FernUniversität in Hagen
Fakultät für Mathematik und Informatik

AIG
Artificial Intelligence Group

Verifikation standardisierter Referenzprozesse zur Entwicklung von ML-Komponenten im Automobilkontext bezüglich Anwendbarkeit im sicherheitskritischen Kontext und Unterstützung einer möglichen Sicherheitsargumentation

Masterarbeit

zur Erlangung des Grades eines Master of Science (M.Sc.)
im Studiengang Informatik

vorgelegt von
Lukas Bergmann

Erstgutachter:	Prof. Dr. Matthias Thimm Artificial Intelligence Group
Betreuer:	Andreas Mertin Volkswagen AG

Erklärung

Ich erkläre, dass ich die Masterarbeit selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder graphische Darstellungen. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Arbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.

	Ja	Nein
Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	☒	☐
Der Veröffentlichung dieser Arbeit auf der Webseite des Lehrgebiets Künstliche Intelligenz stimme ich zu.	☒	☐
Der Text dieser Arbeit ist unter einer Creative Commons Lizenz (CC BY-SA 4.0) verfügbar.	☒	☐
Der Quellcode ist unter einer GNU General Public License (GPLv3) verfügbar.	☒	☐
Die erhobenen Daten sind unter einer Creative Commons Lizenz (CC BY-SA 4.0) verfügbar.	☒	☐

Hildesheim, 08.02.2024 L. Beg

(Ort, Datum) (Unterschrift)

Zusammenfassung

Autonom fahrende Fahrzeuge sollen den Straßenverkehr sicherer machen und verschiedenen benachteiligten Gruppen individuelle Mobilität ermöglichen. Dabei wird *Machine Learning (ML)* als eine der Kerntechnologien angesehen, um diese zu realisieren. ML unterscheidet sich jedoch stark von klassischer Software- und Systementwicklung, weswegen mit dem *Automotive Software Process Improvement and Capability Determination (ASPICE)* Standard in der Version 4.0 neue spezielle Referenzprozesse für die Entwicklung von ML-Komponenten und das Datenmanagement eingeführt wurden, deren Anwendbarkeit mit dieser Masterarbeit verifiziert wurde. Dazu wurde mittels ML eine Verkehrszeichenerkennung entwickelt und die Referenzprozesse dabei praktisch umgesetzt. Weiter wurde gezeigt, wie die Anwendung der standardisierten Referenzprozesse eine mögliche Sicherheitsargumentation gemäß dem Stand der Forschung in vielen Aspekten unterstützen kann. Damit wurde bestätigt, dass die Erreichung eines Prozessfähigkeitslevels von mindestens eins als einer der Nachweise zur Argumentation der Sicherheit einer auf ML basierenden Funktionalität, in zum Beispiel einem autonom fahrenden Fahrzeug, angesehen werden kann. Auch wenn die Umsetzung der Referenzprozesse somit eine Sicherheitsargumentation signifikant unterstützen kann, ist dies jedoch keineswegs als alleiniger Nachweis für die Sicherheitsargumentation ausreichend, welche in hohem Maße von den definierten Anforderungen an das System abhängt und auch ethische Bedenken adressieren sollte.

Abstract

Fully automated vehicles have the capability to make road traffic safer and support individual mobility of various disadvantaged groups. Machine Learning (ML) is seen as a key technology to achieve this. However, ML is inherently different to classical software and system development. To address this, new reference processes for the development of ML components as well as ML data management were standardized within the 4th version of the Automotive Software Process Improvement and Capability Determination (ASPICE) standard. Within this thesis, the applicability of these reference processes was proven through an experimental development of a traffic sign detection based on ML. Furthermore, this thesis highlights how the application of the standardized reference processes may support the safety argumentation in accordance with the state of research in many aspects. Nevertheless, the implementation of the reference processes alone is not sufficient as a standalone approach to the safety argument, as further aspects like the defined requirements as well as ethical aspects need to be considered.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ACWG Assurance Case Working Group

ASIL Automotive Safety Integrity Level

ASPICE Automotive Software Process Improvement and Capability Determination

BP Base Practice

CBS contextual block separable

CNN Convolutional Neural Network

CSP cross stage parial

DL Deep Learning

E/E Systemen elektrischen und/oder elektronischen Systemen

GP Generic Practice

GSN Goal Structuring Notation

GTSRB German Traffic Sign Recognition Benchmark

KI künstliche Intelligenz

ML Machine Learning

MLE Machine Learning Engineering

ODD Operational Design Domain

PA Process Attribute

PAM Process Assessment Model

PRM Process Reference Model

QMC Quality Management Center

ReLU Rectified Linear Unit

SAE Society of Automotive Engineers

SPPF spatial pyramid pooling fast

SUP Supporting

tanh Tangens Hyperbolicus

VDA Verband der Automobilindustrie

YOLO you only look once

1. Einleitung

Bei Verkehrsunfällen in Deutschland sterben jährlich mehrere tausend Menschen. Gesetzliche Änderungen sowie technologischer und medizinischer Fortschritt haben dabei in den letzten 50 Jahren bereits zu einer deutlichen Verringerung der im Straßenverkehr tödlich Verunglückten geführt [(De23]. Abb. 1 zeigt den Verlauf der Todesfälle im Straßenverkehr pro Jahr und hebt markante gesetzliche Änderungen hervor. Darüber hinaus haben die Europäische Union und die Vereinten Nationen sich zum Ziel gesetzt, die Anzahl der Verkehrstoten bis 2030 zu halbieren [Kom23]. Dies soll unter anderem durch den Einsatz autonom fahrender Fahrzeuge, deren Entwicklung eine große Herausforderung in der Automobilindustrie darstellt, erreicht werden. So ist der Anspruch an vollständig autonome Fahrzeuge, welche in jeder Situation alle Aspekte der Fahraufgabe

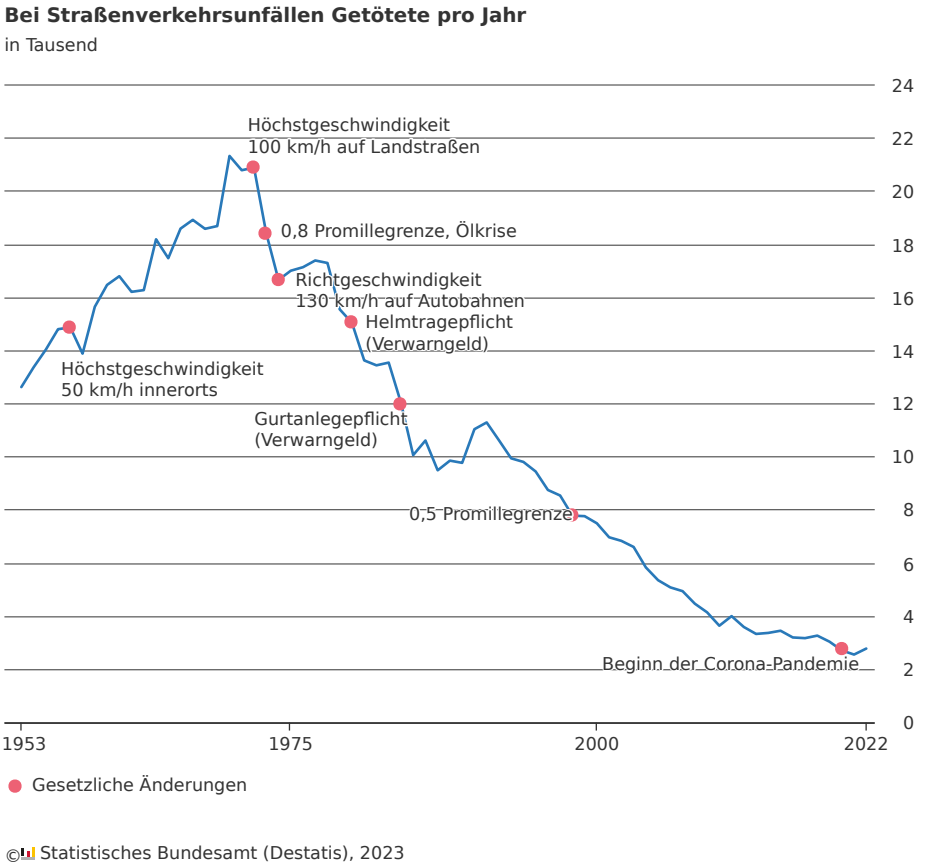


Abbildung 1: Anzahl Straßenverkehrstote in Deutschland pro Jahr, Bildquelle [(De23]

übernehmen und somit ohne einen Fahrer auskommen, dass sie weniger Unfälle erzeugen als es der Normalfahrer statistisch tut. Weiterhin sollen autonom fahrende Fahrzeuge benachteiligten Gruppen, welche nicht in der Lage sind selber ein Fahrzeug zu führen, die Möglichkeit individueller Mobilität verschaffen. Darüber hinaus sollen sie auch eine effizientere Nutzung von Rohstoffen im Straßenverkehr, beispielsweise durch Car-Sharing-Angebote oder die Einbeziehung von Daten über den Verkehrsfluss in die Routenplanung, erzielen [MRR+15].

Erste statistische Analysen, wie in [DLGZ+23] durchgeführt, bestätigen, dass hoch automatisierte selbstfahrende Fahrzeuge in der Lage sind, die Anzahl an Verkehrsunfällen gegenüber einem menschlichen Fahrer signifikant zu verringern. Die für die autonome Fahrt benötigten großen Datenmengen, welche das Fahrzeug während der Fahrt über seine Umgebung und seinen Fahrzustand sammelt, sollen dabei durch den Einsatz von *Machine Learning (ML)* verarbeitet werden und als Eingabe für die Trajektorienplanung des Fahrzeuges dienen. Diese plant die Längs- und Querbeschleunigung für einen festgelegten zeitlichen Vorhersagehorizont unter Berücksichtigung der Fahrzeugphysik sowie gesetzlichen Grenzen und dient als Zielwert für die Steuerung der Fahrzeugaktorik. Die Fahrzeugsteuerung hängt somit direkt von der Qualität der Umfelderfassung ab, weswegen ML als eine Kerntechnologie angesehen wird, um die Entwicklung vollständig autonom fahrender Fahrzeuge zu ermöglichen. Dabei besitzen insbesondere auf ML basierende Umfelderkennungsfunktionen eine hohe Sicherheitskritikalität, denn sollten Objekte in der Umgebung des Fahrzeuges falsch oder gar nicht erkannt werden, so kann es sein, dass das Fahrzeug sein Fahrverhalten nicht angemessen anpasst und es zu Unfällen kommt [DHH23, MRR+15, SQC17, SKS+20]. Der Fahrer ist dabei ab dem durch die *Society of Automotive Engineers (SAE)* standardisierten Automatisierungslevel 3 [On-21] nicht mehr in der Haftung für Unfälle, welche während des autonomen Betriebs entstehen, denn das Fahrzeug hat den Fahrer aktiv aufzufordern, die Kontrolle wieder zu übernehmen. Bei einem vollständig autonom fahrenden Fahrzeug gemäß SAE Level 5 [On-21] hat das Fahrzeug sogar jegliche Situation zu kontrollieren und es existiert kein Fahrer mehr. Der Hersteller muss die Korrektheit und Sicherheit des Fahrzeuges somit über eine ausführliche Sicherheitsargumentation gewährleisten und ist im Rahmen der Produkt- und Produzentenhaftung bei einem Unfall haftbar [Hey19, MRR+15]. Der Aufbau und die Funktionsweise von Komponenten, die auf ML basieren, unterscheiden sich dabei stark von klassischer Software, weswegen sich klassische Methoden der Sicherheitsargumentation innerhalb der Automobilindustrie, wie zum Beispiel in der ISO 26262 [ISO18] standardisiert, nur bedingt auf ML-Komponenten anwenden lassen. Weiterhin ist die Entscheidungsfindung von ML-Komponenten für uns Menschen häufig nicht im Detail nachvollziehbar, was eine Sicherheitsargumentation zusätzlich erschwert und das Vertrauen in die Technologie mindert [DHH23, SQC17]. Wie eine hinreichende Sicherheitsargumentation für automatisierte Fahrzeuge, welche ML-Komponenten beinhalten, aufgebaut werden

kann, ist daher ein aktives Feld der Forschung, in dem Forschungsprojekte wie das PEGASUS Projekt[1] oder KI-Absicherung[2] erste Grundlagen geschaffen haben.

Einen wesentlichen Aspekt in einer möglichen Sicherheitsargumentation werden voraussichtlich auch die verwendeten Entwicklungsprozesse darstellen. Aus diesem Grund und da die Nutzung ML basierter Komponenten im Fahrzeug erforderlich scheint, um die gewünschten Funktionalitäten zukünftiger Fahrzeuge zu ermöglichen, wurden von dem Verband der Automobilindustrie (VDA) mit dem *Automotive Software Process Improvement and Capability Determination (ASPICE)* in der Version 4.0 [VDA23b] standardisierte Referenzprozesse zur Entwicklung von ML-Komponenten eingeführt. Die vier neuen *Machine Learning Engineering (MLE)*-Prozesse und der neue *Supporting (SUP)*-Prozess zum Datenmanagement definieren spezielle Anforderungen an Prozesse zur Entwicklung von ML-Komponenten für den Einsatz in Automobilen und erweitern damit den bisherigen Umfang des ASPICE, welcher sich auf die klassische System- und Softwareentwicklung in der Automobilindustrie beschränkte.

1.1. Zielsetzung

Das Ziel dieser Masterarbeit ist es, die im ASPICE 4.0 standardisierten Referenzprozesse zur Entwicklung einer Softwarekomponente, welche auf ML basiert, bezüglich ihrer Anwendbarkeit in einem sicherheitskritischen Kontext zu verifizieren und ihre Unterstützung innerhalb einer möglichen Sicherheitsargumentation zu bestätigen.

Zunächst werden dazu die fünf, im ASPICE 4.0 [VDA23b] neu eingeführten, ML spezifischen Entwicklungs- und Unterstützungsprozesse innerhalb einer experimentellen Entwicklung einer Verkehrszeichenerkennung pilotiert. Dabei wird zuerst, ausgehend von möglichen Stakeholder-, System- und Softwareanforderungen, analysiert, welche Anforderungen an die ML-Komponente sowie an die zu verwendenden Daten zu stellen sind. Basierend auf den resultierenden Anforderungen an die Daten wird ein, dem Umfang der experimentellen Entwicklung angepasstes, Datenmanagement aufgebaut und Daten für das Training und Testen der ML-Komponente gesammelt. Weiterhin wird basierend auf den Anforderungen an die ML-Komponente eine Architektur dieser erstellt und die Komponente entsprechend dieser Architektur implementiert und trainiert. Abschließend erfolgt ein Test der entwickelten ML-Komponente, welcher die Erfüllung der gestellten Anforderungen bestätigen soll. Die resultierende Verkehrszeichenerkennung hat dabei nicht den Anspruch anschließend in einem realen Fahrzeug verwendet zu werden, sondern soll lediglich die Anwendbarkeit der Referenzprozesse belegen. Trotzdem

[1] `https://www.pegasusprojekt.de`, zuletzt aufgerufen am 23.10.2023
[2] `https://www.ki-absicherung-projekt.de`, zuletzt aufgerufen am 23.10.2023

wird die Verkehrszeichenerkennung als sicherheitskritische Komponente gehandhabt und entsprechende Anforderungen an die Performanz und Robustheit gestellt.

Im zweiten Hauptteil der Masterarbeit werden die, während der experimentellen Entwicklung angewandten, im ASPICE 4.0 standardisierten Referenzprozesse kritisch hinterfragt und bezüglich ihrer Unterstützung einer möglichen Sicherheitsargumentation eingeordnet. Dazu werden zunächst verschiedene Ansätze einer möglichen Sicherheitsargumentation für die Verwendung von ML basierten Komponenten in einem Automobil, welche den Stand der Forschung abbilden, als Ergebnis einer Literaturrecherche vorgestellt. Anschließend wird geprüft, welche Aspekte innerhalb der Ansätze durch die Umsetzung der ML spezifischen Referenzprozesse des ASPICE 4.0 vollständig adressiert oder teilweise unterstützt werden können, um abschließend herauszustellen, ob und bis zu welchem Grad die Anwendung der Referenzprozesse als Nachweis in einer Sicherheitsargumentation fungieren kann.

1.2. Aufbau

Zunächst werden in Abschnitt 2 Grundlagen zu dieser Masterarbeit vermittelt, welche eine einheitliche Wissensbasis für das Verständnis der weiteren Ausarbeitungen schaffen sollen. Zuerst werden dazu in Abschnitt 2.1 Grundlagen zum maschinellen Lernen vorgestellt. Der Fokus liegt dabei zunächst auf einer Abgrenzung zwischen klassischer Softwareentwicklung, künstlicher Intelligenz (KI), ML und *Deep Learning (DL)*. Anschließend wird der Aufbau von tiefen neuronalen Netzen zur Objekterkennung im Detail untersucht und vorgestellt, was für Architekturmodelle sich etabliert haben. Abschließend werden Herausforderungen für den Einsatz von ML in Automobilen und speziell in sicherheitskritischen Anwendungen benannt, welche in einer Sicherheitsargumentation berücksichtigt werden sollten. In Abschnitt 2.2 werden daraufhin das *Process Reference Model (PRM)* und das *Process Assessment Model (PAM)* des ASPICE 4.0 vorgestellt. Dabei werden die verschiedenen Prozessgebiete und ihr Zusammenwirken beschrieben und die MLE-Prozesse in das ASPICE eingeordnet. Die Grundlagen werden abgeschlossen durch die Vorstellung der *Goal Structuring Notation (GSN)* in Abschnitt 2.3, welche eine standardisierte Methode zur Strukturierung einer Sicherheitsargumentation darstellt.

In Abschnitt 3 werden die Ergebnisse der experimentellen Entwicklung einer Verkehrszeichenerkennung, basierend auf einem tiefen neuronalen Netz, vorgestellt. Dabei wird gemäß dem PRM des ASPICE 4.0 zunächst in Abschnitt 3.1 auf die Anforderungsanalyse eingegangen und basierend auf den Anforderungen an die Daten in Abschnitt 3.2 das Datenmanagement, inklusive der Erzeugung einer Datenbasis für das Training und Testen, beschrieben. Weiter wird in Abschnitt 3.3 die gewählte Architektur der Verkehrszeichenerkennung thematisiert und in Abschnitt 3.4 das Training dieser aufgezeigt. Anschließend wird in Abschnitt 3.5

auf den Test der ML-Komponente gegen die spezifizierten Anforderungen eingegangen und damit die Beschreibung der Entwicklungsaktivitäten abgeschlossen.

Weiter wird in Abschnitt 4 der zweite Hauptteil dieser Masterarbeit beschrieben, in welchem das PRM des ASPICE 4.0 hinsichtlich der Unterstützung einer möglichen Sicherheitsargumentation für den Einsatz einer ML-Komponente in einem sicherheitskritischen System, wie zum Beispiel einem autonom fahrenden Fahrzeug, verifiziert wird. Dazu werden in Abschnitt 4.1 zuerst verschiedene Sicherheitsargumentationen vorgestellt und in Abschnitt 4.2 anschließend die ML spezifischen Referenzprozesse des ASPICE 4.0 in diese eingeordnet.

Abgeschlossen wird diese Masterarbeit durch eine Zusammenfassung in Abschnitt 5 und einen Ausblick auf mögliche weitere Forschungs- und Entwicklungsaktivitäten zu dem Thema dieser Masterarbeit in Abschnitt 6.

Angehangen sind außerdem in Anhang A die Anforderungsspezifikation, in Anhang B die Architekturspezifikation, in Anhang C der Trainings- und Validierungsansatz, in Anhang D der Testansatz und in Anhang E der Testbericht als Artefakte, welche während der experimentellen Entwicklung erzeugt wurden. In Anhang F ist weiter eine Tabelle abgebildet, in welcher die Sicherheitsziele aus [WCACP20] auf die *Base Practices (BPs)* der ML spezifischen Referenzprozesse des ASPICE 4.0 [VDA23b] abgebildet werden.

2. Grundlagen

Folgend werden in Abschnitt 2.1 für die weitere Masterarbeit benötigte Grundlagen des maschinellen Lernens vermittelt und einige Grundbegriffe eingeführt. Anschließend wird in Abschnitt 2.2 das PRM und PAM des ASPICE 4.0 vorgestellt. Abschließend wird in Abschnitt 2.3 beschrieben, wie eine Sicherheitsargumentation mit Hilfe der GSN aufgebaut werden kann.

2.1. Maschinelles Lernen

Machine Learning (ML) und künstliche Intelligenz (KI) sind zwei voneinander abzugrenzende, nicht synonyme Begriffe. Die KI bezeichnet generell die Anstrengung eine intellektuelle Aufgabe, welche normalerweise von einem Menschen ausgeführt würde, zu automatisieren. ML hingegen bezeichnet konkret die Fähigkeit eines Modells, aus problemspezifischen Trainingsdaten zu lernen und dieses Wissen auf weitere ähnliche Aufgabenstellungen anzuwenden. ML entspricht also einem Feld innerhalb des Gebietes der KI, welches wiederum verschiedene Ansätze wie zum Beispiel das *Deep Learning (DL)* umfasst, wie in Abb. 2 dargestellt [KS17, VDA23b, W$^+$20]. Da sich DL basierend auf tiefen neuronalen Netzen im Bereich der Objekterkennung und -klassifikation als vielversprechendster Ansatz etabliert hat [LGW$^+$21, ZZXW19], bildet diese Technologie die Grundlage für die experimentelle Entwicklung im Rahmen dieser Masterarbeit. Die spezielle Funktionsweise eines tiefen neuronalen Netzes zur Objekterkennung wird nachfolgend in Abschnitt 2.1.2 detailliert beschrieben. Vorweg werden jedoch in Abschnitt 2.1.1 die Grundprinzipien und Grundbegriffe des ML eingeführt, welche auch für DL gültig sind und im Weiteren vorausgesetzt werden. Abgeschlossen wird dieser Abschnitt, indem in Abschnitt 2.1.3 Herausforderungen für den Einsatz von ML in möglicherweise sicherheitskritischen Anwendungen benannt werden, welche in einer Sicherheitsargumentation zu adressieren sind.

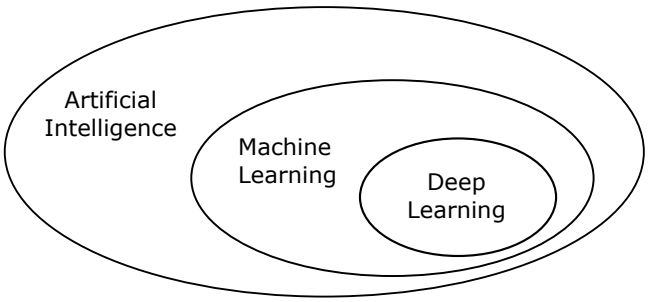
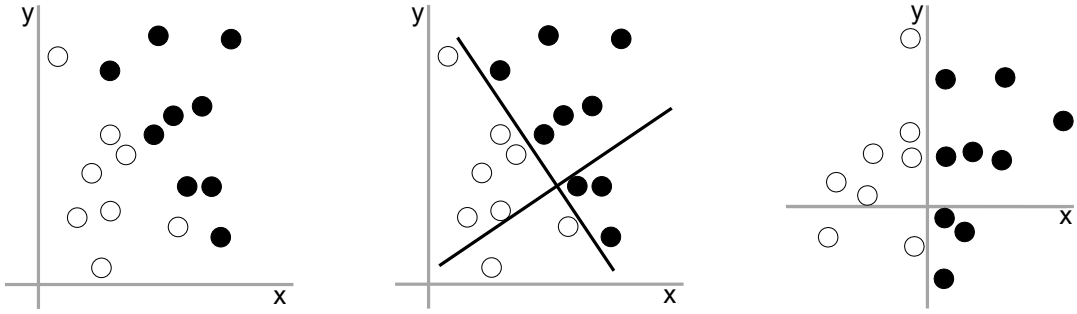


Abbildung 2: Einordnung von ML im Gebiet der KI, Bildquelle [KS17]

2.1.1. Grundprinzipien und Grundbegriffe des ML

Im Gegensatz zu herkömmlicher Software, bei welcher der Programmierer Regeln im Quellcode implementiert, um das Verhalten des Programmes festzulegen, werden beim maschinellen Lernen dem Programm bzw. ML-Modell mögliche Eingangsdaten und gewünschte Ausgaben präsentiert, aus denen das Modell selbstständig Verarbeitungsregeln lernt [KS17, W$^+$20]. Jedes n-te Eingangsdatum x_n entspricht dabei in der Regel einem D-dimensionalen Tensor realer Zahlen. Die Zeilen dieses Tensors werden dabei als ein Datenpunkt bezeichnet und die Spalten repräsentieren die Eigenschaften, sogenannte *Feature*, des Datums, wobei jede Eigenschaft durch $d = 1, ..., D$ indiziert wird [DFO20]. Die genutzten Daten bilden so das zu lösende Problem ab und sind die Grundlage für das ML-Modell, um Zusammenhänge zur Lösung des Problemes zu lernen. Als ein wesentlicher Erfolgsfaktor bedürfen die Daten einer angemessenen Vorverarbeitung, bevor sie für die Entwicklung des ML-Modells verwendet werden können. Dabei ist sicherzustellen, dass das zu lösende Problem in den Daten korrekt und vollständig abgebildet ist, um auszuschließen, dass relevante Einflussfaktoren zur Lösung des Problemes in den Daten nicht repräsentiert sind. Typische Arbeitsschritte sind dabei zunächst das Sammeln oder Erzeugen der Daten, anschließend das Normalisieren der Daten auf ein bestimmtes Intervall und das Labeln der Daten, wobei jedem Datum x_n die gewünschte Ausgabe y_n zugewiesen wird. Abschließend folgt die Prüfung, ob der Datensatz in sich schlüssig ist. Abb. 3 visualisiert ein simples Beispiel für ein Problem, welches durch ein auf ML basierendes Programm gelöst werden soll. Das Programm soll zwei Koordinaten (x, y) als Eingabe entgegennehmen und ausgeben, ob dem Punkt die Farbe Schwarz oder Weiß zuzuordnen ist. Die in Abb. 3a abgebildeten Rohdaten zeigen die ausgewählten und bezüglich des Koordinatensystemes normalisierten Daten, welche für das Training des Modells verwendet werden sollen. Jedem Punkt im Koordinatensystem wurde als Label die jeweilige Farbe zugewiesen, welche die gewünschte Ausgabe des Programms darstellt. Es ist zu sehen, dass es keine Punkte außerhalb des Koordinatensystems oder



(a) Rohdaten (b) Transformation (c) Neue Repräsentation

Abbildung 3: Repräsentation von Daten, Bildquelle [KS17]

große Ausreißer in den Daten gibt, sodass die Daten in sich stimmig und für das Training des ML-Modells geeignet erscheinen. Nicht aus den Rohdaten erkennbar ist, ob die Daten das Problem tatsächlich vollständig beschreiben. So könnte es, über die gesammelten Daten hinaus, in der Realität der Fall sein, dass eine Menge weiterer weißer Punkte für größere x-Koordinaten existiert, welche bei der Erzeugung der Daten fälschlicherweise vernachlässigt wurden. Die vor verarbeiteten Daten werden klassischerweise in drei disjunkte Datensätze, Trainings-, Validierungs- und Testdatensatz, aufgeteilt, welche in den folgend beschriebenen Phasen des maschinellen Lernens nacheinander zum Einsatz kommen [DFO20, KS17, W$^+$20].

Neben der Vorverarbeitung der Daten ist als Vorbereitung für das Training ein ML-Modell zu erzeugen, welches trainiert werden soll. Dazu kann zwischen verschiedenen Modellen und Algorithmen gewählt werden, welche sich in der Vergangenheit für ähnliche Probleme bereits bewährt haben, oder es kann ein neuartiges Modell erstellt werden. Ein ML-Modell kann dabei als eine Funktion $f : \mathbb{R}^D \rightarrow \mathbb{R}$ definiert werden, welche jedes D-dimensionale Eingangsdatum x_n auf eine reelle Zahl als Ausgabe $\hat{y}_n$ abbildet. Die Gewichte θ im Modell parametrisieren dabei die Funktion f und bestimmen, wie diese die Eingangsdaten transformiert, wie in Gleichung (1) am Beispiel einer linearen Regression für einen D-dimensionalen Eingangsvektor x angegeben [CGF$^+$20, DFO20, KS17].

$$f(x|\theta) = \theta_0 + \sum_{d=1}^{D} \theta_d x_d \tag{1}$$

Nach der Auswahl des ML-Modells wird dieses im ersten Schritt mit dem Ziel trainiert, die Gewichte θ im Modell so zu optimieren, dass die verwendeten Trainingsdaten möglichst gut verarbeitet werden. Dazu werden, wie in Gelb in Abb. 4 dargestellt, die Trainingsdaten zunächst vom Modell verarbeitet, wobei dieses die Daten durch Tensoroperationen transformiert und so für jedes Eingangsdatum x_n eine Vorhersage $\hat{y}_n$ erzeugt. Bezugnehmend auf das in Abb. 3 visualisierte Problem würden dem Modell beim Training eine Menge an Punkten zur Verarbeitung übergeben werden, das Modell würde die Daten transformieren und als Ausgabe bestimmen, ob der jeweilige Punkt schwarz oder weiß ist. Dabei ist während des Trainings zu ermitteln, ob die vom Modell erzeugten Vorhersagen mit den gewünschten Antworten übereinstimmen, woraufhin die Gewichte im Modell und somit die vorgenommenen Transformationen der Daten leicht angepasst und in einer weiteren Iteration die Daten erneut verarbeitet werden. Dazu berechnet die verwendete Verlustfunktion einen Distanzwert zwischen der vom Modell erzeugten Vorhersage $\hat{y}_n$ und dem erwarteten Ergebnis in Form des Labels y_n, welcher als Verlustwert bezeichnet wird. Je nach zu lösender Aufgabe können dabei verschiedene Verlustfunktionen verwendet werden. So könnte für eine lineare Regression zum Beispiel die in Gleichung (2) angegebene Quadratsumme der Residuen RSS, auch als L2-

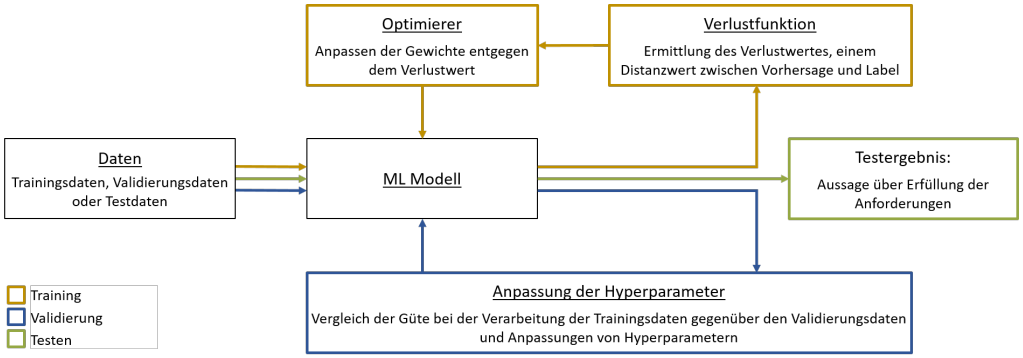


Abbildung 4: Trainings-, Validierungs- und Testworkflow eines ML-Modells, basierend auf [KS17]

Verlust oder Gauß-Verlust bezeichnet, als Verlustwert berechnet werden.

$$RSS = \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 \tag{2}$$

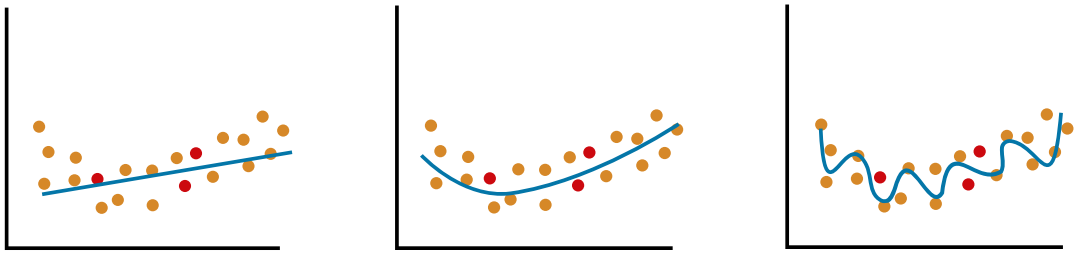
Anschließend berechnet der Optimierer, basierend auf einem *Backpropagation* Algorithmus, den Einfluss jedes Gewichtes in den Verlustwert und optimiert die Gewichte gradientenbasiert leicht entgegen diesem. So wird die vom Modell vorgenommene Transformation der Daten im Training iterativ angepasst und der Verlustwert, als ein Maß für die Güte der Vorhersage, verringert. Im Beispiel aus Abb. 3 entspräche die Genauigkeit der Vorhersage, als Verhältnis zwischen korrekt und falsch klassifizierten Punkten einem möglichen Maß für die Güte der Vorhersage. Mittels dieser Rückmeldung könnte das Modell iterativ die vorgenommene Transformation der Daten verbessern, bis letztlich im besten Fall alle Punkte korrekt klassifiziert werden. Dazu könnte das Modell die in Abb. 3b abgebildete Transformation der Daten lernen und die Daten basierend auf der in Abb. 3c abgebildeten neuen Repräsentation nach der in Algorithmus 1 angegebenen Regel klassifizieren. Das Training endet, nachdem eine zuvor festgelegte Anzahl an Epochen durchlaufen wurde, wobei eine Epoche für eine vollständige Verarbeitungsiteration aller Trainingsdaten steht, oder wenn das Training abgebrochen wird, da zum Beispiel festgestellt wurde, dass keine Verbesserung der Vorhersage mehr eintritt [CGF+20, DFO20, KS17, W+20].

