-ONNX-Export noch enthält. Nach kleinen Anpassungen an veränderte Funktionsnamen und Modul-Strukturen dieser Brevitas-Version lief sie auf allen getesteten Geräten stabil und ohne Fehler. Auch musste der Input-Shape der `INTERMEDIATE_FC_FEATURES` in *src/models/cnv.py* im Modell angepasst werden, dort scheint sich das Verhalten von Brevitas auch verändert zu haben. Des Weiteren zeigte sich, dass die Brevitas-Versionen 0.2.0a bis (mindestens) 0.9.1 mit Python-Versionen größer als 3.12 inkompatibel sind, was zu Fehlern führte. Erfolgreichere Tests wurden mit Python 3.11.7 und 3.12.10 durchgeführt.

[2] „Deprecated FINN ONNX export flow.“ https://github.com/Xilinx/brevitas/releases/tag/v0.10.0

Die benötigte `dependencies`-Paket-Version war abhängig von Brevitas und wurde auf 2.0.1 (Juli 2020) festgesetzt.

Diese bestimmten Python-Paket-Versionen, mit denen der Code nun garantiert ausführbar ist, wurden in einer Python-typischen `requirements.txt`-Datei festgehalten, womit in Zukunft reproduzierbar die richtigen Versionen installiert und damit der Code ausgeführt werden können. Spezifisch für die Ausführung des Codes auf der Linux-Distribution NixOS liegt nun im Repository auch eine `flake.nix` [3] für eine reproduzierbare Entwicklungsumgebung bei.

3.5.3. Trainingsskript Verbesserung

Die unter Abschnitt Abschnitt 3.4 aufgeführten Probleme wurden behoben und das Training generell optimiert. Dabei wurden Bezeichner klarer benannt, der Code modularer gestaltet und aufgeräumt, sowie Dokumentationskommentare ergänzt.

Zuerst wird das Trainingsskript, wie in Listing 3.1 abgebildet, angepasst, dass Overfitting (Definition siehe Unterabschnitt 3.2.1) des Modells weitestgehend vermieden werden kann. Dazu wird nach jeder Trainingsepoche das Modell mit von den Trainingsdaten unabhängigen Validierungsdaten validiert und ein *Loss* berechnet. Verbessert sich dieser über eine bestimmte Anzahl an Epochen nicht, wird das Training abgebrochen („Early Stopping“).

Listing 3.1: Verbesserung: Overfitting vermeiden

```
# if model improved
if mean_validate_loss < best_validate_loss:
    best_validate_loss = mean_validate_loss
    epochs_without_improvement = 0
    self.history["best_epoch"] = epoch+1
    if best_validate_loss < previous_model_best_val_loss:
        # save model only if its better than the last saved model
        overwritten_prev_best_model = True
        self.save_best(epoch, best_validate_loss)
else:
    epochs_without_improvement += 1
    if epochs_without_improvement >= self.patience:
        print(f"Early stopping after epoch {epoch+1}, validate loss did
            not improve for {epochs_without_improvement} epochs")
        break
```

Zudem wird das Modell mit der `save_best()` Methode jedes Mal gespeichert, wenn ein neues bestes Modell (niedrigster Validierungs-Loss) gefunden wurde. Es wird darüber hinaus auch nur ein neues Modell gespeichert, wenn dieses einen niedrigeren Validation-Loss als ein vorher trainiertes aufweist. Somit wird vermieden, dass ein Modell welches im Training ein lokalen Optimum darstellt, nicht das bessere Modell (möglicherweise globales Optimum) überschreibt.

Zudem wurde der Datensatz mit der Methode des Stratified Sampling (vgl. 3.2.2) aufgeteilt, wie in Tabelle 3.1 dargestellt. Es ist zu erkennen, dass so die Klassenverhältnisse nahezu gleich sind und damit reproduzierbare und unverzerrte Trainings, Tests und Validierungen sicherstellen.

[3] Flakes sind ein experimentelles-Feature von NixOS (siehe `https://wiki.nixos.org/wiki/Flakes`)

Label	Trainings-Samples	Validierungs-Samples	Test-Samples
person	566	142	79
noperson	566	142	78

Tabelle 3.1.: Aufteilung des Datensatzes in Trainings-, Validierungs- und Testdaten

Für den Trainingsprozess wurde die Konsolenausgabe verbessert und eine Visualisierung des Train- und Validate-Losses über den Verlauf des Trainings hinzugefügt.

```
[Epoch 25] Average loss over last  10 Batches: 0.378
[Epoch 25] Average loss over last  20 Batches: 0.379
[Epoch 25] Average loss over last  30 Batches: 0.370
[Epoch 26] Average loss over last  10 Batches: 0.366
[Epoch 26] Average loss over last  20 Batches: 0.342
[Epoch 26] Average loss over last  30 Batches: 0.354
Early stopping after epoch 26, validate loss did not improve for 5 epochs
Finished Training in 14.382s.

No new best model found, the previous best model is still the best one with validate loss: 0.445
```

Abbildung 3.2.: Ausschnitt des Konsolenoutputs bezüglich des Trainierens des Modells

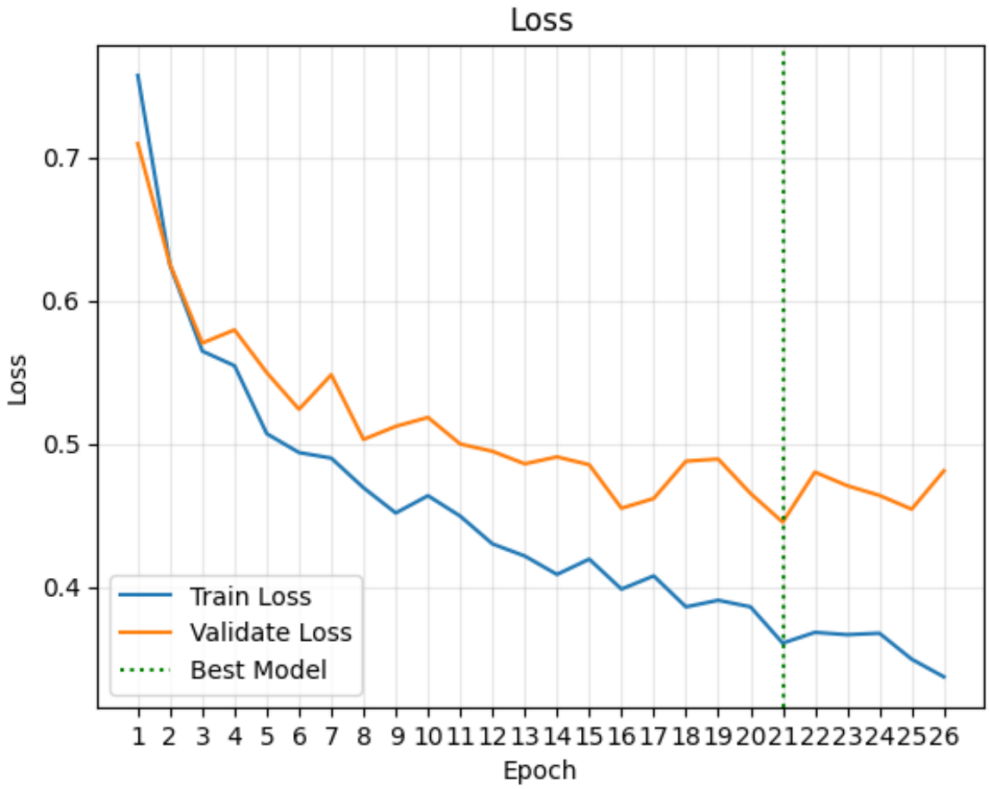


Abbildung 3.3.: Plotting des Verlaufs von Train Loss und Validate Loss

Wie bereits erwähnt (siehe Abschnitt 3.3) konnte das Training in vorherigen Projekten nicht auf einer GPU ausgeführt werden. Der Code wurde nun mit Nvidia CUDA [NVIa] kompatibel gemacht. Dies ermöglicht es, dass das Training auf leistungsfähigen und massiv parallelisierbaren Nvidia-Grafikkarten-Chips laufen kann und die Trainingszeit deutlich reduziert. Die dafür nötigen

Voraussetzungen und weiterführende Ressourcen können im Abschnitt F.1 eingesehen werden. In dem in Tabelle 3.2 dargestellten Vergleich wird deutlich, wie hoch der Performanzgewinn ist [4] [5]. Dabei wird das Training 5 mal mit maximal 100 Epochen durchgeführt, und die durchschnittliche Performanz dieser Trainings betrachtet. Die Tabelle der Einzelzeiten kann im Anhang F eingesehen werden.

Hardware	Durchschnittliche Trainingszeit in s
CPU	426.526
Nvidia Grafikkarten	13.3408

Tabelle 3.2.: Durchschnittliche Trainingszeiten auf verschiedener Hardware

Im Fall der konkreten verwendeten Hardware bietet die dedizierte Grafikkarte eine nahezu 32-fache Verbesserung der Laufzeit gegenüber der CPU.

3.5.4. Testskript Verbesserung

Die Probleme des Testskripts, die in 3.4 beschrieben sind, wurden gelöst. Zudem wurde der Code und die Codestruktur allgemein verbessert. Die konkreten Umsetzungen werden im Folgenden erklärt.

Um die Performance des Skripts zu verbessern, wird das trainierte Modell jetzt einmal initial geladen. Weiterhin ist eine Konsolenausgabe hinzugefügt worden, um den Nutzer darüber zu informieren, dass das Modell erfolgreich geladen wurde und aus welcher Epoche das Modell stammt. Diese Ausgabe ist in Abbildung 3.4 zu sehen.

```
Loaded best model (from epoch 21)
157 images. 74 were predicted wrong. Accuracy: 52.86624203821656%
```

Abbildung 3.4.: Ausgabe des Ladens und der Accuracy des Modells

Außerdem wird nun die Größe des Batches richtig bestimmt, sodass korrekt über alle Bilder eines Batches iteriert wird. Somit werden nun alle Bilder des Testdatensatzes auch wirklich getestet. Dies resultiert in einer Accuracy von etwa $\approx 53\,\%$ (siehe Abbildung 3.4). Dieser Wert ist nicht zufriedenstellend. Geht man davon aus, dass das Testskript nun die korrekte Accuracy liefert, müssen Training und/oder Testdaten geändert werden, um die Genauigkeit des Modells zu verbessern. Durch die Anpassungen am Testskript konnte die Accuracy nur minimal verbessert werden, wodurch die anfängliche Vermutung zu widerlegen ist, dass der schlechte Wert, der im Wintersemester 2021/22 ermittelt wurde, ausschließlich auf die Bugs und Fehler im ehemals Validierungsskript zurückzuführen sind.

Um nicht die gesamten Testbilder zu plotten, wurde das Testskript dahingehend geändert, dass nur noch die falsch klassifizierten Bilder ausgegeben werden. Diese werden mit dem Dateipfad und der Vorhersage versehen (siehe Abbildung 3.5) und die Größe des Plots anhand der Anzahl

[4] Es wurde ein 11th Gen Intel(R) Core(TM) i7-11850H (16) @ 2.50 GHz repräsentativ für das Training auf einer CPU verwendet

[5] Es wurde eine Nvidia RTX 4080 mit 9728 CUDA Kernen repräsentativ für das Training auf einer Grafikkarte verwendet

der zu plottenden Bilder ermittelt. Des Weiteren wurde der detaillierte Textoutput der falsch klassifizierten Bilder entfernt, da dieser keine zusätzlichen Informationen enthält.

Abbildung 3.5.: Ausschnitt der ausgegebenen falsch klassifizierten Bilder; Lizenz Personenbilder: CC BY 4.0 [Cre]

Da die Accuracy allein nicht aussagekräftig genug ist, um das Testen gut bewerten zu können, wurde der Output um eine Konfusionsmatrix und den MCC erweitert. Dies ist in Abbildung 3.6 dargestellt. Der MCC kann in einem Intervall von -1 (schlecht) bis 1 (gut) liegen. Ein Wert von $\approx 0,1$ bedeutet dabei, dass das Modell ähnlich akkurat ist wie ein Münzwurf, was nicht zufriedenstellend ist. Auch die Konfusionsmatrix zeigt, dass die Vorhersagen nur unwesentlich häufiger richtig als falsch sind.

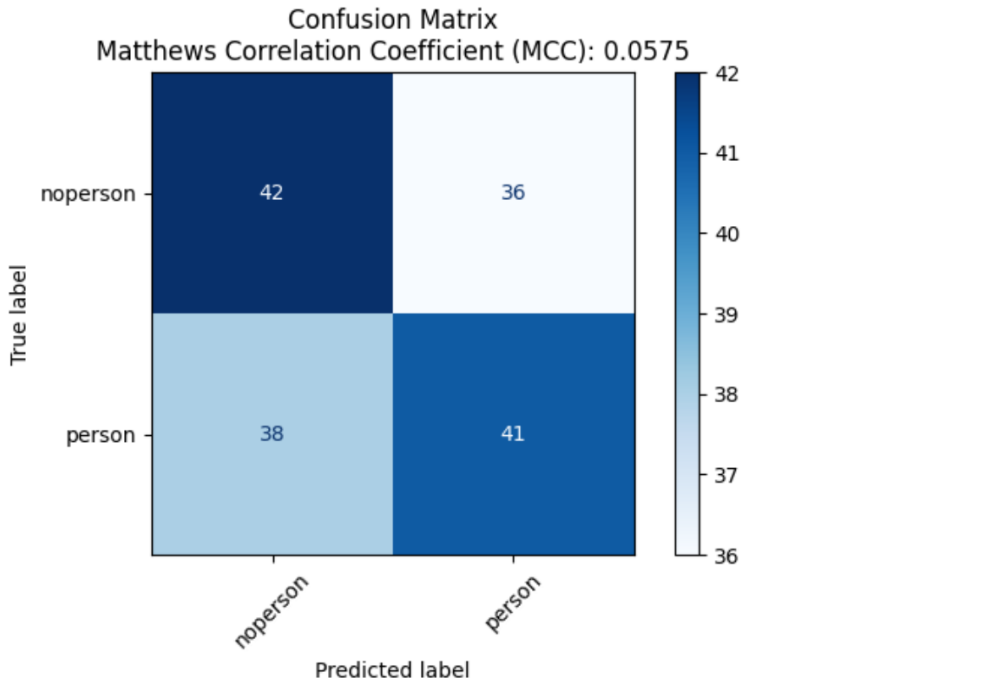


Abbildung 3.6.: Konfusionsmatrix

Um den Code nachvollziehbarer zu gestalten, wurden außerdem Kommentare und Docstrings ergänzt. Durch diese Anpassungen sind nun eine bessere Bewertung eines trainierten Modells und eine präzisere Analyse der Testergebnisse möglich.

3.5.5. Erweiterung des Datensatzes

Weiterhin wurde versucht, dass Trainingsergebnis zu verbessern, indem der Datensatz testweise erweitert wurde. Dazu wurde der freie OpenImages-Datensatz von Google herangezogen [Gooa], der wie der COCO-Datensatz auch unter der Creative Commons Attribution 4.0 License [Cre] veröffentlicht wurde. Dieser Datensatz enthält mehrere Millionen Bilder mit mehr als 5000 Labels. Für eine erste Überprüfung, ob der Datensatz geeignet ist, wurden etwa 1500 Bilder pro Klasse (*PERSON*, *NOPERSON*) hinzugefügt. Für die Klasse *PERSON* wurden ursprünglich 5000 Bilder heruntergeladen, allerdings war ein Großteil der Daten für diesen Anwendungsfall unbrauchbar, da viele Bilder große Menschenmassen enthielten oder Personen nur teilweise oder zu klein zu erkennen waren. Alle unpassenden Bilder wurden manuell aussortiert. Außerdem sollte beachtet werden, dass die Qualität des Labellings teilweise schlecht war. So waren beispielsweise Bilder von Spielzeugfiguren oder gemalten Personen mit dem Label „Person“ versehen. Zudem enthielten einige wenige Bilder verbotene Symbole oder Nacktheit, weshalb die zukünftige Nutzung dieses Datensatzes abgesehen von der Güte des Resultats überdacht werden sollte. Um Bilder für die Klasse *NOPERSON* zu erhalten, wurden Bilder mit verschiedenen Labels verwendet (bspw. „plant“, „house“ oder „vehicle“). Die neuen Daten wurden letztendlich im Verhältnis $90/10$ den bestehenden Trainings- und Testdaten hinzugefügt, wodurch die Menge der Daten sich etwa verdreifacht hat.

Entgegen der Erwartung einer verbesserten Genauigkeit lieferte das Testskript für das Training mit dem erweiterten Datensatz eine Accuracy von nur $\approx 53\,\%$, was eine Verschlechterung des Ergebnisses darstellt. Auch der MCC-Wert ist nach Hinzufügen der neuen Bilddaten auf $\approx 0,04$ gesunken. Somit stellte die Erweiterung des Datensatzes mit den verwendeten Bildern eine leichte Verschlechterung dar, was auf eine unzureichende Qualität der Bilder zurückzuführen sein kann. Daher wurden die Daten des OpenImages-Datensatzes wieder entfernt. Trotzdem besteht weiterhin die Vermutung, dass das Training durch einen umfangreicheren Datensatz verbessert werden kann.

3.6. Formalisierung einer Testmethode

3.6.1. Vorgehen und Hintergrund

Aus dem zuvor gegangenen Bericht war ein sinnvoller Bezug zwischen Testskript (ehemals Validierungsskript) und dem Testszenario nicht erkennbar. Die sichtbare Ausgabe durch die Base Station und die des Testskripts lassen, sich nicht aufeinander abbilden. Während das Skript Bild für Bild vorgeht, um eine Genauigkeit zu ermitteln, lässt das Testszenario nur einen Mittelwert vieler Bilder erkennen. Des Weiteren erzeugen die Aufnahmen der Kamera andere Bilder im Gegensatz zum Skript, welches ähnliche Bilder wie beim Training verwendet.

Erste Überlegungen haben sich darauf beschränkt, das Testszenario wie in [ACB+22, Anhang G] lediglich auf eine Testfolge, in der mehrere Szenarien betrachtet werden, auszuweiten. Ziel war es hier eine objektiv beurteilbare Qualität des End-to-End Tests zu bestimmen und so Netze untereinander vergleichbar zu machen. Zusätzlich sollte eine Vergleichbarkeit mit dem Testskript

erreicht werden. Dafür wurde folgende Tabelle (siehe Tabelle 3.3) erstellt. Hier ist eine beispielhafte verkürzte Darstellung abgebildet. Sie hilft dabei, mehrere Videos aufzunehmen, in denen mögliche Szenarien durchgespielt und „validiert“ werden. Allerdings ist es auch hier nicht möglich, eine klare erwartbare Ausgabe des Tests zu definieren.

Tabelle 3.3.: Testszenario der Personenerkennung

Schritt	**Objekt**	**Dauer**	**Abstand**	**Orientierung**	**Muster**	**Erfolg**
Erkennungsrate unterschiedlicher Abstände						
1	Person	10 s	1 M	Kamera	Keins	
Zuverlässigkeit der Personenerkennung						
7	Stuhl	10 s	2 M			
8	Rucksack	10 s	6 M			

Nach Rückschluss der Projektbeteiligten wurde die wichtige Erkenntnis erlangt, dass eine objektive Beurteilung der Qualität (wenn umsetzbar) nicht zu signifikanten für das Projekt relevanten Ergebnissen führen würde. Eine Beurteilung des Netzes kann bereits mit dem verbesserten Testskript erreicht werden. Zusätzlich wäre die Auswertung nicht mit dem Testskript vergleichbar, da dieses weiterhin andere Inputs erhalten würde. Dieser Ansatz muss somit auch nicht in folgenden Jahren aufgegriffen werden. Der Testentwurf wurde verworfen und ein neues Vorgehen entwickelt. Dieses basiert auf einer vollständigen Ausarbeitung einer Testvorbereitung für die Personenerkennung auf dem Fahrzeug.

Um eine Vergleichbarkeit zwischen Testskript und einem Test auf dem Fahrzeug zu erreichen, muss der Input beider Tests exakt gleich sein. Technisch ließe sich dieser Ansatz umsetzen, indem vorher ein Video aufgenommen wird. Dieses wird anstelle der Kameraaufnahmen in das Fahrzeug übertragen. Das Testskript könnte dann die einzelnen Frames des Videos als Bilder erhalten und auswerten. Dabei ist jedoch wichtig zu betrachten, welche Test-Ziele erreicht werden bzw. welche Komponenten tatsächlich getestet werden. Hier muss sich überlegt werden, welche Verarbeitungsschritte Einfluss auf die Erkennungsrate haben oder ob das neuronale Netz in einer anderen Umgebung arbeitet, die die Erkennungsrate beeinflusst. Der Vergleich mit dem Testskript kann bei der Suche nach Fehlern helfen. Nachfolgend werden diese weiteren Verarbeitungsschritte identifiziert.

Potenzielle Fehlerquellen

Um mögliche Fehlerquellen zu identifizieren, wurden die Schritte nach dem Testskript (ehemals Validierungsskript) betrachtet. Hierzu wird wie im Bericht [ACB+22, Siehe S. 31] erläutert, ein Synthetisierungsskript verwendet, um das im Training exportierte Netz auf das Field-Programmable Gate Array (FPGA) zu laden. Der Bericht [ACB+22, Kapitel 8, S. 64-66] beschreibt die Schritte im Einzelnen. Dabei werden unter anderem Layer transformiert, umgewandelt und aufgeteilt. Auch ohne eine genaue Analyse der einzelnen Schritte ist zunächst davon auszugehen, dass grundsätzlich Fehler auftreten können. Dies kann insbesondere die Qualität des Netzes negativ beeinflussen.

Für die Ausarbeitung von Testzielen lassen sich hieraus noch keine konkreteren Teile ableiten, die direkt getestet werden können. Es kann nur festgestellt werden, ob allgemein eine Verschlechterung in der Erkennungsrate auftritt oder nicht. Diese kann dann auf die potenzielle Fehlerquelle

der Synthetisierung zurückgeführt werden. Eine konkrete Ermittelung von Testschritten, die nur die Synthetisierung überprüfen ist nicht möglich, wenn auch noch andere Faktoren einen Einfluss auf die Erkennungsrate haben. Beispielsweise wurde überlegt, ob die Helligkeit im Bild einen Einfluss haben könnte und unterschiedliche Ergebnisse im Testskript und auf dem FPGA zeigt. Da jedoch nicht auf konkrete Bildinhalte abzuleiten ist, in welcher Form die Synthetisierung das Netz beeinflusst, würden solche Unterscheidungen nicht zielführend sein.

Ein weiterer Aspekt wird bei der Betrachtung der geplanten Systemarchitektur (Abschnitt 2.3) deutlich. Dabei wird auf dem Fahrzeug zur Verarbeitung der Bilder in einem Schritt das Bild herunterskaliert und Tiles erstellt. Wie auch der Bericht [ACB+22, S. 33f.] erläutert, ist die Performance des Fahrzeugs stark abhängig von der Größe und Anzahl der Tiles. Werden hier falsche Konfigurationen vorgenommen, können auch mit einem guten neuronalen Netz schlechte Ergebnisse auf dem Fahrzeug erreicht werden.

Insbesondere wird die Erkennung von Personen, die näher am Fahrzeug bzw. der Kamera oder weiter entfernt stehen, durch das Tiling beeinflusst. Optimale Einstellungen wurden bereits in früheren Projektberichten ermittelt. Dennoch ist nicht getestet, ob das Tiling mit den optimalen Einstellungen trotzdem noch einen signifikanten Einfluss auf die Qualität des Netzes hat. Daher ist ein Test in dieser Richtung sinnvoll.

Erkenntnisse

Es lässt sich zusammenfassend sagen, dass überprüft werden soll, ob das neuronale Netz nach Synthetisierung für das FPGA und in Kombination mit der Logik zur Kachelung eine vergleichbare Erkennungsrate, wie dasselbe neuronale Netz im Testskript erzielt. Zu berücksichtigen ist ein Rauschen, welches auftreten könnte. Es soll nicht konkret überprüft werden, welcher Faktor zu einem Fehler führen könnte. Der Test wird also als Blackbox Test gehandhabt.

Wenn, wie im Ansatz beschrieben, ein Video über die Kamera des FPGA aufgenommen wird und dieses mit dem Testskript ausgewertet wird, kann zusätzlich die Sinnhaftigkeit des COCO-Datensatzes zur Erkennung von Personen evaluiert werden. Sollten reale Aufnahmen von Personen zu vergleichbaren Erkennungsraten, wie die des COCO-Testdatensatzes, im Testskript führen, ist der COCO-Datensatz geeignet. Ist das Ergebnis jedoch schlechter, ist der Datensatz evtl. nicht geeignet für das Training zur Personenerkennung.

Mit der Überarbeitung bzw. Neuentwicklung des Tests zur Personenerkennung soll ein Test entwickelt werden, der primär auf Fehler bei der Integration des neuronalen Netzes auf dem FPGA eingeht. Zusätzlich kann mit diesem Test die Sinnhaftigkeit des COCO-Datensatzes für die Personenerkennung auf dem Fahrzeug evaluiert werden. Zuletzt bietet sich noch die Möglichkeit, die bereits erwähnte subjektive Einschätzung einer Erkennungsrate von mehr als 80 % mithilfe einer verbesserten Methodik neu zu ermitteln.

3.6.2. Test

Testziele festlegen

Eine 100 % richtige Klassifizierung in allen erdenkbaren Szenarien ist ein unrealistisches Ziel. Für ein nutzbares System ist es zusätzlich nicht notwendig eine perfekte Klassifizierung zu erreichen, es muss nur so verlässlich eine Person erkannt werden, dass diese verfolgt werden kann. Die folgenden Testziele sollen daher zunächst dazu dienen, Fehler in der Integration auf dem FPGA bzw. der Kachelung zu identifizieren. Wie bereits erwähnt, kann also keine Aussage darüber

getroffen werden „ob eine Person erkannt wird“, sondern, wenn überhaupt, nur darüber „ob die eine bestimmte Person im Video erkannt wird“. Dies ist aber auch nicht, nötig, da nur der Vergleich relevant für das Ergebnis ist.

1. Das neuronale Netz ist in der Lage, zwischen den Klassen *Person* und *keine Person* zu unterscheiden.
2. Das auf dem FPGA ausgeführte neuronale Netz erkennt *eine Person* mit einer Häufigkeit, die statistisch nicht signifikant[6] von der Referenz[7] abweicht.
3. Das auf dem FPGA ausgeführte neuronale Netz erkennt *keine Person* mit einer Häufigkeit, die statistisch nicht signifikant von der Referenz abweicht.
4. Die Kachelung hat keine statistisch signifikante Auswirkung auf die Erkennungsrate im Bezug auf die Referenz.

Rahmenbedingungen

Bei der Durchführung des Tests gibt es Rahmenbedingungen, die eingehalten werden müssen, um das übergeordnete Ziel und die konkreten Testziele zu erfüllen. Diese Rahmenbedingungen werden nachfolgend näher erläutert.

Um tatsächlich den Einfluss des Tilings testen zu können, müssen die Tests auf der echten Hardware durchgeführt werden, wie es auch im Systemtest bzw. End-to-End Test gemacht wurde [ACB+22, S. 36]. Der Begriff des End-to-End Tests, ist hier allerdings nicht passend, da Komponenten wie die Fahrzeugsteuerung von dem Test ausgenommen werden. Es wird sich auf einen Blackbox-Test der Komponente des neuronalen Netzes im Betrieb auf dem FPGA beschränkt. Um einen Vergleich mit dem Testskript zu ermöglichen, muss jedoch die exakt gleiche Eingabe auf der echten Hardware und im Testskript hineingegeben werden.

Diese Rahmenbedingungen schließen die gleichzeitige Durchführung und Auswertung der Tests aus. Die Problematik entsteht hier aus der Limitation, die exakten Frames eines zuvor durchgeführten Testdurchlaufes zu reproduzieren. Die Lösung wurde im Abschnitt Vorgehen und Hintergrund skizziert und kann darin gefunden werden, zunächst ein Video über die Kamera des FPGA / Autos aufzunehmen. Dabei werden die Aufnahmen der Kamera direkt an einen Laptop übermittelt und das Fahrzeug bleibt zunächst abgeschaltet. Dieses Video wird dann in seine einzelnen Frames aufgeteilt und als einzelne Bilder abgespeichert. Diese Bilder können anschließend in das Testskript geladen werden, um das neuronale Netz vor Synthetisierung auf dem PC auszuwerten. In einem zweiten Schritt wird das gleiche Video als Ersatz zur Kameraaufnahme an das FPGA übertragen. Anstelle der live Kameraaufnahmen erhält das FPGA die vorher aufgenommene Videosequenz. Ein aufwendiges Labeln jeder Videosequenz zur Bestimmung des ground truth, ob eine Person erkannt wird oder nicht, kann vermieden werden. Hierzu werden unabhängige Videosegmente verwendet, welche jeweils für die gesamte Dauer des Videos nur *Person* oder *keine Person* darstellen. Zur Vorbereitung des Videos wird in diesem Bericht ein Testdrehbuch geschrieben werden, welches gegen die definierten Testziele validiert werden kann.

Um das Video auf dem FPGA validieren zu können, ist es ggf. nötig, die Base Station um einen „Testmodus“ zu erweitern. Dieser arbeitet analog zum Testskript und ermittelt eine Accuracy. Die Besonderheit hier ist, dass es diese aus einem Video und nicht einzelnen Bildern ermittelt.

[6] Die Definition welche Werte statistisch signifikant sind, wird hier bewusst offen gelassen, da eine grundsätzliche Entscheidung für alle Erkennungsraten nicht einfach getroffen werden kann.

[7] Für die Referenz kann das Ergebnis des Testskriptes verwendet werden.

Um eine relevante Aussage treffen zu können, muss das Video eine ausreichende Länge bzw. ausreichende Anzahl an Frames beinhalten. Da das Erzeugen einer etwas längeren Video-Sequenz den Aufwand im Vergleich zu einer sehr kurzen Video-Sequenz nur minimal erhöht, ist es nicht notwendig, die minimale Anzahl an Frames für die Relevanz zu bestimmen. In den Testfolgen werden daher Zeitlängen gewählt, die lang genug sind, um die Erkennungsrate beurteilen zu können.

Es ist außerdem wichtig zu beachten, dass das Netz bereits durch das Testskript seine grundlegende Funktionalität bestätigt bekommt. Sollte das Netz dort bereits nicht hinreichend zwischen *Person* und *keine Person* unterscheiden können, ist der Blackbox Test nicht nötig.

Die Tests werden unter der Annahme gemacht, dass das neuronale Netz gedächtnislos ist. Da aktuell ein Convolutional Neural Network (CNN) (convolutional neural network) verwendet wird, ist dies gegeben. Sollte auf ein nicht anderes neuronales Netz gewechselt werden oder die Verarbeitung angepasst werden, sodass vorherige Bilder einen Einfluss auf die Erkennung nachfolgender Bilder haben, können die Tests ohne Anpassungen nicht durchgeführt werden.

Auswahl des Testlings

Da der gewählte Testling verschiedene Konfigurationsmöglichkeiten besitzt, muss dieser vor Durchführung konfiguriert werden. Die Basis dafür kann aus dem Dokument [ACB+22, S. 34] entnommen werden.

Konfiguration des generalisierten Testlings:

1. Das Fahrzeug wird mit dem Programm für die Personenerkennung beladen.
    - Das Tiling Setting wird auf den TILE SPEC:
      „32,32,1,1, 64,64,.75,.75, 128,128,1,1“ eingestellt.
    - Sollte das Laden des Programms nur in Kombination weiterer Funktionalität möglich sein, soll diese per Hard- oder Software deaktiviert werden. Beispielsweise durch das Abklemmen der Motoren.

Testfolge schreiben

Um die Testfolge durchzuführen, müssen zuerst die in den Drehbüchern beschriebenen Videosequenzen, durch die Kamera des Autos aufgenommen werden. Wie in den Rahmenbedingungen erläutert, wird das Video nachher zur Auswertung verwendet.

Ablauf:

1. Das Fahrzeug wird mit einem Abstand von etwa 10 Metern vor einer weißen Wand platziert.
    - Im Blickfeld dürfen keine Tisch- oder Stuhlbeine zu erkennen sein.
2. Die Kamera des Autos wird mit einem Laptop / PC verbunden.
3. Die Testvideos werden nach Anleitung der Drehbücher gedreht, abgespeichert und gelabelt.
4. Das Fahrzeug wird erfolgreich angeschaltet und mit der Base Station auf einem Laptop via WiFi verbunden.
5. Die Testvideos werden durch die Base Station und das Testskript ausgewertet.

Die folgende Tabelle stellt die Videosequenzen, die überprüft werden müssen, dar. Für jede Videosequenz muss der obige Ablauf durchgeführt werden. Siehe Unterunterabschnitt 3.6.2 bezüglich der Spalte „Erwartete Ausgaben“.

Tabelle 3.4.: Testfolge mit Videosequenzen zur Personenerkennung

Schritt	**Eingaben** *Frames*	**Erwartete Ausgaben**	**Erfolg**
1	Testvideo 1	Accuracy Testskript > 0	
2	Testvideo 1	Accuracy Base Station > 0	
3	Testvideo 1	Accuracy Testskript $\approx$ Accuracy Base Station	
4	Testvideo 2	Accuracy Testskript > 0	
5	Testvideo 2	Accuracy Base Station > 0	
6	Testvideo 2	Accuracy Testskript $\approx$ Accuracy Base Station	

Drehbuch Testvideo 1:

- Das Fahrzeug wird auf den Boden auf einen ebenen Untergrund gestellt.
- Das Fahrzeug wird mit einem Abstand von zehn Metern vor eine weiße Wand gestellt. Die Kamera schaut in die Richtung dieser Wand. Der Raum ist hell beleuchtet.
- Eine Person mit dunkler Kleidung stellt sich vor die Kamera. Dabei ist sie vollständig innerhalb des Sichtfeldes der Kamera zu erkennen.
- Die Aufnahme wird gestartet.
- Die Person bewegt sich langsam in Richtung der Kamera. Dabei ist sie zu jedem Zeitpunkt auf dem Video vollständig als Person zu erkennen.
- Die Aufnahme wird beendet und mit *Person* gekennzeichnet.

Drehbuch Testvideo 2:

- Das Fahrzeug wird auf den Boden auf einen ebenen Untergrund gestellt.
- Das Fahrzeug wird mit einem Abstand von zehn Metern vor eine weiße Wand gestellt. Die Kamera schaut in die Richtung dieser Wand. Der Raum ist hell beleuchtet.
- Die Aufnahme wird gestartet.
- Ein Stuhl wird in die Mitte des Sichtfeldes der Kamera als „nicht Personen Objekt“ platziert.[8]
- Die Aufnahme wird beendet und mit *keine Person* gekennzeichnet.

Nachprüfen der Vollständigkeit der Testfolgen

1. Das Testziel 1 wird durch die Kombination aus Testschritt 1&2 und 4&5 vollständig abgedeckt. Wenn die Accuracy größer als Null ist, wurde min. einmal die Klasse erkannt. Da für

[8] Es wird nicht getestet, wie gut ein Stuhl als *keine Person* erkannt wird. Bei der Platzierung eines Objektes vor der Kamera sind jedoch mögliche Unterschiede zwischen Testskript und FPGA amplifiziert im Gegensatz zu einer vollständig weißen Wand und somit besser auswertbar.

Schritt 1&2 die Klasse *Person* erkannt werden muss und für Schritt 4&5 die Klasse *keine Person* sind auch beide Klassen abgedeckt.

2. Das Testziel 2 ist mit Testschritt 3 vollständig abgedeckt. Der Testschritt überprüft genau die gefragte Abweichung, bei der Klasse *Person*.
3. Das Testziel 3 ist mit Testschritt 6 vollständig abgedeckt. Der Testschritt überprüft genau die gefragte Abweichung, bei der Klasse *keine Person*.
4. Das Testziel 4 ist mit Testschritt 3 oder 6 vollständig abgedeckt. Würde die Kachelung einen Einfluss haben, würden diese Testschritte nicht erfolgreich sein und die Erkennungsrate auf dem FPGA abweichen.

Auswertung

Wie auch bei der Durchführung, gibt es bei der Auswertung bestimmte Aspekte, die beachtet werden müssen, wenn Aussagen über die Qualität bzw. die Ergebnisse des Tests gemacht werden.

Wie unter Erkenntnisse und Testziele festlegen erläutert kann dieser Test nicht allgemein validieren, ob Personen immer hinreichend erkannt werden. Es wird lediglich ein Vergleich zwischen Erkennung auf FPGA und Testskript gezogen der die zuvor beschriebenen Testziele überprüfen kann. Es ist daher wichtig zu beachten, dass mit der Auswertung keine solchen allgemeinen Aussagen getätigt werden.

Das Testskript liefert nach der Durchführung eine konkrete accuracy über die Klassifikation der Bilder. Ebenso eine Auswertung durch die Base Station bzw. das FPGA direkt.

Die Testfolge gilt als Erfolg, sollten die Werte nicht statistisch signifikant voneinander abweichen. Die Definition der statistischen Signifikanz einer Abweichung wird hier bewusst offengelassen und muss im weiteren Projektverlauf noch ermittelt werden.

Sollte allerdings *große* Abweichungen zwischen den beiden Methoden auftauchen, kann man bereits Rückschlüsse bezüglich der Synthetisierung und/oder die Eignung des COCO-Datensatzes ziehen.

3.7. Zusammenfassung und Fazit

Es ließ sich in den vorherigen Kapiteln zeigen, dass die in Abschnitt 3.4 aufgezeigten Probleme behoben und der Code für das Training und die Validierung im Allgemeinen verbessert werden konnte.

Zuerst wurde die gesamte Entwicklungsumgebung und die verwendeten Python-Pakete lauffähig gemacht, da diese teilweise veraltet und nicht mehr kompatibel waren. Des Weiteren wurde die Umgebung so angepasst, dass sie über wiederholte Trainings und verschiedene Hardware hinweg weitestgehend reproduzierbar bleibt. Dies ist eine grundlegende Voraussetzung, um Trainings- und Validierungsergebnisse miteinander über Iterationen hinweg vergleichen zu können

Es wurde der Training-, Test- und Validierung-Datensatz besser aufgeteilt und das Training mit klassenerhaltender Aufteilung von Training- und Test-Daten optimiert. Zudem wurde eine Erkennung von Overfitting im Trainingsskript ergänzt und das Speichern des besten Modells optimiert. Außerdem wurde das Training so angepasst, dass es auf performanter Nvidia-Hardware via dem

CUDA Toolkit laufen kann. Nach der Verbesserung läuft das Training deutlich effizienter. Das Training auf performanten GPU-Chips verringert die Trainingszeit um fast das 32-fache.

Weiterhin ließ sich nachweisen, dass sich der von dem vorherigen Validierungsskript ermittelte Accuracy-Wert sich mit denen des verbesserten Skripts deckt. Mit einer Accuracy von $\approx 52\%$, selbst mit verbessertem Trainingsskript, ist das Modell jedoch nicht zufriedenstellend.

Der Code dieses Kapitels wurde auf einem nicht-öffentlichen Gitlab Git Repository, welches unter einer Domain der Hochschule Bremen verfügbar ist, gepflegt. Es ist geplant, diesen zu veröffentlichen, wonach er über die Webseite von Jan Bredereke erreichbar sein soll (`https://homepages.on.hs-bremen.de/~jbredereke/de/index.html`).

Zuletzt wurden sich auch noch viele Gedanken zu dem Systemtest gemacht. Dieser wird nun nicht mehr als Systemtest, sondern als Blackbox Test des neuronalen Netzes im Einsatz auf dem FPGA umgesetzt. Hier werden lediglich Fehler in der Integration bzw. Synthetisierung untersucht. Das Testszenario, wie im [ACB+22, Anhang G] beschrieben, wurde als nicht sinnvoll gewertet. Hier wurde eine formalisierte Testmethode nach dem Standardvorgehen entwickelt. In der Umsetzung dieser werden noch kleinere Anpassungen an dem Programmcode vorausgesetzt. Da es besonders schwierig bis nicht möglich ist, eine *erwartete Ausgabe* eines neuronalen Netzes zu definieren, wird hier besonders der Vergleich zum Referenzwert des Testskripts betrachtet. Zusammenfassend kann man sagen, dass nach erfolgreichen durchlaufen des Tests man mit Sicherheit sagen kann, dass das neuronale Netz auf dem FPGA dieselbe Funktionalität wie das Netz auf *leistungsstarker Hardware* aufweist. Mit diesem Fortschritt sind wir sehr zufrieden.

3.8. Ausblick

Mögliche zukünftige Verbesserungen

Da das Testskript zeigt, dass das Modell mit einer Genauigkeit von etwa 55 % Bilder noch nicht zuverlässig klassifizieren kann, könnte eine Überarbeitung des Datensatzes das Ergebnis möglicherweise weiter verbessern. Daher sollte die Verwendung von mehr Trainings-Daten für eine bessere Modell-Performance evaluiert werden. Um das Training noch mehr zu verbessern, könnte zudem Cross-Validation integriert werden. Aktuell wird für das Training ein Split aus Trainings- und Validierungsdaten verwendet, bei dem die Aufteilung für jede Epoche gleich ist. Wenn man diese Daten in jeder Epoche anders aufteilt, könnte dies eventuell zu einem besseren Trainingsergebnis führen.

Außerdem wird aktuell nicht die neueste Brevitas-Version *0.12.0* (Stand 24.07.2025) genutzt. Die derzeitige Umsetzung des FINN Exports ist allerdings nicht mit der neuesten Brevitas-Version kompatibel. Eine sinnvolle Verbesserung wäre ein Update der Brevitas-Version und Anpassung des FINN Exports an diese Version.

Sollten alle Rahmenbedingungen hinsichtlich der Testfolge des Blackboxtests getroffen sein, kann auch diese durchgeführt werden. Hier müssen vorher noch ggf. die genannten Anpassungen der Base Station getätigt werden.

4. Gestenerkennung

geschrieben von Niklas Otten, Marvin Lünswilken, Jonas Kleimann, Nils Karsten, Tim Jaeschke

4.1. Einführung und Problembeschreibung

Das Folgende Kapitel dokumentiert die Arbeiten und Erkenntnisse der Projektgruppe im Kontext der hardwarebeschleunigten Gestenerkennung auf Embedded-Systemen. Das übergeordnete Teilziel dieser Gruppe war die Inbetriebnahme der Gestenerkennung auf der neuen PYNQ-Z2 Entwicklungsplatine, wobei die Personenerkennung als notwendige Vorstufe ebenfalls berücksichtigt werden sollte. Die Realisierung eines vollständigen Workflows, vom Training neuronaler Netze bis zu deren effizienter Ausführung auf einem FPGA-basierten Board, erfordert jedoch eine äußerst komplexe und spezifische Toolchain. Im Laufe des Projekts wurde schnell deutlich, dass die primäre Herausforderung nicht in der direkten Integration eines bereits voll funktionsfähigen Systems lag. Vielmehr konzentrierte sich die Arbeit auf die Reevaluation der existierenden Toolchain und zugehöriger Ansätze, da sich ehemalige Lösungen und Anleitungen als veraltet, inkompatibel oder fehlerhaft mit aktuellen Softwareversionen und der neuen Hardware erwiesen. Dies führte dazu, dass der Schwerpunkt der Arbeit auf die Sichtung, Modifikation und Ergänzung dieser bestehenden Komponenten und Prozesse verlagert wurde, um eine lauffähige und wartbare Entwicklungsumgebung zu etablieren. Tatsächlich konnten aufgrund dieser grundlegenden Hürden noch keine abschließende Integration auf dem PYNQ-Board erfolgen.

4.2. Systemübersicht Personen-/Gestenerkennung

Phase 1: Datensatz-Erstellung

In der ersten Phase wird die Grundlage für das Training zweier spezialisierter neuronaler Netzwerke geschaffen. Ziel ist es, einen Trainingsdatensatz für ein Personenklassifikationsnetz sowie einen separaten Datensatz für ein Gestenklassifikationsnetz zu generieren. Die Erstellung der Datensätze erfolgt in zwei unabhängig voneinander aufgebauten Datenverarbeitungspfaden.

Für das Personenklassifikationsnetz wird der COCO-Datensatz als Ausgangsbasis herangezogen. Mittels einer manuellen Vorselektion in Adobe Lightroom werden geeignete Bilder identifiziert, welche ausschließlich erkennbare Personen enthalten. Diese Auswahl unterliegt anschließend einer automatisierten Verarbeitung, bei der die Bildbereiche zugeschnitten und in standardisierte Kachelgrößen von 32×32 Pixel unterteilt werden. Das Resultat ist ein normierter Trainingsdatensatz mit hohem Wiedererkennungswert für Personenlokalisierung.

Im zweiten Verarbeitungspfad wird ein eigenständig aufgenommener Bilddatensatz verwendet. Dieser enthält statische Kamerabilder, die manuell in drei Gestenklassen kategorisiert werden: `START`, `STOP` und `NOPOSE`. Nach der Klassifizierung erfolgt eine automatisierte Bildverarbeitung zur Vereinheitlichung hinsichtlich Skalierung und Ausschnitt. Das Ergebnis ist ein robuster Trainingsdatensatz für die Gestenerkennung.

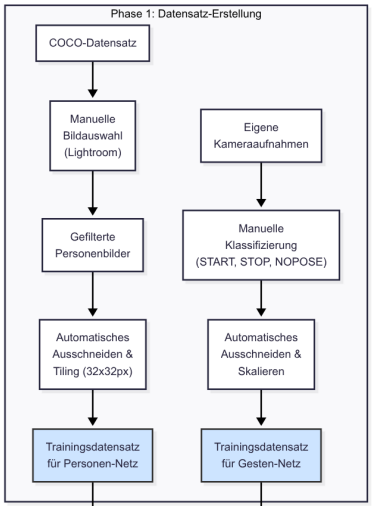


Abbildung 4.1.: Phase 1 – Datensatz-Erstellung

Phase 2: Training

Die zweite Phase umfasst das Training zweier neuronaler Netze auf den vorbereiteten Datensätzen. Beide Netze werden mit PyTorch entwickelt und unter Verwendung des Brevitas-Frameworks quantisiert. Dies ermöglicht eine spätere effiziente Umsetzung auf FPGA-Hardware.
Das Personenklassifikationsnetz wird darauf trainiert, Personen in vollständigen Bildern zu identifizieren und entsprechende Bounding Boxes auszugeben. Es operiert somit auf dem gesamten Kamerabild. Im Gegensatz dazu wird das Gestenklassifikationsnetz auf die Erkennung der drei Posen trainiert und verarbeitet ausschließlich auf die Person zugeschnittene Bildbereiche (Regions of Interest, RoIs).
Nach Abschluss des Trainingsprozesses werden beide Modelle in das ONNX-Format exportiert, welches sich als standardisierte Repräsentation neuronaler Netzwerke zur Weiterverarbeitung in hardwareorientierten Toolchains anbietet. Die resultierenden Modellartefakte `Personen-Netz.onnx` und `Gesten-Netz.onnx` bilden die Eingabe für die nachfolgende Hardware-Synthese.

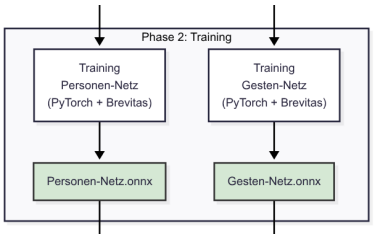


Abbildung 4.2.: Phase 2 – Training

Phase 3: Hardware-Synthese

In Phase drei erfolgt die Konvertierung der trainierten Netzwerke in synthetisierbare Hardwarebeschreibungen. Dazu wird die FINN-Toolchain verwendet, welche eine Transformation quantisierter ONNX-Modelle in FPGA-kompatible IP-Blöcke erlaubt.
Für jedes der beiden Netzwerke wird eine eigene FINN-Verarbeitungspipeline definiert. Diese generiert unabhängig voneinander IP-Blöcke für das Personen- sowie das Gestennetz. Anschließend werden beide Blöcke in ein gemeinsames Vivado-Projekt integriert. Dieses Projekt dient als Grundlage für die logische Synthese sowie den sogenannten Place-and-Route-Prozess.
Das Ergebnis dieser Phase ist ein vollständiger Bitstream (`Combined.bit`) zur Programmierung des Ziel-FPGAs sowie eine zugehörige Hardwarebeschreibung (`Combined.hwh`), die unter anderem die Speicheradressen und Steuerregister der integrierten IP-Blöcke definiert.

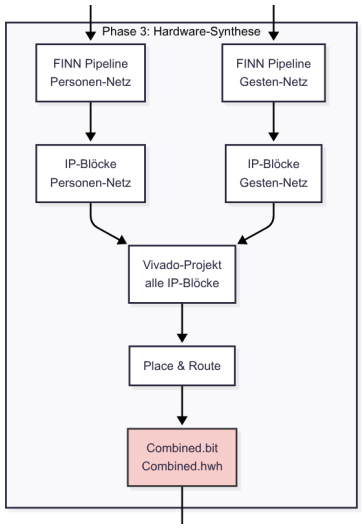


Abbildung 4.3.: Phase 3 – Hardware-Synthese

Phase 4: Deployment und Inferenz

Die vierte Phase umfasst das Deployment des synthetisierten Designs auf der Zielplattform sowie die Ausführung der Inferenz in einer realen Anwendungssituation. Ziel ist es, die im vorherigen Schritt erzeugte FPGA-Bitstream-Datei erfolgreich auf die Zielhardware zu übertragen, in Betrieb zu nehmen und die vollständige Bildverarbeitungskette in Echtzeit zu testen.
Zunächst wird der Bitstream über eine Netzwerkschnittstelle auf die PYNQ-Z2-Plattform übertragen und dort in das FPGA geladen. Die initiale Inbetriebnahme erfolgt dabei über eine auf der ARM-CPU laufende Python-Umgebung, welche unter anderem die Steuerung des FPGA-Overlays sowie den Zugriff auf angeschlossene Peripherie übernimmt.
Eine via USB angeschlossene Kamera liefert kontinuierlich Bilddaten, die durch die ARM-CPU zunächst vorverarbeitet werden (z. B. Skalierung, Farbraumkonvertierung, Tiling). Diese Daten werden dann an das im FPGA implementierte Personenklassifikationsnetz weitergereicht. Dieses erkennt, ob sich eine Person im Bild befindet, und gibt im Falle einer erfolgreichen Detektion eine sogenannte Bounding Box zurück – ein rechteckiger Bereich, der das erkannte Objekt (in diesem Fall: eine Person) umschließt und somit eine Region of Interest (RoI) definiert.
Die ARM-CPU extrahiert anschließend den entsprechenden Bildausschnitt basierend auf den Koordinaten der Bounding Box aus dem ursprünglichen Kamerabild. Dieser Ausschnitt dient als Eingabe für das Gestenklassifikationsnetz, das ebenfalls auf dem FPGA realisiert ist. Das Netz analysiert die innerhalb der RoI enthaltenen Bildinformationen und gibt als Ergebnis eine Gestenklasse zurück, `START`, `STOP` oder `NOPOSE`.
Die erkannte Gestenklasse wird im nächsten Schritt zur Steuerung eines Fahrzeugsystems verwendet. Die Fahrsteuerung entscheidet, wie mit der erkannten Gestenklasse und der übertragenen Position verfahren wird. Je nach Systemdesign kann dies entweder durch eine in Software implementierte Logik auf der ARM-CPU erfolgen oder durch eine dedizierte Steuerlogik direkt im FPGA. In beiden Fällen resultiert die Klassifikation in konkreten Steuerbefehlen für die Motoren des Fahrzeugs.

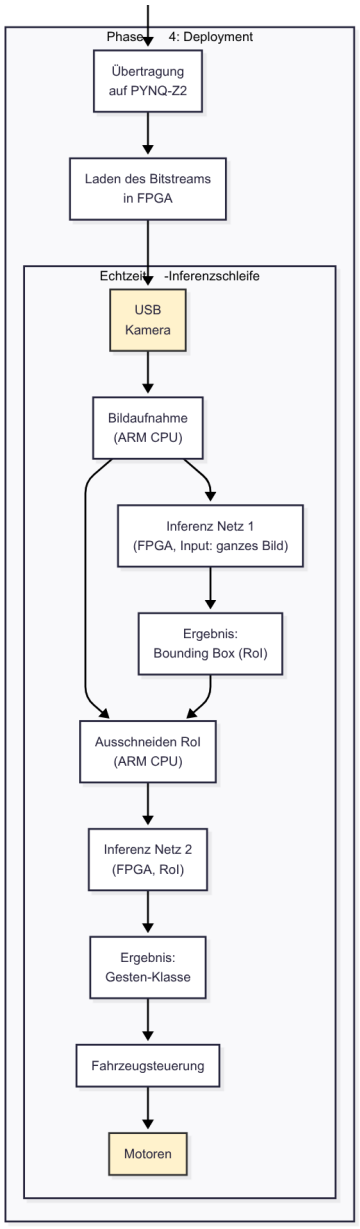


Abbildung 4.4.: Phase 4 – Deployment und Inferenz

4.3. Bisheriger Stand und identifizierte Probleme

4.3.1. Datengrundlage

Die Grundlage für das Training der Modelle basiert auf den Arbeiten von Projektgruppen aus den Vorjahren. Diese haben ein Kategorienschema für Gesten entwickelt, in dem drei Gestentypen

unterschieden werden: START, STOP und NOPOSE. Die vorhandenen, bereits annotierten Daten ermöglichten ein gezieltes Training zur Unterscheidung dieser Kategorien.

Für die Personenklassifikation wurde der COCO-Datensatz [Lin+14] verwendet, der zum Stand Juli 2025 unter `https://cocodataset.org/#download` frei verfügbar war. Der Download und die Vorverarbeitung verliefen ohne technische Probleme.

4.3.2. Ressourcenlimit des FPGA & Optimierungsstrategien

Die meisten neuronalen Netze überschreiten ohne weitere Anpassungen die auf dem PYNQ-Z2 verfügbaren Ressourcen (LUT/BRAM/DSP) bzw. erfüllen die Echtzeitanforderungen nicht. Eine native Portierung ist daher nicht möglich. Es braucht gezielte Maßnahmen zur Reduktion der Netz- und Datenkomplexität. Folgende weitestgehend unabhängige Lösungsansätze lassen sich aus den Ergebnissen der Vorgängerprojekte ableiten.

- **Vorverarbeitung (auf CPU)**
  - **Downsampling**: Reduktion der Bildauflösung.
  - **Region of Interest (ROI)**: Vorauswahl relevanter Bildbereiche durch einfache Heuristiken/Algorithmen; nur diese werden dem NN zugeführt.
  - **Tiling**: Aufteilung des Bildbereichs in schrittweise kleinerwerdende Kacheln und sequentielle Klassifikation.
- **Anpassung der NN-/Toolchain-Architektur**
  - **Quantisierung**: Verwendung von Few-bit-Integer-Gewichten/Aktivierungen; drastische Ressourceneinsparung bei moderatem Genauigkeitsverlust.
  - **Folding**: Serielle Ausführung von Operationen eines Layers zur Reduktion der benötigten Parallelressourcen . Siehe hierzu die FINN-Dokumentation [AMD24]

4.3.3. Training

Beim Versuch, den Trainingscode der vorigen Projektgruppen auszuführen, traten Kompatibilitätsprobleme aufgrund veralteter oder unvollständig dokumentierter Paketabhängigkeiten auf. Insbesondere die Brevitas-Integration führte zu Inkompatibilitäten mit aktuellen Versionen von PyTorch und ONNX. Sämtliche Versuche, den Code durch Anpassung der Abhängigkeiten lauffähig zu machen, scheiterten an unzureichender Dokumentation sowie fehlender Verfügbarkeit stabiler Beispiele. Kenntnisse im Umgang mit Brevitas sowie der Netzarchitektur waren zur Fehleranalyse nicht in ausreichendem Maße vorhanden(siehe auch 3).

Die ursprünglich vorgeschlagenen Methoden zur Durchführung des Trainings in der Google Cloud sind in ihrer früheren Form nicht mehr verfügbar. Alternative Cloud-Plattformen (z.B. AWS, Azure) bieten weiterhin GPU-Ressourcen, jedoch übersteigen deren Nutzungskosten den finanziellen Rahmen unseres Projekts.

Für das lokale Training wird daher empfohlen, auf leistungsfähige Endgeräte zurückzugreifen. Dabei kann das Training grundsätzlich auf der CPU erfolgen, ist jedoch mit hohen Laufzeiten verbunden. Für praktikable Trainingszeiten ist der Einsatz einer dedizierten NVIDIA-GPU empfehlenswert, wie sie in vielen Notebooks und Desktop-PCs zu finden ist.

4.3.4. Hardwaresynthese

Die in früheren Projektphasen verwendeten Anleitungen zur Nutzung der FINN-Toolchain für die Generierung von Bitstream- und HWH-Dateien sind nicht mehr aktuell. Durch Änderungen in der FINN-Toolchain und der zugrundeliegenden Abhängigkeiten war eine direkte Reproduktion der früheren Vorgehensweise nicht mehr möglich.

Die Installation und Nutzung von FINN erfordert tiefgehende Kenntnisse der verwendeten Frameworks (ONNX, Pytorch Quantisierung, Vivado HLS). Aufgrund der gestiegenen Komplexität widmet sich Kapitel 4.4.1 explizit der aktualisierten Vorgehensweise zur Benutzung von FINN zum Stand Juli 2025.

4.3.5. Deployment

Die bestehenden Anleitungen zum Deployment auf dem PYNQ-Z1-Board konnten nicht erfolgreich auf die aktuelle Z2-Plattform übertragen werden. Für einen erfolgreichen Funktionstest muss entweder das Gesamtsystem fertiggestellt sein oder eine softwareseitige Vorbereitung des Boards(PYNQ-Image Installation) erfolgen.

Besonders kritisch war die eingeschränkte Nutzbarkeit des Ethernet-Ports, welcher für die Kommunikation mit dem Hostsystem erforderlich ist. In früheren Konfigurationen war dieser Port teilweise durch Gehäusebauteile blockiert, was die Verbindung verhinderte.

4.3.6. Integration

Aufgrund der noch ausstehenden funktionalen Implementierung der Gesamtpipeline sowie der unvollständigen FPGA-Konfiguration konnten bislang keine erfolgreichen Tests auf physikalischer Hardware durchgeführt werden.

