HTBL Hollabrunn

Höhere Lehranstalt für Informationstechnologie

DIPLOMARBEIT

Gesamtprojekt

TrafficSignDetection

Verkehrszeichenerkennungs-App

Markus Brandstetter 5BHITS

Betreuer: Dipl. -Ing. Gerald Zottl

Schuljahr 2024/25

Abgabevermerk:

Datum: 07.04.2025 übernommen von: Gerald Zottl

Höhere Technische Bundeslehranstalt Hollabrunn
Höhere Lehranstalt für Informationstechnologie

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Markus Brandstetter

Hollabrunn, am 7. April 2025

HINWEISE

Die vorliegende Diplomarbeit wurde für die Abteilung Informationstechnologie der HTL Hollabrunn ausgeführt.

Die in dieser Diplomarbeit entwickelten Prototypen und Software-Produkte dürfen ganz oder auch in Teilen von Privatpersonen oder Firmen nur dann in Verkehr gebracht werden, wenn sie diese selbst geprüft und für den vorgesehenen Verwendungszweck für geeignet befunden haben.
Es wird keinerlei Haftung übernommen für irgendwelche Schäden, die aus der Nutzung der hier entwickelten oder beschriebenen Bestandteile des Projekts resultieren.

Für alle Entwicklungen gilt die GNU General Public License [http://www.gnu.org/licenses/gpl.html] der Free Software Foundation, Boston, USA in der Version 3.

Absatz überarbeiten bzw. Rundschreiben aktualisieren!
Die Diplomarbeit erfüllt die “Standards für Ingenieur- und Technikerprojekte” entsprechend dem Rundschreiben Nr. 60 aus 1999 des BMBWK (GZ.17.600/101-II/2b/99).
[https://www.bmb.gv.at/ministerium/rs/1999_60.html]

SCHLÜSSELBEGRIFFE

Künstliche Intelligenz, Fahrerassistenzsysteme, Automatisierte Fahrsysteme, Verkehrszeichenerkennung, Aufmerksamkeitsmechanismus

DANKSAGUNGEN

Danke an Herr Professor Zottl für die technische und theoretische Expertise und Unterstützung. Ein besonderes Danke geht auch an meine Eltern, welche mich während der Diplomarbeit und der Ausbildung unterstützt haben und diese auch ermöglicht haben.

htl bildung mit zukunft	**HÖHERE TECHNISCHE BUNDESLEHRANSTALT HOLLABRUNN**
	1 Fachrichtung: Informationstechnologie

DIPLOMARBEIT
DOKUMENTATION

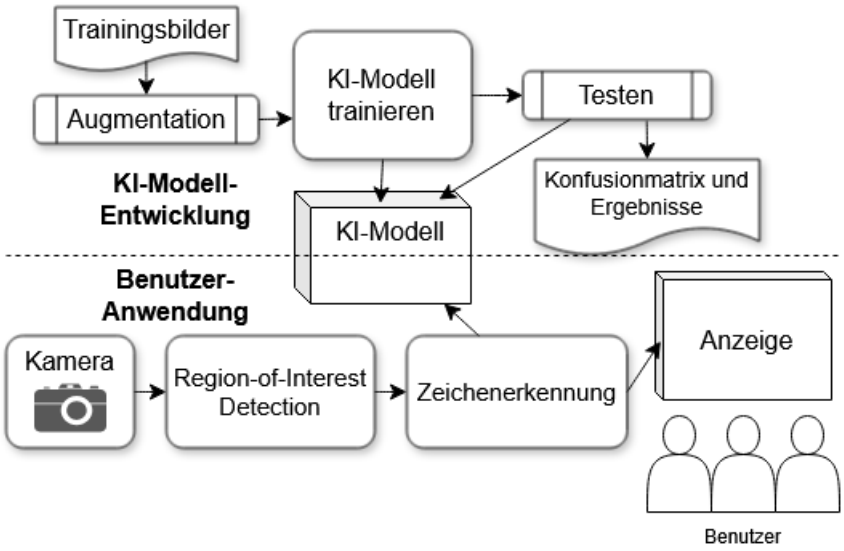
Namen der Verfasser/innen	Markus Brandstetter
Jahrgang Schuljahr	5BHITS
Thema der Diplomarbeit	TrafficSignDetection – Verkehrszeichenerkennungs-App
Kooperationspartner	Gerald Zottl