Algorithm 1 Klassifikation schwarzer und weißer Punkte

```
if x <= 0 then
    return white
else
    return black
end if
```

Nachdem das ML-Modell trainiert und die Gewichte des Modells optimiert wurden, sollte das Modell validiert werden, wie in Abb. 4 in blau dargestellt. Dazu wird dem Modell ein neuer Datensatz in Form der Validierungsdaten als Eingabe zur Verfügung gestellt und von diesem verarbeitet. Wie auch bei den Trainingsdaten erzeugt das Modell Vorhersagen für die Validierungsdaten, welche gemäß dem Maß für die Güte der Vorhersage bewertet werden können. Häufig fällt bei der Validierung auf, dass die neuen Validierungsdaten nicht so gut verarbeitet werden wie die Trainingsdaten, was die Anpassung sogenannter *Hyperparameter* und neues Training erforderlich macht. *Hyperparameter* bezeichnen dabei, anders als die Gewichte, nicht Parameter innerhalb des Modells, welche die Transformation der Daten bestimmen, sondern Parameter des Modells, welche den Lernprozess beeinflussen. Sie legen zum Beispiel die Konfiguration des Modells oder den Aufbau des Trainingsdatensatzes fest. *Hyperparameter* lassen sich dabei nicht wie die Gewichte des Modells numerisch optimieren, sondern es bedarf einer Analyse des Modells und bestimmter Suchtechniken, um geeignete Werte für die *Hyperparameter* auszuwählen. Dass die Vorhersage für die Validierungsdaten schlechter ist als für die Trainingsdaten, kann dabei viele verschiedene Gründe haben und daher Anpassungen verschiedener *Hyperparameter* begründen. Unter anderem könnten die Validierungsdaten Szenarien beinhalten, welche in den Trainingsdaten unterrepräsentiert waren und daher nicht gelernt wurden. In diesem Fall sollte der Trainingsdatensatz entsprechend angepasst und das Modell neu trainiert werden. Weiterhin könnte das Modell, wie in Abb. 5c visualisiert, bezüglich der Trainingsdaten über-angepasst sein. Die blaue Funktion stellt dabei in Abb. 5 jeweils die vom Modell aus den Trainingsdaten erzeugte Regression dar, welche alle Punkte möglichst gut abbilden soll. In dem Fall der Überanpassung hat das Modell nicht wie bei einer guten Anpassung, siehe Abb. 5b, die generellen Aspekte aus den gelb dargestellten Trainingsdaten gelernt, welche es ermöglichen auch die rot dargestellten neuen Validierungsdaten sinnvoll abzubilden, sondern hat zu viele spezifische Details aus den Trainingsdaten gelernt. Diese verbessern zwar die Vorhersagen der Trainingsdaten, verschlechtern aber die Vorhersagen neuer Daten gegenüber dem guten Modell. Da eine Überanpassung verschiedene Gründe haben kann, können auch Anpassungen verschiedener *Hyperparameter* ausprobiert werden, welche zu einer Verbesserung des Modells führen könnten. So kann eine Überanpassung zum Beispiel reduziert werden, indem die Komplexität des Modells reduziert oder die Anzahl der Trainingsiterationen verringert wird, sodass das Modell bei der Wiederholung des Trainings weniger Details aus den Trainingsdaten lernen kann. Es ist hierbei immer ein Optimum zwischen Unteranpassung, siehe Abb. 5a, bei welcher das Modell zu wenig aus den Trainingsdaten gelernt hat, und Überanpassung anzustreben [DFO20, KS17, VDA23b, W$^+$20].

Wenn das Modell sowohl für die Trainingsdaten als auch für die Validierungsdaten eine den Anforderungen entsprechende Güte in der Vorhersage aufweist, sollte



(a) Unteranpassung (b) Gute Anpassung (c) Überanpassung

Abbildung 5: Unter- und Überanpassung, basierend auf `https://de.mathworks.com/discovery/overfitting.html`, zuletzt aufgerufen am 25.10.2023

abschließend das Modell mit einem weiteren neuen Datensatz, den sogenannten Testdaten, getestet werden, in Abb. 4 in grün dargestellt. Dieser Schritt ist erforderlich, da durch das Anpassen der *Hyperparameter* Änderungen auf Basis der Validierungsdaten vorgenommen wurden, welche eine Überanpassung bezüglich der Validierungsdaten ermöglichen. Wenn der Testdatensatz von dem Modell zu einer Ausgabe verarbeitet wurde, kann bewertet werden, ob das Modell neue Daten, den an das Modell gestellten Anforderungen entsprechend, verarbeiten kann und somit für den Einsatz geeignet ist oder ob weitere Trainings- und Validierungsiterationen erforderlich sind. Wichtig ist dabei, dass die verwendeten Testdaten tatsächlich neue Daten sind, auf welchen keine Anpassungen des Modells durchgeführt wurden, um zu verhindern, dass eine Überanpassung gegenüber den Testdaten erfolgt, welche dazu führen könnte, dass im produktiven Einsatz des Modells neue Daten nicht den Anforderungen entsprechend verarbeitet werden [KS17, W$^+$20].

2.1.2. Funktionsweise und Aufbau tiefer neuronaler Netze zur Objekterkennung

Bei tiefen neuronalen Netzen handelt es sich um ML-Modelle, welche aus mehreren Schichten mit Neuronen aufgebaut sind, wie in Abb. 6 schematisch abgebildet. Die als Kreise visualisierten Neuronen der blau gefärbten Eingabeschicht nehmen die Eingangsdaten auf und übergeben diese an die Neuronen der nachfolgenden verdeckten Schicht, in Orange dargestellt, welche eine Transformation der Daten lernt. Die neue Repräsentation der Daten wird anschließend an die rote Ausgabeschicht weitergegeben, welche die abschließende Vorhersage über die Eingangsdaten erzeugt [KS17, W$^+$20]. Ein als Kreis dargestelltes Neuron, welches mit k Neuronen der vorherigen Schicht, über die durch schwarze Linien visualisierten Kanten, verbunden ist, entspricht dabei einer Funktion $f : \mathbb{R}^k \to \mathbb{R}$, welche, wie in Gleichung (3) angegeben, aus einer nichtlinearen Aktivierungsfunktion $g : \mathbb{R} \to \mathbb{R}$ und einer linearen Abbildung $q : \mathbb{R}^k \to \mathbb{R}$ aufgebaut ist, wobei $z_1, ..., z_k$ für die Ausgabe der

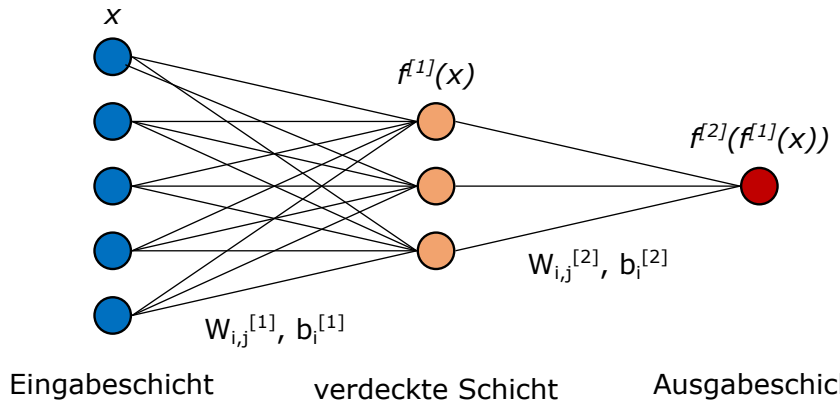


Abbildung 6: Schematische Darstellung eines tiefen neuronalen Netzes, basierend auf [CGF+20, W+20]

Neuronen aus der vorherigen Schicht stehen.

$$f(z_1, ..., z_k) = g(q(z_1, ..., z_k)) \tag{3}$$

Die lineare Abbildung q entspricht der vom Modell zu lernenden Transformation der Daten und ist, wie in Gleichung (4) angegeben, durch die Gewichte $w_1, ...w_k$, welche bestimmen, wie stark sich die Ausgabe eines Neurons als Eingabe auf das nächste Neuron auswirkt, und den Bias b parametrisiert.

$$q(z_1, ..., z_k) = \sum_{j=1}^{k} w_j z_j + b \tag{4}$$

Die nichtlineare Aktivierungsfunktion g sorgt dafür, dass von dem neuronalen Netz bessere Repräsentationen der Daten gelernt werden können, indem sie zum Beispiel festlegt nur Eingangssignale oberhalb eines bestimmten Schwellenwertes weiter zu verarbeiten. Für die verdeckten Schichten in tiefen neuronalen Netzen werden dabei klassischerweise die *Sigmoid*-Funktion, *Tangens Hyperbolicus (tanh)* oder *Rectified Linear Unit (ReLU)* als Aktivierungsfunktionen verwendet, welche in Abb. 7 visualisiert sind. Die in grün dargestellte *Sigmoid*-Funktion entspricht einer geglätteten Stufenfunktion und bildet die Eingangswerte auf einem Wertebereich von null, für kleine Werte, bis eins, für große Werte, ab. Der tanh ähnelt der *Sigmoid*-Funktion, bildet, wie in blau visualisiert, die Eingangswerte jedoch auf einem Bereich von minus eins bis eins ab, sodass Werte nahe null auch gegen null abgebildet werden. Die in Rot gefärbte ReLU Aktivierungsfunktion hingegen bildet negative Werte auf null ab und positive Werte und null linear auf sich selbst. Weiter wird gerade für Klassifikationsprobleme häufig innerhalb der letzten Schicht des neuronalen Netzes die in Gleichung (5) angegebene *Softmax*-Funktion verwendet. Sie hängt von allen Ausgaben der n Neuronen derselben Schicht ab, wobei $l = 1, ..., n$ die n Neuronen der

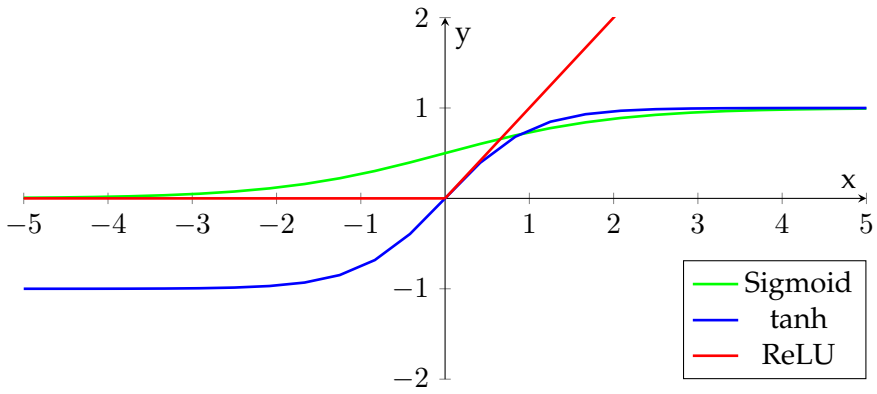


Abbildung 7: Klassische Aktivierungsfunktionen, basierend auf [CGF+20, KS17, W+20]

Schicht bezeichnet und q_l die Ausgabe ihrer jeweiligen linearen Abbildung.

$$g_l(q_1, ..., q_n) = \frac{e^{-q_l}}{\sum_{l=1}^{n} e^{-q_{l'}}} \tag{5}$$

Ein tiefes neuronales Netz $F(x)$ mit n Schichten kann somit, wie in Gleichung (6) angegeben, definiert werden, wobei $W^{[n]}$ und $b^{[n]}$ die Matrix der Gewichte und den Bias-Vektor der n-ten Schicht bezeichnen [CGF+20, KS17, Kut23, W+20].

$$F(x) = g^{[n]}(W^{[n]}g^{[n-1]}(...g^{[1]}(W^{[1]}x + b^{[1]})... + b^{[n-1]}) + b^{[n]}) \tag{6}$$

Für Problemstellungen der Objekterkennung und -klassifizierung haben sich sogenannte *Convolutional Neural Networks (CNNs)* in der Forschung etabliert. Diese verwenden spezielle *Convolutional Layer*, um Muster in der Eingabe zu erfassen, deren Funktionsweise in Abb. 8 visualisiert ist. Als Eingabe nehmen die Neuronen des *Convolutional Layer* eine *Feature Map* entgegen, welche im Beispiel eines Bildes als dreidimensionaler Tensor aufgefasst werden kann, wobei die Breite und Höhe der *Feature Map* durch die Anzahl der Pixel im Bild bestimmt werden. Die Tiefe eines Bildes beträgt weiterhin klassischerweise drei, für die Farbwerte rot, grün und blau. Die Neuronen in dem *Convolutional Layer* unterteilen die *Feature Map* in Abschnitte, sogenannte *Receptive Fields*, die jeweils um klassischerweise einen Pixel verschoben sind und bilden diese durch Tensoroperationen auf Ausgabetensoren ab, welche abschließend als Ausgabe des Neurons wieder zu einer *Feature Map* zusammengeführt werden. Bei den durch das *Convolutional Layer* durchgeführten Transformationen kann dabei zwischen einer Filterung und sogenanntem Pooling unterschieden werden. Bei der Filterung wird ein, als Filter oder Kern bezeichneter, Tensor mit den trainierten Gewichten als Elemente, welche das gelernte Muster darstellen, durch

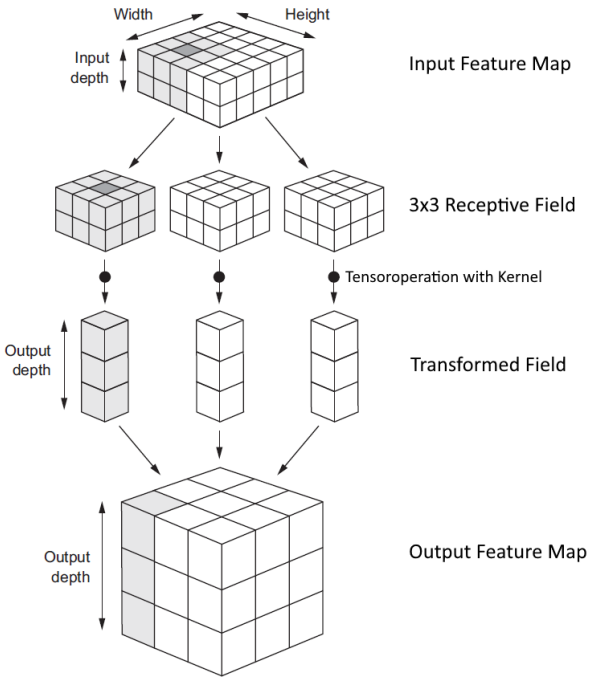


Abbildung 8: Funktionsweise eines Neurons in einem *Convolutional Layer*, basierend auf [KS17]

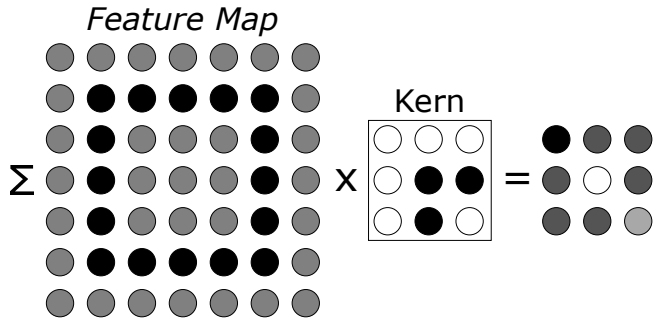


Abbildung 9: Schematische Darstellung der Filterung in einem zweidimensionalen *Convolutional Layer* mit Schrittweite = 2, basierend auf [CGF+20]

eine Faltung mit den Werten des *Receptive Fields* verrechnet und durch eine nicht lineare Aktivierungsfunktion wie *Sigmoid* oder *ReLU* in den Ausgabetensor abgebildet. Für eine zweidimensionale *Feature Map*, wie in Abb. 9 dargestellt, entspricht diese Faltung der Gleichung (7), wobei $f_{n,m}$ für die Gewichte des Kernes steht, welcher als $k * k$-Matrix interpretiert werden kann und s entspricht der Schrittweite in Pixeln, mit welcher sich der Filter bei der Faltung weiterbewegt.

$$q_{i,j} = \sum_{m=1}^{k} \sum_{n=1}^{k} f_{n,m} x_{si-m,sj-n} + b_0 \tag{7}$$

In Abb. 9 entspricht die *Feature Map* einer $7*7$-Matrix mit in Graustufen dargestellten Punkten und der Kern entspricht einer $3*3$-Matrix, welche eine Ecke aus schwarzen Punkten als Muster gelernt hat. Jeder Punkt in der Ausgabe gibt durch die Färbung an, wie stark der Kern und das betrachtete *Receptive Field* übereinstimmen. So ist der Punkt oben links der Ausgabe schwarz, da das Muster des Kernes in dem *Receptive Field* oben links vollständig vorhanden ist, wogegen der Punkt in der Mitte der Ausgabe weiß ist, da das Muster des Kernes in dem *Receptive Field* in der Mitte der *Feature Map* gar nicht vorhanden ist. Die Pooling Operation unterscheidet sich von der Filterung, indem die Transformation der Daten nicht durch Gewichte im Filter gelernt, sondern durch die Art des Poolings, bereits vor dem Training bestimmt wird. So wird zum Beispiel bei einem *max-Pooling Layer* immer das Maximum aus dem *Receptive Field* als Ausgabe weitergegeben [CGF+20, KS17, W+20, ZZXW19].

In Abb. 10 ist am Beispiel von VGG16, einer typischen Convolutional Neural Network (CNN) Architektur für Bildverarbeitung, visualisiert, wie Filterungsschichten und *max-Pooling Layer* in einem CNN kombiniert werden können. Die grau dargestellten Schichten dienen dabei, wie zuvor beschrieben, der Filterung und lernen eine Transformation der Daten, um Muster, unabhängig von der genauen Position dieser in der Eingabe, zu erkennen. So können zum Beispiel Verkehrszeichen, trotz verschiedener Positionen im Bild, gut erkannt werden. In Rot werden weiterhin die *max-Pooling Layer* dargestellt. Diese werden zum einen

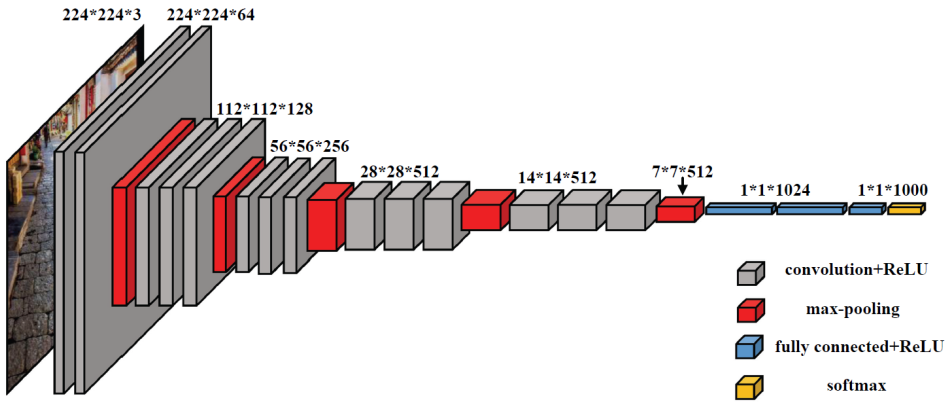


Abbildung 10: Architektur eines CNN am Beispiel VGG16, Bildquelle [ZZXW19]

benötigt, um die Anzahl der zu betrachtenden *Feature Maps* in den folgenden Schichten und somit die Komplexität des Modells zu reduzieren. Zum anderen wird durch das Pooling der in einer Schicht betrachtete Abschnitt in Bezug auf die ursprüngliche Eingabe des CNN stark vergrößert, was zu einer besseren Vorhersage führt. Auf die *Convolutional Layer* folgen in der VGG16 Architektur mehrere Schichten, in denen alle Neuronen mit allen Neuronen der vorherigen Schicht verbunden sind, welche es dem Netzwerk ermöglichen, verschiedene Aufgaben der Bildverarbeitung zu erfüllen. Abschließend folgt ein *Softmax Layer*, welches einen Vektor erzeugt, in dem jeder möglichen vorherzusagenden Kategorie eine Wahrscheinlichkeit zugeordnet ist, sodass sich alle Wahrscheinlichkeiten gemeinsam zu eins aufaddieren [CGF+20, KS17, W+20, ZZXW19].

Tiefe neuronale Netze für komplexe Aufgaben, wie zum Beispiel Objekterkennung oder -klassifizierung, können, wie im Beispiel aus Abb. 10, viele Schichten mit Milliarden von zu optimierenden Parametern besitzen. Diese hohe Anzahl von Parametern sorgt dabei zum einen für ein rechenintensives Training und macht zum anderen den Einsatz des Modells auf der Zielhardware in einem größeren System schwierig. Während des Trainings können einzelne leistungsstarke Computer den notwendigen Zeitaufwand in einem angemessenen Rahmen halten, in dem Produktivsystem hingegen, welches möglicherweise in hoher Stückzahl verkauft werden soll, würde eine solche leistungsstarke Architektur die Kosten unvertretbar stark erhöhen. Daher wird zwischen dem trainierten Modell, welches in der Entwicklung verwendet wird, und dem *Deployed Model*, welches in dem Produktivsystem eingesetzt wird, unterschieden. Das Entfernen von redundanten Neuronen, welche nicht die Güte der Vorhersage erhöhen, ist dabei ein Mittel, die Komplexität des *Deployed Models* gegenüber dem trainierten Modell zu reduzieren, ohne die Güte der Vorhersagen stark zu verschlechtern [LGW+21, VDA23a, VDA23b].

2.1.3. Herausforderungen beim Einsatz von ML

Wie zuvor beschrieben, besitzt ML ein anderes Programmierparadigma als klassische Software, da ML-Modelle selbständig Regeln bzw. Transformationen lernen, um die Eingaben in die gewünschten Ausgaben zu überführen. Die gelernten Transformationen hängen dabei maßgeblich von den während der Entwicklung verwendeten Daten sowie der Korrektheit der Label ab. Ein Sicherheitsbedenken besteht daher darin, ob die verwendeten Daten die Realität des zu lösenden Problemes gut abbilden. Die zu lösenden Probleme sind dabei meist so komplex, dass häufig unklar ist, ob alle relevanten Faktoren, welche das Problem definieren und deren Kombinationen in den verwendeten Daten ausreichend abgedeckt sind. Weiterhin können ungewollte Zusammenhänge in den verwendeten Daten, sogenannter *Bias*, dazu führen, dass das Modell während der Entwicklung zufriedenstellende, den Anforderungen entsprechende Vorhersagen erzeugt, in der Realität jedoch nicht. [RSG16] beschreibt ein Beispiel, in dem ein ML-Modell dazu verwendet werden soll, um Bilder mit Huskys oder Wölfen zu klassifizieren. In den für die Entwicklung verwendeten Daten bestand ein Zusammenhang darin, dass Wölfe immer vor Schnee abgebildet wurden und Huskys nicht. Das Modell hat für die verwendeten Daten Huskys und Wölfe zufriedenstellend klassifiziert, indem es den Zusammenhang zwischen Schnee im Bild und der Klasse Wolf gelernt hat. In der Realität ist dieser Zusammenhang jedoch nicht vorhanden, was zu Fehlern bei der Klassifizierung führen würde. Die Komplexität der realen Welt sorgt außerdem dafür, dass unbekannte Fälle existieren, für die unklar ist, was für Ausgaben das Modell erzeugen wird. Zusätzlich zu diesen Sicherheitsbedenken bezüglich der Korrektheit und Vollständigkeit der verwendeten Daten erzeugen ML-Modelle, selbst für die während der Entwicklung verwendeten Daten, nur mit einer gewissen Wahrscheinlichkeit eine korrekte Vorhersage, woraus sich schließen lässt, dass das Modell regelmäßig falsche Vorhersagen während des Einsatzes erzeugen wird [KS17, SQC17, WSRA20].

ML-Modelle scheinen, sobald sie den Anforderungen entsprechende Vorhersagen erzeugen, das zu lösende Problem verstanden zu haben und für den realen Einsatz geeignet zu sein. Wichtig zu bedenken ist bei dieser Annahme jedoch, dass ein ML-Modell keineswegs Probleme wie wir Menschen versteht, sondern lediglich mathematische Operationen gelernt hat, um die während des Trainings, der Validierung und des Testes genutzten Daten auf die gewünschte Ausgabe abzubilden. Für uns Menschen ist nur schwer nachvollziehbar und erklärbar, wie genau das ML-Modell zu seiner Vorhersage kommt und ob die gelernten Transformationen korrekt sind. Daten, welche sich für uns Menschen kaum merklich von den während der Entwicklung verwendeten Daten unterscheiden, können in einem ML-Modell zu absurd scheinenden, falschen Ausgaben führen.

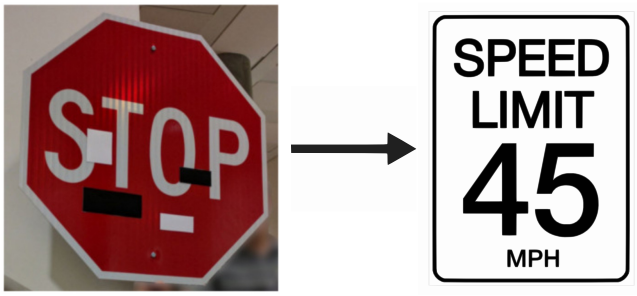


Abbildung 11: Beispiel eines *Adversarial Examples*, basierend auf [EEF+18]

Solche Daten können zufällig in der realen Welt auftreten oder in Form sogenannter *Adversarial Examples* gezielt als Angriff auf das Modell erzeugt werden. Abb. 11 zeigt zum Beispiel ein Stoppschild, welches gezielt mit schwarzen und weißen Aufklebern beklebt wurde und daher von dem im Beispiel verwendeten Modell falsch klassifiziert wurde. Für die meisten Menschen würde dieses Stoppschild problemlos als solches erkannt werden, das ML-Modell hat jedoch aufgrund der Aufkleber ein amerikanisches Schild für eine Geschwindigkeitsbegrenzung auf 45 Meilen pro Stunde erkannt. Gerade in sicherheitskritischen Systemen, wie einem autonom fahrenden Fahrzeug, können natürlich auftretende Variationen oder gezielte *Adversarial Examples* verheerende Folgen haben, welche den Tod mehrerer Personen bedeuten könnten [EEF+18, KS17, WSRA20].

Wurden die zuvor genannten Sicherheitsbedenken während der Entwicklung systematisch adressiert und sich daher dazu entschieden, dass das ML-Modell in dem realen Kontext eingesetzt werden kann, bestehen weitere Sicherheitsbedenken, welche ein konstantes Monitoring des Modells erforderlich machen können. So verändert sich die reale Welt konstant, wodurch Szenarien entstehen können, welche während der Entwicklung des Modells noch nicht bekannt waren und von dem Modell daher mit falscher Vorhersage verarbeitet werden. Ein Beispiel für solche Veränderungen der realen Welt ist das Auftauchen von Elektrokleinstfahrzeugen, sogenannten *E-Scootern*, in städtischen Regionen, welche seit 2019 am öffentlichen Straßenverkehr teilnehmen dürfen [Bun19, WSRA20].

2.2. Das ASPICE

Das ASPICE [VDA23b] standardisiert in der aktuellen Version 4.0 ein *Process Reference Model (PRM)* für die System- und Softwareentwicklung in der Automobilindustrie und beschreibt weiterhin ein *Process Assessment Model (PAM)* zur Bewertung der Fähigkeit von Prozessen in Entwicklungsprojekten. Erarbeitet wurde das ASPICE in dem Arbeitskreis 13 des *Quality Management Center (QMC)* des VDA, in welchem Vertreter verschie-

dener Erstausrüster, Zulieferer und Beratungsunternehmen vertreten sind.

2.2.1. Prozessreferenzmodell des ASPICE 4.0

Das PRM des ASPICE 4.0 [VDA23b], welches in Abb. 12 abgebildet ist, bietet eine standardisierte Referenz notwendiger Prozesse in Entwicklungsprojekten der Automobilindustrie und hilft dadurch geeignete Prozesse in dem Projekt zu etablieren. Gegliedert ist das PRM in drei Gruppen von Prozessen. Die primären Lebenszyklusprozesse, in orange, umfassen die Entwicklungsprozesse für die System-, Software- und Hardwareentwicklung sowie für die Entwicklung einer ML-Komponente. Weiterhin umfasst diese Gruppe den Validierungs-, Beschaffungs- und Auslieferungsprozess. Unterstützt werden die primären Lebenszyklusprozesse zum einen durch die organisatorischen Lebenszyklusprozesse, in blau dargestellt, welche den Management-, Prozessverbesserungs- und Wiederverwendungsprozess beinhalten, und zum anderen durch die unterstützenden Lebenszyklusprozesse, in grün dargestellt. Jeder Prozess wird im ASPICE durch einen Prozesszweck, Prozessresultate, Basispraktiken und Ausgabeinformationselemente beschrieben. Der Prozesszweck beschreibt innerhalb weniger Sätze

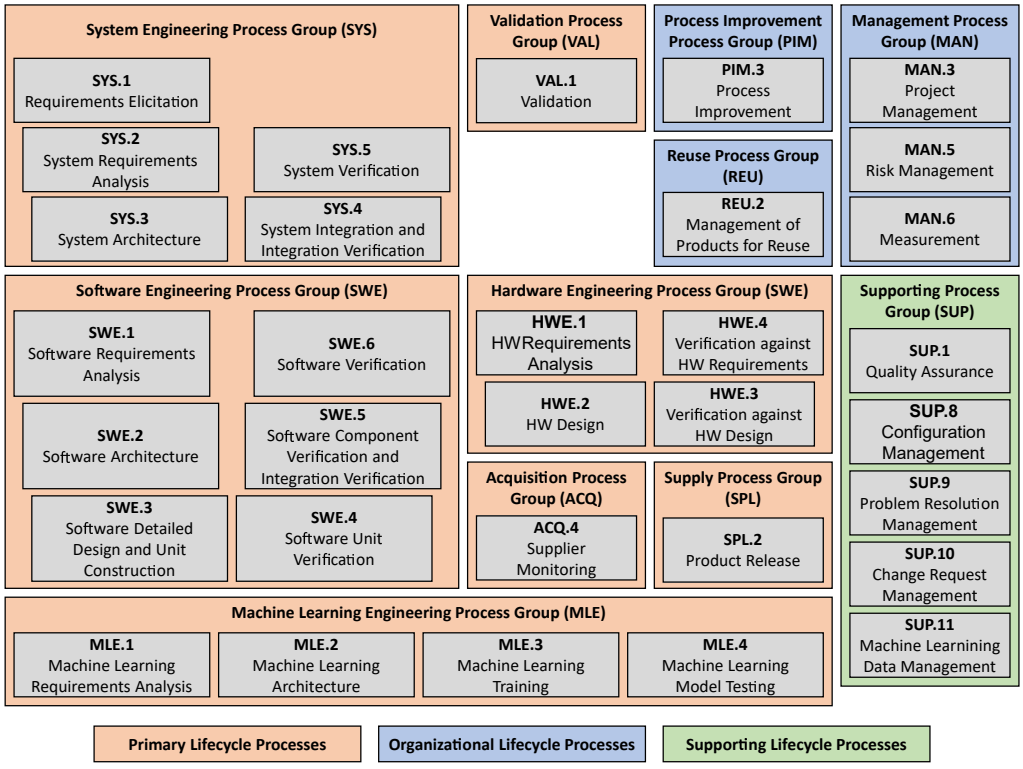


Abbildung 12: ASPICE Prozessreferenzmodell, basierend auf [VDA23b]

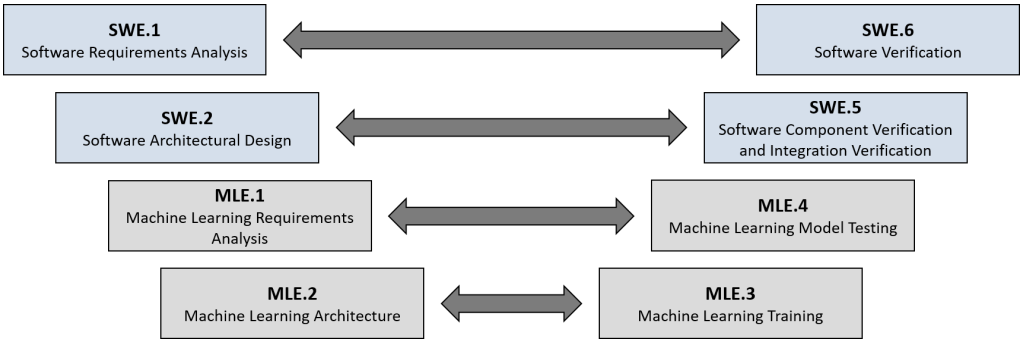


Abbildung 13: Integration der MLE-Prozesse im V-Modell des ASPICE 4.0, Bildquelle [VDA23b]

was das Ziel des Prozesses ist. Ergänzt wird dieser durch die verschiedenen Basispraktiken, welche konkretisieren was für Tätigkeiten in dem Prozess verankert und durchgeführt werden sollten, um den Prozesszweck zu erreichen. Die Prozessresultate beschreiben weiterhin, welche Ergebnisse die Umsetzung der Basispraktiken erzielt und werden dabei durch die Ausgabeinformationselemente ergänzt, welche konkrete Arbeitsprodukte des Prozesses beschreiben.