4.4. FINN Toolchain

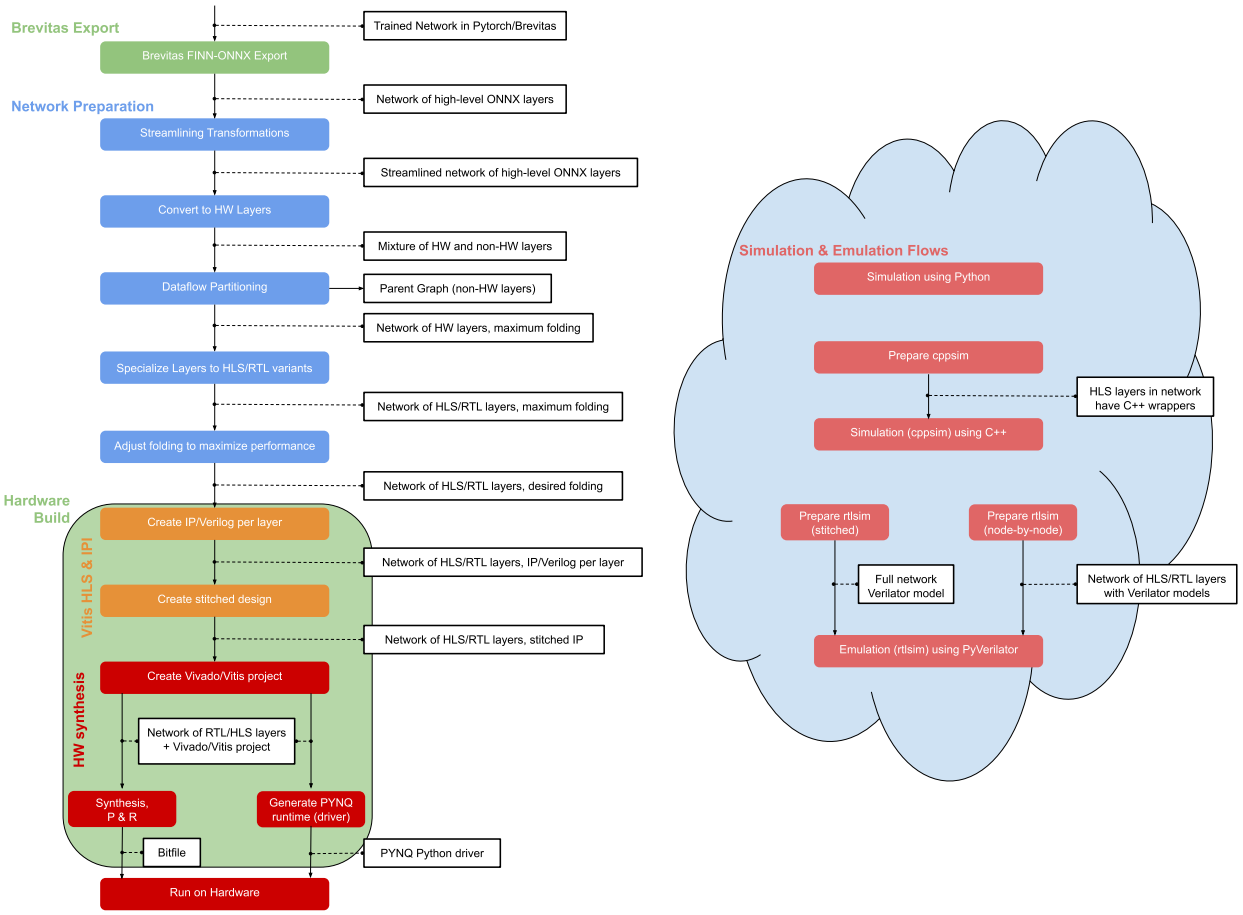


Abbildung 4.5.: FINN Build-Prozess [AMD24]

Um die trainierten neuronalen Netze in ein Hardware-Design für unsere Zielplattform (PYNQ-Z2 Board) zu überführen, nutzen wir den von Umuroglu u. a. bei Xilinx Research (heute AMD Research and Advanced Development) entwickelten FINN Compiler [Umu+17] [Blo+18]. FINN ist ein quelloffenes experimentelles Framework zur Exploration und Implementation von FPGA-Beschleunigern für quantisierte neuronale Netze. Es bündelt alle Schritte – von Netzoptimierungen bis hin zur finalen Synthese – in einer anpassbaren Build-Pipeline. Standardmäßig werden dabei 19 Schritte durchlaufen. Diese im Detail zu verstehen ist für das Arbeiten mit FINN keine Voraussetzung, aber es lohnt sich ein Blick auf die wesentlichen Abläufe, um die Build-Konfiguration ggf. anzupassen und die Ausgaben des Compilers nachzuvollziehen (vgl. Abbildung 4.5 links und [AMD24]).

1. **Modellimport:** Als Eingabe erwartet FINN ein trainiertes Netz im ONNX- bzw. QONNX-Format. Dieses wird klassischerweise mit Brevitas erstellt. Das Modell wird im ersten Schritt geladen und in eine interne Darstellung umgewandelt. Vor der weiteren Verarbeitung werden außerdem Datentypen und Dimensionen der Schichten abgeleitet, Konstanten extrahiert und eindeutige Namen vergeben.

2. **Netzoptimierung:** Durch verschiedene Graphenoperationen und algebraische Umformungen wird die Struktur des Netzes vereinfacht und Fließkommazahloperationen eliminiert. Geeignete Schichten werden in Hardware Abstraction Layers konvertiert, der Graph wird in

Hardware- und Software-Anteile partitioniert, die Hardware Abstraction Layers in HLS bzw. RTL Varianten konvertiert und anschließend das Maß an Parallelisierung angepasst (Stichwort Folding), um die Geschwindigkeitsziele (z. B. Target FPS) zu erreichen. Anschließend wird ein Estimate Report erstellt, in dem verschiedene Performanceparameter der späteren Implementierung geschätzt und mit alternativen Konfigurationen verglichen werden.[1]

3. **Hardware Implementation:** Zur Implementation wird pro Schicht ein RTL Template bzw. Vitis HLS Code generiert und anschließend synthetisiert, um IP-Blöcke zu erhalten. Außerdem werden die Schnittstellen nach außen (DMA Knoten für Ein-/Ausgaben) und innen (FIFOs und Bitbreitenkonvertierung zwischen Schichten) angepasst. Das Ergebnis ist ein IP-Block-Design in Vivado.
4. **Performance Analyse:** Auf Basis des stitched IP Designs werden im Anschluss einige Performancemetriken für die Hardwareimplementation aggregiert und in einem Bericht im JSON-Format gespeichert.
5. **Plattform Integration:** Der letzte Schritt ist die Vorbereitung der Hardwareimplementation für die jeweilige Zielplattform, in unserem Fall ein PYNQ-Z2 Board. Dazu gehören die vollständige Synthese und Generierung eines Bitfiles für den FPGA-Chip, sowie der Treiber für das PYNQ Board. Der Treiber hat die Aufgabe den Ein- und Ausgabedatenstrom zu koordinieren und damit den Hardwarebeschleuniger zu bedienen.

Welche Schritte im Detail durchlaufen werden, lässt sich durch eine `DataflowBuildConfig` aus dem Modul `finn.builder.build_dataflow_config` bestimmen. Auch lassen sich damit die erzeugten Ausgaben anpassen. Für uns ist bspw. das fertige Bitfile unbrauchbar, da das neuronale Netz für die Gestenerkennung zuerst mit weiteren Komponenten im IP-Design integriert werden muss, etwa dem Netz für die Personenerkennung. FINN bietet außerdem einige Möglichkeiten die Zwischenergebnisse an verschiedenen Stellen zu simulieren bzw. zu testen (vgl. Abbildung 4.5 rechts).

4.4.1. Einrichtung von FINN

Ziel war die Einrichtung einer lauffähigen Toolchain für das FINN-Framework, einschließlich der Entwicklungsumgebungen *Vivado* und *Vitis*. Hintergrund war die Migration der im Wintersemester 2021 genutzten Umgebung auf aktuelle Softwarestände und Systeme. Damit soll die langfristige Wartbarkeit sichergestellt und der Einsatz moderner Plattformversionen ermöglicht werden.

Auswahl der Systemumgebung Da sämtliche Projektteilnehmende auf Windows-Systemen arbeiten, wurde ein Installationsweg gesucht, der keine Linux-Installation oder vollständige Virtualisierung über klassische VMs erfordert. Die Wahl fiel auf das Windows Subsystem for Linux Version 2 (WSL2). WSL2 stellt einen vollwertigen Linux-Kernel bereit und erlaubt direkten Zugriff auf Systemressourcen bei gleichzeitig hoher Integration in die Windows-Umgebung. Im Vergleich zu VMs ist WSL2 ressourcenschonender und bietet bessere Dateisystemkompatibilität.

Als Linux-Distribution wurde **Ubuntu 24.04 LTS** verwendet. Die Wahl begründet sich durch langfristigen Support (LTS), Aktualität der enthaltenen Systembibliotheken und Kompatibilität mit moderner Entwicklungssoftware.

[1] Diese Schritte könnten ausreichen, um eine erste Einschätzung für die Synthetisierbarkeit des Netzes abzugeben. Somit spart man sich in der Exploration u. U. viel Zeit, da noch keine Synthese nötig ist.

Evaluation der Windows-basierten Installationen Im ersten Ansatz wurde versucht, bestehende Windows-Installationen von Vivado und Vitis für die Ausführung von FINN zu verwenden. Dazu wurden entsprechende Verzeichnisse unter WSL eingebunden. Diese Methode erwies sich als unzuverlässig:

- WSL kann Windows-Pfade nicht transparent in Linux-konforme Strukturen abbilden.
- Symbolische Links und Umgebungsvariablen werden inkonsistent aufgelöst.

Die Verwendung Windows-basierter Installationen von Vivado und Vitis in Kombination mit WSL wurde daraufhin verworfen.

Native Installation unter WSL2 Zur Sicherstellung konsistenter Dateipfade und Systemaufrufe wurde Vivado und Vitis nativ unter WSL2 installiert. Die Installation erfolgte in das standardisierte Verzeichnis `/opt/Xilinx`. Die Einbindung der Tools erfolgte über `settings64.sh` und Erweiterung der `PATH`-Variable in `.bashrc`.

Kompatibilitätsprobleme mit aktueller Vitis-Version Die initial installierte Version **Vitis 2025.1** enthielt strukturelle Änderungen gegenüber vorherigen Versionen. Insbesondere das Tool `vitis_hls` war nicht wie erwartet im Standardpfad vorhanden, sondern in einem verschachtelten Unterverzeichnis abgelegt. FINN konnte dieses Tool daher nicht automatisch finden. Ein manueller Workaround über das explizite Setzen des Pfads in der Laufzeitumgebung erwies sich als instabil, da in Scripten genutze Referenzen nicht mehr aufgelöst werden konnten.

Verwendung der Version 2024.2 Zur Sicherstellung der Kompatibilität wurde **Vitis 2024.2** installiert. Diese Version entsprach der von FINN getesteten Toolchain und stellte das benötigte HLS-Tool an der erwarteten Stelle bereit. Erst mit dieser Konfiguration war eine fehlerfreie Ausführung der FINN-Komponenten möglich.

Ergebnis Die final funktionierende Konfiguration basiert auf:

- Ubuntu 24.04 LTS unter WSL2
- Native Installation von Vivado und Vitis (Version 2024.2) innerhalb WSL2
- Konfigurierten Umgebungsvariablen in `.bashrc`

Die vollständige, schrittweise Installationsanleitung ist in Anhang E dokumentiert.

Ergänzung Wer keine Windows-Installation besitzt, kann sich auch direkt nativ Ubuntu 24.04 LTS installieren. Bei anderen Linux-Distributionen kann es zu Problemen kommen, dies wurde aber nicht weiter untersucht.

4.4.2. Weiterführende Links zur FINN-Toolchain

Im Folgenden sind zentrale Einstiegspunkte, Dokumentationen und Beispiele zur Arbeit mit dem FINN-Framework aufgelistet. Diese Ressourcen unterstützen bei der Vertiefung der einzelnen Schritte im End-to-End Flow sowie bei der praktischen Umsetzung auf FPGAs:

Thema	Link	Beschreibung
Architektur im Detail	Internals – FINN Documentation	Technische Details zu FINNs ONNX-basiertem IR und internen Komponenten.
Überblick über den Flow	End-to-End Flow – FINN Documentation	Allgemeine Beschreibung des vollständigen Build-Flows von Training bis FPGA.
Build-Modi	Command Line Entry – FINN	Erklärung der einfachen und erweiterten Modi für den Dataflow-Build.
Parameterübersicht	Builder – FINN Documentation	Beschreibung der verfügbaren Parameter für den Build-Prozess.
End-to-End Notebook	End2End Example (GitHub)	Konkretes Jupyter-Beispiel zur Durchführung eines vollständigen Builds mit FINN.

Tabelle 4.1.: Übersicht relevanter FINN-Dokumentationsquellen

4.5. Yolo („You Only Look Once“)

YOLO („You Only Look Once“) ist eine Familie von Objekterkennungs-Netzwerken, die durch hohe Geschwindigkeit und Effizienz überzeugt. Seit der ersten Veröffentlichung als arXiv-Preprint 2015 [Red+15] hat sich YOLO dank verschiedener Forschungsgruppen über YOLOv1 bis YOLOv12 stetig weiterentwickelt, wobei jede Version die Genauigkeit verbessert und die Latenz verringert. Im Unterschied zu zweistufigen Detektoren wie Faster R-CNN erledigt YOLO Bounding-Box-Vorhersage und Objektklassifikation in einem Durchlauf, ideal für Anwendungen, die niedrige Reaktionszeiten verlangen, etwa autonome Fahrzeuge, Live-Videoüberwachung oder Augmented-Reality-Apps.

Als einstufiger Detektor bietet YOLO ein hervorragendes Verhältnis von Erkennungsgenauigkeit zu Ausführungszeit, ideal für den Einsatz auf ressourcenbegrenzten Plattformen. Vortrainierte Gewichte, stabile Exporpfade (z.B. ONNX) und etablierte Quantisierungsverfahren erleichtern die Portierung auf FPGA-Hardware erheblich. Insbesondere kompakte YOLO-Varianten erfüllen die Anforderungen unseres LUT- und DSP-Budgets und sind Edge-tauglich.

4.5.1. Training

Um möglichst schnell ein funktionierendes Netzwerk zu erhalten, wurde ein von Ultralytics bereitgestelltes vortrainiertes Model auf den vorhandenen Trainingsdaten der zu erkennenden Körperhaltungen trainiert. Man spricht bei diesem Vorgehen auch von Fine-Tuning oder Transferlernen [Ult] [GBC16, S. 536]. Mit dieser Vorgehensweise konnte nach weniger als 10 Trainingsdurchläufen eine Trefferquote von 100 % in dem Validierungsdatensatz erreicht werden. Um Fehler in der Validierung auszuschließen, wurde ein Script geschrieben, welches mittels OpenCV das Bild einer Webcam ausgibt, zusammen mit der erkannten Körperhaltung des Netzwerks. Hierbei konnten keine Fehler festgestellt werden. Um das Model weiter in der Größe zu reduzieren, wurde es mit-

tels onnxruntime quantisiert. Dieses quantisierte Model hatte weiterhin eine Genauigkeit von über 96 % bei einem Viertel der ursprünglichen Größe.

4.5.2. Deployment

Beim Versuch, ein YOLO-Modell mit FINN auf einem FPGA zu deployen, treten häufig Probleme während des sogenannten Streamline-Schritts auf, einer Phase, in der das ONNX-Modell vereinfacht und für die spätere Hardware-Synthese vorbereitet wird. Zudem treten solche Fehler teilweise erst nach 25 Minuten auf, da ein kompletter FINN-Durchlauf extrem lange dauern kann.

Ein Problem betrifft dabei die ONNX-Operationen Mul und MatMul. Aktuell unterstützt FINN diese nur, wenn einer der beiden Eingabeparameter konstant ist. Sobald jedoch beide Eingaben dynamisch sind – also nicht zur Compile-Zeit bekannt, schlägt der Streamline-Schritt fehl. Diese Einschränkung stellt insbesondere bei komplexeren Modellen wie YOLO ein wesentliches Hindernis dar, da solche dynamischen Operationen dort häufig verwendet werden. Es gibt zwei Möglichkeiten, dieses Problem zu umgehen:

- **Modellanpassung vor dem FINN-Import:** Das Modell kann vor der Verarbeitung mit FINN so angepasst werden, dass problematische Mul- oder MatMul-Knoten durch Varianten mit konstantem Operand ersetzt oder durch funktional äquivalente Konstrukte ohne dynamische Multiplikation ausgetauscht werden. Dies erfordert eine gezielte Analyse und Transformation des ONNX-Modells.
- **Implementierung einer benutzerdefinierten HLS-Komponente:** Alternativ bietet FINN die Möglichkeit, eigene Hardwaremodule über High-Level Synthesis (HLS) zu integrieren. Man kann eine benutzerdefinierte HLS-Komponente entwickeln, die dynamische Mul- oder MatMul-Operationen unterstützt, und diese dann im FINN-Workflow einbinden. Dies erfordert jedoch gute Kenntnisse in HLS (z.B. mit Vitis HLS oder Vivado HLS) sowie eine entsprechende Anpassung des FINN-Buildprozesses, um die neue Komponente korrekt zu integrieren.

4.5.3. Alternative Deployment-Möglichkeiten für YOLO mit FINN

Eine weitere Möglichkeit, YOLO mit FINN auf einem FPGA lauffähig zu machen, bieten Okcu und Pollo im GitHub-Repository quantized-yolov5 [OP24]. In diesem Repository wird ein modifiziertes Trainings-Framework bereitgestellt, das es erlaubt, verschiedene Varianten von YOLO konkret YOLOv1, YOLOv3 und YOLOv5, mit quantization-aware training (QAT) zu trainieren. Beim QAT-Verfahren werden die Auswirkungen der Quantisierung bereits während des Trainings berücksichtigt, wodurch sich die Genauigkeit des quantisierten Modells im Vergleich zur nachträglichen Quantisierung (Post-Training Quantization, PTQ) deutlich verbessern kann. Das Besondere an diesem Ansatz ist, dass die erzeugten Modelle explizit auf Kompatibilität mit FINN ausgelegt sind. In der Projektbeschreibung wird erwähnt, dass die quantisierten YOLO-Modelle so strukturiert und optimiert wurden, dass sie sich in den FINN-Workflow integrieren lassen unter anderem durch angepasste Netzarchitektur und entsprechende Exportformate. Dadurch lassen sich viele der typischen Probleme (z.B. dynamische Mul/MatMul-Operationen oder inkompatible Layerstrukturen) vermeiden, die sonst bei der Umwandlung und Synthese mit FINN auftreten. Diese Lösung stellt insbesondere für Anwendungen mit YOLOv5 eine vielversprechende Grundlage dar, da sie ein bereits getestetes und offenes Framework für das Training und die Quantisierung bereitstellt.

4.6. Alternative Ansätze

4.6.1. Vitis AI

Neben den bisher betrachteten Ansätzen stellt Vitis AI eine vielversprechende Alternative für die KI-Inferenz auf Embedded- und Edge-Geräten dar. Die Plattform wurde von AMD Xilinx speziell für die effiziente Ausführung neuronaler Netze auf FPGAs und adaptiven SoCs (z.B. Zynq UltraScale+) entwickelt. Ein zentraler Vorteil von Vitis AI liegt in der Möglichkeit, quantisierte KI-Modelle direkt auf Hardwareebene auszuführen. Bestehende Modelle, die mit Frameworks wie TensorFlow, PyTorch oder ONNX erstellt wurden, lassen sich mithilfe unterstützter Toolchains in Vitis AI integrieren und für die jeweilige Zielhardware optimieren. Im Gegensatz zu FINN, das für eine individuell zugeschnittene Hardwarearchitektur für jedes Modell sorgt (dataflow-orientiert), erzeugt Vitis AI eine allgemeine, wiederverwendbare Hardwarearchitektur, die mehrere Modelle oder Modellvarianten ausführen kann. Dies bietet mehr Flexibilität und verkürzt die Entwicklungszeit.

4.6.2. Pose Landmark Detection

Ein ebenfalls vielversprechender Alternativansatz zu der vorgestellten Gestenerkennung durch Bildklassifikatoren ist die Gestenerkennung auf Basis sog. Keypoints. Dabei werden zunächst bestimmte Körperstellen der Person identifiziert (die Keypoints) und auf Grundlage der Position dieser Schlüsselstellen eine Vorhersage über die Pose der Person getroffen.

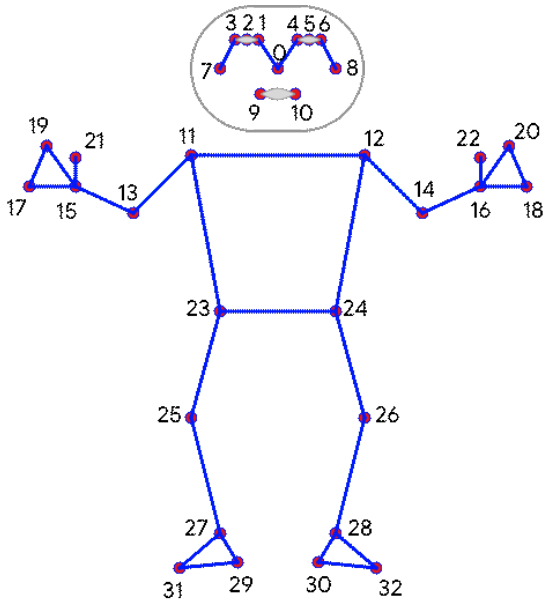


Abbildung 4.6.: BlazePose Keypoints Topologie [Baz+20, Fig. 3]

Abbildung 4.7.: Beispiel einer Keypoint-Extraktion für ein Bild aus dem Gestendatensatz

Im Rahmen des Sommersemesters 2025 wurde zu Demonstrationszwecken ein einfacher Pose Detection Flow erstellt. Die Ergebnisse sind im GitLab Repository *Gestenerkennung* im Verzeichnis *blazepose* hochgeladen. Zur Keypoint-Extraktion wird ein Blazepose-Modell [Baz+20] verwendet, das von Google als Mediapipe Solution [Goob] zur Verfügung gestellt wird. Blazepose extrahiert insgesamt 33 Keypoints wie in Abbildung 4.6 veranschaulicht. Abbildung 4.7 zeigt ein Beispiel-

Foto aus dem Gestendatensatz mit überlagerten Keypoints.

Die Keypoint-Daten werden im nächsten Schritt durch einen einfachen Klassifikator, z. B. einen Entscheidungsbaum oder eine Stützvektormaschine als eine der drei Posen START, STOP, NOPOSE klassifiziert. Um die Präzision der Klassifikatoren zu verbessern, werden die Keypoints zunächst in aussagekräftigere Merkmale transformiert (Stichwort Feature Engineering). Da für die o. g. Posen in erster Linie die Haltung der Arme relevant ist, werden aus den Keypoints für Hüften, Schultern, Ellenbogen und Handgelenke mittels geometrischer Verfahren die Armhebe- und Armbeugungswinkel berechnet. Abbildung 4.8 zeigt, dass sich auf diese Weise schon deutlich erkennbare Gruppierungen in den Daten bilden lassen, die zum Großteil mit den annotierten Klassen übereinstimmen.

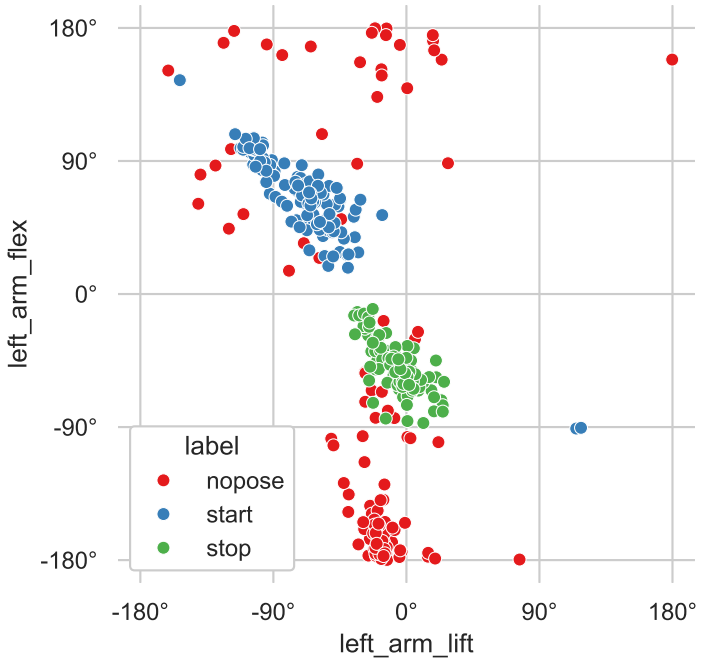


Abbildung 4.8.: Scatterplot der Armhebe- und Armbeugungswinkel (linker Arm) für alle Bilder des Gestendatensatzes

Die dennoch vorhandenen Ausreißer deuten auf Widersprüche in den Gruppenzugehörigkeiten hin, die im Zuge einer genaueren Analyse vor allem auf drei Ursachen zurückgeführt werden konnten.

- Einige Punkte der Klasse NOPOSE liegen nah bei bzw. innnerhalb der vermuteten Clustern für die START- oder STOP-Posen. Dies liegt an der Auswahl der in Abbildung 4.8 dargestellten Merkmale. Wenn die Haltung des linken Arms dem Muster für eine bestimmte Pose entspricht, dann liegt der Punkt bei den anderen Punkten dieser Pose, ungeachtet der Haltung des rechten Arms. Da der rechte Arm in der Visualisierung nicht betrachtet wird, kann eine asymmetrische Armhaltung dort einen falschen Eindruck vermitteln. Für die tatsächliche Klassifikation ist dies aber unproblematisch, weil der Klassifikator die Winkel beider Arme als Eingaben erhält.
- Die Hebe- und Beugungswinkel streuen je Pose sehr stark. Das lässt sich nur zu einem kleinen Teil auf die Variation der Armhaltung zurückführen. Erheblich größeren Einfluss auf die wahrgenommenen Gelenkwinkel hat die Drehung der Person zur Kamera. Steht

die Person nicht mit dem Körper/Gesicht zur Kamera, sondern gedreht, so erfahren die zweidimensionalen Keypoints eine Projektion, welche die berechneten Winkel verfälscht.

- Zwei besonders starke Ausreißer der Klasse START liegen im diagonal gegenüberliegenden Quadranten zu den restlichen Punkten ihrer Klasse. Es handelt sich dabei um Bilder, bei denen die Personen mit dem Rücken zur Kamera steht. Da in diesem Fall die Arme spiegelverkehrt orientiert sind, ist das Vorzeichen für beide Winkel umgekehrt zu einer Aufnahme von vorne.

Trotz dieser Probleme erreicht ein auf dem Trainingsdatensatz (368 Bilder) trainierter Entscheidungsbaum-Klassifikator eine Accuracy von 100 % für den Validierungsdatensatz (30 Bilder). Die Vermutung liegt nahe, dass dieser optimale Score unter anderem auf einen zu kleinen Datensatz und an einer zu geringen Variation unter den Bildern zurückzuführen ist. Deshalb wurde für einen Live-Test ein Python Skript entwickelt, dass den Klassifikator auf den Video-Feed einer Webcam anwendet. Auch hiermit wurden überraschend gute Ergebnisse erzielt, auch wenn noch einige Fehlklassifikationen zu erkennen sind, insbesondere bei Posen, die kaum/nicht im Trainingsdatensatz enthalten sind.

Integration Da es sich bei der Pose Landmark Detection aktuell noch um einen Demonstrator zur Exploration der Machbarkeit und Präzision dieses Ansatzes handelt, wurde der o. g. Klassifikator noch nicht in das Gesamtsystem integriert. Generell wäre die Integration allerdings ähnlich zum bisherigen Ansatz. Mithilfe des FINN Compilers würde ein BlazePose-Modell im ONNX Format (oder anderes Modell, siehe unten) in einen IP-Core umgewandelt und für das Board synthetisiert werden. Der nachgeschaltete Gestenklassifikator für die Keypoints könnte aufgrund der geringen Komplexität in Software implementiert werden.

Vorteile Der wesentliche Vorteil dieses Ansatzes liegt in der starken Abstraktion der eigentlichen Gestenerkennung. Anstatt die Gesten direkt in den Rohbildern zu klassifizieren, erfolgt die Klassifikation hier auf Basis weniger Schlüsselpositionen. Für die Keypoint-Extraktion (der komplexere der beiden Schritte) kann ein fertig trainiertes Netz verwendet werden. Das Training reduziert sich also auf den Gestenklassifikator. Diese Unterteilung beschleunigt das Training erheblich (wenige Sekunden mit dem o. g. Datensatz) und senkt somit auch die Anforderungen an den Umfang der Trainingsdaten.

Probleme und Grenzen des Verfahrens Die größte Herausforderung für das Verfahren ist die Verfälschung der gemessenen Winkel durch Projektion bei seitlicher Aufnahme der Person oder bei Aufnahmen von hinten.

Eine weitere Herausforderung ist möglicherweise die Synthese des BlazePose-Modells für das FPGA. Das Modell könnte unter Umständen wegen nicht unterstützter Knoten oder zu großer Komplexität nicht synthetisierbar sein. Hierzu liegen noch keine Untersuchungen vor.

Verbesserungen Zum Umgang mit o. g. Problem der seitlichen Betrachungen ergeben sich aktuell zwei Lösungsansätze. Zum einen ermöglicht BlazePose zusätzlich die Ausgabe geschätzter Tiefeninformationen (für monoskopische Aufnahmen), die theoretisch eine projektionsrobuste Betrachtung ermöglichen. Allerdings wurde die Präzision/Zuverlässigkeit dieser Informationen noch

nicht beurteilt. Die Alternative wäre, in der Spezifikation unter Angabe konkreter Betrachtungswinkel klarzustellen, dass eine genaue Gesten-Klassifikation nur bei einer Betrachtung von vorne gewährleistet ist.

Die Komplexität der Keypoint-Extraktion könnte durch den Einsatz anderer Modelle reduziert werden. Hier lohnt sich beispielsweise eine Beurteilung der Modelle Lightweight OpenPose [Oso18] und MoveNet [Gooc]. Beide sind für ressourcenbeschränkte Anwendungen und schnelle Inferenz optimiert und geben jeweils geben nur die 17 Keypoints des COCO Datensatzes [Lin+14] aus, was für unsere Anwendung völlig ausreicht.

4.6.3. Kombinierte Personen- und Gestenerkennung

Um visuelle Erkennung auf ressourcenarmen, eingebetteten Systemen wie FPGAs umzusetzen, ist genau abzuwägen, wie groß die verwendeten Netze werden können und wie viel Hardwareleistung zur Verfügung steht. Für die im Projekt geplante Aufgabe, einer Person automatisch zu folgen und sie per Gesten steuern zu können, bieten sich zwei grundlegende Ansätze beim Einsatz von neuronalen Netzen an:

1. **Kaskadierter Ansatz:** Zwei separate, spezialisierte Netzwerke, bei dem ein Personendetektor einem Gestenklassifikator vorgeschaltet ist.
2. **Integrierter Ansatz:** Ein einzelnes Multi-Task-Netzwerk, das beide Aufgaben – Detektion und Klassifikation – simultan durchführt.

Diese Evaluation analysiert die theoretischen Vor- und Nachteile beider Architekturen im Hinblick auf Trainingskomplexität, Inferenzleistung, Ressourcennutzung und Systemmodularität, um eine fundierte Grundlage für zukünftige Entwicklungsentscheidungen zu schaffen.

Analyse des kaskadierten Ansatzes (Zwei separate Netzwerke)

Dieser Ansatz implementiert eine zweistufige Verarbeitungspipeline. In der ersten Stufe analysiert ein schnelles und ressourcenarmes Netzwerk (Netz 1) das gesamte Kamerabild, um eine *Region of Interest (RoI)* zu lokalisieren, die eine Person enthält. Die Koordinaten dieser RoI werden an die CPU übergeben. Diese extrahiert den entsprechenden Bildausschnitt und leitet ihn an ein zweites, auf Details spezialisiertes Netzwerk (Netz 2) weiter, das die Gestenklassifikation (`START`, `STOP`, `NOPOSE`) innerhalb der RoI durchführt.

Vorteile:

- **Modularität und Spezialisierung:** Jedes Netzwerk kann unabhängig voneinander entwickelt, trainiert und optimiert werden. Dies ermöglicht den Einsatz spezialisierter Architekturen für jede Teilaufgabe. Ein Austausch oder eine Verbesserung des Gestenklassifikators erfordert keine Anpassung am Personendetektor.
- **Effiziente Ressourcennutzung:** Netz 1 kann auf geringe Komplexität und hohen Durchsatz optimiert werden. Netz 2 kann zwar komplexer sein, muss seine Inferenz aber nur auf einem kleinen Bildausschnitt durchführen, was die Gesamtlast signifikant reduziert.
- **Vereinfachtes Training und Datenmanagement:** Das Training zweier separater Modelle ist inhärent einfacher. Es können zwei unterschiedliche, optimierte Datensätze verwendet werden, ohne dass eine komplexe, duale Annotation (Bounding Box und Gesten-Label pro Bild) erforderlich ist. Die Konvergenz der Modelle ist leichter zu erreichen und zu debuggen.

Nachteile:

- **Latenz-Akkumulation:** Die End-to-End-Latenz ist die Summe der Latenzen der einzelnen Stufen (Inferenz Netz 1 + CPU-Verarbeitung + Inferenz Netz 2). Dies kann in schnellen Szenarien zu einem Performance-Engpass führen.
- **Fehler-Kaskadierung:** Die Leistung des Gesamtsystems ist direkt von der Leistung der ersten Stufe abhängig. Wenn der Personendetektor eine Person nicht erkennt (False Negative), erhält der Gestenklassifikator keinen Input, und das System versagt für diesen Frame vollständig.
- **Größe:** Obwohl die Netze ähnliche Aufgaben verrichten (Objektklassifikation, speziell für die Klasse Mensch), funktionieren sie unabhängig voneinander. Zwei separate Netze könnten somit unter Umständen nahezu doppelt so viele Ressourcen erfordern, wie ein kombiniertes Netz.

Analyse des integrierten Ansatzes (Ein Multi-Task-Netzwerk)

Ein *Multi-Task-Learning (MTL)* Ansatz nutzt eine einzige Netzwerkarchitektur, typischerweise bestehend aus einem gemeinsamen „Backbone" zur Merkmalsextraktion und zwei oder mehr „Köpfen" (Output Branches) für die spezifischen Aufgaben. Das Netzwerk erhält das gesamte Bild als Input und gibt in einem einzigen Forward-Pass sowohl die Bounding-Box-Koordinaten der Person als auch die Klassifikation der Geste aus.

Vorteile:

- **Reduzierte Latenz und hohe Recheneffizienz:** Die rechenintensiven Convolutional Layers des Backbones werden nur einmal durchlaufen und die extrahierten Merkmale für beide Aufgaben genutzt. Dies reduziert die Gesamtzahl der Operationen und führt potenziell zu einer geringeren End-to-End-Latenz im Vergleich zum kaskadierten Ansatz.
- **Ganzheitliches Lernen und Merkmals-Regularisierung:** Das Netzwerk kann implizite Korrelationen zwischen der Anwesenheit einer Person und der Ausführung einer Geste lernen. Die Notwendigkeit, Merkmale zu erzeugen, die für beide Aufgaben nützlich sind, wirkt als Regularisierungsmechanismus, der die Generalisierungsfähigkeit des Modells verbessern kann.
- **End-to-End-Optimierung:** Das gesamte System kann als eine Einheit trainiert werden. Dies ermöglicht eine globale Optimierung der Gewichte für die kombinierte Aufgabe, was theoretisch zu einer besseren Gesamtperformance führen kann als die lokale Optimierung zweier separater Modelle.

Nachteile:

- **Hohe Trainings- und Datenkomplexität:** Dieser Ansatz erfordert einen aufwendig annotierten Datensatz, bei dem jede Bildinstanz sowohl eine Bounding Box als auch ein Gesten-Label aufweist. Das Training selbst ist anspruchsvoller, da die Loss-Funktionen beider Köpfe sorgfältig gewichtet und balanciert werden müssen, um eine stabile Konvergenz zu gewährleisten.
- **Potenziell höhere Ressourcenanforderungen:** Obwohl recheneffizient, kann die kombinierte Architektur eines Multi-Task-Netzwerks in ihrer Gesamtgröße und Komplexität die Ressourcen eines kleinen FPGAs übersteigen. Es handelt sich um eine monolithische Implementierung, die als Ganzes auf den Chip passen muss.

- **Negative Task-Interferenz:** Es besteht das Risiko, dass die Optimierung für eine Aufgabe die Leistung der anderen beeinträchtigt. Der Kompromiss im gemeinsamen Backbone könnte dazu führen, dass keine der beiden Aufgaben ihr maximales Genauigkeitspotenzial erreicht.

Fazit

Beide vorgestellten Architekturen stellen valide Lösungsansätze dar, deren Eignung stark von den spezifischen Randbedingungen des Projekts abhängt. Der **kaskadierte Ansatz** bietet einen pragmatischen, modularen und risikoarmen Weg, der besonders für ressourcenlimitierte Umgebungen und iterative Entwicklungszyklen geeignet ist. Er zeichnet sich durch seine Einfachheit in Training und Implementierung aus, birgt jedoch das Risiko von Latenz-Akkumulation und Fehler-Kaskadierung.

Der **integrierte Multi-Task-Ansatz** repräsentiert eine elegantere und potenziell performantere Lösung, die eine niedrigere Latenz und eine bessere Generalisierung verspricht. Dem gegenüber stehen jedoch ein signifikant höherer Aufwand in der Datenerstellung und im Training sowie das Risiko, dass die monolithische Architektur die Hardware-Ressourcen des Ziel-FPGAs übersteigt.

Eine rein theoretische Analyse kann keine endgültige Entscheidung liefern. Die optimale Architektur für dieses spezifische Szenario muss durch **empirische Evaluation** ermittelt werden. Zukünftige Arbeiten sollten daher die prototypische Implementierung des Multi-Task-Netzwerks umfassen. Ein direkter Vergleich basierend auf quantitativen Metriken wie End-to-End Frames per Second (FPS), Detektions- und Klassifikationsgenauigkeit (mAP, Accuracy), sowie der tatsächlichen FPGA-Ressourcennutzung (insbesondere BRAM) ist unerlässlich, um die für dieses Projekt geeignetste Lösung zu identifizieren.

4.7. Ausblick

Basierend auf den durchgeführten Analysen und den identifizierten Herausforderungen lassen sich für zukünftige Arbeiten mehrere Forschungsthemen und Arbeitspakete ableiten. Ziel künftiger Projektphasen sollte es sein, die bestehende Toolchain zu stabilisieren, ihre Kompatibilität mit aktuellen Systemumgebungen sicherzustellen und alternative Ansätze zur Gestenerkennung weiter zu evaluieren.

4.7.1. Vervollständigung der FPGA-basierten Inferenzkette

Ein zentrales Ziel besteht in der vollständigen Integration und Inbetriebnahme der Personen- und Gestenklassifikationsnetze auf der PYNQ-Z2-Plattform. Dazu sind insbesondere folgende Schritte erforderlich:

- Integration beider Netze in ein gemeinsames Vivado-Design unter Beachtung der Ressourcenbeschränkungen des Zynq-7000 SoC.
- Erfolgreiche Synthese der FINN-IP-Blöcke mit aktuellem Softwarestand (Vivado/Vitis 2024.2).
- Ggf. Anpassung des Treibers zur Ansteuerung der FPGA-Komponenten über die CPU.
- Validierung des Echtzeitverhaltens mit Live-Videodaten und Analyse der End-to-End-Latenz.

4.7.2. Automatisierung der Toolchain und Build-Prozesse

Die hohe Komplexität des FINN-Workflows sowie die starke Abhängigkeit von spezifischen Softwareversionen legen nahe, die relevanten Build-Prozesse durch Skripte zu automatisieren. Empfohlen wird:

- Erstellung reproduzierbarer Docker-Container mit definierten Abhängigkeiten.
- Verwendung von Build-Skripten zur vereinfachten Ausführung der FINN-Phasen (Import, Optimierung, Synthese).
- Dokumentation und Versionierung aller Konfigurationen in einem dedizierten Git-Repository.

4.7.3. Evaluation alternativer Gestenerkennungsverfahren

Neben der bestehenden Bildklassifikation auf Basis von 32×32-Kacheln bietet insbesondere die Pose-basierte Erkennung mittels Keypoint-Extraktion sowie das Fine-Tuning vortrainierter YOLO-Modelle eine vielversprechende Alternative. Im Fokus zukünftiger Arbeiten könnte stehen:

- Training und Integration leichter Klassifikatoren (z. B. Entscheidungsbäume, SVMs) auf Basis berechneter Gelenkwinkel.
- Vergleich der Klassifikationsgüte beider Ansätze (Pose vs. CNN) anhand eines standardisierten Testdatensatzes.
- Untersuchung der Robustheit bei veränderten Aufnahmebedingungen (Beleuchtung, Kameraposition, Kleidung).

5. Fahrsteuerung

geschrieben von Tim Ranft

Die Fahrsteuerung bildet das zentrale Glied zwischen der Fahrabsicht und der physikalischen Ansteuerung des Vehikels. Während die erste Abstraktionsschicht die Ansteuerung der Motoren mittels Pulsweitenmodulation (PWM) vornimmt, bildet eine weitere Abstraktionsschicht die Schnittstelle für eine Reihe komplexerer Fahranweisungen. Die Schnittstelle für andere Programme und Tests der Fahrsteuerung wird mittels Hypertext Transfer Protocol (HTTP)-Endpunkten angesprochen. Die existierende Fahrsteuerung wurde im Zuge des Upgrades auf das PYNQ-Z2-Board entwickelt und getestet.

5.1. Grundlagen

geschrieben von Tim Ranft

Im Folgenden werden grundlegende Konzepte für die Entwicklung der Fahrsteuerung dargelegt. Insbesondere werden wichtige Grundlagen für das Verständnis der Fahrsteuerung erläutert, sowie deren Entscheidungsgrundlage anhand der Vor- und Nachteile einzelner Designentscheidungen diskutiert.

5.1.1. Speicherort und Ausführung der Anwendung

Die Dateien sind im Heimverzeichnis des Benutzers *Xilinx* (~/Fahrsteuerung_Server/) auf dem PYNQ-Z2 Board #5 gespeichert.

Der Server (server.py) für die HTTP-Webschnittstelle der Fahrsteuerung ist vollständig gestartet, nachdem die vier Leuchtdioden am unteren rechten Teil des PYNQ-Z2-Boards einmal aktiv waren und dann erlöschen. Das Erlöschen der Dioden bestätigt, dass das für die Fahrsteuerung gebaute Overlay für das FPGA geladen wurde.

Für die Ausführung des Servers ist ein eigens angelegter Service verantwortlich. Dieser wird nach dem Start des Systems automatisch gestartet. Bei Verbindung zum Board kann unter Port 5000 die Fernbedienung (beispielsweise *http://192.168.50.1:5000/* bei Aufruf über Access Point (Modus) (AP)) aufgerufen werden. Die vollständige Einrichtung des Services für eine Replikation auf ein anderes Board ist dem Anhang unter Anhang D beigefügt.

5.1.2. PDM

geschrieben von Alexander Hoegen-Jupp

Pulsdauermodulation (PDM) (früher auch Pulsweitenmodulation (PWM)) ist eine Methode um analoge Signale durch digitale Signale darzustellen [Koh20] [Hir22]. Hierbei wird ein Signal mit

einer gewissen Frequenz ein und wieder ausgeschaltet. Dabei beschreibt der *duty cycle* das Verhältnis, wie lang das Signal aktiv ist im Vergleich zu wie lange das Signal aus ist [Koh20]. Dies erlaubt es einem Gerät mithilfe von digitalen Signalen eine variable Spannung zu liefern [Hir22], um bspw. die Helligkeit von LEDs oder die Geschwindigkeit von Motoren zu steuern.

Anwendung auf dem PYNQ-Z2-Board mit dem Motor-Driver

geschrieben von Tim Ranft

mitgearbeitet haben Kilian Müller und Philipp Hennken

Hinweis:

Der nachfolgende Teil erfordert tiefes Verständnis über das Processing System (PS) und Processing Layer (PL) auf dem PYNQ-Board, sowie Verständnis über *Vivado* und Intellectual Property (IP)-Cores.

Auf dem PYNQ-Z2-Board bieten sich verschiedene Möglichkeiten für eine Umsetzung der PWM. Eine mögliche Lösung besteht über das Anbinden der schwarzen Verbindungskabel vom Motor-Driver mittels einzelner DuPont Verbindungskabel (Male to Male) in die Steckbuchsen von PMODA oder PMODB (siehe Pinbelegung in Abbildung 5.1).

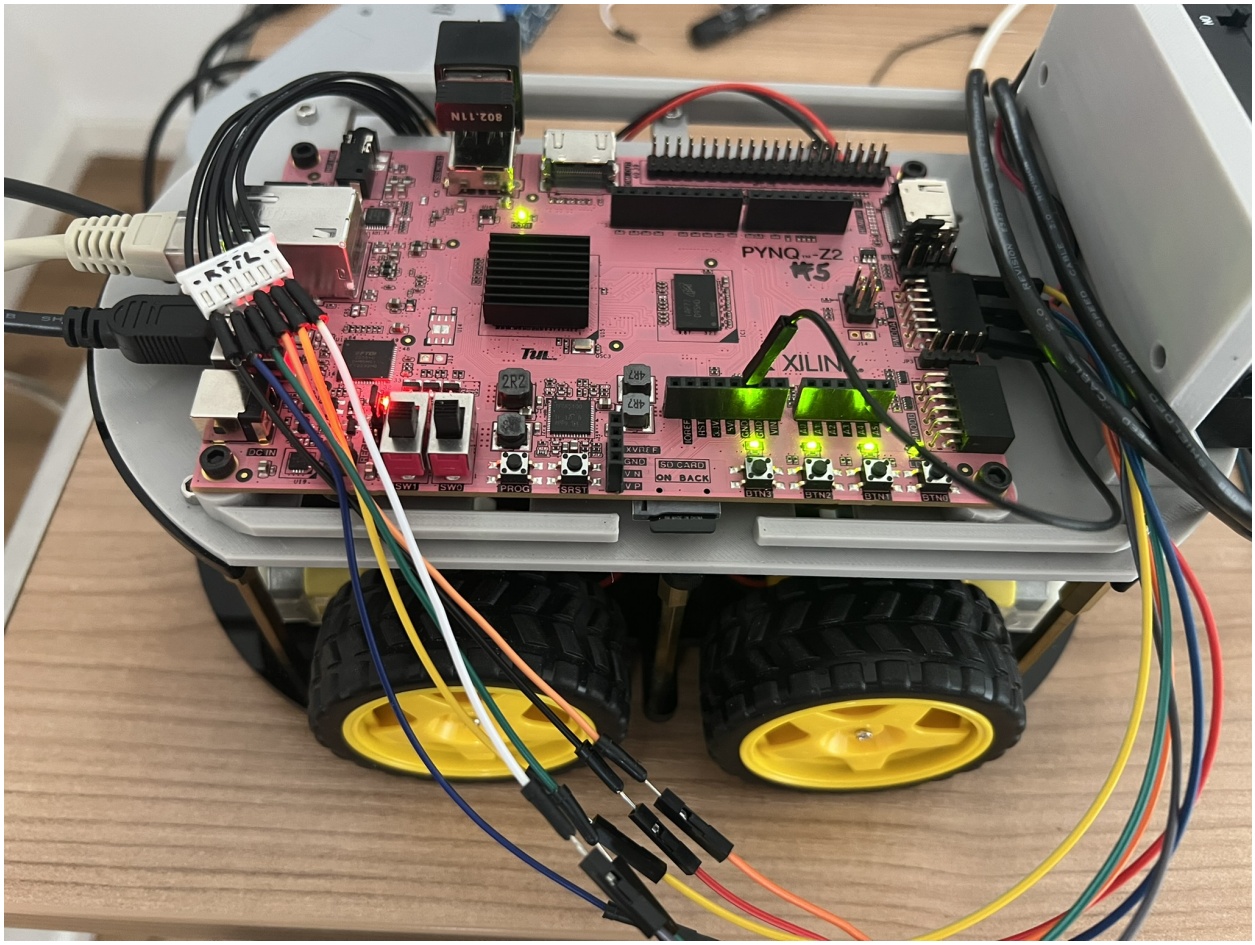


Abbildung 5.1.: Belegung der Motor-Driver-Pins mittels DuPond-Verbindungen an PMODA
Bild: Tim Ranft

Die Anschlussweise dieses Verfahrens hat jedoch einige Nachteile. Wie in der Abbildung zu erkennen ist, sind die Kabel nicht in einer Reihe nebeneinander eingesteckt, sondern durch die Belegung der Pins im Base-Overlay[1] vorgegeben. Außerdem ist eine Wartung bzw. das Ein- und Ausstecken durch das neue Chassis und der damit verbundenen Positionierung des Batterie-Packs zwar möglich, aber unkomfortabel. Auch sind die verschiedenfarbigen DuPont-Steckverbindungen fehleranfälliger für eine Falschbelegung und es ergeben sich vermehrte Verbindungsstellen (potenzielle *points of failure*), die sowohl nicht richtig eingesteckt, als auch einen versehentlichen Kontakt mit anderen Pins auf dem Board verursachen können. Die Folgen sind hierbei ein nicht ansteuerbarer Motor oder im schlimmsten Fall ein Kurzschluss, verursacht durch den Kontakt mit anderen Pins.

Zu beachten ist, dass **alle** auf dem Board verbauten Pins und Pin-Buchsen über das PS angesprochen werden und für eine Ansteuerung über das PL entsprechende Brücken eingerichtet werden müssen. Diese Verbindungen werden durch das Base-Overlay zwar teilweise übernommen, sind jedoch nicht mit mehreren PWM-Generatoren ausgestattet, wodurch der Betrieb zweier unabhängiger Motorseiten (bpsw. links 40 %; rechts 50 %) nicht möglich ist.

Aufgrund dieses entscheidenden, weiteren Nachteils mit der Nutzung des Base-Overlays verhält sich das zuletzt initialisierte PWM-Modul in der Ansteuerung nicht wie erwartet. Ein einfaches Beispielskript zeigte in einem Test, dass sich die Wellen der beiden Module überlagern, was zu einem Fehlerfall führt. Auf dem zuletzt initialisierten Signal-Pin ist dauerhaft *HIGH* auf vorwärts und rückwärts für die entsprechende Motorseite angelegt. Sind sowohl vorwärts als auch rückwärts mit dem Signal *HIGH* belegt, dreht sich der Motor nicht und lässt sich im Weiteren auch nicht mehr steuern. Zudem hängt sich der Programmcode nach dieser Ansteuerung auf.

Finale Umsetzung der Pinbelegung

Nachdem die beiden Ports PMODA und PMODB nun nicht mehr für die Ansteuerung zur Auswahl stehen, eignen sich die zusammenhängenden Pins AR8 bis AR13, welche im Base-Overlay normale *GPIO*-Pins darstellen. Dadurch, dass diese direkt nebeneinander liegen, kann der Stecker der schwarzen Verbindungskabel direkt in die Ports gesteckt werden.

[1] Bitstream, der für das Board von den Entwicklern zur Verfügung gestellt wurde

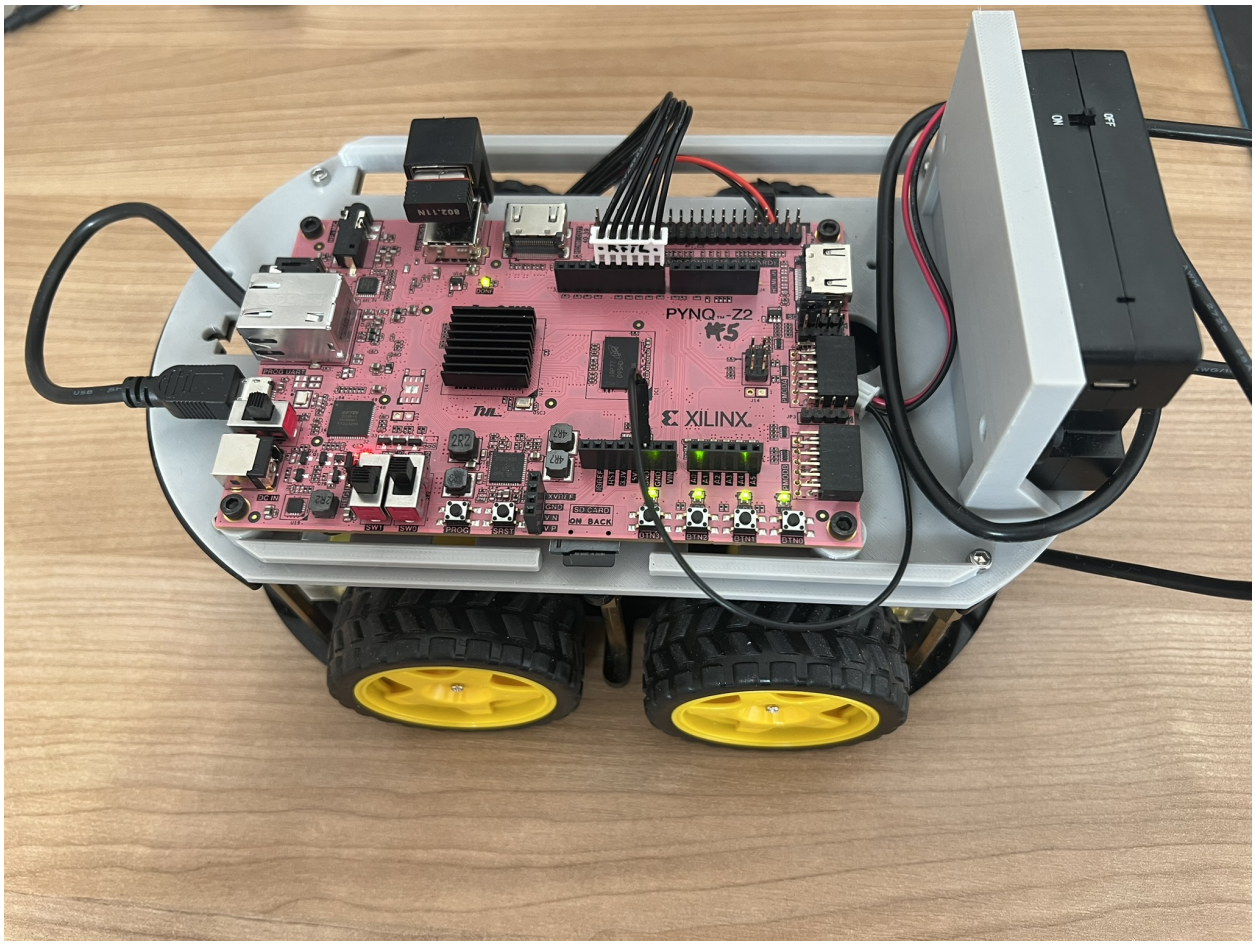


Abbildung 5.2.: Finale Pinbelegung auf dem Board Bild: Tim Ranft

Der in Abbildung 5.2 abgebildete weiße Stecker ist folgendermaßen beschriftet

•; rechts; $\overline{\text{rechts}}$; $\overline{\text{links}}$; links; •

Dies dient der Orientierung bei der Belegung der Pins in den Ports.

Die alternativ auch zusammenhängenden Pins A0 bis A5 sind nicht geeignet, da diese noch einen internen Kondensator vor den Ausgang geschaltet haben, welcher bei einer PWM wie ein Tiefpassfilter agiert und damit wesentliche Hochfrequenzanteile des PWM-Signals dämpft.

Da die ausgewählten Pins im Base-Overlay nicht für eine Modulation mittels PWM ausgelegt sind (durch fehlende Generatoren), muss für die Ansteuerung ein eigenes Overlay, also ein eigener Bitstream mithilfe von *Vivado* angelegt werden. Dieser beinhaltet vier eigene AXI-PWM-Module und entsprechende *HIGH*-Pins für die *Enable*-Eingänge, welche nicht durch den Softwareteil dauerhaft auf *HIGH* gesetzt werden müssen. Außerdem werden weitere IP-Cores für das Zurücksetzten mittels Software und für den Basistakt eingebunden.

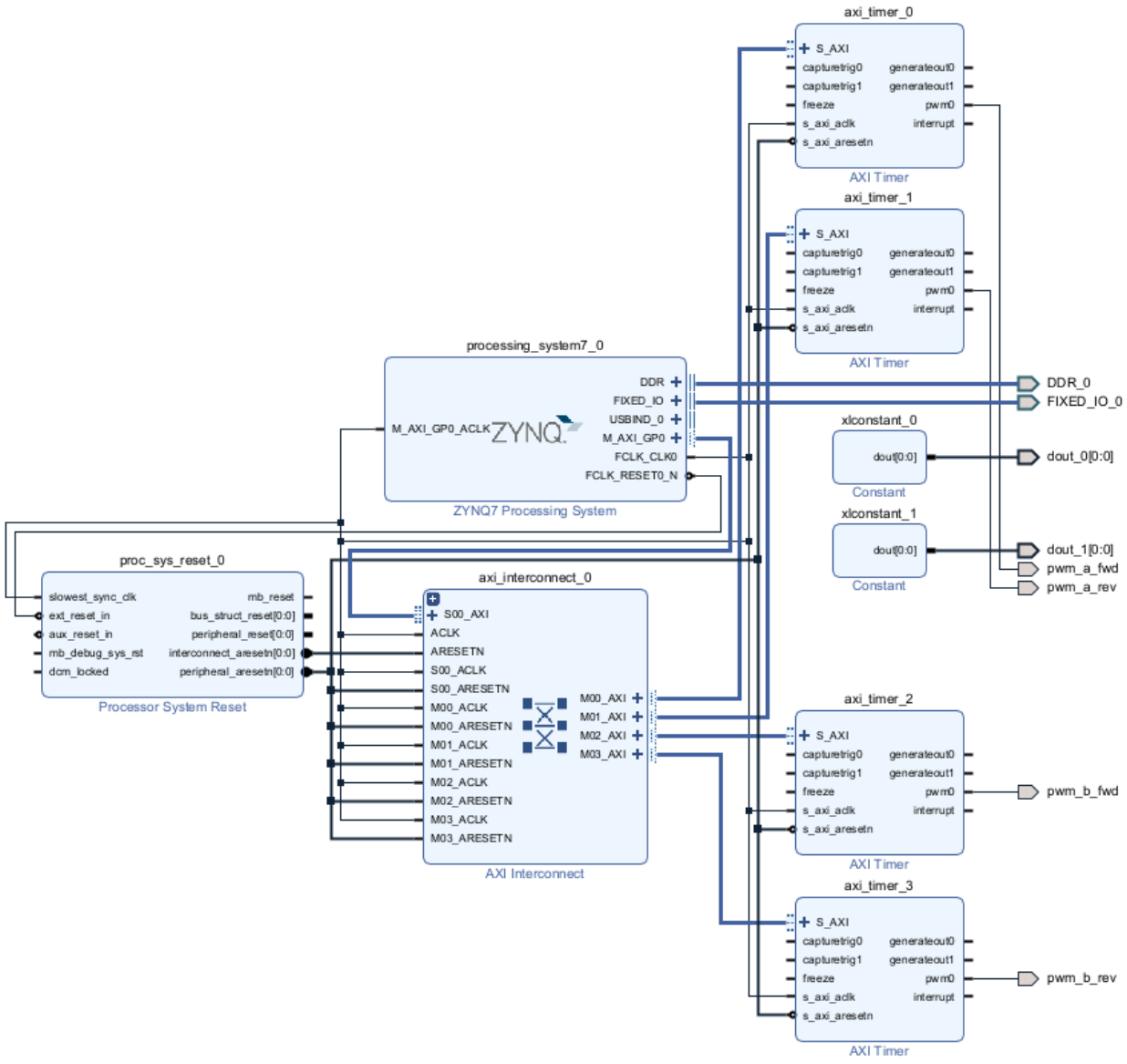


Abbildung 5.3.: Blockschaltbild der Ausgangsmodulatoren in Vivado Screenshot: Tim Ranft

Die .xdc-Datei sorgt für die Zuordnung der Ausgabeleitungen des Blockschaltbilds zu den entsprechenden Pins auf dem Z2-Board.

```
1 set_property -dict {PACKAGE_PIN N17 IOSTANDARD LVCMOS33} [get_ports
    {dout_1}]
2 set_property -dict {PACKAGE_PIN P18 IOSTANDARD LVCMOS33} [get_ports
    {pwm_a_fwd}]
3 set_property -dict {PACKAGE_PIN R17 IOSTANDARD LVCMOS33} [get_ports
    {pwm_a_rev}]
4 set_property -dict {PACKAGE_PIN T16 IOSTANDARD LVCMOS33} [get_ports
    {pwm_b_fwd}]
5 set_property -dict {PACKAGE_PIN V18 IOSTANDARD LVCMOS33} [get_ports
    {pwm_b_rev}]
6 set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports
    {dout_0}]
```

Die entsprechende Benennung der Pins können den pinken Labels aus dem *PYNQ Z2 Pinout* entnommen werden (siehe Abbildung A.1). Die letzte und erste Zeile repräsentieren die dauerhaft auf *HIGH* belegten Ausgänge für die Enable-Pins.