Aufgabenstellung	**Es soll ein System zur Erkennung von Verkehrszeichen in Echtzeit entwickelt werden. Aus den Videodaten sollen relevanten Verkehrszeichen erfasst, betitelt und verarbeitet werden.** • Nutzung von Bildverarbeitung und maschinellem Lernen • Interpretation der erkannten Verkehrszeichen • Schwerpunkt auf Geschwindigkeitsbegrenzungen • Klare Darstellung der gültigen Geschwindigkeitsbegrenzung • Fokus auf Übersichtlichkeit und minimaler Ablenkung des Fahrers

Realisierung	Zuerst wurden Bilder aus dem Internet und Bildausschnitte aus selbst erstellten Bildern und Videos gesammelt, um damit das Modell zu trainieren und zu testen. In den einzelnen Bildern eines Videos bzw. der Kamera werden dann zunächst mögliche Tafeln als geometrische Formen erkannt. Nur diese interessanten Bildausschnitte werden dann der KI zur Erkennung übergeben. Eine App-Konzept wurde erarbeitet, aber nicht durchgeführt.

Ergebnisse	Das System erkennt Verkehrszeichen in Echtzeit mit hoher Genauigkeit und stellt insbesondere die aktuelle Geschwindigkeitsbegrenzung übersichtlich dar. Es reduziert Fahrerablenkung und ist eine vielversprechende Basis für fortschrittliche Fahrerassistenzsysteme.

htl bildung mit zukunft	**HÖHERE TECHNISCHE BUNDESLEHRANSTALT HOLLABRUNN**
	1 Fachrichtung: Informationstechnologie

Typische Grafik, Foto etc. (mit Erläuterung)	 *Die technische Übersichtsgrafik zeigt den Prozess der KI-Modell-Entwicklung, bei dem ein Datensatz durch Augmentation erweitert, ein Modell trainiert, gespeichert und getestet wird. In der Benutzer-Anwendung erfasst die Kamera zunächst Bilddaten, die durch die Region-of-Interest Objekterkennung weiterverarbeitet und über die Zeichenerkennung mit dem gespeicherten Modell zur Erkennung von Verkehrszeichen übermittelt werden, wobei die erkannten Zeichen dem Benutzer ausgegeben werden.*

Teilnahme an Wettbewerben, Auszeichnungen	

Möglichkeiten der Einsichtnahme in die Arbeit	HTL Hollabrunn Anton Ehrenfriedstraße 10 2020 Hollabrunn	
Approbation (Datum / Unterschrift)	Prüfer/Prüferin	Direktor/Direktorin Abteilungsvorstand/Abteilungsvorständin

htl bildung mit zukunft	**HÖHERE TECHNISCHE BUNDESLEHRANSTALT HOLLABRUNN** **COLLEGE of ENGINEERING**
	2 Department: Information Technology

DIPLOMA THESIS
Documentation

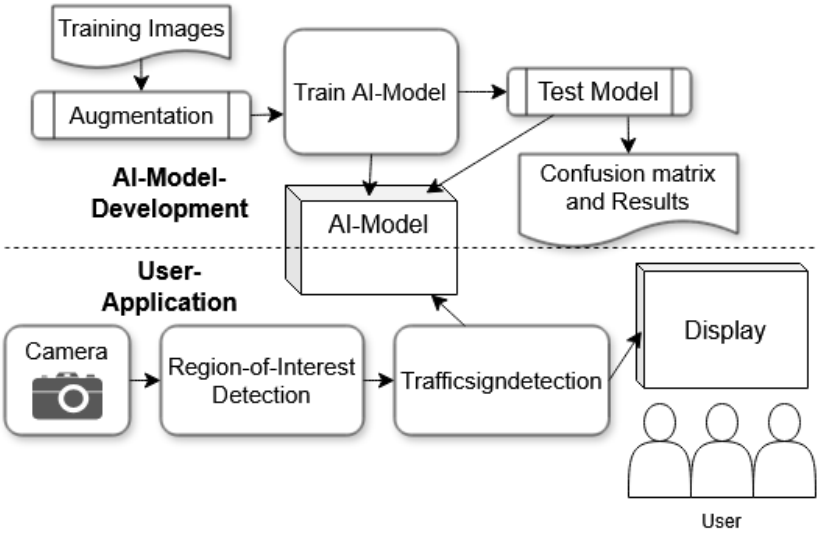
Author(s)	Markus Brandstetter
Form Academic year	5BHITS
Topic	TrafficSignDetection
Co-operation partners	Gerald Zottl