Die Entwicklungsprozesse des ASPICE folgen der Logik eines Softwareentwicklungsprozesses gemäß dem V-Modell [MM10, Roo86], bei welchem jeder Prozess auf der linken Seite des Prozessmodells zu genau einem Prozess auf der rechten Seite in Verbindung steht. Zunächst werden im ASPICE auf der linken Seite dazu Prozesse beschrieben, welche die Spezifikation von Anforderungen und einer Architektur vorsehen, bis hin zu der Implementierung des Quellcodes. Die rechte Seite beschreibt Testprozesse, gegen die jeweiligen Ebenen auf der linken Seite, von der Verifikation einer einzelnen *Unit* bis hin zu der abschließenden System Verifikation und Validierung. Besonders wichtig ist dabei die Konsistenz der Arbeitsprodukte über die verschiedenen Prozesse hinweg, welche durch eine bidirektionale Rückverfolgbarkeit zwischen den Arbeitsprodukten unterstützt wird.

Da ML sich in einigen Aspekten von klassischer Softwareentwicklung stark unterscheidet, wie in Abschnitt 2.1 herausgearbeitet, wurde das ASPICE in der Version 4.0 um fünf spezielle Prozesse für die Entwicklung von ML-Komponenten ergänzt. Wie in Abb. 13 dargestellt, lassen sich die *Machine Learning Engineering (MLE)*-Prozesse dabei unterhalb des Software Architekturdesigns anordnen, in dem entschieden werden kann, dass bestimmte Funktionalitäten der Software durch ML implementiert werden sollen. Basierend auf den Softwareanforderungen und den Informationen aus der Softwarearchitektur sind dann, wie in den Basispraktiken von MLE.1 beschrieben, Anforderungen an die ML-Komponente und die zu verwendenden Daten zu spezifizieren, zu strukturieren und bezüglich ihrer Korrektheit, Umsetzbarkeit, Testbarkeit und Auswirkungen auf die

Betriebsumgebung zu analysieren. Anschließend ist eine ML-Architektur als Grundlage für das Training, wie in MLE.2 beschrieben, zu erstellen, welche zu den ML-Anforderungen konsistent sein muss. Der Prozess der Architekturerstellung sieht weiterhin vor, dass die einzelnen Elemente der Architektur nach, im Projekt festgelegten, Kriterien evaluiert werden und Schnittstellen zwischen den Architekturelementen sowie Ressourcenverbrauchsziele für die Architekturelemente definiert werden. Anschließend ist das ML-Modell, wie im MLE.3 beschrieben, zu trainieren. Dazu ist zunächst ein systematischer Ansatz für das Training und die Validierung des ML-Modells zu spezifizieren. Dieser sollte im Speziellen einen Ansatz für die Anpassung der *Hyperparameter* beinhalten und muss definieren, wie der Datensatz fürs Training zu erstellen und zu modifizieren ist. Der Ansatz ist während des Trainings umzusetzen und so das ML-Modell systematisch und nachvollziehbar zu trainieren. Abgeschlossen werden die MLE-Prozesse durch den Prozess MLE.4, in welchem standardisiert ist, was für Basispraktiken beim Testen des ML-Modells anzuwenden sind. Auch hier ist zunächst im Projekt ein Ansatz zu spezifizieren, wie getestet werden soll. Dieser muss im Speziellen die Verteilung und Häufigkeiten von Testszenarien innerhalb der Testdaten beschreiben und für jedes Testdatum das erwartete Testergebnis festlegen. Weiterhin ist im Testansatz zu spezifizieren, wie der Testdatensatz zu erstellen und zu modifizieren ist. Bei der eigentlichen Testausführung unterscheidet das ASPICE im MLE.4 zwischen dem Test des trainierten ML-Modells und dem Test des *Deployed ML Model*, deren Unterschied in Abschnitt 2.1 erklärt wurde. Nach erfolgreicher Ausführung des MLE.4 ist die ML-Komponente isoliert, hinsichtlich der Erfüllung der an sie gestellten ML-Anforderungen getestet und kann, wie im Prozess der Softwarekomponenten- und Integrationsverifikation beschrieben, mit den weiteren Softwarekomponenten integriert und gemeinsam getestet werden.

Zusätzlich zu den MLE-Prozessen wurde mit dem ASPICE 4.0 der SUP.11 zum *Machine Learning Data Management* als weiterer Prozess eingeführt. Dieser verfolgt den Zweck, integre und qualitative Daten für das maschinelle Lernen zu erzeugen und zur Verfügung zu stellen, welche mit den an sie gestellten Anforderungen konsistent sind. Dazu sieht der Prozess vor, dass ein ML-Datenmanagementsystem eingeführt wird sowie ein Ansatz zur Sicherstellung der Datenqualität entwickelt und bei der Erzeugung und Aufbereitung der Daten eingesetzt wird.

2.2.2. Prozessassessmentmodell des ASPICE 4.0

Neben dem PRM bietet das ASPICE 4.0 [VDA23b] ein PAM, welches Indikatoren zur Verfügung stellt, um durch Assessoren zu bewerten, ob die im PRM beschriebenen Prozesszwecke, durch die im Projekt gelebten Prozesse, erfüllt werden. Als Indikatoren sind dabei in einem Assessment zum einen die Erfüllung der BPs und *Generic Practices (GPs)* zu bewerten und zum anderen die Existenz der geforderten

N	nicht erreicht	0% bis $\leq 15\%$	Es sind wenige oder keine Evidenzen für die Erfüllung des definierten Prozessattributes vorhanden.
P	teilweise erreicht	$> 15\%$ bis $\leq 50\%$	Es gibt etwas Evidenzen für einen Ansatz und die Erreichung des definierten Prozessattributes. Manche Aspekte können unvorhersehbar sein.
L	größtenteils erreicht	$> 50\%$ bis $\leq 85\%$	Es gibt Evidenzen für einen systematischen Ansatz und signifikante Erreichung des definierten Prozessattributes. Ein paar Schwächen können existieren.
F	vollständig erreicht	$> 85\%$ bis $\leq 100\%$	Es gibt Evidenzen für einen vollständigen und systematischen Ansatz und vollständige Erreichung des definierten Prozessattributes. Es existieren keine signifikanten Schwächen.

Tabelle 1: Bewertungsskala des ASPICE PAM, basierend auf [VDA23b]

Ausgabeinformationselemente des jeweiligen Prozesses. Auf Basis dieser Indikatoren werden die folgenden neun, aufeinander aufbauenden *Process Attributes (PAs)* jedes, im Assessment betrachteten, Prozesses auf der in Tabelle 1 dargestellten Skala von nicht erreicht bis vollständig erreicht bewertet.

- PA 1.1: *Process performance process attribute*
- PA 2.1: *Process performance management process attribute*
- PA 2.2: *Work product management process attribute*
- PA 3.1: *Process definition process attribute*
- PA 3.2: *Process deployment process attribute*
- PA 4.1: *Quantitative analysis process attribute*
- PA 4.2: *Quantitative control process attribute*
- PA 5.1: *Process innovation process attribute*
- PA 5.2: *Process innovation implementation process attribute*

Je nach angestrebter Prozessfähigkeit in dem Projekt oder der Organisationseinheit ist es dabei möglich, den Umfang des Assessments so anzupassen, dass nicht alle neun PAs bewertet werden, sondern nur die ersten.

Basierend auf der Bewertung der PAs kann die erreichte Prozessfähigkeit, des betrachteten Prozesses, in dem assessierten Projekt oder der assessierten

Level 0: unvollständiger Prozess	Der Prozess ist nicht implementiert oder erzielt nicht den Prozesszweck.
Level 1: ausgeführter Prozess	Der implementierte Prozess erzielt seinen Prozesszweck.
Level 2: gemanagter Prozess	Der ausgeführte Prozess wird geplant, überwacht und angepasst und seine Arbeitsprodukte werden angemessen kontrolliert und gepflegt.
Level 3: etablierter Prozess	Der gemanagte Prozess wurde mit Hilfe eines definierten Standardprozesses implementiert, welcher in der Lage ist, die geforderten Prozessresultate zu erzielen.
Level 4: vorhersehbarer Prozess	Der etablierte Prozess ist durch die Erhebung und Analyse von Messdaten in der Lage, Ursachen für Abweichungen von Managementbedürfnissen zu identifizieren und es werden Korrekturmaßnahmen durchgeführt, um die Abweichungsursachen zu beheben.
Level 5: innovativer Prozess	Der vorhersehbare Prozess wird kontinuierlich verbessert, um auf organisatorische Veränderungen zu reagieren.

Tabelle 2: Prozessfähigkeitslevel des ASPICE PAM, basierend auf [VDA23b]

Organisationseinheit in den, in Tabelle 2 beschriebenen, sechs Leveln angegeben werden. Level 0, bei welchem der Zweck des betrachteten Prozesses nicht erreicht wird, beschreibt dabei die geringste Prozessfähigkeit und Level 5, bei dem der Prozess kontinuierlich verbessert wird, die höchste. Level 0 liegt vor, wenn PA 1.1 und damit die Prozessausführung, gemäß der Bewertungsskala in Tabelle 1, höchstens teilweise erreicht wurde. Level 1 besteht sobald PA 1.1 mindestens größtenteils erreicht wurde. Die weiteren Level sind genau dann erzielt, wenn die Prozessattribute der darunterliegenden Level vollständig erreicht wurden und die zwei Prozessattribute für dieses Level mindestens größtenteils erreicht wurden. Für Level 2 muss also PA 1.1 vollständig erreicht sein und PA 2.1 und PA 2.2 mindestens größtenteils erreicht. Jedes Level baut somit auf dem darunterliegenden Level auf und erhöht die Prozessfähigkeit entsprechend.

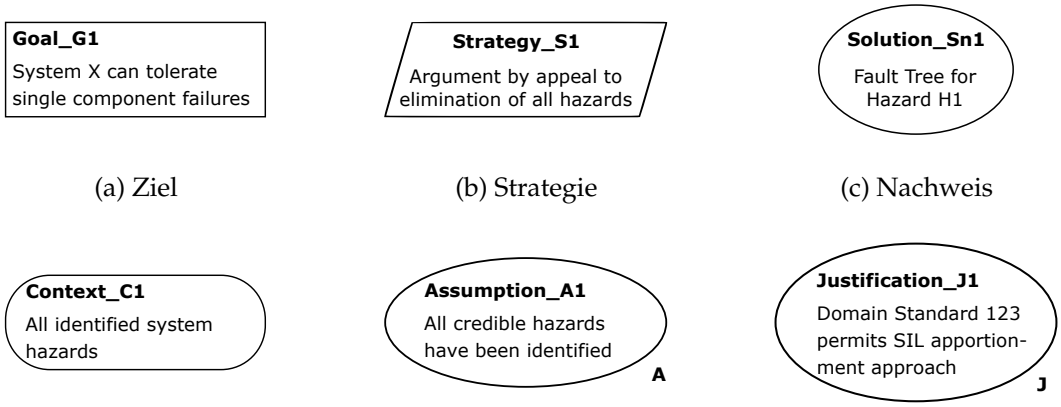
2.3. Aufbau einer Sicherheitsargumentation

Wie in Abschnitt 2.1.3 beschrieben gibt es einige Sicherheitsbedenken bezüglich des Einsatzes von ML in sicherheitskritischen Systemen, wie zum Beispiel

einem autonom fahrenden Fahrzeug. Diese Sicherheitsbedenken sind durch angemessene Methoden zu mitigieren und darauf aufbauend ist in einer Sicherheitsargumentation zu belegen, warum das System inklusive des ML-Modells für den Einsatz, im definierten realen Kontext, akzeptabel sicher ist. Die *Goal Structuring Notation (GSN)* ist eine der am weitesten verbreiteten Darstellungsform für eine solche Argumentationskette. Sie wurde von der *Assurance Case Working Group (ACWG)* standardisiert und kann genutzt werden, um eine Sicherheitsargumentation graphisch darzustellen. Der Fokus liegt dabei darauf, die Beziehungen zwischen den Elementen der Sicherheitsargumentation sowie den Kontext dieser zu veranschaulichen [Ass18, GRP+12, Kel04, WSRA20].

Jede Sicherheitsargumentation besteht aus Anforderungen, Behauptungen, Nachweisen und Informationen in Bezug auf den Kontext. Die Anforderungen oder Ziele definieren, welches Maß an Sicherheit das System erfüllen muss. Abgeleitet werden können sie aus verschiedenen Quellen wie zum Beispiel aus Sicherheitsstandards, aus Anforderungen eines Kunden oder aus einer Gefährdungsanalyse. Die Nachweise belegen weiterhin, welche Maßnahmen umgesetzt wurden, um die geforderte Sicherheit des Systems zu gewährleisten. Im Beispiel einer ML-Komponente könnte zum Beispiel als Maßnahme eine Datenbeschaffungsstrategie im Projekt definiert und nachweislich eingehalten worden sein oder spezielle Daten beim Testen verwendet worden sein. Eine solche bloße Sammlung von Nachweisen sagt jedoch nicht genug darüber aus, ob die Maßnahmen tatsächlich angemessen und ausreichend sind, um alle Sicherheitsanforderungen zu erfüllen. Daher sind die Behauptungen oder Argumente, welche eine Beziehung zwischen den Anforderungen und den Nachweisen aufbauen, für die Sicherheitsargumentation erforderlich. Sie beschreiben, warum die Nachweise geeignet sind, um zu argumentieren, dass die Sicherheitsanforderungen erfüllt wurden und können auch aufzeigen, an welcher Stelle möglicherweise noch Lücken in der Argumentationskette sind. Dabei beziehen sich die Behauptungen in der Regel auf einen ganz bestimmten Kontext, in welchem das System nachweislich die Anforderungen erfüllt. Dieser grenzt somit ab, unter welchen Bedingungen das System sicher verwendet werden kann und legt den vorgesehenen Einsatzbereich fest. Erkannte Lücken in der Sicherheitsargumentation könnten entweder durch weitere Nachweise, weitere Argumente oder eine Einschränkung des Kontextes adressiert werden. Im Bereich des autonomen Fahrens wird der Kontext, in welchem das ML-Modell sicher Vorhersagen erzeugen soll, häufig auch als *Operational Design Domain (ODD)* bezeichnet. Für eine, auf ML basierende, Personenerkennung eines autonom fahrenden Fahrzeuges könnte die ODD dabei beispielsweise bezüglich Faktoren wie Witterungsbedingungen, Straßenverhältnissen oder der Geschwindigkeit des Fahrzeuges eingeschränkt werden [Ass18, Kel04, WSRA20]. Das Unternehmen Cruise[3] legte so zum Beispiel über die Beschreibung der ODD in [Cru22a] fest, dass die eingesetzte Flotte in Kalifornien unter anderem nicht bei Schnee, Hagel

[3] `https://getcruise.com/`, zuletzt aufgerufen am 17.01.2024



(a) Ziel (b) Strategie (c) Nachweis

(d) Kontext (e) Annahme (f) Rechtfertigung

Abbildung 14: Elemente der GSN, basierend auf [Ass18]

oder starkem Regen sowie maximal mit 35 Meilen pro Stunde autonom fahren darf.

Die standardisierte GSN verwendet die in Abb. 14 dargestellten sechs Formen als Grundelemente, um eine Sicherheitsargumentation als sogenannte *Goal Structure* zu visualisieren und zu strukturieren. Die Ziele, siehe Abb. 14a, stellen dabei die Anforderungen hinsichtlich der Sicherheit des Systems dar. Sie können ganz oben in der *Goal Structure* stehen oder als Teil- bzw. Unterziel auf tieferen Ebenen. Die Strategien, siehe Abb. 14b, bilden die Argumente ab und beschreiben die Beziehungen zwischen Zielen auf verschiedenen Ebenen. Weiter wird mit Nachweisen, siehe Abb. 14c, eine Lösung für ein Ziel dargestellt, indem sie direkt mit dem Ziel in Beziehung gesetzt werden. Informationen über den Kontext eines Ziels oder einer Strategie können durch entsprechende Kontextelemente, siehe Abb. 14d, dokumentiert werden. Zusätzlich zu den vier klassischen Elementen einer jeden Sicherheitsargumentation ist es in der GSN möglich, zum einen Annahmen, siehe Abb. 14e, und zum anderen Rechtfertigungen, siehe Abb. 14f, zu visualisieren. Die Annahmen dienen dazu, zu dokumentieren, unter welchen angenommenen Randbedingungen das zugeordnete Ziel oder die zugeordnete Strategie gültig sind. Darüber hinaus ermöglichen Rechtfertigungen Begründungen für einzelne Elemente zu dokumentieren, sodass der Ersteller zum Beispiel festhalten kann, aus welchem Grund er eine bestimmte Strategie oder ein bestimmtes Ziel verwendet. Beziehungen zwischen Elementen in der *Goal Structure* können dabei durch zwei Arten von Pfeilen, siehe Abb. 15, dargestellt werden. Der Pfeil mit der ausgefüllten Spitze, siehe Abb. 15a, stellt eine sogenannte *SupportedBy* Beziehung dar und wird verwendet, um Schlussfolgerungen oder Beweisketten zu visualisieren. Erlaubt ist dieser Pfeil daher nur für Beziehungen zwischen Zielen, zwischen einem Ziel und einer Strategie oder zwischen einem Ziel und einem Nachweis. Der zweite Pfeil mit der unausgefüllten Spitze, siehe Abb. 15b, visualisiert eine

(a) *SupportedBy* (b) *InContextOf*

Abbildung 15: Beziehungen der GSN, basierend auf [Ass18]

InContextOf Beziehung, welche zwischen einem Ziel oder einer Strategie mit einem Kontext, einer Annahme oder einer Rechtfertigung bestehen kann. Abschließend besteht in der grundlegenden GSN noch die Möglichkeit, durch eine unausgefüllte Raute mittig unterhalb einem Ziel oder einer Strategie, anzugeben, dass das jeweilige Elemente bewusst nicht weiter betrachtet wurde [Ass18, GRP+12, Kel04].

Über die grundlegenden Elemente der GSN hinaus gibt es verschiedene Erweiterungen der GSN, welche weitere Elemente einführen, um spezielle Aspekte zu visualisieren. Beschreibungen einzelner solcher Erweiterungen sind an den GSN Gemeinschaftsstandard [Ass18] angehangen, wie zum Beispiel eine Erweiterung der GSN, um Argumentationsmuster zu visualisieren, welche die Abstraktion und Instanziierung der Sicherheitsargumentation ermöglichen. Außerdem formuliert dieser Standard Regeln, wie Beziehungen zwischen den Elementen zu visualisieren und wie die Aussagen in den jeweiligen Elementen zu formulieren sind.

Abb. 16 zeigt ein Beispiel für eine *Goal Structure*, basierend auf den Grundelementen der GSN, welche eine Argumentationskette dafür visualisiert, dass die Logik des betrachteten Kontrollsystems fehlerfrei ist. Dieses Ziel auf höchster Ebene wird in der Argumentation über zwei Strategien heruntergebrochen. Diese begründen die Fehlerfreiheit des Systems durch die Argumentation, dass zum einen alle an das System gestellten Sicherheitsanforderungen erfüllt sind, siehe S1, und zum anderen alle identifizierten Gefahren in der Software entfallen, siehe S2. Die identifizierten Gefahren in der Software sind dabei als Kontext zu S2 angegeben. Bezüglich Strategie S1 sind weiterhin drei Teilziele identifiziert worden, von denen eines bewusst nicht weiterentwickelt wurde. Die anderen beiden Teilziele der Strategie S1 können entweder durch weitere Ziele mit Lösungen oder direkt durch eine Lösung nachgewiesen werden. Für die Strategie S2 wurden ebenfalls zwei Teilziele identifiziert, welche jeweils durch dieselben zwei Lösungen nachgewiesen werden können. Für dieses Beispiel sind also *Black Box* Testergebnisse (Sn1), eine Zustandsmaschine des Kontrollsystems (Sn2), eine Fehlerbaumanalyse bezüglich eines bestimmten Events (Sn3) sowie Testergebnisse bezüglich der Gefahren (Sn4) zu erzeugen, um die Fehlerfreiheit der Logik des Systems nachzuweisen [Kel04].

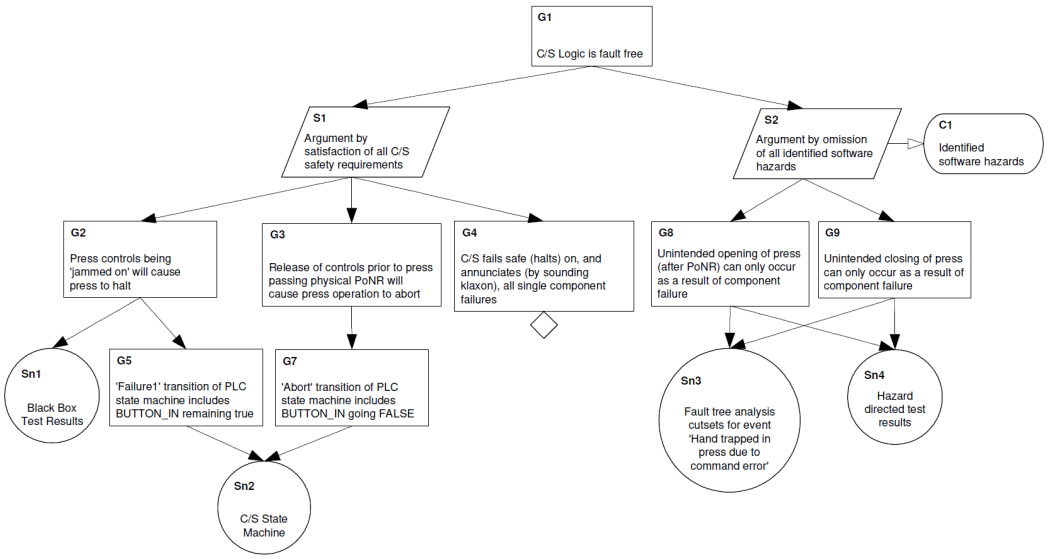


Abbildung 16: Beispiel einer *Goal Structure*, Bildquelle [Kel04]

3. Experimentelle Entwicklung einer Verkehrszeichenerkennung

Eine Verkehrszeichenerkennung stellt einen wichtigen, sicherheitskritischen Bestandteil der Umfeldwahrnehmung für Fahrerassistenzsysteme und autonom fahrende Fahrzeuge dar. Sie erlaubt es dem Fahrzeug, selbständig Verkehrszeichen zu erkennen und diese korrekt zu deuten und ermöglicht dem Fahrzeug, sicher und entsprechend der gesetzlichen Vorgaben am Verkehr teilzunehmen. Fehler in der Verkehrszeichenerkennung können wiederum schwere Folgen haben. So wäre es zum Beispiel denkbar, dass ein autonom fahrendes Fahrzeug zu schnell auf eine Baustelle zufährt und in dieser verunfallt, weil die entsprechenden Schilder mit Geschwindigkeitsbegrenzungen nicht erkannt oder falsch klassifiziert wurden [MRR+15, QY21, ZY22].

Die auf einem CNN basierende entwickelte Verkehrszeichenerkennung hat als Ziel, Verkehrszeichen, deren Positionen bereits ermittelt wurden, zu klassifizieren. Dabei wird nicht angestrebt das Modell in einem realen Fahrzeug einzusetzen, sondern die experimentelle Entwicklung dient ausschließlich dazu, die Anwendbarkeit der im ASPICE 4.0 [VDA23b] neu eingeführten fünf Referenzprozesse, welche in Abschnitt 2.2.1 vorgestellt wurden, exemplarisch für die Entwicklung einer ML-Komponente zu zeigen. Die nachfolgende Beschreibung der experimentellen Entwicklung gliedert sich daher nach den fünf angewandten Referenzprozessen. So wird zunächst in Abschnitt 3.1 auf die Anforderungsanalyse eingegangen und beschrieben, wie Anforderungen an die Verkehrszeichenerkennung sowie an die benötigten Daten abgeleitet, strukturiert und analysiert wurden. Anschließend wird in Abschnitt 3.2 das angewandte Datenmanagement wiedergegeben und in Abschnitt 3.3 vorgestellt, wie auf Basis der Anforderungen eine Architektur für die Verkehrszeichenerkennung erstellt wurde. Abschließend wird in Abschnitt 3.4 der genutzte Trainingsprozess des CNN erklärt und mit dem, in Abschnitt 3.5 beschriebenen, durchgeführten Testprozess die Erfüllung der gestellten Anforderungen an die Verkehrszeichenerkennung nachgewiesen. Der Fokus liegt dabei immer darauf, wie die in den BPs beschriebene Theorie in der experimentellen Entwicklung praktisch umgesetzt wurde. Die Umsetzung der GPs des ASPICE zur Erreichung der Fähigkeitslevel oberhalb von eins ist dabei nicht im Fokus, da diese sich zwischen den ML spezifischen Prozessen und klassischen Prozessen des ASPICE nicht unterscheiden und auch nicht mit der Version 4.0 des ASPICE neu eingeführt wurden.

3.1. Anforderungsanalyse

Der Referenzprozess MLE.1 *Machine Learning Requirements Analysis* des ASPICE 4.0 [VDA23b] verfolgt mit sechs BPs den Zweck, die für das ML relevanten Softwareanforderungen in eine Menge von ML-Anforderungen zu verfeinern, welche als

Grundlage für die weiteren Prozesse dienen. Da die während der experimentellen Entwicklung implementierte Verkehrszeichenerkennung jedoch in kein größeres Softwaresystem zu integrieren ist, bestehen keine ML relevanten Softwareanforderungen aus einem im V-Modell übergeordneten Softwareentwicklungsprozess, welche verfeinert werden könnten. Stattdessen wurden alle ML-Anforderungen für die experimentelle Entwicklung auf Basis eines Brainstormings sowie einer Recherche in Gesetzen und Normen initial erzeugt und bei Bedarf erweitert. Die vollständige Anforderungsspezifikation kann in Anhang A nachgeschlagen werden.

Die Hauptaktivität während der Anforderungsanalyse bestand in der eigentlichen Spezifikation der Anforderungen an die zu entwickelnde ML-Komponente. BP.1 des Prozesses MLE.1 sieht dabei vor, dass sowohl funktionale als auch nicht funktionale ML-Anforderungen sowie auch Anforderungen an die zu verwendenden Daten spezifiziert werden. Als funktionale Anforderungen innerhalb der experimentellen Entwicklung wurde zum Beispiel spezifiziert, dass die ML-Komponente ein CNN verwenden muss, welches Bilder im .ppm und .jpg Format als Eingabe entgegennehmen kann, und unter welcher Nummerierung die Wahrscheinlichkeit für welche Klasse im Ausgabetensor des Modells zu erzeugen ist. Als nicht funktionale Anforderungen wurde unter anderem definiert, was für Spezifikationen der für das Training und Testen zu verwendende Computer zu erfüllen hat und wie viel Prozent der Test- und Absicherungsdaten das Modell abschließend korrekt klassifizieren muss. Der festgelegte Wert von mehr als 98% korrekter Vorhersagen für den Testdatensatz und mehr als 95% für den Absicherungsdatensatz orientiert sich dabei an der EU Verordnung 2021/1958 [Eur21], welche festlegt, dass eine Geschwindigkeitsbegrenzung während einer bestimmten Testfahrt zu mindestens 90% korrekt ermittelt werden muss. Der festgelegte Zielwert für den Absicherungsdatensatz ist dabei etwas geringer als für den Testdatensatz, da der Absicherungsdatensatz gezielt die Limitationen des Modells testen soll, indem er durch natürliche Umwelteinflüsse gestörte Daten beinhaltet. Die weiteren Anforderungen an die Daten sollen gemäß BP.1 zumindest Datencharakteristiken und deren Verteilung innerhalb der zu verwendenden Datensätze angeben. Im Kontext der experimentellen Entwicklung kann als Datencharakteristik zum Beispiel das Vorhandensein eines bestimmten Verkehrszeichens in den Daten angesehen werden. So wurde in der Anforderungsspezifikation festgelegt, von welchen Verkehrszeichen Bilder in den Daten enthalten sein müssen und dass eine Gleichverteilung dieser im Test- und Absicherungsdatensatz bestehen muss. Aufgrund der begrenzten Rechenleistung des zu verwendenden Computers wurde sich dazu entschieden, das Modell für die Klassifizierung von zehn verschiedenen Verkehrszeichen zu trainieren, welche in Abb. 17 dargestellt sind. Diese Auswahl deckt mit den Verkehrszeichen 101 und 123 beispielhaft die verschiedenen Gefahrenzeichen ab, mit den Verkehrszeichen 205, 301 und 306 wichtige Vorschriften und Richtzeichen, welche die Vorfahrt regeln, sowie mit den Verkehrszeichen 267 und 276 beispielhaft zwei Verbote. Weiterhin wurden drei verschiedene Geschwindig-

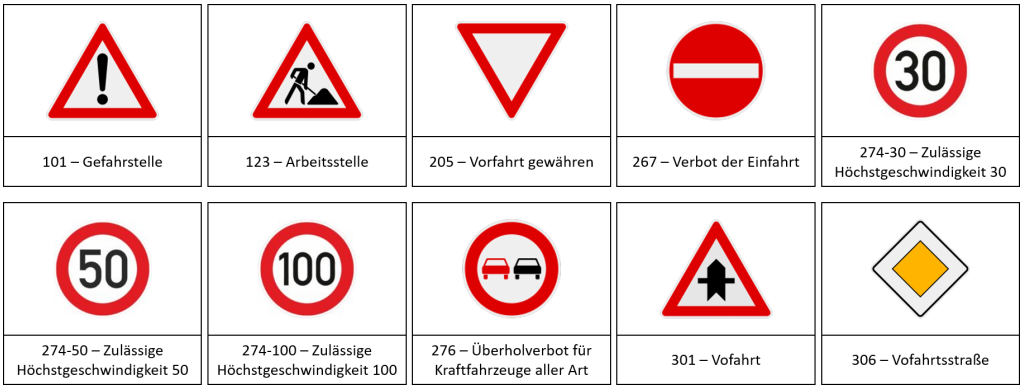


Abbildung 17: Zu klassifizierende Verkehrszeichen, basierend auf [Kor22] und `https://www.strassenausstatter.de/?s=274&post_type=product`, zuletzt aufgerufen am 06.02.2024

keitsbegrenzungen ausgewählt, welche zum Beispiel in Gefahrenstellen, Ortschaften oder als Geschwindigkeitsbegrenzung auf Autobahnen vorkommen [Kor22].

Neben der inhaltlichen Spezifikation von Anforderungen sieht der Referenzprozess MLE.1, wie in BP.2 festgelegt, vor, dass die ML-Anforderungen strukturiert und priorisiert werden. Dies erfolgte zum einen indem die Anforderungen hinsichtlich ihres Bezuges zu den Daten als ML-Anforderungen oder Datenanforderungen klassifiziert wurden und zum anderen indem die Anforderungsspezifikation in insgesamt fünf Abschnitte gegliedert wurde. Dabei wurde in den Abschnitten zum Beispiel zwischen funktionalen sowie Performanz und Ressourcen Anforderungen unterschieden und die Datenstruktur gesondert beschrieben. Auf eine abgestufte Priorisierung der Anforderungen wurde verzichtet, da keine verschiedenen Entwicklungs- bzw. Auslieferungsstände mit verschiedener Funktionalität vorgesehen sind und daher alle Anforderungen als gleichwertig und verbindlich angesehen werden.

MLE.1 BP.3 und BP.4 sehen weiterhin vor, dass die spezifizierten Anforderungen zum einen bezüglich ihrer gegenseitigen Abhängigkeiten und zum anderen bezüglich ihres Einflusses auf die Betriebsumgebung analysiert werden. Wobei die Betriebsumgebung der ML-Komponente in diesem Fall auf die Entwicklungsumgebung für das Training und Testen, welche in Abschnitt 3.4 genauer beschrieben wird, beschränkt ist, da die entwickelte Verkehrszeichenerkennung nicht in einem realen System eingesetzt werden soll. Durch die Analysen soll die Korrektheit, technische Machbarkeit und Testbarkeit der Anforderungen gewährleistet und das Projektmanagement bezüglich der Einhaltung getroffener Annahmen, wie zum Beispiel der angesetzten Zeit für die Durchführung der experimentellen Entwicklung von sieben Wochen, unterstützt werden. Durchgeführt wurden die Analysen

zum einen im Anschluss an das initiale Brainstorming und zum anderen bei jeder Ergänzung oder Änderung einer Anforderung. Aufgrund der übersichtlichen Anzahl von Anforderungen für die experimentelle Entwicklung konnten bei der Analyse immer alle Anforderungen auf Abhängigkeiten untereinander geprüft werden. Als Ergebnis der Analysen wurde die Anforderung in der Spezifikation entweder als freigegeben markiert oder angepasst und anschließend freigegeben.

Die letzten beiden BPs des MLE.1 sind im Rahmen der experimentellen Entwicklung, aufgrund der Projektrahmenbedingungen, nicht anwendbar gewesen. BP.5 besagt, dass eine Konsistenz und beidseitige Rückverfolgbarkeit zwischen den ML-Anforderungen und den Softwareanforderungen bzw. der Softwarearchitektur erzeugt werden soll. Da im Rahmen der experimentellen Entwicklung jedoch nicht vorgesehen war, die Verkehrszeichenerkennung in eine größere Software zu integrieren, existieren keine Softwareanforderungen oder eine Softwarearchitektur, zu welchen eine Konsistenz und Rückverfolgbarkeit hergestellt werden könnte. Die letzte BP jedes MLE-Prozesses und des SUP.11 fordert weiterhin, dass die Ergebnisse aus der Anwendung des Prozesses, im Falle des MLE.1 die vereinbarten ML-Anforderungen sowie die Ergebnisse der Analyse hinsichtlich der Betriebsumgebung, an alle von den Ergebnissen betroffenen Parteien kommuniziert werden. Im Rahmen dieser experimentellen Entwicklung existieren jedoch keine weiteren Parteien, welche durch die Ergebnisse aus der experimentellen Entwicklung betroffen sind, weswegen eine Kommunikation dieser nicht möglich war. In den weiteren Abschnitten wird daher auf den Aspekt der Kommunikation von Arbeitsprodukten an relevante Parteien nicht erneut eingegangen.