Da eine die Generierung des Bitstreams besonders von der Einrichtung des Projekts in Vivado (Version 2023.2) abhängig gewesen ist, wurde das komplette Projekt (pwm_overlay_neu) als komprimiertes Archiv in GitLab hochgeladen. In diesem befindet sich das Design und die Constraints (pynq-constraints.xdc), sowie die getroffenen Einstellungen für das PYNQ-Z2-Board. Das Projekt darf unter Angabe des Autors als Vorlage für weitere Iterationen genutzt werden.

5.2. Erkenntnisse aus vorherigen Projektberichten

geschrieben von Alexander Hoegen-Jupp

Im Folgenden werden die Erkenntnisse aus den vorangegangenen Berichten aufgegriffen und dargestellt. Zuerst wird in Unterabschnitt 5.2.1 die Implementierung der Fahrsteuerung analysiert und bewertet. Danach wird das Problem der Geradeausfahrt des Fahrzeugs betrachtet und in den Projektkontext eingeordnet (siehe Unterabschnitt 5.2.2). Zuletzt wird die Regulierung der Geschwindigkeit, welche in Vorgängerprojekten als ein Problem erkannt wurde, betrachtet.

5.2.1. Umsetzung der Fahrsteuerung

Durch mangelnde Dokumentation, welche auch auf dem Z2 Board anwendbar ist, ist es nicht gelungen, die bestehende Lösung der Fahrsteuerung zu testen und zu evaluieren (vgl.Unterabschnitt 5.3.5). Das Team hat sich in auf eine Neuentwicklung entschieden, welche im Folgenden beschrieben ist.

Für die Ansteuerung der einzelnen Pins mittels PWM wurde die bisher in C++ umgesetzte Lösung (vgl. [ACB+22]) verworfen und auf dem neuen PYNQ-Z2-Board eine Pyhton3-Lösung ausgearbeitet. Viele der Beispiele, wie die Einbindung eines Bitstreams [Xil] [Gio21] oder die Ansteuerung einzelner Hardwarekommandos sind in Python bereits umgesetzt und speziell für PYNQ-Entwicklungsboards in einem Python-Libary-Package zur Verfügung gestellt.

Hinzu kommt, dass die Fahrsteuerung sowie deren Implementation seit einigen Iterationen nicht im Fokus gestanden hat. Dies hat dazu geführt, dass die Fahrsteuerung seit der Iteration 2021/2022 nicht mehr aktualisiert wurde.

Der in der Iteration 2021 verfolgte Ansatz, den Code in Klassen zu modularisieren ist von der Konzeption gelungen, allerdings sind dort C++-Erweiterungen zum Einsatz gekommen. [ACB+21, S. 69–79].

Diese Programmiersprache wird regulär nicht an der Hochschule Bremen (HSB) gelehrt, weshalb es eine zusätzliche Belastung für die Studierenden ist, sich in diese Sprache einzuarbeiten.

Die Fahrsteuerung wurde zuvor als Zustandsautomat entwickelt. In der Iteration 2021/22 [ACB+22, S. 33] wurde herausgefunden, dass die Fahrsteuerung nie getestet wurde und der Automat nicht frei von Fehlern ist. Da es zu diesem Zeitpunkt auch keine funktionierende Gestenerkennung gab, wurde der Automat aufgrund mangelnder Tests größtenteils auskommentiert und mit einfacheren Konstrukten ersetzt. Zudem wurde die Empfehlung ausgesprochen, dass der endliche Automat erst wieder betrachtet werden sollte, wenn die Gestenerkennung funktioniert.

5.2.2. Probleme bei der Geradeausfahrt

geschrieben von Tim Ranft und Kilian Müller

Auffällig ist die Abweichung bei einfachen Fahrmanövern wie der linearen Vorwärtsbewegung. Konkret zeigt sich, dass das autonome Fahrzeug beim Folgen einer geraden Fahrspur allmählich von der idealen Linie abweicht, wobei die Richtung und der Grad der Abweichung nicht gleichmäßig sind.

Für die weitere Entwicklung des Prototyps ist es wichtig, dass das Auto zuverlässig in eine Richtung fahren kann. So ist beispielsweise eine Sprachsteuerung mit dem Kommando *vorwärts Marsch* darauf angewiesen, die Fahrtrichtung beizubehalten, um Kollisionen mit rechts und links neben dem Fahrzeug liegenden Hindernissen zu vermeiden und potenziellen Verklemmungen zu entgehen.

Anders sieht es bei der Verfolgung von Personen mittels Kamera aus. Da voraussichtlich ausreichend oft ein Steuerbefehl für eine Korrektur der Geschwindigkeit und Richtung des Fahrzeugs ankommen wird, ist eine Korrektur basierend auf den Bilddaten der Kamera gegeben. Wenn das Auto während einer solchen Verfolgungsfahrt durch die Motoren abweicht, wird das Kamerasystem/-Programm dem Auto rechtzeitig eine Korrektur in die entgegengesetzte Richtung geben, wobei darauf zu achten ist, dass diese Bewegung mit ausreichend Geschwindigkeit geschieht, um ein nicht drehen einzelner Motoren zu vermeiden.

Darüber hinaus kann festgestellt werden, dass der Aufbau und die Konzeption des Vehikels eine ausreichende Steuerung der Motoren für den auf der Erde angewandten Einsatzzweck zulassen.

5.2.3. Regulierung der Geschwindigkeit

geschrieben von Tim Ranft und Alexander Hoegen-Jupp

Des Weiteren soll das Auto die Möglichkeit haben, langsamer fahren zu können. Aus den Projektberichten der Vorjahre geht hervor, dass die Geschwindigkeit in direktem Zusammenhang zu Vibrationen des Fahrzeugs steht und eine langsamere Fahrt für mehr Stabilität sorgen würde. Die Stabilität wird in Bezug auf die Kamera benötigt, welche während der Fahrt oft durch zu verwackelte Bilder wenig bis keine verwendbaren Daten liefern konnte.

Auch könnte bei einer langsamen Fahrt die Geräuschemission durch die Motoren geringer sein, welches der Sprachsteuerung, genauer der Messung durch das Mikrofon, zugutekommen könnte. Weitere Verbesserungen der Bildqualität in Bezug auf die Verbesserungen am Fahrzeug sind in Abschnitt 8.2 aufgeführt.

5.3. Erarbeitete Ergebnisse

geschrieben von Tim Ranft und Alexander Hoegen-Jupp

In diesem Kapitel sind die von der Arbeitsgruppe erzielten Ergebnisse dokumentiert. Außerdem werden die Konzeptentscheidungen und Änderungen der bisherigen Implementation beleuchtet.

Zuerst wird in Unterabschnitt 5.3.1 darauf eingegangen, wie die Abweichung des Fahrzeugs bei einer Geradeausfahrt quantifiziert wird und potenzielle Lösungsansätze betrachtet. Unterabschnitt 5.3.2 wird die Einrichtung eines AP auf dem PYNQ-Z2-Boards beschrieben. Es folgt die Beschreibung eines neuen Konzepts für die Modularisierung der Fahrsteuerung in Unterabschnitt 5.3.3 sowie eine Erläuterung, wie das PYNQ-Z2 Board angesteuert wurde (Unterabschnitt 5.3.4). Danach wird in Unterabschnitt 5.3.5 der Versuch skizziert, die Fahrsteuerung aus

den vorherigen Iterationen zu kompilieren. Als vorletztes wird auf die neue Struktur und das Konzept der Fahrsteuerung eingegangen (Unterabschnitt 5.3.6). Zuletzt wird das Konzept und die Funktionsweise der grafischen Benutzeroberfläche beschrieben, welche für die Steuerung des Fahrzeugs genutzt werden kann (siehe Unterabschnitt 5.3.7).

5.3.1. Versuch zur Quantifizierung der Abweichung bei Geradeausfahrt

Identifizierung und Analyse des Problems

geschrieben von Kilian Müller, Tim Ranft und Philipp Hennken

mitgearbeitet haben Alexander Hoegen-Jupp

Zu Beginn des Projekts wurde das bestehende Problem mit der Abweichung des Fahrzeugs von der Idealgeraden aus den vergangenen Iterationen angegangen [ABB+25, Kapitel 11.5.3]. Für einen ersten Funktionstest wurde zunächst jeder einzelne der vier Motoren des Fahrzeugs mit dem Board #5 an eine Spannungsversorgung angeschlossen. Hierdurch konnte ein Gefühl für die zu erwartende Umdrehungsgeschwindigkeit der Motoren entwickelt werden.

Anschließend wurde an diesem Fahrzeug eine Spannungsversorgung angebracht, welche die vier Motoren des Fahrzeugs gleichermaßen ansteuert, um das Fahrzeug in einen fahrtüchtigen Zustand zu bringen. Die Motoren wurden dabei allerdings mit der vollen, nicht regulierten Spannung angesteuert, wodurch sich das Fahrzeug zügig und mühelos in Bewegung setzte. In einem ersten Test ohne konkreten Versuchsaufbau konnte beobachtet werden, dass das Fahrzeug eine Linksneigung in Fahrtrichtung besitzt.

Für eine konkrete Quantifizierung der Abweichung von der Idealgeraden für diesen fahrtüchtigen Prototyp wurde ein Versuch durchgeführt. Das Ziel ist die Bestimmung der Abweichung des Fahrzeugs auf einer Strecke von fünf Metern. Dafür wurde eine Strecke von fünf Metern im Laborraum markiert und ein orthogonal ausgerichteter Zollstock am Ende der Strecke platziert. Zunächst sollte eine Testreihe mit zehn Messwerten aufgenommen werden (siehe Tabelle 5.1). Während des Versuchs wurde festgestellt, dass eine angemessene Reproduzierbarkeit der Fahrzeugausrichtung vor dem Start nicht garantiert werden konnte.

Tabelle 5.1.: Geradeausfahrt: Messergebnisse der ersten Testreihe

Messung	Richtung der Abweichung aus {L;0;R}	Gemessene Abweichung in [m]
1	L	1,77
2	L	0,47
3	L	1,10
4	L	0,69
5	R	0,43
6	L	0,82
7	L	1,44
8	L	1,53

Mit dem ersten Versuchsaufbau konnte nicht ausreichend sichergestellt werden, dass die Messergebnisse unabhängig von der Positionierung des Fahrzeugs bei Versuchsstart sind. Basierend auf dieser Beobachtung wurde die Durchführung der Testreihe bereits nach acht von zehn Messungen vorzeitig beendet.

Um eine präzisere Ausrichtung des Autos gewährleisten zu können, wurde eine zweite Testreihe aufgenommen, bei welcher der genaue Startpunkt jedes Reifens markiert wurde, sodass das Fahrzeug bei jeder Messung von derselben Startposition losfährt. Die entsprechenden Stellen wurden außerdem mit Kreppklebeband am Boden befestigt. Die Ergebnisse der zweiten Testreihe sind in der Tabelle 5.2 zu finden.

Tabelle 5.2.: Geradeausfahrt: Messergebnisse der zweiten Testreihe

Messung	**Richtung der Abweichung aus {L;0;R}**	**Gemessene Abweichung in [m]**
1	L	0,47
2	L	0,45
3	L	1,81
3	L	1,25
4	L	1,54
5	L	1,26
6	L	0,64
7	L	1,22
8	L	0,8
9	L	1,4
10	L	0,01

Auswertung der Testergebnisse

geschrieben von Kilian Müller

Die Auswertung der beiden Versuchsdurchläufe zeigt eine deutlich schwankende Abweichung des Fahrzeugs von der Idealgeraden. Im ersten Durchlauf konnte bereits eine überwiegende Linksabweichung festgestellt werden, die Aussagekraft ist durch die unpräzise Startposition jedoch eingeschränkt. In der zweiten Testreihe, bei welcher der Startpunkt des Fahrzeugs reproduzierbar war, konnte die Linksabweichung bestätigt werden, da ausschließlich Messungen mit einer Linksabweichung beobachtet wurden. In der Linksabweichung lassen sich allerdings keine Regelmäßigkeiten finden, da die Streuung der einzelnen Werte ($0,01$ m bis $1,81$ m) auch hier sehr stark ist.

Die Messergebnisse deuten auf eine Ursache im Aufbau oder der Ansteuerung des Fahrzeugs hin. Es ist zu vermuten, dass hier mechanische und/oder elektrische Ungleichgewichte im Fahrzeug ursächlich sind.

Eine stabile Geradeausfahrt des Fahrzeugs ist unter diesen Umständen nicht sicher möglich, vor allem nicht über eine längere Strecke.

Fehlerquellenanalyse und Lösungsansätze

geschrieben von Kilian Müller und Tim Ranft

Im Anschluss an die Auswertung der Messergebnisse wurde das Fahrzeug auf mögliche Ursachen für die beobachtete systematische Abweichung hin untersucht. Dabei konnten mehrere potenzielle Fehlerquellen identifiziert werden, die die Geradeausfahrt beeinflussen könnten:

Der Aufbau des Fahrzeugs in Bezug auf die Fahrsteuerung ist geprägt durch die Stellung der einzelnen Reifen, welche durch die Befestigung am Chassis (Fahrgestell) bereits leicht verzo-

gen sind. Auch sind die Schrauben nicht perfekt gleichmäßig angezogen, was zu einer ungleichen Klemmkraft der Reifen am Chassis führt. Beides hat zur Folge, dass die Motoren eine unterschiedliche Kraft aufbringen müssen, um die Haftreibung der beweglichen Komponenten überwinden zu können.

Die Fahrsteuerung wird über vier individuelle DC-Motoren bewältigt. Diese setzen jedes Rad individuell in Bewegung, wobei immer zwei pro Seite gleichzeitig durch den Motorcontroller L298N angesteuert werden. Durch die oben aufgeführte Bauweise und Abweichungen bei der Produktion der anzusteuernden Motoren drehen sich einige Räder unter Volllast schneller als andere. Auch die weiterhin wirkende Gleitreibung wirkt sich während der Fahrt auf die Abweichung bei der Geradeausfahrt aus.

Besonders ist die Haftreibung und die Gleitreibung beim Anfahren der Motoren sichtbar. Bei Versorgung über eine parallele Leitung drehen einige Räder früher und andere später. Da diese Effekte in Summe abhängig von der Umgebung sind, sind diese nicht präzise bestimmbar, um beispielsweise eine Nachberechnung für die Regelung der Ansteuerung vorzunehmen. Es konnte somit nicht festgestellt werden, dass eine Reifen-Motor-Kombination immer anfälliger ist als eine andere.

5.3.2. Einrichtung und Verbindung eines AP

geschrieben von Tim Ranft

Um das PYNQ-Z2-Board demnächst direkt über eine drahtlose Schnittstelle mobil und zuverlässig ansteuern zu können wurde ein Access Point (Modus) (AP) auf dem Board eingerichtet. Dieses Vorgehen hat den entscheidenden Vorteil, dass die Konfiguration für Folgeiterationen nicht von den mobilen Hotspots dieser Iteration abhängen. Außerdem können mehrere Personen ohne physische Verbindung via Kabel eine Verbindung zum Board aufrufen.

Das Board stellt hierbei das WLAN-Signal wie bei einem Router zur Verfügung und verwaltet die Anmeldungen. So ist es in der Lage beispielsweise die Steuerbefehle über die Webschnittstelle entgegenzunehmen oder SSH-Sessions aufzubauen.

Die Einrichtung ist auf dem Board #5 bereits abgeschlossen und kann direkt verwendet werden. Um in weiteren Iterationen auch ein anderes Board mit dieser Lösung auszustatten, ist eine entsprechende Anleitung erstellt worden, die sich im Anhang dieses Dokuments befindet (siehe Anhang C).

5.3.3. Konzept für Modularisierung der Fahrsteuerung

geschrieben von Philipp Hennken und Alexander Hoegen-Jupp

mitgearbeitet haben Tim Ranft und Kilian Müller

Ein Teilprojektziel für diesen Projektdurchlauf ist die Modularisierung der Fahrsteuerung, sodass eine einfache Schnittstelle für darauf angewiesenen Untergruppen zur Verfügung steht. Der logische Aufbau der Fahrsteuerung ist in Abbildung 5.4 dargestellt. Angedacht ist ein System mit mehreren Abstraktionsebenen. Wenn die Sprachsteuerung einen auditiven Befehl erkannt hat, so wird dieser an die Fahrsteuerung weitergeleitet. Diese ist die „intelligente“ Schicht und verarbeitet diese Eingabe in Befehle, welche die Motorsteuerung bereitstellt. Die einzige Aufgabe der Motor-

steuerung ist es, die Motoren physisch anzusteuern. Dazu stellt sie die in Tabelle 5.3 aufgeführten Befehle als Schnittstelle bereit.

Der Vorteil dieser Struktur ist ihre Modularität, welche unabhängige Änderungen an den einzelnen Komponenten ermöglicht. Zusätzlich lässt sich die Logik der Fahrsteuerung anpassen, ohne dass der Motorsteuerungscode angepasst werden muss. So kann in der Fahrsteuerungsschicht auch geregelt werden, ob und wie das Fahrzeug auf nur die Gestenerkennung, nur die Spracherkennung oder auf beide reagieren soll.

Ein weiteres Ziel der Architektur ist es, so viele sinnvolle Aufgaben wie möglich auf die CPU bzw. das PS auszulagern, um möglichst viel Platz für die Neural Network (NN) zu lassen. Da es aber keine direkte Verbindung zwischen dem PS und den General Purpose Input/Output (GPIO)-Pins gibt, ist eine Hardware-Brücke notwendig, welche die Pins mit dem PS verbindet (vgl. Abbildung 2.2 auf Seite 12).

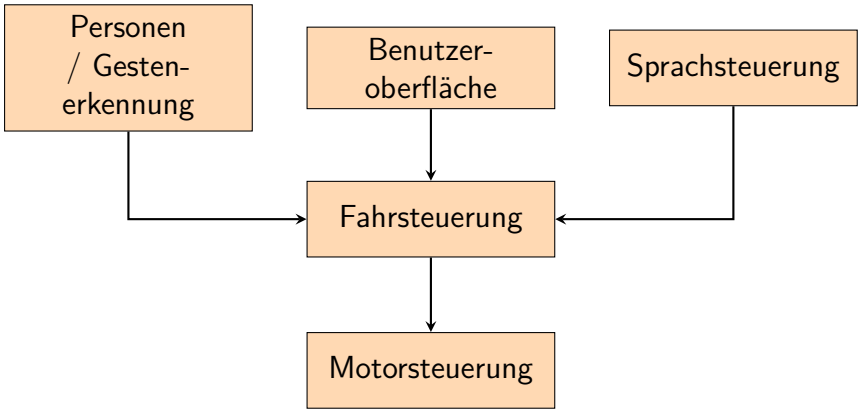


Abbildung 5.4.: Logischer Aufbau der Fahrsteuerung

Tabelle 5.3.: Grundlegende Befehle der untersten Motorsteuerungsschicht

Befehl	Aktion
vorwärts	Das Fahrzeug fährt kontinuierlich nach vorne bis ein anderer Befehl kommt
rückwärts	Das Fahrzeug fährt kontinuierlich entgegen der Fahrtrichtung bis ein anderer Befehl kommt
links	Das Fahrzeug dreht sich kontinuierlich in Fahrtrichtung links bis ein anderer Befehl kommt
rechts	Das Fahrzeug dreht sich kontinuierlich in Fahrtrichtung rechts bis ein anderer Befehl kommt

5.3.4. Ansteuerung des PYNQ-Z2 Boards

geschrieben von Philipp Hennken

mitgearbeitet haben Tim Ranft, Alexander Hoegen-Jupp und Kilian Müller

Zusammen mit der Gruppe der Gestenerkennung wurde probiert, das PYNQ-Z2-Board anzusteuern. Das zunächst trivial scheinende Arbeitspaket erwies sich als zeitintensiver als angenommen. Nach der Verbindung via USB wurde die IP-Adresse des Boards ermittelt, um eine SSH-Verbindung aufzubauen. Während dieses Schrittes traten verschiedene Verbindungsprobleme auf. Eine Schwierigkeit war beispielsweise, dass das eduroam-Netz der Hochschule durch die Anmeldung via Zertifikat keine Einbindung des Gerätes ermöglicht hat. Das Problem wurde durch das

Verwenden privater Hotspots gelöst. Das Ergebnis dieses Arbeitsschrittes ist eine Verbindung via SSH direkt zum PYNQ-Z2-Board.

Anschließend wurden unterschiedliche Tests zur Ansteuerung des Boards mit Python-Skripten durchgeführt. Eine einfache Ansteuerung der Board-LEDs hat funktioniert und im Weiteren wurde das Board auf ansteuerbare Pins analysiert.

5.3.5. Fahrsteuerung der Vorgruppe auf dem Z2-Board zum Laufen bringen

geschrieben von Philipp Hennken

mitgearbeitet haben Alexander Hoegen-Jupp

Nachdem die simple Ansteuerung des Boards funktioniert hat, wurde probiert die Fahrsteuerung der Vorjahre zu kompilieren und auf dem Board auszuführen. Dabei sollte auch ein Vergleich zu dem in den Vorjahren genutzten PYNQ-Z1 Board gezogen werden. Der Versuch scheiterte beim Kompilieren. Sowohl der reguläre als auch der Force-Compile haben nicht funktioniert. Die Hauptursache dafür ist, dass die benötigte Header-Datei „opencv2“ nicht gefunden werden konnte.

- Cross-Compile nach Anleitung durchgeführt
- ./run.sh build_x86 ausgeführt, Docker Container wurde gestartet
- Nicht erreichbare Domain (https://Education/v2/) aufgerufen mit Fehler (no such host)
- Ergebnis: Beide Kompilierungsanleitungen der vorherigen Gruppen funktionieren nicht wie angegeben
- Umweg muss gesucht werden

5.3.6. Neu-Implementation der Fahrsteuerung in Python

geschrieben von Philipp Hennken

mitgearbeitet haben Tim Ranft und Kilian Müller

Wie in Unterabschnitt 5.3.3 beschrieben ist das Ziel, die Fahrsteuerung in mehrere Module aufzuteilen. In dieser Iteration des Projekts wurden zwei Module umgesetzt. Die unterste Schicht ("Engine") und die oberste Schicht ("Server").

Das Server-Modul stellt HTTP-REST-API Endpunkte (vgl. Tabelle 5.4) mit Flask zur einfachen und abstrakten Steuerung des Fahrzeugs zur Verfügung. Der Flask-Server lauscht auf Port 5000 und akzeptiert alle IPs. Dies erlaubt die Ansteuerung des Fahrzeugs von anderen Geräten im gleichen Netzwerk. Überall müssen fortan nur HTTP-Requests zur Ansteuerung verwendet werden. Da dies einen Branchen-Standard umsetzt, hilft dies zur Vereinfachung der Fahrzeugsteuerung für andere Gruppen und nimmt die Komplexität für diese ab.

Die zur Verfügung stehenden Endpunkte des Server-Moduls sind:

Tabelle 5.4.: Verfügbare HTTP-Endpunkte zur Fahrzeugsteuerung

Pfad	**Beschreibung**	**Query-Parameter**
`/forward`	Fährt das Fahrzeug vorwärts	`speed` (optional, Default: 99)
`/backwards`	Fährt das Fahrzeug rückwärts	`speed` (optional, Default: 99)
`/turnleft`	Dreht das Fahrzeug nach links (linke Motoren rückwärts, rechte vorwärts)	`speed` (optional, Default: 99)
`/turnright`	Dreht das Fahrzeug nach rechts (rechte Motoren rückwärts, linke vorwärts)	`speed` (optional, Default: 99)
`/stop`	Stoppt beide Motoren	–
`/drive`	Direktsteuerung der linken und rechten Motoren	`left`, `right` (jeweils -99 bis 99, Default: 0)
`/`, `/<file>`	Statische Auslieferung der Weboberfläche aus dem Verzeichnis `frontend`	–

Das Server-Modul leitet die Requests dann an die zentrale Engine-Instanz, das unterste Modul, weiter.

Dieses Engine-Modul kommuniziert direkt mit der Hardware, indem es auf die physische Motorensteuerung über die Ausgänge des PYNQ-Z2-Boards zugreift. Dafür wurde ein eigenes Overlay erarbeitet, das es ermöglicht, die nötigen Pins anzusprechen. Hier ist die konkrete Steuerungslogik zum Ansteuern der Motoren implementiert. Die Geschwindigkeit lässt sich durch das PWM-Prinzip einstellen. Die PWM-Frequenz beträgt 50 Hz, was einer Periodendauer von 20ms entspricht. Dies erlaubt eine Auflösung der Motorgeschwindigkeit von 1%. Die Methode speed() übersetzt die eingehenden Werte in die Ansteuerung der Motoren. Die 4 Motoren werden nicht alle einzeln angesteuert, sondern es wird nur zwischen den linken und rechten Motoren unterschieden. Die Geschwindigkeit hängt von den Parametern "left" und "right" ab, die von der Server-Schicht gesetzt werden. Die eingehenden Werte sind ganze Zahlen von -99 bis 99. Negative Zahlen stehen dabei für die Rückwärts- und positive Zahlen für die Vorwärtsfahrt. Sind die Werte beider Motoren auf 0, stoppt das Fahrzeug. Dafür kann auch die stop()-Methode verwendet werden. Aufgrund der verbauten Motoren funktioniert die Geschwindigkeitssteuerung nicht so präzise wie gewünscht. So übertragen die Motoren erst genügend Kraft, wenn Werte über 30 bis 35 bzw. unter -30 bis -35 gesendet werden. Dabei fangen die unterschiedlichen Reifen auch erst bei unterschiedlichen Werten an, sich zu drehen.

5.3.7. Benutzeroberfläche zur Steuerung

geschrieben von Kilian Müller

mitgearbeitet haben Tim Ranft

Das folgende Kapitel stellt die Konzeptionierung und Entwicklung der zuvor bereits erwähnten Benutzeroberfläche dar. Aus diesem Grund sind einige Formulierungen enthalten, die zeitlich nicht perfekt mit den anderen Inhalten der Dokumentation übereinstimmen. Grundlegend ist der gesamte Inhalt in diesem Kapitel, sofern nicht anders beschrieben, als bereits implementiert zu betrachten, da die Entwicklung des Frontends abgeschlossen ist.

Um die in Python neu entwickelte Fahrsteuerung als Einzelkomponente zu testen, bevor sie in die Gesamtarchitektur überführt werden kann, ist eine Benutzeroberfläche zu entwickeln, mit welcher das Fahrzeug über einen Joystick gesteuert und die Funktionen der Fahrsteuerung ausgiebig

manuell getestet werden können. Das Frontend soll mithilfe der Sprachen Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und JavaScript entwickelt werden.

Allgemeins Konzept

Hierfür wurde zunächst ein Konzept erstellt, wobei die Benutzeroberfläche als intuitive Fernbedienung dienen soll. Dabei werden die Motorgeschwindigkeiten der linken und rechten Seite separat visualisiert und als Hauptkomponente wird auf einen Joystick gesetzt, welcher in einem Quadrat frei bewegt werden kann. Die Bewegung des Joysticks soll dabei der Bewegung des Fahrzeugs entsprechen. In der Mitte soll es einen Bereich geben, welcher die Funktion „Stop“ abbildet. Bei Bewegung des Joysticks in diesen Bereich wird einmalig der entsprechende API-Endpunkt zur Deaktivierung der Motoren aufgerufen. Außerhalb dieses Bereichs wird die Position des Joysticks erfasst und in Werte für die Geschwindigkeit der beiden Motoren übersetzt, sodass sich das Fahrzeug der Joystick-Position entsprechend bewegt. Dabei wird ein Endpunkt nur dann aufgerufen, wenn sich der Wert im Vergleich zum vorherigen Aufruf verändert hat, da die Fahrsteuerung so aufgebaut ist, dass ein gesendetes Kommando ausgeführt wird, bis ein Neues ankommt. Wenn der Joystick losgelassen wird, springt dieser automatisch in den mittleren Bereich, sodass das Fahrzeug anhält. Dies kann verhindert werden, wenn der sogenannte „Einrastmodus“ eingeschaltet wird. Hier behält der Joystick seine Position auch dann bei, wenn dieser losgelassen wird. Dadurch kann eine bestimmte Bewegungsrichtung oder Geschwindigkeit über einen beliebigen Zeitraum konstant gehalten werden. Die Bedienung der Fernbedienung soll sowohl mit der Maus an Geräten wie Desktop-Computern oder Laptops möglich sein, als auch mithilfe von Touch-Bewegungen am Smartphone. Dafür soll die Benutzeroberfläche möglichst minimalistisch gestaltet und für alle Gerätegrößen entsprechend optimiert werden. Eine Skizze dieser Oberfläche ist in der Abbildung 5.5 zu sehen und wird dem zu schreibenden Prompt beigefügt.

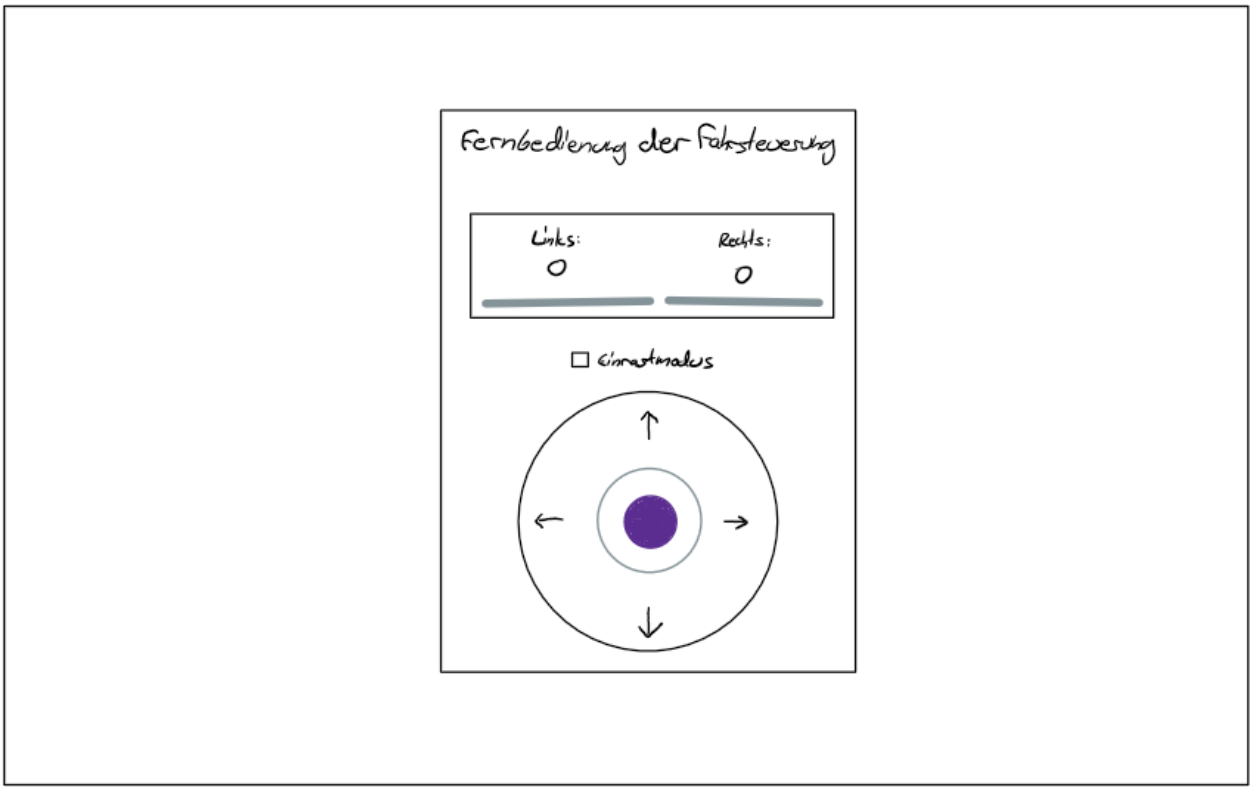


Abbildung 5.5.: Skizze Frontend-Konzept

Verwendung von künstlicher Intelligenz bei der Entwicklung

Da dieses Vorgehen eher als Nebenaufgabe betrachtet wird und nicht zu viel Zeit in Anspruch nehmen soll, wird für ausgewählte Komponenten generative, künstliche Intelligenz unterstützend eingesetzt. Dafür wird das oben beschriebene, allgemeine Konzept zusammen mit dem folgenden Code-Konzept in einen entsprechend optimierten Prompt umgewandelt, welcher die künstliche Intelligenz dazu auffordert den entwickelten HTML-Code zu analysieren und ggf. zu optimieren, eine optisch ansprechende und intuitive Gestaltung mithilfe von CSS umzusetzen und den im Folgenden konzipierten JavaScript-Teil in konkreten Code zu übersetzen. Außerdem sollen wichtige Stellen zur Nachvollziehbarkeit ausführlich kommentiert werden. Hierfür wurde das Modell *Claude Sonnet 4* der Firma *Anthropic* genutzt. Im Repository der Fahrsteuerung lässt sich anhand der Commits außerdem erkennen, welche Änderungen vollständig oder mit Hilfe künstlicher Intelligenz entstanden sind. Die betroffenen Commits tragen den Präfix „[AI]“.

Neben der Skizze und den folgenden Konzepten wird dem Prompt ebenfalls der vorhandene Python-Code der Fahrsteuerung für ein besseres Verständnis zur Verfügung gestellt.

Code-Konzept

Der Code soll zur maximalen Kompatibilität in nativem HTML, CSS und JavaScript-Code geschrieben werden, sodass keine zusätzlichen Paketinstallationen erforderlich werden und der gesamte Code entsprechend klein gehalten wird.

HTML Der Aufbau der HTML-Datei ist trivial und besteht im Wesentlichen aus einem Div-Element, welches alle Elemente der Fernbedienung zentriert auf der Seite darstellt. In diesem Container soll sich eine Überschrift, die beiden Geschwindigkeitsanzeigen nebeneinander, eine Checkbox zur Aktivierung des Einrastmodus sowie ein Eingabefeld zur Festlegung des sogenannten Nullpunkts. Der Nullpunkt ist der Geschwindigkeitswert, ab welchem die Motoren die Reifen des Fahrzeugs tatsächlich zum Bewegen bringen. Er soll über das Frontend dynamisch eingestellt werden können. Am Ende des Containers ist dann der Joystick bzw. die Steuerungsfläche platziert. Die Elemente sollen bereits passende Klassennamen bekommen, für die dann der entsprechende CSS-Code erstellt werden kann.

CSS Die Gestaltung mittels CSS soll eine intuitive Benutzererfahrung sowie ein modernes Erscheinungsbild gewährleisten. Ein helles Farbschema mit den Farben Weiß und Violett, wie in der Abbildung 5.5 zu sehen, und die Anlehnung an ein bekanntes Design-System wie beispielsweise *Material Design* der Firma *Google* ist sinnvoll. Es soll, wie oben bereits beschrieben, nur nativer CSS-Code ohne externe Bibliotheken oder Frameworks genutzt werden. Dadurch wird die Kompatibilität verbessert und der Codeumfang reduziert.

Wichtig ist außerdem die Anpassbarkeit des Frontends für Geräte wie Desktop-Computer, Laptops, Tablets und Smartphones. Hierfür soll sowohl die Bedienung per Maus, als auch per Touch ermöglicht werden. Die Klassennamen des bestehenden HTML-Codes sollen genutzt und gegebenenfalls verbessert werden, wenn die Semantik davon profitiert. Der Code sollte klar zwischen Layout, Funktion und Interaktion trennen und die Modularität und Erweiterbarkeit im möglichen Rahmen sichern.

Die Steuerungsfläche soll quadratisch sein, um größere Bewegungsspielräume zu ermöglichen. Außerdem soll es einen farblich hervorgehobenen Bereich für die Totzone geben.

JavaScript Die JavaScript-Datei soll aus einer Klasse mit dem Namen *JoystickRemote* bestehen. In einem Konstruktor sollen zentrale Variablen wie die URL des Python-Servers, das Abfrageintervall, relevante HTML-Elemente und Zustände des Joysticks deklariert und initialisiert. Am Ende des Konstruktors muss zudem die Initialisierung des Joysticks bzw. des Quadrats (Berechnung der Größe, Zentrierung des Handles) vorgenommen werden.

Außerdem müssen die sogenannten *EventListeners* aufgesetzt werden. Diese sind unter anderem dafür da, die Touch- und Mausbewegungen zu erkennen und die darauf basierenden Berechnungen in anderen Funktionen auszuführen. Diese Initialisierungen sollten am besten in eine oder zwei getrennte Funktionen ausgelagert werden, um die Übersichtlichkeit zu verbessern. Die zuvor erwähnten Berechnungen sollten in weitere Funktionen aufgeteilt sein. Wichtige Funktionen sind hierbei das Aktualisieren der Position des Joysticks, die Berechnung der zu sendenden Motorgeschwindigkeiten aufgrund der aktuellen Position, das Aktualisieren der Geschwindigkeitsanzeige und der Aufruf der API, um die berechneten Daten an den Server zu senden.

Die Bestimmung der API-URL soll dynamisch anhand der aktuell eingegebenen URL in der Adresszeile des Browsers stattfinden. Dafür soll das genutzte Protokoll und die IP-Adresse mithilfe von JavaScript-Code extrahiert und mit dem Port 5000 ergänzt werden. Dies fungiert dann als Basis-URL an welche die einzelnen Endpunkte angehangen werden können.

Die Geschwindigkeit, die an die API gesendet wird, soll generell anhand der Position des Joysticks innerhalb des Quadrats ermittelt werden. Dabei stehen alle Punkte oberhalb der horizontalen Achse für positive, alle Punkte unterhalb für negative Werte. Eine Ausnahme gilt nur für den unmittelbaren Bereich um die horizontale Achse, welche im weiteren Verlauf erläutert wird. Um die Position des Joysticks zu ermitteln soll der Abstand und der Winkel zum Mittelpunkt genutzt werden. Ein größerer Abstand zum Mittelpunkt steht dabei grundsätzlich für eine größere Geschwindigkeit.

Die Daten sollen mit einem Mindestabstand von 100 ms und nur bei Änderungen an die API gesendet werden. Solange der Joystick in einer Position verharrt, soll das Frontend nicht mit der API kommunizieren.

Auf der Steuerungseinheit soll es grundsätzlich die vier Richtungen „Vorwärts“, „Rückwärts“, „Links“ und „Rechts“ geben. Wenn der Joystick nur auf der vertikalen Achse bewegt wird, also vorwärts oder rückwärts, werden beide Motoren synchron und mit identischer Geschwindigkeit angesteuert. Auch hier gilt, dass die Höhe der Geschwindigkeit mit dem Abstand zum Mittelpunkt berechnet wird. Die Richtungen „Links“ und „Rechts“ stellen den zuvor erwähnten Sonderfall dar. Das Fahrzeug soll sich bei diesen Richtungen auf der Stelle drehen, wobei hier beide Motoren denselben Betragswert erhalten sollen, je nach Abstand zur Mitte. Der Motor, der auf der Richtungsseite liegt, erhält dabei den negativen Wert, der Motor auf der anderen Seite den positiven Wert. Diese Berechnung soll lediglich in einem kleinen Korridor um die horizontale Achse Anwendung finden. Für den Rest der Steuerungseinheit gilt eine andere Berechnungslogik, welche zuvor bereits erwähnt wurde.

Für Positionen, die nicht direkt auf den Achsen liegen, gilt grundsätzlich, dass immer der Motor auf der Seite der Bewegungsrichtung schwächer angesteuert wird als der andere, damit das Fahrzeug auch in die entsprechende Richtung bewegt wird. Je näher der Joystick an der horizontalen Achse platziert ist, desto größer wird der Abstand zwischen den beiden Motorgeschwindigkeiten, da die Geschwindigkeit des stärkeren Motors bei gleichem Abstand zur Mitte konstant bleibt, während die Geschwindigkeit des schwächeren Motors linear abnimmt.

Der zuvor erwähnte Nullpunkt soll dynamisch zur Laufzeit angepasst werden können, da sich der Schwellenwert für die Drehung der Reifen eventuell von Fahrzeug zu Fahrzeug unterscheidet. Mit

dem getesteten Modell liegt der Schwellenwert bei ungefähr 30, weshalb dieser Wert als Standard gesetzt werden soll. Die Werte, die an die API geschickt werden, sollen mit Ausnahme der 0 nur zwischen dem Nullpunkt und dem Maximalwert liegen, wobei auch die jeweiligen negativen Äquivalente erlaubt sind, um das Auto rückwärts fahren zu lassen. Der Maximalwert liegt in diesem Fall bei 99 bzw. -99. Damit die Darstellung der Geschwindigkeiten im Frontend einheitlich ist, wird dem Nutzer stets der vollständige Wertebereich zwischen der 0 und Maximalwert angezeigt. Bevor die Werte an die API geschickt werden, müssen diese anhand des Nullpunktes neu berechnet werden, sodass der zuvor formulierte Wertebereich eingehalten wird. So wird gewährleistet, dass sich an der gewohnten Bedienung des Frontends unabhängig vom Nullpunkt des Fahrzeugs bzw. Motors für den Nutzer nichts ändert. Die an die API gesendeten Geschwindigkeiten sollen zudem in einem kleinen Debug-Feld unter den anderen Geschwindigkeiten angezeigt werden.

Des Weiteren ist es wichtig, dass das Senden des Stopp-Signals gewährleistet wird, sobald sich der Joystick in der Mitte bzw. innerhalb der Totzone befindet, damit das Fahrzeug nie ungewollt fährt. Dadurch werden potenzielle Unfälle vermieden, die insbesondere bei unerfahrenen Nutzern vermehrt auftreten können. Dies muss auch gesichert sein, wenn der Joystick mit sehr schnellen Bewegungen oder genau während eines anderen API-Aufrufs zur Mitte bewegt wird.

Die zuvor erwähnte Totzone soll außerdem so abgebildet werden, dass die Position des Joysticks und damit die Werterfassung erst beginnt, wenn der Joystick mit mindestens der Hälfte seiner Fläche über die Totzone hinausragt.

Python Das Frontend soll über den Python-Server ausgeliefert werden, um einen zusätzlich geöffneten Port bzw. parallel laufenden Server zu vermeiden. Dafür muss der ursprüngliche Code um zwei Endpunkte erweitert werden. Zunächst soll ein Endpunkt für die Route /<*path:filename*> erstellt werden. Dieser soll die Funktion *send_from_directory* nutzen, um die Dateien aus dem Frontend-Ordner auszuliefern. Das ist nötig, damit die HTML-Datei, welche vom neuen Endpunkt mit der Route / ausgeliefert wird, Zugriff auf die CSS- und JavaScript-Datei hat. Die Benutzeroberfläche läuft dann auf dem Port des Servers und ist ohne speziellen Pfad aufrufbar.

Vorgehen und Ergebnis

Um die oben beschriebenen Konzepte in konkreten Code umzusetzen wurde die Entwicklung mit Unterstützung von künstlicher Intelligenz in einem iterativen Prozess durchgeführt. Dabei wurde die Gesamtaufgabe in kleinere Aufgabenpakete geteilt und diese Schritt-für-Schritt umgesetzt, bis die Anforderungen erfüllt waren. So konnte auch der generierte Code durch die kleineren Zwischenergebnisse gründlicher geprüft werden. Die Aufgabenpakete waren im Wesentlichen aufgeteilt in die Umsetzung des HTML-Codes, des CSS-Codes und des JavaScript-Codes. Außerdem wurden kleinere Fehler oder übersehene Lücken in der Anforderungsumsetzung zwischendurch gelöst.

Die Benutzeroberfläche ist so zu bedienen, wie es in Kapitel 5.3.7 beschrieben ist. In Abbildung 5.6 ist das fertige Frontend zu erkennen.

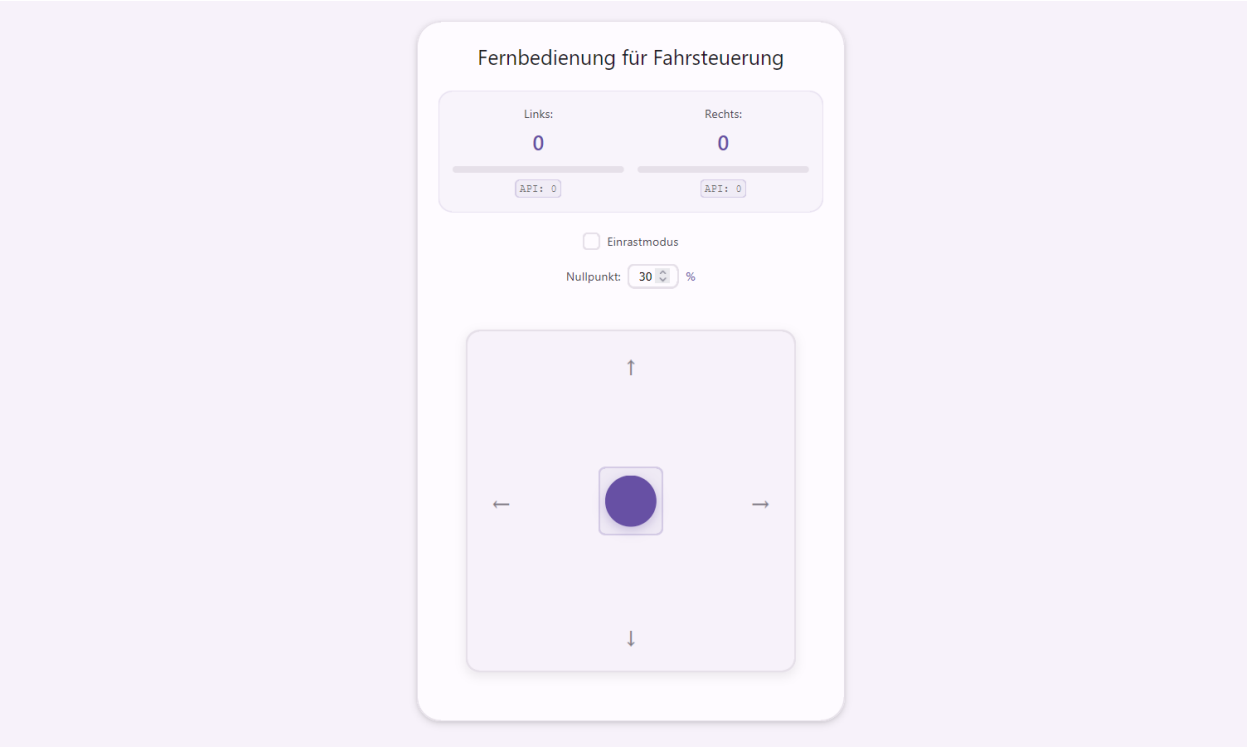


Abbildung 5.6.: Fertige Benutzeroberfläche

5.4. Nächste Schritte

geschrieben von Tim Ranft

Damit die nächste Iteration der Fahrsteuerung schnell in die Bearbeitungsphase kommt, sind im Folgenden die aus unserer Sicht nächsten Schritte aufgeführt, um dem Projektziel näherzukommen.

5.4.1. Integration der Fahrsteuerung parallel zu anderen Schaltungsteilen

geschrieben von Tim Ranft

Für den FPGA-Chip kann maximal ein Bitstream gleichzeitig geladen werden. Dieser Bitstream wird jedoch künftig von mehreren Gruppen generiert, da das FPGA die Verschaltung mehrerer Teilkomponenten übernehmen soll. Ziel ist eine Integration und Optimierung der einzelnen Logikschaltungen, sodass ein einzelner Bitstream herauskommt und die einzelnen Anwendungen gleichzeitig auf die Berechnungslogik vom FPGA zugreifen können.

Darüber hinaus soll es möglich sein, im Vivado Block-Design einen oder mehrere *Hierarchical Blocks* als *Partial-Reconfig-Regionen* zu definieren. Später wird dann ein einziges Overlay geladen und reservierte Bereiche/Blocks darin aus anderen Quellen (dynamisch) geladen. Dies wurde in Anbetracht der Zeit in dieser Iteration nicht weiter verfolgt.

5.4.2. Optimierung der Geradeausfahrt

geschrieben von Tim Ranft und Philipp Hennken

Wie in Unterabschnitt 5.2.2 bereits erwähnt, hat das Auto massive Probleme bei der Geradeausfahrt. Grund für diese Probleme sind die verwendeten Komponenten. Um eine Geradeausfahrt zu realisieren, müssen alle Teile sehr präzise und fest verbaut sein. Dies trifft im Moment auf die Motoren und Reifen nicht zu. Ebenfalls ist nicht garantiert, dass sich alle Motoren exakt gleich verhalten, was mit der Verkabelung der Motoransteuerung zusammenhängt. Die Gewichtsverteilung wird darüber hinaus auch einen Einfluss haben und muss dementsprechend angepasst werden. Eine mechanische Lösung fordert unseres Erachtens nach neue, hochwertigere Bauteile und eine sorgfältigere Montage dieser.

Eine andere Lösungsmöglichkeit des Problems wäre eine softwaregesteuerte Lösung. Beispielsweise besteht die Möglichkeit neue Sensoren zu verbauen, welche die Beschleunigung und Richtung des Autos unabhängig von der zurückgelegten Fahrstrecke bzw. den Umdrehungen der Motoren messen können. Eine erste angedachte Lösung war eine IMU (Inertial Measurement Unit). Da eine Bestellung eines solchen Sensors zu lange gedauert hätte, wurde es nicht umgesetzt und nicht weiter evaluiert.

Ein anderer Ansatz besteht in der Kalibrierung der Einzelkomponenten. Bekommt das Auto den Steuerbefehl einer Geradeausfahrt mit 50 % der Maximalleistung, so könnte die unterste Ebene der Fahrsteuerung ein Discountfaktor für die entsprechende Leistung auf Modulationsebene vornehmen. Zu beachten ist, dass sich dieser Faktor bei unterschiedlichen Geschwindigkeiten anders verhalten kann und zusätzlich nicht nur jeder Reifen unterschiedlich dreht, sondern auch jedes individuelle Auto unterschiedliche mechanische Anforderungen mitbringt. Eine Kalibrierung hat den Vorteil, dass der Faktor vor Fahrtantritt für jedes Auto ermittelt werden kann, auch wenn dies nicht die genauste Methode sein wird.

5.4.3. Neue Endpunkte für den Server

geschrieben von Philipp Hennken

Das Server-Modul könnte um weitere Endpunkte erweitert werden, die insgesamt „intelligenter“ sind. Dazu könnten Befehle gehören wie eine Drehung um einen bestimmten Grad-Wert oder die Bewegung mit einer gewissen Geschwindigkeit, die man in einer bestimmten Geschwindigkeits-Einheit angeben kann. Jene neuen Endpunkte können zukünftig relevant für Anforderungen anderer Gruppen sein. Genaue Endpunkte und deren Implementation sollten mit der Gruppe für die Gesamtarchitektur und mit den betroffenen Gruppen konkret definiert werden.

Für die Umsetzung dieser neuen Endpunkte könnten die im Kapitel 5.4.2 benannten, neuen Sensoren verwendet werden.

6. Evaluation des neuen Mikrofons

geschrieben von Max Tepe, Frederic Goretzky, Milan Becker und Simon Koger

Im Vergleich zu dem Vorgängermodell, dem PYNQ-Z1 Board verfügt das neu eingebaute Pynq-Z2 Board über das auf dem Board verbauten Audio Codec analog Gerät ADAU1761 [TUL18], welches Audiodateien eines externen Mikrofons über einen 3,5 mm TRRS Klinkenstecker komprimieren und dekodieren kann. Bei der vorherigen Projektdurchführung aus dem Wintersemester 24/25 wurde die Sprachaufnahme noch über ein Headsetmikrofon aufgenommen und das BaseOverlay verwendet um die Audio Signale als Datei abzuspeichern.[Bre25]

Mit der Verwendung eines neuen Mikrofons wird erwartet, dass sich die Audioqualität verbessert, Störgeräusche wie bspw. die Motorengeräusche durch die unterstütze Software Sigma Studio [TUL18] gefiltert werden können und generelle weniger Störgeräusche aufgenommen werden. Aus diesem Grund wurde eine ausführliche Evaluierung geeigneter Mikrofone vorgenommen.

Anforderungen an das Mikrofon

Für die Integration in das autonome Fahrzeug ergeben sich spezifische Anforderungen an das verwendete Mikrofon. Zur Herleitung der Anforderungen wurden sich Use Cases überlegt. Aufgrund der zur Verfügung stehenden Ressourcen wurden sich lediglich zwei Use Cases überlegt:

UC-01: Mikrofon im Fahrzeug integrieren *Ziel:* Mikrofon wird korrekt montiert und an den Klinkenanschluss angeschlossen. *Akteur:* Projektmitglied *Ablauf:* Mikrofon montieren, über 3,5 mm TRRS anschließen.

UC-02: Sprachaufnahme vor dem Auto mit Abstand *Ziel:* Klare Sprachaufnahme trotz Motorgeräuschen. *Akteur:* Benutzer *Ablauf:* Der Benutzer steht über 1 m entfernt von dem Auto und gibt diesem ein Sprachbefehl. Das Auto reagiert auf den Sprachbefehl und reagiert dementsprechend.

UC-03: Sprachaufnahme hinter dem Auto mit Abstand *Ziel:* Klare Sprachaufnahme trotz Motorgeräuschen auch hinter dem Auto möglich. *Akteur:* Benutzer *Ablauf:* Der Benutzer steht hinter dem Auto und gibt diesem ein Sprachbefehl. Das Auto reagiert auf den Sprachbefehl und reagiert dementsprechend.

Aus diesen Use-Cases lassen sich folgende funktionalen Anforderungen ableiten:

1. Das Mikrofon soll über einen 3,5 mm TRRS-Klinkenanschluss betrieben werden können.
2. Es soll keine zusätzliche Spannungsversorgung (z. B. Batterie, USB) benötigen.
3. Das Mikrofon soll mit dem ADAU1761 Audio-Codec kompatibel sein.
4. Das Mikrofon soll Störgeräusche, insbesondere Motoren, möglichst gut unterdrücken können.

5. Das Mikrofon muss eine klar verständliche Audioaufnahme liefern (Sprachverständlichkeit im Fokus).
6. Das Mikrofon soll mechanisch, vibrationsarm montierbar sein.
7. Das Mikrofon soll nicht zu empfindlich gegenüber Umgebungsgeräuschen sein (z. B. Motorengeräusche).
8. Das Mikrofon soll Sprachquellen zuverlässig auch auf weitere Entfernung aufnehmen sein.
9. Das Mikrofon soll Sprachquellen sowohl von vorne als auch hinten aufnehmen.
10. Das Mikrofon muss zuverlässig auch leisere Stimmen aufnehmen.

Methodik der Mikrofon-Evaluation

Zur Auswahl eines geeigneten Mikrofons wurden zunächst verschiedene Modelle aus dem semiprofessionellen und professionellen Bereich recherchiert. Die Bewertung erfolgte anhand technischer Datenblätter und Herstellerangaben. Für die Bewertung der Mikrofone wurden folgende Kriterien verwendet:

- Richtcharakteristik
- Empfindlichkeit
- Montagemöglichkeit
- benötigte externe Versorgung
- Kompatibilität mit dem 3,5 mm TRRS-Anschluss
- Preis

Richtcharakteristik von Mikrofonen

Der folgende Abschnitt basiert auf den Ausführungen von Bernstein und fasst die wesentlichen Punkte zur Richtcharakteristik von Mikrofonen zusammen[Ber19].

Die Richtcharakteristik eines Mikrofons beschreibt, aus welchen Richtungen und wie empfindlich es Schall aufnimmt. Sie ist ein zentrales Auswahlkriterium, da sie maßgeblich beeinflusst, wie gut ein Mikrofon gewünschte Schallquellen erfassen und unerwünschte Störgeräusche unterdrücken kann.

Zu den für unseren Anwendungsfall wichtigsten Richtcharakteristiken zählen:

- *Omnidirektional (Kugelcharakteristik):* Mikrofone mit dieser Charakteristik nehmen Schall gleichmäßig aus allen Richtungen auf. Sie eignen sich für natürliche Klangabbildung, sind aber anfällig für Umgebungsgeräusche.
- *Niere (Kardioid):* Diese Mikrofon-Richtcharakteristik ist am empfindlichsten für Schall von vorne, während seitliche Geräusche abgeschwächt werden. Rückwärtige Geräusche werden bei dieser Richtcharakteristik besonders schwach aufgenommen. Sie sind besonders für Sprachaufnahmen geeignet, wenn Umgebungsgeräusche reduziert werden sollen.