Assignment of tasks	**A system for recognizing traffic signs in real time is to be developed. Relevant traffic signs are to be recorded, titled and processed from the video data.** • Use of computer vision and machine learning • Interpretation of recognized traffic signs • User-friendly display of recognized signs • Clear display of the applicable speed limit • Focus on clarity and minimal driver distraction

Realisation	First, images from the internet and image sections from self-created images and videos were collected to train and test the model. In the individual frames of a video or camera, possible panels are first recognized as geometric shapes. Only these interesting image sections are then passed to the AI for recognition. An app concept was developed but not implemented.

Results	The system detects speed limits in real time with high accuracy and displays them clearly. It reduces driver distraction and is a promising basis for advanced driver assistance systems.

htl bildung mit zukunft	**HÖHERE TECHNISCHE BUNDESLEHRANSTALT HOLLABRUNN** **COLLEGE of ENGINEERING**
	2 Department: Information Technology

Illustrative graph, photo (incl. explanation)	 *The technical overview graphic shows the process of AI model development, in which a dataset is extended through augmentation, a model is trained, stored and tested. In the user application, the camera first captures image data, which is further processed by region-of-interest object recognition and transmitted via character recognition with the stored model for the recognition of traffic signs, whereby the recognized characters are output to the user.*

Participation in competitions Awards	

Accessibility of final project thesis	HTL Hollabrunn Anton Ehrenfriedstraße 10 2020 Hollabrunn

Approval (Date / Signature)	Examiner/s	Head of Department / College

Berufsbildende Schulen

Erklärung

Die unterfertigten Kandidaten/Kandidatinnen haben gemäß den geltenden schulrechtlichen Bestimmungen die Ausarbeitung einer abschließenden Arbeit (Diplomarbeit bzw. Abschlussarbeit) mit folgender Aufgabenstellung gewählt:

TrafficSignDetection

Individuelle Aufgabenstellungen im Rahmen des Gesamtprojektes:

- Markus Brandstetter (5BHITS): **Verkehrszeichenerkennungs-App**

Die Kandidaten/Kandidatinnen nehmen zur Kenntnis, dass die abschließende Arbeit in eigenständiger Weise und außerhalb des Unterrichtes zu bearbeiten und anzufertigen ist, wobei Ergebnisse des Unterrichtes mit einbezogen werden können, die jedenfalls als solche entsprechend kenntlich zu machen sind.

Die Abgabe der vollständigen abschließenden Arbeit hat in digitaler und in zweifach ausgedruckter Form bis spätestens **07.04.2025** beim zuständigen Betreuer/der zuständigen Betreuerin zu erfolgen.

Die Kandidaten/Kandidatinnen nehmen auch zur Kenntnis, dass ein Abbruch der abschließenden Arbeit nicht möglich ist.

Kandidaten/Kandidatinnen:	**Datum und Unterschrift bzw. Handysignatur:**
Markus Brandstetter (5BHITS)	Markus Brandstetter 27.09.2024