Die während der Umsetzung des Prozesses MLE.1 spezifizierten ML-Anforderungen inklusive der Anforderungen an die Daten dienen als Eingabe für alle weiteren MLE-Prozesse und den Prozess SUP.11, welche in den folgenden Unterkapiteln genauer beschrieben werden.

3.2. Datenmanagement

Wie in Abschnitt 2.1 beschrieben, hängt der Erfolg von maschinellem Lernen stark von den während der Entwicklung verwendeten Daten ab, weswegen viele Sicherheitsbedenken und Herausforderungen beim Einsatz von ML auf der Qualität der verwendeten Daten basieren [WSRA20]. Der SUP.11 *Machine Learning Data Management* des ASPICE 4.0 [VDA23b] adressiert diesen Aspekt innerhalb von sechs BPs und verfolgt den Zweck, den spezifizierten Anforderungen entsprechende Daten für das maschinelle Lernen zu erzeugen, die Integrität und Qualität der erzeugten Daten zu gewährleisten und diese allen relevanten Parteien zur Verfügung zu stellen.

Als Grundlage für das Datenmanagement sieht der Prozess SUP.11 innerhalb von BP.1 vor, dass ein Datenmanagementsystem eingeführt wird, welches Datenmanagementaktivitäten, relevante Quellen für Daten, einen Datenlebenszyklus inklusive eines Statusmodells sowie Schnittstellen zu betroffenen Parteien unterstützt. Die Richtlinien zum ASPICE 4.0 [VDA23a] schränken jedoch ein, dass das Datenmanagementsystem auf die Speicherung der Daten und Metadaten beschränkt werden kann, insofern dies im Projektkontext angemessen ist. Für diese experimentelle Entwicklung ist diese Einschränkung zutreffend, da sich frühzeitig aufgrund der Anforderung, dass alle verwendeten Daten entweder selbst erzeugt oder frei verfügbar sein müssen, dazu entschieden wurde, den frei verfügbaren *German Traffic Sign Recognition Benchmark (GTSRB)* Datensatz[4] als Grundlage für die Entwicklung der Verkehrszeichenerkennung zu verwenden. Dieser Datensatz beinhaltet über 50.000 Bilder von Verkehrszeichen, welche insgesamt 43 verschiedene Klassen abbilden. Darunter sind alle zehn Verkehrszeichen, für welche die Verkehrszeichenerkennung während der experimentellen Entwicklung trainiert werden soll, enthalten. Die Bilder des Datensatzes zeichnen sich weiter durch ihre Realitätsnähe aus, da sie verschiedene Distanzen zum Verkehrszeichen, Belichtungen, Witterungsbedingungen, Verdeckungen und Rotationen berücksichtigen und sind somit auch für den Absicherungsdatensatz geeignet [SSSI11]. Datenmanagementaktivitäten wie das Beschaffen und Labeln weiterer Daten sowie auch ein Datenlebenszyklus inklusive einem Statusmodell sind daher in der experimentellen Entwicklung nicht erforderlich. Weiterhin werden keine Schnittstellen vom Datenmanagementsystem zu weiteren Datenquellen oder anderen betroffenen Parteien benötigt, da diese in dem Projektkontext nicht existieren. Aus diesen Gründen wurde sich dazu entschieden, den Windows Explorer als Datenmanagementsystem während der experimentellen Entwicklung zu verwenden und notwendige Metadaten wie ein Identifikator und die Klassenzuordnung des Bildes über verschiedene Excel Tabellen zu speichern.

Neben der Einführung eines Datenmanagementsystems fordert der SUP.11 innerhalb von BP.2, dass ein Ansatz beschrieben wird, um die Qualität der Daten sicherzustellen. Dieser soll zum einen auf definierten ML-Qualitätskriterien aufbauen und zum anderen die Vermeidung von *Bias* unterstützen. Innerhalb des GTSRB Datensatzes wurden folgende drei Qualitätskriterien für die Daten festgelegt, wobei als Track eine Sequenz von Bildern desselben realen Verkehrszeichen über verschiedene Zeitpunkte der Vorbeifahrt an dem Verkehrszeichen hinweg bezeichnet wird, wie in Abb. 18 dargestellt.

- Ein Track benötigt mindestens 30 Bilder.
- Eine Klasse an Verkehrszeichen muss in mindestens 9 Tracks abgebildet sein.
- Ein Track mit mehr als 30 Bildern muss auf 30 Bilder reduziert werden, welche entlang der Sequenz zeitlich gleich weit entfernt sind.

[4] `https://benchmark.ini.rub.de/gtsrb_dataset.html`, zuletzt abgerufen am 30.01.2024

Abbildung 18: Track eines Verkehrszeichens, Bildquelle [SSSI11]

Zum einen stellen diese Qualitätskriterien sicher, dass die Verkehrszeichen des Datensatzes in der realen Welt eine minimale Häufigkeit und Relevanz besitzen, da sie in mindestens 9 Tracks abgebildet sein müssen. Zum anderen wird durch das Reduzieren jedes Tracks auf genau 30 Bilder erreicht, dass innerhalb des Datensatzes eine gewisse Diversität besteht. Andernfalls könnten zum Beispiel aufgrund einer langsamen Vorbeifahrt an einem Verkehrszeichen sehr viele ähnliche Bilder im Datensatz zu einem *Bias* führen. Weiterhin wurde festgelegt, dass die Aufzeichnung der Verkehrszeichen sowohl im Frühling als auch im Herbst erfolgen soll, wodurch zusätzlich die Diversität in dem Datensatz durch verschiedene Umweltfaktoren erhöht wird [SSSI11]. Neben den für den GTSRB Datensatz definierten ML-Qualitätskriterien wurde während der experimentellen Entwicklung, wie innerhalb der Anforderungen spezifiziert, als weiteres Qualitätskriterium festgelegt, dass für die zehn relevanten Klassen jeweils mindestens 1400 unterschiedliche Bilder existieren müssen. Auch dies wird durch den GTSRB Datensatz erfüllt.

Die weiteren BPs drei bis fünf beschreiben, dass die ML-Daten zu sammeln und vorzuverarbeiten sind und dabei der definierte Ansatz zur Sicherstellung der Datenqualität anzuwenden ist. Die Sammlung der Daten im GTSRB Datensatz erfolgte, indem zirka zehn Stunden Videomaterial während der Fahrt auf verschiedenen öffentlichen Straßen in Deutschland im März, Oktober und November 2010

aufgezeichnet wurden. Diese Daten wurden anschließend manuell bezüglich der je Frame enthaltenen Verkehrszeichen annotiert und diese als Bilder extrahiert. Weiterhin wurden die zuvor beschriebenen drei ML-Qualitätskriterien angewandt, sodass aus 70 Klassen an Verkehrszeichen mit in Summe 2.416 Instanzen abschließend 43 Klassen mit insgesamt 1.700 Instanzen den finalen Datensatz bildeten [SSSI11]. Für die experimentelle Entwicklung wurden die 43 Klassen auf die zehn für diese Arbeit relevanten reduziert und die Testdaten des GTSRB Datensatz in einem weiteren Vorverarbeitungsschritt in .jpg Dateien konvertiert.

Mit der Umsetzung des Prozesses SUP.11 ist eine Datenbasis für das Training und Testen der Verkehrszeichenerkennung erstellt worden, deren Qualität über einen systematischen Ansatz sichergestellt wurde. Die Aufteilung der Datenbasis in verschiedene Datensätze für das Training, die Validierung und das Testen sowie auch die Verwendung der Daten werden im ASPICE durch die Prozesse MLE.3 und MLE.4 adressiert und daher im Folgenden genauer beschrieben.

3.3. Architekturerstellung

Bevor die ML-Komponente implementiert und trainiert werden kann, ist zunächst innerhalb einer Architektur festzulegen, wie diese aufgebaut sein muss, um die geforderte Funktionalität bereitstellen zu können. Diesen Schritt der Architekturerstellung beschreibt das ASPICE 4.0 [VDA23b] innerhalb des Prozesses MLE.2, welcher neben der eigentlichen Erstellung der Architektur auch den Zweck verfolgt, die Konsistenz zwischen der ML-Architektur und den spezifizierten Anforderungen sicherzustellen und die ML-Architektur bezüglich im Projekt definierter Kriterien zu evaluieren. Die vollständige aus der Anwendung des Prozesses MLE.2 entstandene Architekturspezifikation kann in Anhang B nachgeschlagen werden.

Die erste BP des Prozesses MLE.2 besagt, dass eine ML-Architektur zu erstellen und zu dokumentieren ist, welche die einzelnen Elemente der Architektur spezifiziert. Als mögliche Elemente der Architektur nennt das ASPICE Elemente zur Vor- oder Nachverarbeitung von Eingangsdaten und Ausgaben des ML-Modells, *Hyperparameter* des ML-Modells, welche für das Training und Testen benötigt werden, und das ML-Modell selbst. Um ein geeignetes ML-Modell für eine Verkehrszeichenerkennung auf Basis eines CNN zu erstellen, wurde innerhalb der experimentellen Entwicklung zunächst eine Literaturrecherche durchgeführt. Als Ergebnis dieser wurde sich dazu entschieden, die Verkehrszeichenklassifikation auf Basis eines *you only look once (YOLO)v5*-Klassifikationsmodells aufzubauen, denn verschiedene Quellen zeigen, dass die YOLO-Architektur eine hohe Performanz bei der Verkehrszeichenerkennung erreichen kann und gerade hinsichtlich der benötigten Rechenzeit anderen Architekturen wie zum Beispiel *Faster R-CNN* überlegen ist [NAIH22, QY21, ZY22]. Innerhalb der klassischen YOLOv5-Architektur werden

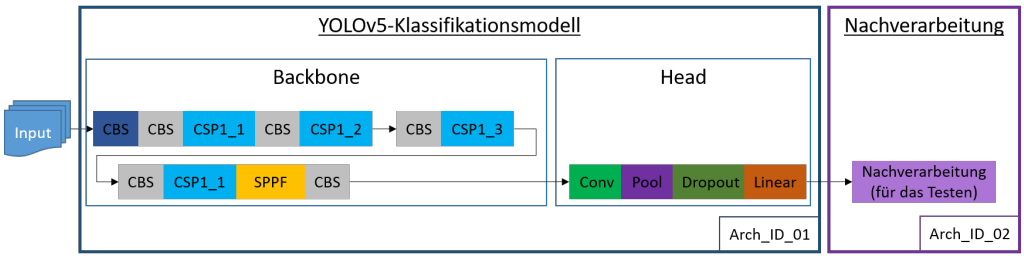


Abbildung 19: Blockdiagramm der ML-Architektur, basierend auf [YBS+23]

aus den Eingangsdaten durch einen *Feature Extractor Feature Maps* erzeugt. Diese werden anschließend von dem Detektor zur Ermittlung von Begrenzungsrahmen um die Objekte weiterverarbeitet und abschließend werden die gefundenen Objekte durch den Klassifizierer einer Klasse zugeordnet [SP21, ZY22]. Bei den YOLOv5-Klassifikationsmodellen im Speziellen handelt es sich um, auf Basis von ImageNet[5] vortrainierte Modelle für die reine Klassifikation von Bildern, welche eine vereinfachte Architektur ohne den Detektor besitzen [JCS+22, YBS+23]. Wie das in Abb. 19 abgebildete Blockdiagramm der entwickelten ML-Architektur zeigt, wird aufgrund der Verwendung des YOLOv5-Klassifikationsmodells kein Element zur Vorverarbeitung der Eingangsdaten benötigt, da dieses Bilder verschiedener Größen verarbeiten kann. Das YOLOv5-Klassifikationsmodell selber besteht aus dem sogenannten *Backbone*, welcher durch *FOCUS*, *cross stage parial (CSP)*, *contextual block separable (CBS)* und *spatial pyramid pooling fast (SPPF)* Strukturen *Feature Maps* aus den Eingangsdaten extrahiert und dem *Head*, welcher aus drei *Convolutional Layern* besteht und die Vorhersage für die Eingabe erzeugt, indem er jeder Klasse eine bestimmte Wahrscheinlichkeit zuweist [YBS+23]. Da die Ausgabe des Modells für die Tests als je eine Textdatei pro Bild erzeugt wird, in welcher die Wahrscheinlichkeiten der einzelnen Klassen angegeben sind, wird zur Nachverarbeitung bei den Tests zusätzlich ein Element benötigt, welches die Textdokumente einliest und mit der tatsächlichen Klasse des Bildes abgleicht.

Neben den im Blockdiagramm dokumentierten Elementen der ML-Architektur fordert die BP.1 auch, dass die benötigten *Hyperparameter* dokumentiert werden. BP.2 sieht darüber hinaus vor, dass die Wertebereiche der *Hyperparameter* definiert und initiale Werte für den Start des Trainings festgelegt werden. Innerhalb der Dokumentation zu dem YOLOv5-Klassifikationsmodell sind bereits die anpassbaren *Hyperparameter* des Modells beschrieben und jeweils Standardwerte angegeben, von denen ein Auszug für das Training in Tabelle 3 aufgeführt ist. So ist zum Beispiel standardmäßig festgelegt, dass für zehn Epochen mit einer *Batch* Größe von 64 trainiert wird und dabei das vortrainierte „yolov5s-cls.pt“ Klassifikationsmodell mit etwa 5,4 Millionen Parametern verwendet wird. Alternativ könnte bei einer starken Überanpassung, während des Trainings, auch

[5] `https://www.image-net.org/`, zuletzt aufgerufen am 28.09.2023

Parameter	Beschreibung	Standardwert
–model	Pfad zum vortrainierten Modell	yolov5s-cls.pt
–data	Pfad zu den Trainings- und Validierungsdaten	imagenette160
–epochs	Anzahl der Epochen	10
–batch-size	Größe eines *Batch*	64
–cache	Zwischenspeichern im RAM oder auf der Festplatte	ram
–device	Angabe der zu verwendenden GPU oder CPU	
–optimizer	Zu verwendender Optimierer	Adam
–lr0	Initiale Lernrate	0,001
–decay	Gewichtsabnahme	5e-5
–dropout	Menge des *Dropouts*	None

Tabelle 3: Auszug der *Hyperparameter* für das Training des YOLOv5-Klassifikationsmodells, basierend auf `https://github.com/ultralytics/yolov5/blob/master/classify/train.py`, zuletzt aufgerufen am 03.11.2023

eine kleine Variante „yolov5n-cls.pt“ verwendet werden, welche nur 2,5 Millionen Parameter besitzt oder bei einer Unteranpassung auch eines von drei größeren Modellen mit bis zu 48,1 Millionen Parametern. Innerhalb des Kapitels vier der Architekturspezifikation wurde auf die Dokumentation des YOLOv5-Klassifikationsmodells Bezug genommen und die Parameter, welche für die experimentelle Entwicklung gegenüber den Standardwerten anzupassen sind, beschrieben. So wurde für das Training definiert, dass die GPU null verwendet werden soll und dass „../TrainVal_Dataset_01“ für den Parameter „–data“ als Dateipfad zu den Trainings- und Validierungsdaten übergeben werden soll. Für die Parameter bezüglich des Testens wurde in gleicher Weise in der Architekturspezifikation auf die Dokumentation des YOLOv5-Klassifikationsmodells verwiesen und die von den Standardwerten abweichenden Parameter explizit beschrieben. Ein wichtiger Parameter dabei ist zum Beispiel „–save-txt“, wodurch die Speicherung der Ergebnisse pro Bild innerhalb einer Textdatei aktiviert wird. Neben den *Hyperparametern* des YOLOv5-Klassifikationsmodells wurde außerdem in der Architekturspezifikation festgelegt, dass der Trainingsdatensatz initial aus 750 Bildern pro Klasse bestehen soll und der Validierungsdatensatz aus 150 Bildern pro Klasse.

Die dritte BP des Prozesses zur Architekturerstellung sieht vor, dass Kriterien für die Analyse der Architekturelemente definiert und entsprechend angewendet werden. Innerhalb der experimentellen Entwicklung wurden als Analysekriterien

die Performanz, Erklärbarkeit und freie Verfügbarkeit sowie Ziele hinter jedem Kriterium in Kapitel fünf der Architekturspezifikation definiert. Alle drei Kriterien wurden während der Literaturrecherche für die Auswahl eines geeigneten ML-Modells bereits berücksichtigt und erfüllen für die definierte ML-Architektur das beschriebene Ziel. Bezüglich der Erklärbarkeit der Funktionsweise der Architektur besteht für das YOLOv5-Klassifikationsmodell zum Beispiel eine umfangreiche Dokumentation der Ersteller[6] und unter anderem in [YBS+23] wird der Algorithmus sowie die einzelnen Architekturelemente im Detail beschrieben.

Der Prozess MLE.2 fordert weiter innerhalb der BP.4, dass die Schnittstellen sowohl innerhalb der ML-Komponente zwischen den einzelnen Architekturelementen dokumentiert werden als auch die Schnittstellen von der ML-Komponente zu den weiteren Softwarekomponenten. Da es innerhalb der experimentellen Entwicklung nicht vorgesehen ist, die Verkehrszeichenerkennung in eine größere Software mit weiteren Komponenten zu integrieren, bestehen auch keine Schnittstellen von der ML-Komponente nach außen, welche dokumentiert werden könnten. Innerhalb der ML-Komponente besteht zumindest für das Testen eine Schnittstelle zwischen dem YOLOv5-Klassifikationsmodell und dem Element für die Nachverarbeitung, welche in Kapitel sechs der Architekturspezifikation genau beschrieben wurde. Weitere Schnittstellen zum Beispiel zwischen den verschiedenen Schichten innerhalb des YOLOv5-Klassifikationsmodells wurden während der experimentellen Entwicklung nicht weiter beschrieben, da diese bei der Verwendung des vortrainierten ML-Modells nicht anzupassen sind.

Innerhalb der Architektur der ML-Komponente sind weiterhin gemäß der BP.5 Ziele bezüglich des Ressourcenverbrauches aller relevanten Architekturelemente zu definieren und zu dokumentieren. Für die experimentelle Entwicklung ergaben sich die Ressourcenziele aus den spezifizierten Anforderungen bezüglich der Performanz und der Ressourcen, welche innerhalb des Kapitels sieben der Architekturspezifikation jeweils an die beiden Architekturelemente, dem ML-Modell und dem Element zur Nachverarbeitung der Ausgaben für den Test, zugewiesen wurden. Für das Nachverarbeitungselement wurde so zum Beispiel basierend auf der Anforderung Req_010 definiert, dass dieses maximal fünf Minuten für die Auswertung der Testergebnisse hinsichtlich des Verhältnisses korrekter Vorhersagen, auf der in den Anforderungen definierten Hardware, benötigen darf.

Um die Konsistenz zwischen den ML-Anforderungen und der ML-Architektur sicherzustellen, was ein wesentlicher Bestandteil des Prozesszweckes von MLE.2 ist, fordert die BP.6, dass eine bidirektionale Rückverfolgbarkeit zwischen den Architekturelementen und den spezifizierten Anforderungen hergestellt und die Konsistenz gewährleistet wird. Eine bidirektionale Rückverfolgbarkeit meint

[6] `https://github.com/ultralytics/yolov5/blob/master/README.md`, zuletzt aufgerufen am 28.09.2023

dabei, dass einerseits für jede Anforderung nachvollzogen werden kann, innerhalb welcher Architekturelemente diese umgesetzt wird, und dass andererseits für jedes Architekturelement nachvollzogen werden kann, welche Anforderungen dieses umzusetzen hat. Im Rahmen der experimentellen Entwicklung wurde die bidirektionale Rückverfolgbarkeit über die Anforderungsspezifikation, siehe Anhang A, sichergestellt. Innerhalb dieser wurde zum einen für jede Anforderung unter Referenzen angegeben, welche Architekturelemente von der Anforderung betroffen sind, und zum anderen kann über das Attribut „Architectural Element" nach den einzelnen Architekturelementen gefiltert werden und so die Gesamtheit aller für ein Architekturelement relevanten Anforderungen nachvollzogen werden. Für Anforderungen, welche keinen Bezug zu der Architektur besitzen, wurde ein Schrägstrich in dem Attribut eingetragen. Dieses Konzept zur Erzeugung der Rückverfolgbarkeit ist innerhalb des Kapitels acht der Architekturspezifikation ebenfalls einmal erklärt und die Referenz zu der Anforderungsspezifikation dokumentiert. Die Konsistenz zwischen der Architektur und den spezifizierten Anforderungen wurde während des Eintragens der Referenzen in der Anforderungsspezifikation geprüft und mit dem Eintragen einer Referenz bestätigt. Die gesamtheitliche Konsistenz zwischen der Anforderungsspezifikation und der Architektur ist darüber gewährleistet, dass für jede Anforderung entweder ein umsetzendes Architekturelement eingetragen ist, oder die Anforderung durch einen Schrägstrich als nicht relevant für die ML-Architektur markiert wurde.

Als Ergebnis der Architekturerstellung ist definiert, wie die ML-Komponente für die Verkehrszeichenerkennung implementiert werden muss, um die spezifizierten Anforderungen erfüllen zu können und auf Basis welcher initialen Werte für die *Hyperparameter* das im folgenden Unterkapitel thematisierte Training gestartet werden soll.

3.4. Machine Learning Training

Wie in Abschnitt 2.1 beschrieben lernt ein ML-Modell, während des Trainings, selbstständig aus Trainingsdaten Regeln bzw. Transformationen der Daten, um das repräsentierte Problem zu lösen und eine gewünschte Ausgabe zu erzeugen. Dabei ist während des Trainings eine Validierung und die Anpassung der *Hyperparameter* erforderlich, um sicherzustellen, dass das Modell die generell gültigen Zusammenhänge aus den Daten lernt und es nicht zu einer Über- oder Unteranpassung gegenüber den Trainingsdaten kommt [KS17, W$^+$20]. Das ASPICE 4.0 [VDA23b] fasst aus diesem Grund innerhalb des Prozesses MLE.3 die Validierung mit dem Training zusammen und verfolgt mit dem Prozess den Zweck, dass das ML-Modell optimiert wird, sodass dieses die spezifizierten Anforderungen erfüllt. Für die experimentelle Entwicklung soll während des Trainings die zuvor

beschriebene ML-Komponente auf Basis der Daten aus dem GTSRB Datensatz[7] lernen, Bilder von zehn verschiedenen Verkehrszeichen korrekt zu klassifizieren.

Die erste BP des Prozesses MLE.3 sieht vor, dass ein genereller Ansatz für das Training und die Validierung des ML-Modells spezifiziert wird, welcher dabei unterstützt die definierten ML-Anforderungen zu erfüllen und mindestens die folgenden vier Themen abdeckt. In Anhang C kann der entsprechende, während der experimentellen Entwicklung spezifizierte Trainings- und Validierungsansatz vollständig nachgeschlagen werden.

- Kriterien für den Start und das Ende des Trainings und der Validierung.
- Einen Ansatz für die Anpassung der *Hyperparameter*.
- Einen Ansatz für die Erzeugung und Modifikation des Trainings- und Validierungsdatensatzes.
- Eine Beschreibung der Trainings- und Validierungsumgebung.

Als Startkriterium des Trainings wurde innerhalb des Kapitels zwei des Trainings- und Validierungsansatzes aufgelistet, dass Anforderungen an die ML-Komponente, eine ML-Architektur sowie eine Datenbasis für das Training als Ergebnisse der zuvor beschriebenen Prozesse freigegeben vorhanden sein müssen. Weiter wurden zwei alternative Trainingsendekriterien beschrieben, welche zum einen den positiven Abschluss des Trainings als auch den Abbruch des Trainings nach einer maximal zur Verfügung stehenden Trainingszeit definieren.

Bezüglich der Anpassung der *Hyperparameter* wurde in Kapitel drei des Trainings- und Validierungsansatzes zum einen beschrieben, dass die falsch klassifizierten Validierungsdaten bezüglich ihrer Zusammenhänge zu betrachten sind, um systematischen Problemen des Modells oder *Bias* innerhalb der Daten durch die Anpassung der *Hyperparameter* entgegen wirken zu können. Zum anderen wurde beschrieben, dass die durchschnittliche Korrektheit der Vorhersage für die Validierungsdaten und der *Trainings- und Validierungsloss* gemäß der Kreuzentropie, wie in Gleichung (8) angegeben, pro Epoche als Metriken zu erheben und im Anschluss an die Trainingsiteration zu interpretieren sind.

$$l(x, y) = \frac{\sum_{n=1}^{N} - \sum_{c=1}^{C} w_c \log \frac{exp(x_{n,c})}{\sum_{i=1}^{C} exp(x_{n,i})} y_{n,c}}{N} \tag{8}$$

Dabei steht x für die Eingangsdaten und y für die Label, w entspricht den trainierten Gewichten, C der Anzahl an Klassen und N der Anzahl an verwendeten Daten [PyT23]. Konkreter besagt der spezifizierte Ansatz zur Anpassung der *Hyperparameter* dabei zum Beispiel, dass, wenn der *Trainings- und Validierungsloss* in mehreren abschließenden Epochen auf einem konstanten

[7] `https://benchmark.ini.rub.de/gtsrb_dataset.html`, zuletzt abgerufen am 30.01.2024

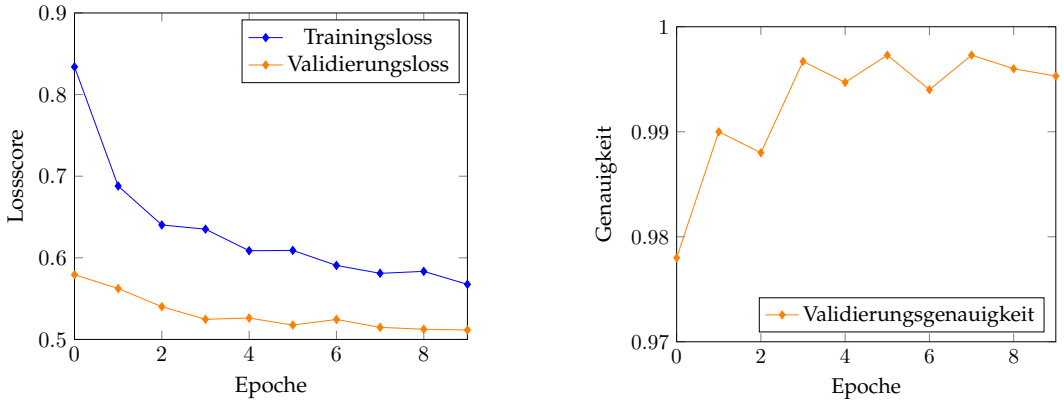
Niveau stagnieren, die Anzahl der Epochen in einer weiteren Trainingsiteration zu reduzieren ist, wodurch eine Überanpassung vermieden werden soll.

Als Drittes wurde in Kapitel vier des spezifizierten Trainings- und Validierungsansatzes auf den Ansatz zur Erzeugung und Modifikation der Trainings- und Validierungsdaten eingegangen. Für die Erzeugung der Datensätze ist dabei ein VBA Skript, das innerhalb einer .xlsm Datei implementiert wurde, zu verwenden, welches eine parametrisierbare Anzahl an zufälligen Trainings- und Validierungsdaten aus der durch die Anwendung des SUP.11 erzeugten Datenbasis auswählt und als Trainings- bzw. Validierungsdatensatz speichert. Jedem Trainings- bzw. Validierungsdatensatz ist dabei ein eindeutiger Identifikator zuzuweisen, sodass dokumentiert werden kann, mit welchen Daten das ML-Modell trainiert wurde, um das Training reproduzieren zu können. Bezüglich der Modifikation der Trainings- und Validierungsdaten wurde weiter beschrieben, dass im Falle systematischer Probleme aufgrund der verwendeten Daten, zum Beispiel bei der Klassifikation von aufgrund natürlicher Umwelteinflüsse gestörten Bildern, der verwendete Trainings- und Validierungsdatensatz entweder neu zufällig erzeugt oder gezielt um Daten des GTSRB Datensatzes ergänzt oder reduziert werden kann, wobei ein neuer Identifikator zu vergeben ist.

Abschließend thematisiert der Trainings- und Validierungsansatz wie die Trainings- und Validierungsumgebung für die experimentelle Entwicklung aufzusetzen ist. Dabei wurde zum einen darauf eingegangen, wie die Ordnerstruktur für das Training anzulegen ist, sodass das Modell auf die korrekten Trainings- und Validierungsdaten zugreifen kann. Zum anderen wurde definiert, welche Hardware zu verwenden und welche Software für die Verwendung des YOLOv5-Klassifikationsmodells zu installieren ist.

Die zweite und dritte BP des Prozesses MLE.3 fordern die eigentliche Erzeugung des Trainings- und Validierungsdatensatzes sowie die Implementierung und Optimierung des ML-Modells gemäß des beschriebenen Trainings- und Validierungsansatzes und der erstellten Architekturspezifikation. Für die experimentelle Entwicklung wurde diesbezüglich die Trainingsumgebung, wie im Trainings- und Validierungsansatz beschrieben, aufgesetzt, indem Python[8], Git[9], Pytorch[10], das Nvidia Cuda Toolkit[11] und Jupyter Notebook[12] installiert und das YOLOv5-*Repository*[13] geklont wurden. Der Trainings- und Validierungsdatensatz wurde, durch die Anwendung des im Trainings- und Validierungsansatz beschriebenen VBA Skripts, zufallsbasiert erzeugt. Dabei wurde in der Ex-

[8] `https://www.python.org/`, zuletzt aufgerufen am 02.11.2023
[9] `https://git-scm.com/`, zuletzt aufgerufen am 02.11.2023
[10] `https://pytorch.org/`, zuletzt aufgerufen am 02.11.2023
[11] `https://developer.nvidia.com/cuda-toolkit`, zuletzt aufgerufen am 02.11.2023
[12] `https://jupyter.org/`, zuletzt aufgerufen am 02.11.2023
[13] `https://github.com/ultralytics/yolov5`, zuletzt aufgerufen am 02.11.2023



(a) *Trainings- und Validierungsloss* je Epoche

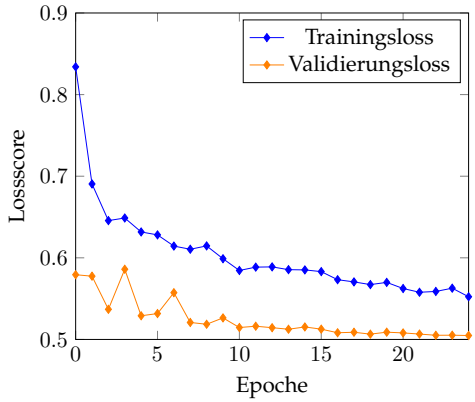
(b) Validierungsgenauigkeit je Epoche

Abbildung 20: Metriken der ersten Trainingsiteration, eigene Darstellungen

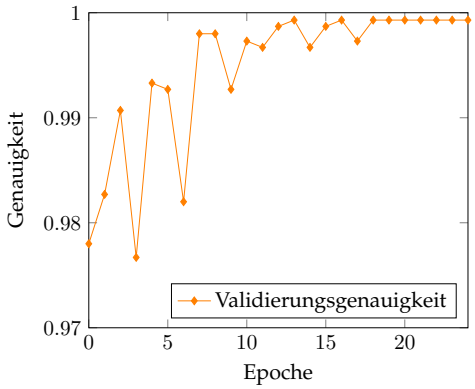
cel Tabelle für jede Klasse an Verkehrszeichen ein Tabellenblatt angelegt und zunächst die Dateinamen aller Bilder der zehn relevanten Klassen an Verkehrszeichen des GTSRB Datensatzes in jeweils eine Zeile des jeweiligen Tabellenblatts eingefügt. Anschließend wurde jedem Dateinamen durch die Methode „WorksheetFunction.RandBetween(Arg1, Arg2)“[14] eine Zufallszahl, zwischen eins und der Anzahl an Bildern der Klasse, zugewiesen. Weiter wurden alle Dateinamen nach ihrer Zufallszahl aufsteigend sortiert und die ersten 750 Bilder pro Klasse als Trainingsdaten in den Trainingsdatensatz kopiert sowie die nächsten 150 Bilder als Validierungsdaten in den Ordner des Validierungsdatensatzes. Für das eigentliche Training des YOLOv5-Klassifikationsmodells wurde ein Jupyter Notebook erstellt, über welches das Training und die Validierung, durch den Aufruf der Methode „classify/train.py“ mit den in der Architekturspezifikation, siehe Anhang B, festgelegten initialen *Hyperparametern*, durchgeführt wurde.