- *Superniere/Hyperniere:* Diese Formen bieten eine noch stärkere Fokussierung auf die Frontalrichtung und unterdrücken seitliche Geräusche besonders effektiv. Anders als bei der Nierencharakteristik hingegen nehmen sie ebenfalls rückwärtige Geräusche abgeschwächt auf. Sie sind ideal für den Einsatz bei lauten seitlichen Störgeräuschen, wie sie im Fahrzeuginnenraum mit Motorgeräuschen auftreten.
- *Keule (Lobe/Lobar):* Mikrofone mit Keulencharakteristik besitzen eine äußerst gerichtete Aufnahme, die stark auf die Frontalachse fokussiert ist. Sie weisen eine noch engere Richtwirkung als Supernieren auf und blenden seitliche sowie rückwärtige Schallquellen weitgehend aus. Diese Charakteristik kommt vor allem bei Shotgun-Mikrofonen zum Einsatz. In lauten oder hallenden Umgebungen ermöglichen sie eine gezielte Sprachaufnahme aus größerer Entfernung, vorausgesetzt die Schallquelle befindet sich exakt in der Achse. Für den Einsatz an einem autonomen Fahrzeug kann diese Charakteristik von Vorteil sein, wenn Sprachbefehle frontal und aus größerer Entfernung gegeben werden, allerdings erfordert sie eine exakte Ausrichtung des Mikrofons.

Die Auswahl einer geeigneten Richtcharakteristik ist für die Anwendung im autonomen Fahrzeug entscheidend, da sie die Aufnahmequalität und die Störgeräuschunterdrückung maßgeblich beeinflusst.

Empfindlichkeit und Signal-Rausch-Verhältnis von Mikrofonen

Die Empfindlichkeit eines Mikrofons beschreibt das Verhältnis zwischen der erzeugten elektrischen Ausgangsspannung und dem Schalldruck, der auf das Mikrofon trifft. Sie wird meist in Millivolt pro Pascal (mV/Pa) oder in Dezibel Volt relativ zu 1 Pascal dbV/Pa angegeben. Eine höhere Empfindlichkeit bedeutet, dass das Mikrofon bereits bei geringem Schalldruck eine stärkere elektrische Spannung erzeugt, was insbesondere bei leisen Schallquellen vorteilhaft ist [Ear12].

Vergleich der Mikrofonmodelle

Im Rahmen der Evaluation wurden verschiedene Mikrofone miteinander verglichen und eine erste Auswahl für eine Evaluierung getroffen. Unter der Auswahl sind Lavaliermikrofone wie das RØDE smartLav+, das BOYA BY-M1 und das Audio-Technica ATR3350xiS sowie Richtmikrofone wie das COMICA VM10 PRO und das Hama RMN Uni.

Die Lavaliermikrofone zeichnen sich durch eine omnidirektionale Richtcharakteristik aus, welche insbesondere bei Sprachaufnahmen im Nahbereich Vorteile bietet, allerdings auch eine höhere Anfälligkeit für Umgebungsgeräusche mit sich bringt. Richtmikrofone wie das Hama RMN Uni und COMICA VM10 Pro verfügen über eine Supernierencharakteristik, die eine gezielte Unterdrückung seitlicher und rückwärtiger Störquellen ermöglicht und somit in lärmintensiven Umgebungen, wie dem Fahrzeuginnenraum, besonders effektiv ist.

Zur Ergänzung wurden weitere Mikrofone berücksichtigt, darunter Modelle mit Niere- und Kugelcharakteristik, wie das Shure SM58 (Niere) und das Behringer ECM80000 C414 (Kugel). Diese bieten unterschiedliche Vorteile bezüglich Richtwirkung und Aufnahmequalität und erweitern die Bewertungsskala in der Kriterienmatrix.

Zur systematischen Bewertung der Mikrofone wurde eine Kriterienmatrix erstellt, die technische Spezifikationen, Richtcharakteristik, Empfindlichkeit, Signal-Rausch-Abstand sowie Monta-

gefreundlichkeit und Preis gegenüberstellt. Diese Matrix dient als Grundlage für die fundierte Auswahl des am besten geeigneten Mikrofons für den Einsatz im autonomen Fahrzeug.

Tabelle 6.1.: Kriterienmatrix zur Evaluierung der Mikrofone

Modell	Richt.	Empf. *	Mont.	Vers.	TRRS	Preis**	Quelle
RØDE smartLav+	Omnidirectional	-35 dBV/Pa	Ansteckbar	Passiv	Ja	99,99 €	[Ltd25a]
BOYA BY-M1	Omnidirectional	-30 dBV/Pa	Ansteckbar	1,5 V	Ja	17,95 €	[Ltd25b]
Audio-Technica ATR3350xiS	Omnidirectional	-54 dBV/Pa	Ansteckbar	1,5 V	Ja	39,99 €	[Ltd25c]
COMICA VM10 PRO	Superniere	-38 dBV/Pa	Shock-mount	Passiv	Ja	69,00 € ***	[She25]
Hama RMN Uni	Superniere	-32 dBV/Pa	Anti-Shock-Frame	Passiv	Ja	68,00 €	[KG25a]
Shure SM58	Niere	-54,5 dBV/Pa	Handmikrofon	Passiv	Nein, XLR	119,00 €	[Gmb14]
Behringer ECM8000	Omnidirectional	-70 dBV/Pa	Handmikrofon	15-48 V	Nein, XLR	22,00 €	[KG25b]
Sennheiser ME66	Superniere/Keule	-26,5 dbV/Pa	Handmikrofon	12-48 V	Nein, K6 oder K6P	199,00 € ***	[KG25c]

Abkürzungen: Richt. = Richtcharakteristik, Empf. = Empfindlichkeit, Mont. = Montageart, Vers. = Versorgung, TRRS = direkte 3,5 mm TRRS-Kompatibilität

* Alle Preise und Informationen sind vom Stand 18.07.2025.

** Alle Empfindlichkeitsangaben in dBV/Pa bei 1 kHz und 1 Pa, soweit in den Herstellerangaben nicht anders spezifiziert.

*** Preisangaben von Webshops: Amazon, meinmic.com, Stand 18.07.2025.

Begründung der Mikrofonauswahl

Für die Integration eines Mikrofons in das autonome Fahrzeug wurden spezifische Anforderungen auf Basis der definierten Use Cases (UC-01 bis UC-03) abgeleitet. Diese Anforderungen (siehe Abschnitt 6) stellen zusammen mit der Kriterienmatrix (siehe Tabelle 6.1 die Grundlage für die Evaluierung und Auswahl des Mikrofons dar.

Das *Hama RMN Uni* erfüllt die zentrale Anforderung, dass das Mikrofon über einen 3,5 mm TRRS-Klinkenanschluss betrieben werden kann, was eine direkte Kompatibilität mit dem verwendeten ADAU1761 Audio-Codec sicherstellt. Im Gegensatz dazu besitzen einige Modelle, wie das Shure SM58 oder das Behringer ECM8000, nur XLR-Anschlüsse, was zusätzliche Adapter oder Schnittstellen erfordert und die Integration erschwert.

Eine weitere wichtige Voraussetzung ist, dass das Mikrofon *keine externe Spannungsversorgung benötigt*, da im Fahrzeug keine zusätzliche Energiequelle für das Mikrofon vorgesehen ist. Das Hama RMN Uni ist passiv versorgt, im Gegensatz zu Kondensatormikrofonen wie dem Sennheiser ME66 oder Behringer ECM8000, die Phantomspannung benötigen und dadurch den Integrationsaufwand erhöhen würden.

Zur Erfüllung der Use Cases UC-02 und UC-03, die klare Sprachaufnahmen trotz Motorgeräuschen bei einem Abstand von über 1 Meter vor und hinter dem Fahrzeug fordern, ist eine Mikrofoncharakteristik entscheidend, die Störgeräusche möglichst gut unterdrückt. Das Hama RMN Uni besitzt eine *Supernieren-Richtcharakteristik*, welche gezielt Schallquellen aus einer Richtung aufnimmt und somit Umgebungsgeräusche wie Motorenlärm effektiv minimiert. Des weiteren nimmt das Hama RMN Uni auch leicht abgeschwächt Geräusche von hinten war. Im Gegensatz dazu erfassen omnidirektionale Mikrofone wie das RØDE smartLav+ oder das BOYA BY-M1 alle Schallquellen gleichmäßig, was die Sprachverständlichkeit im lauten Umfeld stark beeinträchtigen kann.

Die Montageanforderung wird durch den mitgelieferten *Anti-Shock-Frame* des Hama RMN Uni optimal erfüllt, der mechanische Erschütterungen und Fahrzeugvibrationen dämpft. Dies ist ein Vorteil gegenüber Ansteckmikrofonen wie dem COMICA VM10 PRO oder RØDE smartLav+ und

Handmikrofonen wie beispielsweise dem Shure SM58, die oft keine eigene Vibrationsdämpfung bieten und dadurch anfälliger für Störgeräusche durch Bewegung sind.

Bezüglich der *Empfindlichkeit* bietet das Hama RMN Uni mit etwa -32 dBV/Pa eine ausgewogene Sensibilität. Es ist empfindlich genug, um auch leisere Stimmen zuverlässig aufzunehmen und Sprachquellen auch bei einem Meter Abstand klar zu erfassen, ohne jedoch durch zu hohe Empfindlichkeit verstärkt störende Umgebungsgeräusche aufzunehmen. Mikrofone wie das Sennheiser ME66 sind zwar sehr empfindlich -26,5 dBV/Pa, was die Aufnahmequalität verbessert, aber gleichzeitig auch Motorengeräusche verstärken kann.

Die Richtcharakteristik des Hama RMN Uni erlaubt darüber hinaus, dass Sprachquellen sowohl von vorne als auch von hinten aufgenommen werden können, was es für die in den Use Cases spezifizierten Sprachbefehle vor und hinter dem Fahrzeug prädestiniert. Die Supernierencharakteristik stellt hier einen guten Kompromiss zwischen Richtwirkung und seitlicher Aufnahmefähigkeit dar.

Zusätzlich überzeugt das Hama RMN Uni durch ein *attraktives Preis-Leistungs-Verhältnis* und praktische Montageeigenschaften, die es für den geplanten Einsatz im autonomen Fahrzeug besonders geeignet machen. Zusammenfassend erfüllt das Mikrofon alle wesentlichen Anforderungen und übertrifft viele der alternativen Modelle im Hinblick auf die praktische Nutzbarkeit im Projektkontext.

7. Audioerkennung per neuronalem Netz

geschrieben von Simon Köhler

Die Audioerkennung per neuronalem Netz soll dazu verwendet werden, bestimmte Sprachbefehle des Nutzers zu erkennen und basierend darauf Steuerungsaktionen auf dem Fahrzeug durchzuführen. Die Steuerungsbefehle sind *back*, *forward*, *left*, *right* und *stop*. Diese Iteration des Projektes baut auf den Arbeiten der vorherigen Durchläufe auf. Der Fokus lag in diesem Semester darauf, die Arbeiten der vorhergegangenen Semester einzusortieren, zu bewerten und einen soliden Grundstein für die kommenden Jahrgänge anzufertigen, sodass die Einarbeitungszeit in Zukunft minimiert werden kann.
Es wurden jedoch auch technische Fortschritt erzielt. So wurden grundlegende Fehler in den relevanten Arbeiten der Vorgänger entdeckt, diese teils behoben und auch konkrete Lösungs- und Implementationsvorschläge gegeben. Eine der größten Inkremente ist hierbei das Identifizieren des Problems für die schlechte Erkennungsrate während der Live Inferenz. In diesem Durchlauf wurde der Code nicht auf der Hardware ausgeführt. Der Code aus dem zugehörigen Repository wurde auf den privaten Geräten der Studierenden ausgeführt und nicht auf dem Mikrocontroller oder dem FPGA des Fahrzeuges.

7.1. Grundlagen

7.1.1. Neuronale Netze

geschrieben von Simon Köhler

Neuronale Netze sind ein Teilgebiet des maschinellen Lernens und orientieren sich in ihrer Struktur und Funktionsweise grundlegend am menschlichen Gehirn. Ein neuronales Netz besteht aus mehreren künstlichen Neuronen, die in Schichten angeordnet sind. Diese Neuronen kommunizieren über Verbindungen miteinander. Jeder Verbindung ist ein Gewicht zugeordnet, welches bei den Berechnungen im Netz berücksichtigt wird. Neuronale Netze eigenen sich dafür, aus Trainingsdaten zu lernen und später in anderen Daten die gelernten Muster zu erkennen und Vorhersagen oder Klassifikationen zu treffen. Im Folgenden werden die Grundlagen von neuronale Netzen anhand der Architektur eines Feed Forward Netzes erläutert. Es existieren noch viele weitere Architekturen von neuronalen Netzen, die sich in der Anordnung von Neuronen und der Anzahl und Art der Verbindungen unterscheiden. Dazu zählen zum Beispiel rekurrente oder konvolutionale Netze. Die hier erläuterten Grundlagen wie Gewichtung und Aktivierungsfunktionen sind jedoch übertragbar.

Zuerst muss ein neuronales Netz trainiert werden. Während des Trainingsprozesses werden die Gewichte des neuronalen Netzes so angepasst, dass es möglichst genaue Vorhersagen oder Klassifikationen auf Grundlage der Eingabedaten treffen kann. Dazu muss ein Trainingsdatensatz zur Verfügung stehen. Dieser Trainingsdatensatz enthält Paare aus den Eingabewerten sowie den zugehörigen erwarteten Ausgaben. Im ersten Schritt, der Forwardpropagation werden die Daten

durch das Netz geführt und ein Ergebnis berechnet. Dieses berechnete Ergebnis wird mit dem tatsächlichen Wert verglichen woraus sich ein Fehler ergibt. Wie schwerwiegend dieser Fehler ist, wird durch eine Loss-Funktion berechnet. Im nächsten Schritt, der Backpropagation, wird unter Verwendung eines Gradientenabstiegsverfahren berechnet, wie die Gewichte angepasst werden müssen, damit der Fehler in zukünftigen Durchläufen minimiert wird [Sel24]. ein ganzer Trainingsprozess wird auch als Epoche bezeichnet und so häufig wiederholt, bis das Modell eine zufriedenstellende Erkennungsrate erreicht hat.

Ein Deep Feed Forward Netz ist in drei Arten von Schichten gegliedert [Boh25]. In der Eingabeschicht, auch Input Layer genannt, werden die Eingabewerte eingelesen. Jedes Neuron der Eingabeschicht ist mit jedem Neuron der zweiten Schicht, der Hidden Layer verbunden. Häufig haben neuronale Netze mehr als eine versteckte Schicht. Je mehr versteckte Schichten, desto höher ist das potential des Netzes komplexe Muster in den Daten zu erkennen. Allerdings steigt damit auch die Komplexität und somit die Hardwareanforderungen. Ebenfalls steigt mit erhöhter Netzkomplexität die Gefahr beim Trainieren zu overfitten. Beim Overfitting passt sich das Netz zu stark an die Trainingsdaten an und generalisiert nicht mehr was dazu führt, dass neue Daten schlechter erkannt werden. Nachdem die Daten von der Input Layer an die erste Hidden Layer weitergegeben worden sind, werden sie mit den Gewichten der Verbindungen zwischen den Neuronen multipliziert und die Ergebnisse addiert. Anschließend wird das Ergebnis an eine Aktivierungsfunktion übergeben, die berechnet, ob und wie stark das Neuron ein Signal an die nächste Schicht weiterleitet [Ert25]. Dieser Vorgang wird für die Anzahl der versteckten Schichten untereinander bis zur letzten versteckten Schicht durchgeführt. In der letzten Hidden Layer werden die Signale an die Output Layer weitergegeben. In der Output Layer werden dann die Signale aus der vorherigen Schicht ein letztes Mal gewichtet, aufsummiert und durch die Aktivierungsfunktion verarbeitet. Das Ergebnis der Output Layer stellt dann das Ergebnis des neuronalen Netzes dar.

7.1.2. Mono und Stereo Sound

geschrieben von Simon Köhler

Bei der Aufnahme, Speicherung und Wiedergabe von Audiodateien gibt es grundsätzlich zwei verschiedene Arten. Zum Einen lässt sich Sound monophon verarbeiten. Dies wird auch als Mono Sound bezeichnet und bedeutet, dass nur ein Audiokanal verwendet wird. Dem Gegenüber steht die stereophonische Soundverarbeitung. Hier werden zwei Kanäle, ein linker und ein rechter, verwendet um den Ton einzufangen. Durch die Teilung der Aufnahme in zwei Kanäle entsteht ein räumlicherer Klang, wo hingegen Stereo Sound flacher wirkt [www25].

Neben den akustischen Unterschieden gibt es jedoch auch Verschiedenheiten in der Speicherung der Formate, die für die Umsetzung der Sprachsteuerung wichtig sind. Für die Speicherung von rohen Audiodateien wird häufig Pulse Code Modulation (PCM) verwendet. Dabei werden die analogen Amplituden des Audiosignals in regelmäßigen Zeitabständen, definiert durch die Abtastrate, zum Beispiel $44,1\ kHz$, in diskrete digitale Werte umgewandelt wodurch eine portable, unabhängige Speicherung ermöglicht wird [BG03]. Bei Stereo Sound werden der linke sowie der rechte Audiokanal abwechselnd im Audioarray gespeichert. So ergibt sich die Form $links1, rechts1, links2, rechts2$ und so weiter. Bei Mono Sound hingegen wird nur der eine Kanal gespeichert. Daraus ergibt sich auch, dass die Rohdaten einer Stereoaufnahme in etwa doppelt soviel Speicherplatz benötigen wie die Aufnahme im Monoformat mit der selben Abtastrate und

Bittiefe.

7.2. Ausgangssituation/ Bisherige Ergebnisse

geschrieben von Simon Köhler

Der letzte Durchlauf des Projektes fand im Wintersemester 24/25 statt. Zuerst wurden diese Ergebnisse untersucht und das vorhandene Repository geklont und die angefertigte Dokumentation gelesen. Aus dem Bericht ging hervor, dass die Ergebnisse der früheren Durchgänge hier nicht berücksichtigt worden sind und stattdessen ein komplett neuer Ansatz verfolgt worden ist und eigene Modelle verwendet worden sind. Es ist allerdings nicht gelungen dieses neue Modell zu trainieren[ABB+25]. Da die Ergebnisse aus dem WiSe 24/25 also nicht verwendbar waren, hat sich die Arbeitsgruppe aus diesem Semester dazu entschieden auf den Ergebnissen aus dem WiSe 22/23 aufzubauen. Auf Basis der Arbeit des Durchlaufes konnte erfolgreich ein Modell trainiert werden. Dabei bestätigten sich die Ergebnisse dieser Gruppe. Während des Trainings erreicht das Modell Erkennungsraten von über 90%, während die Erkennungsrate über ein Mikrofon mit $5\ \%$ äußerst gering ist [BJB+23]. Im Folgenden werden mögliche Ursachen für diese schlechte Rate untersucht.

7.3. Untersuchung der Ursache für niedrige Erkennungsraten

geschrieben von Jonas Landwehr

Nachfolgend werden mögliche Ursachen für die niedrige Erkennungsrate nach einem erfolgten Training des Modells untersucht. Dafür wird das Skript *EmbedsKi.py* mit dem Argument *–test* aufgerufen. Während einiger Tests wurde außerdem mithilfe des Arguments *–audioFile* eine zuvor aufgenommene Audio-Datei übergeben. Dieser Ansatz wurde gewählt, um stets mit derselben Datengrundlage zu Arbeiten.

7.3.1. Korrektes Audio-Format

geschrieben von Jonas Landwehr

Zunächst wurde die Audiodatei *LEFT - 17.mp3* aus dem Trainingsdatensatz übergeben, um ein korrektes Training zu validieren. Um einen Zufallstreffer ausschließen zu können, wurde der Test fünfmal durchgeführt. Alle fünf Durchläufe gaben ergaben dabei eine Ausgabe des korrekten Kommandos *left* mit einer Sicherheit von 100%.
Nach diesem erfolgreichen Test wurde zunächst die Datei *LEFT - stereo.mp3*, welche im Ordner *testData* des Repository zu finden ist aufgenommen und erneut fünfmal getestet. Diese Tests lieferten die folgenden Ergebnisse:

Tabelle 7.1.: Ergebnisse der Test mit der Datei *LEFT - stereo.mp3*

Durchlauf	**Vorhersage 1**	**Vorhersage 2**
1	*back - 100%*	*stop - 0%*
2	*stop - 100%*	*forward - 0%*
3	*stop - 97%*	*forward - 3%*
4	*stop - 100%*	*none - 0%*
5	*stop - 100%*	*forward - 0%*

Wie in der Tabelle 7.1 zu sehen ist, ist die Erkennungsrate des korrekten Kommandos bei der Datei äußerst schlecht.
Eine genauere Untersuchung zeigt, dass die verwendeten Trainingsdaten nur über eine Tonspur verfügen, also im Mono-Audioformat vorliegen. Für das Durchführen der Inferenz werden allerdings auch andere Formate, wie in diesem Fall Stereo, zugelassen, wodurch es zu den Ungenauigkeiten kam. Um diese Theorie zu beweisen, wurde die zuvor verwendete Testdatei dahingehend modifiziert, dass nur die erste Tonspur gespeichert wurde. Sie wurde also in das Mono-Audioformat transformiert. Die geänderte Version der Datei ist als *LEFT - mono.mp3* ebenfalls im Ordner *testData* abgelegt.
Auch mit dieser modifizierten Datei wurden wieder fünf Testdurchläufe gestartet:

Tabelle 7.2.: Ergebnisse der Test mit der Datei *LEFT - mono.mp3*

Durchlauf	**Vorhersage 1**	**Vorhersage 2**
1	*left - 100%*	*forward - 0%*
2	*left - 100%*	*forward - 0%*
3	*left - 100%*	*forward - 0%*
4	*left - 100%*	*forward - 0%*
5	*left - 100%*	*forward - 0%*

Wie der Tabelle 7.2 entnommen werden kann, hat die Modifikation der Datei auf das Monoformat die zuvor bestehenden Probleme komplett entfernt, sodass das Modell nun in allen Durchläufen das korrekte Kommando mit einer Sicherheit von 100% vorhersagt.
Diese Umwandlung wurde auch im Python-Quellcode angepasst. Sowohl beim Einlesen von Audiodateien, als auch beim Verwenden des Mikrofons, wird nun nur noch der erste Kanal verwendet, wodurch sichergestellt ist, dass alle Audios im Monoformat verarbeitet werden.
Um diese Änderung zu verifizieren wurden erneut Tests mit der bereits bekannten Stereo-Version der Testdatei durchgeführt, dabei wurde das nachfolgende Ergebnis erzielt:

Tabelle 7.3.: Ergebnisse der Test mit der Datei *LEFT - stereo.mp3* und Softwareumwandlung

Durchlauf	**Vorhersage 1**	**Vorhersage 2**
1	*left - 97%*	*forward - 3%*
2	*left - 100%*	*right - 0%*
3	*left - 100%*	*right - 0%*
4	*left - 100%*	*forward - 0%*
5	*left - 100%*	*forward - 0%*

Auch hier wird eine deutliche Verbesserung deutlich. Das korrekte Kommando wird immer erkannt und in 4 von 5 Fällen ist sich das Modell zu 100% sicher. Lediglich der erste Durchlauf liefert

eine Wahrscheinlichkeit, von nur 97% für das Kommando *left*. Da das korrekte Kommando aber trotzdem weit vorne liegt, kann diese Abweichung als vernachlässigbar angesehen werden.

7.3.2. Laute Hintergrundgeräusche

geschrieben von Jonas Landwehr

Während der Tests im Seminarraum und später auf dem Auto sind laute Hintergrundgeräusche, etwa durch den Motor des Autos und durch etwaige Hintergrundgespräche, zu erwarten. Um dies auch mit der Testdatei, welche ohne Hintergrundgeräusche aufgenommen wurde, zu simulieren, wurde ein Ausschnitt der Datei *Car.mp3*, welche unter den Trainingsdaten zu finden ist, unter die bereits zuvor verwendete Testdatei *LEFT - mono.mp3* gelegt. Das Ergebnis wurde in der Datei *LEFT - background.mp3* im Ordner *testData* gespeichert.
Auch für diese Variante wurden fünf Testläufe durchgeführt und in der folgenden Tabelle dokumentiert:

Tabelle 7.4.: Ergebnisse der Test mit der Datei *LEFT - background.mp3*

Durchlauf	Vorhersage 1	Vorhersage 2
1	*left - 91%*	*forward - 8%*
2	*left - 100%*	*right - 0%*
3	*left - 100%*	*forward - 0%*
4	*left - 100%*	*forward - 0%*
5	*back - 73%*	*left - 25%*

Wie in Tabelle 7.4 zu sehen ist, wird das richtige Kommando auch trotz lauter Hintergrundgeräusche in 4 von 5 Fällen korrekt erkannt. Lediglich im fünften Durchgang, wurde mit *back* das falsche Kommando ausgegeben. Aber auch in diesem Fall wird mit einer Wahrscheinlichkeit von 25% das korrekte Kommando *left* erkannt. Insgesamt lässt sich also eine Verschlechterung durch laute Hintergrundgeräusche feststellen, die Erkennungsrate wird also negativ beeinflusst. Allerdings liegt der Erkennungsrate auch trotz der Hintergrundgeräusche noch immer bei 4 von 5. Somit lässt sich die schlechte Erkennungsrate bei der Verwendung eines Mikrofons als Eingabe nur zu kleinen Teilen durch die Hintergrundgeräusche erklären.

7.3.3. Modell auf eine Stimmlage übertrainiert

geschrieben von Jonas Landwehr

Ein möglicher systematischer Fehler könnte darin liegen, dass das Modell aufgrund der zur Verfügung stehenden Trainingsdaten speziell auf eine bestimmte Stimme oder Stimmlage trainiert ist und bei anderen Stimmen schlechtere Ergebnisse liefert.
Aufgrund der Ergebnisse der vorherigen Tests konnte diese Möglichkeit jedoch schnell widerlegt werden, da die Person, die die Testdateien aufgenommen hat, nicht an der Aufnahme der Trainingsdateien aus den vorherigen Jahrgängen beteiligt war. Da das Modell trotzdem, wie etwa in Tabelle 7.3 zu sehen ist, sehr gute Ergebnisse liefert, kann ausgeschlossen werden, dass das Modell nur auf eine Stimme gut reagiert.

7.3.4. Positionierung des Kommandos innerhalb der Aufnahme

geschrieben von Jonas Landwehr

Ein weiterer möglicher systematischer Fehler könnte in der Positionierung des Kommandos innerhalb der Aufnahme liegen, also ob das Kommando für sich steht, oder vorher oder nachher noch etwas folgt. Die Vorgängergruppe aus dem Wintersemester 2022/23 hat zu diesem Zweck bereits eine Vorverarbeitung implementiert, die dazu dienen soll, das Kommando zufällig innerhalb eines vorgegebenen Zeitrahmens zu positionieren[BJB+23]. Die korrekte Funktionalität dieser Vorverarbeitung wurde im Folgenden durch uns getestet.
Für die nachfolgenden Tests wurden drei weitere Audiodateien, mit einer Länge von je 4 Sekunden, angefertigt. Die Länge von 4 Sekunden wurde gewählt, da dies die Standardlänge für die Mikrofonaufnahmen ist. Als Grundlage dient dabei weiterhin die bereits bekannte Datei *LEFT - mono.mp3*. Diese wird je einmal am Anfang der 4 Sekunden, einmal in der Mitte und einmal am Ende platziert, der Rest wurde mit Stille aufgefüllt. Die drei resultierenden Dateien sind ebenfalls in dem Ordner *testData* abgelegt.
Für alle drei Varianten wurden erneut jeweils fünf Durchläufe gestartet, deren Ergebnisse im Folgenden dokumentiert sind.

Tabelle 7.5.: Ergebnisse der Test mit der Datei *LEFT - anfang.mp3*

Durchlauf	**Vorhersage 1**	**Vorhersage 2**
1	*left - 100%*	*forward - 0%*
2	*left - 100%*	*forward - 0%*
3	*left - 100%*	*forward - 0%*
4	*left - 100%*	*forward - 0%*
5	*left - 100%*	*forward - 0%*

Tabelle 7.5 zeigt deutlich, dass die Erkennung des richtigen Kommandos auch dann noch gut funktioniert, wenn nach diesem noch einige Sekunden Stille herrscht. Bei allen fünf Durchläufen wird das korrekte Kommando mit einer Genauigkeit von 100% erkannt.

Tabelle 7.6.: Ergebnisse der Test mit der Datei *LEFT - mitte.mp3*

Durchlauf	**Vorhersage 1**	**Vorhersage 2**
1	*left - 100%*	*right - 0%*
2	*left - 100%*	*right - 0%*
3	*left - 100%*	*right - 0%*
4	*left - 100%*	*right - 0%*
5	*left - 100%*	*right - 0%*

Auch eine Positionierung in der Mitte der Aufnahme hat, wie in Tabelle 7.6 erkenntlich ist, keine Auswirkung auf die Genauigkeit der Erkennung. Diese liegt weiterhin bei 5 korrekt erkannten Kommandos aus 5 Durchläufen.

Tabelle 7.7.: Ergebnisse der Test mit der Datei *LEFT - ende.mp3*

Durchlauf	Vorhersage 1	Vorhersage 2
1	*left - 99%*	*right - 1%*
2	*left - 99%*	*right - 1%*
3	*left - 99%*	*right - 0%*
4	*left - 99%*	*right - 0%*
5	*left - 99%*	*right - 0%*

Auch aus Tabelle 7.7 wird deutlich, dass eine Positionierung des Kommandos ans Ende der Aufzeichnung so gut wie keinen Einfluss auf das Ergebnis hat. Es wurde weiterhin in allen Durchläufen das korrekte Kommando erkannt. Hierbei fällt jedoch auf, dass im Gegensatz zu den vorangegangenen Tests stets eine Wahrscheinlichkeit von 99% ausgegeben wurde. Diese liegt somit minimal unter den vorherigen Ergebnissen. Da die Abweichung allerdings nur ein Prozent beträgt und weiterhin das korrekte Kommando erkannt wird, ist diese an dieser Stelle zu vernachlässigen.
Somit lässt sich festhalten, dass die Position des Kommandos in einem ansonsten stillen Audioblock keinen Einfluss auf die Erkennungsrate des Netzes hat.

7.3.5. Hintergrundrauschen bei langer Aufnahme

geschrieben von Jonas Landwehr

Bis jetzt wurde bei den Tests separat der Einfluss von Hintergrundgeräuschen (siehe Abschnitt 7.3.2) und von der Positionierung beziehungsweise Länge (siehe Abschnitt 7.3.4) betrachtet. Im Folgenden werden die beiden Möglichkeiten nun kombiniert. Dafür wird erneut eine Audiodatei mit einer Länge von 4 Sekunden erstellt. Als Kommando wird etwa mittig in der Tonspur die bereits bekannte Datei *LEFT - mono.mp3* eingefügt. Im Gegensatz zu der Datei aus Abschnitt 7.3.4 wird der Rest der Datei nun jedoch nicht mit Stille ausgefüllt. Stattdessen wurde das leise Rauschen eines Laptop-Lüfters, zu finden unter der Datei *LaptopLüfter.mp3* im Ordner *testData*, als beispielhaftes Hintergrundgeräusch eingefügt. Die resultierende Audiospur wurde in der Datei *LEFT - combined.mp3* gespeichert.
Auch mit dieser Datei wurden fünf Testdurchläufe durchgeführt:

Tabelle 7.8.: Ergebnisse der Test mit der Datei *LEFT - combined.mp3*

Durchlauf	Vorhersage 1	Vorhersage 2
1	*none - 99%*	*forward - 1%*
2	*none - 99%*	*forward - 1%*
3	*none - 99%*	*forward - 1%*
4	*none - 99%*	*forward - 1%*
5	*none - 99%*	*forward - 1%*

Tabelle 7.8 zeigt eindeutig, dass die Erkennungsrate deutlich abgenommen hat. In keinem der fünf Durchläufe liegt das korrekte Kommando in den ersten beiden Plätzen der Vorhersage des Modells. Aufgrund dieses Ergebnisses kann erkannt werden, dass das Hintergrundgeräusch beziehungsweise die fehlende Stille für die schlechten Erkennungsraten verantwortlich sind.

7.3.6. Schlussfolgerung

geschrieben von Jonas Landwehr

Aus den oben dargestellten Untersuchungsergebnissen ergibt sich die Schlussfolgerung, dass eine schlechte Adaption des Netzes an Hintergrundstörgeräusche Grund für die niedrigen Erkennungsraten in der realen Welt sind. Dabei liegt die Vermutung nahe, dass das Modell Anfangs- und Endpunkte des Kommandos innerhalb des Audiobereichs erkennt, indem der nur Bereich betrachtet wird, der nicht mit konstanten Nullen gefüllt ist.
Da sich solche Hintergrundgeräusche in der Realität nicht vermeiden lassen, muss entweder das Training des Modells an diese Gegebenheiten angepasst werden oder die zu verarbeitende Aufnahme so angepasst werden, dass auch hier Nullen zur Markierung des Kommandobereichs verwendet werden.

7.4. Filterung des Audiosignals

geschrieben von Simon Köhler

Wie in den Untersuchungen und in der Schlussfolgerung beschrieben liegt die schlechte Erkennungsrate des Netzes an den Hintergrundgeräuschen die zum Beispiel während der Live Inferenz auftreten. Um dieses Problem zu beheben wurde eine Funktionalität entwickelt, die das eingesprochene Signal vor der Übergabe in das Netz filtert. Die Filterung wurde in der *filter_voice_command* Funktion in der `inference/__init__.py` Datei implementiert. Die Filterung funktioniert folgendermaßen. Das Array, welches die digitalisierten Audiosignale des Sprachkommandos enthält wird durchsucht und die Daten anhand der höchsten Amplitude auf einen Wert zwischen Null und Eins normalisiert. Daraufhin wird anhand eines Schrankenwertes, der in Prozent angegeben ist, entschieden, welche Werte behalten werden und welche auf null gesetzt werden. In dem aktuellen Stand liegt die Schranke bei $5\ \%$. So werden alle Werte auf null gesetzt, die Leiser sind als $5\ \%$ der höchsten Amplitude.
Durch diese Implementierung ist es gelungen, die Erkennungsrate des Netzes während der Live Inferenz deutlich zu steigern, da das Hintergrundrauschen gefiltert wird. Im Folgenden ist ein Testdurchlauf dargestellt, der die Ergebnisse enthält. Dafür wurde der Befehl *left* fünf Mal eingesprochen.

Tabelle 7.9.: Ergebnisse der Live Inferenz mit Filterung

Durchlauf	**Vorhersage 1**	**Vorhersage 2**
1	*left - 80%*	*right - 10%*
2	*left - 80%*	*forward - 19%*
3	*left - 99%*	*forward - 0%*
4	*left - 97%*	*forward - 3%*
5	*left - 98%*	*forward - 2%*

Wie in Tabelle 7.9 zu erkennen, wird das Kommando in diesem Testdurchlauf bei jedem Mal korrekt klassifiziert. Auch die Sicherheit des Netzes liegt hier für das Kommando *left* jeweils bei mindestens $80\ \%$ und im Mittel bei $90,8\ \%$. Es wurden ebenfalls jeweils 5 Durchläufe für die anderen Sprachbefehle getestet. Für den Befehl *right* wurden fünf von fünf Durchläufe korrekt klassifiziert. Die Erkennungsrate lag hier im Mittel bei $100\ \%$ für den richtigen Befehl. Für den Befehl *stop* wurden ebenfalls jedes Mal korrekt klassifiziert und auch eine perfekte Sicherheit von

$100\ \%$ erzielt. Für den Befehl *back* wurde ebenfalls jedes Mal korrekt klassifiziert, die Sicherheit lag hier im Mittel bei $83,4\ \%$. Für den Befehl *forward* konnte zunächst leider keine zufriedenstellende Erkennungsrate erzielt werden. Nach einer Analyse fiel auf, dass der Befehl sehr häufig mit dem *stop* Befehl verwechselt wird. Nach weiteren Versuchen ließ sich feststellen, dass je langsamer und deutlicher der *forward* Befehl ausgesprochen wird die Erkennungsrate und Sicherheit des Netzes wieder steigt. Bei langsamer und klarer Aussprache konnte für einen erweiterten Testdurchlauf aus 10 eingesprochenen *forward* Befehlen 10 von 10 korrekt klassifiziert werden. Die Sicherheit lag im Mittel bei $95\ \%$. Die Erkennung für die Sprachbefehle konnte also im Vergleich zu den Ergebnissen aus [BJB+23] deutlich verbessert werden, wo die Erkennungsrate der Live Inferenz über ein Mikrophon bei nur $5\ \%$ lag.

Die vorgestellte Lösung unterliegt jedoch auch einigen Einschränkungen. Da das lauteste Signal in der Aufnahme als Referenzpunkt dient, muss das lauteste Signal in der Aufnahme auch der jeweilige Befehl sein. Bei sehr lauten Hintergrund- oder Nebengeräuschen würden sonst auch Teile des Befehls entfernt werden. Des Weiteren muss der Sprachbefehl in möglichst einer konstanten Lautstärke gesprochen werden. Bei hohen Schwankungen in den Amplituden des Signals kann es dazu kommen, dass Teile des Befehls abgeschnitten und somit nicht mehr erkannt werden.

7.5. Ausblick

geschrieben von Jonas Landwehr

Im Rahmen dieses Projektes ist es gelungen eine erste Lösung zu entwickeln, mit der die Erkennungsrate einer Aufnahme am Laptop erheblich erhöht werden konnte.
Weiterhin wurden viele Tests durchgeführt, welche starke Indizien für die möglichen Fehlerquellen lieferten. Aufbauend auf diesen Ergebnissen wurden im Folgenden mögliche zusätzliche Lösungsansätze sowie weitere Schritte zusammengefasst, welche von nachfolgenden Gruppen implementiert werden könnten.

7.5.1. Anpassung des Paddings

geschrieben von Jonas Landwehr

Bis jetzt werden die Trainingsdateien an beiden Seiten mit Nullen, also mit Stille, erweitert, sodass die Ziellänge für das Modell erreicht wird. Das hat zur Folge, dass das Modell diese konstanten Werte ebenfalls anlernt und zur Differenzierung zwischen Kommando und kein Kommando verwendet.
Zukünftige Gruppen sollten daher die Paddingmethode in der Datei *DataPreprocessing.py* aus dem Package *neuralNetwork* anpassen. Es sollte statt das klassischen Paddings mit konstanten Werte lieber Ausschnitte aus Hintergrundgeräuschen verwendet werden, welche aber auch randomisiert verwendet werden sollten, um ein Anlernen des Modells zu verhindern. Als Hintergrundgeräusche kommen dabei etwa die bereits verfügbaren Dateien *Car.mp3* aus den Trainingsdaten oder *LaptopLüfter.mp3* aus den Testdaten in Frage.

7.5.2. Implementierung eines Sliding Windows

geschrieben von Jonas Landwehr

In der aktuellen Version ist die Aufnahme über das Mikrofon so implementiert, dass immer für vier Sekunden aufgenommen wird. Diese Aufnahme wird dann verarbeitet und ein Ergebnis ausgegeben. Nach der Ausgabe wird die nächste Aufnahme gestartet und der nächste Durchlauf beginnt. Dadurch entsteht das Problem, dass ein Kommando theoretisch genau auf der Grenze von zwei Aufnahmen gegeben wird und somit in beiden halb enthalten ist, allerdings von keiner richtig erkannt wird.
Um dieses Problem zu beheben, sollte eine Art Sliding Window Mechanismus eingebaut werden. Die Idee dabei ist, dass immer zwei Aufnahmen um eine halbe Periode versetzt zueinander gestartet und verarbeitet werden. Auf diese Weise wird sichergestellt, dass gegebene Kommandos immer auf mindestens einer der Aufnahmen vollständig vorhanden sind. Außerdem wird dadurch auch die Reaktionsfähigkeit des Systems erhöht, da bereits nach einer halben Periode, also standardmäßig nach 2 Sekunden statt den 4 Sekunden einer vollständigen Periode, eine Auswertung stattfindet.

7.5.3. Implementierung einer Spracherkennung

geschrieben von Jonas Landwehr

Eine weitere Möglichkeit wäre die Implementierung einer Spracherkennung zum Vorverarbeiten des eingegebenen Audios. Mithilfe einer solchen Erkennung könnte etwa der Teil in dem gesprochen wird, der also das Kommando enthält, isoliert werden. Dieses isolierte Kommando würde anschließend die normale Vorverarbeitung mit Padding durchlaufen. Wie in Unterabschnitt 7.3.4 gezeigt wurde, wird durch das korrekte Padding eine sehr viel höhere Erkennungsrate erreicht, sodass mit dieser Methode eine sehr gute Erkennungsrate zu erwarten ist.
Weiterhin wäre denkbar eine Sprachaktivierung zu implementieren, sodass die Aufnahme immer nur bei Bedarf verarbeitet wird. Eine solche Implementierung würde wie bei der Spracherkennung zu einer sehr guten Erkennungsrate führen. Darüber hinaus würde eine Sprachaktivierung dafür sorgen, dass die Reaktionsdauer des Modells deutlich reduziert werden würde, da die Aufnahme gestoppt wird, sobald keine Stimme mehr erkannt wird. Die Verarbeitung der Aufnahme könnte also direkt mit dem Ende des Kommandos beginnen.

8. Elektronik und Mechanik

geschrieben von Max Tepe, Frederic Goretzky, Milan Becker und Simon Koger

In dieser Iteration des Gesamtprojektes bestand die Notwendigkeit, die Hardware des Basisfahrzeugs anzupassen. Diese Notwendigkeit ergibt sich aus dem Umstand, dass das Upgrade vom PYNQ-Z1 Board zum neueren PYNQ-Z2 Board mit veränderten Abmessungen [Dig18a; TUL20] und veränderten Positionen von Input- und Output-Schnittstellen daherkommt [Dig18b; TUL18]. Ebenso musste ein Richtmikrofon als neues Bauteil in das Fahrzeug integriert werden, da das neue Board kein integriertes Mikrofon mehr besitzt und die Audio-Aufnahmequalität so deutlich verbessert wird.
Die personelle Stärke dieser Iteration des Gesamtprojektes hat es dem Entwicklungsteam außerdem erlaubt, diverse Anpassungen und Verbesserungen der Mechanik und Elektronik des Fahrzeugs vorzunehmen. Zur Planung der Integration wurde außerdem ein 3D-Modell des Fahrzeugs erstellt.

Um während der Entwicklung jederzeit ein fahrbereites und funktionierendes Basisfahrzeug zu haben, wurden sämtliche Anpassungen und Erweiterungen nur an einem der zwei verfügbaren Fahrzeuge durchgeführt. Damit in folgenden Iterationen dieses Projektes nicht immer mehrere Projektberichte bezüglich der Hardware durchsucht werden müssen, folgt an dieser Stelle zuerst eine Beschreibung des alten Fahrzeugs als Ausgangsbasis für die durchgeführten Verbesserungen. Danach folgt die genaue Beschreibung der Modifikationen für das aktuelle Fahrzeug. Am Ende dieses Kapitels findet sich dann eine detaillierte Liste aller Bauteile mit Bezugsquelle sowie die vorzunehmenden Einstellungen für den 3D-Druck für die selbst gedruckten Bauteile.
Dieses Kapitel kann daher auch als Anleitung verstanden werden kann, das zweite Fahrzeug in einen baugleichen Zustand zu bringen. Außerdem werden getroffene Designentscheidungen begründet, sodass die neue Fahrzeugbasis längerfristig Bestand hat und erst in einigen Iterationen angepasst werden muss.

8.1. Ausgangssituation

Nach aktuellem Stand ist das PYNQ-Z1 Board mit einem Klettkabelbinder an der Acrylplatte des Fahrzeugs befestigt. In die Stativaufnahme der Kamera ist eine Schraube geschraubt, welche über eine Eigenkonstruktion mit der oberen Acrylplatte des Fahrzeugs verbunden ist. Das Akkupaket zur Spannungsversorgung der Motoren befindet sich, an der im hinteren Teil des Fahrzeugs befindlichen, dafür vorgesehenen Halterung auf der Acrylplatte. Die Powerbank zur Spannungsversorgung für das PYNQ-Z1 Board befindet sich unterhalb der oberen Acrylplatte und ist ebenfalls mit Klettkabelbindern, sowie einem Klebepad befestigt. Die restlichen Teile des Fahrzeugs (Motor und Treiber) sind entsprechend dem Bausatz von Elegoo an ihren dafür vorgesehen Orten verbaut.
In diesem Teil liegt der Fokus auf dem Ersetzen des PYNQ-Z1 durch das PYNQ-Z2 Board, dem Optimieren der Kameraaufnahme, sowie dem Hinzufügen eines externen Mikrofons. Das

Folgekapitel beschreibt die durchgeführten Schritte und dahinterliegenden Beweggründe für das Design.

8.2. Modifikation

Wie bereits in der Einleitung erwähnt, wurde im Rahmen dieser Iteration der mechanische Aufbau eines der beiden Fahrzeuge überarbeitet. Der Umbau war nötig, da das neue PYNQ-Z2 Board länger ist als das zuvor verwendete PYNQ-Z1 Board. Zusätzlich war die bestehende Kamerahalterung instabil, und eine Montagemöglichkeit für ein externes Mikrofon war nicht vorhanden. Des Weiteren bietet das neue Z2 Board Schraublöcher zur sicheren Befestigung.

Zur Umsetzung wurde eine neue Adapterplatte entworfen, die auf der vorhandenen Acrylplatte des Fahrzeugbausatzes aufliegt. Hinzu kommt ein Gehäuseoberteil, welches das Z2 Board zum einen schützt und zum anderen die Montage von Kamera und Mikrofon ermöglicht. Beide Teile wurden durch 3D-Druck realisiert.

Für die Planung wurde ein vollständiges 3D-Modell des Fahrzeugs in Fusion 360 erstellt. Die dafür verwendeten Komponenten basierten auf öffentlich verfügbaren STEP-Dateien für PYNQ-Z2 Board [McC22], den Fahrzeugbausatz von Elegoo in Version 3.0 [Pet19], als Platzhalter für die JellyComb Webcam, die Logitech C920[Pet17] und als Einpressgewinde M3 Gewindeeinsätze von ruthex [rut20]. Nicht passend war das Akkupack, weshalb ein 3D-Modell davon erstellt wurde. Das fertige Modell lässt sich unter [Gor+25] finden.

8.2.1. Adapterplatte

Die Adapterplatte soll auf die vorhandene Acrylplatte gesetzt werden, da auf der Unterseite bereits die Powerbank befestigt ist, sie bietet gezielt angepasste Aufnahmepunkte für das PYNQ-Z2 Board sowie das Akkupack. Die bisherigen Befestigungslöcher und Kabeldurchführungen wurden vollständig übernommen, um Kompatibilität mit dem bestehenden Unterbau sicherzustellen.

Das Board wurde im vorderen linken Teil der Adapterplatte positioniert. Da sich auf der linken Seite nur der SD-Karteneinschub befindet, kann die Platine nahezu bis zum Rand verschoben werden. Die rechte Seite bleibt frei zugänglich für USB-Anschluss, 3,5 mm-Klinkenbuchse sowie im hinteren Teil die PMOD-Steckplätze.

Das 12 V-Akkupack wurde um $90°$ gedreht und auf der Seite stehend montiert. Die Halterung dafür wurde speziell konstruiert und bietet Löcher zur Aufnahme von M3-Einpressgewinden zur Verschraubung mit dem Akkupack. Eine Aussparung in der Batteriehalterung aufseiten des Boards ermöglicht den Zugang zu den PMOD-Steckplätzen.

Zur sicheren Befestigung des Boards wurden M3-Einpressgewinde verwendet. Diese ermöglichen eine wiederholbare und stabile Verschraubung, ohne dass das Material ermüdet, wie es bei 3D gedruckten Gewinden der Fall wäre. Die Adapterplatte wird mit den bereits vorhandenen Schrauben auf die obere Acrylplatte des Fahrzeugs montiert.

Auf der linken Seite der Schwalbenschwanzverbindung wurde eine Aussparung integriert, um den Zugriff auf die SD-Karte auch im montierten Zustand zu ermöglichen. Zusätzlich befinden sich im Bodenbereich Durchführungen zur Führung von Leitungen. Das fertige Modell lässt sich unter [Gor+25] finden.

8.2.2. Gehäuseoberteil

Das Gehäuseoberteil wird von vorne aufgeschoben und über eine zweigeteilte Schwalbenschwanzverbindung mit der Adapterplatte verbunden. Diese Art der Verbindung erlaubt ein einfaches Aufschieben des Gehäuses. Das Gehäuseoberteil kann durch die Durchlässe in den Wänden mithilfe von Kabelbindern fest mit der Adapterplatte befestigt werden. Da es sich hier um einen Prototyp handelt und das häufige Entfernen des Gehäuseoberteils zu erwarten ist, werden die Durchführungen verwendet, um die Komponente mit Klettkabelbindern über das Akkupack zu befestigen.

Auf der linken Seite wurde eine Aussparung integriert, um den Zugriff auf die SD-Karte auch im montierten Zustand des Gehäuseoberteils zu ermöglichen.

Da die Höhe des Gehäuseoberteils vorerst durch den L-USB-Stecker limitiert wurde, kann die Höhe des Bauteils über einen Parameter in Fusion 360 gewählt werden. Im fertigen Zustand besitzt das Gehäuseoberteil genau die Höhe, um auf dem Akkupack aufzuliegen und dadurch zusätzliche Stabilität zu gewinnen. Das fertige Modell lässt sich unter [Gor+25] finden.

8.2.3. Kameraaufnahme

Die bestehende Kamerahalterung hatte mehrere Mängel: Sie verdeckte den Ethernet-Port des PYNQ-Boards und war insgesamt instabil. Die Montage erfolgte mit einer M6-Schraube, deren metrisches Gewinde jedoch nicht zum 1/4-Zoll-Innengewinde der Kamera passt.

Nach DIN 13 besitzt eine M6-Schraube eine Gewindesteigung von 1 mm [wwwa], während das 1/4-Zoll-Gewinde gemäß DIN 4503 eine Steigung von 20 TPI (ca. 1,27 mm) aufweist [wwwb]. Diese Unterschiede führen zu inkompatiblen Gewinden.

Die neue Kameraaufnahme wird durch Bohrungen in dem Gehäuseoberteil realisiert, über die die Kamera mit dem Gehäuse verschraubt werden kann. Dabei ist die Kamera frontal ausgerichtet und mithilfe der Gelenke der Kamera über Kopf am Gehäuseoberteil angebracht, damit das dahinter montierte Mikrofon nicht blockiert wird. Da keine 1/4“-Schrauben verfügbar waren, wurden passende Schrauben im 3D-Druckverfahren selbst gefertigt. Zur optimalen Belastbarkeit der Schrauben wurden diese in drei Teilen entworfen. Linke und rechte Hälfte der Schraube, sowie einem Verbindungsstück. Zur Montage wurden die drei Teile verklebt. Der Vorteil an dieser Fertigung ist, dass die schwächsten Verbindungen zwischen den Layerlines entgegengesetzt zur Kraft ausgerichtet sind.

8.2.4. Mikrofonhalterung

Das neu hinzugefügte externe Mikrofon wurde ebenfalls auf dem Gehäuse montiert. Die Halterung wurde so konzipiert, dass das Mikrofon über die mitgelieferte Stativaufnahme sicher fixiert werden kann. Zur Montage werden ebenfalls die selbst gefertigten 1/4“-Schrauben verwendet.

8.2.5. Fertigung und Materialwahl

Die gesamte Konstruktion wurde im FDM-3D-Druckverfahren gefertigt. Aufgrund der Verfügbarkeit wurden die Adapterplatte und die 1/4“-Schrauben mit PETG gefertigt und das Gehäuseoberteil mit PLA. Um eine langlebige und wiederverwendbare Montage zu ermöglichen, wurden an kritischen Verbindungsstellen M3-Einpressgewinde verwendet.

Die Druckteile Gehäuseoberteil und Adapterplatte wurden mit 0,2 mm Schichthöhe und 25 % Infill gedruckt, um eine gute Kombination aus Stabilität und Druckzeit zu erreichen. Für optimale Details und beste Festigkeit wurden die Schrauben liegend, mit adaptiver Schichthöhe gedruckt. Stützstrukturen sind nicht notwendig, da die Teile ganzheitlich maximale Überhänge von $45°$ besitzen. Die Adapterplatte wurde mit einem Prusa XL und das Gehäuseoberteil sowie die 1/4“-Schrauben auf einem BambuLab P1S gefertigt. Insgesamt wird eine Druckfläche von mindestens 240 $\times$ 150 mm benötigt und circa 350 g Filament.

8.2.6. Weitere Verbesserungen der bestehenden Hardware

Im bestehenden Fahrzeug sind sowohl die USB-Kamera als auch der WLAN-USB-Adapter über den einzigen USB-A Port des alten PYNQ-Z1 Boards angeschlossen. Dieser Port befindet sich auf der rechten (Draufsicht von oben) Seite des Boards [TUL20]. Um nicht zu weit über den Rand des Fahrzeugs hinaus zu stehen, wurde ein USB-L-Adapter genutzt, der den Port nach oben richtet. An diesen L-Adapter wurde dann ein USB-Hub in Form eines ungefähr 30 Zentimeter langen Kabels angeschlossen. In den einen Port dieses Hubs wurde der WLAN-USB-Adapter angeschlossen, an den anderen Port die USB-Kamera.
Dieser Aufbau ist im neuen Fahrzeug deutlich verbessert worden, insbesondere die Länge der Kabel wurde reduziert, sodass auf und an dem Fahrzeug nicht mehr unnötig viele Kabelmeter untergebracht werden müssen.

Im ersten Schritt wurden die Anschlüsse des neuen PYNQ-Z2 Boards mit denen des alten Boards verglichen. Dabei lässt sich feststellen, dass im Bereich USB und Ethernet keine Änderungen des Herstellers vorgenommen wurden [Dig18b; TUL18]. Es gibt weiterhin einen USB 2.0 Port, welcher nur den Host-Modus unterstützt (dies bedeutet, dass das Board selbst nur als Host fungieren kann). Außerdem existiert ein Micro-USB-Port welcher die Funktionen der UART-Bridge und des JTAG Programming Circuitry vereint. Als Netzwerkschnittstelle steht ein Ethernet RJ45 Port zur Verfügung.

Anschließend wurde evaluiert, ob es möglich ist, den USB-Hub als Komponente zu entfernen, indem entweder die Kamera oder der WLAN-Adapter an eine andere Schnittstelle als den USB 2.0 Port angeschlossen wird. Dies ist aus folgenden Gründen leider nicht möglich:

- Der Micro-USB Port unterstützt nur JTAG und UART und dient in der aktuellen Version auch zur Stromversorgung des Boards. An diesen Port kann also weder die Kamera noch der WLAN-Adapter angeschlossen werden, da dieser Port nur zur seriellen Kommunikation (UART) und Programmierung (JTAG) dient und keine USB-Host-Funktionalität besitzt.
- Es ist außerdem nicht sinnvoll möglich, den WLAN-Adapter an den freien Ethernet-Port anzuschließen oder einen neuen Adapter zu kaufen, welcher dort angeschlossen werden kann. Es gibt zwar Adapter, die diese Funktionalität bieten, aber diese benötigen zwangsläufig immer eine externe Stromversorgung, da der Ethernet-Port im Gegensatz zum USB 2.0 Port keine Stromversorgung bereitstellen kann. Diese Stromversorgung bereitzustellen, wäre deutlich mehr Aufwand als den vorhandenen USB-Hub zu verwenden.

Trotzdem sollte dieser Aspekt des Fahrzeugs optimiert werden und es wurde daher ein neuer USB-Hub mit minimalen Abmessungen gekauft, welcher auch nicht mehr in Form eines Kabels ist, sondern ein Gehäuse der Abmessungen 40 mm $\times$ 30 mm $\times$ 10 mm hat. Um unter den neu designten Deckel des Fahrzeugs zu passen, wurde der USB-L-Adapter durch einen neuen USB-U-Adapter ersetzt, welcher den USB 2.0 Port in einer $180°$ Biegung zurück über das Board legt. An diesen U-Adapter wurde dann der neue USB-Hub eingesteckt. In den einen Port des Adapters

wurde dann der bereits existierende WLAN-Adapter eingesteckt, in den anderen Port das Kabel der USB-Kamera.
Um dann noch die unnötig lange Kabelverbindung der USB-Kamera zu optimieren, wurde das USB-Kabel der Kamera von 150 cm auf 60 cm eingekürzt. Wie genau man ein USB Kabel kürzt, ist in vielen Videos und bebilderten Anleitungen im Internet zu finden, an dieser Stelle werden nur kurz die Schritte aufgelistet, um zukünftigen Projektteilnehmern einen Einstieg zur Recherche zu geben, falls nicht bereits bekannt ist, wie man diese Modifikation durchführen kann.

1. Kabel zweimal an den entsprechenden Stellen mit einem scharfen Seitenschneider durchtrennen.
2. Äußere Isolierung des Kabels an beiden Seiten 2 cm entfernen. Die vier farblichen dünnen Leitungen (zwei Datenleitungen und zwei Stromversorgungsleitungen) auf beiden Seiten 1 cm abisolieren.
3. Jetzt einen Schrumpfschlauch vor dem Verlöten der Leitungen aufschieben!
4. Die freien Leitungen farblich korrekt mit einem Lötkolben und Lötzinn fest verbinden. Diese Verbindungen dann mit Isolierband isolieren, es darf keine leitende Verbindung zwischen den Leitungen entstehen.
5. Die Abschirmung zurück über die Leitungen biegen und auch verlöten. Sollte diese zerstört worden sein, kann haushaltsübliche Aluminiumfolie als Ersatz dienen. Ziel ist es, einen Schutz gegen elektromagnetische Störungen der Datenübertragung zu erzeugen.
6. Den Schrumpfschlauch über die neue Verbindungsstelle schieben und mit Hitze schrumpfen lassen

Diese Modifikationen ändern nichts an der Funktionalität des Fahrzeugs, sie verbessern lediglich die Handhabung durch Reduzierung der Kabelmenge.

8.3. Materialliste des Gesamtfahrzeugs

Im Folgenden sind sämtliche im Fahrzeug verbaute Komponenten übersichtlich in tabellarischer Form aufgeführt. Die Tabelle dient sowohl der Dokumentation des ursprünglichen Zustands als auch der Nachverfolgbarkeit aller vorgenommenen Änderungen im Rahmen der Umgestaltung.

Zur besseren Unterscheidung der Komponenten wurden diese mit einem Kürzel versehen, das den jeweiligen Status im Vergleich zum ursprünglichen Design beschreibt:

- 1: Die Komponente war bereits im ursprünglichen Design enthalten und wird weiterhin verwendet.
- 2: Die Komponente war im ursprünglichen Design enthalten, entfällt jedoch im neuen Aufbau.
- 3: Die Komponente wurde im neuen Design zusätzlich eingeführt.

Durch diese Kategorisierung wird deutlich, welche Bestandteile aus dem ursprünglichen Konzept übernommen wurden, welche entfernt und welche im Rahmen des neuen Designs ergänzt wurden. Die Einteilung in „Altes Design“, „Neues Design“ und „Gedruckte Teile“ ermöglicht zudem eine klare Strukturierung nach Baugruppen und Herkunft der jeweiligen Komponenten.