Inhaltsverzeichnis

1 Einleitung 12
1.1 Motivation für die Arbeit 12
1.1.1 Warum Verkehrszeichenerkennung wichtig ist 12
1.1.2 Einsatz von Künstlicher Intelligenz (KI) zur Bildverarbeitung 12
1.2 Problemstellung und Zielsetzung 12
1.2.1 Probleme bei der Verkehrszeichenerkennung 12
1.2.2 Ziel der Arbeit 12
1.3 Technologische Relevanz 12
1.4 Abgrenzung und Umfang der Arbeit 12
1.5 User Stories 13
1.6 Logodesign, Marktanalyse und Wettbewerbsanalyse 13
1.6.1 Logodesign 13
1.6.2 Marktanalyse 13
1.6.3 Wettbewerbsanalyse 14
2 Grundlagen und Stand der Technik 14
2.1 Einsatz von Deep Learning und CNNs 14
2.2 Relevante KI-Modelle und Algorithmen 14
2.2.1 Convolutional Neural Networks (CNNs) 14
2.2.2 Keras als Framework für Deep Learning 15
2.3 Vergleich bestehender Ansätze zur Verkehrszeichenerkennung 15
2.4 Herausforderungen und offene Fragen 15
3 Projektplanung und Systemarchitektur 16
3.1 Definition der Projektstruktur 16
3.1.1 Verwendung von Git als Versionskontrolle 16
3.1.2 Aufteilung der Softwarekomponenten 16
3.2 Technologie-Stack und eingesetzte Werkzeuge 17
3.2.1 Visual Studio Code – Entwicklungsumgebung 17
3.2.2 Python & Keras/TensorFlow – Deep Learning Framework 17
3.2.3 JavaScript & React Native – Frontend-Entwicklung 17
3.2.4 Pandas, NumPy, Matplotlib, Excel – Datenanalyse & Visualisierung 17
3.3 Aufbau der Softwarekomponenten 18
3.4 Fahrtenaufnahmen und Kamera 19
3.4.1 Herausforderungen bei der Videoaufnahme 19
4 KI-Modell-Entwicklung 20
4.1 Datenerfassung und Vorverarbeitung 20
4.1.1 Auswahl und Verwendung von Trainingsdatensätzen 20
4.1.2 Datenaugmentation 24
4.1.3 Vorbereitung der Daten 24
4.2 Entwicklung des KI-Modells 25
4.2.1 Architektur des CNN-Modells 25
4.2.2 Das KI-Modell 25
4.2.3 Trainingsprozess und Optimierung 26

5 Benutzeranwendung 27
5.1 Integration der Kamera-Funktionalität 27
5.1.1 Überlappende Region of Interest Rechtecke 28
5.2 Region of Interest Detection 29
5.3 KI-Modellanwendung 30
5.4 Echtzeit-Verkehrszeichenerkennung 32
5.5 Anzeige der Ergebnisse der KI 32
5.6 App-Konzept 34
5.6.1 Cross-Platform-App-Konzept 34
5.6.2 Herausforderungen und Probleme 34
6 Testphase und Auswertung 34
6.1 Erste Tests und Debugging-Prozesse 34
6.2 Modell-Validierung und Performance-Analyse 35
6.3 Verbesserungspotenziale und Optimierungsstrategien 35
7 Ergebnisse und Diskussion 36
7.1 Zusammenfassung der wichtigsten Ergebnisse 36
7.1.1 Die Genauigkeit des KI-Modells 36
7.1.2 Echtzeit-Verkehrszeichenerkennung 39
7.1.3 100 zufällige Testbilder 38
7.1.4 Ausschnitte aus den aufgenommenen Fahrten 38
7.2 Interpretation der Testergebnisse 50
7.3 Vergleich mit bestehenden Lösungen 51
7.3.1 Stärken unseres Ansatzes 52
7.3.2 Schwächen 52
8 Fazit und Ausblick 52
8.1 Erreichte Ziele und Projekterfolge 52
8.2 Weitere Zielsetzungen 52
8.3 Herausforderungen und Lösungsansätze während der Umsetzung 53
8.4 Zukünftige Weiterentwicklungen 53
9 Quellen 54
10 Verzeichnis der Abbildungen 54
11 Verzeichnis der Code-Listings 55
12 Begleitprotokoll 55

1 Einleitung

1.1 Motivation für die Arbeit

1.1.1 Warum Verkehrszeichenerkennung wichtig ist

Die Verkehrszeichenerkennung ist ein zentraler Bestandteil autonomer Fahrzeuge und der Verkehrsüberwachung. Sie ermöglicht es Fahrzeugen, ihre Umgebung zu verstehen und sicher auf unterschiedliche Verkehrsbedingungen zu reagieren. Eine zuverlässige Erkennung der Verkehrszeichen ist entscheidend für die Fahrsicherheit und die Automatisierung des Straßenverkehrs.