Als Ergebnis der ersten Trainingsiteration konnte, wie in Abb. 20 dargestellt, eine maximale Genauigkeit bzw. Korrektheit der Vorhersagen auf den Validierungsdaten von circa 99,7% erreicht werden. Da die Reduzierungen des *Trainings- und Validierungsloss* bis zu den letzten Epochen nicht stagniert sind, wurde sich gemäß dem spezifizierten Ansatz für die Anpassung der *Hyperparameter* dazu entschieden, innerhalb einer zweiten Trainingsiteration, unter Verwendung derselben Datensätze, das gleiche Klassifikationsmodell für insgesamt 25 Epochen neu zu trainieren. Wie in Abb. 21b dargestellt, konnte innerhalb dieser zweiten Trainingsiteration eine maximale Genauigkeit auf den Validierungsdaten von circa 99,9% in den Epochen 13, 16 und konstant ab Epoche 18 erreicht werden. Weiterhin

[14] `https://learn.microsoft.com/de-de/office/vba/api/excel.worksheetfunction.randbetween`, zuletzt aufgerufen am 02.11.2023



(a) *Trainings- und Validierungsloss* je Epoche

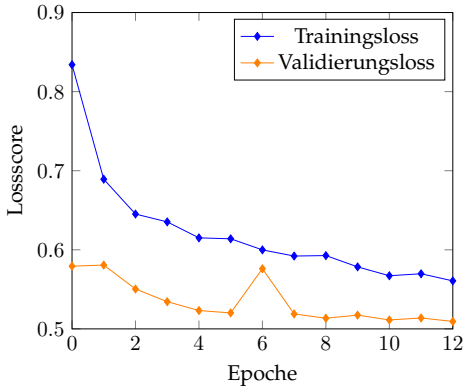


(b) Validierungsgenauigkeit je Epoche

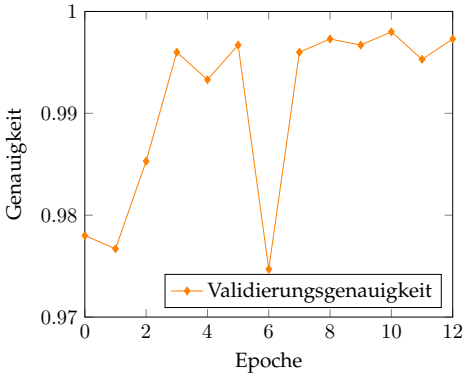
Abbildung 21: Metriken der zweiten Trainingsiteration, eigene Darstellungen

ist in Abb. 21a ersichtlich, dass *Trainings- und Validierungsloss* zwar bis zu den letzten Epochen fallen, sich der Validierungsloss jedoch ab Epoche zwölf nur noch sehr gering reduziert und teilweise innerhalb einzelner Epochen sogar ansteigt. Aus diesem Grund wurde sich dazu entschieden, die Anzahl der Epochen, in einer dritten Trainingsiteration mit denselben Datensätzen, auf 13 zu reduzieren. Wie Abb. 22 zeigt, konnte so eine maximale Validierungsgenauigkeit von 99,8% erreicht werden und in der letzten Epoche, welche den geringsten *Trainings- und Validierungsloss* besitzt, eine Genauigkeit auf den Validierungsdaten von circa 99,7%. Da *Trainings- und Validierungsloss* nur in den letzten Epochen zu stagnieren beginnen und eine Korrektheit der Vorhersagen für die Validierungsdaten von mehr als 98% erreicht wurde, sieht der spezifizierte Ansatz für die Anpassung der *Hyperparameter* keine weitere Anpassung dieser auf Basis der erhobenen Metriken vor.

Für das trainierte Modell wurde, wie im Trainings- und Validierungsansatz definiert, nun weiter analysiert, ob Zusammenhänge in den falsch klassifizierten Bildern existieren, um systematische Probleme des Modells auszuschließen. Die sechs falsch klassifizierten Bilder sind in Abb. 23 abgebildet. Auffällig ist, dass fünf dieser Bilder den Klassen „4_ZulässigeHöchstgeschwindigkeit30“ und „5_ZulässigeHöchstgeschwindigkeit50“ angehören und jeweils als eine andere zulässige Höchstgeschwindigkeit klassifiziert wurden. Das sechste falsch klassifizierte Verkehrszeichen gehört der Klasse „8_Vorfahrt“ an und wurde als Vorfahrt gewähren Schild klassifiziert. Alle falsch klassifizierten Bilder verfügen mit weniger als 40x40 Pixeln über eine vergleichsweise geringe Auflösung und bei vier der sechs falsch klassifizierten Bilder ist eine hohe bzw. geringe Belichtung erkennbar. Da innerhalb des Validierungsdatensatzes jedoch auch einige vergleichbare kleine und hoch bzw. niedrig belichtete Bilder korrekt klassifiziert wurden, ist nicht von einem syste-



(a) *Trainings- und Validierungsloss* je Epoche



(b) Validierungsgenauigkeit je Epoche

Abbildung 22: Metriken der dritten Trainingsiteration, eigene Darstellungen

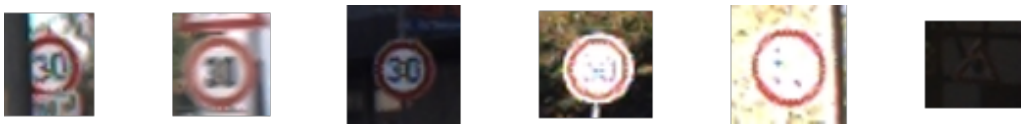


Abbildung 23: Falsch klassifizierte Bilder innerhalb der dritten Trainingsiteration, basierend auf dem GTSRB Datensatz

matischen Problem oder einem *Bias* innerhalb der Daten auszugehen, sodass mit der dritten Trainingsiteration das im Trainings- und Validierungsansatz definierte Trainingsendekriterium für den positiven Abschluss des Trainings erreicht wurde.

BP.4 des Prozesses MLE.3 sieht weiter vor, dass eine bidirektionale Rückverfolgbarkeit zwischen den verwendeten Trainings- und Validierungsdaten und den relevanten ML-Anforderungen an die Daten sichergestellt wird. Für die experimentelle Entwicklung wurde für den Trainings- und Validierungsdatensatz daher, über den spezifizierten Trainings- und Validierungsansatz, eine Namenskonvention festgelegt und beschrieben, dass bei der Modifikationen des Datensatzes der jeweilige Identifikator im Namen des Datensatzes hoch zu zählen ist. Innerhalb der Anforderungsspezifikation wurde über das Attribut „Training & Validation Data“ die Referenz zu dem verwendeten Trainings- und Validierungsdatensatz eingefügt, wobei Anforderungen, welche keine Relevanz zu den für das Training verwendeten Daten besitzen, durch einen Schrägstrich gekennzeichnet wurden. Wie auch für die Rückverfolgbarkeit der Architekturelemente und Anforderungen kann durch Filterung innerhalb der Anforderungsspezifikation nachvollzogen werden, welche Anforderungen jeweils für die beiden Datensätze relevant sind und innerhalb welchem Datensatz eine bestimmte Anforderung umgesetzt wurde. Da während der experimentellen Entwicklung nur ein Trainings-

und Validierungsdatensatz erzeugt wurde, sind alle relevanten Anforderungen innerhalb desselben initialen Datensatzes mit dem Identifikator 01 umgesetzt worden.

Abschließend fordert der Prozess MLE.2 mit der BP.5, dass die Ergebnisse der Optimierung des ML-Modells zusammengefasst und alle relevanten Parteien über das vereinbarte trainierte ML-Modell informiert werden. Dies wurde für die experimentelle Entwicklung durch die vorangegangene Zusammenfassung der Optimierung innerhalb der drei Trainingsiterationen sichergestellt, wobei wie zuvor beschrieben keine Kommunikation des vereinbarten trainierten ML-Modells im Rahmen der experimentellen Entwicklung erfolgen konnte.

Mit der Umsetzung des MLE.3 wurde in der experimentellen Entwicklung, basierend auf der bereitgestellten Datenbasis, ein ML-Modell gemäß der spezifizierten ML-Architektur trainiert, welches in der Lage zu sein scheint, innerhalb der folgend beschriebenen Tests, die gestellten ML-Anforderungen zu erfüllen.

3.5. Machine Learning Model Testing

Da durch die Anpassung der *Hyperparameter* auf Basis der verwendeten Validierungsdaten Informationen über diese in das ML-Modell aufgenommen werden, kann es wie in Abschnitt 2.1 beschrieben zu einer Überanpassung auf den Validierungsdaten kommen, obwohl nie direkt mit diesen Daten die Gewichte innerhalb des ML-Modells angepasst wurden. Aus diesem Grund ist die Genauigkeit des trainierten ML-Modells auf den Validierungsdaten als Maß für die Generalisierungsfähigkeit nicht ausreichend, sondern es sind abschließende Tests auf neuen Daten durchzuführen, welche von dem Modell bisher nicht verarbeitet worden sind [KS17]. Das ASPICE 4.0 [VDA23b] beschreibt dieses abschließende Testen in dem Prozess MLE.4, welcher den Zweck verfolgt, die Einhaltung der spezifizierten ML-Anforderungen durch das trainierte und das im realen System einzusetzende ML-Modell sicherzustellen.

Die erste BP des Prozesses MLE.4 besagt, wie auch beim MLE.3, dass ein genereller Ansatz für die Umsetzung des Prozesses zu spezifizieren ist, welcher mindestens die folgenden sechs Themen abdecken muss. Der entsprechend während der experimentellen Entwicklung spezifizierte Testansatz kann vollständig in Anhang D eingesehen werden.

- Kriterien für den Start und das Ende des Testens.
- Einen Ansatz für die Erzeugung und Modifikation des Testdatensatzes.
- Eine Beschreibung der in den Anforderungen definierten Testszenarien inklusive der Verteilung ihrer Charakteristiken.

- Die Verteilung und Häufigkeit jedes ML-Testszenarios innerhalb des Testdatensatzes.
- Das erwartete Ergebnis jedes Testdatums.
- Eine Beschreibung der Testumgebung und -infrastruktur.

Als Kriterien für den Start des Testens wurde, wie in Kapitel zwei des Testansatzes beschrieben, definiert, dass freigegebene ML-Anforderungen, ein abgestimmtes trainiertes ML-Modell sowie eine qualitätsgesicherte Datenbasis für das Testen zur Verfügung stehen müssen. Weiter wurde als Testendekriterium festgelegt, dass die erzeugten Testdaten von dem ML-Modell verarbeitet wurden und eine Aussage über die Erfüllung aller für den Test relevanten Performanz und Ressourcen Anforderungen getroffen wurde.

In Kapitel drei des Testansatzes wurde weiter auf die Erzeugung und Modifikation des Testdatensatzes sowie die Beschreibung jedes ML-Testszenarios inklusive der Verteilung seiner Charakteristiken und der Verteilung und Häufigkeit jedes ML-Testszenarios innerhalb des Testdatensatzes eingegangen. Dazu wurde beschrieben, dass zwei getrennte Datensätze für das Testen zu erzeugen sind, welche keine identischen Bilder beinhalten dürfen. Für den Absicherungsdatensatz wurde dabei vorgegeben, dass systematisch aus der Datenbasis der Testdaten des GTSRB Datensatzes[15] zehn Bilder je Testszenario bzw. Klasse des Verkehrszeichens auszuwählen sind, welche als Charakteristik eine Störung durch natürliche Umweltbedingungen aufweisen. Beispielsweise eine außergewöhnlich hohe Belichtung oder teilweise Verdeckung. Der Testdatensatz dagegen ist gemäß dem Testansatz aus je 100 zufällig ausgewählten Bildern je Klasse zu bilden. Für beide Datensätze sind dabei eindeutige Identifikatoren zu vergeben, sodass die Tests mit denselben Daten reproduziert werden können. Zur Erfüllung der ASPICE Guideline [VDA23a] fordert der erstellte Testansatz hinsichtlich der Modifikation der Daten für den Test weiter, dass beide Datensätze für den Test mindestens teilweise durch neue Daten abzuändern sind, insofern nach dem Testen weitere Anpassungen des getesteten ML-Modells vorgenommen werden. Dadurch soll die Aussagefähigkeit der Testdaten für die Erfüllung der ML-Anforderungen gewahrt werden.

Hinsichtlich der Auswertung der Testergebnisse wurde in Kapitel vier des Testansatzes definiert, dass gemäß der Architekturspezifikation ein VBA Skript innerhalb einer .xlsm Datei zu entwickeln ist, welches automatisch die Vorhersagen des YOLOv5-Klassifikationsmodells als Textdokumente einliest und mit den erwarteten Ergebnissen je Testdatum abgleicht. Diese sind dazu zunächst aus der Datei GroundTruth-Test.csv des GTSRB Datensatzes zu entnehmen. Anschließend wurde beschrieben, dass weiter ein Testbericht zu erstellen ist, welcher eine Aussage über die Erfüllung der spezifizierten Performanz und

[15] `https://benchmark.ini.rub.de/gtsrb_dataset.html`, zuletzt abgerufen am 30.01.2024

Ressourcen Anforderungen enthält und die falsch klassifizierten Daten aufführt. Abschließend wurde in Kapitel fünf des Testansatzes spezifiziert, dass als Testumgebung die aufgesetzte Trainingsumgebung weiter zu verwenden und um zwei Ordner für die beiden Datensätze für das Testen zu ergänzen ist.

Die BP.2 des MLE.4 sieht, wie auch beim MLE.3, die eigentliche Erzeugung des zu verwendenden Datensatzes gemäß dem beschriebenen Ansatz vor und wird durch die BP.3 ergänzt, welche das Ausführen der Tests des trainierten ML-Modells mit den ausgewählten Daten fordert. Im Falle der experimentellen Entwicklung wurde der Testansatz wie zuvor beschrieben angewandt und das trainierte ML-Modell mit einem Testdatensatz bestehend aus 1000 zufällig ausgewählten Bildern und einem Absicherungsdatensatz mit 100 anderen systematisch ausgewählten Bildern getestet. Die Daten des Absicherungsdatensatzes wurden dafür manuell aus den verfügbaren Testdaten des GTSRB Datensatzes selektiert und anschließend durch ein VBA Skript aus der verbliebenen Datenmenge zufällig je 100 Bilder pro Klasse als Testdaten ausgewählt. Dazu wurden, wie auch bei den Trainings- und Validierungsdaten, die Dateinamen aller verfügbaren Daten in eine Excel Tabelle eingefügt und durch „WorksheetFunction.RandBetween(Arg1, Arg2)“[16] jeder Datei eine Zufallszahl zwischen eins und der Anzahl verfügbarer Bilder zugewiesen. Nach dieser wurden die Dateien aufsteigend sortiert und anschließend die ersten 100 pro Klasse als Testdaten in den Testdatensatz kopiert. Während der Testausführung, welche durch den Aufruf der Methode „classify/predict.py“ mit den in der Architekturspezifikation, siehe Anhang B, festgelegten Parametern erfolgte, wurde eine Korrektheit der Vorhersagen von 99,3% mit dem Testdatensatz erreicht und für den Absicherungsdatensatz eine Korrektheit von 98%. Damit wurden für beide Datensätze die Performanz Anforderungen hinsichtlich der Korrektheit der Vorhersagen des ML-Modells erfüllt. Auch die weiteren Performanz Anforderungen, hinsichtlich der benötigten Rechenzeit des ML-Modells von im Durchschnitt maximal einer Sekunde pro Bild und der Ergebnisauswertung von maximal fünf Minuten für alle Testergebnisse, wurden erreicht. Die Verarbeitung eines Bildes auf dem Test- und dem Absicherungsdatensatz hat dabei im Durchschnitt circa 81 Millisekunden benötigt und die Auswertung aller Testergebnisse hat circa 30 Sekunden in Anspruch genommen. Der Speicherbedarf der ML-Komponente betrug insgesamt circa 58 Megabyte und erfüllt damit ebenfalls die spezifizierte Anforderung.

BP.4 und BP.5 des Prozesses MLE.4 legen weiter fest, dass aus dem trainierten und erfolgreich getesteten ML-Modell ein *Deployed ML Model* für die Integration mit der weiteren Software und dem Einsatz im realen System erzeugt und dieses ebenfalls getestet wird. Da im Rahmen der experimentellen Entwicklung das ML-Modell jedoch nicht in eine größere Software oder ein reales System zu integrieren ist, wurde aus dem trainierten ML-Modell kein Modell

[16] `https://learn.microsoft.com/de-de/office/vba/api/excel.worksheetfunction.randbetween`, zuletzt aufgerufen am 02.11.2023

für die Integration erzeugt und daher auch keine weiteren Tests durchgeführt.

Wie auch die anderen MLE-Prozesse fordert der MLE.4 innerhalb der vorletzten BP die Herstellung einer bidirektionalen Rückverfolgbarkeit, um in diesem Fall die Konsistenz zwischen den spezifizierten ML-Anforderungen, dem Testansatz und den Testergebnissen sowie zwischen den verwendeten Testdaten und den Anforderungen an die Daten sicherzustellen. Bezüglich der Rückverfolgbarkeit zwischen den Anforderungen an die Daten und den beiden verwendeten Datensätzen wurde, wie auch für den MLE.3, innerhalb der Anforderungsspezifikation, welche in Anhang A abgelegt ist, in dem Attribut „Test Data“ unter Referenzen angegeben, welche Anforderungen für welchen Datensatz relevant sind. Für Anforderungen, welche nicht relevant für die Testdaten sind, wurde ein Schrägstrich eingetragen. Hinsichtlich der Rückverfolgbarkeit zwischen den ML-Anforderungen und dem Testansatz wurde sowohl in dem Attribut „Test Approach“ der Anforderungsspezifikation bei relevanten Anforderungen ein Verweis auf das jeweilige Kapitel des Testansatzes eingefügt als auch im Testansatz die relevanten Anforderungen durch ihre Identifikatoren im Text referenziert. Weiter wurde im Testansatz auf den Testbericht verwiesen und im Testbericht sowohl das getestete Modell, die verwendeten Datensätze als auch die relevante Version des Testansatzes benannt. Der erzeugte Testbericht kann vollständig unter Anhang E eingesehen werden.

Die BP.7 fordert als letzte Basispraktik des Prozesses MLE.4, dass die Ergebnisse aus dem *Machine Learning Model Testing* zusammengefasst und zusammen mit dem entwickelten ML-Modell an alle betroffenen Parteien kommuniziert werden. Die geforderte Zusammenfassung der Testergebnisse erfolgte in der experimentellen Entwicklung durch den in Anhang E angehangenen Testbericht, welcher sowohl Auskunft über die Erfüllung der relevanten Performanz und Ressourcen Anforderungen gibt, als auch die falsch klassifizierten Testdaten auflistet.

Mit der Anwendung des vierten MLE-Prozesses wurde für das während der experimentellen Entwicklung implementierte ML-Modell zur Verkehrszeichenerkennung nachgewiesen, dass die spezifizierten ML-Anforderungen erfüllt wurden und die Entwicklung somit erfolgreich abgeschlossen.

4. Verifikation der ASPICE Referenzprozesse hinsichtlich der Unterstützung einer Sicherheitsargumentation

Autonom fahrende Fahrzeuge, deren Funktionalitäten unter anderem auf maschinellem Lernen basieren, sollen zukünftig den öffentlichen Straßenverkehr sicherer machen und versprechen eine Vielzahl von Vorteilen gegenüber heutigen Kraftfahrzeugen [MRR+15]. Während in heutigen Fahrzeugen mit einer Automatisierung bis zu dem SAE Level 2 [On-21] eine große Sicherheitslast auf dem Fahrer liegt, welcher zu jeder Zeit in der Verantwortung ist, das Fahrzeug sicher zu steuern und zum Beispiel angemessen auf ungewöhnliche Situationen zu reagieren, entfällt bei autonomen Fahrzeugen des SAE Level 5 diese Rückfallebene vollständig. Das Fahrzeug muss selbständig alle Aspekte der Fahraufgabe in jeder Situation sicher übernehmen, was von dem Fahrzeughersteller verantwortet wird. Häufig ist der Fahrer heutzutage selber für Unfälle verantwortlich, da er sich nicht an die Straßenverkehrsordnung gehalten hat, was durch die Nutzung autonom fahrender Fahrzeuge verhindert werden kann [Koo23]. Die Entwicklung autonomer Fahrzeuge stellt jedoch eine große Herausforderung in der Automobilindustrie dar, denn Funktionen zur Realisierung des autonomen Fahrens besitzen diverse Fehlerpotenziale und damit verbundene Sicherheitsbedenken, welche in der Entwicklung systematisch adressiert und auf ein akzeptables Restrisiko mitigiert werden müssen [Koo23, MRR+15, SM+22, SQC17, WSRA20]. In Abschnitt 4.1 werden als Ergebnis einer Literaturrecherche verschiedene Ansätze zur Sicherheitsargumentation von ML-Komponenten innerhalb eines komplexen autonomen Fahrzeuges, gemäß dem Stand der Forschung und Normung, vorgestellt und in Abschnitt 4.2 anschließend die im ASPICE 4.0 [VDA23b] neu eingeführten Referenzprozesse für die Entwicklung einer ML-Komponente sowie dem *Machine Learning Data Management* hinsichtlich der Unterstützung dieser Ansätze zur Sicherheitsargumentation eingeordnet. Dabei liegt der Fokus darauf herauszustellen, wo die in den BPs geforderten Aktivitäten konkret in die Sicherheitsargumentation einzahlen, wo die BPs unabhängig von der Sicherheitsargumentation sind und wo das Prozessreferenzmodell des ASPICE möglicherweise sogar konträr zu den Forderungen der Sicherheitsargumentation ist.

4.1. Sicherheitsargumentation für den Einsatz von ML

Maschinelles Lernen wird als eine der Kerntechnologien angesehen, um autonome Fahrfunktionen zu realisieren, unterscheidet sich, wie in Abschnitt 2.1 beschrieben, jedoch stark von klassischer Softwareentwicklung und führt daher zu verschiedenen speziellen Sicherheitsbedenken hinsichtlich des Einsatzes in sicherheitskritischen Systemen [SKS+20]. Wie in Abschnitt 2.3 eingeführt, sind diese Sicherheitsbedenken durch angemessene Methoden zu adressieren, um innerhalb einer Sicherheitsargumentation zu belegen, dass keine unverhältnis-

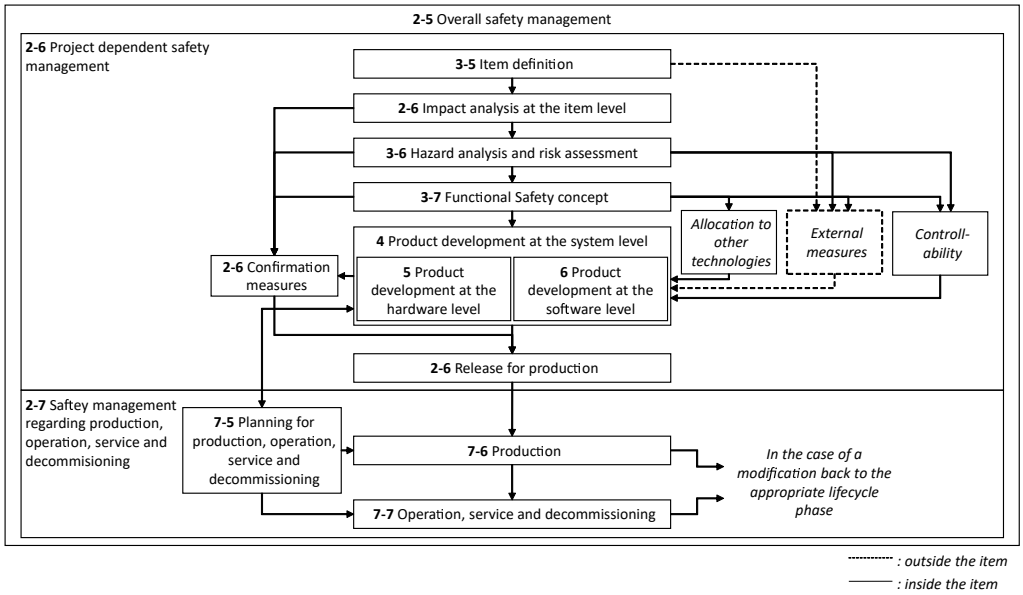


Abbildung 24: Management Aktivitäten in Bezug zu dem Sicherheitslebenszyklus der ISO26262, basierend auf [ISO18]

mäßigen Risiken aufgrund von Gefährdungen durch die ML-Komponente sowie des Gesamtsystems existieren. Die im Folgenden vorgestellten Ansätze dieser Sicherheitsargumentation bilden das Ergebnis einer umfassenden Literaturrecherche und reichen von bestehenden Standards in der Automobilindustrie über Ergebnisse aus der Forschung bis hin zu Strategien von Technologieanbietern.

Hinsichtlich klassischer System- und Softwareentwicklung existieren in der Automobilindustrie mit der ISO 26262 [ISO18] und der ISO 21448 [ISO22] Standards, welche die Sicherheit von Straßenfahrzeugen adressieren. Die ISO 26262 beschreibt dabei innerhalb von zwölf Teilen, wie die funktionale Sicherheit von elektrischen und/oder elektronischen Systemen (E/E Systemen) zu erreichen ist, welche sie als die Abwesenheit eines unverhältnismäßigen Risikos aufgrund von Gefährdungen durch das Fehlverhalten von E/E Systemen definiert. Dazu legt die ISO 26262 einen Sicherheitslebenszyklus fest, welcher in Abb. 24 abgebildet ist, und führt mit dem *Automotive Safety Integrity Level (ASIL)* ein Schema zur Risikoklassifikation ein, in Abhängigkeit dessen der Standard verschiedene Anforderung an die Entwicklung des Produktes stellt. Die ISO 21448 beschreibt weiter die Sicherheit der beabsichtigten Funktion, welche sie als Abwesenheit von unverhältnismäßigen Risiken aufgrund von Gefährdungen, welche durch funktionale Unzulänglichkeiten entstehen, definiert. Funktionale Unzulänglichkeiten können dabei gemäß dem Standard aus Unzulänglichkeiten der Spezifikation oder der Performanz resultieren und beim Eintreten sogenannter *Triggering Conditions* zu einer Gefährdung führen. Beide Standards adressieren ML jedoch

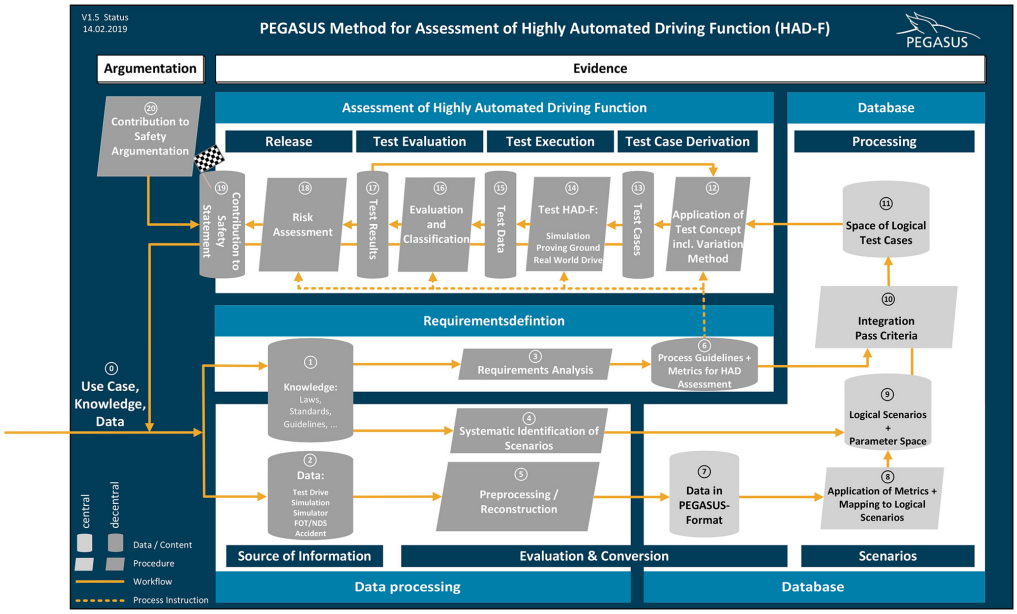


Abbildung 25: *PEGASUS Method for Assessment of Highly Automated Driving Function*, Bildquelle [Pro19a]

nicht direkt und sind daher nicht vollständig anwendbar. Über ihre Erfüllung hinaus werden somit weitere Nachweise benötigt, um die Sicherheit für den Einsatz von ML-Komponenten in sicherheitskritischen Systemen zu argumentieren [BBB+20, RVM18, SKS+20, SQC17, WCACP20]. Andere Standards wie die UL 4600 [UL 23] adressieren diese Lücke, indem sie konkret auf automatisiertes Fahren und die Verwendung von ML eingehen, bleiben dabei jedoch auf einer hohen Abstraktionsebene und beschreiben generelle Ziele anstelle von konkreten Anforderungen an die Entwicklung und dem zu erreichenden Restrisiko [BBB+20, Cru22b, Koo23]. Weitere Standards, welche konkreter auf den Einsatz von sicherheitskritischen ML-Komponenten in Automobilen eingehen, sind aktuell noch in der Erstellung, so soll zum Beispiel die ISO PAS 8800 die Sicherheit von KI in Straßenfahrzeugen adressieren und entsprechende Sicherheitsprinzipien, Methoden und Nachweise, im Einklang mit der ISO 26262 und ISO 21448, definieren [ISOng].

Mit der in Abb. 25 dargestellten *PEGASUS Method for Assessment of Highly Automated Driving Function* wurde innerhalb des PEGASUS Projektes[17] ein Konzept aus fünf Phasen entwickelt, um die Sicherheit von hochautomatisierten Fahrfunktionen durch Tests zu verbessern. Als Ausgangspunkt dient dabei eine Beschreibung des Anwendungsfalles des Testobjektes, vorhandenes Wissen aus zum Beispiel vorangegangenen Zyklen der PEGASUS Methode und vorhandene Daten für die Tests. Auf Basis dieser Informationen sind zunächst Daten mit dem Ziel vorzuverarbei-

[17] `https://www.pegasusprojekt.de`, zuletzt aufgerufen am 23.10.2023

ten, logische Testszenarien bezüglich des Testobjektes zu identifizieren und Informationen aus bereits ausgeführten Szenarien in einem bestimmten Format zur Verfügung zu stellen. Parallel dazu sind Anforderungen bezüglich des Verhaltens des Testobjektes zu stellen, welche als Evaluationskriterien der Szenarien dienen. Die Evaluationskriterien sind im dritten Schritt mit den formatierten Testszenarien zu kombinieren und bilden die logischen Testfälle der Datenbasis, welche im vierten Schritt in Simulationen, auf Testgeländen oder in einem Testfeld auszuführen sind. Die erzeugten Testergebnisse sind anschließend gegenüber den Evaluationskriterien zu prüfen und durch eine Risikobewertung eine Aussage zur Sicherheit des Testobjektes zu erzeugen. Diese Aussage bezüglich der Sicherheit ist abschließend als Evidenz mit der Sicherheitsargumentation abzugleichen, um sicherzustellen, dass die Evidenzen die Sicherheitsargumentation unterstützen. Die im PEGASUS Projekt erstellte Sicherheitsargumentation ist dabei als ein konzeptioneller Rahmen für eine Sicherheitsargumentation hochautomatisierter Fahrfunktionen zu verstehen und besteht aus folgenden fünf aufeinander aufbauenden Ebenen, welche im PEGASUS Projekt nur teilweise exemplarisch adressiert wurden [Pro19a, Pro19b].

- Ebene 0 - Akzeptanzmodell
  Einbettung einer Sicherheitsargumentation in einem breiteren Kontext zur Darstellung der Technologieakzeptanz.
- Ebene 1 - Übergeordnete Sicherheitsziele
  Ableiten von Sicherheitszielen aus vereinbarten Designprinzipien, um Erwartungen wie zum Beispiel eine positive Risikobilanz zu erfüllen.
- Ebene 2 - Logische Struktur
  Aufbauen einer logischen Beziehung zwischen den zu erfüllenden Sicherheitszielen und den, die Sicherheitsargumentation unterstützenden, Evidenzen.
- Ebene 3 - Aktionen, Methoden und Werkzeuge
  Entwickeln und implementieren von Aktionen, Methoden und Werkzeugen, um die Sicherheitsziele zu erfüllen.
- Ebene 4 - Evidenzen
  Einbeziehen der aus der Anwendung der Aktivitäten, Methoden und Werkzeuge erbrachten Ergebnisse.

Innerhalb des Forschungsprojektes KI-Absicherung[18] wurde weiter ein Ansatz einer Sicherheitsargumentation speziell für die Absicherung von auf KI basierenden Funktionen im Fahrzeug erzeugt, welcher in Abb. 26 abgebildet ist. Dieser Ansatz basiert auf der Einbeziehung der Spezifikation der KI-Funktion in eine initiale Sicherheitsanalyse, welche in Anlehnung an die ISO 26262 und die ISO 21448 durchzuführen ist. Aus den identifizierten Gefahren und Risiken sind anschließend, wie auch in Ebene eins der PEGASUS Sicherheitsargumentation beschrieben, Sicherheitsziele aufzustellen und aus diesen Sicherheitsanforderungen

[18] `https://www.ki-absicherung-projekt.de`, zuletzt aufgerufen am 23.10.2023

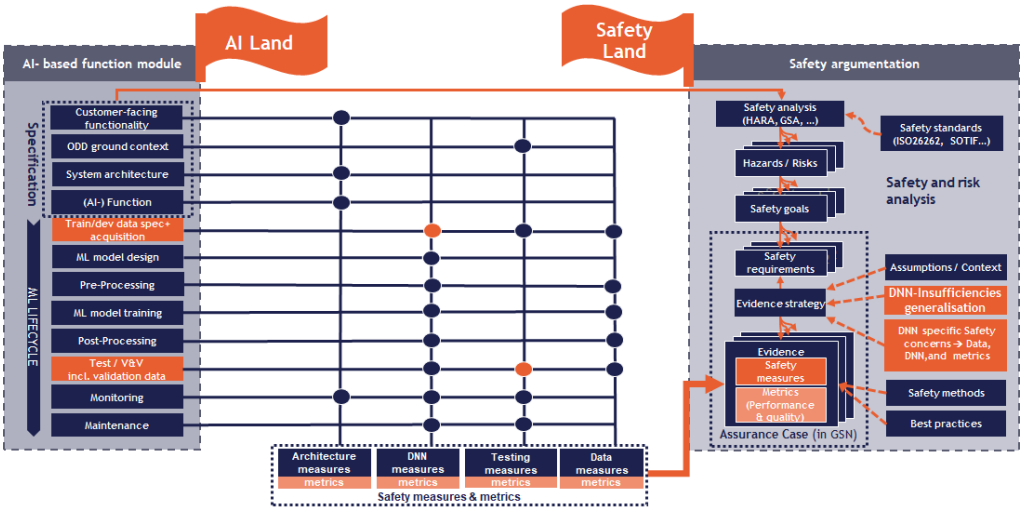


Abbildung 26: Gesamtansatz aus KI-Absicherung zur Absicherung von KI-Funktionen im Fahrzeug, Bildquelle [SM+22]

abzuleiten. Diese Sicherheitsanforderungen bilden die oberste Ebene innerhalb des sogenannten *Assurance Case*, welcher zum Beispiel in der GSN dargestellt werden kann. Der *Assurance Case* beinhaltet dabei eine Strategie, gemäß welcher die Sicherheitsanforderungen mit den Evidenzen aus der Anwendung von Sicherheitsmethoden und Maßnahmen unterstützt werden und deckt somit die Ebenen zwei bis vier der PEGASUS Sicherheitsargumentation ab. Im KI-Absicherungsprojekt wird bezüglich der Evidenzstrategie speziell auf spezifische Sicherheitsbedenken von tiefen neuronalen Netzen, welche zum Teil bereits in Abschnitt 2.1.3 erklärt wurden und die Generalisierungsfähigkeit als Unzulänglichkeit von tiefen neuronalen Netzen eingegangen [MSB+21, SM+22]. Bezüglich der unzureichenden Generalisierungsfähigkeit kann dabei zwischen einfachen Generalisierungsproblemen, logischen und Stabilitätsproblemen unterschieden werden. Einfache Generalisierungsprobleme meinen eine unzureichende Vorhersagegenauigkeit bei Daten, welche den verwendeten Trainingsdaten semantisch ähnlich sind. Logische Probleme beziehen sich auf die vom Modell, aus den Trainingsdaten, gelernten Zusammenhänge bzw. Transformationen der Daten, und Stabilitätsprobleme bezeichnen die fehlende Robustheit des Modells gegenüber Störungen in den Daten. Aus den drei Kategorien an Unzulänglichkeiten wurden in [SKS+20], wie in Abb. 27 in der GSN dargestellt, beispielhafte Sicherheitsanforderungen als Ziele und entsprechende Strategien zur Erreichung der Ziele auf einer hohen Abstraktionsebene, gemäß dem Ansatz aus KI-Absicherung, ausgearbeitet.