Tabelle 8.1.: Hardware-Komponenten

Komponente	**Verweis**	**Kürzel**
Altes Design		
Elegoo Smart Robot Car Kit v3.0	[ELE20]	1
Charmast Powerbank 10400 mAh, 5 V, 3 A	[Cha25]	1
Jelly Comb 1080P HD Webcam	[Jel24]	1
L-Adapter	[iJi24]	2
Abstandshalter m3 × m3	[Gmb22]	3
Xilinx PINQ-Z1	[Dig18a], [Dig18b]	2
Neues Design		
Hama Richtmikrofon „Rmn Uni“	[KG]	3
Einpressgewinde m3 × 4 × 5	[rut20]	3
U-Adapter	[LAN24]	3
USB-Hub	[na]	3
Schrauben m3 × 6	-	3
Xilinx PINQ-Z2	[TUL20], [TUL18]	3
Gedruckte Teile		
1/4“-Schrauben	[Gor+25]	3
Deckel	[Gor+25]	3
Adapterplatte	[Gor+25]	3

9. Auswirkungen von Rechenfehlern im neuronalen Netz abschätzen

9.1. Einleitung

geschrieben von Arian Dannemann

Künstliche Intelligenz in Form von neuronalen Netzen spielt eine immer wichtigere Rolle im alltäglichen Leben. Ob Gesichtserkennung im Smartphone, Erkennung von Sprache und Video, medizinischen Diagnosen oder sogar dem autonomen Fahren von Vehikeln, NNs begleiten den Durchschnittsbürger im Alltag auf zahlreiche Weisen. Darum ist es von zunehmender Bedeutung, die Zuverlässigkeit intelligenter Systeme in Bezug auf ihre Fehleranfälligkeit besser zu verstehen.

Dieses Kapitel gibt eine Übersicht über Herausforderungen bei sicherheitsrelevanten Systemen. Es werden weiter relevante Faktoren für Fehler innerhalb eines NN aufgezeigt und Evaluationsmethoden zur Fehlertoleranz von NNs vorgestellt. Anschließend werden Auswirkungen von Rechenfehlern beschrieben und schließlich Maßnahmen zur Abhärtung von NNs erläutert.

9.1.1. Neuronale Netze

Die Grundlagen neuronaler Netze sind in dem Abschnitt 7.1.1 ausführlich dargestellt.

9.1.2. Beschleuniger

geschrieben von Arian Dannemann

Unter einem Accelerator (deutsch: Beschleuniger) wird jede Hardware verstanden, welche die Aufgabe annimmt NNs schneller auszuführen als eine reine Software-Lösung auf einer CPU es könnte. Dazu gehören beispielsweise GPUs, Application-Specific Integrated Circuits (ASICs), FPGAs und mehr. Diese Technologien sind oft vielseitig einsetzbar, beispielsweise wird eine GPU in der Regel für grafikintensive Anwendungen benutzt, bietet allerdings ebenfalls die Möglichkeit, ein NN von einem Prozessor auf speziellere Hardware auszulagern.

Oft wird der Begriff Accelerator mit bestimmten neuronalen Netzen zusammen verwendet, also Deep Neural Network (DNN) Accelerator oder Quantized Neural Network (QNN) Accelerator. Ein DNN Beschleuniger besteht grundlegend aus einer beliebigen Anzahl von Processing Engines (PEs), einem globalen Puffer und einer Verbindung zum Dynamic Random Access Memory (DRAM) um Daten von und zur CPU zu transferieren. So ein Aufbau kann in Abbildung 9.1 gesehen werden.

Systolic Arrays

geschrieben von Kevin Lingk

Eine besonders effiziente Architektur für DNN-Beschleuniger sind Systolic Arrays. Bei den Systolic Arrays handelt es sich um eine strukturierte Anordnung von PEs, die Daten in einem regelmäßigen Takt zwischen den PEs weitergibt. Aufgrund der lokalen Verbindung arbeiten die PEs synchron und eignen sich besonders gut für matrixbasierte Operationen, wie sie typischerweise in NNs auftreten. Eine konkrete Systolic Array Architektur wird von Opriţoiu et al. [OV23] beschrieben.

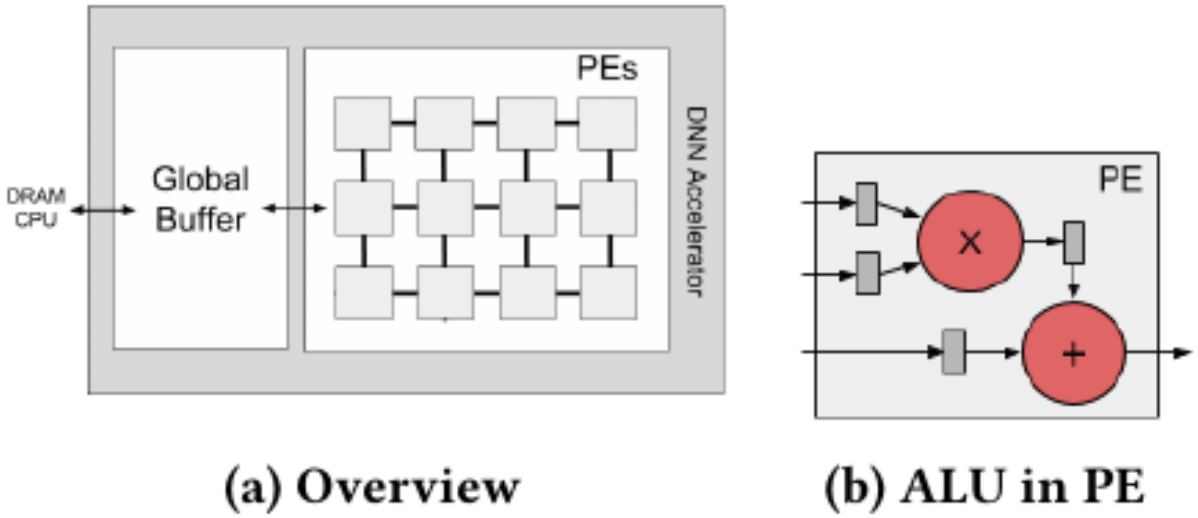


Abbildung 9.1.: Beispielhafter Aufbau eines üblichen DNN Beschleunigers. In (b) ist weiterhin der genauere Aufbau der ALU innerhalb eines PE zu sehen (out-of-scope für diesen Bericht). [Li+17]

9.1.3. Relevante Fehlerarten

geschrieben von Tuanna Karadas

In diesem Unterkapitel werden die relevanten Fehlertypen in NNs strukturiert, detailliert erläutert und hinsichtlich ihrer Auswirkungen beschrieben. Die folgenden Absätze basieren auf dem Paper von Torres-Huitzil und Girau [TG17], die eine umfassende Analyse zur Fehlertoleranz und Fehlerklassifikation in neuronalen Netzen vorgenommen haben.

In der Fehlertoleranzforschung für neuronale Netze werden Fehler (faults), Fehlwirkungen (errors) und Ausfälle (failures) als fundamentale Konzepte unterschieden, die eine Kausalkette bilden (Abbildung 9.2). Ein Fehler bezeichnet eine physische Anomalie im Systemsubstrat (z. B. Transistordefekt, Leiterbahnbruch), die eine Fehlwirkung bzw. eine logische Abweichung vom erwarteten Zustand (z B. falscher Synapsenwert) verursacht. Unbehandelte Fehlwirkungen propagieren zum Ausfall, bei dem das System seine spezifizierte Funktion nicht mehr erfüllt. Fehler werden nach ihrem zeitlichen Verhalten klassifiziert ([TG17, Kapitel „Fault Types“, Abschnitt 1-2].

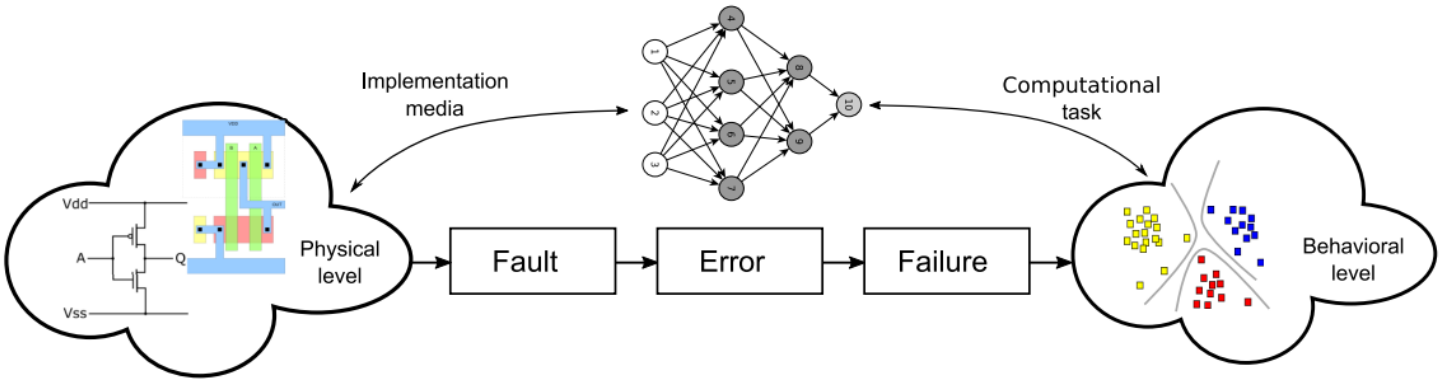


Abbildung 9.2.: Kausalkette zwischen Fault, Error und Failure [TG17]

Die in Abbildung 9.3 dargestellte Klassifikation von Fehlertypen in neuronalen Netzen und digitalen Systemen basiert auf zwei Dimensionen. Dabei werden typische Ursachen und Mechanismen aufgeführt, die zur Entstehung dieser Fehler führen. Diese werden durch die entsprechenden Modelle für permanente bzw. transiente Fehler beschrieben, die in Abbildung 9.2 als graue, abgerundete Kästchen dargestellt sind. [TG17]

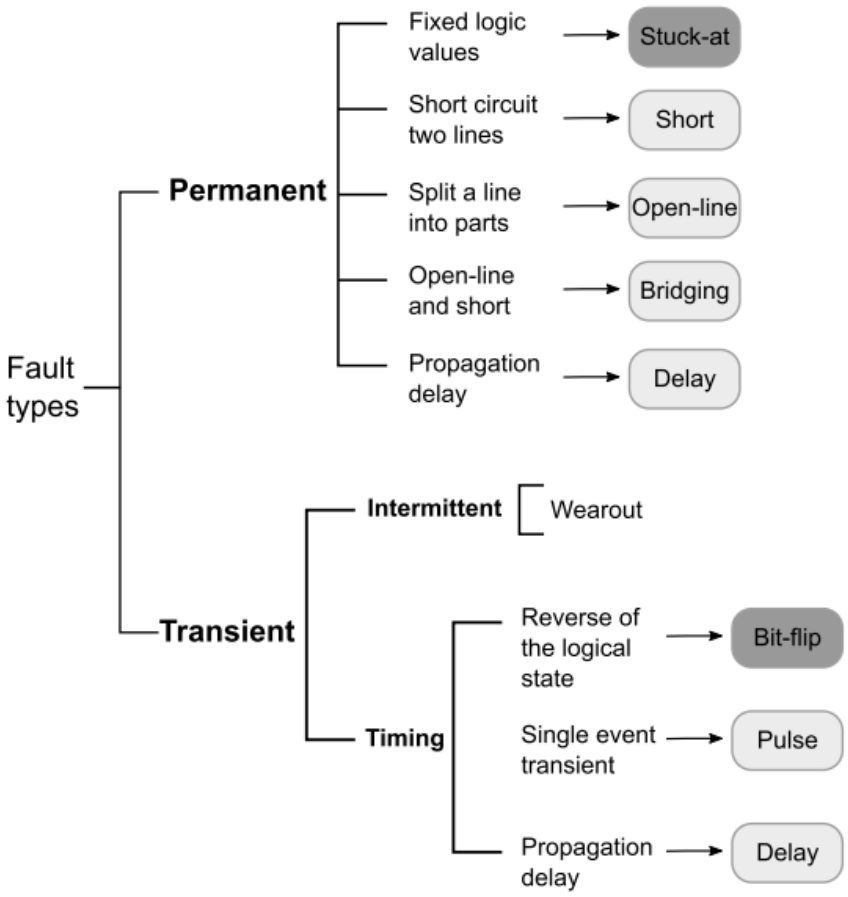


Abbildung 9.3.: Fehlertypen [TG17]

Permanente Fehler: Permanente Fehler (engl. Permanent Faults) sind durch ihre Stabilität und Irreversibilität gekennzeichnet. Sie entstehen typischerweise durch physische Defekte im Hardware-Substrat. Ein häufiges Beispiel sind sogenannte Stuck-at-Fehler, bei denen ein Signal dauerhaft auf einem festen Logikzustand (0 oder 1) verharrt.

Weitere Erscheinungsformen sind Kurzschlüsse zwischen Leitungen (Bridging), etwa infolge lithografischer Fertigungsfehler, oder Leitungsunterbrechungen (Open-line), die zu hochohmigen und instabilen Zuständen führen, insbesondere kritisch bei CMOS-Eingängen. Auch Propagation Delays, also zeitliche Verzögerungen in der Signalweitergabe durch defekte Transistoren oder parasitäre Effekte, zählen zu dieser Kategorie. Die häufigsten Fehlermodelle zur Beschreibung dieser Phänomene sind Stuck-at, Open-line, Bridging und Delay, wie in Abbildung 9.2 grau hervorgehoben. [TG17]

Transiente Fehler: Im Gegensatz dazu sind transiente Fehler (engl. Transient Faults) flüchtiger Natur und treten häufig unter dem Einfluss externer Störungen auf, etwa durch ionisierende Strahlung oder Spannungsschwankungen. Zu dieser Klasse zählen unter anderem intermittierende Fehler, die durch alterungsbedingte Instabilitäten, wie etwa verschleißende Transistoren, entstehen können. Auch Single Event Transients, also kurzzeitige Signalverfälschungen infolge einzelner Teilchenereignisse, sowie temporäre Timing-Störungen durch Rauschen oder Variabilität im Schaltungsverhalten gehören hierzu. Besonders kritisch ist die sogenannte Logikumkehr (Bit-flip), bei der einzelne Speicherzellen spontan ihren Zustand ändern. Das ist ein Phänomen, das beispielsweise durch kosmische Strahlung ausgelöst werden kann. Entsprechende Fehlermodelle sind Bit-flip, Pulse und Delay, die ebenfalls in Abbildung 9.2 grau hinterlegt dargestellt sind.[TG17]

Im Folgenden werden die einzelnen Fehlerklassen näher betrachtet, die den zuvor beschriebenen permanenten und transienten Fehlertypen zugrunde liegen. Dazu zählen insbesondere Hard Errors und Soft Errors sowie deren spezifische Ausprägungen wie Single Event Upset (SEU), Single Bit Upset (SBU) und Multi Bit Upset (MBU), die jeweils charakteristische physikalische Ursachen und Auswirkungen auf das Systemverhalten aufweisen.

Hard Errors

Dieser Abschnitt basiert auf dem Paper von Yi He et al. [He+23], in der Hardwarefehler im Kontext von Deep-Learning-Trainingssystemen systematisch untersucht werden.

Als Hard Errors, auch bezeichnet als Hardware Fehler oder permanente Fehler, versteht man physische Defekte oder dauerhafte Alterungserscheinungen in den integrierten Schaltkreisen moderner Deep-Learning-Beschleuniger wie GPUs. Im Gegensatz zu Soft Errors, die durch äußere Einflüsse wie kosmische Strahlung verursacht werden, beruhen Hard Errors auf irreversiblen Schäden innerhalb der Hardwarestruktur.

Laut aktueller Forschung in Paper [He+23] zählen hierzu insbesondere:

- Fertigungsbedingte Restfehler, die durch Tests nicht erkannt wurden
- Materialalterung (engl. Circuit Aging)
- Frühausfälle (engl. Early Life Failures)
- Spannungsschwankungen (engl. Voltage Variations)
- sowie umweltbedingte Belastungen wie hohe Temperaturen oder mechanische Vibrationen.

Diese Fehler treten vor allem in den Logikkomponenten der Chips auf, etwa in arithmetischen Einheiten, ALUs oder Steuerpfaden. Speicherbereiche sind in der Regel besser geschützt, etwa durch Error Correction Code (ECC).

Ein typisches Merkmal von Hard Errors ist ihr nicht-deterministisches Verhalten:
Obwohl der Defekt permanent vorhanden ist, treten Fehler beim Ausführen identischer Workloads nur sporadisch auf – beispielsweise bei 3 von 10 Läufen. Solche Effekte sind schwer nachweisbar und erhöhen die Unsicherheit im Betrieb deutlich. Studien zeigen, dass etwa ein fehlerhafter Core pro 1.000 bis 10.000 Serverknoten betroffen ist [Dix+22], was bei rechenintensiven Anwendungen wie dem Training großer neuronaler Netze ein erhebliches Ausfallrisiko darstellt.[He+23]

Besonders kritisch ist die Verbreitung solcher Fehler innerhalb von Trainingsprozessen: Fehler in bestimmten Flip-Flops, etwa in Konfigurationsregistern oder in Datenpfaden (z.B. Exponentenbits in Gleitkommazahlen), können zu extremen Werten in internen Optimierungsvariablen führen, etwa bei Adaptive Fault-Tolerant Approximate Multiplier (AdAM) oder in Normalisierungsschichten. Solche Abweichungen wirken sich in der Regel nicht sofort sichtbar aus, führen aber zu sogenannten latenten Fehlermodi wie *SlowDegrade* oder *SharpDegrade* in Abbildung 9.4. Die Abbildung 9.4 beschreibt experimentell beobachtete Fehlermuster in NNs, die durch SBU oder MBU ausgelöst werden. Sie zeigt, unter welchen Bedingungen bestimmte Fehlertypen auftreten und in welchen Iterationen (t) bzw. Wertebereichen sie beobachtet wurden. Die Studie zeigt, dass diese Werte eine notwendige Bedingung für später auftretende fehlerhafte Trainingsergebnisse sind und bereits innerhalb von zwei Iterationen nach der Fehlerinjektion entstehen.[He+23, Abschnitt 4.2.6]

Outcomes	Necessary conditions	When conditions observed	Ranges observed in experiments
SlowDegrade	Large absolute gradient history values in optimizer	iter. t	3.6e9-1.1e19
Sharp SlowDegrade		iter. t	2.7e8-1.2e19
SharpDegrade	Large absolute *mvar* values in normalization layers	iter. $t+1$	6.5e16-1.2e38
LowTestAccuracy		iter. t	7.3e17-7.1e37
Short-term INFs/NaNs		iter. $t+1$	2.9e38-3.0e38

Abbildung 9.4.: Latente Fehlermodi [He+23]

Soft Errors

Die nachfolgende Analyse stützt sich auf die Ergebnisse des Papers von Ibrahim et al.[Ibr+20] in der die Ursachen, Auswirkungen und Typen von Software-Errors im Kontext von DNN-Beschleuniger (siehe Unterabschnitt 9.1.2) detailliert untersucht werden.

Die Soft Errors sind transiente Hardwarefehler, die durch externe physikalische Einflüsse wie ionisierende Strahlung, Temperaturfluktuationen oder elektromagnetische Störungen verursacht werden. Anders als permanente Defekte bzw. Hard Errors, führen Soft Errors nicht zu dauerhaften Schäden an der Hardware, können jedoch zu Bitfehlern in Speicherzellen (z.B. Static Random Access Memory (SRAM), DRAM) oder zu temporären Signalverfälschungen in Recheneinheiten führen [Ibr+20, Abschnitt 2.3, S. 4].

Aufgrund der hohen Parallelität moderner DNN-Beschleuniger wie GPUs, FPGAs oder ASICs kann bereits ein einzelner Soft Error erhebliche Auswirkungen auf die Modellgenauigkeit haben. Solche Fehler können sich über mehrere Schichten hinweg fortpflanzen und zu fehlerhaften Netzwerkausgaben führen [Ibr+20, Abschnitt 1, S. 1]. In Abbildung 9.15 wird etwa gezeigt, wie ein Bitflip in einem Gewichtswert dazu führt, dass ein neuronales Netz ein Objekt falsch klassifiziert

– beispielsweise ein Lastwagen irrtümlich als Vogel erkannt wird. In sicherheitskritischen Anwendungen wie der autonomen Fahrzeugsteuerung oder in der medizinischen Diagnostik kann ein derartiges Fehlverhalten schwerwiegende Konsequenzen haben. [Ibr+20]

Soft Errors lassen sich typischerweise in drei Hauptkategorien unterteilen: SEU, SBU und MBU. Diese werden im weiteren Verlauf des Kapitels detailliert analysiert.

Besonders anfällig für solche Fehler sind neben der Speicherhierarchie, also globalen und lokalen Speichern wie Caches und Registern, auch die Recheneinheiten selbst. Letztere führen zentrale Berechnungen innerhalb der Processing Elements durch und sind damit essenziell für die Funktionsfähigkeit des Netzwerks [Ibr+20, Abschnitt 2.2, S. 4]. Hinzu kommt, dass bei SRAM-basierten FPGAs auch die Konfigurationsbits betroffen sein können. Ein Soft Error in diesen Speicherbereichen kann zu einer fehlerhaften Logikverdrahtung führen und im schlimmsten Fall zu dauerhaften Fehlfunktionen [Ibr+20, S. 11–12].

Darüber hinaus zeigen sich je nach Netzwerkarchitektur unterschiedliche Auswirkungen auf die Fehlertoleranz. CNNs, die eine hohe Datenwiederverwendung in ihren Faltungsschichten aufweisen, reagieren besonders sensibel auf Fehler in Gewichtsmatrizen [Ibr+20, S. 10]. Im Gegensatz dazu weisen Multilayer Perceptronss (MLPs), die auf voll vernetzten Schichten basieren, eine höhere interne Fehlertoleranz auf, da sich Störungen aufgrund der vielen redundanten Verbindungen teilweise kompensieren lassen [Ibr+20, S. 13].

SEU

Die nachfolgenden Inhalte basieren auf dem Paper von Cai et al. [Cai+24], die sich mit der Bewertung und Minderung von SEUs im Zusammenhang mit Gewichtswerten in CNNs befassen.
Ein SEU beschreibt einen transienten Hardwarefehler, der entsteht, wenn ein einzelnes hochenergetisches Teilchen, typischerweise aus dem Weltraum oder der oberen Atmosphäre, in ein elektronisches Bauelement eindringt. Durch Ionisation wird eine elektrische Störung verursacht, die zum Umschalten eines logischen Zustands, also einem sogenannten Bit-Flip, führen kann. [Cai+24]

Zu den am stärksten betroffenen Elementen zählen SRAM-Zellen, in denen die Gewichte und Aktivierungen neuronaler Netze gespeichert sind. Ein SEU in einer solchen Speicherzelle kann die betroffenen Werte dauerhaft verändern und somit sämtliche nachfolgenden Lesevorgänge verfälschen (siehe Abbildung 9.5a). Ebenso gefährdet sind Register, in denen Zwischenergebnisse während der Berechnung verarbeitet werden. Fehler in diesen Registern können sich entlang der Signalpfade ausbreiten und die Netzwerkausgabe signifikant beeinflussen (siehe Abbildung 9.5b).

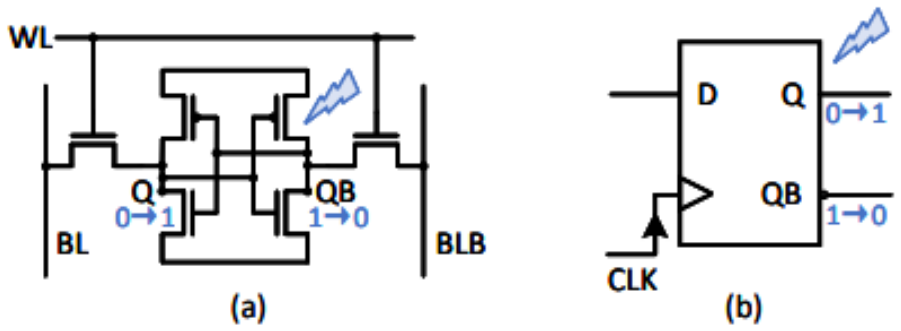


Abbildung 9.5.: Auswirkungen von SEU auf (a) SRAM und (b) Register [Cai+24]

Die unmittelbare Folge solcher Störungen ist eine Reduktion der Inferenzgenauigkeit - die Wahr-

scheinlichkeit der korrekten Vorhersagen - da zentrale Rechenwerte wie Gewichtungen und Aktivierungen nicht mehr korrekt verarbeitet werden [Du+19]. Besonders relevant ist dieses Phänomen in raumfahrtspezifischen Anwendungen [WA08], wie der satellitengestützten Fernerkundung, in denen neuronale Beschleuniger dauerhaft einer hohen Teilchenstrahlung ausgesetzt sind.

SEUs unterscheiden sich dabei klar von permanenten Defekten, wie sie beispielsweise durch Elektromigration oder Materialalterung entstehen. Während letztere zu dauerhaften Schäden führen, handelt es sich bei SEUs um flüchtige Weichfehler, die nur temporär auftreten. Zudem ist zu beachten, dass kombinatorische Logik von SEUs in der Regel nicht betroffen ist. Die Änderungen in diesen Strukturen wirken sich nicht auf den Zustand und damit auch nicht direkt auf die Genauigkeit neuronaler Netze aus. [DM03]

SEU ist der Oberbegriff für einen Soft Error, der durch ein einzelnes physikalisches Ereignis (z. B. kosmische Strahlung) ausgelöst wird. Ein SEU kann entweder ein einzelnes Bit (SBU) oder mehrere Bits (MBU) betreffen. Im Folgenden folgt die Differenzierung zwischen SBU und MBU.

SBU und MBU

Ein SEU kann dazu führen, dass der Inhalt einer einzelnen Speicherzelle verändert wird, bspw. das gespeicherte Bit kippt in seinen gegenteiligen Zustand. Dieses Phänomen wird als SBU bezeichnet. Je nach Art und Verlauf des Teilcheneinschlags kann es jedoch auch vorkommen, dass mehrere benachbarte Zellen gleichzeitig betroffen sind. In solchen Fällen spricht man von einem MBU [RS20, Kapitel 2, Absatz 3].

MBUs treten zwar nicht regelmäßig und auch nicht vollständig vorhersehbar auf, folgen jedoch bestimmten physikalischen Mustern. Laut Rohde et al. [RS20] können beim Einschlag eines hochenergetischen Teilchens ausschließlich benachbarte Speicherzellen beeinflusst werden. Eine einzelne Kollision wirkt sich somit nicht auf weit voneinander entfernte Zellen aus [Rao+14, Kapitel 2, Absatz 4].

Neuere Untersuchungen haben typische Fehlerverteilungen im Zusammenhang mit MBUs in FPGAs dokumentiert. Wie in Abbildung 9.6 dargestellt, treten Fehler bevorzugt in Zellen auf, die zwei Positionen entfernt in beliebiger Richtung liegen. Über die Jahre hat sich zudem gezeigt, dass mit fortschreitender Technologie die Wahrscheinlichkeit für Bitfehler in mehreren benachbarten Zellen deutlich gestiegen ist. [Rao+14, Kapitel 2, Absatz 5]

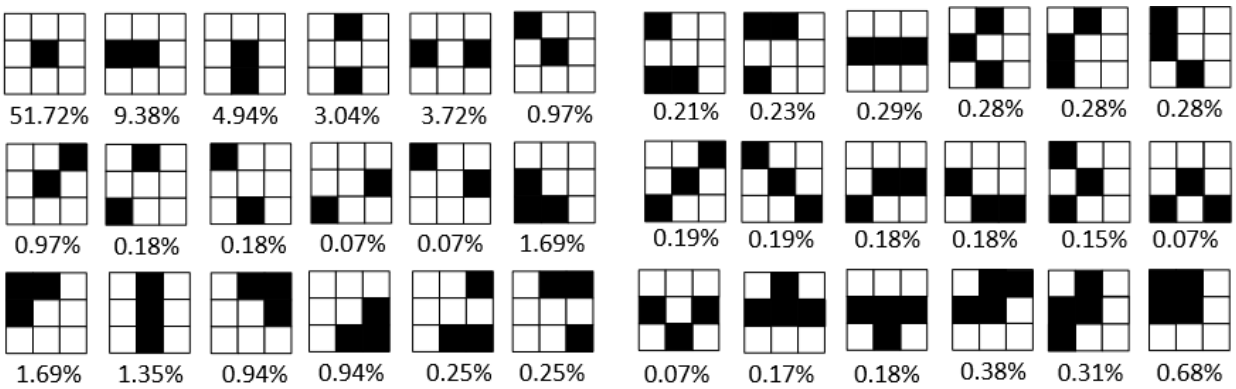


Abbildung 9.6.: Fehlerstandard für FPGA 45nm [Rao+14]

Die elektrische Wirkung eines solchen Teilchenereignisses auf die Schaltung selbst bleibt dabei grundsätzlich gleich. Allerdings bringt die stetige Verkleinerung der Strukturgrößen in der Halbleitertechnik die Speicherzellen räumlich näher zusammen, wodurch die Anfälligkeit für benachbarte

Fehler steigt. Dies wird auch durch den Vergleich zweier Speichertechnologien mit unterschiedlichen Strukturgrößen veranschaulicht: Bei identischer elektrischer Einwirkung zeigt sich, dass kleinere Zellen häufiger von MBUs betroffen sind (siehe Abbildung 9.7]). [Ama+16]

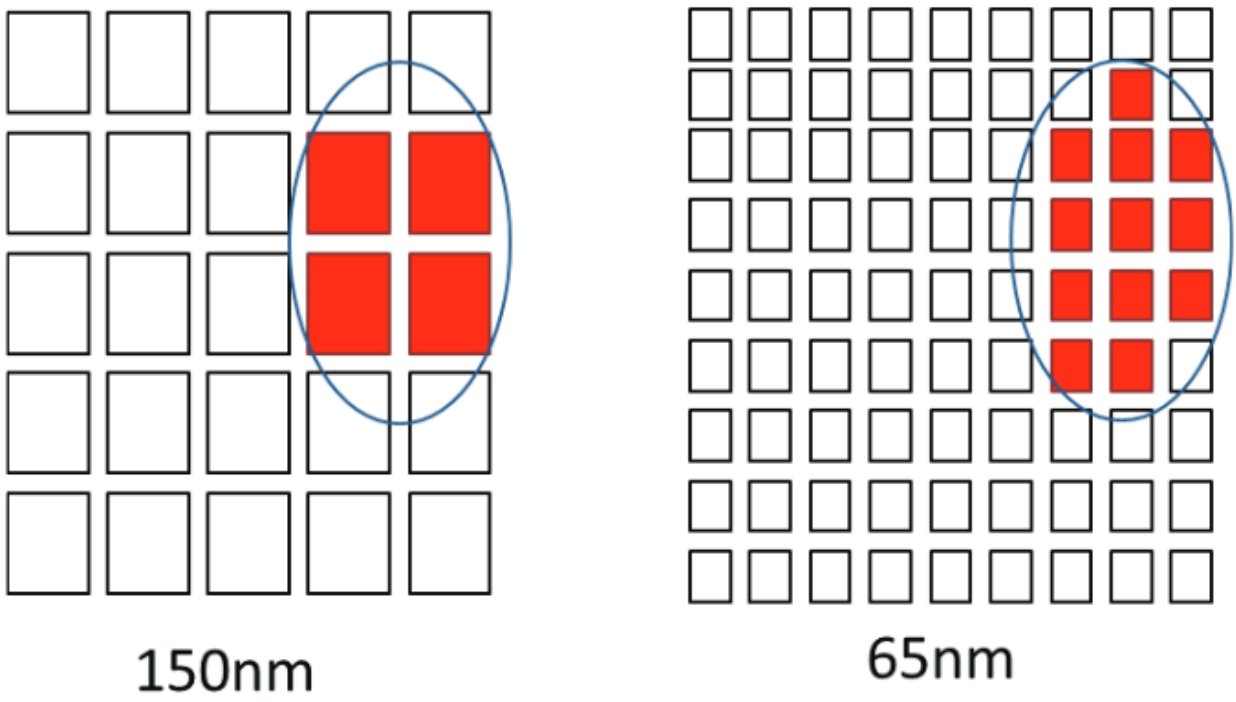


Abbildung 9.7.: Single Event Effekt in verschiedenen Speicherkapazitäten. [Ama+16]

9.1.4. Herausforderungen bei sicherheitsrelevanten Systemen

geschrieben von Tuanna Karadas

Die Integration NNs in sicherheitskritische Systeme im Weltraumkontext stellt eine anspruchsvolle Herausforderung dar, die sich aus den besonderen Umweltbedingungen des Weltraums sowie den strengen Anforderungen an Zuverlässigkeit und Sicherheit ergibt. Wie Forsberg et al. [For+20] herausstellen, lassen sich die zentralen Problembereiche in mehreren thematischen Schwerpunkten systematisch analysieren. Im Folgenden werden die zentralen Problemfelder zusammenhängend und strukturiert erläutert.

Repräsentativität und Qualität der Trainingsdaten

Ein zentrales Problem betrifft die Repräsentativität und Qualität der Trainingsdaten. Da die Leistungsfähigkeit neuronaler Netze wesentlich von der Qualität, Vielfalt und Vollständigkeit der verwendeten Trainingsdaten abhängt, stellt die eingeschränkte Verfügbarkeit realer Weltraumdaten ein erhebliches Hindernis dar. So muss die Aufteilung der Daten in Trainings-, Validierungs- und Testmengen sorgfältig erfolgen, um statistische Unabhängigkeit und Repräsentativität zu gewährleisten. Das ist ein Anspruch, der in der Raumfahrt aufgrund begrenzter Datenverfügbarkeit häufig schwer umzusetzen ist [For+20, S.2]. Infolgedessen kommt synthetischen Daten eine zentrale Rolle zu. Studien zeigen, dass eine geeignete Kombination aus realen und simulierten Datensätzen zu einer verbesserten Generalisierungsfähigkeit führt [Gai+16, Abschnitt II.A.1, S.

2]. Umwelteinflüsse wie kosmische Strahlung, extreme Temperaturschwankungen und das Vakuum im All können zudem zu signifikanten Verfälschungen in Sensordaten führen, was robuste und multimodale Sensorsysteme, etwa auf Infrarot- oder Strahlungssensorbasis, erforderlich macht. Zusätzlich ermöglichen Werkzeuge wie die Scenic-Szenariobeschreibungssprache eine programmatische Generierung synthetischer Weltraumszenarien. Auf diese Weise lassen sich seltene oder extrem belastende Situationen wie Sonnenstürme realistisch simulieren. [For+20, S. 2-3]

Unsicherheiten in der Klassifikation

Ein weiteres zentrales Thema ist der Umgang mit Unsicherheiten in den Klassifikationsergebnissen neuronaler Netze. Diese liefern in der Regel probabilistische Aussagen, was im Kontext sicherheitskritischer Weltraumanwendungen problematisch sein kann. Dabei wird zwischen aleatorischer Unsicherheit, die durch zufällige Schwankungen in den Eingangsdaten entsteht, und epistemischer Unsicherheit, die auf unvollständige oder unzureichende Abdeckung seltener oder extremer Weltraumszenarien in den Trainingsdaten zurückzuführen ist, unterschieden [For+20, Abschnitt 2.B, S. 3].

Ansätze wie Binarized Neural Network (BNN) ermöglichen es, Unsicherheiten durch Wahrscheinlichkeitsverteilungen explizit zu modellieren und auf diese Weise Vorhersagen mit hoher Konfidenz zu erkennen, die tatsächlich falsch sein könnten [LV20]. Ebenso findet das Monte-Carlo-Dropout-Verfahren Anwendung zur quantitativen Unsicherheitsabschätzung und wurde erfolgreich in Sicherheitsanalysen getestet [Hen+20]. [For+20, Abschnitt 3.B, S. 3]

Herausforderungen bei der Zertifizierung von Hardware und Software

Auch in Hinblick auf die Zertifizierung ergeben sich tiefgreifende Herausforderungen. Klassische Verfahren, die auf deterministischen Modellen beruhen, stoßen bei datenabhängigen neuronalen Netzen an ihre Grenzen. Da deren Verhalten wesentlich durch die verwendeten Trainingsdaten geprägt ist, ist eine vollständig deterministische Funktionsbeschreibung nicht möglich, was eine statistisch fundierte Verhaltensbeschreibung erforderlich macht [For+20, S. 3].

Das sogenannte W-Modell erweitert klassische Entwicklungsprozesse um den Aspekt der „Learning Assurance“ und zielt darauf ab, systematische Fehler während des Trainingsprozesses frühzeitig zu erkennen und zu vermeiden (Cluzeau et al., [Clu+20]).[For+20, S. 3]

Hinzu kommen besondere Anforderungen an die Hardware: Hardware-Beschleuniger für NNs sind anfällig für transiente Fehler, was den Einsatz von Redundanzmechanismen oder Fehlererkennungstechniken erforderlich macht [For+20, Abschnitt 2.C, S. 4].

Robustheit gegenüber Störungen und gezielten Angriffen

In sicherheitskritischen Szenarien müssen neuronale Netze zudem eine hohe Robustheit gegenüber externen Störeinflüssen und gezielten Angriffen, sogenannten adversarial attacks, aufweisen. Bereits minimale, gezielt manipulierte Veränderungen in den Eingabedaten können zu gravierenden Fehlklassifikationen führen [BBM20].

Besonders anfällig sind dynamisch lernende Netze, da sie auch im Betrieb ihre Gewichte verändern können. Aus Sicherheitsgründen wird in der Raumfahrt bevorzugt auf statisch trainierte Modelle gesetzt, die im Einsatz keine weiteren Lernprozesse mehr durchlaufen.[For+20, S. 4]).

Kontextabhängige Leistungsanforderungen

In sicherheitskritischen Anwendungen müssen die Leistungsmetriken eines neuronalen Netzes stets kontextabhängig definiert werden. In der Luftfahrt etwa ist eine exakte Klassifikation von Landebahnen essenziell. Hierbei ist eine geringe Rate an False Positives entscheidend. Gleichzeitig müssen Hindernisse zuverlässig erkannt werden, was eine möglichst geringe False Negative-Rate erfordert.
Unter False Positive wird definiert, wenn ein neuronales Netz ein Objekt oder Ereignis fälschlicherweise als vorhanden klassifiziert, obwohl es in Wirklichkeit nicht existiert. In sicherheitskritischen Szenarien kann dies zu unnötigen Systemreaktionen oder Fehlalarmen führen. Dagegen beschreibt ein False Negative den Fall, in dem ein reales, sicherheitsrelevantes Objekt vom neuronalen Netz nicht erkannt wird. Dies gilt als besonders kritisch, da potenziell gefährliche Situationen unbehandelt bleiben und somit ein erhöhtes Risiko für Personen oder Systeme entsteht. [For+20]

Fehleranfälligkeit der Hardwarekomponenten

Transient auftretende Hardwarefehler (Soft Errors) können die Ausgabe eines neuronalen Netzes beeinträchtigen. Ein Lösungsansatz besteht in der Verwendung sogenannter Monitor-Netze Error Detection and Mitigation Networks (EDMNs), die als zusätzliches neuronales Netz die Ausgabe des Hauptmodells überwachen und Anomalien erkennen können. Alternativ bieten klassische Redundanzverfahren wie Triple Modular Redundancy (TMR) einen bewährten, wenngleich ressourcenintensiven Schutzmechanismus [For+20, Abschnitt 2.F, S. 4].

Komplexität neuronaler Netze und deren Vereinbarkeit mit Sicherheitsanforderungen

Ein abschließender Aspekt betrifft das Spannungsfeld zwischen Modellkomplexität und Sicherheitsanforderungen. Während neuronale Netze in der Regel komplexe Architekturen aufweisen, fordert die sicherheitsorientierte Systementwicklung möglichst einfache und nachvollziehbare Lösungen. Der Einsatz zusätzlicher Überwachungsmechanismen erhöht zwar die Systemsicherheit, steigert jedoch auch die Komplexität [For+20, Abschnitt IV, S.6]. Um neuronale Netze dennoch sicher betreiben zu können, sind neue Ansätze zur Zertifizierung erforderlich, etwa in Form von Assurance Cases, die auch für lernende Systeme eine nachvollziehbare Risikobewertung ermöglichen. [For+20]

9.2. Relevante Faktoren für Fehler

geschrieben von Matthias Löwen und Arian Dannemann

Die Wahrscheinlichkeit für das Auftreten von Fehlern wird maßgeblich durch verschiedene technologische und physikalische Parameter, wie beispielsweise Strukturgröße, Spannung und Strahlung beeinflusst. Insbesondere bei neuronalen Netzen stellen diese Faktoren ein großes Risiko dar, da deren Ausführung speicherintensiv ist und stark parallelisiert erfolgt.

Im Folgenden werden verschiedene Aspekte, welche einen Einfluss auf die Fehlerrate von NNs insbesondere im Hinblick auf FPGAs haben können, aufgeführt. Dabei wird besonderes Augenmerk darauf gelegt, in welchem Verhältnis bestimmte Parameter zur Reduktion der Ergebnisqualität eines NN stehen.

9.2.1. Physikalische Fehlerarten

geschrieben von Matthias Löwen

Das Paper von Dixit und Wood von Oracle Labs [DW11] beschreibt die Durchführung von Neutronenstrahlungstests auf verschiedene Mikrocontroller und wie sich die verschiedenen Parameter auf die Soft Error Rate (SER) auswirken.

Ein relevanter Faktor ist die Strukturgröße eines Mikrocontrollers. In dem erwähnten Paper werden Strukturgrößen von 40nm bis 250nm betrachtet. Die Skalierung der Strukturgröße zeigt einen Einfluss auf die SEU-Rate. Während die Technologieknoten von 180nm bis 65nm eine kontinuierliche Abnahme der SEU-Rate pro Kilobit aufweisen, kehrt sich dieser Trend um. Bei der nächst kleineren Strukturgröße von 40nm steigt sowohl die SEU-Rate als auch die FITs/kbit (siehe Abbildung 9.8).

Tech. (nm)	***Relative SEU rate in FITs/kbit***	**Approx. Mbits per microprocessor**	**Relative uncorrected SEU rate per microprocessor (kFIT)**
250	3.2	1.52	5.0
180	3.0	1.52	4.3
130	2.4	3.28	7.9
90	1.0	33.6	33.6
65	0.7	44.3	30.5
40	0.94	71.0	67.0

Abbildung 9.8.: SEU-Rate für Mikroprozessoren unterschiedlicher Größe [DW11, S. 4]

Bei einer Strukturgröße von 40nm bis 90nm ist der Anteil der SEUs bei Flip-Flops höher als beim SRAM. Dies hängt laut Dixit und Wood [DW11] damit zusammen, dass die Schutzmechanismen von Flip-Flops, wie z.B die Zustandskodierung technisch komplexer und ressourcenintensiver sind, als ECC bei SRAM. Mit sinkender Strukturgröße werden Flip-Flops daher zu einem relevanten Fehlerfaktor, auch wenn sie zahlenmäßig den SRAM-Zellen deutlich unterlegen sind. Ein weiteres Problem von einer kleinen Strukturgröße ist das häufigere Auftreten von Multi Cell Upset (MCU). Diese betreffen dabei häufig zusammenhängende Bits, wenn die Architektur keine Gegenmaßnahmen getroffen hat, und können so schwieriger korrigiert werden und zu einer höheren Fehlerwahrscheinlichkeit führen.

Die Betriebsspannung hat einen signifikanten Einfluss auf die Fehleranfälligkeit. Hierbei gilt, dass je kleiner die Betriebsspannung ist, desto höher ist die SEU-Rate. Bei einem 40nm großem Technologieknoten steigt die SEU-Rate um 30% pro 0,1V, während die Spannung von 1,25V auf 0,5V gesenkt wird.

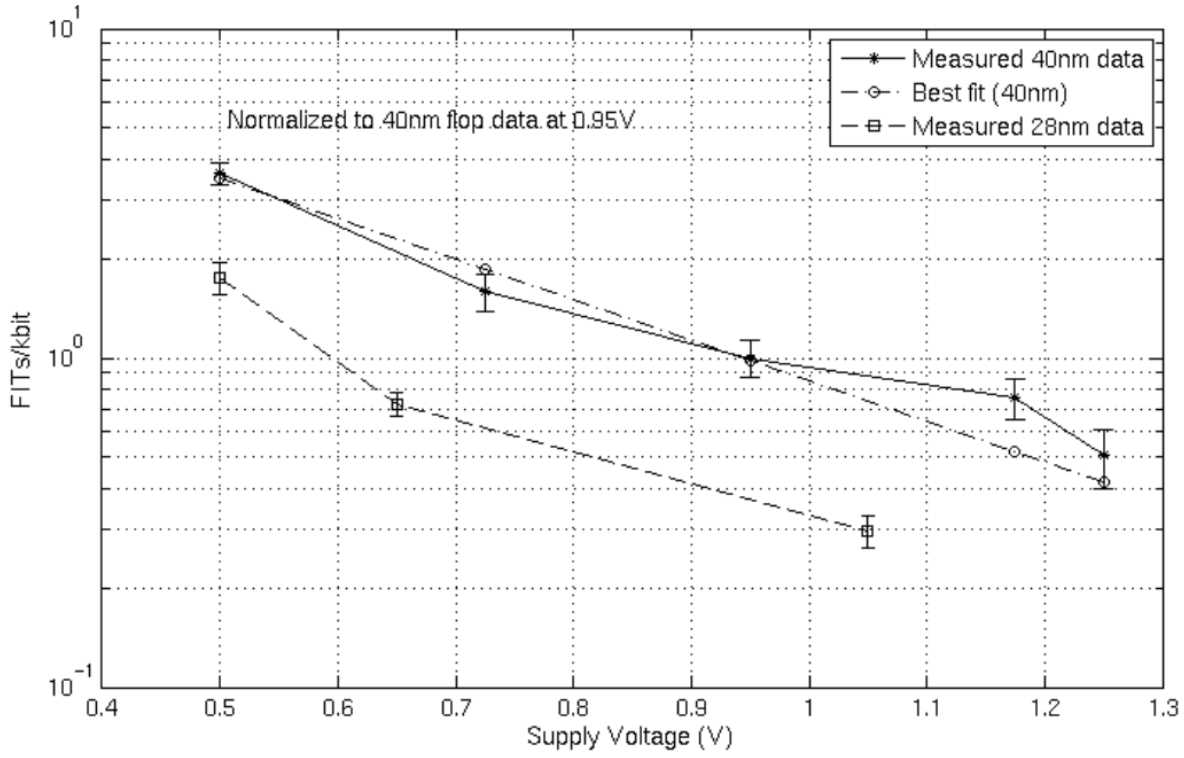


Abbildung 9.9.: SEU-Rate in Abhängigkeit von der Versorgungsspannung [DW11, S. 3]

Der Zusammenhang lässt sich dadurch erklären, dass eine niedrigere Spannung auch eine geringere kritische Ladung zur Änderung des logischen Zustands bedeutet. Außerdem kommt es bei Mikrocontrollern, welche ein dynamisches Spannungs- und Frequenzscaling zum Senken des Stromverbrauchs nutzen, zu einem Anstieg der SEU-Rate. Als eine Stromsparmaßnahme, welche die SEU-Rate nicht signifikant steigen lässt wurde die back-bias Technik genannt. Bei der wird eine voreingestellte Spannung am Rückkontakt eines Transistors angelegt, um gezielt den Schwellwert zu beeinflussen.

9.2.2. Bitbreite

geschrieben von Arian Dannemann

Ein weiterer Faktor, der die Fehleranfälligkeit eines neuronalen Netzes direkt beeinflusst, ist seine Topologie. Ein Ansatz, um den Platzbedarf eines NN zu reduzieren ist es, das Netz zu quantisieren. Unter Quantisierung versteht man das „Herunterbrechen“ eines kontinuierlichen Eingangssignals, beispielsweise einer physikalischen Spannung, auf eine Anzahl kleiner Teilbereiche, beispielsweise eine digitale Variable vom Typen `Float`, welche die Spannung abbilden soll. Im Bereich der NNs ist der Begriff der Quantisierung besonders auf die Breite der Datenleitungen zwischen den Neuronen zu verstehen. Statt einem üblichen `Float` können die Werte der Gewichte und Aktivierungen zum Beispiel auf einen `Int` quantisiert werden.

Khoshavi et al. [KBB20] betrachteten den Unterschied zwischen einem QNN und einem BNN, also einem Netzwerk bei dem die Bitbreiten für Gewichte sowie Aktivierungen auf eine definierte Genauigkeit quantisiert wurden. Dabei beschreibt QNN ein beliebig quantisiertes Netzwerk, während ein BNN voll binarisiert wurde, also lediglich die Werte $\{-1, 1\}$ abbildet [Cou+16].

Ein BNN kann im Vergleich zu einem DNN Speicherplatz sparen, da es für die Darstellung der Gewichte und Aktivierungen nur noch ein Bit des Speichers benötigen, während beide im Falle eines herkömmlichen `Float`-Wertes bereits 32 Bit bräuchten.

Während ein BNN aus Gründen der Reduktion der benötigten Speichergröße durchaus für Raumfahrtsysteme relevant sein kann, muss betrachtet werden, dass hier die Auswirkungen von SEUs und MBUs signifikant erhöht sind.

Khoshavi et al. [KBB20] haben eine Reihe von Fault-Injection-Tests auf ein BNN sowie ein QNN durchgeführt. Verglichen wurde eine Convolutional Network Topology (CNV)[1] in zwei Varianten. Die erste Variante verwendet jeweils zwei Bits für die Gewichte und Aktivierungen, die zweite Variante jeweils ein Bit.

Es wurden hier mehrere Versuchsdurchläufe durchgeführt. Getestet wurde jedes CNV seperat in einer Reihe von Schritten. Die Netze waren bereits trainiert und es wurden im laufenden Betrieb Fehler injiziert, je nach aktuellem Durchlauf entweder innerhalb der Gewichte oder der Aktivierungen des Netzes. Für jedes CNV wurden zweitausend Durchläufe mit den folgenden Fehleranzahlen pro Durchlauf durchgeführt: $\{1, 2, 5, 10, 20, 50, 100\}$. Dabei ist aufgefallen, dass das „ein-Bit-cnv“ bei dem extremsten Fall von einhundert injizierten MBUs einen durchschnittlichen Genauigkeitsverlust von 8.43% hatte, während das „zwei-Bit-cnv“ einen geringeren Verlust von 0.7% erlitt. Hier liegt also die Schlussfolgerung nahe, dass die Reduktion von Speicherbits einen erheblichen Einfluss auf das Ergebnis des NNs hat. Ein Ausschnitt der Vergleichstabelle ist in Abbildung 9.10 zu sehen.

Network Topology	# of injected faults	Activation			
		SEU		MBU	
		Accuracy Reduction	Effective Faults (%)	Accuracy Reduction	Effective Faults (%)
cnvW1A1	1 Fault	0.0001 (0.01%)	62.0	0.00056 (0.07%)	66.0
	2 Faults	-0.00034 (-0.04%)	77.0	0.00096 (0.12%)	75.0
	5 Faults	0.00159 (0.20%)	87.0	0.00224 (0.28%)	88.0
	10 Fualts	0.00345 (0.43%)	94.0	0.00482 (0.60%)	97.0
	20 Faults	0.00762 (0.95%)	98.0	0.00883 (1.10%)	95.0
	50 Faults	0.01963 (2.44%)	99.0	0.03740 (4.65%)	100.0
	100 Faults	0.05270 (6.54%)	100.0	0.06787 (8.43%)	100.0
cnvW2A2	1 Fault	-0.00148 (-0.17%)	29.0	-0.00058 (-0.07%)	38.0
	2 Faults	-0.00118 (-0.14%)	65.0	-0.00123 (-0.14%)	65.0
	5 Faults	-0.00216 (0.25%)	90.0	-0.00190 (-0.22%)	86.0
	10 Faults	-0.00085 (-0.10%)	91.0	-0.00064 (-0.08%)	90.0
	20 Faults	0.00054 (0.06%)	95.0	-0.00058 (0.07%)	91.0
	50 Faults	0.00193 (0.23%)	94.0	0.00232 (0.27%)	97.0
	100 Faults	0.00711 (0.83%)	97.0	0.00601 (0.70%)	92.0

Abbildung 9.10.: Ausschnitt der Vergleichstabelle für die Fehleranfälligkeit von BNNs. Die Werte der Versuche, in denen Fehler innerhalb der Gewichte des Netzes injiziert wurden, wurde aus der Grafik ausgeschnitten. Es konnte kein Zusammenhang zwischen den (ausgeschnittenen) injizierten Fehlern und einem Genauigkeitsverlust festgestellt werden, die Lesbarkeit der übrigen Werte hatte Vorrang. [KBB20]

[1] Der Aufbau eines CNV ist für diesen Abschnitt nicht weiter relevant, der Begriff wird der Vollständigkeit halber erwähnt. Da alle Versuchsdurchläufe eine CNV benutzen, ist der Unterschied in der Bitbreite der Gewichte und der Aktivierungen der entscheidende Unterschied.

In Abbildung 9.10 ist außerdem zu erkennen, dass der Verlust der Genauigkeit nicht nur mit der Bitbreite, sondern auch direkt mit der Anzahl von aufgetretenen Soft-Errors zusammenhängt.

Beides ist intuitiv nachvollziehbar. Sollte die Bitbreite der Datenleitungen reduziert werden, fallen die wenigen vorhandenen Bits (oder im Falle eines BNN das eine vorhandene Bit) stärker ins Gewicht. Desto mehr fehlerhafte Bits hinzukommen, desto schlimmer wird dieser Effekt.

9.2.3. Topologie

geschrieben von Arian Dannemann

Der Ort der Fehlerquelle ist nicht nur auf dem Board, sondern auch innerhalb eines NNs entscheidend. Je nachdem, in welcher inneren Schicht eines NNs ein Bit-Flip auftritt wurden unterschiedlich drastische Reduktionen der Genauigkeit festgestellt.

Khoshavi et al. [KBB20] fanden durch Tests heraus, dass Fehler die in früheren Schichten eines Netzwerks auftreten einen höheren Einfluss auf die Genauigkeit des Ergebnisses haben. Dabei wurden die Ergebnisse der Aktivierungsfunktionen einer bestimmten Ebene eines CNV durch injizierte SEUs gestört und die Klassifikationsgenauigkeit des Ergebnisses bestimmt. Die Resultate sind in Abbildung 9.11 zu sehen.

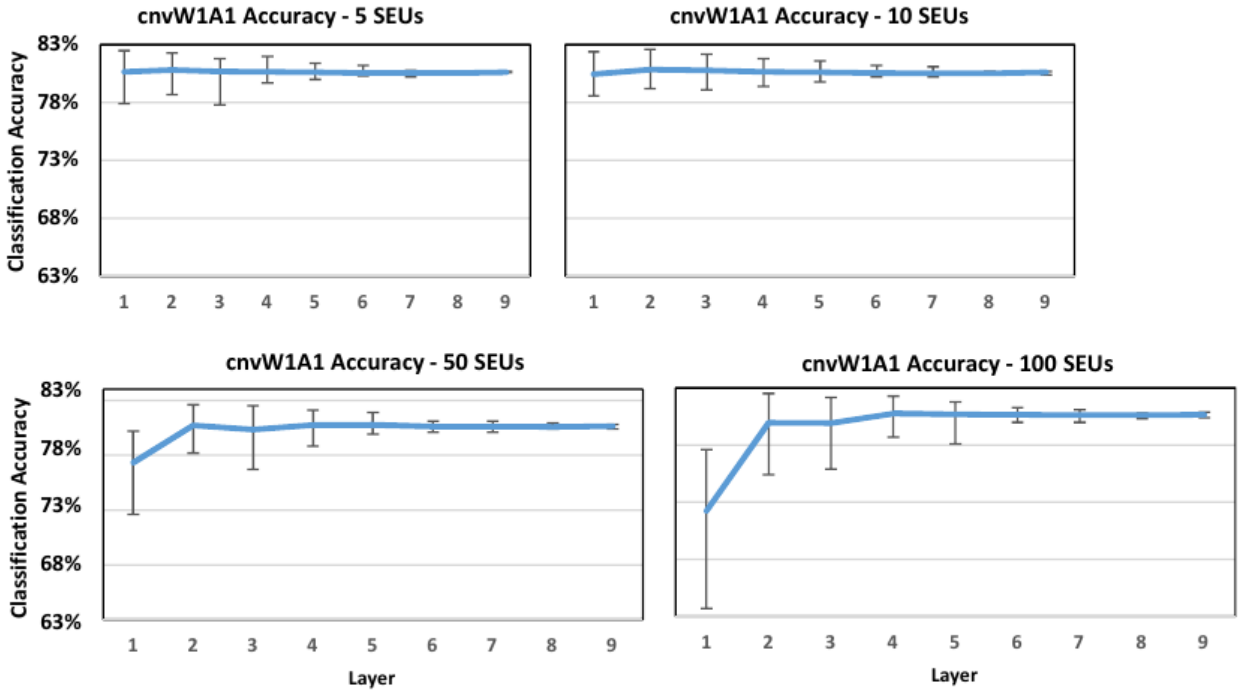
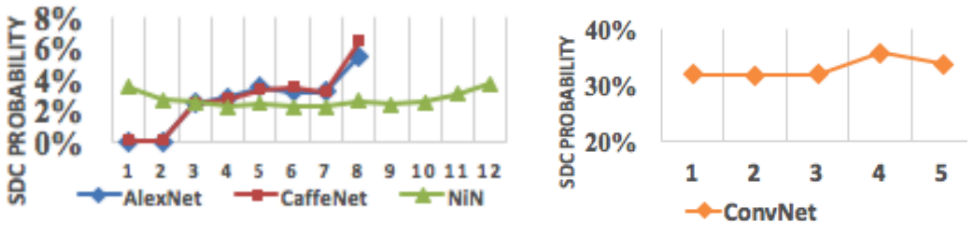


Abbildung 9.11.: Auswirkung von SEUs bezogen auf die Ebenen der Neuronen innerhalb der Topologie. Betrachtet wurde eine CNV mit je einem Bit für Gewichte und Aktivierungsfunktionen [KBB20]

Tritt also ein SEU in Schicht eins auf (nahe der Eingabeneuronen), so sind die potentiellen Auswirkungen gravierender als in Schicht neun (nahe der Ausgabeneuronen). Der Grund dafür ist die Fortpflanzung von Fehler innerhalb eines NN. Wenn ein Fehler innerhalb der ersten Schicht auftritt, dann werden die fehlerhaften Ausgaben der Neuronen der ersten Schicht als Eingaben der Neuronen der folgenden Schichten verwendet. Wer mit falschen Werten rechnet, erhält aller Wahrscheinlichkeit nach ebenfalls falsche Werte.