In [WCACP20] wurden die zuvor beschrieben grundlegenden Ansätze der Sicherheitsargumentation durch ein, auf der GSN basierendes, Muster für die Argumentation der Sicherheit von ML-Komponenten weiter konkretisiert. Auf der höchsten

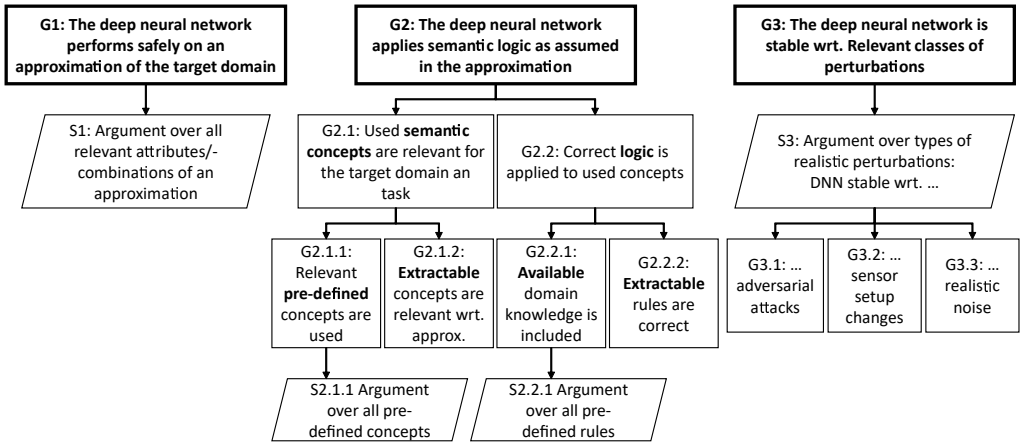


Abbildung 27: Sicherheitsanforderungen und Argumentationsstrategien, in GSN, basierend auf [SKS+20]

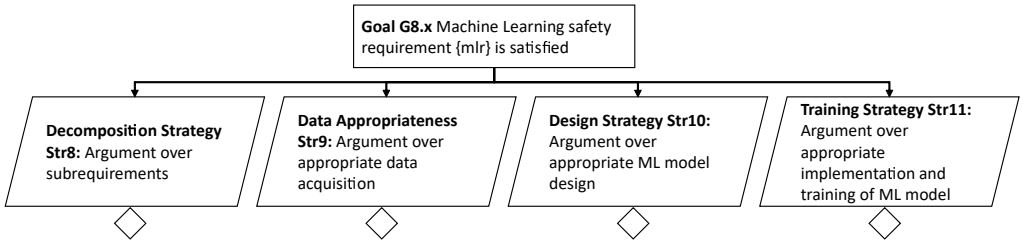


Abbildung 28: Oberste Ebene des Musters für die Argumentation der Sicherheit von ML-Komponenten, in GSN, basierend auf [WCACP20]

Ebene argumentiert das Muster, wie in Abb. 28 abgebildet, durch vier Strategien die Erfüllung einer bestimmten Sicherheitsanforderung. Die drei Strategien zur Angemessenheit der Daten, des Designs und des Trainings werden weiter innerhalb des Musters durch in Summe 39 Ziele detailliert, welche in der Tabelle in Anhang F aufgelistet sind. Die detaillierten Ziele gehen dabei zum Beispiel auf die Qualität der Label, das Herunterbrechen der Sicherheitsanforderung in dem Design und die Robustheit des ML-Modells ein. Außerdem wurde in [WCACP20] zur Evaluation des eingeführten Musters dieses am Beispiel eines sogenannten *Pedestrian Avoidance Systems*, welches ML zur Fußgängererkennung einsetzt, befüllt und gezeigt, wie das Muster zur Sicherheitsargumentation praktisch verwendet werden kann.

Durch [RVM18] wurden weiter Ansätze der Sicherheitsargumentation aus verschiedenen Domänen auf die Automobilindustrie adaptiert und das in Abb. 29 abgebildete Konzept eingeführt, um die Sicherheit eines neuronalen Netzes in einem sicherheitskritischen Kontext nachzuweisen. Die Argumentation beschreibt als oberstes Ziel, dass das neuronale Netz akzeptabel sicher ist, um in einem definierten System mit einer festgelegten Rolle zu operieren und bricht dieses

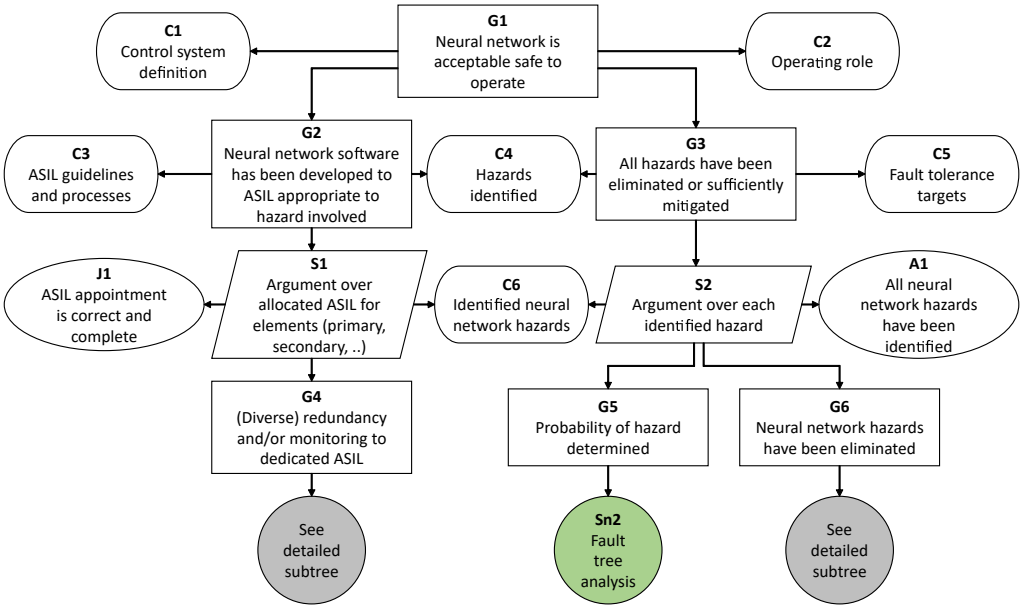


Abbildung 29: Sicherheitsargumentation neuronaler Netze, in GSN, basierend auf [RVM18]

Ziel in Anlehnung zu der ISO26262 über die identifizierten Gefahren des Systems und die Anforderungen hinsichtlich des ASIL herunter. Auf der tiefsten Ebene der *Goal Structure* schlägt [RVM18] Methoden vor, um die heruntergebrochenen Ziele zu lösen, so kann zum Beispiel durch eine Fehlerbaumanalyse das Ziel adressiert werden, die Wahrscheinlichkeit einer Gefahr zu bestimmen. Weiter werden zum Beispiel Plausibilitätsprüfungen und eine Analyse des Programmflusses vorgeschlagen, um das Ziel einer Überwachung des neuronalen Netzes zu erreichen, und Fehlerbehebung sowie Fehlertoleranz als Lösungen für das Ziel einer Redundanz und Fehlbehandlung. Die Liste an vorgeschlagenen Methoden wird von den Autoren jedoch nicht als vollständig und ausreichend angesehen, um die Verwendbarkeit eines neuronalen Netzes zu argumentieren.

Die vorgestellten Ansätze aus der Standardisierung und Forschung unterscheiden sich stark darin, wie genau sie spezielle Aspekte von ML innerhalb der Sicherheitsargumentation berücksichtigen. In der generellen Struktur ähneln sich die Ansätze jedoch sehr, so ist generell zunächst zu beschreiben, welche Aufgaben im System durch die ML-Komponente gelöst werden sollen und diese entsprechend zu spezifizieren. Anschließend ist auf dieser Grundlage zu analysieren, welche Gefahren durch die Komponente entstehen und daraus Anforderungen an die Entwicklung abzuleiten, welche durch Methoden und Maßnahmen zu adressieren und mit entsprechenden Evidenzen innerhalb der Sicherheitsargumentation nachzuweisen sind. Die vorgestellten Ansätze sollten aus diesem Grund

nicht losgelöst voneinander betrachtet werden, sondern bilden, in dem sie sich in ihrer Detaillierung ergänzen, gemeinsam einen Ansatz der Sicherheitsargumentation für den Einsatz von ML-Komponenten innerhalb eines komplexen autonomen Fahrzeuges, gemäß dem Stand der Forschung und Standardisierung.

Über den Stand der Forschung und Standardisierung hinaus haben verschiedene Technologieanbieter, welche bereits erste hoch automatisiert fahrende Flotten betreiben, Strategien entwickelt, um die erreichte Sicherheit ihrer Fahrzeuge zu argumentieren. Unternehmen wie Cruise[19], Mobileye[20] und Waymo [21] argumentieren Sicherheit durch das Design des Systems zu erreichen, welches sie zunächst in einer eingeschränkten ODD pilotieren, wodurch bestehende Risiken und Sicherheitsbedenken reduziert werden. Dabei verwenden sie zum Beispiel menschliche Sicherheitsfahrer, die im Falle eines Fehlverhaltens des Systems oder beim Verlassen der ODD angemessen reagieren können. Weiter setzten alle drei Unternehmen auf Redundanzen im System, um die Robustheit ihrer Systeme gegenüber zum Beispiel neuen Szenarien zu erhöhen und Fehler einer Komponente zu kompensieren. Die durchgeführten Tests dienen dabei als Nachweise der Sicherheit des Systems und als Argumentation die Funktionalität und ODD zu erweitern [Cru22b, Mob23, WSL+20].

4.2. Einordnung der Referenzprozesse in die Sicherheitsargumentationen

Wie durch die in Abschnitt 3 beschriebene experimentelle Entwicklung gezeigt, sind die standardisierten ML spezifischen Referenzprozesse des ASPICE 4.0 [VDA23b] für die Entwicklung einer ML-Komponente anwendbar. Folgend wird daher weiter diskutiert, inwiefern die Anwendung der standardisierten Referenzprozesse die verschiedenen in Abschnitt 4.1 vorgestellten Ansätze zur Sicherheitsargumentation für den Einsatz von ML in einem sicherheitskritischen System unterstützt. Dazu wird zunächst aufgezeigt, auf welche Aspekte der Ansätze die Anwendung der standardisierten Referenzprozesse einzahlt, um anschließend hervorzuheben, welche Aspekte der Sicherheitsargumentation durch die BPs des ASPICE 4.0 nicht adressiert oder sogar konterkariert werden.

Die ISO 26262 [ISO18] betont, dass das Erreichen der funktionalen Sicherheit grundsätzlich durch die angewandten Produktions-, Service-, Management- und Entwicklungsprozesse beeinflusst wird und fasst dabei unter Entwicklungsprozessen Aktivitäten wie Anforderungsspezifikation, Design, Implementierung, Integration, Verifizierung, Validierung und Konfiguration zusammen, die durch

[19] `https://getcruise.com/`, zuletzt aufgerufen am 17.01.2024
[20] `https://www.mobileye.com/`, zuletzt aufgerufen am 17.01.2024
[21] `https://waymo.com/`, zuletzt aufgerufen am 17.01.2024

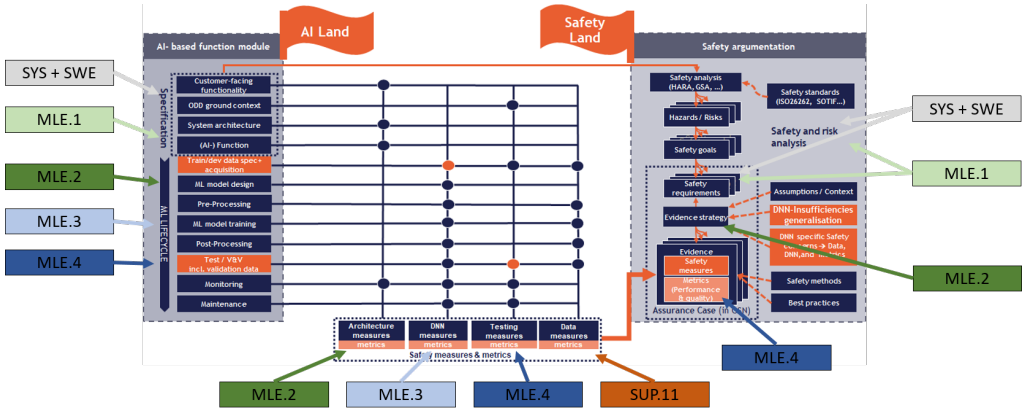


Abbildung 30: Einordnung der ASPICE Prozesse zur Unterstützung des Gesamtansatzes aus KI-Absicherung, basierend auf [SM+22]

die verschiedenen Prozesse des ASPICE [VDA23b], welche in Abschnitt 2.2 eingeführt wurden, adressiert werden. Der im PEGASUS Projekt[22] erstellte konzeptionelle Rahmen für eine Sicherheitsargumentation sowie der konkretere Gesamtansatz aus dem Forschungsprojekt KI-Absicherung[23] betonen ebenfalls die Abhängigkeit zwischen den Entwicklungsaktivitäten und der eigentlichen Sicherheitsargumentation [Pro19b, SM+22]. Dabei sieht der Gesamtansatz zunächst die Spezifikation der Kundenfunktion, ODD, Systemarchitektur und KI-Funktion vor, was durch die Anwendung der System- und Softwareentwicklungsprozesse sowie des Prozesses MLE.1 abgedeckt wird. Weiter beschreibt der Gesamtansatz einen ML-Lebenszyklus, welcher Aktivitäten vom Design, Training und Testen des ML-Modells bis zur Überwachung und Wartung umfasst und zumindest bis zum Testen von der Umsetzung der Prozesse MLE.2 bis MLE.4 unterstützt wird. Auf der Seite der Sicherheitsargumentation sieht der Ansatz aus KI-Absicherung zunächst eine Sicherheits- und Risikoanalyse vor, um Sicherheitsziele abzuleiten, welche in Sicherheitsanforderungen heruntergebrochen werden können. Die Sicherheits- und Risikoanalyse kann dabei als ein Aspekt der Anforderungsanalysen auf System-, Software- und ML-Komponentenebene angesehen werden, welche unter anderem durch die BP3 und BP4 des MLE.1 gefordert wird. Weiter formuliert der Gesamtansatz vier Arten von Maßnahmen und Metriken, welche als Evidenzen in die Sicherheitsargumentation einzahlen und durch die Anwendung der Prozesse MLE.2 bis MLE.4 sowie den SUP.11 erzeugt werden können [MSB+21, SM+22]. In Abb. 30 wurden diese Abhängigkeiten zwischen den ASPICE Prozessen und dem in Abb. 26 bereits visualisiertem Gesamtansatz aus KI-Absicherung durch die Ergänzung entsprechender Verknüpfungen visualisiert.

[22] `https://www.pegasusprojekt.de`, zuletzt aufgerufen am 23.10.2023
[23] `https://www.ki-absicherung-projekt.de`, zuletzt aufgerufen am 23.10.2023

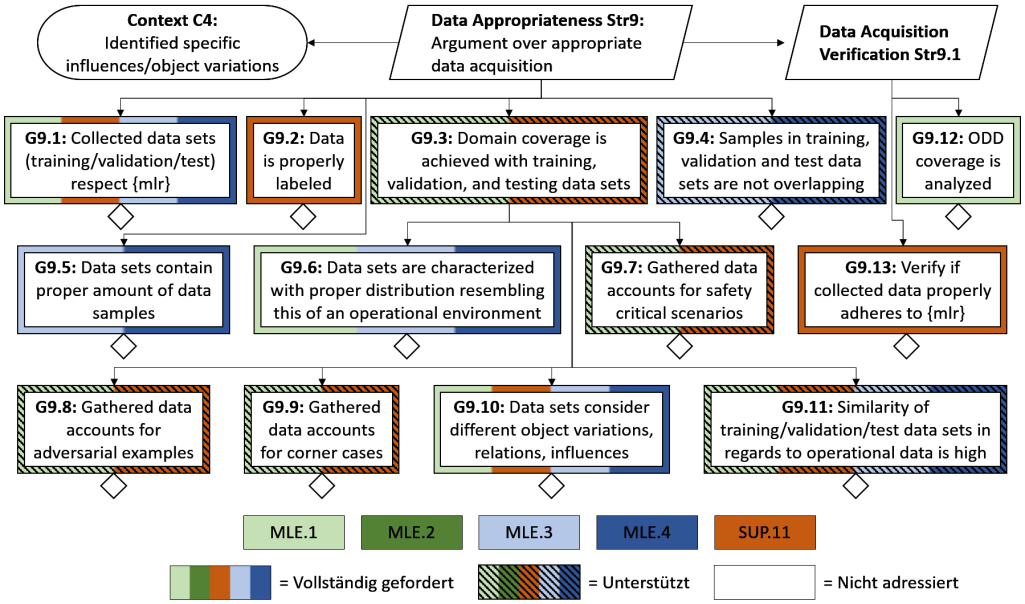


Abbildung 31: Einordnung der ASPICE Prozesse zur Unterstützung des Musters zur Argumentation angemessener Daten, basierend auf [WCACP20]

Genauer auf die Evidenzstrategie, als Argumentationskette der Erfüllung der einzelnen ML-Sicherheitsanforderungen, wird in dem Argumentationsmuster von [WCACP20] eingegangen. Die in Abb. 28 dargestellten vier Hauptziele lassen sich dabei genau auf die fünf ML spezifischen Referenzprozesse des ASPICE 4.0 abbilden. So adressiert der Prozess MLE.1 sowohl das Herunterbrechen der Anforderungen, welches in dem Argumentationsmuster nicht weiter ausgeführt wurde, als auch die Angemessenheit der Daten, auf welche auch der Prozess SUP.11 einzahlt. Der Prozess MLE.2 thematisiert die Angemessenheit des Designs des ML-Modells und die Prozesse MLE.3 und MLE.4 die Angemessenheit der Implementierung sowie des Trainings des ML-Modells. In der Tabelle in Anhang F wurden weiter die 39 detaillierten Ziele aus dem Argumentationsmuster auf die einzelnen BPs der ML spezifischen Referenzprozesse des ASPICE abgebildet, worauf folgend genau eingegangen wird.

Wie in Abb. 31 visualisiert, können alle 13 Ziele, welche in [WCACP20] in dem Muster für die Argumentation der Angemessenheit der Daten aufgeführt wurden, auf die ML spezifischen Prozesse des ASPICE abgebildet werden. Dabei werden die sieben durch einen ausgefüllten Rahmen hervorgehobenen Ziele vollständig von den Referenzprozessen gefordert, wogegen die sechs Ziele mit einem gestreiften Rahmen durch die Referenzprozesse nur teilweise adressiert werden. So wird zum Beispiel über die Prozesse MLE.1, MLE.3, MLE.4 und den SUP.11 das Ziel G9.1, welches aussagt, dass die gesammelten Datensätze die Sicherheitsanforderungen

an die ML-Komponente berücksichtigen, vollständig gefordert. Das MLE.1 sieht mit der BP.1, 3, 4 und 5 vor, dass Anforderungen an die zu verwendenden Daten spezifiziert werden, welche sowohl in sich als auch zu den Anforderungen der höheren Softwareebene und somit auch mit den spezifizierten Sicherheitsanforderungen konsistent sein müssen. Weiter fordert der Prozess SUP.11, dass eine Datenbasis erzeugt wird, welche mit den an sie gestellten Anforderungen und Qualitätskriterien konsistent ist und somit auch die Sicherheitsanforderungen berücksichtigt. Diese Datenbasis soll gemäß der Prozesse MLE.3 und MLE.4 weiter als Grundlage für die, während des Trainings und Testens verwendeten Datensätze, dienen, womit das Ziel G9.1 vollständig im ASPICE abgebildet ist. Das Ziel G9.3, welches aussagt, dass durch den Trainings-, Validierungs- und Testdatensatz die Abdeckung der Domäne erreicht wurde, und das Ziel G9.11, welches besagt, dass die Ähnlichkeit zwischen dem Trainings-, Validierungs- und Testdatensatz zu den realen operativen Daten hoch ist, werden durch das PRM nur teilweise adressiert. So werden beide Ziele zwar durch die Spezifikation und Analyse der Anforderungen an die Daten sowie dem zu den Anforderungen konsistenten Datenmanagement als Grundlage für die verwendeten Datensätze unterstützt, jedoch ist dies je nach Projektkontext, unter anderem aufgrund der ständigen Veränderung der Realität, nicht ausreichend, um die tatsächliche Abdeckung der Domäne sowie die Ähnlichkeit zwischen den verwendeten Daten und der Realität sicherzustellen. Wie in dem Gesamtansatz aus KI-Absicherung und in [WSRA20] beschrieben, ist daher über die Umsetzung der BPs hinaus möglicherweise eine konstante Überwachung der Domäne erforderlich, um die beiden Ziele G9.3 und G9.11 zu erreichen. Da das ASPICE nicht vorgibt, welche Anforderungen an die verwendeten Daten zu stellen sind, wird die Erreichung der Ziele G9.7, G9.8 und G9.9 durch das PRM ebenfalls nur teilweise adressiert. Insofern in dem Projekt als Anforderung an die Daten dokumentiert wurde, dass sicherheitskritische Szenarien, *Adversarial Examples* und Ausnahmefälle in der Datenbasis enthalten sein müssen, so würden die Prozesse MLE.1 und SUP.11 die Konsistenz zwischen den Daten und diesen Anforderungen einfordern und somit auch die Erfüllung der drei Ziele. Genauso gibt das ASPICE auch nicht vor, dass die Trainings-, Validierungs- und Testdaten nicht überlappend sein dürfen, weswegen das Ziel G9.4 ebenfalls nur unterstützt wird. Stattdessen fordert das PRM über die BP.1 der Prozesse MLE.3 und MLE.4, dass ein Ansatz für das Training und Testen spezifiziert wird, in welchem dieser Punkt, je nach Projektkontext, aufgenommen werden könnte oder nicht.

Von den elf in [WCACP20] aufgestellten Zielen zur Argumentation eines angemessenen Designs des ML-Modells werden, wie in Abb. 32 dargestellt, sechs Ziele vollständig durch die ML spezifischen Referenzprozesse des ASPICE 4.0 gefordert, drei Ziele werden teilweise adressiert und zwei Ziele werden gar nicht adressiert, in der Abbildung durch einen weißen Rahmen um das Ziel visualisiert. Als vollständig gefordert kann dabei zum Beispiel das Ziel G10.1 angesehen werden, welches besagt, dass durch das Design des ML-Modells die Sicherheitsanforderun-

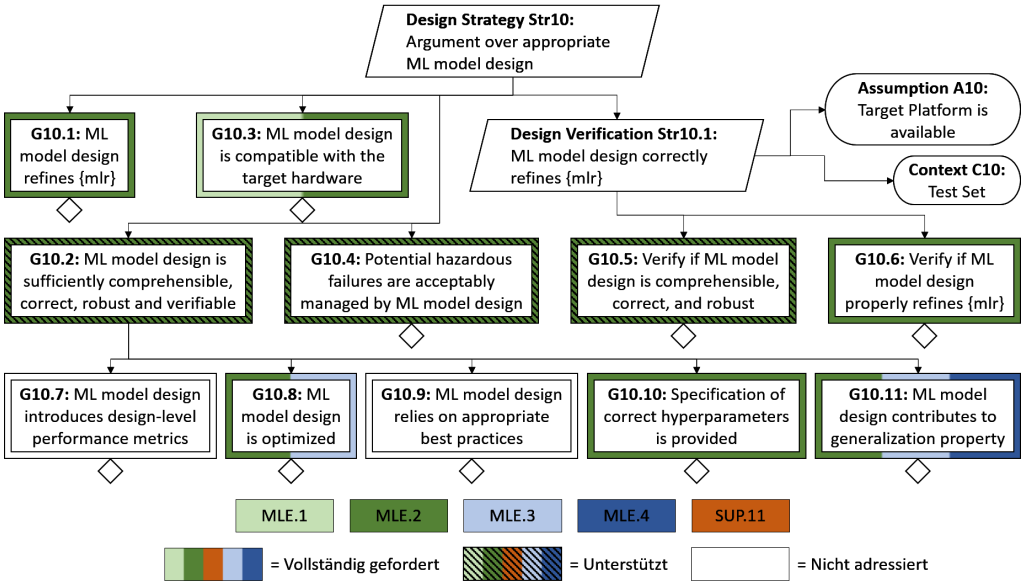


Abbildung 32: Einordnung der ASPICE Prozesse zur Unterstützung des Musters zur Argumentation eines angemessenen Designs des ML-Modells, basierend auf [WCACP20]

gen verfeinert werden, da die Konsistenz zwischen den Anforderungen und der Architektur ein wesentlicher Bestandteil des Prozesszwecks von MLE.2 ist und, wie in Anhang F dokumentiert, durch BP.1 und BP.6 eingefordert wird. Durch das PRM teilweise adressiert werden einerseits die Ziele G10.2 und G10.5, welche fordern, dass das ML-Modell ausreichend verständlich, korrekt, robust und verifizierbar designt wurde und dies auch verifiziert wurde. Diese vier Aspekte stellen mögliche Evaluationskriterien für die durch MLE.2 geforderte Evaluation der Architekturelemente dar, werden jedoch im ASPICE nicht vorgegeben. Weiter wird das Ziel G10.4, welches aussagt, dass potentiell gefährliche Fehler durch das Design des ML-Modells akzeptabel behandelt werden, zwar generell durch die Erstellung und Evaluierung einer ML-Architektur unterstützt, jedoch beinhaltet keine BP des ASPICE die Forderung, dass Mechanismen in der Architektur der ML-Komponente vorhanden sein müssen, welche den Betrieb beim Auftreten eines Fehlers sicherstellen. Das Ziel G10.7, welches vorsieht, dass in dem Design des ML-Modells Performanzmetriken auf Architekturebene eingeführt werden, und das Ziel G10.9, welches aussagt, dass das Design des ML-Modells auf angemessenen empfohlenen Vorgehensweisen basiert, werden in dem ASPICE durch keine BP gefordert.

Bezüglich der Angemessenheit der Implementierung und des Trainings des ML-Modells werden, wie in Abb. 33 dargestellt, sieben Ziele des Argumentationsmusters aus [WCACP20] vollständig durch die ML spezifischen Referenzprozesse des ASPICE adressiert, sechs Ziele werden teilweise unterstützt und zwei Ziele

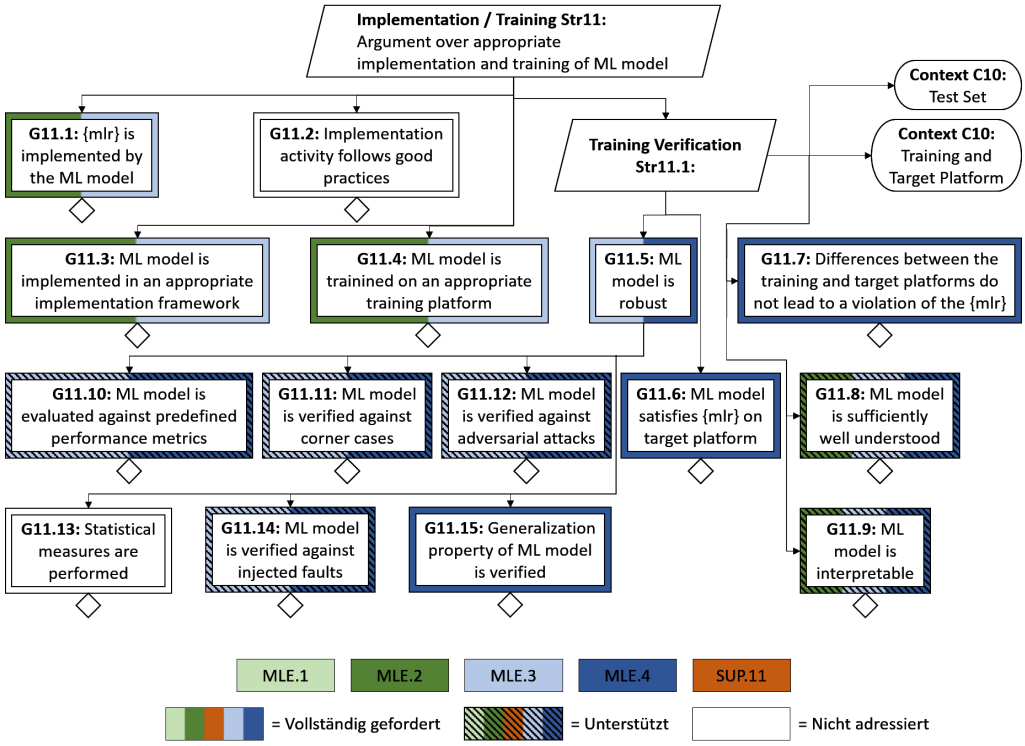


Abbildung 33: Einordnung der ASPICE Prozesse zur Unterstützung des Musters zur Argumentation einer angemessenen Implementierung und eines angemessenen Trainings des ML-Modells, basierend auf [WCACP20]

werden gar nicht adressiert. Durch das PRM vollständig gefordert wird dabei zum Beispiel das Ziel G11.7, welches beschreibt, dass Unterschiede zwischen der Trainings- und Zielplattform zu keinen Verletzungen der Sicherheitsanforderungen führen, denn der Prozess MLE.4 sieht innerhalb von BP.4 und BP.5 vor, dass ein Modell für den Einsatz auf der Zielplattform erzeugt und ebenfalls getestet wird. Teilweise adressiert werden zum einen die Ziele G11.10, G11.11, G11.12 und G11.14, welche besagen, dass das ML-Modell gegenüber definierten Performanzmetriken, Ausnahmefällen, *Adversarial Examples* und Fehlerinjektionen evaluiert bzw. verifiziert wurde. Das ASPICE sieht zwar eine Optimierung und Tests des ML-Modells innerhalb der Prozesse MLE.3 und MLE.4 vor, fordert aber nicht konkret, dass Performanzmetriken zu erheben sind und dass Ausnahmefälle, *Adversarial Examples* und bewusste Fehler in den verwendeten Daten enthalten sein müssen. Weiter werden die Ziele G11.8 und G11.9, welche festlegen, dass das ML-Modell ausreichend gut verstanden wurde und interpretierbar ist, durch die Anwendung der BP.3 des MLE.2 hinsichtlich der Evaluierung der ML-Architektur und durch die Umsetzung der Prozesse MLE.3 und MLE.4 unterstützt, wobei im ASPICE nicht konkret festgelegt wird, dass zu bewerten ist,

wie gut das Modell verständlich und interpretierbar ist. Gar nicht in den ML spezifischen Referenzprozessen adressiert wird das Ziel G11.2, hinsichtlich der Anwendung von guten Praktiken bei der Implementierung des ML-Modells, und das Ziel G11.13, welches aussagt, dass statistische Messungen durchgeführt wurden.

Anders als in dem durch [WCACP20] entwickelten Muster zur Argumentation der Sicherheit einer ML-Komponente werden in dem durch [RVM18] vorgeschlagenen Ansatz zur Sicherheitsargumentation konkrete Methoden auf der tiefsten Ebene der *Goal Structure* vorgeschlagen, um die Ziele in der Sicherheitsargumentation zu erfüllen. Die Ziele der Sicherheitsargumentation selbst sind dabei zwischen beiden Ansätzen vergleichbar und werden ebenfalls durch die Anwendung der ML spezifischen Referenzprozesse des ASPICE unterstützt. Die in [RVM18] vorgeschlagenen, konkreten Methoden als Nachweise der Ziele werden vom ASPICE jedoch nicht adressiert, da das ASPICE im PRM lediglich beschreibt was zu tun ist und nicht festlegt wie etwas getan werden soll.

Durch die Anwendung eines Entwicklungsprozesses, welcher zu den Referenzprozessen des ASPICE konform ist, wird, wie zuvor gezeigt, eine Vielzahl der Aspekte einer möglichen Sicherheitsargumentation adressiert. Der alleinige Nachweis der Umsetzung des PRM in den Entwicklungsprozessen ist jedoch keineswegs ausreichend für die Sicherheitsargumentation, welche in hohem Maße von den definierten Anforderungen an das System abhängt. Die Erreichung eines Prozessfähigkeitslevels von mindestens eins kann somit lediglich als einer von mehreren Nachweise zur Argumentation der Sicherheit einer auf ML basierenden Funktionalität in zum Beispiel einem autonom fahrenden Fahrzeug angesehen werden. Keine der BPs aus dem PRM steht dabei in einem Widerspruch zu den zuvor beschriebenen Ansätzen einer möglichen Sicherheitsargumentation. Das ASPICE ist an einigen Stellen jedoch nicht so konkret wie die Ziele und vorgeschlagenen Lösungen innerhalb der Ansätze, weswegen einige Aspekte nur von den Referenzprozessen unterstützt und nicht direkt gefordert werden. Weiter zeigte sich, dass gerade die Forderung hinsichtlich der Konsistenz von den Anforderungen über die Daten und die ML-Architektur bis hin zu den Testergebnissen einen besonders großen Mehrwert in der Sicherheitsargumentation liefert und mehrere Sicherheitsziele adressiert. Nicht relevant für die vorgestellten Ansätze einer Sicherheitsargumentation ist zum einen die BP.2 des Prozesses MLE.1, welche vorsieht, dass die Anforderungen strukturiert werden, um diese im Projekt handhabbar zu machen, und jeweils die letzte BP der fünf ML spezifischen Prozesse, welche die Kommunikation der Prozessausgaben an alle relevanten Parteien fordert. Da die Kommunikation in einem Projekt generell jedoch sehr wichtig für den Erfolg des Projektes ist [Fre16], ist die BP zur Kommunikation trotzdem für alle Projekte mit mehreren Entwicklungsparteien relevant.

5. Zusammenfassung

Technologischer Fortschritt und gesetzliche Regularien führen bereits seit mehreren Jahrzehnten zu mehr Sicherheit im Straßenverkehr und einem Rückgang der Anzahl bei Straßenverkehrsunfällen getöteten Menschen pro Jahr. So wurde diese Anzahl von circa 21.000 Getöteten im Jahr 1970 bereits auf circa 2.800 getötete Menschen im Jahr 2022 reduziert [(De23]. Um das Ziel der Europäischen Union und der Vereinten Nationen zu erreichen, bis 2030 diese Anzahl weiter zu halbieren [Kom23], werden auf KI basierende automatisierte und autonome Fahrfunktionen als eine Schlüsseltechnologie angesehen. Diese sollen das Fahrzeug befähigen, selbständig die Umgebung zu erfassen, auszuwerten und Entscheidungen zu treffen, um das Fahrzeug in jeder Situation sicher zu steuern [MRR+15, SKS+20, SQC17].

Die KI umfasst dabei ein breites Feld, in welches sich unter anderem ML und darunter auch DL einordnen [KS17]. Hinsichtlich Aufgaben der Objekterkennung und -klassifikation, wie zum Beispiel einer Verkehrszeichenerkennung, haben sich dabei insbesondere CNNs durchgesetzt [LGW+21, ZZXW19]. CNNs besitzen wie andere tiefere neuronale Netze mehrere miteinander verbundene Schichten mit Neuronen. Innerhalb dieser lernen sie iterativ aus Trainingsdaten Transformationen der Eingangsdaten, um diese auf eine gewünschte Ausgabe abzubilden. Die Neuronen in einem *Convolutional Layer* nehmen dazu eine *Feature Map* als Eingabe entgegen, unterteilen diese in Abschnitte und bilden sie durch eine Filterung oder *Pooling* auf einen Ausgabetensor ab. Die Filterung dient dabei dazu, gelernte Muster in Form der trainierten Gewichte in den Daten wiederzuerkennen, wogegen das Pooling benötigt wird, um die Komplexität des Modells zu reduzieren und den in einer Schicht betrachteten Abschnitt zu vergrößern, wodurch die Vorhersage des CNN verbessert wird [CGF+20, KS17, Kut23, W+20, ZZXW19]. Aufgrund der hohen Komplexität von ML-Modellen mit zum Teil Milliarden von im Training zu optimierenden Parametern und der starken Abhängigkeit des Lernens von den bei der Entwicklung verwendeten Daten bestehen verschiedene Sicherheitsbedenken für den Einsatz dieser Technologie in sicherheitskritischen Systemen wie zum Beispiel einem autonom fahrenden Fahrzeug. Die zu lösenden Probleme sind häufig so komplex, dass unklar ist, ob die verwendeten Daten diese vollständig und korrekt abbilden. Somit scheint, selbst wenn das ML-Modell während der Entwicklung den Anforderungen entsprechend funktioniert, dies als hinreichender Sicherheitsnachweis für den realen Einsatz nicht auszureichen [EEF+18, KS17, SQC17, WSRA20]. Stattdessen sind Sicherheitsbedenken in der Entwicklung systematisch zu identifizieren, durch angemessene Gegenmaßnahmen auf ein akzeptables Restrisiko zu mitigieren und so eine nachvollziehbare Sicherheitsargumentation aufzubauen [Koo23, MRR+15, SM+22, SQC17, WSRA20].

Mit dem PRM des ASPICE 4.0 [VDA23b] wurden fünf neue Referenzprozesse für eine systematische Entwicklung von auf ML basierenden Komponenten

in der Automobilindustrie standardisiert. Diese adressieren die Anforderungsdefinition, die Architekturerstellung, das Training und das Testen der ML-Komponente sowie ein Datenmanagement. Jeder Referenzprozess ist dabei durch seinen Prozesszweck, seine Prozessresultate, seine Basispraktiken und seine Ausgabeinformationselemente beschrieben, welche die Grundlage für die Prozessbewertung gemäß des im ASPICE ebenfalls standardisierten PAM bilden.

Im Rahmen dieser Masterarbeit wurden die fünf ML spezifischen Referenzprozesse während der experimentellen Entwicklung einer Verkehrszeichenerkennung zur Klassifikation von zehn verschiedenen Verkehrszeichen praktisch umgesetzt und damit ihre Anwendbarkeit gezeigt. Dabei wurden zunächst auf Basis eines Brainstormings und einer Recherche in Gesetzen und Normen Anforderungen an die zu entwickelnde ML-Komponente erhoben und in einer Anforderungsspezifikation strukturiert dokumentiert. Weiter wurden alle Anforderungen hinsichtlich ihrer gegenseitigen Abhängigkeiten sowie ihrem Einfluss auf die Betriebsumgebung analysiert und so unter anderem die technische Machbarkeit sowie ihre Angemessenheit für die experimentelle Entwicklung sichergestellt. Anschließend wurde auf Basis der ML-Anforderungen an die Daten ein Datenmanagement aufgebaut und eine Datenbasis für das Training und Testen des ML-Modells erhoben. Dabei wurde sich dazu entschieden, den GTSRB Datensatz[24] zu verwenden, welcher sich durch seine Realitätsnähe auszeichnet [SSSI11] und alle zehn vorgesehenen Verkehrszeichen abdeckt. Basierend auf den weiteren ML-Anforderungen wurde eine Architekturspezifikation als Grundlage für die Implementierung der ML-Komponente entwickelt. Diese spezifiziert die statische Architektur und ihre Schnittstellen, definiert die zu verwendenden *Hyperparameter* des ML-Modells sowie ihre initialen Werte und legt Ziele bezüglich des Ressoucenverbrauches der Architekturelemente fest. Außerdem wurde die beschriebene Architektur bezüglich ihrer Performanz, Erklärbarkeit und freien Verfügbarkeit analysiert und die Konsistenz und Rückverfolgbarkeit zwischen den Architekturelementen und den ML-Anforderungen sichergestellt. Wie in der Architekturspezifikation festgelegt, wurde die ML-Komponente zur Verkehrszeichenerkennung auf Basis eines YOLOv5-Klassifikationsmodells implementiert und gemäß des definierten Trainings- und Validierungsansatzes trainiert. Dieser legt dafür Start- und Endekriterien des Trainings fest und spezifiziert einen Ansatz für die Anpassung der *Hyperparameter*, Erzeugung sowie Modifikation des Trainings- und Validierungsdatensatzes und die Trainings- und Validierungsumgebung. In der dritten Trainingsiteration hat das entwickelte ML-Modell die Trainingsendekriterien erfüllt, indem es eine maximale Validierungsgenauigkeit von 99,8% erreichte und dabei keine systematischen Probleme in der Vorhersage des Modells auffällig wurden. In dem anschließenden Test des trainierten ML-Modells wurde, wie im erstellten Testansatz beschrieben, das Modell auf zwei Datensätzen mit neuen Daten des GTSRB Datensatzes getestet. Dabei konnten sowohl für den systematisch erzeugten

[24] `https://benchmark.ini.rub.de/gtsrb_dataset.html`, zuletzt abgerufen am 30.01.2024

Absicherungsdatensatz, bestehend aus zehn durch natürliche Umweltbedingungen gestörten Bildern pro Klasse, und dem zufällig erzeugten Testdatensatz, bestehend aus je 100 anderen Bildern pro Klasse, alle spezifizierten Anforderungen erfüllt werden. Aufgrund der Rahmenbedingungen der experimentellen Entwicklung wurde kein *Deployed ML Model* zur Integration in einem größeren Softwaresystem erzeugt und die Kommunikation der Prozessergebnisse an relevante Parteien konnte nicht wie im ASPICE gefordert umgesetzt werden, da keine weiteren Parteien existierten.

Innerhalb dieser Masterarbeit wurde, nachdem mit der experimentellen Entwicklung die generelle Anwendbarkeit der fünf im ASPICE 4.0 [VDA23b] neu eingeführten ML spezifischen Referenzprozesse gezeigt wurde, verifiziert, inwiefern die Umsetzung dieser standardisierten Referenzprozesse eine mögliche Sicherheitsargumentation unterstützt. Bestehende Standards wie die ISO 26262 [ISO18], welche die Sicherheit von Straßenfahrzeugen adressieren, betonen zwar bereits, dass das Erreichen einer angemessenen Sicherheit grundsätzlich von den angewandten Produktions-, Service-, Management- und Entwicklungsprozessen abhängt, adressieren jedoch ML nicht direkt und sind daher nicht vollständig anwendbar und ihre Erfüllung nicht ausreichend, um die Sicherheit für den Einsatz von ML-Komponenten zu argumentieren [BBB+20, RVM18, SKS+20, SQC17, WCACP20]. Verschiedene Beiträge aus der Forschung zeigen daher auf, wie eine Sicherheitsargumentation für den Einsatz von ML-Komponenten in sicherheitskritischen Systemen, wie zum Beispiel einem autonom fahrenden Fahrzeug, aufgebaut werden kann. Generell ist dazu zunächst zu spezifizieren, welche Aufgabe die ML-Komponente im System übernehmen soll und anschließend zu analysieren, welche Gefahren durch die Komponente entstehen können. Daraus sind weiter Anforderungen an die Entwicklung abzuleiten, welche durch Methoden und Maßnahmen zu adressieren und mit entsprechenden Evidenzen innerhalb der Sicherheitsargumentation nachzuweisen sind [MSB+21, Pro19a, Pro19b, SKS+20, SM+22]. In [WCACP20] wird dieser generelle Ansatz in einem Muster zur Argumentation der Sicherheit einer ML-Komponente mit vier Hauptzielen und weiteren detaillierten Unterzielen weiter heruntergebrochen und in [RVM18] werden darüber hinaus konkrete Methoden benannt, welche den Nachweis der Sicherheit eines neuronalen Netzes unterstützen. Auf Basis dieser Ansätze, welche den Stand der Forschung bilden, wurde gezeigt, dass die Umsetzung der im ASPICE beschriebenen BPs die Erreichung einer Vielzahl der festgelegten Ziele innerhalb einer Sicherheitsargumentation unterstützt und einige Ziele sogar vollständig adressiert. Weiter konnte gezeigt werden, dass keine BP der ML spezifischen Referenzprozesse in einem Widerspruch zu den Ansätzen der Sicherheitsargumentation stehen. Jedoch wurde auch deutlich, dass das ASPICE in einigen Aspekten nicht so konkret ist wie die Sicherheitsargumentationen, und daher zum Beispiel die in [RVM18] vorgeschlagenen Methoden als Nachweise der Zielerfüllung nicht vom ASPICE gefordert und adressiert werden. Die Sicherheitsargumentationen betonen weiter, dass die Sicherheit einer ML-Komponente nicht einmalig während der

Entwicklung nachzuweisen ist, sondern konstant überwacht werden muss, um Sicherheitsbedenken aufgrund der sich stetig verändernden Realität zu adressieren [MSB+21, SM+22, WSRA20]. Auch dieser Aspekt ist im ASPICE nicht adressiert, da dieses ausschließlich Referenzprozesse während der Entwicklung beschreibt.

Mit dieser Masterarbeit konnte durch die experimentelle Entwicklung einer Verkehrszeichenerkennung von zehn Verkehrszeichen basierend auf ML die Anwendbarkeit der im ASPICE 4.0 [VDA23b] neu eingeführten fünf ML spezifischen Referenzprozesse exemplarisch gezeigt werden. Außerdem wurde verifiziert, dass die Umsetzung der Referenzprozesse eine Sicherheitsargumentation für den Einsatz einer ML-Komponente in einem sicherheitskritischen System, nach dem Stand der Forschung, unterstützt und keine BP in einem Widerspruch zu der Argumentationskette steht. Die Ergebnisse dieser Masterarbeit betonen damit den Mehrwert der im ASPICE 4.0 standardisierten Referenzprozesse als eine Grundlage für sichere Entwicklung von ML-Komponenten in der Automobilindustrie.

6. Ausblick

Innerhalb dieser Masterarbeit wurden die im ASPICE 4.0 [VDA23b] neu eingeführten standardisierten Referenzprozesse zur Entwicklung einer ML-Komponente sowie dem *Machine Learning Data Management* innerhalb einer experimentellen Entwicklung einer Verkehrszeichenerkennung für zehn Verkehrsschilder umgesetzt und damit ihre grundsätzliche Anwendbarkeit gezeigt. Dabei wurden die einzelnen BPs bezüglich des Projektkontextes der experimentellen Entwicklung interpretiert und geforderte Artefakte erzeugt, um den jeweiligen Prozesszweck zu erfüllen. Die Projektrahmenbedingungen und die Komplexität der experimentellen Entwicklung unterschieden sich jedoch stark von einem realen Projekt für die Entwicklung eines Steuergerätes eines modernen Automobils, weswegen durch zukünftige Projekte die Referenzprozesse weiter zu erproben und ihre Anwendbarkeit zu bewerten ist.

Die experimentelle Entwicklung verfolgte das Ziel, die Anwendbarkeit der Referenzprozesse zu zeigen, und es bestand nicht der Anspruch, die entwickelte Verkehrszeichenerkennung in eine größere Software und ein reales System zu integrieren. Aus diesem Grund existierten während der experimentellen Entwicklung keine Anforderungen aus höheren Ebenen des V-Modells, welche innerhalb der Anforderungsanalyse gemäß MLE.1 verfeinert werden konnten. Das Herunterbrechen der relevanten Softwareanforderungen in konkrete ML-Anforderungen und Anforderungen an die Daten stellt zwar einen wesentlichen Aspekt des Prozesses MLE.1 dar, ist jedoch in vergleichbarer Weise bereits im ASPICE 3.1 [VDA17] für das Herunterbrechen von Systemanforderungen zu Softwareanforderungen enthalten und daher kein spezieller neuer Aspekt des ASPICE 4.0. Weiter wurde, da die entwickelte Verkehrszeichenerkennung nicht in ein größeres Softwaresystem integriert werden sollte, kein *Deployed ML Model* aus dem trainierten ML-Modell abgeleitet und erneut getestet. Das Erstellen eines *Deployed ML Model* ist dabei ein spezieller Aspekt des Prozesses MLE.4 für den durch weitere Forschungs- und Entwicklungsprojekte gezeigt werden muss, wie dieser angemessen umgesetzt werden kann. Aktuelle Forschungen zeigen dazu zum Beispiel, dass das Entfernen von redundanten Neuronen, welche die Güte der Vorhersage des ML-Modells nicht erhöhen, eine Möglichkeit ist, die Komplexität des *Deployed Models* zu reduzieren ohne die Güte der Vorhersage stark zu verschlechtern [LGW+21]. Das Testen des *Deployed ML Model* basiert dann im Wesentlichen auf einer angepassten ML-Architektur, dem Testansatz und den Testdaten, welche in gleicher Weise für das trainierte ML-Modell erstellt worden sind. Da der Entwicklungsumfang der experimentellen Entwicklung gegenüber dem zu erwartenden Umfang einer realen Entwicklung für den Einsatz einer ML-Komponente innerhalb eines Fahrzeuges stark reduziert war, konnte die letzte BP jedes MLE-Prozesses hinsichtlich der Kommunikation der Prozessergebnisse an relevante Parteien nicht umgesetzt werden. Verschiedene weitere BPs konnten außerdem aus diesem Grund simpler realisiert werden, als es in einem realen Projekt zu erwarten ist. So konnte in der experimentellen Entwicklung zum

Beispiel die Konsistenz und Rückverfolgbarkeit zwischen den Entwicklungsartefakten durch eine Excel Tabelle und Namenskonventionen manuell realisiert werden. In umfangreicheren Projekten wäre hingegen eine toolunterstützte Rückverfolgbarkeit zu verwenden, um den manuellen Aufwand zur Erzeugung und Pflege der Rückverfolgbarkeit und Konsistenz, was eine große Herausforderung darstellen kann, zu minimieren [MSS18]. Weiter traten in der experimentellen Entwicklung, da die Entwicklungsdauer und die Anzahl an *Stakeholdern* gegenüber einer realen Entwicklung deutlich geringer waren, keine Änderungen an den Anforderungen auf. Dadurch waren keine verschiedenen Iterationen der Anwendung der Prozesse erforderlich, sodass viele Artefakte nur in einer Version erzeugt werden mussten. All diese Aspekte vereinfachten die Anwendbarkeit der im ASPICE neu eingeführten ML spezifischen Referenzprozesse in der experimentellen Entwicklung und machen somit weitere realitätsnähere Projekte in größerem Rahmen erforderlich, um die Ergebnisse dieser Masterarbeit zu bestätigen.

Nachdem innerhalb der experimentellen Entwicklung die Anwendbarkeit der ML spezifischen Referenzprozesse gezeigt wurde, wurden die BPs dieser kritisch hinterfragt und bezüglich ihrer Unterstützung einer möglichen Sicherheitsargumentation für den Einsatz einer ML-Komponente in einem sicherheitskritischen System eingeordnet. Als Grundlage dazu dienten verschiedene Ansätze einer Sicherheitsargumentation gemäß dem aktuellen Stand der Forschung. Diese sind jedoch überwiegend in der Praxis bisher nicht erprobt und besitzen keinen Anspruch auf Vollständigkeit. Das Ergebnis dieser Masterarbeit, dass durch die Anwendung eines Entwicklungsprozesses, welcher zu den Referenzprozessen des ASPICE 4.0 konform ist, eine Vielzahl der Aspekte einer möglichen Sicherheitsargumentation adressiert werden und die Erreichung eines Prozessfähigkeitslevels von mindestens eins somit als einer der Nachweise zur Argumentation der Sicherheit einer auf ML basierenden Funktionalität in zum Beispiel einem autonom fahrenden Fahrzeug angesehen werden kann, ist somit zusammen mit den betrachteten Ansätzen zur Sicherheitsargumentation noch durch weitere Forschungsprojekte und reale Entwicklungen zu bestätigen. Insofern weitere Aspekte für die Sicherheitsargumentation, über die betrachteten Ansätze hinaus, identifiziert werden, so ist für diese ebenfalls zu bewerten, ob sie durch die Anwendung der Referenzprozesse des ASPICE unterstützt werden oder die standardisierten BPs möglicherweise in einem Widerspruch zu diesen stehen. Dabei ist zu beachten, dass über die Nachweise innerhalb der Sicherheitsargumentation lediglich die Erreichung eines tolerierbaren Restrisikos nachgewiesen werden wird. Wobei bisher nicht festgelegt ist, wie genau dieses Restrisiko bestimmt werden kann und wie hoch dieses maximal sein darf [SM+22, WSRA20].

Neben den in dieser Masterarbeit behandelten Sicherheitsbedenken, welche innerhalb einer Sicherheitsargumentation und über die Anwendung angemessener Entwicklungsprozesse adressiert werden können, bestehen auch ethische

Fragen und Bedenken hinsichtlich des Einsatzes von ML-Komponenten in Automobilen, welche in dieser Masterarbeit nicht diskutiert wurden. Ethische Bedenken adressieren dabei vor allem fehlende Transparenz darüber, wie eine ML-Komponente zu seiner Vorhersage gekommen ist und negative Folgen einer Entscheidung, bei denen Menschen oder Dinge zu Schaden kommen. Trainierte Algorithmen können zum Beispiel Minderheiten, welche in den verwendeten Daten unterrepräsentiert waren, benachteiligen, wodurch das Gleichheitsgebot zwischen verschiedenen Menschen verletzt wird [Man19]. Um diese Bedenken zu adressieren sind sie sowie auch Sicherheitsbedenken während der Entwicklung durch Anforderungen an das System zu berücksichtigen und ihre Umsetzung nachzuweisen. Mit [IEE22] wurde dazu bereits ein Standard eingeführt, welcher Entwickler unterstützt, ihre Systeme ausreichend transparent aufzubauen. Durch zukünftige Forschungs- und Entwicklungsprojekte sind ethische Bedenken und Sicherheitsbedenken zu kombinieren, um eine gesamtheitliche Argumentation aufzubauen, dass eine entwickelte ML-Komponente tatsächlich für den realen Einsatz geeignet ist. Dabei ist zu prüfen, inwiefern die Umsetzung der im ASPICE 4.0 standardisierten Referenzprozesse auch ethische Bedenken adressieren und was darüber hinaus in der Entwicklung zu berücksichtigen ist.

Literaturverzeichnis

[Ass18] Assurance Case Working Group. Goal structuring notation community standard v2. Standard, Safety-Critical Systems Club, 2018.

[BBB+20] John Birch, David Blackburn, John Botham, Ibrahim Habli, David Higham, Helen Monkhouse, Gareth Price, Norina Ratiu, and Roger Rivett. A structured argument for assuring safety of the intended functionality (sotif). In *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops: DECSoS 2020, DepDevOps 2020, USDAI 2020, and WAISE 2020, Lisbon, Portugal, September 15, 2020, Proceedings 39*, pages 408–414. Springer, 2020.

[Bun19] Bundesamt für Justiz. Verordnung über die Teilnahme von Elektrokleinstfahrzeugen am Straßenverkehr (Elektrokleinstfahrzeuge-Verordnung - eKFV), 2019. Verfügbar unter: `https://www.gesetze-im-internet.de/ekfv/BJNR075610019.html`, zuletzt aufgerufen am 30.01.2024.

[CGF+20] Kenny Choo, Eliska Greplova, Mark H Fischer, Titus Neupert, T Neupert, and Mark Fischer. *Machine learning kompakt*. Springer, 2020.

[Cru22a] Cruise. Addendum: Operation design domain - driverless deployment in california, 2022. Verfügbar unter: `https://www.cpuc.ca.gov/-/media/cpuc-website/divisions/consumer-protection-and-enforcement-division/documents/tlab/av-programs/cruise-driverless-deployment-odd-2023-08.pdf`, zuletzt aufgerufen am 17.01.2024.

[Cru22b] Cruise. Cruise safety report, 2022. Verfügbar unter: `https://assets.ctfassets.net/95kuvdv8zn1v/zKJHD7X22fNzpAJztpd5K/ac6cd2419f2665000e4eac3b7d16ad1c/Cruise_Safety_Report_2022_sm-optimized.pdf`, zuletzt aufgerufen am 11.12.2023.

[(De23] Statistisches Bundesamt (Destatis). Verkehrsunfälle, 2023. Verfügbar unter: `https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Verkehrsunfaelle/_inhalt.html`, zuletzt aufgerufen am 28.11.2023.

[DFO20] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.

[DHH23] Akshay Dhonthi, Ernst Moritz Hahn, and Vahid Hashemi. Backdoor mitigation in deep neural networks via strategic retraining. In *International Symposium on Formal Methods*, pages 635–647. Springer, 2023.

[DLGZ+23] Luigi Di Lillo, Tilia Gode, Xilin Zhou, Margherita Atzei, Ruoshu Chen, and Trent Victor. Comparative safety performance of autonomous-and human drivers: A real-world case study of the waymo one service. *arXiv preprint arXiv:2309.01206*, 2023.

[EEF+18] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1625–1634, 2018.

[Eur21] Europäische Kommission, Generaldirektion Binnenmarkt, Industrie, Unternehmertum und KMU. Delegierte Verordnung (EU) 2021/1958 der Kommission, 2021. Verfügbar unter: `https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32021R1958`, zuletzt aufgerufen am 26.01.2024.

[Fre16] Matthias Freitag. *Kommunikation im Projektmanagement*. Springer, 2016.

[GRP+12] Xiaocheng Ge, Rui Rijo, Richard F Paige, Tim P Kelly, and John A McDermid. Introducing goal structuring notation to explain decisions in clinical practice. *Procedia Technology*, 5:686–695, 2012.

[Hey19] Tim Hey. *Die außervertragliche Haftung des Herstellers autonomer Fahrzeuge bei Unfällen im Straßenverkehr*. Springer, 2019.

[IEE22] IEEE standard for transparency of autonomous systems. *IEEE Std 7001-2021*, pages 1–54, 2022.

[ISO18] ISO/TC 22/SC 32. ISO 26262: Road vehicles - functional safety. Standard, International Organization for Standardization, 2018.

[ISO22] ISO/TC 22/SC 32. ISO 21448: Road vehicles - safety of the intended functionality. Standard, International Organization for Standardization, 2022.

[ISOng] ISO/TC 22/SC 32. ISO/CD PAS 8800: Road vehicles - safety and artificial intelligence. Standard, International Organization for Standardization, In Entwicklung.

[JCS+22] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, Yonghye Kwon, Kalen Michael, Jiacong Fang, Colin Wong, Zeng Yifu, Diego Montes, et al. ultralytics/yolov5: v6. 2-yolov5 classification models, apple m1, reproducibility, clearml and deci. ai integrations. *Zenodo*, 2022.

[Kel04] Tim Kelly. A systematic approach to safety case management. *SAE transactions*, pages 257–266, 2004.

[Kom23] Europäische Kommission. Straßenverkehrssicherheit in der EU: Anzahl der tödlich Verunglückten unter vorpandemischem Niveau – weiterhin jedoch zu langsame Fortschritte. Pressemitteilung, 2023. Verfügbar unter: https://www.mobilitaetsforum.bund.de/SharedDocs/Downloads/DE/Sonstiges/PM_EU_Strassenverkehrsstatistik2022.pdf;jsessionid=CC88CC7627478AE2D49BA763B76D72D7.live21321?__blob=publicationFile&v=3, zuletzt aufgerufen am 26.01.2024.

[Koo23] Philip Koopman. Ul 4600: What to include in an autonomous vehicle safety case. *Computer*, 56(05):101–104, 2023.

[Kor22] Uli Korsch. Katalog der Verkehrszeichen (VzKat), 2022. Verfügbar unter: http://www.vzkat.de/2017/VzKat.htm, zuletzt aufgerufen am 06.02.2024.

[KS17] Nikhil Ketkar and Eder Santana. *Deep Learning with Python*, volume 1. Springer, 2017.

[Kut23] Gitta Kutyniok. An introduction to the mathematics of deep learning. *8th European Congress of Mathematics*, 2023.

[LGW+21] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

[Man19] Axel Mangelsdorf. Normen und standards in der ki. *Künstliche Intelligenz: Technologie | Anwendung | Gesellschaft*, pages 48–57, 2019.

[MM10] Sonali Mathur and Shaily Malik. Advancements in the v-model. *International Journal of Computer Applications*, 1(12):29–34, 2010.

[Mob23] Mobileye. The mobileye safety methodology, 2023. Verfügbar unter: https://www.mobileye.com/technology/safety-methodology/, zuletzt aufgerufen am 11.12.2023.

[MRR+15] Richard Matthaei, Andreas Reschka, Jens Rieken, Frank Dierkes, Simon Ulbrich, Thomas Winkle, and Markus Maurer. Autonomes fahren. *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*, pages 1139–1165, 2015.

[MSB+21] Michael Mock, Stephan Scholz, Frédérik Blank, Fabian Hüger, Andreas Rohatschek, Loren Schwarz, and Thomas Stauner. An integrated approach to a safety argumentation for ai-based perception functions in automated driving. In *Computer Safety, Reliability, and Security. SAFECOMP 2021 Workshops: DECSoS, MAPSOD, DepDevOps, USDAI, and WAISE, York, UK, September 7, 2021, Proceedings 40*, pages 265–271. Springer, 2021.

[MSS18] Salome Maro, Jan-Philipp Steghöfer, and Miroslaw Staron. Software traceability in the automotive domain: Challenges and solutions. *Journal of Systems and Software*, 141:85–110, 2018.

[NAIH22] Omran Nacir, Maraoui Amna, Werda Imen, and Belgacem Hamdi. Yolo v5 for traffic sign recognition and detection using transfer learning. In *2022 IEEE International Conference on Electrical Sciences and Technologies in Maghreb (CISTEM)*, volume 4, pages 1–4. IEEE, 2022.

[On-21] On-Road Automated Driving (ORAD) Committee. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, 2021.

[Pro19a] PEGASUS Projekt. PEGASUS method an overview, 2019. Verfügbar unter: `https://www.pegasusprojekt.de/files/tmpl/Pegasus-Abschlussveranstaltung/PEGASUS-Gesamtmethode.pdf`, zuletzt aufgerufen am 23.10.2023.

[Pro19b] PEGASUS Projekt. Safety argument goal - boot no. 29, 2019. Verfügbar unter: `https://www.pegasusprojekt.de/files/tmpl/Pegasus-Abschlussveranstaltung/29_Safety_Argument.pdf`, zuletzt aufgerufen am 23.10.2023.

[PyT23] PyTorch. Crossentropyloss, 2023. Verfügbar unter: `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`, zuletzt aufgerufen am 02.11.2023.

[QY21] Zhongbing Qin and Wei Qi Yan. Traffic-sign recognition using deep learning. In *Geometry and Vision: First International Symposium, ISGV 2021, Auckland, New Zealand, January 28-29, 2021, Revised Selected Papers 1*, pages 13–25. Springer, 2021.

[Roo86] Paul Rook. Controlling software projects. *Software engineering journal*, 1(1):7–16, 1986.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[RVM18] Alexander Rudolph, Stefan Voget, and Jürgen Mottok. A consistent safety case argumentation for artificial intelligence in safety related automotive systems. In *ERTS 2018*, 2018.

[SKS+20] Gesina Schwalbe, Bernhard Knie, Timo Sämann, Timo Dobberphul, Lydia Gauerhof, Shervin Raafatnia, and Vittorio Rocco. Structuring the safety argumentation for deep neural network based perception in

automotive applications. In *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops: DECSoS 2020, DepDevOps 2020, USDAI 2020, and WAISE 2020, Lisbon, Portugal, September 15, 2020, Proceedings 39*, pages 383–394. Springer, 2020.

[SM+22] Stephan Scholz, Michael Mock, et al. Abschlussbericht ki absicherung, 2022. Verfügbar unter: `https://www.ki-absicherung-projekt.de/fileadmin/KI_Absicherung/Downloads/KI-A_Abschlussberich_PU_final.pdf`, zuletzt aufgerufen am 19.10.2023.

[SP21] Daria Snegireva and Anastasiia Perkova. Traffic sign recognition application using yolov5 architecture. In *2021 International Russian Automation Conference (RusAutoCon)*, pages 1002–1007. IEEE, 2021.

[SQC17] Rick Salay, Rodrigo Queiroz, and Krzysztof Czarnecki. An analysis of iso 26262: Using machine learning safely in automotive software. *arXiv preprint arXiv:1709.02435*, 2017.

[SSSI11] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *The 2011 international joint conference on neural networks*, pages 1453–1460. IEEE, 2011.

[UL 23] UL Standard. UL 4600: Standard for safety for evaluation of autonomous products. Standard, ANSI/UL, 2023.

[VDA17] VDA QMC Working Group 13 / Automotive SIG. Automotive spice process assessment / reference model v3.1. Standard, Verband der Automobilindustrie Qualitäts-Management-Center, 2017.

[VDA23a] VDA QMC. Automotive spice guidelines v2. Richtlinie, Verband der Automobilindustrie Qualitäts-Management-Center, 2023.

[VDA23b] VDA Working Group 13. Automotive spice process assessment / reference model v4.0. Standard, Verband der Automobilindustrie Qualitäts-Management-Center, 2023.

[W+20] Phil Wennker et al. *Künstliche Intelligenz in der Praxis*. Springer, 2020.

[WCACP20] Ernest Wozniak, Carmen Cârlan, Esra Acar-Celik, and Henrik J Putzer. A safety case pattern for systems with machine learning components. In *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops: DECSoS 2020, DepDevOps 2020, USDAI 2020, and WAISE 2020, Lisbon, Portugal, September 15, 2020, Proceedings 39*, pages 370–382. Springer, 2020.

[WSL+20] Nick Webb, Dan Smith, Christopher Ludwick, Trent Victor, Qi Hommes, Francesca Favaro, George Ivanov, and Tom Daniel. Waymo's safety methodologies and safety readiness determinations. *arXiv preprint arXiv:2011.00054*, 2020.