Wichtig zu betrachten ist die Art der Schichten, in denen ein Fehler auftritt. Während die CNV diese Eigenschaft aufweist, können NNs mit einem Local Response Normalization Layer (LRN) den gegenteiligen Effekt zeigen. Ein LRN normalisiert die Aktivierungen eines Neurons in Bezug auf die Aktivierungen seiner Nachbarn. Dies geschieht, indem die Aktivierung eines Neurons durch die Summe der Aktivierungen in einem bestimmten lokalen Bereich geteilt wird. Durch die Normalisierung der ersten Schichten in den NNs der Arbeit von [Li+17], sichtbar in Abbildung 9.12, können große Abweichungen mitigiert werden.



(a) AlexNet, CaffeNet and NiN (Error bar is up to 0.39%)

(b) ConvNet (Error bar is up to 0.77%)

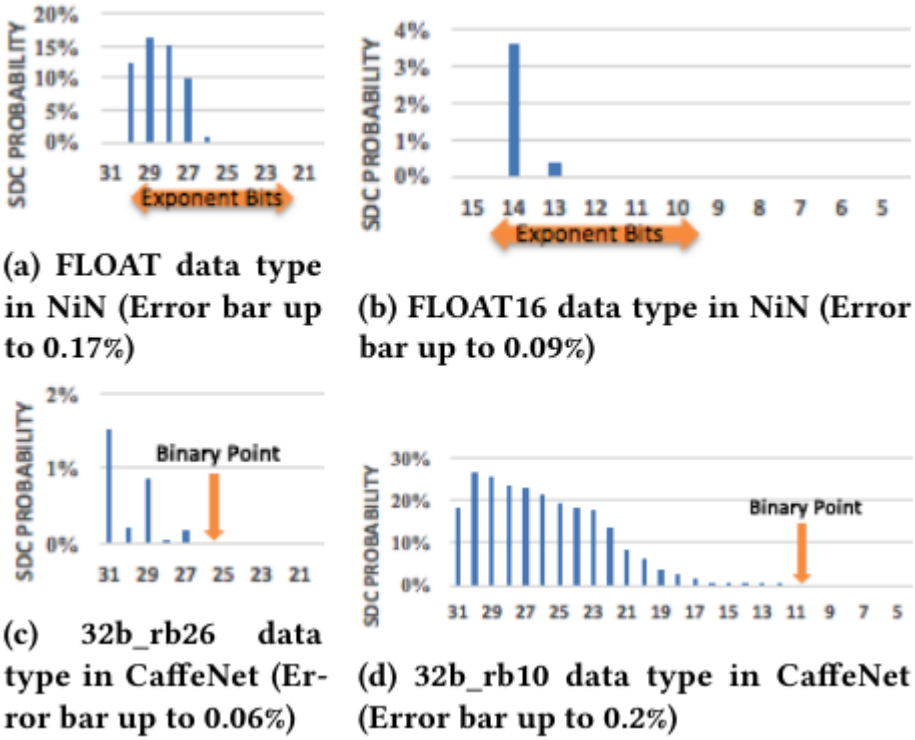
Abbildung 9.12.: Wahrscheinlichkeit von SDCs bezogen auf die Ebenen der Neuronen innerhalb der Topologie. Die verwendeten Netze können geneigte Leser unabhängig recherchieren, relevant ist hier vor allem die Differenz zu Abbildung 9.11, dass die ersten zwei Schichten in Abbildung (a) von AlexNet und CaffeNet deutlich *weniger* anfällig für SDCs sind. [Li+17]

Es kann festgehalten werden, dass verschiedene Topologien eines NN drastisch unterschiedliche Anfälligkeiten für Fehler mit sich bringen. Bevor ein NN auf einem Beschleuniger wie einem FPGA benutzt werden kann, sollte eine detaillierte Recherche zu den Fehleranfälligkeiten sämtlicher geplanter Schichtarchitekturen durchgeführt werden.

9.2.4. Bitposition

geschrieben von Arian Dannemann

Li et al. [Li+17] untersuchten die Fehlerfortpflanzung in DNNs, dabei ist die Bit-Position innerhalb eines Wertes für die Auftrittswahrscheinlichkeit eines SDC relevant. Betrachtet wurden verschiedene Datentype wie `Float` oder `Int`, bei denen einzelne Bits invertiert wurden. Anhand der Wahrscheinlichkeit des Auftretens eines SDC-Ereignisses nach Flippen des Bits konnte nachvollzogen werden, wie hoch der Einfluss des Bits auf die Gesamtergebnisse der Berechnung sind.



(a) FLOAT data type in NiN (Error bar up to 0.17%)

(b) FLOAT16 data type in NiN (Error bar up to 0.09%)

(c) 32b_rb26 data type in CaffeNet (Error bar up to 0.06%)

(d) 32b_rb10 data type in CaffeNet (Error bar up to 0.2%)

Abbildung 9.13.: SDC-Wahrscheinlichkeit in Abhängigkeit der Bitposition im Datentypen. Nicht gezeigte Bit-Positionen haben eine SDC Wahrscheinlichkeit von 0%. Li et al. [Li+17] beschreiben die Datentypen wie `32b_rb10` genauer, für diesen Bericht ist allerdings nur wichtig zu verstehen, dass verschiedene Bits innerhalb des gleichen Datentypen einen unterschiedlichen Einfluss auf das Resultat haben können. [Li+17]

9.2.5. Flip-Richtung

geschrieben von Arian Dannemann

Ebenfalls darf die „Richtung“ des Bit-Flips nicht ignoriert werden. Ein höherwertiges Exponentenbit welches von 0 zu 1 springt, birgt eine höhere Wahrscheinlichkeit ein SDC zu erzeugen, als wenn es von 1 zu 0 springen würde.

Dies lässt sich dadurch erklären, dass in vielen verwendeten NNs die korrekten Werte der Aktivierungsfunktionen der Neuronen üblicherweise um 0 herum gehäuft sind [Li+17]. In Abbildung 9.14 werden die Abweichung zwischen den eigentlichen korrekten Werten und den tatsächlichen Werten nach Bit-Flip innerhalb eines NN aufgezeigt. Relevant ist hier der Zusammenhang zur erhöhten Auftrittswahrscheinlichkeit einer SDC.

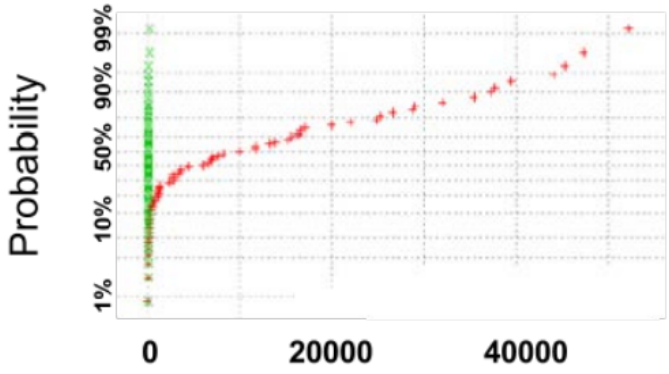


Abbildung 9.14.: Abhängigkeit zwischen Abweichungen zur korrekten Aktivierung und Auftrittswahrscheinlichkeit von SDCs im NN AlexNet mit `Float`-Datentypen. Die X-Achse stellt den erwarteten (grün) und tatsächlichen (rot) Wert der Aktivierung dar, die Y-Achse stellt die Wahrscheinlichkeit des Auftretens eines SDC-Ereignisses bei der beschriebenen Abweichung dar. [Li+17]

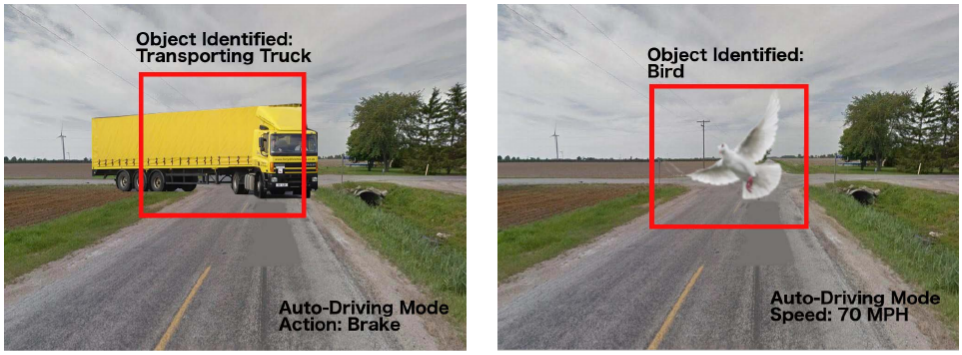
9.3. Auswirkungen

geschrieben von Arian Dannemann

Die Bandbreite der möglichen Auswirkungen von Rechenfehlern innerhalb eines NN ist groß, die der Schweregrade ebenso.

Während eine Fehlklassifizierung, beispielsweise bei einer Gesichtserkennung, keine Konsequenzen abseits der emotionalen Belastung der nutzenden Person aufgrund ihres Zeitverlustes nach sich führt, und der Prozess leicht erneut durchgeführt werden kann, wären die Nachwirkungen eines selbstfahrenden Lasters, der einen entgegenkommenden Laster nicht als Laster, sondern als Vogel erkennt und somit kein Bremsverhalten einleitet, nicht nur dramatisch, sondern auch tödlich.

Um solche Fehler zu vermeiden, wurde in der ISO 26262 eine Norm beschrieben, welche die Failure In Time (FIT)-Rate der Hardwarekomponenten eines Kraftfahrzeugs vorgibt. Diese schreibt vor, dass bei einer Milliarde Stunden Laufzeit maximal zehn Fehler auftreten dürfen [Ahm+24a].



(a) Fault-free execution: A truck is identified by the DNN and brakes are applied

(b) SDC: Truck is incorrectly identified as bird and brakes may be not applied

Abbildung 9.15.: Beispiel einer dramatischen Fehlklassifizierung [Li+17]

Wie bereits beschrieben, können sich Fehler innerhalb eines NN fortpflanzen und somit bedeutende Ungenauigkeiten bewirken. In den nachfolgenden Unterabschnitten wird genauer auf das beobachtbare Fehlverhalten, welches durch die oben genannten Fehler entstehen kann, eingegangen.

9.3.1. Klassifikation

geschrieben von Arian Dannemann

Bei NNs, die Eingabedaten klassifizieren, können bei einer Fehlklassifizierung verheerende Auswirkungen entstehen.

Koshavi et al. [KBB20] untersuchten die Reduktion in der Klassifikationsgenauigkeit. Hierbei wurde ein identischer Untersuchungsprozess zu dem in Unterabschnitt 9.2.2 beschriebenem durchgeführt. In diesem Versuch lag das Augenmerk allerdings vor allem auf der Klassifikationsgenauigkeit eines Layer Fully Connected (LFC) Netzwerkes mit einer Bitbreite von 1 für je die Gewichte und Aktivierungen. Es besteht auf vier voll verbundenen Schichten. Die durchschnittliche Reduktion bei Injizierung von einhundert MBUs in ein LFC liegt bei 5.14% und wirkt somit verhältnismäßig schmal. Die Varianz der Ergebnisse jedoch ist drastischer und nicht zu vernachlässigen.

Im genannten Beispiel kann im Worst-Case-Szenario eine Genauigkeitsreduktion von bis zu 76.7% vorliegen (von 98.4% auf 22.92%). Die vorliegenden Ergebnisse können in Abbildung 9.16 eingesehen werden.

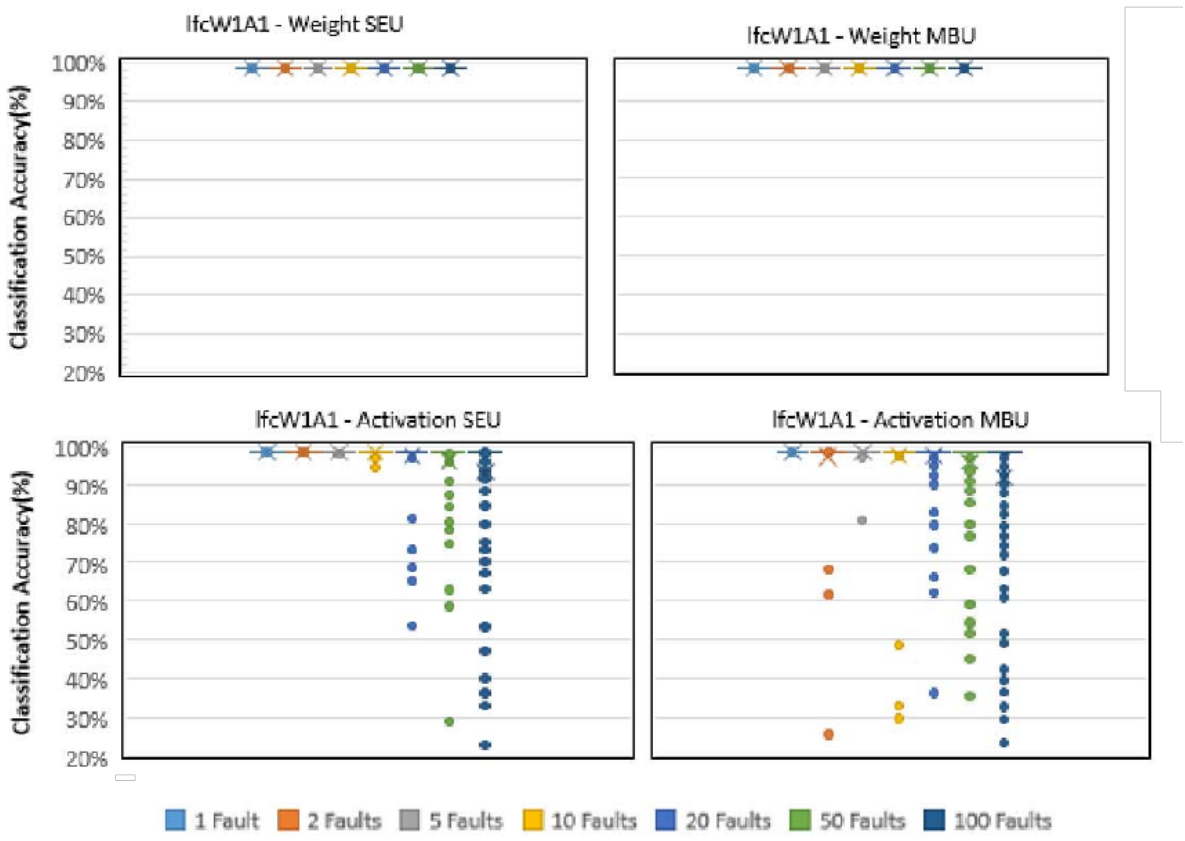


Abbildung 9.16.: Distribution der Klassifikationsgenauigkeitsreduktion im Verhältnis zu den injizierten Fehlern im LFC [KBB20]

Es kann festgehalten werden, dass die Auswirkung von SDCs extrem sein können und nicht vernachlässigt werden darf.

9.3.2. Verstärkung durch Parallelisierung

geschrieben von Matthias Löwen

DNN-Beschleuniger setzten stark auf Parallelität, um die vielen Rechnungen in möglichst kurzer Zeit abarbeiten zu können. Dieser Vorteil ist allerdings eine Schwäche in Hinblick auf die Fehlerausbreitung. So kann nach Ibrahim et al. [Ibr+20] ein einzelner Bit-Flip in einer Matrixmultiplikation durch die parallele Weiterverarbeitung gleichzeitig mehrere Neutronen negativ beeinflussen.

Falls als DNN-Beschleuniger eine GPU genutzt wird, sollte beachtet werden, dass durch die parallelisierte Speicherhierarchie und das Teilen von verschiedenen Ressourcen auch hier ein Bit-Flip an der falschen Stelle gleich mehrere Fehler verursachen kann.

Die Wirkung eines Fehlers hängt auch von seiner Position im neuronalen Netz ab. Falls ein Fehler schon früh in einem Netzwerk auftritt, kann er sich durch das gesamte neuronale Netz fortpflanzen, falls es keine Gegenmaßnahmen gibt. Allerdings hängt die genaue Fehlerabhängigkeit der verschiedenen Schichten stark von der genutzten Architektur ab [Ibr+20, S. 9].

9.4. Maßnahmen

geschrieben von Kevin Lingk

Zur Verhinderung der in Abschnitt 9.3 genannten Auswirkungen gibt es einige unterschiedliche Methoden, um Hardware-Fehler zu erkennen und zu kompensieren. Dadurch soll sichergestellt werden, dass die neuronalen Netze auch bei auftretenden Fehlern stabil arbeiten und korrekte Ergebnisse liefern. Obwohl DNNs intrinsische, fehlertolerante und fehlerresistente Eigenschaften wie Redundanz durch Überprovisionierung, verteilte Verarbeitung und teilweise Unabhängigkeit der Neuronen besitzen [TG17], reichen diese nicht aus, wenn die spezifischen Eigenschaften der DNN-Beschleuniger nicht berücksichtigt werden.

In diesem Abschnitt werden deshalb Redundanzverfahren, sowie adaptive Approximationsansätze betrachtet, um die Fehlertoleranz der neuronalen Netze zu erhöhen. Die vorgestellten Maßnahmen reichen von hardwarebasierten Redundanzlösungen wie TMR und selektiver Härtung über adaptive Multiplikationsverfahren bis hin zu softwarebasierten Fehlererkennungsmethoden.

9.4.1. TMR: Triple Modular Redundancy

geschrieben von Matthias Löwen

Eine Möglichkeit, um sich vor dem Fehlerfall in neuronalen Netzen zu schützen, ist die Implementierung von TMR. Dabei werden Module dreifach ausgeführt und das korrekte Ergebnis über eine Mehrheitsentscheidung bestimmt. Diese Technik kann SEUs präzise maskieren, sorgt allerdings für einen sehr hohen Ressourcenbedarf. Dieser Overhead beläuft sich auf 200%. [AHL20]

Im Folgenden wird eine Architektur vorgestellt, die die Vorteile der TMR nutzt, dabei jedoch die typischen Nachteile, wie den hohen Ressourcenverbrauch, weitgehend vermeidet [Sha+24]. Damit die Vorteile von der TMR genutzt werden können, ohne dass dreimal so viele Ressourcen

benötigt werden, wird die TMR nur selektiv eingesetzt. Es werden dabei vor allem die Module mit einer Redundanz abgesichert, bei denen ein hohes Fehlerrisiko besteht. Dazu zählen das Befehlsparser-Modul (instruction parsing module siehe Abbildung 9.17) und die erste Schicht des CNN. Das hat den Grund, dass das Befehlsparser-Modul eine zentrale Rolle bei der Berechnung des Beschleunigers spielt. Das Modul sorgt für die Dekodierung und Verteilung von Befehlen.

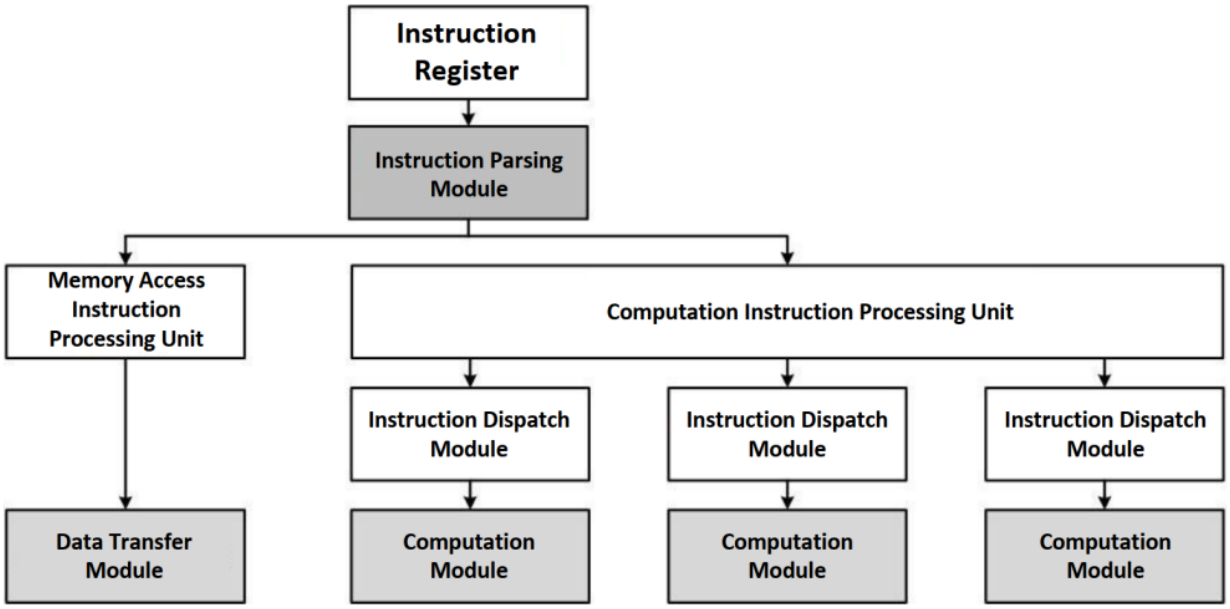


Abbildung 9.17.: Befehlskodierungs-Architektur [Sha+24, S. 11]

Außerdem wird die erste Schicht des neuronalen Netzes durch ein TMR abgesichert, da sich veränderte Gewichtungen ansonsten über das gesamte Netz fortpflanzen und zu fehlerhaften Klassifikationen führen könnten.

9.4.2. AdAM: Adaptive Fault-Tolerant Approximate Multiplier

geschrieben von Kevin Lingk

Neben der TMR gibt es AdAM, welches von Taheri et al. [Tah+24a] als ein Verfahren zur adaptiven Fehlertoleranz in der approximierten Multiplikation beschrieben wird.

Um die Notwendigkeit und den AdAM-Ansatz selbst zu verstehen, ist zunächst eine Betrachtung der zugrundeliegenden Problemstellung erforderlich. In DNNs ist die Multiplikation die ressourcenintensivste Operation, bei der Multiplizierer die primären arithmetischen Bausteine darstellen. Diese bringen erhebliche Anforderungen an die Hardware mit sich, darunter:

- einen hohen Flächenverbrauch auf dem Chip,
- einen hohen Energieverbrauch,
- und längere Verzögerungen durch die intensiven und komplexen Rechenoperationen.

Dadurch, dass die Prozessoren und kombinatorischen Schaltungen immer komplexer und kleiner werden, nimmt die Soft Error Rate (SER) zu, wodurch unvorhersehbare Effekte bspw. in Umgebungen mit viel Strahlung (Weltall) immer häufiger werden [Tah+24a].

Um diesem Problem entgegenzuwirken und einen optimalen Kompromiss zwischen der Hardware-Effizienz und Fehlertoleranz zu erreichen, kommt das AdAM-Verfahren zum Einsatz. Dabei wird das Power-Delay Product (PDP) als Produkt aus Leistungsbrauch und Rechenzeit als zentrale Metrik zur Bewertung der Energieeffizienz von Schaltungen verwendet. Herkömmliche Ansätze wie die TMR, die bereits beschrieben wurde, erreichen zwar hohe Fehlertoleranz, verursachen jedoch ein stark erhöhtes PDP durch den nahezu 200%igen Hardware-Mehraufwand. [Tah+24a]

Nach Taheri et al. [Tah+24a] unterscheidet sich das AdAM-Verfahren grundlegend von der TMR, da es auf einem adaptiven Ansatz basiert, bei dem die Multiplikation mit approximierten Werten erfolgt. Dabei werden bei der Multiplikation nur die wichtigsten Komponenten (Most Significant Bit (MSB)) geschützt und doppelt berechnet. Zum Bestimmen des Produktes der Multiplikation arbeitet AdAM mit den approximierten logarithmischen Zahlen der Inputs, welche anschließend addiert werden. Dabei basiert dieser Ansatz auf dem Mitchell-Multiplizierer.

Die Funktionsweise des adaptiven Schutzes der MSB wird in Abbildung 9.18 deutlich. Mithilfe eines Leading One Detectors (LODs) wird die führende „1“ bestimmt, damit festgelegt werden kann, wie viel Bits geschützt werden. Der Grund dafür ist, dass bei großen Zahlen die unteren Bits nur einen geringen Einfluss auf den Gesamtwert haben und daher weniger geschützt werden müssen. Bei kleinen Zahlen hingegen tragen mehr Bits wesentlich zum Gesamtwert bei, sodass ein Verlust dieser Bits stärker ins Gewicht fallen würde. [Tah+24a]

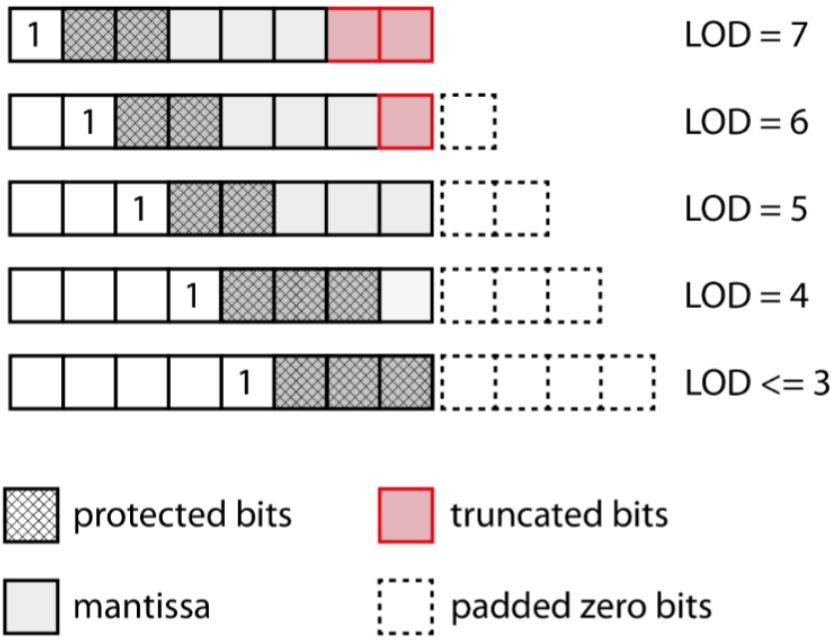


Abbildung 9.18.: Fehlertoleranz und eingeführter Fehler in Abhängigkeit der Position der führenden Eins [Tah+24a]

In Abbildung 9.18 ist erkennbar, dass die Anzahl der geschützten Bits (grau schraffiert) mit abnehmender führenden Eins zunimmt. Bei größeren Zahlen (LOD $= 7$) werden nur die zwei wichtigsten Bits der Mantissa geschützt. Taheri et al. [Tah+24a] schreiben hierzu, dass aufgrund einer Mantissa-Verkürzung bis zu zwei der Least Significant Bits (LSBs) von der Berechnung ausgeschlossen werden. Dieser Fall betrifft allerdings nur die Fälle LOD $= 7$ und LOD $= 6$. Der Grund für die Anpassung ist, dass es im Vergleich zum ursprünglichen Mitchell-Algorithmus zu einem geringen Approximationsfehler kommt. Bei kleineren Zahlen hingegen (bspw. LOD ≤ 3) werden alle Bits geschützt und Nullen als Padding hinzugefügt, damit die führende Eins mit

dem MSB ausgerichtet wird. Dadurch wird eine einheitliche logarithmische Verarbeitung der Operanden ermöglicht.

Taheri et al. [Tah+24a] beschreiben, dass für die Fehlererkennung die Ergebnisse der approximierten Multiplikation doppelt berechnet und anschließend verglichen werden. Bei einem Unterschied wird ein Fehler erkannt und die fehlerhaften Bits werden auf „0“ gesetzt. Durch dieses Verfahren können mit AdAM die Fehler in der Multiplikation erkannt und minimiert werden.

Zur Bewertung von AdAM wird ein Experiment durchgeführt, dessen Ergebnisse ausführlich im Abschnitt „Experimental Results“ dargestellt sind [Tah+24a].

In den Ergebnissen der Experimentes von Taheri et al. [Tah+24a] wird deutlich, dass das AdAM-Verfahren innerhalb des Experimentes eine Fehlertoleranz erreicht, die nahe an die der klassischen TMR-Methode herankommt. Gleichzeitig hat AdAM einen reduzierten Ressourcenbedarf gegenüber TMR und benötigt etwa 63,5% weniger Chip-Fläche. Außerdem ist das PDP etwa 39% niedriger. Dadurch ist das AdAM-Verfahren ein attraktiver Kompromiss zwischen Effizienz und Zuverlässigkeit.

9.4.3. Symptom-based Error Detectors (SED)

geschrieben von Arian Dannemann

Als Sympton-based Error Detector (SED) versteht man Fehlererkennungsmaßnahmen, die bestimmte, anwendungsspezifische Symptome verwenden, um Anomalien zu erkennen. Dabei können die erwähnten Symptome verschiedene Formen annehmen, auf die zugegriffen wurde. Beispiele sind Werte von Variablen, Anzahlen von Schleifen-Iterationen oder Adressbereiche. Um einen Error Detector einzusetzen, müssen zwei Schritte durchgeführt werden:

1. Lernen: Für jedes verwendete Symptom muss der Error Detector „lernen“, welche Werte den Normalbereich darstellen. Es muss also ein NN fehlerfrei ausgeführt werden, um zu erkennen, welche Werte in welchem Bereich liegen dürfen. Werte außerhalb des bestimmten Bereichs werden als fehlerhaft angenommen. In Abbildung 9.19 sind diese Werte für einige Netzwerke beispielhaft aufgeführt.

Network	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5
AlexNet	-691.813 662.505	-228.296 224.248	-89.051 98.62	-69.245 145.674	-36.4747 133.413
CaffeNet	-869.349 608.659	-406.859 156.569	-73.4652 88.5085	-46.3215 85.3181	-43.9878 155.383
NiN	-738.199 714.962	-401.86 1267.8	-397.651 1388.88	-1041.76 875.372	-684.957 1082.81
ConvNet	-1.45216 1.38183	-2.16061 1.71745	-1.61843 1.37389	-3.08903 4.94451	-9.24791 11.8078

Abbildung 9.19.: Ausschnitt von den erkannten Wertebereichen für verschiedene NNs [Li+17]

2. Einsetzen (aus dem Englischen übersetzt von: „deploy“): Während der Laufzeit des Systems müssen nun anhand von den Detektoren aus Schritt 1 Fehler erkannt werden. Hierfür gibt es verschiedene Methoden (auf die aus rahmen-technischen Gründen nicht näher eingegangen wird), allerdings sei erwähnt, dass zeitlicher und räumlicher Overhead bedacht

werden müssen. Ein SED kann beispielsweise auf die Ausgaben einer bestimmten Schicht an Neuronen angewandt werden und ihre Ausgaben mit den in Schritt 1 erfassten Daten vergleichen. Wenn diese Werte nicht in einem 10%igen Intervall der bekannten Daten liegen wird ein Fehler gemeldet.

Ein SED wird dabei von der CPU implementiert. Das NN muss so konzipiert sein, dass es während der Ausführung eines Layers die Ergebnisse des letzten Layers in den globalen Puffer legt, wo sie vom SED eingelesen und überprüft werden können. Dieser Prozess kann asynchron zur Ausführung des aktuellen Layers passieren, was die Geschwindigkeit erhöhen könnte.

Im Testfall von Li et al. [Li+17] wurde ein SED auf verschiedene DNNs angewendet und konnte durchschnittlich 92,5% der SDC-verursachenden Fehler erkennen.

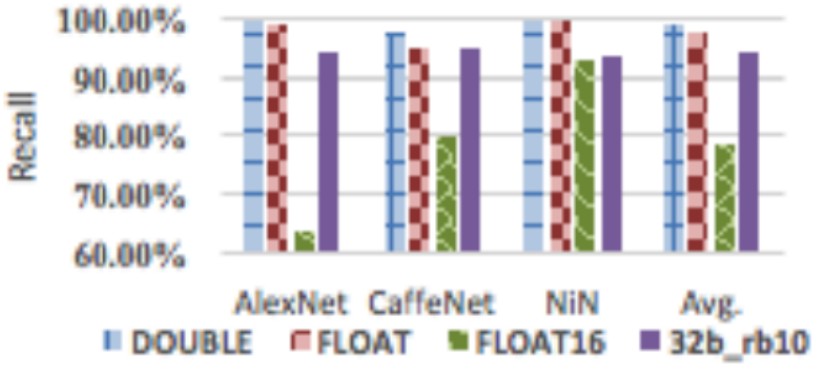


Abbildung 9.20.: Ausschnitt von der „Recall“-Genauigkeit des von Li et al. [Li+17] angewandten SED. Auf der Y-Achse sind die verschiedene NNs und ihre jeweils getesteten Datentypen aufgelistet. Auf der X-Achse ist die Recall-Genauigkeit zu sehen. Sie gibt an, wie viel Prozent der SDC-erzeugenden Fehler erkannt wurden. [Li+17]

9.4.4. Selective Latch Hardening (SLH)

geschrieben von Arian Dannemann

Selective Latch Hardening (SLH) ist eine Hardware-Fehlerminderungstechnik, welche vorsieht, redundante Schaltkreise gezielt für bestimmte Speicherelemente (latches) zu implementieren. Hintergedanke ist eine Reduktion des zusätzlich benötigten Speicherplatzes gegenüber TMR, welches besonders für komplexere neuronale Netze den vorhandenen Platz auf einem FPGA eventuell zu sehr ausreizt.

In Unterabschnitt 9.2.2 wurden bestimmte Bits innerhalb verwendeter Datentypen identifiziert, die im Falle eines Bit-Flips eine deutlich höhere Auswirkung auf das Ergebnis haben als andere Bits. Indem diese Bits das Ziel für den Härtungsprozess darstellen, kann mit deutlich reduziertem Platzaufwand (im Vergleich zu TMR) bereits die SDC-Rate verringert werden. Wenn einzelne Bits einen hohen Einfluss auf die Genauigkeit des Ergebnisses haben, dann müssen unter Umständen nur diese Bits gehärtet werden, um eine deutliche Reduktion der FIT-Rate zu erreichen.

Li et al. [Li+17] untersuchen die folgenden Härtungsmaßnahmen für bestimmte, fehlerempfindliche Bits:

- Strike Suppression (Reinforcing Charge Collection (RCC))
- Redundant Node (Single Event Upset Tolerant (SEUT))

- Triple Modular Redundancy (TMR)

Weniger im Fokus standen hierbei die einzelnen Härtungsmaßnahmen an sich, sondern vielmehr die Effektivität, die durch das Anwenden der Härtungsmaßnahmen auf lediglich bestimmte Bits erreicht werden kann. Eine Erklärung von RCC und SEUT kann unter [Sei+10] nachgelesen werden.

In einem untersuchten Fall konnten alle drei Härtungstechniken angewandt werden, um die FIT-Rate um ein hundertfaches zu reduzieren. Der zusätzlich benötigte Platz stieg lediglich um 20%, im Vergleich zu dem 30% höheren Platzaufwand die TMR bei (geringerer) 6.3-facher FIT-Reduktion liefert.

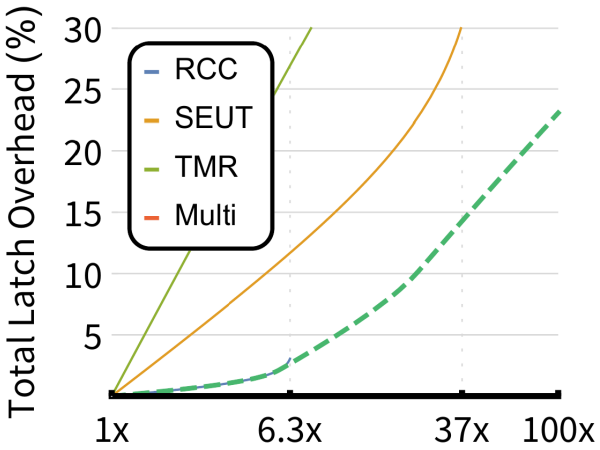


Abbildung 9.21.: Ausschnitt des Diagramms aus [Li+17]. Besonders zu beachten ist hier der Verlauf der Funktionen „TMR“ (in der Farbe „strong yellow green“) im Vergleich zum Verlauf der Funktion „Multi“ (in der Farbe „brilliant green“). Er zeigt, dass die Target Latch FIT Reduction (X-Achse) im Falle einer Kombination aller drei Härtungstechnicken („Multi“) bei einer 6.3-fachen Reduktion der FIT-Rate mit zirka 2.5% Overhead sehr viel schlanker ist, als die Methode TMR mit bereits 30% Overhead im gleichen Szenario. Die Namen der Farben wurden der Universal Color Language (ISCC-NBS System) (UCL) entnommen.

9.4.5. FORTUNE: Fault TOleRance TechniqUe

geschrieben von Kevin Lingk

Bei Fortune handelt es sich um eine hardware-agnostische Methode zur Erhöhung der Fehlertoleranz in DNNs. Laut Nazari et al. [Naz+24] verwendet Fortune dabei die Quantisierung, um durch die Speichereinsparung gezielt das MSB zu schützen, da dieses einen kritischen Teil in DNNs darstellt. Dadurch kombiniert Fortune sowohl die Reduzierung des Speicherbedarfs, erhöht aber gleichzeitig die Robustheit gegenüber Fehler. Die durch die Quantisierung frei gewordenen Speicherressourcen werden genutzt, um das MSB innerhalb desselben Speicherelements redundant abzulegen. Konkret wird das MSB mithilfe von TMR repliziert, also zweimal redundant gespeichert. Dadurch entsteht negativer Speicheraufwand, da ein Modell weniger Speicher als das ursprüngliche Modell benötigt und ist gleichzeitig fehlertoleranter ist. Nazari et al. [Naz+24] schreiben hierzu, dass die Zuverlässigkeit des QNN auch in der Gegenwart von Speicherfehlern sichergestellt werden kann.

Dadurch adressiert Fortune ein zentrales Problem in der Anwendung von neuronalen Netzwerken in sicherheitskritischen Bereichen. Es soll eine Kombination aus Effizienz und Zuverlässigkeit ermöglicht werden.

Zur Funktionsweise von Fortune schreiben Nazari et al. [Naz+24], dass es in den folgenden drei Schritten abläuft:

1. Quantisierung
2. Zuverlässigkeitsbewertung und -verbesserung
3. Speicherverbrauch und Ausführungsdauer

Im Schritt eins, der Quantisierung, werden die Gewichte des zur Verfügung gestellten Modells durch lineare Quantisierung auf die gewählte Bitbreite reduziert. Zur Bewertung der Zuverlässigkeit werden anschließend gezielt Bitfehler in die quantisierten Gewichte injiziert und anschließend die Genauigkeit des Modells bestimmt. Diese wird mit der Genauigkeit eines Referenzmodells verglichen, um so zu prüfen, ob die kritischen Bits unter Fehlereinwirkung zuverlässig bleiben. Zum Schluss werden Speicherbedarf und Laufzeit des Modells berechnet, um ein ausgewogenes Verhältnis zwischen Effizienz und Zuverlässigkeit sicherzustellen. [Naz+24]

Die technischen Details werden von Nazari et al. [Naz+24] beschrieben und der Quellcode ist zudem als Open-Source-Projekt verfügbar[2].

In Abbildung 9.22 ist die Funktionsweise von Fortune exemplarisch dargestellt. Auf der linken Seite ist der Ausgangszustand zu sehen, der die ursprünglichen Gewichtungen eines neuronalen Netzes zeigt. Diese Gewichtungen sind als 32-Bit-Floating-Point-Zahlen dargestellt, wobei jede Schicht ein einzelnes 32-Bit-Gewicht repräsentiert.

In der Mitte befinden sich die Gewichte nach der Quantisierung. Die 32-Bit Gewichte wurden auf 3-Bit Integer quantisiert, wodurch sie deutlich weniger Speicherbedarf haben. Der Nachteil dadurch ist allerdings, dass diese eine höhere Anfälligkeit gegenüber SEUs und MBUs aufweisen, wie es in Unterabschnitt 9.2.2 bereits beschrieben wurde.

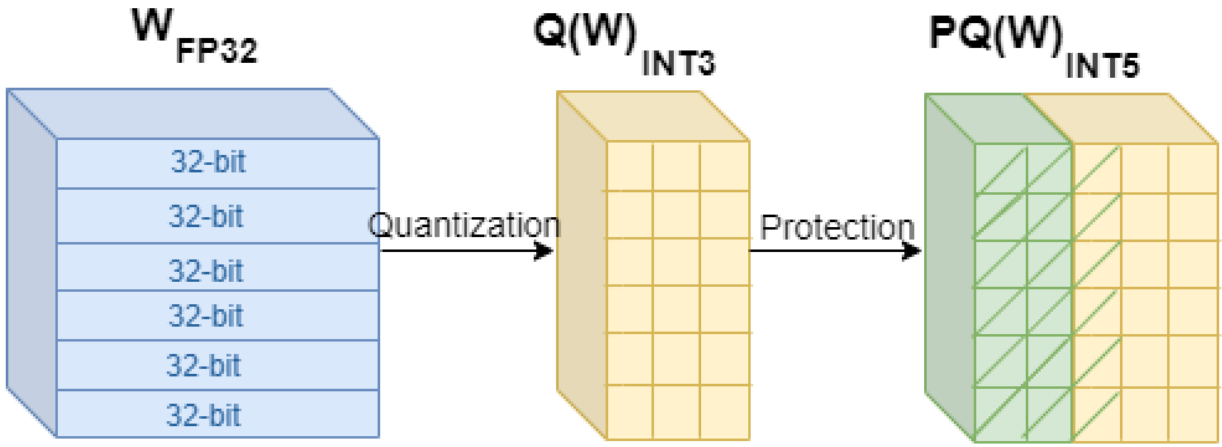


Abbildung 9.22.: Beispiel für fehlertolerante 3-Bit-Gewichte

Auf der rechten Seite befindet dann das quantisierte Gewicht, welches durch TMR geschützt wird. Für den Schutz werden durch Fortune zwei weitere Bits benötigt, wodurch das Gewicht auf

[2] https://github.com/nilay1400/DNN-Quantization

5-Bit erhöht wird. Entscheidend ist dabei allerdings, dass dieser Wert immer noch weit unter den ursprünglichen 32-Bit liegt. Die zwei weiteren Bits werden für die den MSB-Schutz verwendet. Erkennbar werden die redundanten Kopien des MSB in dem grün schraffierten Bereich. Der gelbe Bereich stellt die ursprünglich quantisierten 3-Bit Gewichte dar. Das MSB wird insgesamt dreimal gespeichert (2x grün und 1x gelb).

Dadurch repliziert Fortune also nur gezielt das wichtigste Bit anstelle aller Bits wie es TMR macht, wodurch eine erhebliche Speichereinsparung bei gleichzeitig gesteigerter Fehlertoleranz möglich wird.

9.5. Evaluationsmethoden zur Fehlertoleranz von neuronalen Netzen

geschrieben von Kevin Lingk

Die Zuverlässigkeitsbewertung von DNNs ist ein zentraler Bestandteil, insbesondere bei sicherheitskritischen Anwendungen wie das autonome Fahren oder dem Gesundheitswesen. Um die Fehlertoleranz von neuronalen Netzen zu überprüfen und zu quantifizieren, stehen verschiedene Evaluationsmethoden zur Verfügung, die eingesetzt werden können.

Taheri [Tah25] unterteilt die Zuverlässigkeitsbewertung von DNNs in drei Hauptkategorien, die teilweise selbst nochmals in Unterkategorien gegliedert werden können:

1. Fault Injection
2. Analytische Methoden
3. Hybride Methoden

9.5.1. Fault Injection

geschrieben von Kevin Lingk

Fault Injection (FI) beschreibt die Technik zur gezielten Einbringung von Fehlern in ein neuronales Netz, um dessen Verhalten unter Störbedingungen zu analysieren und die Robustheit zu bewerten. Dadurch sollen die Auswirkungen von temporären oder permanenten Fehlern systematisch untersucht werden. [Ruo+23]

Laut Taheri [Tah25] kann die Fault Injection in drei weitere Unterkategorien unterteilt werden:

- Fault Simulation (FS)
- Fault Emulation (FE)
- Irradiation

Dabei weist Taheri [Tah25] auch darauf hin, dass ein Großteil der Arbeiten zur Zuverlässigkeitsbewertung von DNNs auf Methoden der Fault Injektion zurückgreift. Während bei der Fault Simulation (FS) DNNs in Software oder mittels Hardware Description Languages (HDLs) modelliert und dort Fehler injiziert werden, erfolgt die Ausführung bei der Fault Emulation (FE) auf echter Hardware wie FPGAs oder GPUs, wobei die Fehlerinjektion gezielt vorgenommen wird. Die Irradiation nutzt reale Strahlenbelastung auf physischer Hardware zur Fehlersimulation.

Fault Simulation (FS)

geschrieben von Kevin Lingk

Ein bedeutender Ansatz innerhalb der Fault Injection ist die bereits erwähnte Fault Simulation. Bei der Fault Simulation werden künstlich erzeugte Fehler in Modelle (DNNs) eingebracht, um somit das Verhalten des DNNs unter fehlerhaften Bedingungen zu betrachten. Taheri [Tah25] unterscheidet dabei zwischen hardwareunabhängigen, hardwarebewusste und Register-Transfer Level (RTL)-basierten Simulationen.

Die praktische Umsetzung dieser Ansätze erfolgt durch spezialisierte Frameworks wie SAFFIRA für Systolic-Array-basierte DNN-Beschleuniger und DeepAxe für FPGA-basierte Design Space Exploration, die im Folgenden vorgestellt werden.

SAFFIRA

geschrieben von Kevin Lingk

Mit SAFFIRA stellen Taheri et al. [Tah+24b] einen neuen Ansatz zur Zuverlässigkeitsbewertung von Systolic-Array-basierten DNN-Beschleunigern vor. Dieser Ansatz soll dabei eine softwarebasierte und hardwarebewusste Fault Injection Methode sein, um den zeitaufwändigen Prozess von traditionellen Fault Injection (FI)-Methoden zu optimieren, um dadurch den Aufwand zu reduzieren ohne einen Kompromiss bei der Genauigkeit der Bewertung eingehen zu müssen.

Laut Taheri et al. [Tah+24b] muss die Hardware, auf der die Berechnung ausgeführt wird, zusätzlich bei der Zuverlässigkeitsbewertung von DNNs betrachtet werden. Der Grund dafür ist, dass die eingesetzte Hardware entsprechend hardwarespezifische Fehler aufweisen kann. Da Systolic Arrays eine bedeutende Architektur für DNN-Hardware-Beschleuniger darstellen, soll durch das OpenSource-Tool SAFFIRA, welches ausgeschrieben für „*Systolic Array simulator Framework for Fault Injection based Reliability Assessment*“ steht, eine vollautomatisierte Toolchain zur effizienten Zuverlässigkeitsbewertung bereitgestellt werden. Dabei soll SAFFIRA genauer sein als hardware-agnostische Tools und gleichzeitig deutlich schneller als RTL-basierte Simulationen arbeiten.

Zu der Funktionsweise von SAFFIRA schreiben Taheri et al. [Tah+24b], dass dem Framework ein bereits trainiertes neurales Netzwerk bereitgestellt werden kann. Der Ablauf danach erfolgt nach Taheri et al. [Tah+24b] in einem dreistufigen Prozess:

1. Generierung einer Fehlerliste
2. Durchführung der Fehlerinjektion
3. Bewertung der Netzwerkzuverlässigkeit

Bei der Erstellung der Fehlerliste im ersten Schritt kann der Nutzer selbst festlegen, welche potenziellen Fehlerstellen im DNN abgedeckt werden sollen, beispielsweise ob transiente oder permanente Fehler simuliert werden und welche Aktivierungen betroffen sind. Darüber hinaus bietet das SAFFIRA-Framework die Möglichkeit, die Fehlerliste automatisch zu generieren (siehe Abbildung 9.23 Nr. 1).

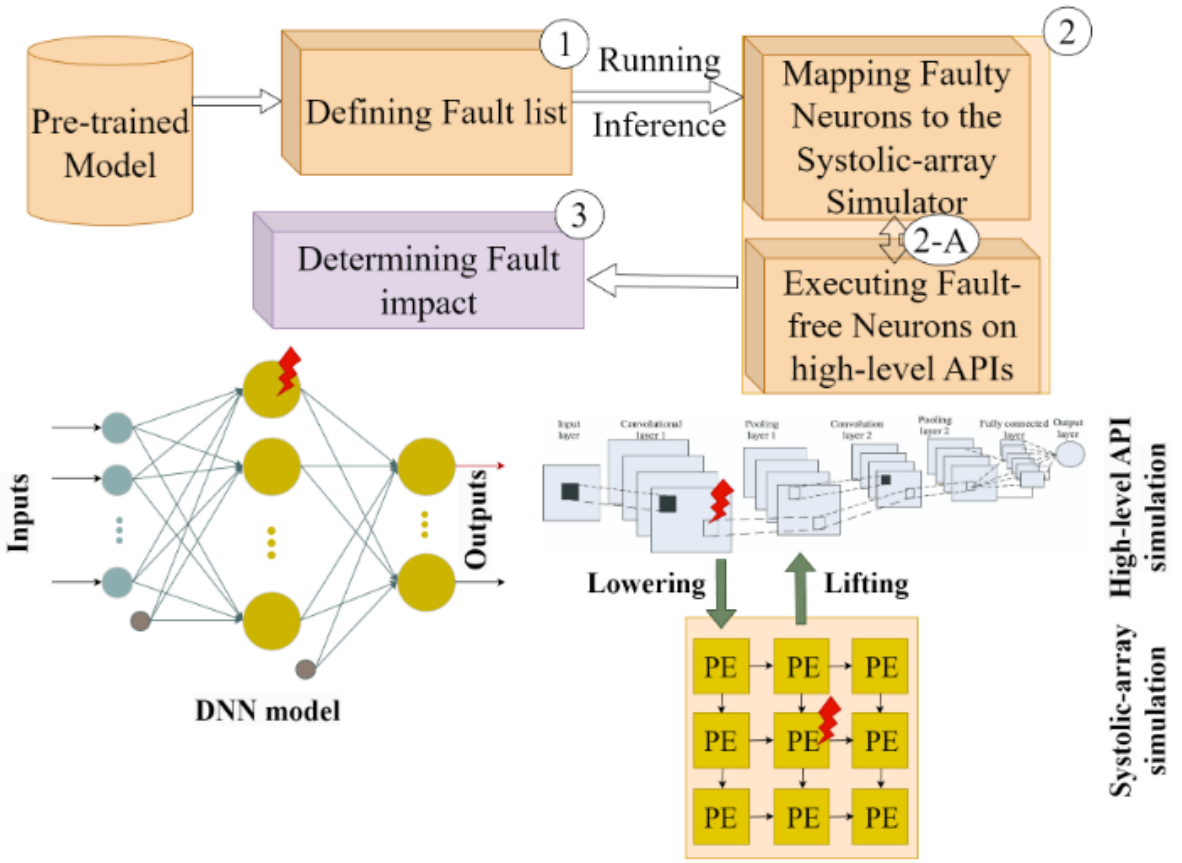


Abbildung 9.23.: Funktionsweise von SAFFIRA [Tah+24b]

Im zweiten Schritt erfolgt die eigentliche Durchführung der Fehlerinjektion basierend auf der zuvor erstellten Fehlerliste. Taheri et al. [Tah+24b] beschreiben, dass SAFFIRA dabei die Fehler mit einer Simulationsumgebung, welche in Python läuft, in das Systolic Array einbringt (siehe Abbildung 9.23 Nr. 2). Damit anschließend die Verarbeitung der anderen Netzwerkoperationen beschleunigt werden kann, werden diese beispielsweise mit PyTorch als High-Level-API durchgeführt (9.23 Nr. 2A). Zum Wechsel zwischen der Simulationsumgebung und der High-Level-API beschreiben Taheri et al. [Tah+24b] eine Methode mit dem Namen „LoLif“, die sich zusammensetzt aus „Lowering“ und „Lifting“. Die Methode ermöglicht es, Faltungsoperationen auf dem Systolic Array in Form von Matrixoperationen durchzuführen (Lowering) und die resultierenden Daten anschließend wieder in das ursprüngliche Format zu überführen (Lifting). Erkennbar wird dies im unteren Teil der Abbildung 9.23. Erläutert wird diese Methode genauer im Abschnitt „Hardware Simulation“ von Taheri et al. [Tah+24b].

Im dritten und abschließenden Schritt findet die Bewertung der Netzwerkzuverlässigkeit statt (siehe Abbildung 9.23 Nr.3). Hierbei werden laut Taheri et al. [Tah+24b] die Ausgaben des fehlerbetroffenen neuronalen Netzes mit den Referenzausgaben des fehlerfreien neuronalen Netz (auch „Golden Model genannt“) verglichen. Für die Bewertung führen Taheri et al. [Tah+24b] eine neue Metrik mit dem Namen „Faulty Distance“ ein, bei der die Klassifikationswahrscheinlichkeiten der beiden Modellausgaben miteinander verglichen werden. Dabei werden die generierten Wahrscheinlichkeiten für alle Ausgabeklassen, sowohl des fehlerfreien als auch des fehlerbehafteten Modells betrachtet. Ein Wert von 0 bedeutet korrekte Klassifikation, höhere Werte deuten auf schwerwiegendere Fehlklassifikationen hin. Taheri et al. [Tah+24b] erläutern diese Metrik im Abschnitt „Fault Classification“ im Detail.

Die Ergebnisse von Taheri et al. [Tah+24b] zeigen, dass SAFFIRA in der experimentellen Umgebung eine deutliche Effizienzsteigerung gegenüber bestehenden Fault Injection Methoden erreicht. Nach Taheri et al. [Tah+24b] erzielt SAFFIRA im Experiment eine rund dreifache Geschwindigkeitssteigerung im Vergleich zu aktuellen hybriden Hardware-Software-Fault-Injection-Frameworks. Gegenüber RTL-basierten Simulationen konnte eine über 2000-fache Zeitersparnis erreicht werden, ohne die Genauigkeit der Bewertung zu beeinträchtigen.

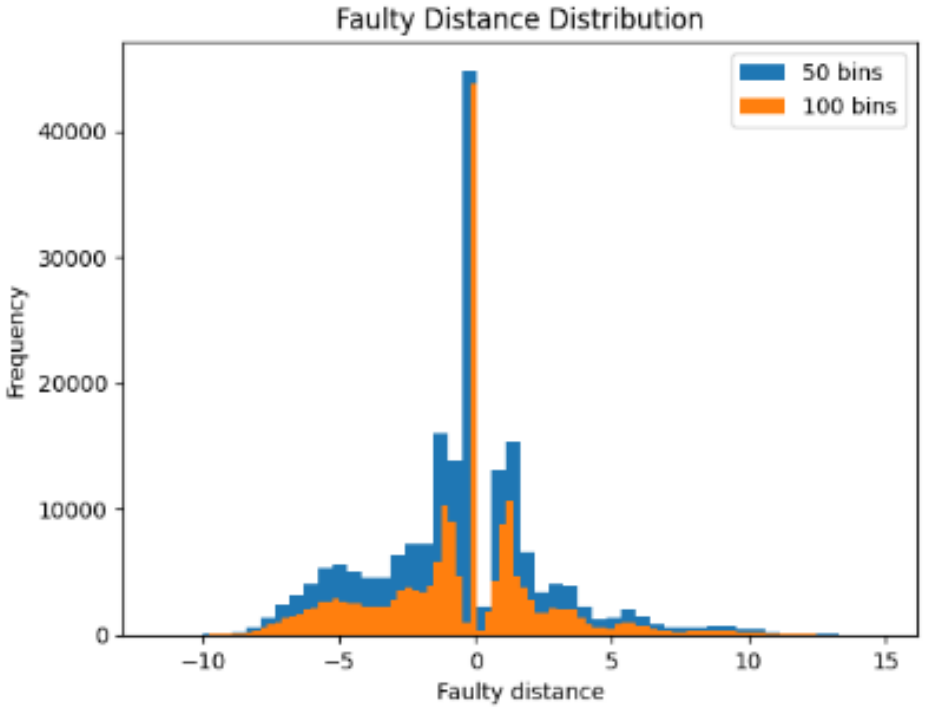


Abbildung 9.24.: 8-bit Netzwerk – Faulty Distance – Verteilung

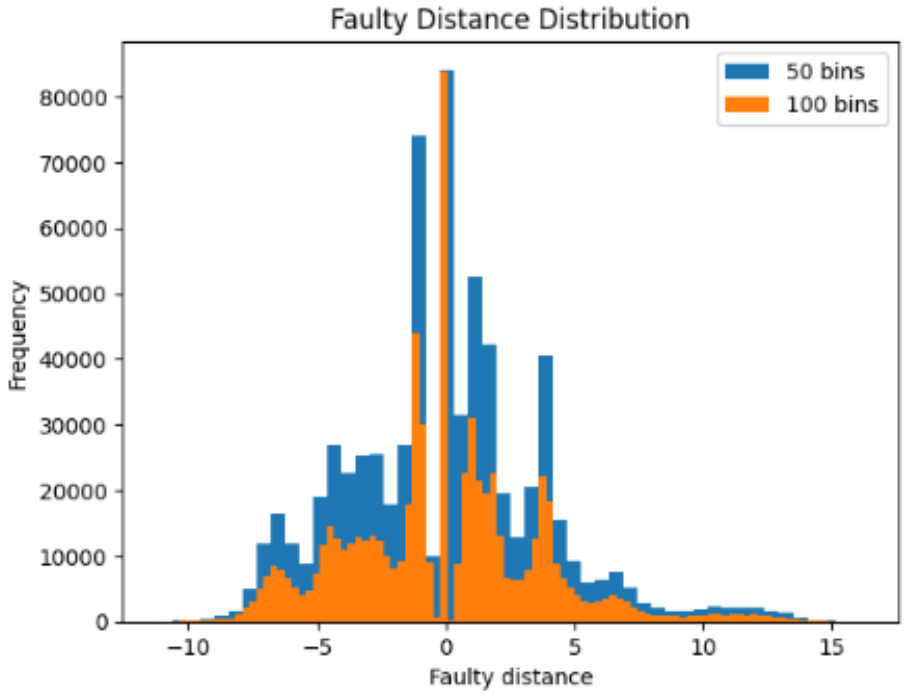


Abbildung 9.25.: 16-bit Netzwerk – Faulty Distance – Verteilung

Bezüglich der neu eingeführten Faulty Distance-Metrik stellen Taheri et al. [Tah+24b] fest, dass diese eine wertvolle Grundlage für die detaillierte Analyse des Netzwerkverhaltens unter Fehlereinfluss bietet. In den Abbildungen 9.24 und 9.25 wird deutlich, dass die histografische Darstellung einen Aufschluss über die Art und Schwere von Fehlklassifikationen gibt. Darüber hinaus verdeutlichen die Ergebnisse, dass verschiedene Quantisierungsstufen zu unterschiedlichen Verteilungsmustern führen. Insbesondere weist das 16-Bit-Netzwerk bei Fehlerinjektion eine höhere Anfälligkeit für Fehlklassifikationen auf als das 8-Bit-Netzwerk.

DeepAxe

geschrieben von Kevin Lingk

Als weiteren Ansatz neben SAFFIRA stellen Taheri et al. [Tah+23a] das Framework „DeepAxe“ vor, welches für die Design Space Exploration von FPGA-basierten DNN-Implementierungen verwendet wird. Die „Design Space Exploration“ bezeichnet einen systematischen Prozess zur Bewertung verschiedener Entwurfsalternativen und deren Trade-offs, mit dem Ziel, Pareto-optimale Konfigurationen zu identifizieren, die sowohl Approximation als auch Fehlertoleranz berücksichtigen. Bei DeepAxe stehen dabei insbesondere die drei zentralen Entwurfsalternativen „Genauigkeit“, „Zuverlässigkeit“ und „Hardware-Performance“ im Fokus. Mit der Pareto-optimalen Konfiguration wird dabei beschrieben, dass keiner der genannten Parameter verbessert werden kann, ohne einen der anderen zu verschlechtern. [Tah+23a]

Dabei wird vorwiegend der Einfluss von Approximation und Fehlerinjektion auf die oben genannten Aspekte untersucht. Nach Taheri et al. [Tah+23a] ist DeepAxe das erste Framework, das sowohl die Widerstandsfähigkeit gegenüber temporären Fehlern als auch die Leistung der Hardware als zentrale Entwurfsfaktoren bei DNN-Beschleunigern berücksichtigt.

Im Unterschied zu SAFFIRA, das sich in erster Linie mit der Simulation von Fehlern und deren Auswirkungen auf neuronale Netze beschäftigt, kombiniert DeepAxe die Bewertung der Zuverlässigkeit mit Ansätzen der approximierten Berechnung. Nach Taheri et al. [Tah+23a] sollen dadurch gezielt Recheneinheiten im Netzwerk vereinfacht werden, damit Ressourcen gespart werden können, ohne die anderen drei genannten zentralen Aspekte zu beeinträchtigen. DeepAxe ist dabei in die High Level Synthesis (HLS)-Umgebung DeepHLS integriert und analysiert systematisch, welchen Einfluss die Vereinfachungen auf die Genauigkeit, Fehlertoleranz und Hardware-Performance haben, um anschließend eine HLS-basierte FPGA-Implementierung zu generieren. Riazati et al. [Ria+20] beschreiben DeepHLS als eine vollständige Toolchain, die speziell entwickelt wurde, um die automatische Synthese von DNNs auf FPGAs zu ermöglichen. Dabei können DNNs, die in abstrakten Frameworks wie Keras entwickelt wurden, in eine einfache, flache und synthesefähige C-Darstellung umgewandelt werden. Durch diese Umwandlung können HLS-Tools optimal genutzt werden, um eine effiziente FPGA-Implementierung zu erzeugen.

Zur Funktionsweise von DeepAxe schreiben Taheri et al. [Tah+23a], dass dem Framework ein bereits trainiertes Keras-Modell bereitgestellt werden kann. Der Ablauf erfolgt nach Taheri et al. [Tah+23a] in einem fünfstufigen Prozess:

1. Bereitstellung eines bereits trainierten Keras-Modells
2. Vorverarbeitung und Quantisierung
3. Festlegen der Approximations-Konfiguration
4. Fehlerinjektion mittels Fault Simulation, sowie Zuverlässigkeitsanalyse
5. HLS-Implementierung

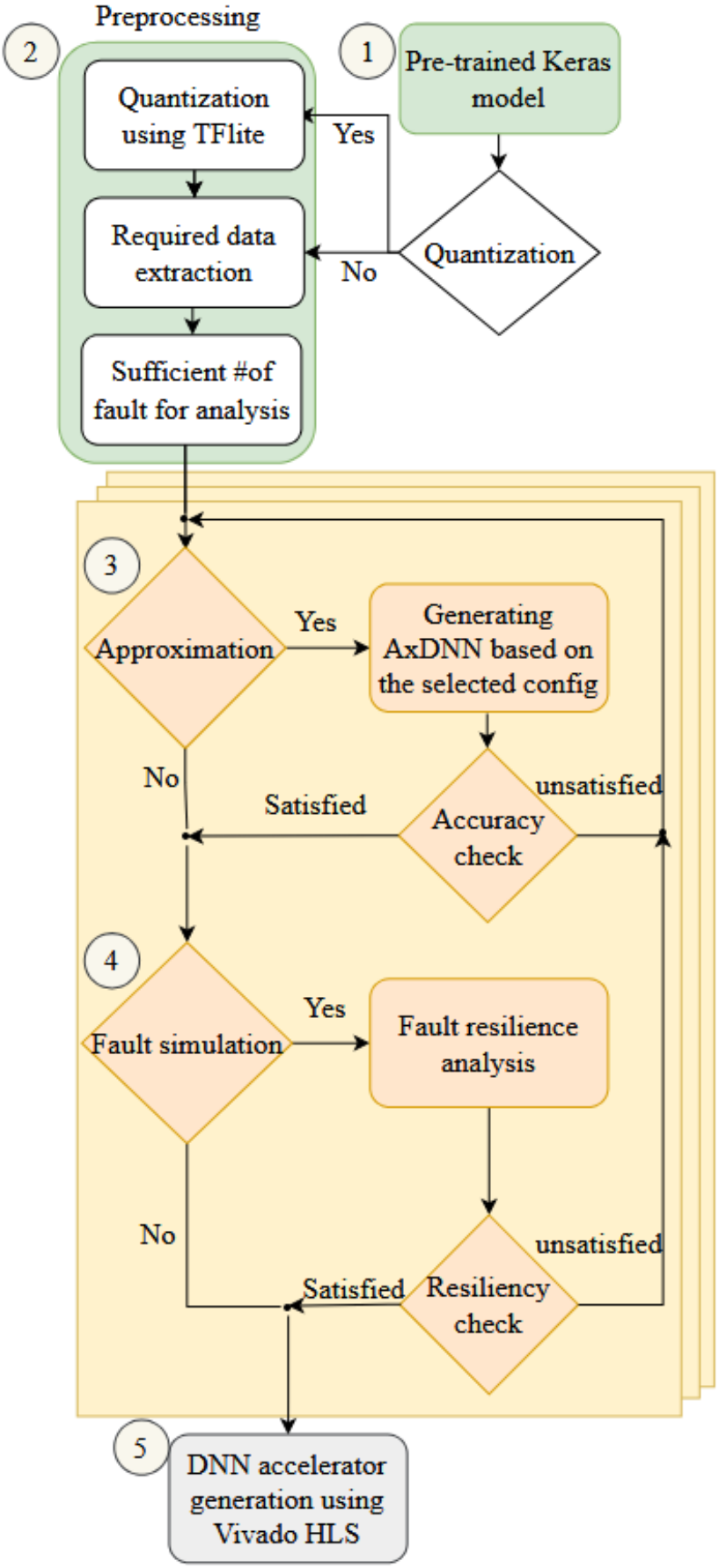


Abbildung 9.26.: DeepAxe – Flowchart [Tah25]

Im ersten Schritt erfolgt das Laden des bereits trainierten Keras-Modells in das DeepAxe-Framework (siehe Abbildung 9.26 Schritt 1). Des Weiteren kann die Konfiguration vorgenommen werden, dass eine Quantisierung stattfinden soll, falls diese benötigt wird.

Der zweite Schritt umfasst dann die Durchführung der Quantisierung, falls diese aktiviert wurde. Dabei werden alle Aktivierungsfunktionen, Gewichte und Verzerrungen anpasst. Taheri [Tah25, S. 54] beschreibt, dass eine Quantisierung bis auf 8-Bit möglich ist. Zusätzlich wird in der Vorverarbeitung eine ausreichende Anzahl von Fehlern für die spätere Zuverlässigkeitsanalyse, basierend auf der Neutronenanzahl, bestimmt (siehe Abbildung 9.26 Schritt 2) [Tah+23a].

Die Schritte drei und vier finden innerhalb eines iterativen Prozesses statt, durch den sowohl Genauigkeit als auch Zuverlässigkeit schrittweise optimiert werden können [Tah+23a]. Der dritte Schritt umfasst die Konfiguration der Approximation, falls diese eingesetzt werden soll. Auf Basis dieser Konfiguration wird anschließend ein approximiertes DNN erzeugt, wie Taheri [Tah25, S.55] näher beschreibt. Anschließend wird eine Genauigkeitsprüfung durchgeführt, die bei unzureichenden Ergebnissen eine Anpassung der Approximations-Konfiguration ermöglicht (siehe Abbildung 9.26 Schritt 3).

Im vierten Schritt (siehe Abbildung 9.26 Schritt 4) kann dann optional eine Fehlerinjektion zur Bewertung der Zuverlässigkeit durchgeführt werden. Nach Taheri et al. [Tah+23a] umfasst dieser Schritt außerdem eine Zuverlässigkeitsbewertung des generierten neuronalen Netzes. Falls die Ergebnisse nicht wie gewünscht ausfallen, kann in den Schritten zurückgegangen werden, damit die Konfigurationen überarbeitet werden können.

Zum Schluss wird das generierte und angepasste neuronale Netz an die HLS-Implementierung übergeben, wodurch eine optimierte Konfiguration für DNN-Beschleuniger erstellt wird [Tah+23a].

In den Ergebnissen von Taheri et al. [Tah+23a] wird deutlich, dass DeepAxe erfolgreich die Analyse von Genauigkeit, Zuverlässigkeit und Hardware-Performance bei approximierten DNN-Implementierungen mit angewandter Fehlerinjektion ermöglicht. Für die Fehlerinjektion findet ein „Single Bit-Flip“ in einem zufällig ausgewählten Neuron in einem zufällig ausgewählten Layer statt, wobei dieser Prozess häufiger wiederholt wird, um eine repräsentative Stichprobe zu erhalten. Zur Validierung des Frameworks wurde dieses repräsentativ an drei neuronalen Netzwerken getestet. Taheri et al. [Tah+23a] beschreiben, dass 600, 800 und 1000 zufällige Single Bit-Flip Fehlerinjektionen durchgeführt wurden. Die experimentellen Ergebnisse werden im Abschnitt „Experimental Results“ ausführlich dargestellt [Tah+23a, S. 4–7].

Taheri et al. [Tah+23a] stellen mit der Pareto-Analyse in Abbildung 9.27 exemplarisch für eines der drei neuronalen Netzwerke (LeNet-5) den Trade-off zwischen Ressourcennutzung und Genauigkeitsverlust bei verschiedenen Approximations-Konfigurationen unter Anwendung von Fehlerinjektion dar. Die violetten Punkte stellen die verschiedenen getesteten Approximations-Konfigurationen dar. Die linke obere Ecke zeigt Konfigurationen mit hoher Genauigkeit, aber höherem Ressourcenverbrauch, während die rechte untere Ecke Konfigurationen mit geringerer Ressourcennutzung, aber höherem Genauigkeitsverlust darstellt. Die orange Linie stellt die Pareto-Grenze dar und markiert die optimalen Trade-off-Punkte, wobei alle Punkte darüber schlechter sind. Der Grund dafür ist, dass diese Approximations-Konfigurationen mehr Ressourcen für eine vergleichbare Genauigkeit benötigen. Es wird deutlich, dass durch gezielte Approximation signifikante Einsparungen bei der Ressourcennutzung möglich sind und Pareto-optimale Konfigurationen gefunden werden können.

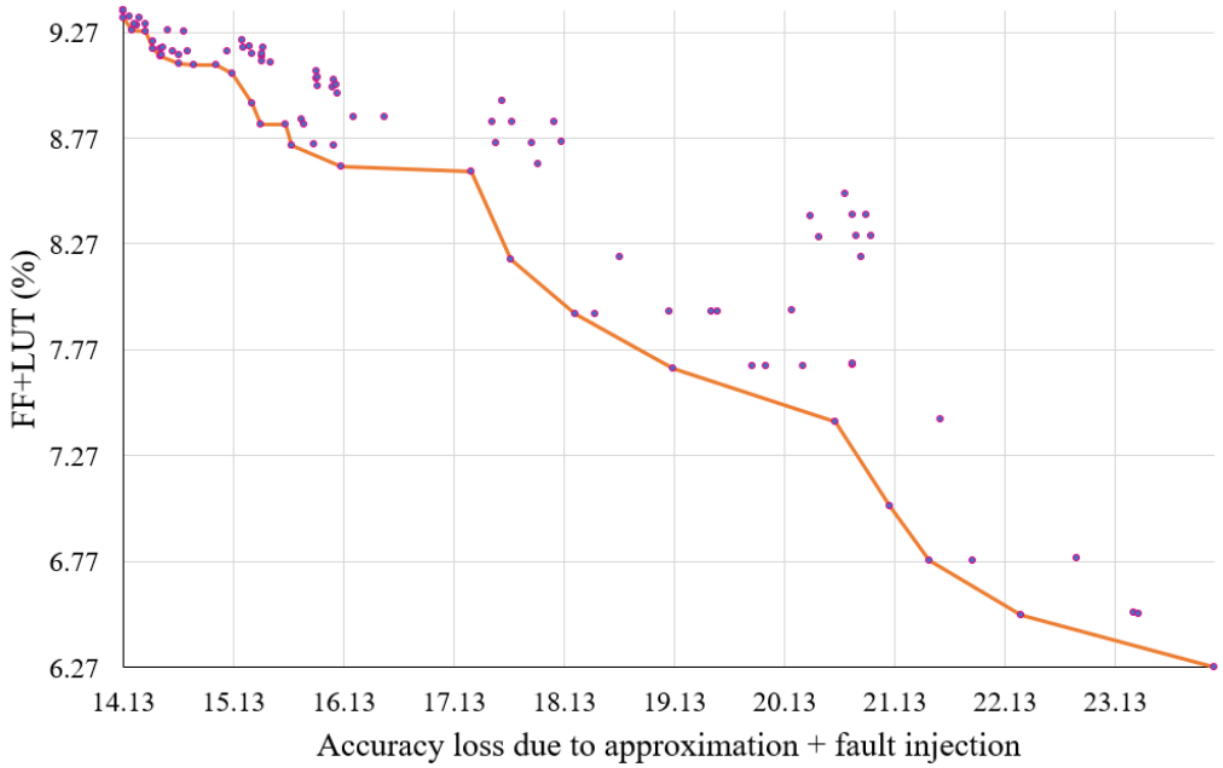


Abbildung 9.27.: Genauigkeitsverlust durch Approximation + Fehlerinjektion. Die X-Achse beschreibt den Genauigkeitsverlust in Prozent, während die Y-Achse die Ressourcennutzung (FF + LUT) in Prozent darstellt. Die orange Linie zeigt die Pareto-Grenze der optimalen Trade-off-Punkte [Tah+23a]

Fault Emulation (FE)

(geschrieben von Kevin Lingk)

Die FE ist eine Methode, bei der neuronale Netze, die auf der echten Hardware wie FPGAs oder GPUs ausgeführt werden, gezielt mit Fehlern injiziert werden, um somit die Fehlertoleranz zu bewerten. Durch diese Vorgehensweise werden realistischere Ergebnisse als bei rein softwarebasierten Simulationen erzielt, allerdings sorgt sie auch für einen hohen Zeit- und Ressourcenaufwand (vgl. [Tah25, S. 23]). Nach Taheri et al. [Tah+23b] ist der Grund dafür, dass jeder einzelne Fehler iterativ eingespeist werden muss, wodurch dies zu zahlreichen Speicherzugriffen führt, die für eine Pipeline-Unterbrechung sorgen. Des Weiteren setzt dies eine komplexe Steuer- und Kontrollarchitektur voraus, bestehend aus einem FI-Controller und zugehörigen Verbindungsstrukturen (siehe Abbildung 9.28 Teil A).

APPRAISER: DNN Fault Resilience Analysis Employing Approximation Errors

Taheri et al. [Tah+23b] beschreiben mit APPRAISER einen innovativen Ansatz im Bereich der Fault Emulation. Das Ziel dieses Verfahrens ist es, den hohen Zeit- und Ressourcenaufwand von traditionellen FI-Methoden deutlich zu reduzieren, in dem statt echter Fault Injection der gezielte Einsatz von Approximationsfehlern genutzt werden soll. Dadurch soll eine deutlich effizientere Analyse der Fehlerresistenz von DNNs erreicht werden.

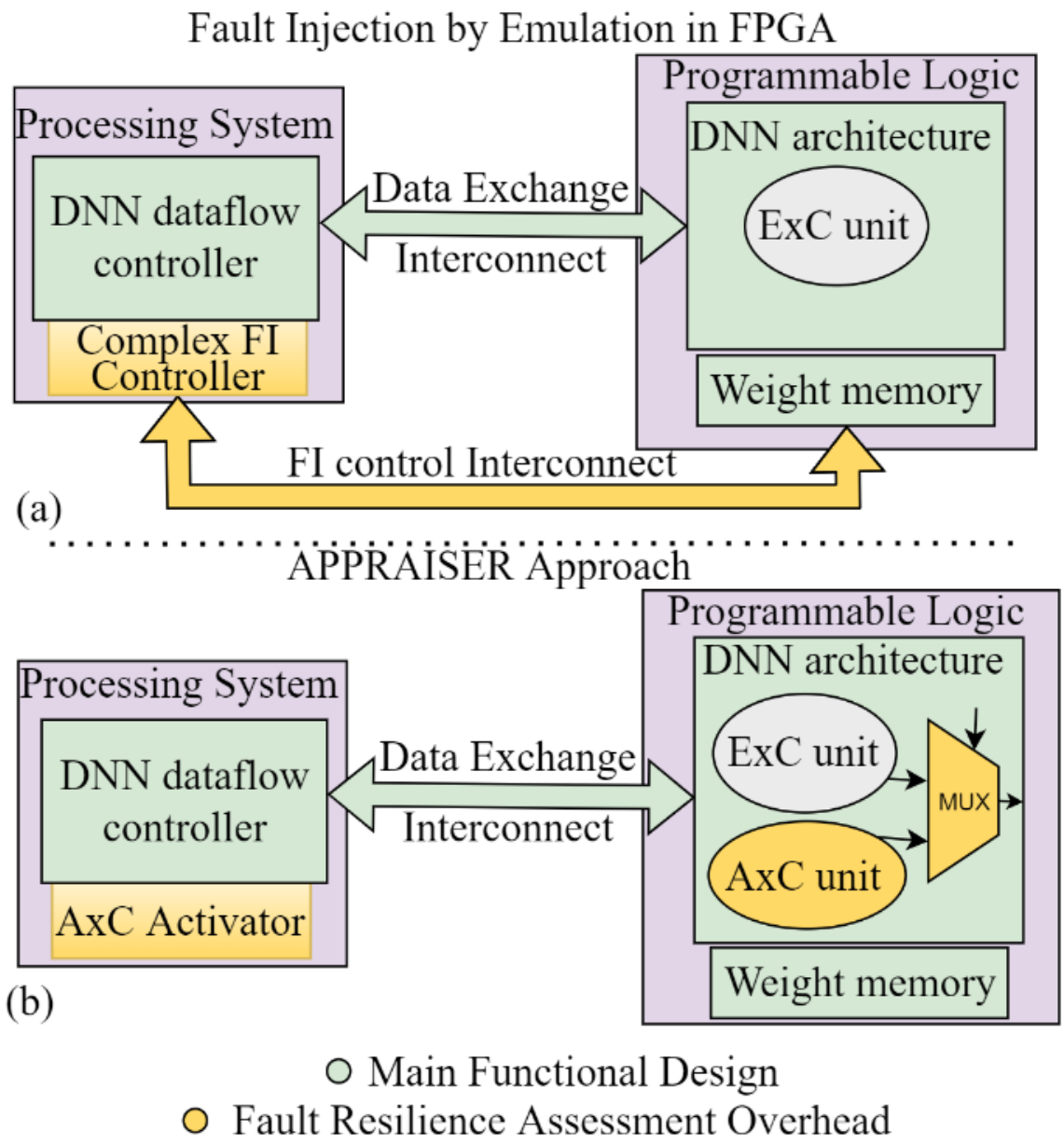


Abbildung 9.28.: Methoden zur Bewertung der Fehlerresilienz neuronaler Netze: (a) Fehlerinjektion durch Emulation im FPGA; (b) APPRAISER-Ansatz mit Fehlererzeugung durch approximative Recheneinheiten (AxC). [Tah+23b]

Laut Taheri et al. [Tah+23b] soll APPRAISER eine deutlich einfachere und effizientere Durchführung von Analysen der Fehlerresistenz von DNNs im Vergleich zur klassischen Fault Injection ermöglichen. Dadurch wird beispielsweise kein zusätzlicher FI-Controller für die Fault Injection mehr benötigt, sondern ein Approximate Computing (AxC)-Activator kommt zum Einsatz (siehe Abbildung 9.28 Teil B). Diese Einheit sorgt dafür, dass keine Pipeline-Unterbrechungen mehr notwendig sind, sondern die Verarbeitung kann während der Einspeisung der Approximationsfehler weiterlaufen. Dadurch wird ebenfalls ermöglicht, dass für das Auslesen der Gewichte keine separate Verbindung mehr benötigt wird, wodurch die Analyse vereinfacht wird.

Taheri et al. [Tah+23b] beschreiben, dass bestimmte Recheneinheiten (z.B. Multiplikatoren) in DNN-Schichten durch AxCs-Einheiten ersetzt werden, die gezielt Approximationsfehler erzeugen und so typische Hardwarefehler darstellen. Damit der Approximationsfehler verwendet werden

kann, wird durch den AxC-Activator zwischen der exakten und approximierten Multiplikation hin- und hergeschaltet. Erkennbar wird der Ablauf in Abbildung 9.29.

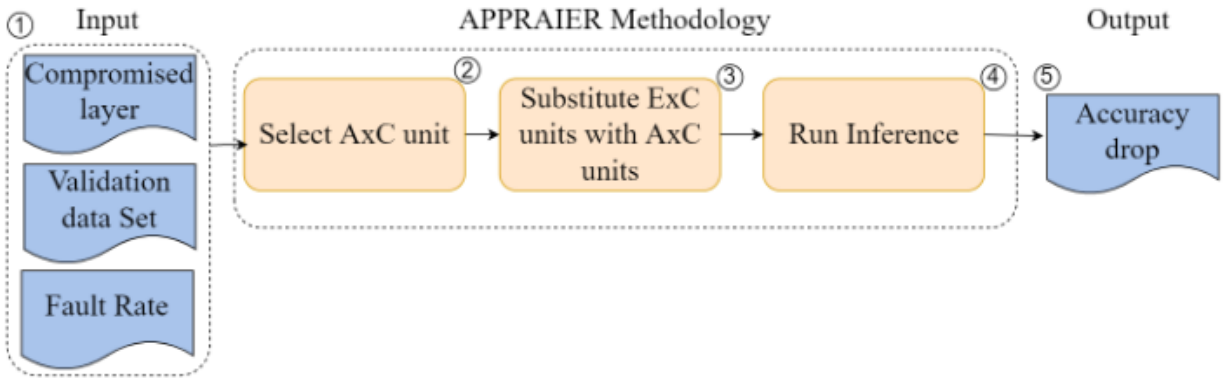


Abbildung 9.29.: APPRAISER-Methodik [Tah+23b]

Nach Taheri et al. [Tah+23b] wird dabei der Accuarcy- und Recall-Abfall als Metrik für die Fehlerresistenz von DNNs genommen. Zum Vergleich, ob Bit-Flips in späteren Layern aufgetreten sind, werden die Outputs mit einem sogenannten „Golden Model“ verglichen. Das „Golden Modell“ ist ein fehlerfreies Referenzmodell, das als Grundlage zur Erkennung von Abweichungen dient.

Die Ergebnisse von Taheri et al. [Tah+23b] in Abbildung 9.30 (grüner Kasten) zeigen, dass APPRAISER bis zu 4800-mal schneller ist und dabei keine Unterbrechung der Verarbeitung erfordert.

Network	Area LUT utilization	Analysis Control Circuitry	Interconnects	DNN execution time in FPGA
Base CNN	12%	N/A	Data Exchange Interconnect	131ms
Fault Resilience Assessment				
CNN instrumented with FI	**23%**	Complex FI Controller	(Data Exchange + FI) Interconnect	632,000ms
CNN instrumented with APPRAISER	~29%	**Simple AxC Activator**	**Data Exchange Interconnect**	**131ms**

Abbildung 9.30.: Overheads von APPRAISER im Vergleich zur Referenz-Fehlerinjektionsmethode (Conv1-Schicht) [Tah+23b]

Um die Effizienz von APPRAISER zu demonstrieren, vergleichen Taheri et al. [Tah+23b] die Ausführungszeiten mit traditionellen FI-Methoden (siehe Abbildung 9.31).

Affected/Measured Layers	Bitflips in subsequent layers			
	Injection of a single fault		Injection of a double fault	
	Fault Injection (reference) [%]	Approximation with MULT1 [%]	Approximation with MULT2 [%]	Fault Injection (reference)[%]	Approximation with MULT1 [%]	Approximation with MULT2 [%]
Conv1/Conv1	10.00	9.97	9.98	9.99	10.00	9.99
Conv1/Pool1	9.03	9.03	9.03	9.06	9.06	9.05
Conv1/Conv2	16.73	16.72	16.74	16.74	16.74	16.74
Conv1/Pool2	16.40	16.45	06.50	16.55	16.50	16.45
Conv1/FC	9.25	9.25	8.50	9.30	9.30	9.30
Conv2/Conv2	16.71	16.72	16.71	16.76	16.74	16.74
Conv2/Pool2	16.40	16.45	16.41	16.50	16.50	16.50
Conv2/FC	10.10	8.50	7.80	10.10	9.30	8.30
Affected Layer	DNN Accuracy/Recall drop					
Conv1	2.3/4.7	2.7/8.0	2.2/6.7	4.7/14.0	5.8/17.4	4.2/12.7
Conv2	1.8/6.0	1.6/5.0	2.7/8.0	9.1/26.4	9.1/26.4	8.9/26.7

Abbildung 9.31.: Bitflips und Genauigkeits-/Recall-Abfall verursacht durch APPRAISER im Vergleich zur Referenz-Fehlerinjektionsmethode [Tah+23b]

Die Ergebnisse von Taheri et al. [Tah+23b] zeigen eine bemerkenswerte Übereinstimmung zwischen den FI-Methoden und dem APPRAISER Approximationsansatz mit nahezu identischen Bitflip-Raten (siehe gelbe Markierung in Abbildung 9.31) beim Injizieren von Fehlern.

Irradiation

geschrieben von Matthias Löwen

Laut Ahmadilivani et al. [Ahm+24b] ist eine besonders realistische Methode zur Bewertung der Fehlertoleranz mittels FI die Bestrahlung von DNN-Beschleunigern mit hochenergetischen Teilchen, wie Neutronen, Protonen oder Ionen. Dadurch können Soft Errors unter realen physikalischen Bedingungen erzeugt und deren Auswirkungen auf diese untersucht werden.

Für die Betrachtung von FPGA-basierten Systemen wurden in einem Großteil der Studien Zynq SoC-Boards mit Neutronen bestrahlt. Dixit und Wood von Oracle Labs [DW11] haben beispielsweise die LANSCE-Einrichtung in Los Alamos genutzt, da deren Neutronenstrahler eine Energieverteilung ähnlich der kosmischen Neutronenstrahlung der Erdoberfläche entspricht, allerdings eine etwa 100 Millionen Mal höhere Intensität beinhaltet.

Abbildung 9.32.: Testaufbau von Oracle in Los Alamos [DW11, S. 2]

Neben FPGAs wurden laut Ahmadilivani et al. [Ahm+24b] auch GPUs und Tensor Processing Units (TPUs) unter Anwendung von Strahlung getestet und evaluiert. Die getesteten GPUs sind dabei ausschließlich von NVIDIA und haben verschiedene Architekturen. Dabei wurden in manchen Studien bis zu 110.000 Jahre Strahlung an der Hardware simuliert und sowohl mit als auch ohne ECC getestet.

Für die Bewertung wird die SER und FIT betrachtet. Um die SER zu berechnen wird zunächst der Durchschnittswert σ mit folgender Gleichung beschrieben:

$$\sigma = errors/Flux$$

Unter *errors* wird die Anzahl der beobachteten Fehler und unter *Flux* die Gesamtanzahl der Partikel, welche die Oberfläche des Boardes erreicht haben, verstanden. Der σ-Wert hat die

Einheit cm^2 und wird für die Berechnung von der SER genutzt:

$$SER = \sigma \cdot \phi$$

wobei ϕ der Teilchenfluss mit der Einheit $[\frac{1}{cm^2 \cdot s}]$ ist.

Außerdem nutzen laut Ahmadilivani et al. [Ahm+24b] mehrere Studien zur Evaluation zusätzlich die Fehlerklassifikation basierend auf den Ausgaben des DNNs. Dabei wird zwischen tolerierbaren Fehlern, kritischen Fehlern und Systemabstürze unterschieden. Ein Fehler ist hierbei tolerierbar, wenn er nicht zu einer Fehlklassifizierung führt. Die kritischen Fehler werden auch als SDC bezeichnet und sorgen für eine falsche Klassifizierung. Für jede dieser Fehlerkategorien wird die FIT-Rate separat erfasst, um die Zuverlässigkeit der Hardware unter realitätsnaher Strahlung quantitativ einschätzen zu können.

9.5.2. Analytische Methoden

geschrieben von Matthias Löwen

Analytische Methoden bewerten die Zuverlässigkeit von DNNs, ohne aktiv Fehler auf Hardwareebene zu injizieren. Stattdessen analysieren sie die mathematische Struktur und Funktionsweise eines Netzwerks, um Rückschlüsse auf dessen Fehlertoleranz zu ziehen. Dadurch sind diese Verfahren deutlich leichtgewichtiger als die FI und sind für alle DNN-Beschleuniger anwendbar. Ziel ist die Identifikation besonders fehleranfälliger Komponenten, wie zum Beispiel Neuronen oder Gewichtungen, um gezielt Schutzmechanismen einsetzen zu können.

Ahmadilivani et al. [Ahm+24b] unterscheiden vier Hauptansätze:

- Layerwise Relevance Propagation (LRP)-basierte Analyse
- Gradientenbasierte Analyse
- Schätzungsbasierte Analyse
- Machine Learning (ML)-basierte Analyse

LRP-basierte Analyse

Im ersten Ansatz wird das DNN anhand des LRP-Algorithmus analysiert. Die Methode basiert auf der Hypothese von Schorn et al. [SGA18], dass Neuronen, die einen hohen Einfluss auf das Ergebnis des DNNs haben, auch eine höhere Auswirkung auf die Klassifikationsgenauigkeit besitzen und somit kritischer für die Zuverlässigkeit sind.

Unter Anwendung des LRP-Algorithmus wird jedes Neuron vom Ausgang des Netzes aus rückwärts durch das Modell für jede Schicht entlang der beteiligten Pfade propagiert. Bei jedem Knoten werden die relevanten Merkmale der Eingangsdaten für das Zustandekommen der Ergebnisse berechnet. Dabei wird für jedes Neuron über alle Schichten ein Relevanzwert berechnet und in einer Liste gespeichert. Auf Grundlage dieser Werte lassen sich gezielte Schutzmaßnahmen ableiten, um besonders kritische Neuronen besser abzusichern. Nach Schorn et al. [SGA19] ist eine ausgewogene Verteilung der Resilienz innerhalb des Netzes der erste Ansatz einer Methode, welche die Klassifizierungsgenauigkeit erhöht.

Gradientenbasierte Analyse

Ein weiterer Ansatz der analytischen Bewertung basiert auf der Analyse der Gradienten. Dabei wird die Sensitivität einzelner Gewichte oder *feature maps* in Bezug auf die Ergebnisse des neuronalen Netzes ermittelt. Ziel ist es, die Gewichte zu identifizieren, deren Änderung durch Bitfehler einen besonders starken Einfluss auf das Resultat hätte. [Ahm+24b]

Nach Ahmadilivani et al. [Ahm+24b] wird die Sensitivität eines Gewichts w als Produkt aus dem Gradienten dieses Gewichtes und dem Erwartungswert des Fehlers ε_w definiert:

$$Sensitivity_w = gradient_w \cdot \varepsilon_w$$

Je größer die Sensitivität, desto größer ist der potenzielle Einfluss eines Fehlers dieses Gewicht auf die Ausgabe des Netzes. Daraus ergibt sich die Möglichkeit, empfindliche Gewichte durch robuste Hardware abzusichern.

Schätzungsbasierte Analyse

Die schätzungsbasierten Methoden zielen darauf ab, die Vulnerabilität von *feature maps* durch die Berechnung des Maximalwertes eines Neurons, die Spannweite einer *feature map* sowie den durchschnittlichen $L2$-Wert zu bestimmen. Der $L2$-Wert beschreibt dabei den Abstand zwischen dem betrachteten Vektor zum Ursprung in einem euklidischen Raum. Die drei Werte geben Aufschluss darüber, wie stark eine Komponente das Verhalten des DNNs beeinflusst.

Laut Ahmadilivani et al. [Ahm+24b] ist ein anderer Ansatz für die schätzungsbasierte Methode die Anwendung einer Gleichung, welche die Fehlklassifikationsrate eines CNNs berechnet. Dabei wird davon ausgegangen, dass jede Rechenoperation, die als Ergebnis nicht null hat, als potenziell kritisch betrachtet wird, da ein Fehler das Ergebnis verfälschen könnte. Die Soft Error Rate of Neural Networks (SERN) wird über folgende Gleichung bestimmt:

$$SERN = \frac{Crit\_OPs_i + \sum_{i+1}^{n} OPs}{\sum_{i}^{n} OPs}$$

Dabei bezeichnet $Crit\_OPs_i$ die Anzahl der kritischen Operationen in Schicht i und OPs die Gesamtzahl der Operationen in den nachfolgenden Schichten.

ML-basierte Analyse

Laut Ahmadilivani et al. [Ahm+24b] ist ein ML-basierter Ansatz für die analytische Betrachtung das Anwenden des Open-Set Recognition (OSR)-Verfahren. Das Konzept des OSR-Verfahrens besteht darin zu erkennen, ob die Ausgabe des DNNs zu einer bekannten Klasse gehört oder nicht.

Die Analyse basiert auf der Auswertung der Ausgabe-Logits. Dieser ist die Ausgabe der letzten Schicht, bevor die Softmax-Funktion angewendet wird. Durch das Festlegen von Schwellwerten für diese Logits lassen sich kritische Fehler identifizieren, die das DNN in einen unsicheren Zustand versetzen könnten. Die unterschiedlichen OSR-basierten Methoden können in dem Paper von Garvini et al. [Gav+22] genauer nachvollzogen werden.

Zusammenfassung

Die analytischen Methoden verzichten auf die Fehlerinjektion und analysieren stattdessen die mathematische Struktur und das Verhalten des neuronalen Netzes. Sie sind leicht skalierbar, ausschließlich auf Software-Ebene implementierbar und können auf verschiedene DNNs-Beschleuniger angewendet werden. Diese Verfahren sind somit deutlich effizienter als die Methoden der Fault Injection, liefern allerdings nur angenäherte Werte, deren Ergebnisse stark von der Größe und Qualität des Netzes abhängt. Die schätzungsweisen Verfahren sind dabei besonders schnell, liefert allerdings nicht so genaue Werte, wie die restlichen Verfahren.

9.5.3. Hybride Methoden

geschrieben von Mathias Löwen

Hybride Methoden kombinieren analytische Verfahren mit FI-Techniken, um die Effizienz der analytischen Modelle mit der realitätsnahen Genauigkeit von Fault Injection zu verbinden. Laut Ahmadilivani et al. [Ahm+24b] lassen sich hybride Methoden effizient einsetzen, um kritische Komponenten frühzeitig zu erkennen und zu testen. Dadurch wird die Evaluierung sowohl beschleunigt als auch präzisiert.

Als Beispiel einer hybriden Methode wird von Ahmadilivani et al. [Ahm+24b] das „FIdelity Framework“ genannt. Es nutzt zunächst eine analytische Analyse zur Identifikation fehleranfälliger Neuronen oder Komponenten und führt anschließend FI gezielt auf diese durch. Durch die Kombination beider Ansätze kann eine umfassende, ressourcenschonende Evaluation der Fehlertoleranz durchgeführt werden.

9.6. Zusammenfassung

geschrieben von Arian Dannemann

In diesem Kapitel wurden verschiedene Aspekte der Auswirkungen von Rechenfehlern im neuronalen Netz beleuchtet.

Einleitend wurden relevante Fehlerarten, wie permanente Fehler, transiente Fehler, Hard Errors, Soft Errors, Single Event Upsets (SEUs) und Multi Bit Upsets (MBUs) erklärt. Anschließend wurde näher auf die Herausforderungen sicherheitsrelevanter Systeme eingegangen und erläutert, warum Rechenfehler drastische Auswirkungen haben können. Auf diese Auswirkungen wurden schließlich genauer eingegangen, besonders auf mögliche Fehlklassifikationen von neuronalen Netzen. Um dem entgegenzuwirken, wurden Maßnahmen zur Abhärtung neuronaler Netze vorgestellt, darunter klassische Methoden wie Triple Modular Redundancy (TMR) aber auch spezifische Ansätze wie der Adaptive Fault-Tolerant Approximate Multiplier (AdAM), Symptom-bases Error Detectors (SED), Selective Latch Hardening (SLH) und FORTUNE. Schließlich wurden mögliche Evaluationsmethoden zur Fehlertoleranz von neuronalen Netzen genannt, darunter Fault Injection (FI) sowie analytische und hybride Methoden.

Zusammenfassend lässt sich sagen, dass Rechenfehler in neuronalen Netzen insbesondere in sicherheitsrelevanten Systemen drastische Auswirkungen haben können. Moderne Härtungsmaßnahmen ermöglichen jedoch eine platzeffiziente Reduktion der Fehlerraten.

Die Forschung auf diesem Themengebiet ist sehr aktuell, wodurch sich noch viel Raum für weiterführende Arbeiten ergibt.

10. Zusammenfassung der Ergebnisse

geschrieben von Jan Bredereke

Wir verfolgen weiter unsere Forschungsfrage, wie man neuronale Netze auf FPGAs am besten nutzen kann, die so leistungsbeschränkt wie strahlungsfeste FPGAs sind (und etwas allgemeiner entsprechend neuronale Netze auf FPGAs für Edge-Computing). Ein autonomes Modellauto als konkretes Vehikel dient uns dabei als Stellvertreter für ein autonomes Raumfahrzeug. Im Projektdurchgang, der im vorliegenden Bericht beschrieben ist, haben wir an mehreren Teilaspekten gearbeitet und die folgenden Ergebnisse erzielt.

Für Leser über den Kreis zukünftiger Studierender im Projekt hinaus sind insbesondere zwei Ergebnisse interessant: Die Literaturrecherche zu Hardware-bedingten Rechenfehlern in neuronalen Netzen (s.u.), motiviert durch die Problematik der Weltraumstrahlung, gibt einen Überblick über die aktuelle Forschung zu schaltungstechnischen Maßnahmen zur Eindämmung der Fehlerwirkungen sowie zu Meßansätzen, um den Erfolg solcher Maßnahmen nachzuweisen. Die experimentelle Untersuchung systematischer Fehler beim Erkennen von Sprachkommandos (s.u.) ergab Einsichten, worauf beim Training eines neuronalen Netzes dafür zu achten ist.

Gesamtarchitektur erheblich aktualisiert und wieder verfügbar gemacht: Die Gesamtarchitektur des angestrebten Systems hatte sich über etliche Jahre eher ungesteuert weiterentwickelt. Die Systems-Engineering-Leistung in Kap. 2 erarbeitet zunächst die für die Architektur relevanten Dokumentstellen aus der Historie und und stellt diese und einige wichtige Grundbegriffe im Überblick dar. Dann klärt sie die aktuell angestrebte Gesamtarchitektur in Abstimmung mit den Teilteams und dem Dozenten, arbeitet sie weiter aus und stellt sie schließlich in der zentralen Abbildung 2.2 auf Seite 12 übersichtlich und trotzdem detailreich dar, ergänzt durch erklärenden Begleittext.

Erkennen der Position einer Person in einem Videostrom verbessert und Methode zum Messen der Qualität entwickelt: Im bisher im Projekt verwendeten Verfahren für das Training des neuronalen Netzes wurden etliche erhebliche Probleme erkannt und erfolgreich behoben (Kap. 3). Darauf aufbauend ist nun ausgearbeitet, wie die Qualität des Erkennens der Position mit Hilfe des trainierten neuronalen Netzes möglichst objektiv gemessen werden kann. Dabei zeigt sich, dass unter anderem insbesondere die Bildvorverarbeitung durch Kachelung die Qualität stark beeinflusst und ebenfalls berücksichtigt werden muss.

Auf Basis eines Fehlermodells werden systematisch Testziele und daraus eine Testfolge abgeleitet. Deren Vollständigkeit wird methodisch nachgeprüft. Als Ergebnis dieser Methode kann die Erkennungsqualität verglichen werden von einerseits dem neuronalen Netz auf der FPGA-Hardware, mit Vorverarbeitung durch Kachelung und mit einem definierten Video als Eingabe, und andererseits von dem neuronalen Netz auf einem PC, mit Bildern aus dem Standard-COCO-Datensatz. Dies erlaubt Rückschlüsse u.a. über Fehler in der FPGA-Synthetisierung und über die Eignung der Parametrierung der Kachelung einerseits und über die Eignung des COCO-Datensatzes andererseits.

Zur Beurteilung der Qualität der Erkennung werden nun neu eine Konfusionsmatrix und Matthews Correlation Coefficient (MCC) eingesetzt. Aktuelle praktische Experimente ergaben, dass die Qualität der Erkennung durch das eigentliche neuronale Netz auf einem PC bisher und auch weiterhin unbefriedigend ist. Aus Zeitgründen konnten die Ursachen nicht untersucht werden. Diese sollten sinnvollerweise angegangen werden, bevor auf Basis einer praktischen Durchführung des obigen Vergleichstests weitere Qualitätsverbesserungen vorgenommen werden.

Keine praktischen Erfolge beim Erkennen des Gestenkommandos in einem bereits identifizierten Bild mit einer Person: Hier hätten die bereits erarbeitete Gestenerkennung [ACB+21] (siehe auch Kap. 3.3 im aktuellen Bericht) und auch die bereits erarbeitete Kachelung ([MKB+21, Kap. 6.4], [Mül21], [ACB+21, Kap. 9.2.5]; siehe ebenfalls auch Kap. 3.3 im aktuellen Bericht) in das Gesamtsystem integriert werden können. Auch hätten die Vorschläge [ACB+21, Kap. 12.2], um das neuronale Netz für die Gestenerkennung zu verbessern, ausprobiert werden können.

Kapitel 4 des vorliegenden Projektberichts enthält allerdings keine erfolgreiche Auseinandersetzung mit den eben genannten bisherigen Ergebnissen, und auch nicht mit der in Kap. 2 erarbeiteten Architektur des Gesamtsystems. Weiterhin bezieht sich Kap. 4 nicht auf die inhaltlich verwandte Bilderkennung zwecks Personenerkennung in Kap. 3. Stattdessen werden einige andere Ansätze zur Bilderkennung beschrieben, allerdings ohne diese Ansätze in Beziehung zu den konkreten Randbedingungen zu setzen.

Verbesserte Neuimplementierung der Fahrsteuerung für das Vehikel: Passend zur Renovierung der Gesamtarchitektur (s.o.) wurde die Struktur der Fahrsteuerung neu konzipiert (Kap. 5). In der Folge wurde der bisherige Code, der ohnehin auf einem fehlerbehafteten endlichen Automaten basierte, verworfen unnd neu konzipiert und implementiert. Genauer wurde zunächst nur eine Smartphone-basierte manuelle Fernsteuerung des Vehikels umgesetzt, die für Tests nützlich sein kann. Die Integration der in der überarbeiteten Architektur vorgesehenen Schnittstellen zu den anderen die Fahrt kontrollierenden Systemkomponenten steht noch aus, ebenso wie die Umsetzung der Komponente, die die Kommandos koordiniert, die über diese Schnittstellen kommen werden.

Weiterhin wurden die mechanische Ungenauigkeit des Vehikels bei Geradeausfahrt wurde quantitativ bestimmt und Ursachen analysiert. Zur Unterstützung der manuellen Fernsteuerung wurde ein WLAN-Access-Point auf dem Vehikel realisiert, einschließlich einer Anleitung zum Einrichten auf zukünftigen Vehikeln.

Auswahl eines für das Vehikel geeigneten Mikrofons: Nach der vorangegangenen Entscheidung, das Board PYNQ-Z1 durch das Board PYNQ-Z2 zu ersetzen, das kein eingebautes Mikrofon mehr hat, wurde nun unternommen, ein zu den Anforderungen des Vehikels passendes Mikrofon systematisch auszuwählen (Kap. 6). Dabei war neben grundlegenden mechanischen und elektrischen Randbedingungen insbesondere die Art der Richtcharakteristik zu beachten.

Beheben bzw. Ausschließen einiger systematischer Fehler beim Erkennen von Sprachkommandos: Es wurde herausgefunden, dass die eigentlich gute Qualität des Erkennens von Sprachkommandos mit dem neuronalen Netz aus einem früheren Projektdurchgang [BJB+23] vollständig zusammenbricht, sobald die Test-Eingabe vor und nach dem Sprachkommando zusätzlich ein leises Hintergrundstörgeräusch enthält (Kap. 7). Es liegt die Vermutung nahe, dass das

trainierte neuronale Netz gezielt nach dem Zeitbereich sucht, der nicht absolute Stille enthält, und ihn bei einem Störgeräusch nicht mehr findet. Es wurden Experimente mit einer Vorabfilterung des Audiosignals angestellt. Voraussetzung für die Anwendbarkeit ist, dass das Sprachkommando signifikant lauter als alle Hintergrundgeräusche ist. Angedacht, aber nicht mehr umgesetzt wurde, schon die Trainingsdaten mit Hintergrundgeräuschen zeitlich zu umgeben, um sie realistischer zu machen.

Im Rahmen von Experimenten konnten bestimmte weitere systematische Fehler ausgeschlossen werden: Laute Hintergrundgeräusche während (nicht vor oder nach) dem Sprachkommando sind kein wesentliches Problem. Ebenso ist das Netz nicht nur auf eine bestimmte Stimme oder Stimmlage trainiert. Es wurde bestätigt, dass die zeitliche Position des Sprachkommandos in der Aufnahme das Ergebnis praktisch nicht beeinflusst. (Letzteren Aspekt hatte ein vorheriger Projektdurchgang bereits explizit ins Training integriert.)

Weiterhin wurde vorab ein technischer Datenformat-Fehler aus dem früheren Projektdurchgang [BJB+23] gefunden, der ebenfalls dazu führte, dass ein Test des Erkennens von Sprachkommandos mit Realwelt-Mikrofonaufnahmen sehr viel schlechtere Ergebnisse ergab als auf Basis der dedizierten Testdaten erwartet.

Schließlich wird die Idee aufgebracht, ein zweites neuronales Netz zu verwenden, um zunächst nur zu erkennen, ob überhaupt gerade Sprache zu hören ist, bevor das eigentlich neuronale Netz die Sprachkommandoerkennung durchführt. Diese Idee entspricht dem Konzept der „Region of Interest“ (RoI) in der Bildverarbeitung. Hierbei wäre allerdings zu klären, ob der zusätzliche Hardware-Aufwand im FPGA für die RoI-Erkennung zu größeren Hardware-Einsparungen im FPGA bei der Sprachkommandoerkennung führt, sprich, ob die Sprachkommandoerkennung durch die RoI erheblich erleichtert wird. Denn die Hardware-Ressourcen für beide neuronalen Netze sind dauerhaft reserviert, auch wenn sie nicht zu jeder Zeit genutzt werden.

Den Aufbau aus mechanischen und elektronischen Komponenten verbessert: Im Zuge des Ersetzens des PYNQ-Z1-Boards durch das PYNQ-Z2-Board wurde der gesamte mechanische und auch elektronische Aufbau des Vehikels verbessert, und es wurde dabei ein vollständiges 3D-Modell des Vehikels erstellt (Kap. 8). Es gibt nun neu ein 3D-gedrucktes Gehäuse (vergleiche Abb. 1.1 auf Seite 2) und stabilere Halterungen. Die USB-Verkabelung wurde platzsparender gelöst. Die Beschreibung des Umbaus zusammen mit einer expliziten Materialliste dürften es leicht machen, nach dem jetzt umgebauten Vehikel auch das zweite Vehikel auf den verbesserten Stand zu bringen.

Literaturrecherche zu Hardware-bedingten Rechenfehlern in neuronalen Netzen: Motiviert durch die Problematik der Weltraumstrahlung in der Raumfahrt wurde eine Literaturrecherche zu Hardware-bedingten Rechenfehlern in neuronalen Netzen durchgeführt. Kapitel 9 stellt zunächst die technischen Grundlagen sowie die aus den Rechenfehlern resultierenden Herausforderungen bei sicherheitsrelevanten Systemen dar. Anschließend werden relevante Faktoren für solche Fehler herausgearbeitet sowie ihre Auswirkungen systematisch betrachtet. Eine Darstellung aktueller Forschungsarbeiten zu (meist) schaltungstechnischen Maßnahmen zur Eindämmung der Fehlerwirkungen folgt, die mehr als die Maßnahme der bekannten Triple Modular Redundancy (TMR) leisten: Adaptive Fault-Tolerant Approximate Multiplier (AdAM), Symptom-Based Error Detectors (SED), Selective Latch Hardening (SLH) und Fault Tolerance Technique (FORTUNE). Abgeschlossen wird die Recherche mit einem Überblick über weitere aktuelle Forschungsarbeiten, die die Fehlertoleranz von neuronalen Netzen systematisch messen, um den Erfolg von Eindäm-

mungsmaßnahmen nachzuweisen: etliche Ansätze zu Fault Injection, einige analytische Methoden und ein Beispiel für eine hybride Methode.

11. Ausblick

geschrieben von Jan Bredereke

Auch nach diesem Projektdurchgang bleiben noch viele interessante Aufgaben und Herausforderungen, um unsere Forschungsfrage zu beantworten. Die folgenden bieten sich besonders für weitere Projektdurchgänge an:

Schaltungstechnische Maßnahmen gegen Hardware-bedingte Rechenfehler in neuronalen Netzen Ideen aus den aktuellen Forschungsansätzen in der umfangreichen Literaturrecherche in Kap. 9 können weiterverfolgt werden, um die Rechenleistung eines neuronalen Netzes im Weltraumeinsatz über das hinaus zu erhöhen, was bisher durch Strahlungsfestigkeit möglich ist, die bloß aus genügend großen Chip-Strukturbreiten resultiert. Soweit eine höhere Fehlerrate aufgrund einer eigentlich zu kleinen Strukturbreite durch geeignete schaltungstechnische Maßnahmen ausgeglichen werden kann, können leistungsfähigere Chips zum Einsatz kommen. Hier wäre zum Beispiel konkret zu untersuchen, inwieweit diese Maßnahmen mit dem bisher verwendeten Framework umsetzbar sind, das ein neuronales Netz auf ein FPGA bringt, oder inwieweit passende andere Frameworks zur Verfügung stehen. Dabei ginge es zunächst nur um die Machbarkeit des konkreten Einsatzes. Das Nachweisen des Erfolgs, die Fehlerrate tatsächlich im Griff zu haben, wäre nicht ohne den in Kap. 9.5 beschriebenen erheblichen Messaufwand zu haben.

Verbessern des Erkennens der Position einer Person durch das eigentliche neuronale Netz: Es sind Wege zu finden, die bisher unbefriedigende Qualität der Erkennung durch das eigentliche neuronale Netz auf einem PC (ohne FPGA-Umgebung) zu erhöhen. Kapitel 3.8 macht hier Vorschläge. Dort steht auch mehr zu dem im Anschluss sinnvollen Durchführen des neuentwickelten Vergleichstests aus Kap. 3, um Qualitätsverbesserungen beim Erkennen auf der FPGA-Hardware vornehmen zu können.

Es könnte weiterhin erprobt werden, ob das Verwenden eines neuronalen Netzes speziell für Single-Class-Detection, also eines, das nur ja/nein-Antworten für die Personenerkennung gibt, Vorteile bringt. Und das Verfahren Non-Maximum-Suppression (NMS) [ACB+21, Kap. 12.2] zum Verbessern der Kachelung könnte ausprobiert werden.

Integrieren des Erkennens des Gestenkommandos in einem bereits identifizierten Bild mit einer Person: Die schon in Kap. 10 beschriebenen Aufgaben sollten angegangen werden: Die bereits erarbeitete Gestenerkennung [ACB+21] (siehe auch Kap. 3.3 im aktuellen Bericht) und auch die bereits erarbeitete Kachelung ([MKB+21, Kap. 6.4], [Mül21], [ACB+21, Kap. 9.2.5]; siehe ebenfalls auch Kap. 3.3 im aktuellen Bericht) sollten in das Gesamtsystem integriert werden. Auch sollten die Vorschläge [ACB+21, Kap. 12.2], um das neuronale Netz für die Gestenerkennung zu verbessern, ausprobiert werden.

Als kleine Modifikation der Gesamtsystemarchitektur in Kap. 2 kann dabei erwogen werden, nach erfolgter Erkennung der Position der Person das dann eher wenig zeitkritische neuronale Netz zur Erkennung einer Geste auf dem Mikrocontroller und nicht im FPGA auszuführen.

Verbessern der Bilderkennung zwecks Personenerkennung und Gestenerkennung Bei der Bilderkennung sowohl zur Personenerkennung als auch zur Gestenerkennung könnte untersucht werden, ob ausgenutzt werden kann, dass aufeinanderfolgende Bilder in einem Videostrom meist recht ähnlich sind. Insbesondere beim Verfolgen der Position einer Person könnte ausgenutzt werden, dass die Person im nächsten Bild meist recht nahe an ihrer bisherigen Position zu sehen sein wird. Dies könnte die Kachelung beschleunigen, da zuerst diejenigen Kacheln auf eine Person hin untersucht werden sollten, die der bisherigen Position nahe sind. (Vergleiche den Block „Region of Interest-Bestimmung“ in Abb. 2.2 auf Seite 12 in Kap. 2 mit der Gesamtsystemarchitektur.)

Erkennen von Sprachkommandos: Beim Erkennen von Sprachkommandos durch ein neuronales Netz sollte das Training des Netzes so modifiziert werden, dass die Trainingsaufnahmen über ihre gesamte Länge mit Hintergrundgeräuschen unterlegt werden, um die Erkennung in Realwelt-Aufnahmen zu verbessern.

Weiterhin steht die Integration des neuronalen Netzes ins FPGA, dann in die gesamte Audio-Datenflusskette und schließlich ins Vehikel noch aus. Im einzelnen sind dies insbesondere die nach dem vorigen Projektdurchgang [ABB+25, Kap. 11.2] schon genannten Punkte:

Der Analog-Digital-Wandler auf dem PYNQ-Z2-Board sollte so tief verstanden werden, dass er gezielt in eigene FPGA-Projekte eingebunden werden kann.

Für die Transformation der rohen Audiodaten in Mel-Frequenz-Cepstrum-Koeffizienten sollten die nötigen Schritte noch klarer beschrieben und ggf. ausgewählt werden (vergleiche [ABB+25, Kap. 10]). Anschließend sollten diese Schritte in VHDL umgesetzt oder mit fertigen IP-Blöcken realisiert werden.

Klassifikation von Video- und Audiodaten allgemein: Eine Idee aus dem vorigen Projektdurchgang [ABB+25, Kap. 11.2], um die Effizienz der Sprachkommandoerkennung zu steigern, ist, gezielt einen für diese spezielle Aufgabe besonders geeigneten Klassifikator für einen Frequenzkoeffizientenstrom zu suchen. Bisher wurden nur gefaltete neuronale Netze (CNNs) verwendet. In [ABB+25, Kap. 10] wurde bereits dikutiert, dass Long-Short-Term-Memory-Netze (LSTM-Netze) grundsätzlich besser geeignet sein könnten, um eine Zeitreihenanalyse wie die Sprachkommandoerkennung effizient durchzuführen. Voraussetzung dafür dürfte sein, dass sich auch diese Netze quantisieren lassen. Insbesondere Werkzeugunterstützung dürfte wichtig sein. Dies kann recherchiert, prototypisch ausprobiert und bei Erfolg tiefer weiterverfolgt werden.

Auch der Blick auf ganz andere Klassifikatoren könnte Fortschritte bringen, z.B. auf die sich derzeit sehr stark entwickelnden Transformer. Sie erscheinen ebenfalls für eine Zeitreihenanalyse sehr geeignet. In einem ersten Schritt wäre zu klären, welche Rechenoperationen dort zu Grunde liegen, und ob sie sich, insbesondere mit Quantisierung, grundsätzlich effizient auf ein FPGA bringen lassen.

Falls sich neuronale Netze zur Bilderkennung als nicht ausreichend erweisen sollten, könnten ebenfalls andere Klassifikatoren erwogen und auf ihre Eignung für eine Implementation in einem

FPGA untersucht werden. Insbesondere eine Recherche nach fertigen Klassifikatoren für Bildströme (Video, s.o.) könnte sinnvoll sein.

Vervollständigen der Fahrsteuerung für das Vehikel: Für die Fahrsteuerung steht die Integration der in der überarbeiteten Architektur vorgesehenen Schnittstellen zu den anderen die Fahrt kontrollierenden Systemkomponenten noch aus, ebenso wie die Umsetzung der Komponente, die die Kommandos koordiniert, die über diese Schnittstellen kommen werden.