[WSRA20] Oliver Willers, Sebastian Sudholt, Shervin Raafatnia, and Stephanie Abrecht. Safety concerns and mitigation approaches regarding the use of deep learning in safety-critical perception tasks. *arXiv preprint arXiv:2001.08001*, 2020.

[YBS+23] Xiao Yang, Ramesh Bist, Sachin Subedi, Zihao Wu, Tianming Liu, and Lilong Chai. An automatic classifier for monitoring applied behaviors of cage-free laying hens with deep learning. *Engineering Applications of Artificial Intelligence*, 123:106377, 2023.

[ZY22] Yanzhao Zhu and Wei Qi Yan. Traffic sign recognition based on deep learning. *Multimedia Tools and Applications*, 81(13):17779–17791, 2022.

[ZZXW19] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232, 2019.

A. Anforderungsspezifikation

Anforderungbeschreibung			Anforderungsattribute		Referenzen			
ID	Anforderungstext	Ergänzung	Analyse	Klassifizierung	Architectural Element	Training & Validation Data	Test Approach	Test Data
1. Generelle Anforderungen								
Req_001	Die ML-Komponente muss auf einem CNN basieren.		Freigegeben	ML-Req	Arch_ID_01	/	/	/
Req_002	Alle verwendeten Daten müssen entweder selber erzeugt oder frei verfügbar sein.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	Test_Dataset_01, Assurance_Dataset_01
Req_003	Verwendeter Quellcode, eingebundene Bibliotheken und Architekturelemente müssen selber geschrieben oder frei verfügbar sein.		Freigegeben	ML-Req	Arch_ID_01, Arch_ID_02	/	/	/
Req_004	Es ist ein Computer mit 16GB Ram, einem Intel i7-4790K Prozessor, einer Nvidia GTX 970 Grafikkarte und Windows 10 als Betriebssystem während der Entwicklung zu verwenden.		Freigegeben	ML-Req	/	/	Testansatz_Kap.5	/
2. Funktionale Anforderungen								
Req_005	Das CNN muss .ppm Bilder mit verschiedener Auflösung als Input für das Training entgegennehmen können.		Freigegeben	ML-Req	Arch_ID_01	/	/	/
Req_006	Das CNN muss .jpg Bilder mit verschiedener Auflösung als Input für das Testing entgegennehmen können.		Freigegeben	ML-Req	Arch_ID_01	/	/	/
Req_007	Die Ausgabe des Modells muss die Wahrscheinlichkeit für die Klassifizierung folgender Verkehrszeichen unter der folgenden Nummerierung angeben: 0 = Verkehrszeichen 101 (Gefahrstelle) 1 = Verkehrszeichen 123 (Baustelle) 2 = Verkehrszeichen 205 (Vorfahrt gewähren) 3 = Verkehrszeichen 267 (Einfahrt verboten) 4 = Verkehrszeichen 274-30 (zulässige Höchstgeschwindigkeit 30) 5 = Verkehrszeichen 274-50 (zulässige Höchstgeschwindigkeit 50) 6 = Verkehrszeichen 274-100 (zulässige Höchstgeschwindigkeit 100) 7 = Verkehrszeichen 276 (Überholverbot für Kraftfahrzeuge aller Art) 8 = Verkehrszeichen 301 (Vorfahrt) 9 = Verkehrszeichen 306 (Vorfahrtsstraße)		Freigegeben	ML-Req	Arch_ID_01	/	/	/
3. Performanz und Ressourcen Anforderungen								
Req_008	Eine Trainingsiteration darf maximal 3h Rechenzeit in Anspruch nehmen.		Freigegeben	ML-Req	Arch_ID_01	/	/	/
Req_009	Die Verarbeitung eines Bildes aus den Test- und Absicherungsdaten darf im Durchschnitt nicht mehr als 1sec benötigen.		Freigegeben	ML-Req	Arch_ID_01	/	Testansatz_Kap.2	/
Req_010	Die Auswertung der Testergebnisse hinsichtlich dem Verhältnis korrekter Vorhersagen muss innerhalb von 5min möglich sein.		Freigegeben	ML-Req	Arch_ID_02	/	Testansatz_Kap.2	/
Req_011	Die ML-Komponente muss bei dem abschließenden Test mit dem Testdatensatz mit einer Wahrscheinlichkeit von >98% eine korrekte Vorhersage erzeugen.	Höher als EU_2021/1958_3.4.2.5.2	Freigegeben	ML-Req	Arch_ID_01, Arch_ID_02	/	Testansatz_Kap.2	/
Req_012	Die ML-Komponente muss bei dem Test mit dem Absicherungsdatensatz mit einer Wahrscheinlichkeit von >95% eine korrekte Vorhersage erzeugen.	Höher als EU_2021/1958_3.4.2.5.2	Freigegeben	ML-Req	Arch_ID_01, Arch_ID_02	/	Testansatz_Kap.2	/
Req_013	Die ML-Komponente darf maximal einen Speicherplatz von 100 MB in Anspruch nehmen.		Freigegeben	ML-Req	Arch_ID_01, Arch_ID_02	/	Testansatz_Kap.2	/
4. Datenstruktur								
Req_014	Bilder müssen eine Auflösung von mindestens 15x15 Pixeln haben und müssen genau ein Verkehrszeichen abbilden.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	Test_Dataset_01, Assurance_Dataset_01
Req_015	Die 4 verschiedenen Datensätze (Training, Validierung, Test, Absicherung) müssen den **Klassenkatalog (siehe 5.)** vollständig abdecken.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	Testansatz_Kap.3	Test_Dataset_01, Assurance_Dataset_01
Req_016	Die 4 verschiedenen Datensätze (Training, Validierung, Test, Absicherung) dürfen keine vollständig identischen Bilder beinhalten.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	Testansatz_Kap.3	Test_Dataset_01, Assurance_Dataset_01
Req_017	Pro Klasse müssen in der Datenbasis mindestens 1400 Bilder vorhanden sein.		Freigegeben	Data-Req	/	/	/	/
Req_018	Für das Training müssen die Bilder des Trainings- und Validierungsdatensatzes jeweils als .ppm Datei vorliegen.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	/
Req_019	Für das Training müssen die Bilder des Trainings- und Validierungsdatensatzes jeweils in einem Ordner pro Klasse separiert werden, welche die Nummer und den Klassennamen durch einen Unterstrich getrennt als Namen besitzt z.B. 2_VorfahrtGewähren.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	/
Req_020	Die Verteilung der Daten auf den Trainings- und Validierungsdatensatz muss zufällig aus einer gemeinsamen Grundmenge erfolgen.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	/
Req_021	Der Validierungsdatensatz muss pro Klasse min. 150 .ppm Bilder enthalten.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	/
Req_022	Der Testdatensatz muss pro Klasse genau 100 .jpg Bilder enthalten.		Freigegeben	Data-Req	/	/	Testansatz_Kap.3	Test_Dataset_01
Req_023	Der Absicherungsdatensatz muss pro Klasse genau 10 durch natürliche Umwelteinflüsse gestörte .jpg Bilder enthalten (z.B. außergewöhnlich hohe Belichtung, Verdeckung etc.).		Freigegeben	Data-Req	/	/	Testansatz_Kap.3	Assurance_Dataset_01
5. Klassenkatalog (Was für Klassen / Schilder müssen enthalten sein)								
Req_024	Es müssen Bilder des Verkehrszeichens 101 (Gefahrstelle) enthalten sein.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	Test_Dataset_01, Assurance_Dataset_01
Req_025	Es müssen Bilder des Verkehrszeichens 123 (Baustelle) enthalten sein.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	Test_Dataset_01, Assurance_Dataset_01
Req_026	Es müssen Bilder des Verkehrszeichens 205 (Vorfahrt gewähren) enthalten sein.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	Test_Dataset_01, Assurance_Dataset_01
Req_027	Es müssen Bilder des Verkehrszeichens 267 (Einfahrt verboten) enthalten sein.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	Test_Dataset_01, Assurance_Dataset_01
Req_028	Es müssen Bilder des Verkehrszeichens 274-30 (zulässige Höchstgeschwindigkeit 30) enthalten sein.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	Test_Dataset_01, Assurance_Dataset_01
Req_029	Es müssen Bilder des Verkehrszeichens 274-50 (zulässige Höchstgeschwindigkeit 50) enthalten sein.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	Test_Dataset_01, Assurance_Dataset_01
Req_030	Es müssen Bilder des Verkehrszeichens 274-100 (zulässige Höchstgeschwindigkeit 100) enthalten sein.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	Test_Dataset_01, Assurance_Dataset_01
Req_031	Es müssen Bilder des Verkehrszeichens 276 (Überholverbot für Kraftfahrzeuge aller Art) enthalten sein.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	Test_Dataset_01, Assurance_Dataset_01
Req_032	Es müssen Bilder des Verkehrszeichens 301 (Vorfahrt) enthalten sein.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	Test_Dataset_01, Assurance_Dataset_01
Req_033	Es müssen Bilder des Verkehrszeichens 306 (Vorfahrtsstraße) enthalten sein.		Freigegeben	Data-Req	/	TrainVal_Dataset_01	/	Test_Dataset_01, Assurance_Dataset_01

B. Architekturspezifikation

Architekturspezifikation

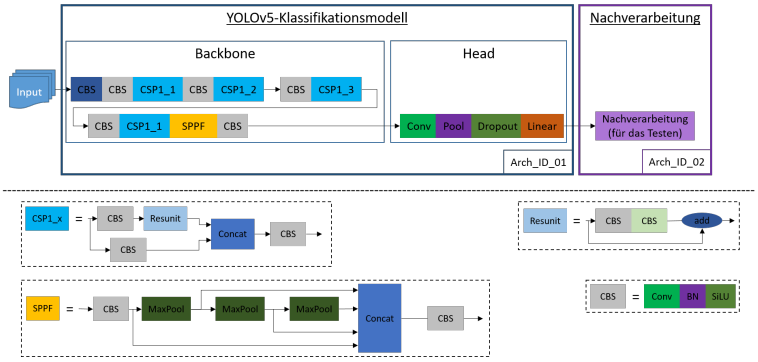
1. Vorwort

Dieses Dokument entspricht der Spezifikation der Architektur der Verkehrszeichenerkennung, welche während der experimentellen Entwicklung innerhalb der Masterarbeit zu dem Thema „Verifikation standardisierter Referenzprozesse zur Entwicklung von ML Komponenten im Automobilkontext bezüglich Anwendbarkeit im sicherheitskritischen Kontext und Unterstützung einer möglichen Sicherheitsargumentation“ angefertigt wurde.

2. Zweck

Die Architekturspezifikation verfolgt den Zweck, alle Aspekte der Architektur zu spezifizieren, welche benötigt werden, um die Verkehrszeichenerkennung zu implementieren, trainieren und zu testen.

3. Blockdiagramm / Statische Architektur



Vollständiges Blockdiagramm der Architektur, basierend auf [YBS+23]

4. Parameter

a) Trainingsparameter

Für das Training sind initial 750 Bilder im Trainingsdatensatz und 150 Bilder im Validierungsdatensatz zu verwenden.

Folgend werden die relevanten Parameter für das Training auf Basis der classify/train.py beschrieben sowie ihre initialen Werte für den Trainingsstart. Nicht beschriebene Parameter der train.py sind während des Trainingsstartes nicht zu verwenden und ihre Defaultwerte nicht anzupassen.

Parameter	Beschreibung	Wertebereich	initialer Wert
--model	Dateipfad des vortrainierten Modells mit den initialen Gewichten.	yolov5n-cls.pt yolov5s-cls.pt yolov5m-cls.pt yolov5l-cls.pt yolov5x-cls.pt	yolov5s-cls.pt

--data	Dateipfad zu den Trainings- und Validierungsdaten	String	../TrainVal_Dataset
--device	Angabe der Grafikkarte / cuda Device Nummer	String	0

b) Testparameter

Folgend werden die relevanten Parameter für das Testing auf Basis der classify/predict.py beschrieben sowie die anzugebenden Werte. Nicht beschriebene Parameter der predict.py sind nicht zu verwenden und ihre Defaultwerte nicht anzupassen.

Parameter	Beschreibung	Wertebereich	Wert
--weights	Dateipfad zu dem trainierten Modell mit den trainierten Gewichten / Weights.	String	runs/train-cls/exp<*INDEX*>/weights/best.pt
--source	Dateipfad zu den Testdaten.	String	../Test_Dataset_JPG
--device	Angabe der Grafikkarte / cuda Device Nummer.	String	0
--save-txt	Aktiviert die Speicherung der Ergebnisse innerhalb einer txt Datei pro Bild.	/	Gesetzt

5. Analysekriterien

Die folgenden Kriterien sind bei der Analyse der ML Architektur und ihrer Elemente zu beachten:

Analysekriterium	Zielbeschreibung
Performanz	Innerhalb anderen, bestenfalls ähnlichen Klassifikationsproblemen, sollte eine den Anforderungen entsprechende Performanz der Architektur bzw. Architekturelemente nachgewiesen worden sein.
Erklärbarkeit	Es sollte eine Dokumentation oder Literatur bezüglich der Architektur bzw. Architekturelemente vorliegen, welche die Funktion beschreibt.
Freie Verfügbarkeit	Die Architektur bzw. Architekturelemente müssen selber programmiert oder frei verfügbar sein.

6. Schnittstellenbeschreibung

Da die Integration der ML Komponente in eine größere Software nicht vorgesehen ist, bestehen keine Schnittstellen zu anderen Softwarekomponenten. Innerhalb der ML Komponente besteht folgende Schnittstelle:

Quelle	Senke	Beschreibung
Arch_ID_01	Arch_ID_02	YOLOv5-cls speichert die Vorhersagen innerhalb einer .txt Datei, dabei wird pro Zeile zuerst die Wahrscheinlichkeit angegeben und anschließend durch ein Leerzeichen getrennt die Klassenbezeichnung. Es werden dabei immer nur die fünf Klassen mit höchster Wahrscheinlichkeit in dem Textdokument, absteigend gemäß ihrer Wahrscheinlichkeit sortiert, angegeben. - Die Wahrscheinlichkeit wird als Dezimalzahl mit zwei Nachkommastellen in englischer Notation mit Punkt anstelle eines Kommas notiert. Das Element für die Nachverarbeitung muss die .txt Datei öffnen und die Klasse inkl. der Wahrscheinlichkeit einlesen.

7. Ziele bezüglich des Ressourcenverbrauches

Aus den spezifizierten Anforderungen ergeben sich folgende Ziele bezüglich des Ressourcenverbrauches der Architekturelemente unter Verwendung der in Req_004 spezifizierten Zielhardware:

Betroffenes Architekturelement	Zielbeschreibung	Anforderungsreferenz
Arch_ID_01	Das YOLOv5-cls Modell darf maximal 3h Rechenzeit für eine vollständige Trainingsiteration benötigen.	Req_008
Arch_ID_01	Das YOLOv5-cls Modell darf maximal 1sec im Durschnitt für die Verarbeitung eines Bildes aus dem Test- und Absicherungsdatensatz benötigen.	Req_009
Arch_ID_02	Das Element für die Nachverarbeitung darf maximal 5min für die Auswertung der Testergebnisse hinsichtlich dem Verhältnis korrekter Vorhersagen benötigen.	Req_010
Arch_ID_01, Arch_ID_02	Das YOLOv5-cls Modell und das Element für die Nachverarbeitung dürfen gemeinsam nicht mehr als 100 MB Speicherplatz in Anspruch nehmen.	Req_013

8. Traceability Konzept

Zur Sicherstellung der bidirektionalen Rückverfolgbarkeit (engl. Traceability) ist innerhalb der Anforderungsspezifikation pro Anforderung in dem Attribut „Architectural Element“, durch den Identifikator des Architekturelementes Arch_ID_xx, anzugeben, innerhalb welchem Architekturelement die Anforderung umgesetzt wird. Insofern eine Anforderung nicht durch ein Architekturelement umzusetzen ist, ist ein „/“ in dem Attribut einzutragen.

Durch Filterung innerhalb des Attributes „Architectural Element“ kann weiterhin die Gesamtheit der für ein Architekturelement relevanten Anforderungen nachvollzogen werden.

C. Trainings- und Validierungsansatz

Trainings- und Validierungsansatz

1. Vorwort

Dieses Dokument entspricht der Beschreibung des Ansatzes für das Training und die Validierung während der experimentellen Entwicklung innerhalb der Masterarbeit zu dem Thema „Verifikation standardisierter Referenzprozesse zur Entwicklung von ML Komponenten im Automobilkontext bezüglich Anwendbarkeit im sicherheitskritischen Kontext und Unterstützung einer möglichen Sicherheitsargumentation“.

2. Start- und Endekriterien

Folgende Start- und Endekriterien für das ML Training sind zu berücksichtigen.

Startkriterien:
- Anforderungen an die ML Komponente sind spezifiziert, gemäß ihrer Korrektheit, technische Machbarkeit, Testbarkeit sowie ihrem Einfluss auf die Betriebsumgebung analysiert und freigegeben.

UND
- Die ML Architektur ist spezifiziert, bezüglich im Projekt definierter Kriterien analysiert und freigegeben.

UND
- Eine den spezifizierten Anforderungen an die Daten entsprechende Menge an Daten ist verfügbar, welche gemäß definierten Kriterien qualitätsgesichert wurde.

Endekriterien:
- Das ML Modell klassifiziert mindesten 98% der ausgewählten Validierungsdaten korrekt (meint die richtige Klasse wird mit der höchsten Wahrscheinlichkeit vorhergesagt) und es sind keine systemischen Probleme auffällig, sodass das Modell in der Lage scheint, die spezifizierten Anforderungen zu erfüllen.

ODER
- Das Training der ML Komponente wurde für 8 Wochen mit mindestens 10 Trainingsiterationen durchgeführt, ohne dass die nötige Performanz erreicht werden konnte.

3. Hyperparameter Tuning

Der Trainingsloss, Validierungsloss (basierend auf der nn.CrossEntropyLoss von PyTorch) sowie die durchschnittliche Korrektheit der Vorhersagen für die Validierungsdaten sind als Metriken während der Trainingsiteration pro Epoche zu ermitteln und als Input für das Hyperparameter Tuning zu verwenden. Folgende Reaktionen sind als Hyperparameter Tuning auf Basis der drei Metriken möglich.

Auffälligkeit	Hyperparameter Anpassung in neuer Trainingsiteration
Reduzierung des Trainings- und Validierungslosses ist bis zur letzten Epoche nicht stagniert.	Erhöhung der Anzahl der Epochen.
Trainings- und Validierungsloss stagnieren seit mehreren abschließenden Epochen auf einem konstanten Niveau.	Verringerung der Anzahl der Epochen.
Accuracy unterhalb des Zielwertes.	- Anpassung des Trainings- und Validierungsdatensatzes gemäß der in 4. beschriebenen Modifikation des Datensatzes. ODER - Verwendung eines anderen vortrainierten YOLOv5 Klassifikationsmodells.

Neben der Analyse der drei Metriken sind die falsch klassifizierten Validierungsdaten hinsichtlich ihrer Zusammenhänge zu analysieren, um systematischen Problemen des Modells oder Bias innerhalb der Daten entgegenzuwirken.

4. Erzeugung und Modifikation des Datensatzes

Es sind initial durch die Verwendung des VBA Skriptes der Datei AuswahlTrainingsdaten.xlsm zwei getrennte Datensätze für das Training und die Validierung aus den durch die Anwendung von SUP.11 bereitgestellten Trainingsdaten des GTSRB Datensatzes zu erstellen. Innerhalb der Datei sind zunächst die violett hinterlegten Felder zu befüllen und so anzugeben, welche Trainingsiteration vorliegt und wie viele Trainings- und Validierungsdaten pro Klasse ausgewählt werden sollen. Über den Button in der Excel Datei ist anschließend ein Marko zu starten, welches zufällig, entsprechend der angegebenen Anzahl pro Klasse, Bilder auswählt und diese in die angelegte Ordnerstruktur kopiert.

Insofern systematische Probleme aufgrund der verwendeten Daten während des Trainings auffallen, können die zufällig erzeugten Datensätze entweder neu zufällig erzeugt werden oder systematisch um Daten des GTSRB Datensatzes ergänzt oder reduziert werden. Dabei ist darauf zu achten, dass der Identifikator des Datensatzes gemäß der in der Ordnerstruktur festgelegten Benamung hochgezählt wird.

5. Trainings- und Validierungsumgebung

Für das Training ist gemäß *Req_004* ein Computer mit 16GB Ram, einem Intel i7-4790K Prozessor, einer Nvidia GTX 970 Grafikkarte sowie Windows 10 als Betriebssystem zu verwenden. Auf diesem Computer ist zunächst folgende Software zu installieren:

1. Python der Version 3.8 (https://www.python.org/)
2. Git der Version 2.42.0 (https://git-scm.com/)
3. Pytorch Version 2.0.1 (https://pytorch.org/)
4. Nvidia Cuda Toolkit der Version 10 (https://developer.nvidia.com/cuda-toolkit)

Nachdem diese Installationen durchgeführt worden sind, ist das YOLOv5 Repository von https://github.com/ultralytics/yolov5 zu klonen und durch den Befehl *pip install -r requriements.txt* alle in der requirements.txt angegeben Anforderungen von YOLOv5 an die Entwicklungsumgebung zu installieren. Weiterhin ist durch den Befehl *pip install notebook* Jupyter Notebook zu installieren.

Für jede Trainingsiteration in der andere Trainings- oder Validierungsdaten oder ein anderes YOLOv5 Klassifikationsmodell verwendet werden soll, ist eine eigene wie folgt dargestellte Ordnerstruktur anzulegen.

Ordnerstruktur je Trainingsiteration, eigene Darstellung

D. Testansatz

Testansatz

1. Vorwort

Dieses Dokument entspricht der Beschreibung des Ansatzes für das Testing während der experimentellen Entwicklung innerhalb der Masterarbeit zu dem Thema „Verifikation standardisierter Referenzprozesse zur Entwicklung von ML Komponenten im Automobilkontext bezüglich Anwendbarkeit im sicherheitskritischen Kontext und Unterstützung einer möglichen Sicherheitsargumentation“.

2. Start- und Endekriterien

Folgende Start- und Endekriterien für das ML Testing sind zu berücksichtigen.

Startkriterien:
- Anforderungen an die ML Komponente sind spezifiziert, gemäß ihrer Korrektheit, technischen Machbarkeit, Testbarkeit sowie ihrem Einfluss auf die Betriebsumgebung analysiert und freigegeben.

UND
- Ein abgestimmtes ML Modell steht als trainiertes Modell für das Testing zur Verfügung.

UND
- Eine den spezifizierten Anforderungen an die Daten entsprechende Datenbasis ist verfügbar, welche gemäß definierten Kriterien qualitätsgesichert wurde.

Endekriterien:
- Die gemäß 3. erzeugten Daten für den Test wurden von dem ML Modell verarbeitet und gemäß 4. wurde eine Aussage über die Erfüllung aller, für den Test relevanten Performanz und Ressourcen Anforderungen (*Req_009, Req_010, Req_011, Req_012 und Req_013*) erzeugt.

3. Erzeugung und Modifikation des Datensatzes

Für das Testing sind zwei Datensätze, ein Absicherungs- und ein Testdatensatz, aus den Testdaten des GTSRB Datensatzes zu erzeugen, welche keine identischen Bilder enthalten dürfen (*Req_016*) aber den gesamten spezifizierten Klassenkatalog abdecken (*Req_015*).

Für die Erstellung des Absicherungsdatensatzes sind für alle 10 relevanten Klassen an Verkehrszeichen, die jeweiligen Testdaten des GTSRB Datensatzes zu betrachten und jeweils 10 Bilder pro Klasse auszuwählen, welche eine Störung durch natürliche Umwelteinflüsse wie z.B. außergewöhnlich hohe oder niedrige Belichtung, Verdeckung, Rotation etc. besitzen (*Req_023*). Die ausgewählten 100 Bilder sind anschließend als .jpg Dateien in dem Ordner Assurance_Dataset_JPG_<*ID*> abzulegen.

Für die Erstellung des Testdatensatzes sind über das VBA Skript in der Datei AuswahlTestdaten.xlsm jeweils 100 Bilder pro Klasse zufällig aus den Testdaten des GTSRB Datensatzes auszuwählen (*Req_022*) und als .jpg Datei in den Ordner Test_Dataset_JPG_<*ID*> zu kopieren. Damit für den Testdatensatz keine Bilder ausgewählt werden, welche bereits in dem Absicherungsdatensatz enthalten sind, sind diese in der Excel Datei zu markieren.

Sollten nach Verarbeitung der für den Test verwendeten Daten weitere Anpassungen des getesteten ML Modells vorgenommen werden, so sind beide Datensätze für den Test mindestens teilweise durch neue Daten abzuändern und die Identifikatoren der Datensätze gemäß der in der Ordnerstruktur festgelegten Benamung hochzuzählen.

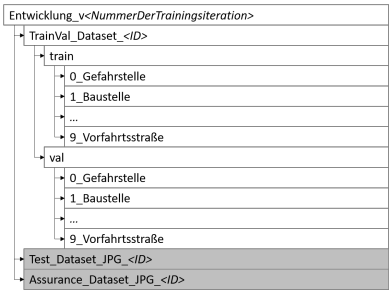
4. Testauswertung gemäß Arch_ID_02

Wie in der Architekturspezifikation als Arch_ID_02 beschrieben, ist eine .xlsm Datei anzulegen, welche die als .txt Dokument gespeicherten Vorhersagen des trainierten ML Modells, für die Daten des Test, einliest und mit der korrekten Vorhersage abgleicht, um die prozentuale Korrektheit der Vorhersagen auf den für den Test verwendeten Daten zu bestimmen. Die .xlsm Datei muss dafür eine Liste alle korrekten Klassen der Testdaten des GTSRB Datensatzes beinhalten, welche aus der Datei „GroundTruth-Test.csv“ des GTSRB Datensatzes zu entnehmen ist.

Anschließend an die Testauswertung ist ein Testreport_*<ID>* zu erstellen, welcher eine Aussage darüber enthält, ob die relevanten Performanz und Ressourcen Anforderungen (Req_009, Req_010, Req_011, Req_012, Req_013) erfüllt wurden und die falsch klassifizierten Test- und Absicherungsdaten aufführt.

5. Testumgebung

Für das Testing ist die im Trainings- und Validierungsansatz beschriebene und während des Trainings, gemäß *Req_004*, aufgesetzte Trainingsumgebung weiter zu verwenden, indem die angelegte Ordnerstruktur wie folgend abgebildet, um die zwei in grau hinterlegten Ordner ergänzt wird.



Erweiterung der Ordnerstruktur der Trainingsiteration für das Testing, eigene Darstellung

E. Testbericht

Testreport

1. Vorwort

Dieses Dokument entspricht dem Testreport zu dem Test des ML Modells aus Trainingsiteration 03 auf den Datensätzen *Test_Dataset_01* und *Assurance_Dataset_01* gemäß dem Testansatz in Version 01.

2. Anforderungserfüllung

Anforderung	Wert	Status
Req_009: Die Verarbeitung eines Bildes aus den Test- und Absicherungsdaten darf im Durchschnitt nicht mehr als 1sec benötigen.	≈81ms	Erfüllt
Req_010: Die Auswertung der Testergebnisse hinsichtlich dem Verhältnis korrekter Vorhersagen muss innerhalb von 5min möglich sein.	≈30s	Erfüllt
Req_011: Die ML-Komponente muss bei dem abschließenden Test mit dem Testdatensatz mit einer Wahrscheinlichkeit von >98% eine korrekte Vorhersage erzeugen.	99,3%	Erfüllt
Req_012: Die ML-Komponente muss bei dem Test mit dem Absicherungsdatensatz mit einer Wahrscheinlichkeit von >95% eine korrekte Vorhersage erzeugen.	98%	Erfüllt
Req_013: Die ML-Komponente darf maximal einen Speicherplatz von 100 MB in Anspruch nehmen.	≈58 MB	Erfüllt

3. Fehlerhafte Klassifikationen

Folgende Bilder des Testdatensatzes wurden falsch klassifiziert.

Dateiname	Bild	Erzeugte Vorhersage (Wahrscheinlichkeit)	Korrekte Vorhersage (Wahrscheinlichkeit)
01276.txt		8_Vorfahrt (47%)	0_Gefahrstelle (11%)
05149.txt		1_Baustelle (42%)	0_Gefahrstelle (11%)
06766.txt		8_Vorfahrt (51%)	0_Gefahrstelle (11%)
07886.txt		1_Baustelle (46%)	0_Gefahrstelle (11%)
10641.txt		5_ZulässigeHöchstgeschwindigkeit 50 (83%)	4_ZulässigeHöchstgeschwindigkeit 30 (9%)
11089.txt		1_Baustelle (53%)	0_Gefahrstelle (11%)
12616.txt		1_Baustelle (33%)	0_Gefahrstelle (11%)

Folgende Bilder des Absicherungsdatensatzes wurden falsch klassifiziert.

Dateiname	Bild	Erzeugte Vorhersage (Wahrscheinlichkeit)	Korrekte Vorhersage (Wahrscheinlichkeit)
01022.txt		1_Baustelle (72%)	8_Vorfahrt (13%)
07579.txt		4_ZulässigeHöchstgeschwindigkeit 30 (45%)	5_ZulässigeHöchstgeschwindigkeit 50 (27%)

Version 01

INTERNAL

F. Abbildung der Sicherheitsziele aus [WCACP20] auf das ASPICE 4.0 [VDA23b]

Ziel		Relevante BPs				
ID	Zieltext	MLE.1	MLE.2	MLE.3	MLE.4	SUP.11
Str9: Angemessenheit der Daten						
G9.1	Collected data sets (training/validation/test) respect {mlr}	1, 3, 4, 5	/	1, 2, 4	1, 2, 4	1, 2, 3, 4, 5
G9.2	Data is properly labeled	/	/	/	/	2, 4, 5
G9.3	Domain coverage is achieved with training, validation, and testing data sets	1, 3, 5	/	/	/	1, 2, 3, 4, 5
G9.4	Samples in training, validation and test data sets are not overlapping	/	/	1, 2, 4	1, 2, 4	/
G9.5	Data sets contain proper amount of data samples	/	/	1, 2, 4	1, 2, 4	/
G9.6	Data sets are characterized with proper distribution resembling this of an operational environment	1, 3, 5	/	1, 2, 4	1, 2, 4	/
G9.7	Gathered data accounts for safety critical scenarios	1, 3, 5	/	/	/	3
G9.8	Gathered data accounts for adversarial examples	1, 3, 5	/	/	/	3
G9.9	Gathered data accounts for corner cases	1, 3, 5	/	/	/	3
G9.10	Data sets consider different object variations, relations, influences	1, 3, 5	/	1, 2, 4	1, 2, 4	3
G9.11	Similarity of training/validation/test data sets in regards to operational data is high	1, 3, 5	/	1, 2, 4	1, 2, 4	1, 2, 3, 4, 5
G9.12	ODD coverage is analyzed	1, 3, 5	/	/	/	/
G9.13	Verify if collected data properly adheres to {mlr}	/	/	/	/	2, 5
Str10: Angemessenheit des Designs des ML-Modells						
G10.1	ML model design refines {mlr}	/	1, 6	/	/	/
G10.2	ML model design is sufficiently comprehensible, correct, robust and verifiable	/	1, 3, 6	/	/	/
G10.3	ML model design is compatible with the target hardware	1, 3, 4, 5	1, 3, 4, 5, 6	/	/	/
G10.4	Potential hazardous failures are acceptably managed by ML model design	/	1, 3, 6	/	/	/
G10.5	Verify if ML model design is comprehensible, correct, and robust	/	3	/	/	/
G10.6	Verify if ML model design properly refines {mlr}	/	3, 6	/	/	/
G10.7	ML model design introduces design-level performance metrics	/	/	/	/	/
G10.8	ML model design is optimized	/	1, 2, 3	1, 2, 3	/	/
G10.9	ML model design relies on appropriate best practices	/	/	/	/	/
G10.10	Specification of correct hyperparameters is provided	/	2, 3	/	/	/
G10.11	ML model design contributes to generalization property	/	1, 2, 3, 6	1, 2, 3, 4	1, 2, 3, 6	/
Str11: Angemessenheit der Implementierung und des Trainings des ML-Modells						
G11.1	{mlr} is implemented by the ML model	/	1, 6	1, 3	/	/
G11.2	Implementation activity follows good practices	/	/	/	/	/
G11.3	ML model is implemented in an appropriate implementation framework	/	1	1, 3	/	/
G11.4	ML model is trainined on an appropriate training platform	/	1	1, 3	/	/
G11.5	ML model is robust	/	/	1, 2, 3, 4	1, 2, 3, 6	/
G11.6	ML model satisfies {mlr} on target platform	/	/	/	1, 2, 4, 5, 6	/
G11.7	Differences between the training and target platforms do not lead to a violation of the {mlr}	/	/	/	1, 2, 4, 5, 6	/
G11.8	ML model is sufficiently well understood	/	1, 2, 3, 4	1, 2, 3, 4	1, 2, 3, 4, 5 ,6	/
G11.9	ML model is interpretable	/	1, 2, 3, 4	1, 2, 3, 4	1, 2, 3, 4, 5 ,6	/
G11.10	ML model is evaluated against predefined performance metrics	/	/	1, 3	1, 3, 5, 6	/
G11.11	ML model is verified against corner cases	/	/	1, 2, 3	1, 2, 3, 5, 6	/
G11.12	ML model is verified against adversarial attacks	/	/	1, 2, 3	1, 2, 3, 5, 6	/
G11.13	Statistical measures are performed	/	/	/	/	/
G11.14	ML model is verified against injected faults	/	/	1, 2, 3	1, 2, 3, 5, 6	/
G11.15	Generalization property of ML model is verified	/	/	/	1, 2, 3, 4, 5, 6	/