Umbau auch des zweiten Vehikels: Nach dem erfolgreichen mechanischen und elektrischen Umbau des ersten Vehikels (siehe Kap. 8) soll nun entsprechend auch das zweite Vehikel auf den verbesserten Stand gebracht werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AdAM Adaptive Fault-Tolerant Approximate Multiplier. 95, 110–112

ALU Arithmatic Logical Unit. 92, 94, 138

AP Access Point (Modus). vi, 51, 57, 60, 155–157

API Application Programming Interface. 154

ASIC Application-Specific Integrated Circuit. 91, 95

AxC Approximate Computing. 124, 125

AXI Advanced eXtensible Interface. 54

BNN Binarized Neural Network. 99, 102–104, 139

CNN Convolutional Neural Network. 29, 96, 110, 129

CNV Convolutional Network Topology. 103–105, 139

CPU Central Processing Unit. 16, 23, 61, 91, 113

CSS Cascading Style Sheets. 64, 65, 67

DHCP Dynamic Host Configuration Protocol. 154

DHCP Local Area Network. 154

DNN Deep Neural Network. 91, 92, 95, 103, 105, 109, 110, 113, 114, 116, 117, 120, 122–125, 127–130

DNS Domain Name System. 158

DRAM Dynamic Random Access Memory. 91, 95

ECC Error Correction Code. 94, 101, 127

EDMN Error Detection and Mitigation Network. 100

FE Fault Emulation. 116, 123

FF Flip Flop. 123, 140

FI Fault Injection. 116, 117, 123–125, 127, 128, 130

FIT Failure In Time. 107, 113, 114, 127, 128

FPGA Field-Programmable Gate Array. 26–28, 30–32, 51, 68, 91, 95, 100, 105, 116, 120, 123

FS Fault Simulation. 116

GPIO General Purpose Input/Output. 61

GPU Graphics Processing Unit. 15, 16, 19, 22, 32, 91, 94, 95, 116, 123, 127

HDL Hardware Description Language. 116

HLS High Level Synthesis. 120, 122

HSB Hochschule Bremen. 56

HTML Hypertext Markup Language. 64–67

HTTP Hypertext Transfer Protocol. 154, 155

HTTP Hypertext Transfer Protocol. 51

IP Intellectual Property. 52, 54, 155

LED light-emitting diode. 52

LFC Layer Fully Connected. 108, 139

LOD Leading One Detector. 111

LRN Local Response Normalization Layer. 105

LRP Layerwise Relevance Propagation. 128

LSB Least Significant Bit. 111

LUT Look-Up Table. 123, 140

MBU Multi Bit Upset. 94–98, 103, 108, 115

MCC Matthews Correlation Coefficient. 15, 24, 25

MCU Multi Cell Upset. 101

ML Machine Learning. 128, 129

MLP Multilayer Perceptrons. 96

MSB Most Significant Bit. 111, 112, 114, 116

NN Neural Network. 61, 91, 92, 95, 98–100, 102–108, 112, 113, 139

OSR Open-Set Recognition. 129

PDM Pulsdauermodulation. 51

PDP Power-Delay Product. 111, 112

PE Processing Engine. 91, 92, 138

PL Processing Layer. 52, 53

PS Processing System. 52, 53, 61

PWM Pulsweitenmodulation. 51, 53, 54, 56, 63

QNN Quantized Neural Network. 91, 102, 103, 114

RCC Reinforcing Charge Collection. 113, 114

REST Representational State Transfer. 154

RTL Register-Transfer Level. 117

SBC single board computer. 154

SBU Single Bit Upset. 94–97

SDC Silent Data Corruption. 105–107, 109, 113, 128, 139

SED Sympton-based Error Detector. 112, 113

SER Soft Error Rate. 101, 110, 127, 128

SERN Soft Error Rate of Neural Networks. 129

SEU Single Event Upset. 94, 96, 97, 101–104, 109, 115, 139

SEUT Single Event Upset Tolerant. 113, 114

SLH Selective Latch Hardening. 113

SRAM Static Random Access Memory. 95, 96

SSH secure shell. 62, 154, 155

TMR Triple Modular Redundancy. 100, 109–116

TPU Tensor Processing Unit. 127

UCL Universal Color Language (ISCC-NBS System). 114, 139

WLAN Wireless Local Area Network. ix, 60, 155

Literaturverzeichnis

[ACB+22] Philipp Altnickel, Ferhat Cansu, Jan Bredereke u. a. *Personenerkennung durch schwache FPGAs in autonomen Fahrzeugen mittels Neuronaler Netze*. Techn. Ber. HSB, 1. März 2022.

[ACB+21] Philipp Altnickel, Tuncer Catalkaya, Jan Bredereke u. a. *Gesten- und Objekterkennung durch schwache FPGAs in autonomen Fahrzeugen mittels neuronaler Netze*. Techn. Ber. HSB, 1. Sep. 2021.

[Mül21] Felix Müller. „Dynamisches Tiling auf schwachen FPGAs zur Objekterkennung mithilfe kleiner neuronaler Netze“. Bachelorthesis. HSB, 23. Juni 2021.

[MKB+21] Felix Müller, Niklas Krekel, Jan Bredereke u. a. *Applying Binarized Neural Networks on FPGAs to an Autonomous Driving Problem*. Englisch. Techn. Ber. HSB, 31. März 2021.

[HSB+20] Colin von Huth, Marvin Soldin, Jan Bredereke u. a. *Bericht zum Projekt ‚Neuronale Netze auf strahlungstoleranten FPGAs für die Raumfahrt‘*. Techn. Ber. HSB, 14. Feb. 2020.

[Bre23] Jan Bredereke. „Enabling Neural Network Edge Computing on a Small Robot Vehicle“. Englisch. In: *Intelligent Distributed Computing XV* (Bremen, Germany, 14.–15. Sep. 2022). Hrsg. von Lars Braubach, Kai Jander und Costin Bădică. Bd. 1089. Studies in Computational Intelligence. Springer, 2023, S. 33–40. ISBN: 978-3-031-29103-6. DOI: 10.1007/978-3-031-29104-3_4. Im Erscheinen.

[HHB+22] Fynn Hagen, Jan Hartig, Jan Bredereke u. a. *Neural Network on an FPGA for Speech Command Recognition on an Autonomous Vehicle*. Englisch. Techn. Ber. Hochschule Bremen, 8. März 2022. 49 S.

[BJB+23] Jarno Burggräf, Nils Jahns, Jan Bredereke u. a. *Neuronale Netze auf einem FPGA zur Sprachkommandoerkennung*. Techn. Ber. Hochschule Bremen, 8. März 2023. 55 S.

[ABB+25] Mohammad Alibrahim, Mattes Bielefeld, Jan Bredereke u. a. *Sprachkommandoerkennung auf einem FPGA. Projektbericht zur Lehrveranstaltung Embedded Systems*. Techn. Ber. Hochschule Bremen, 26. März 2025. 172 S. DOI: 10.26092/elib/3777.

[Arm25] Arm. *AXI Protocol Overview*. Dokument ID 102202_0300, „Learn the architecture – An introduction to AMBA AXI“. Arm Limited. 2025. URL: https://developer.arm.com/documentation/102202/0300/AXI-protocol-overview (Abgerufen am 01. 08. 2025).

[Log+00] Beth Logan u. a. „Mel frequency cepstral coefficients for music modeling.“ In: *Ismir*. Bd. 270. Plymouth, MA. 2000, S. 1–11.

[WBM23] David John Wang, Daniel J Bell und Candace Makeda Moore. *Validation split (machine learning)*. letzte Überarbeitung am 3.4.2023. 2023. DOI: 10.53347/rID-61721. URL: https://radiopaedia.org/articles/61721.

[Yin19] Xue Ying. „An Overview of Overfitting and its Solutions“. In: *Journal of Physics: Conference Series* 1168 (Feb. 2019), S. 022022. DOI: 10.1088/1742-6596/1168/2/022022.

[CT10] Gavin C. Cawley und Nicola L. C. Talbot. „On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation“. In: *Journal of Machine Learning Research* 11 (2010). Hrsg. von Isabelle Guyon, S. 2079–2107. URL: https://www.jmlr.org/papers/volume11/cawley10a/cawley10a.pdf.

[Mos12] Ivo Mossig. *Stichproben, Stichprobenauswahlverfahren und Berechnung des minimal erforderlichen Stichprobenumfangs*. 2012. URL: https://www.econstor.eu/bitstream/10419/90425/1/73621089X.pdf#:~:text=Man%20erh%C3%A4lt%20eine%20geschichtete%20Stichprobe,eine%20reine%20Zufallsstichprobe%20nach%20den (Abgerufen am 25. 07. 2025).

[LLC22] Osval Antonio Montesinos López, Abelardo Montesinos López und José Crossa. *Multivariate Statistical Machine Learning Methods for Genomic Prediction*. 1. Aufl. Springer Cham, 2022, S. XXIV, 691. ISBN: 978-3-030-89010-0. DOI: 10.1007/978-3-030-89010-0. URL: https://doi.org/10.1007/978-3-030-89010-0.

[COC] COCO Consortium. *COCO Dataset*. Abgerufen am 24. Juli 2025. URL: https://cocodataset.org/#home.

[Cre] Creative Commons. *Attribution 4.0 International*. URL: https://creativecommons.org/licenses/by/4.0/ (Abgerufen am 24. 07. 2025).

[FPF25] Giuseppe Franco, Alessandro Pappalardo und Nicholas J Fraser. *Xilinx/brevitas*. 2025. DOI: 10.5281/zenodo.3333552. URL: https://doi.org/10.5281/zenodo.3333552 (Abgerufen am 26. 06. 2025).

[Reg] Regents of the University of California. *3-Clause BSD License*. URL: https://opensource.org/license/BSD-3-clause (Abgerufen am 24. 07. 2025).

[NVIa] NVIDIA Corporation & affiliates. *CUDA Toolkit Documentation*. URL: https://docs.nvidia.com/cuda/index.html (Abgerufen am 24. 07. 2025).

[Gooa] Google LLC. *Open Images Dataset V7*. URL: https://storage.googleapis.com/openimages/web/factsfigures_v7.html (Abgerufen am 24. 07. 2025).

[Lin+14] Tsung-Yi Lin u. a. „Microsoft COCO: Common Objects in Context“. In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[AMD24] AMD (ehemals Xilinx). *End-to-End Flow – FINN documentation*. 2024. URL: https://finn.readthedocs.io/en/latest/end_to_end_flow.html (Abgerufen am 24. 07. 2025).

[Umu+17] Yaman Umuroglu u. a. „FINN: A Framework for Fast, Scalable Binarized Neural Network Inference“. In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '17. ACM, 2017, S. 65–74.

[Blo+18] Michaela Blott u. a. „FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks“. In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11.3 (2018), S. 1–23.

[Red+15] Joseph Redmon u. a. „You Only Look Once: Unified, Real-Time Object Detection“. In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: https://arxiv.org/abs/1506.02640.

[Ult] Ultralytics. *Transfer Learning with Frozen Layers in YOLOv5*. URL: https://docs.ultralytics.com/yolov5/tutorials/transfer_learning_with_frozen_layers/ (Abgerufen am 25. 07. 2025).

[GBC16] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://alvarestech.com/temp/deep/Deep%20Learning%20by%20Ian%20Goodfellow,%20Yoshua%20Bengio,%20Aaron%20Courville%20(z-lib.org).pdf (Abgerufen am 25. 07. 2025).

[OP24] Sefa Burak Okcu und Giovanni Pollo. *quantized-yolov5: Low Precision YOLO (v1, v3, v5) Training with QAT*. https://github.com/sefaburakokcu/quantized-yolov5. GitHub repository, genutzt für quantization-aware training (QAT) und FINN-Kompatibilität. 2024.

[Baz+20] Valentin Bazarevsky u. a. „BlazePose: On-device Real-time Body Pose tracking“. In: *CoRR* abs/2006.10204 (2020). arXiv: 2006.10204. URL: https://arxiv.org/abs/2006.10204.

[Goob] Google. *Pose landmark detection guide*. URL: https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker (Abgerufen am 25. 07. 2025).

[Oso18] Daniil Osokin. „Real-time 2D Multi-Person Pose Estimation on CPU: Lightweight OpenPose“. In: *CoRR* abs/1811.12004 (2018). arXiv: 1811.12004. URL: http://arxiv.org/abs/1811.12004.

[Gooc] Google. *MoveNet: Ultra fast and accurate pose detection model.* URL: https://www.tensorflow.org/hub/tutorials/movenet (Abgerufen am 25. 07. 2025).

[Koh20] Kaleb Kohlhase. *Pulse Width Modulation (PWM): What Is It? How Can I Use It?* 15. Sep. 2020. URL: https://www.digikey.pl/pl/blog/pulse-width-modulation (Abgerufen am 09. 08. 2025).

[Hir22] Timothy Hirzel. *Basics of PWM (Pulse Width Modulation)*. 15. Dez. 2022. URL: https://docs.arduino.cc/learn/microcontrollers/analog-output/ (Abgerufen am 09. 08. 2025).

[Xil] Xilinx. *Overlay - Python productivity for Zynq (Pynq)*. URL: https://pynq.readthedocs.io/en/v2.6.1/pynq_libraries/overlay.html (Abgerufen am 23. 07. 2025).

[Gio21] Enrico Giordano. *PWM on PYNQ: how to control a stepper motor*. 16. Nov. 2021. URL: https://www.makarenalabs.com/pwm-on-pynq-how-to-control-a-stepper-motor/ (Abgerufen am 23. 07. 2025).

[TUL18] TUL Technology Unlimited. *PYNQ-Z2 Reference Manual*. Abgerufen am 10. Juli 2025. TUL Technology Unlimited. 2018. URL: https://dpoauwgwqsy2x.cloudfront.net/Download/pynqz2_user_manual_v1_0.pdf.

[Bre25] Pr. Dr. Jan Bredereke. *Persönliches Gespräch*. Persönliche Kommunikation. Geführt am 10. Juni 2025. 2025.

[Ber19] Herbert Bernstein. „Mikrofone“. In: *Elektroakustik: Mikrofone, Klangstufen, Verstärker, Filterschaltungen und Lautsprecher*. Wiesbaden: Springer Fachmedien Wiesbaden, 2019, S. 179–209. ISBN: 978-3-658-25174-1. DOI: 10.1007/978-3-658-25174-1_5. URL: https://doi.org/10.1007/978-3-658-25174-1_5.

[Ear12] John Eargle. „Microphone Measurements, Standards, and Specifications“. In: *The Microphone Book*. 2nd. Burlington, MA: Focal Press, 2012. ISBN: 9780240519616. URL: https://learning.oreilly.com/library/view/the-microphone-book/9780240519616/.

[Ltd25a] Freedman Electronics Pty Ltd. *SMARTLAV+*. Abgerufen am 17. Juli 2025. 2025. URL: https://edge.rode.com/pdf/page/377/modules/1354/smartlav-plus_datasheet.pdf.

[Ltd25b] Shenzhen Jiayz Photo Industrial. Ltd. *SPECs*. Abgerufen am 17. Juli 2025. 2025. URL: https://www.boyamic.com/product/lavalier-microphone-by-m1.

[Ltd25c] Audio-Technica Ltd. *Spezifikationen*. Abgerufen am 17. Juli 2025. 2025. URL: https://www.audio-technica.com/de-de/atr3350xis.

[She25] Ltd. Shenzhen Commlite Technology Co. *Specifications*. Abgerufen am 17. Juli 2025. 2025. URL: https://www.comica-audio.com/product/VM10-PRO.

[KG25a] Hama GmbH & Co KG. *Technische Daten*. Abgerufen am 17. Juli 2025. 2025. URL: https://support.hama.com/00004647/hama-richtmikrofon-rmn-uni-fuer-zubehoerschuh.

[Gmb14] Shure Europe GmbH. *SM WIRED MICROPHONES*. Abgerufen am 17. Juli 2025. 2014. URL: https://content-files.shure.com/publications/specSheet/en/sm58-spec-sheet.pdf.

[KG25b] Sennheiser Electronic SE & Co. KG. *Dauerpolarisiertes Kondensatormikrofonmodul passend zu den Speiseadaptern K6 und K6P*. Abgerufen am 17. Juli 2025. 2025. URL: https://warehousesound.com/r/behringerECM8000manual.pdf.

[KG25c] Sennheiser Electronic SE & Co. KG. *Dauerpolarisiertes Kondensatormikrofonmodul passend zu den Speiseadaptern K6 und K6P*. Abgerufen am 17. Juli 2025. 2025. URL: https://assets.sennheiser.com/downloads/8e061ae7c57a2ab2f976604dcc40285f.pdf.

[Sel24] Stefan Selle. *Training – Supervised Learning: Ein praktischer Einstieg ins überwachte maschinelle Lernen*. Saarbrücken, Deutschland: Springer Vieweg, 2024. ISBN: 978-3-662-67959-3. URL: https://doi.org/10.1007/978-3-662-67960-9.

[Boh25] Uta Bohnebeck. *Big Data and Machine Learning – Neural Networks*. Abgerufen am 26. Juli 2025. 2025. URL: https://aulis.hs-bremen.de/ilias.php?baseClass=ilrepositorygui&cmdNode=yj:nm&cmdClass=ilObjFileGUI&cmd=sendfile&ref_id=2257162.

[Ert25] Wolfgang Ertel. *Grundkurs Künstliche Intelligenz: Eine praxisorientierte Einführung*. Wiesbaden: Springer Vieweg, 2025. ISBN: 978-3-658-44954-4. URL: https://doi.org/10.1007/978-3-658-44955-1.

[www25] www.sonos.com. *Mono Sound und Stereo Sound im Vergleich Was ist der Unterschied?* Abgerufen am 18. Juli 2025. 2025. URL: https://www.sonos.com/de-de/blog/mono-vs-stereo-sound?srsltid=AfmBOoo5n0hV-tGCg_AsORZpsTs05D7_YoQpG4Yr38wtx9pev_s4VemK.

[BG03] Marina Bosi und Richard E. Goldberg. *Introduction to Digital Audio Coding and Standards*. New York: Springer Science, 2003. ISBN: 978-1-4613-5022-4. URL: https://www.pce-fet.com/common/library/books/34/1314_%5BMarina_Bosi,__Richard_E._Goldberg__(auth.)%5D_Intro(b-ok.org).pdf.

[Dig18a] Digilent Inc. *PYNQ-Z1 Board Specifications*. Abgerufen am 10. Juli 2025. 2018. URL: https://digilent.com/reference/programmable-logic/pynq-z1/start.

[TUL20] TUL Technology Unlimited. *PYNQ-Z2 Board Specifications*. Abgerufen am 10. Juli 2025. 2020. URL: https://www.tulembedded.com/FPGA/images/PYNQ-Z2_PA_v2_pp_20201209_STD.pdf.

[Dig18b] Digilent Inc. *PYNQ-Z1 Reference Manual*. Abgerufen am 10. Juli 2025. Digilent Inc. 2018. URL: https://digilent.com/reference/programmable-logic/pynq-z1/reference-manual.

[McC22] Cathal McCabe. *Step Datei Pynq-Z2*. Abgerufen am 10. Juli 2025. 2022. URL: https://discuss.pynq.io/t/pynq-z2-pcb-3d-step-files/2645/2.

[Pet19] Stefan Peters. *ELEGOO Smart Robot Car V3.0*. Abgerufen am 10. Juli 2025. 2019. URL: https://grabcad.com/library/elegoo-smart-robot-car-v3-0-1.

[Pet17] Alex Petuhov. *Logitech HD Pro Webcam C920*. Abgerufen am 10. Juli 2025. 2017. URL: https://grabcad.com/library/logitech-hd-pro-webcam-c920-1.

[rut20] ruthex. *ruthex Step-Data threaded inserts M2-M8*. Licensed under CC BY 4.0, Abgerufen am 10. Juli 2025. 2020. URL: https://www.thingiverse.com/thing:4682424/files.

[Gor+25] Frederic Goretzky u. a. *CAD-Modelle für das aktualisierte Fahrzeug*. Abgerufen am 10. Juli 2025. 2025. URL: https://gitlab.on.hs-bremen.de/labor-bredereke/esysp/sose25/CAD-Modelle.

[wwwa] www.gewinde-normen.de. *ISO Metric Coarse Thread DIN 13-1*. Abgerufen am 10. Juli 2025. URL: https://www.gewinde-normen.de/en/iso-coarse-thread.html.

[wwwb] www.gewinde-normen.de. *Gewinde im Fotobereich*. Abgerufen am 10. Juli 2025. URL: https://www.gewinde-normen.de/foto-gewinde.html.

[ELE20] ELEGOO. *ELEGOO Smart Robot Car v3.0 Manual*. Abgerufen am 10. Juli 2025. 2020. URL: https://eu.elegoo.com/de/blogs/arduino-projects/elegoo-smart-robot-car-kit-v3-0-plus-v3-0-v2-0-tutorial.

[Cha25] Charmast. *Charmast Powerbank*. Abgerufen am 10. Juli 2025. 2025. URL: https://www.amazon.de/Charmast-Powerbank-Ladeger%C3%A4t-kompatibel-mehr%EF%BC%88Wei%C3%9F%EF%BC%89/dp/B0B7RYGBXP?th=1.

[Jel24] Jelly Comb. *1080p HD Webcam mit Objektivdeckel*. Abgerufen am 10. Juli 2025. 2024. URL: https://jelly-cam.de/produkt/jelly-comb-1080p-hd-webcam-mit-objektivdeckel/.

[iJi24] iJiGui. *USB-Winkeladapter, 90-Grad-Winkelstecker*. Abgerufen am 10. Juli 2025. 2024. URL: https://www.amazon.de/dp/B0BR4RZ43P.

[Gmb22] Würth Elektronik eiSos GmbH & Co. KG. *Abstandhalter*. Abgerufen am 10. Juli 2025. 2022. URL: https://www.mouser.de/ProductDetail/Wurth-Elektronik/971200351?qs=wr8lucFkNMUwtasSz9e3JQ%3D%3D.

[KG] Hama GmbH & Co KG. *Richtmikrofon "RMN Uni", für Zubehörschuh*. Abgerufen am 10. Juli 2025. URL: https://www.hama.com/de/de/richtmikrofon-rmn-uni-fuer-zubehoerschuh-00004647?srsltid=AfmBOoqjyQriUxDbuxe4EIAJSgIwHL3ookehDpf46HPz0dDO8YNMpjMl.

[LAN24] LANMINGLEL. *USB 2.0 U-Adapter, 90 Grad Winkelstecker*. Abgerufen am 10. Juli 2025. 2024. URL: https://www.amazon.de/dp/B0D5BFJYQ2.

[na] n.a. *Leehitech USB Splitter Mini USB-Hub*. Abgerufen am 10. Juli 2025. URL: https://www.amazon.de/Leehitech-Verteiler-Multiport-Doppelstecker-kompatibel-Wei%C3%9F/dp/B0DYNRR6VZ?__mk_de_DE=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=WS62PAIF75W4&dib=eyJ2IjoiMSJ9.rg3cVvoVt19jbCiCVOD_XFSXeXHj_GRhwXZQzGbapISp3q3Kd0YoYYts5x-Ww0d4sTdGA4LFZXBRuwdMff7FDURUqpRQjO_Cb5Y0mx1S6Ow0YDqUBDhGmU-ufxxM_lio0q4z7ooRkMXfHnrgP6kvZU3eubQjIhZN3iX7phxX7Strb9R8ycqhDxwjRaZ7q27w.Eentqwy0D8hxjN_3OGr8QxBNNpgSRI69Jfsk4B-H08A&dib_tag=se&keywords=usb%2B2.0%2BTwin%2BHub&qid=1751535875&sprefix=usb%2B2.0%2Btwin%2Bhub%2Caps%2C94&sr=8-69&xpid=-pB7q6Z55URBK&th=1.

[OV23] Flavius Opriţoiu und Mircea Vlăduţiu. „Systolic Array Architecture for Educational Use“. In: *2023 27th International Conference on System Theory, Control and Computing (ICSTCC)*. 2023, S. 18–23. DOI: 10.1109/ICSTCC59206.2023.10308496.

[Li+17] Guanpeng Li u. a. „Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications“. In: *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*. 2017, S. 1–12.

[TG17] Cesar Torres-Huitzil und Bernard Girau. „Fault and Error Tolerance in Neural Networks: A Review“. In: *IEEE Access* 5 (2017), S. 17322–17341. DOI: 10.1109/ACCESS.2017.2742698.

[He+23] Yi He u. a. „Understanding and Mitigating Hardware Failures in Deep Learning Training Systems“. In: *Proceedings of the 50th Annual International Symposium*

on Computer Architecture. ISCA '23. Orlando, FL, USA: Association for Computing Machinery, 2023. ISBN: 9798400700958. DOI: 10.1145/3579371.3589105. URL: https://doi.org/10.1145/3579371.3589105.

[Dix+22] Harish Dattatraya Dixit u. a. *Detecting silent data corruptions in the wild*. 2022. arXiv: 2203.08989 [cs.AR]. URL: https://arxiv.org/abs/2203.08989.

[Ibr+20] Younis Ibrahim u. a. „Soft errors in DNN accelerators: A comprehensive review". In: *Microelectronics Reliability* 115 (2020), S. 113969. ISSN: 0026-2714. DOI: https://doi.org/10.1016/j.microrel.2020.113969. URL: https://www.sciencedirect.com/science/article/pii/S0026271420308003.

[Cai+24] Yulong Cai u. a. „Evaluation and Mitigation of Weight-Related Single Event Upsets in a Convolutional Neural Network". In: *Electronics* 13.7 (2024). ISSN: 2079-9292. DOI: 10.3390/electronics13071296. URL: https://www.mdpi.com/2079-9292/13/7/1296.

[Du+19] Boyang Du u. a. „On the Reliability of Convolutional Neural Network Implementation on SRAM-based FPGA". In: *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 2019, S. 1–6. DOI: 10.1109/DFT.2019.8875362.

[WA08] Fan Wang und Vishwani D. Agrawal. „Single Event Upset: An Embedded Tutorial". In: *21st International Conference on VLSI Design (VLSID 2008)*. 2008, S. 429–434. DOI: 10.1109/VLSI.2008.28.

[DM03] P.E. Dodd und L.W. Massengill. „Basic mechanisms and modeling of single-event upset in digital microelectronics". In: *IEEE Transactions on Nuclear Science* 50.3 (2003), S. 583–602. DOI: 10.1109/TNS.2003.813129.

[RS20] Tiago Mallmann Rohde und João Baptista dos Santos Martins. „Multi-Bit-Upset Memory Using New Error Correction Code Methodology". In: *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*. 2020, S. 1–4. DOI: 10.1109/LASCAS45839.2020.9069032.

[Rao+14] Parthasarathy M. B. Rao u. a. „Protecting SRAM-based FPGAs Against Multiple Bit Upsets Using Erasure Codes". In: DAC '14. San Francisco, CA, USA: Association for Computing Machinery, 2014, S. 1–6. ISBN: 9781450327305. DOI: 10.1145/2593069.2593191. URL: https://doi.org/10.1145/2593069.2593191.

[Ama+16] Motoki Amagasaki u. a. „An area compact soft error resident circuit for FPGA". In: *2016 International Conference on IC Design and Technology (ICICDT)*. 2016, S. 1–4. DOI: 10.1109/ICICDT.2016.7542046.

[For+20] Håkan Forsberg u. a. „Challenges in Using Neural Networks in Safety-Critical Applications". In: *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. 2020, S. 1–7. DOI: 10.1109/DASC50938.2020.9256519.

[Gai+16] Adrien Gaidon u. a. *Virtual Worlds as Proxy for Multi-Object Tracking Analysis*. 2016. arXiv: 1605.06457 [cs.CV]. URL: https://arxiv.org/abs/1605.06457.

[LV20] A. Levin und N. Vidimlic. *Improving Situational Awareness in Aviation: Robust Vision-Based Detection of Hazardous Objects*. Dissertation. Dissertation. 2020.

[Hen+20] M. Henne u. a. *Benchmarking Uncertainty Estimation Methods for Deep Learning with Safety-Related Metrics*. In SafeAI@AAAI, pp. 83–90. Workshop on Artificial Intelligence Safety, AAAI. 2020.

[Clu+20] J. M. Cluzeau u. a. *Concepts of Design Assurance for Neural Networks (CoDANN)*. Public Report Extract, EASA AI Task Force and Daedalean AG, Version 1.0. Released March 31, 2020. März 2020.

[BBM20] E. R. Balda, A. Behboodi und R. Mathar. „Adversarial Examples in Deep Neural Networks: An Overview“. In: *Deep Learning: Algorithms and Applications*. Hrsg. von W. Pedrycz und S.-M. Chen. Bd. 865. Studies in Computational Intelligence. Cham: Springer, 2020.

[DW11] Anand Dixit und Alan Wood. „The impact of new technology on soft error rates“. In: *IEEE International Reliability Physics Symposium Proceedings* (Mai 2011), 5B.4.1–5B.4.7. DOI: 10.1109/IRPS.2011.5784522.

[KBB20] Navid Khoshavi, Connor Broyles und Yu Bi. „Compression or Corruption? A Study on the Effects of Transient Faults on BNN Inference Accelerators“. In: *2020 21st International Symposium on Quality Electronic Design (ISQED)*. 2020, S. 99–104. DOI: 10.1109/ISQED48828.2020.9137006.

[Cou+16] Matthieu Courbariaux u. a. „Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1“. In: (Feb. 2016). DOI: 10.48550/arXiv.1602.02830.

[Ahm+24a] Mohammad Hasan Ahmadilivani u. a. „Special Session: Reliability Assessment Recipes for DNN Accelerators“. In: *2024 IEEE 42nd VLSI Test Symposium (VTS)*. 2024, S. 1–11. DOI: 10.1109/VTS60656.2024.10538707.

[AHL20] Tooba Arifeen, Abdus Sami Hassan und Jeong-A Lee. „Approximate Triple Modular Redundancy: A Survey“. In: *IEEE Access* 8 (2020), S. 139851–139867. DOI: 10.1109/ACCESS.2020.3012673.

[Sha+24] Yingzhao Shao u. a. „Research on Spaceborne Neural Network Accelerator and Its Fault Tolerance Design“. In: *Remote Sensing* 17 (Dez. 2024), S. 69. DOI: 10.3390/rs17010069.

[Tah+24a] Mahdi Taheri u. a. „AdAM: Adaptive Fault-Tolerant Approximate Multiplier for Edge DNN Accelerators“. In: *2024 IEEE European Test Symposium (ETS)*. 2024, S. 1–4. DOI: 10.1109/ETS61313.2024.10567161.

[Sei+10] N. Seifert u. a. „On the radiation-induced soft error performance of hardened sequential elements in advanced bulk CMOS technologies“. In: *2010 IEEE International Reliability Physics Symposium*. 2010, S. 188–197. DOI: 10.1109/IRPS.2010.5488831.

[Naz+24] Samira Nazari u. a. „FORTUNE: A Negative Memory Overhead Hardware-Agnostic Fault TOleRance TechniqUe in DNNs“. In: *2024 IEEE 33rd Asian Test Symposium (ATS)*. 2024, S. 1–6. DOI: 10.1109/ATS64447.2024.10915463.

[Tah25] Mahdi Taheri. „Reliability Assessment of Deep Neural Networks in Hardware Accelerators“. Englisch. Diss. Tallinn University of Technology, 2025. URL: https://digikogu.taltech.ee/et/Item/9cf79768-17bc-44ec-a828-e4ccf6cf93f1 (Abgerufen am 01. 07. 2025).

[Ruo+23] Annachiara Ruospo u. a. „A Survey on Deep Learning Resilience Assessment Methodologies“. In: *Computer* 56.2 (2023), S. 57–66. DOI: 10.1109/MC.2022.3217841.

[Tah+24b] Mahdi Taheri u. a. „SAFFIRA: a Framework for Assessing the Reliability of Systolic-Array-Based DNN Accelerators“. In: *2024 27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*. 2024, S. 19–24. DOI: 10.1109/DDECS60919.2024.10508925.

[Tah+23a] Mahdi Taheri u. a. „DeepAxe: A Framework for Exploration of Approximation and Reliability Trade-offs in DNN Accelerators“. In: *2023 24th International Symposium on Quality Electronic Design (ISQED)*. 2023, S. 1–8. DOI: 10.1109/ISQED57927.2023.10129353.

[Ria+20] Mohammad Riazati u. a. „DeepHLS: A complete toolchain for automatic synthesis of deep neural networks to FPGA“. In: *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 2020, S. 1–4. DOI: 10.1109/ICECS49266.2020.9294881.

[Tah+23b] Mahdi Taheri u. a. „APPRAISER: DNN Fault Resilience Analysis Employing Approximation Errors“. In: *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. 2023, S. 124–127. DOI: 10.1109/DDECS57882.2023.10139468.

[Ahm+24b] Mohammad Hasan Ahmadilivani u. a. „A Systematic Literature Review on Hardware Reliability Assessment Methods for Deep Neural Networks“. In: *ACM Comput. Surv.* 56.6 (Jan. 2024). ISSN: 0360-0300. DOI: 10.1145/3638242. URL: https://doi.org/10.1145/3638242.

[SGA18] Christoph Schorn, Andre Guntoro und Gerd Ascheid. „Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators“. In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2018, S. 979–984. DOI: 10.23919/DATE.2018.8342151.

[SGA19] Christoph Schorn, Andre Guntoro und Gerd Ascheid. „An Efficient Bit-Flip Resilience Optimization Method for Deep Neural Networks“. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, S. 1507–1512. DOI: 10.23919/DATE.2019.8714885.

[Gav+22] G. Gavarini u. a. „Open-Set Recognition: an Inexpensive Strategy to Increase DNN Reliability“. In: *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2022, S. 1–7. DOI: 10.1109/IOLTS56730.2022.9897805.

[Ené23] Daniel Enériz. *PYNQ Z2 pinout V2*. 5. Mai 2023. URL: https://discuss.pynq.io/t/pynq-z2-pinout/4256/5 (Abgerufen am 30. 07. 2025).

[PyTa] PyTorch, The Linux Foundation. *PyTorch*. URL: https://pytorch.org/ (Abgerufen am 25. 07. 2025).

[NVIb] NVIDIA Corporation. *CUDA GPU Compute Capability*. URL: https://developer.nvidia.com/cuda-gpus (Abgerufen am 25. 07. 2025).

[NVIc] NVIDIA Corporation. *LEGACY CUDA GPU Compute Capability*. URL: https://developer.nvidia.com/cuda-legacy-gpus (Abgerufen am 25. 07. 2025).

[PyTb] PyTorch, The Linux Foundation. *Get Started*. URL: https://pytorch.org/get-started/locally/ (Abgerufen am 25. 07. 2025).

A. Pinbelegung auf dem PYNQ-Z2-Board

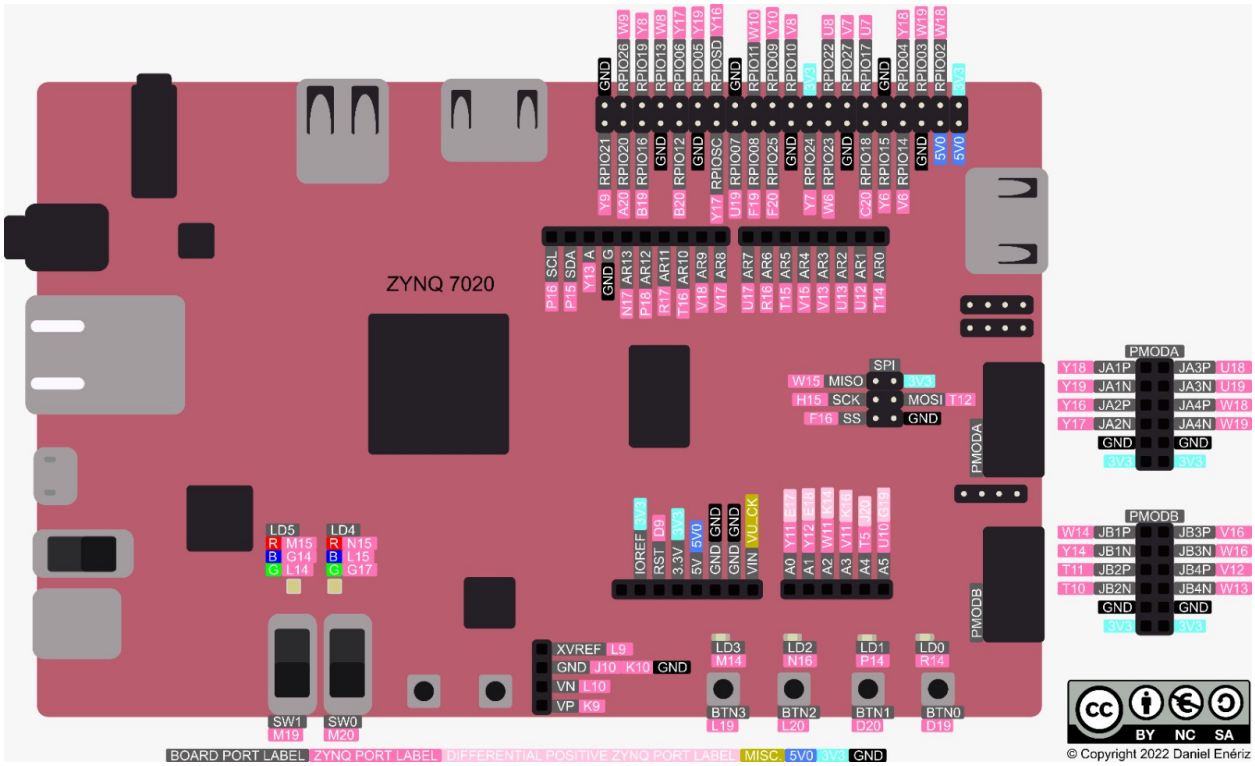


Abbildung A.1.: Pinbelegung des PYNQ-Z2-Boards [Ené23]

B. Anleitung zur Nutzung der Fahrsteuerung

geschrieben von Alexander Hoegen-Jupp und Philipp Hennken

Im Folgenden wird beschrieben, wie die Fahrsteuerung eingerichtet sowie genutzt wird.

B.1. Voraussetzungen

Die Fahrsteuerung ist für ein ELEGOO Smart Robot Car Kit V3.0 mit einem Xilinx PYNQ-Z2 Board ausgelegt. Es besteht die Möglichkeit, die Fahrsteuerungssoftware auf einem Mikrocontroller oder single board computer (SBC) auszuführen, welcher Linux und Python unterstützt. Allerdings ist diese im Rahmen des Projekts nicht getestet worden.

B.2. Überblick über die Architektur

Die Fahrsteuerung besteht aus zwei Komponenten, einmal den **hardwarenahen** Teil (auch *engine* genannt) und einmal den *nutzernahen* Teil (auch *server*) genannt.

Der Nutzer sendet die gewünschten Fahranweisungen via Hypertext Transfer Protocol (HTTP) an eine Representational State Transfer (REST)-Application Programming Interface (API). Diese führt die gewünschten Aktionen über Funktionsaufrufe von dem engine aus.

Für Testzwecke wird ein Front-End zur Verfügung gestellt, mit dem das Fahrzeug grafisch gesteuert werden kann.

B.3. Aufbau des Fahrzeugs

Das Fahrzeug muss über mindestens über zwei Motoren verfügen. Die Motoren müssen links und rechts vom Fahrzeug angebracht sein und verkabelt werden. Zudem müssen die Motoren an einen L298N Motortreiber angeschlossen sein.

B.4. Verbindung mit dem Board

Die einfachste Art sich mit dem PYNQ-Z2 Board zu verbinden ist über secure shell (SSH). Dafür muss das Board über den Local Area Network (DHCP)-Port an einen Router oder Dynamic Host Configuration Protocol (DHCP)-fähigen Switch angeschlossen werden. Im Labor stehen dafür zwei Switches zur Verfügung, einer auf Basis eines Raspberry Pis und ein kommerzieller. Während der Versuche wurde hauptsächlich der kommerzielle Switch verwendet. Erfahrungsgemäß vergibt

der Switch dem ersten Gerät, welches mit ihm verbunden wird, die Adresse `192.168.10.130`, dementsprechend ist es empfehlenswert das PYNQ-Z2 Board immer als erstes Gerät an den Switch anzuschließen, damit die IP-Adresse gleich bleibt. Das Gerät, welches sich mit dem PYNQ-Board verbinden soll, muss nun ebenfalls an den Switch angeschlossen werden.

Mit dem Befehl `ssh xilinx@192.168.10.130` lässt sich eine SSH Verbindung zum Board aufbauen. Falls mehrere SSH-keys auf dem Entwicklungsgerät vorhanden sind, wird ein *Too many Authentication Failures* Fehler auftreten. In diesem Fall kann dieser Befehl `ssh -o PreferredAuthentications=password xilinx@192.168.10.130` verwendet werden, welcher dieses Problem behebt. Unter Umständen kann es sein, dass die IP-Adresse in den Befehlen angepasst werden muss. Das Passwort für den `xilinx` User ist `xilinx`. Bei der ersten Verbindung kann die folgende Nachricht auftauchen:

```
1 This key is not known by any other names.
2 Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

Sollte dies erscheinen, einfach `yes` eingeben und weitermachen.

Nun sollte eine SSH-Verbindung zwischen den beiden Geräten hergestellt worden sein und man hat nun Terminal-Zugriff auf das PYNQ-Board. Mit dem `scp`-Befehl lassen sich Dateien per SSH zwischen den beiden Geräten kopieren.

Im root des Repos kann der folgende Befehl ausgeführt werden, um das Repo auf das Board zu kopieren: `scp -r -o PreferredAuthentications=password ./ xilinx@192.168.10.130:~/`

Dieser Befehl kopiert den Inhalt des Repos in das Home-Verzeichnis des xilinx-Nutzers.

B.5. Einrichtung des Board

Auf dem Board müssen zwei Dienste eingerichtet werden. Zum einen ein Wireless Local Area Network (WLAN)-AP und einmal ein Python Server. Die Einrichtung des APs wird in Anhang C beschrieben und die des Python Servers in Anhang D.

B.6. Verbinden mit dem WLAN des Boards

Wenn das Board eingerichtet worden ist, kann man mit einem WLAN-fähigem Endgerät sich mit dem WLAN des Boards verbinden. Die Verbindungsdaten hierfür lauten

```
1 WLAN SSID: pynq
2 Password: pynq1234
```

B.7. Nutzung des Front-Ends

Das Front-End bietet eine intuitive Bedienung mithilfe eines Joysticks an. Durch das Bewegen des Joysticks werden im Hintergrund die passenden HTTP-Requests an den Server gesendet. Es kann durch den Aufruf von `192.168.50.1:5000` auf einem Endgerät, welches mit dem WLAN des Boards verbunden ist, erreicht werden.

C. Einrichten eines WLAN-AcessPoints (AP)

geschrieben von Tim Ranft

Im Folgenden ist beschrieben, wie die Einrichtung eines AP auf einer Basisinstallation des PYNQ-Z2-Boards erfolgt. Hierfür wird der USB Wireless Adapter am USB-Port des Boards (auch an USB-Splitter möglich) angeschlossen. Gerade bei der Installation am Splitter sollte auf ausreichend Halt geachtet werden, da sich die Verbindung zurücksetzt, wenn dieser einen Wackelkontakt aufweist.

C.1. Einrichtung

geschrieben von Tim Ranft

```
# Verbinden über eine ssh verbindung zu Board
ssh xilinx@pynq
[ODER]
ssh xilinx@192.168.XXX.XXX <- mit aktueller Adresse am Router
```

```
# System und Pakete aktualisieren ist immer eine gute idee um zu beginnen
sudo apt update
sudo apt upgrade
```

```
# USB-Geräte anzeigen
lsusb
```

Erwartete Ausgabe ist hierbei mindestens der *Ralink Technology, Corp. RT5370 Wireless Adapter*. Ist dieser nicht aufgeführt, so ist der USB-WLAN-Adapter nicht angeschlossen oder nicht erkannt worden.

```
# Treiber prüfen
lsmod | grep rt2800usb
```

Ist der nötige Treiber bereits installiert, so wird mindestens einmal *rt2800usb* als Ausgabe angezeigt

```
# Unterstützte Interface-Modi des WLAN-Chips anzeigen
iw list | grep -A 10 "Supported interface modes"
```

Ein geeigneter USB-Adapter sollte bei dieser Ausgabe mindestens *AP* aufgelistet haben.

```
# Netzwerkinterfaces anzeigen
ip link show
```

Die erwartete Ausgabe ist mindestens die Auflistung der Schnittstelle *wlan0*

```
# IP-Adresse für AP konfigurieren
sudo ip addr add 192.168.50.1/24 dev wlan0
sudo ip link set wlan0 down
sudo ip addr flush dev wlan0
sudo ip link set wlan0 up
```

Mit diesen Befehlen wird der Adressraum festgelegt, in welchem der Adapter später den AP zur Verfügung stellt.

```
# Validieren, dass wlan0 den Adressbereich übernommen hat
ip addr show wlan0
```

```
# Hostapd konfigurieren
sudo nano /etc/hostapd/hostapd.conf
which hostapd
```

Das Paket *Hostapd* sorgt dafür, dass auf der Schnittstelle ein AP angeboten werden kann. Das Verzeichnis der Konfigurationsdatei muss bereits existieren! andernfalls ist das ein Zeichen dafür, dass das Paket nicht oder fehlerhaft installiert ist. Bei letzterem Befehl sollte mindestens der Pfad */usr/sbin/hostapd* als Ausgabe erscheinen.

```
# Hostapd installieren (erforderlich, falls die notwendigen Abhängigkeiten im vorherigen Schritt
    nicht gegeben sind)
sudo apt update
sudo apt install hostapd
sudo nano /etc/hostapd/hostapd.conf
```

Die Konfigurationsdatei sollte folgendermaßen beschrieben sein:

```
interface=wlan0
driver=nl80211
ssid=zynq_ap
hw_mode=g
channel=6
auth_algs=1
wmm_enabled=1
wpa=2
wpa_passphrase=pynq1234
wpa_key_mgmt=WPA-PSK
rsn_pairwise=CCMP
```

Die *ssid* ist der Name des Netzwerks, auf dem der AP sendet und kann frei gewählt werden. Die *wpa_passphrase* kann ebenfalls frei gewählt werden, muss jedoch mindestens acht Zeichen enthalten.

```
# Hostapd den Pfad der Konfigurationsdatei mitgeben
sudo hostapd /etc/hostapd/hostapd.conf
```

```
# Start-Skript und Systemd-Service anlegen
sudo nano /usr/local/bin/start_ap.sh
sudo chmod +x /usr/local/bin/start_ap.sh
```

```
sudo nano /etc/systemd/system/ap.service
sudo systemctl enable ap.service
sudo systemctl start ap.service
sudo systemctl status ap.service
```

Der Status des angelegten Services sollte **active (running)** sein und keine Fehler in der Ausgabe aufweisen. Der Hinweis **Cannot open RFKILL control device** kann ignoriert werden.

Die nachfolgende Befehlsabfolge mit einer Neuinstallation des DNS-Dienstes gekoppelt. Dieser ist zum Zeitpunkt der Konfiguration bereits einmal eingerichtet worden, um mittels WLAN-Adapter direkt auf ein Netzwerk (Client-Modus) zugreifen zu können. Dies sorgte für eine Fehlkonfiguration, welche sich in Form eines bereits genutzten Port durch den DNS-Dienst abbildete. Daher wurde eine entsprechende Neuinstallation vorgenommen.

```
# (Optional) System neu starten und DNS-Dienst installieren
sudo reboot
sudo apt install dnsmasq
```

```
# Konfigiration der dnsmasq Umgebung für die dynamische Adressverteilung via DNS
sudo nano /etc/dnsmasq.conf
```

Der Inhalt dieser Datei sollte komplett auskommentiert sein und mit folgenden Zeilen ergänzt werden:

```
port=0
interface=wlan0
bind-interfaces
dhcp-range=192.168.50.10,192.168.50.100,12h
```

Der DNS-Dienst stellt durch diese Konfiguration auf dem entsprechenden Interface ein Adressbereich zur Verfügung. Angemeldete Geräte erhalten eine valide Address-Lease für zwölf Stunden.

```
# Neustarten und Validieren des DNS-Service
sudo systemctl restart dnsmasq
sudo systemctl status dnsmasq
```

Der DNS-Service sollte auch den Status *active (running)* aufweisen. Bei erfolgreichem Einrichten, sollte mindestens die Zeile *DHCP, IP range 192.168.50.10 – 192.168.50.100, lease time 12h* im unteren Teil der Ausgabe erscheinen.

```
# Gerät neustarten und Anmeldungen überwachen
sudo reboot
```

Nach einem Neustart sollten es möglich sein, sich mit einem oder mehreren Geräten auf das Board zu verbinden. Die Geräte erhalten eine Adresse aus dem zuvor angelegten Adressbereich und können nun ohne Kabel mit dem Netzwerk kommunizieren, um beispielsweise eine Terminal-Verbindung aufzubauen.

Mit dem nachfolgenden Befehl kann eine Verteilung einer Address-Lease innerhalb der letzten fünf Minuten bestätigt werden. Wenn sich kein Gerät innerhalb der letzten fünf Minuten verbunden hat, ist die Ausgabe – *No entries* –.

```
sudo journalctl -u dnsmasq --since "5 minutes ago"
```

D. Einrichtung des Python-Servers als Service

geschrieben von Tim Ranft

Um die Endpunkte des Servers automatisch bei Systemstart auszuführen, wurde der **myserver.service** angelegt, welche das Script *run_server.sh* ausführt. Das Script ist im GitLab verfügbar. Das Vorgehen zum Anlegen dieses Services ist für eine Replikation auf einem anderen Board im Folgenden beschrieben. Anzumerken ist, dass der Server auch lokal ausgeführt werden kann, jedoch nur wenn der Service nicht gestartet ist, da er sonst den Port 5000 bereits belegt.

```
# Anlegen der Deamon-Datei
sudo nano /etc/systemd/system/myserver.service
```

```
# Inhalt der Datei
[Unit]
Description=Fahrsteuerung-Server
After=network.target udev.service

[Service]
Type=simple
# User und Group entfernen, damit root gestartet wird
WorkingDirectory=/home/xilinx/Fahrsteuerung_Server
ExecStart=/bin/bash -lc '/home/xilinx/Fahrsteuerung_Server/run_server.sh'
Restart=always
RestartSec=5
StandardOutput=journal
StandardError=journal

[Install]
WantedBy=multi-user.target
```

```
# Anwenden der Änderungen und Starten des Services
sudo systemctl daemon-reload
sudo systemctl start myserver.service
```

D.1. Manuelles Starten des Servers in der richtigen Umgebung

Für die Nutzung der Pins und dem Beschreiben des Overlays sind *sudo*-Rechte erforderlich. Mit dem Flag -E wird die Umgebung des ausführenden Nutzers erhalten, was für die Importe der Basisbibliotheken des Boards relevant ist.

```
# Manueller Start der Datei server.py
cd ~/Fahrsteuerung_Server/
sudo -E /usr/local/share/pynq-venv/bin/python3 server.py
```

D.2. Paketanforderungen an die Umgebung

Die bereits vorhanden Python3-Paket-Bibliothek wurde um *Flask* erweitert. Dies kann mit folgenden Befehlen reproduziert werden:

```
# Nachinstallation flask und flask-cors
sudo apt install python3-flask
python3 -m pip install flask_cors
```

Für die Debug ausgaben des Python Scripts kann folgender Befehl genutzt werden:

```
sudo journalctl -u myserver.service -f
```

E. Installationsanleitung: FINN und Xilinx Toolchain unter WSL2

Zielsetzung Diese Anleitung beschreibt die vollständige und reproduzierbare Einrichtung einer Entwicklungsumgebung für das Framework `FINN`, welches die hardwarebeschleunigte Inferenz quantisierter neuronaler Netzwerke (QNNs) ermöglicht. Die Installation erfolgt unter `Ubuntu 24.04 LTS` innerhalb von `WSL2` (Windows Subsystem for Linux) unter Windows 11. Die Anleitung richtet sich auch an Anwenderinnen und Anwender ohne tiefgreifende Kenntnisse im Umgang mit Linux oder WSL2.

Voraussetzungen

- Windows 11 mit aktiviertem WSL2
- Docker Desktop mit aktivierter WSL2-Integration
- Ubuntu 24.04 LTS unter WSL2
- Aktivierte Hardware-Virtualisierung im BIOS/UEFI

Komponente	Version / Auswahl
Host-Betriebssystem	Windows 11 Pro 23H2
Linux-Umgebung	Ubuntu 24.04 LTS (WSL2)
FINN-Framework	GitHub, Branch `main` (Stand: 19.06.2025)
Vivado/Vitis	Version 2024.2

Tabelle E.1.: Verwendete Systemkomponenten

WSL2 und Ubuntu installieren

1. Öffnen Sie die Windows PowerShell mit Administratorrechten.
2. Führen Sie folgenden Befehl aus, um WSL2 mit Ubuntu 24.04 zu installieren:

```
wsl --install -d Ubuntu-24.04
wsl --set-default-version 2

```

3. Starten Sie anschließend Ubuntu über das Startmenü, richten Sie einen Benutzernamen und ein Passwort ein. Führen Sie danach ein Systemupdate durch und installieren Sie die benötigten Werkzeuge:

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y git unzip

```

Docker Desktop einrichten

1. Installieren Sie Docker Desktop für Windows: https://www.docker.com/products/docker-desktop/
2. Aktivieren Sie in den Einstellungen unter `Settings > General` die Option: **Use the WSL 2 based engine**.
3. Aktivieren Sie unter `Settings > Resources > WSL Integration` die Integration für die zuvor installierte Ubuntu-Distribution.
4. Testen Sie die erfolgreiche Integration in der Ubuntu-Konsole mit folgenden Befehlen:

```
docker version
docker run hello-world
```

Installation von Vitis/Vivado in WSL2

1. Laden Sie den Linux Unified Installer von der offiziellen Webseite herunter: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vitis/2024-2.html
2. Kopieren Sie die Installationsdatei in Ihr Home-Verzeichnis unter Ubuntu:

```
cp /mnt/c/Users/<BENUTZERNAME>/Downloads/<DATEINAME>/
chmod +x Xilinx_Unified_2024.2_*.bin
```

3. Erstellen Sie das Zielverzeichnis für die Installation:

```
sudo mkdir -p /opt/Xilinx
sudo chown $USER /opt/Xilinx
```

4. Starten Sie nun den Installer:

```
./Xilinx_Unified_2024.2_*.bin
```

5. Während der Installation:
   - Zielverzeichnis: `/opt/Xilinx`
   - Ausgewählte Komponenten: `Vivado`, `Vitis`, `XRT`
   - Zielgerät: Aktivieren Sie `Zynq-7000 Series` (z. B. für PYNQ-Z2)

FINN installieren

1. Fügen Sie die folgenden Umgebungsvariablen Ihrer `/.bashrc`-Datei hinzu:

```
export FINN_XILINX_PATH="/opt/Xilinx"
export FINN_XILINX_VERSION="2024.2"
```

Übernehmen Sie die Änderungen:

```
source ~/.bashrc

```

2. Klonen Sie das FINN-Repository von GitHub und wechseln Sie in das Verzeichnis:

```
git clone https://github.com/Xilinx/finn.git
cd finn

```

3. Führen Sie zur Überprüfung der Installation den Schnelltest aus:

```
./run-docker.sh quicktest

```

F. Training des Modells für die Personenerkennung

geschrieben von Keno Albers, Jonas Broda, Julius Böttger, Jannik Hennicke, Nils van Rijsinge

Hardware	Trainingszeit in s					Durchschnitt in s
CPU	396.837	361.716	507.018	471.086	395.973	426.526
Nvidia GPU	13.291	13.487	13.441	13.316	13.169	13.3408

Tabelle F.1.: Trainingszeiten auf verschiedener Hardware

F.1. Installation von Nvidia CUDA und Integration mit torch

Nvidia CUDA ist ein Toolkit mit welchem hochperformante Umgebungen entwickelt und optimiert werden können. In unserem Kontext wird CUDA in der Python-Bibliothek PyTorch [PyTa] verwendet, um KI-Training auf parallel-performanten Chips durchzuführen.

Für die Installation und Verwendung von Nvidia CUDA mit PyTorch gelten folgende Voraussetzungen:

1. Das zu verwendende System besitzt eine Nvidia-Grafikkarte mit „*CUDA-Cores*“. Die neuesten Chip-Modelle unterstützen die aktuellen CUDA Versionen [NVIb], für ältere Modelle müssen Legacy-Versionen [NVIc] verwendet werden.
2. Es müssen die proprietären Treiber und CUDA-Pakete zum Ansprechen dieser „*CUDA-Cores*“ installiert sein. Die offiziellen Treiber sind unter `https://developer.nvidia.com/cuda-toolkit-archive` (Stand 25.07.2025) zu finden.
3. Es muss PyTorch zu der passenden CUDA Version installiert sein (siehe [PyTb] für eine ausführliche Anleitung).

Wenn diese Voraussetzungen erfüllt sind kann das Training über die GPU passieren und die deutliche höhere Rechenleistung der Grafikkarte benutzt werden.

G. Überblick über die Git-Repositories

geschrieben von Jan Bredereke

Dieser Anhang beschreibt, welche Git-Repositories es gibt, worum es darin geht, und wo sie zu finden sind. Wenn möglich, sind sie öffentlich zugänglich gemacht.

In allen Repositories beschreibt eine Datei `README.md`, die auf der obesten Ebene liegt, den Inhalt.

G.1. Öffentliche Repositories

Drei der im aktuellen Projektdurchgang verwendeten Repositories wurden nach Projektende auf GitHub kopiert. Die Commit-Historie wurde dabei nicht mitkopiert.

URL bei GitHub	**Beschreibung**
https://github.com/JanBredereke/esysp25_end2end-tests	Hier stehen der Code und die Trainingsdaten zu Kap. 3 „Personenerkennung“.
https://github.com/JanBredereke/esysp25_fahrsteuerung	Hier steht der Code zu Kap. 5 „Fahrsteuerung“.
https://github.com/JanBredereke/esysp25_sprachsteuerung	Hier stehen der Code und die Trainingsdaten zu Kap. 7 „Audioerkennung per neuronalem Netz“. Das Unterverzeichnis `finnModel/` wurde hier vor Veröffentlichung entfernt, da es im Bericht nicht beschrieben wird, und da die Rechte daran nicht klar sind.

G.2. Hochschulinterne Repositories

Drei weitere Repositories können nicht allgemein veröffentlicht werden, da die nötigen Rechte dafür fehlen. Sie enthalten u.a. Material, das zwar aus dem Internet heruntergeladen werden darf, das aber nicht von uns öffentlich abrufbar gemacht werden darf. Zwei weitere Repositories wurden angelegt, sind aber einfach komplett leer.

Sämtliche zum aktuellen Projektdurchgang gehörigen Git-Repositories, auch die obigen öffentlichen Repositories, sind hochschulintern unter der folgenden URL zu finden. Dieser Zugang ist nur für Hochschulangehörige mit dem Rechenzentrums-Passwort möglich. Außerdem muss Prof. Bredereke den konkreten Zugang freigeschaltet haben.

https://gitlab.on.hs-bremen.de/labor-bredereke/esysp/sose25/

Name des Repositories	Beschreibung
`CAD-Modelle`	Mechanische Konstruktionszeichnung des Fahrzeugs. Vergleiche Kap. 8.2.5 „Fertigung und Materialwahl“ in Kap. 8 „Elektronik und Mechanik“. Hier wurden Konstruktionszeichnungen aus anderen Quellen für den Fahrzeugbausatz, für das PYNQ-Z2-Board, für die Webcam und für M3-Einpressgewinde integriert.
`gestenerkennung`	Vergleiche Kap. 4 „Gestenerkennung“. Hier wurden Frameworks aus anderen Quellen verwendet, wobei der Code nicht klar abgrenzbar ist.
`literatur-rechenfehler-in-nn`	Vergleiche Kap. 9 „Auswirkungen von Rechenfehlern im neuronalen Netz abschätzen“. Dies sind heruntergeladene Kopien der im Literaturverzeichnis aufgeführten Werke. Das Literaturverzeichnis gibt an, wie man sie sich beschaffen kann.
`systemarchitektur`	Dieses Repository ist komplett leer. Es war für Kap. 2 „Überblick über die Systemarchitektur“ vorgesehen, wurde aber nicht benötigt.
`protokolle`	Dieses Repository hätte die wöchentlichen internen Besprechungsprotokolle enthalten können. Es ist allerdings komplett leer.