1.1.2 Einsatz von Künstlicher Intelligenz (KI) zur Bildverarbeitung

KI, insbesondere Deep Learning, hat in den letzten Jahren enorme Fortschritte gemacht und wird zunehmend für die Bildverarbeitung eingesetzt. Durch den Einsatz von Convolutional Neural Networks (CNNs) können Bilder von Verkehrszeichen effizient und präzise analysiert und interpretiert werden.

1.2 Problemstellung und Zielsetzung

1.2.1 Probleme bei der Verkehrszeichenerkennung

Die Verkehrszeichenerkennung steht vor mehreren Herausforderungen, wie z.B. wechselnden Lichtverhältnissen, unterschiedlichen Wetterbedingungen, Verschmutzung, verschiedenen Formen und Größen der Schilder und einer großen Variabilität der Verkehrszeichen. Zudem kann die Qualität der Aufnahmen je nach Kamera und Umgebung variieren. Diese Faktoren sollen mittels Data Augmentation beim Modell-Training berücksichtigt werden. Außerdem ist es wichtig die relevanten Stellen herauszufinden in einem Bild, wo Verkehrszeichen sind, um diese dann für das Modell aufbereitet an das Modell zu übergeben.

1.2.2 Ziel der Arbeit

Das Ziel dieser Arbeit ist die Entwicklung eines robusten KI-Modells, das in der Lage ist, Verkehrszeichen zuverlässig und in Echtzeit zu erkennen, unabhängig von den Umgebungsbedingungen oder der Qualität der Bildaufnahme.

1.3 Technologische Relevanz

In modernen Technologien, insbesondere im Bereich autonomes Fahren, Smart Cities und der Verkehrsüberwachung, ist die Verkehrszeichenerkennung ein Schlüsselthema. Sie ermöglicht eine effiziente und sichere Verkehrssteuerung, hilft Unfälle zu vermeiden und trägt zur Schaffung einer intelligenten Infrastruktur bei.

1.4 Abgrenzung und Umfang der Arbeit

In dieser Arbeit liegt der Fokus auf der Erkennung von Verkehrszeichen mithilfe eines vordefinierten Datensatzes (z.B. GTSRB (German Traffic Sign Recognition Benchmark)). Es

wird bei der Anwendung auf real-time Object Detection Frameworks wie z.B. YOLO (You Only Look Once) oder anderen vortrainierten Modells wie EfficientNetB0 verzichtet, um eine detailliertere und spezifischere Betrachtung der Verkehrszeichenerkennung mit klassischen eigenen Einstellungen eines Deep-Learning-Prozesses zu ermöglichen.

1.5 User Stories

- Als Benutzer möchte ich, dass die Anwendung in Echtzeit arbeitet und, dass Verkehrszeichen erfasst und erkannt werden.
- Als Benutzer möchte ich die Bedeutung des erkannten Verkehrszeichens angezeigt bekommen, einschließlich aktueller Geschwindigkeitsbegrenzungen, Ortstafeln, Richtungen oder anderen relevanten Informationen.
- Als Benutzer möchte ich, dass die Anwendung auch offline funktioniert, damit ich sie auch in Gebieten ohne Internetverbindung nutzen kann.
- Als Benutzer möchte ich, dass die Anwendung intuitiv und benutzerfreundlich ist.
- Als Entwickler möchte ich Videos von Fahrten hochladen können, welche dann analysiert werden können.

1.6 Logodesign, Marktanalyse und Wettbewerbsanalyse

1.6.1 Logodesign

Das Logodesign wurde zunächst bei den ersten Schritten der Erstellung der Diplomarbeit designt. Sie repräsentiert die Diplomarbeit und ist auch als Logografik im Code eingebaut.

Abbildung 1: Logo

Die grafischen Komponenten sollen einfach in schlichter und simpler Gestaltung einen grafischen Logoanblick darstellen, wobei man eine Art Kamerasymbol hineininterpretieren kann. Der Text gibt einfach den Namen der Diplomarbeit „Trafficsigndetection“ mit der Endung „.ai“ an, falls das Projekt über eine passende Domain veröffentlicht wird.

1.6.2 Marktanalyse

Künstliche Intelligenz und Automatisierung steigen in Nachfrage exponentiell. So auch die Nachfrage nach Automatisierungssoftware, mittels Apps auch im Straßenverkehr. Dabei handelt es sich jedoch um integrierte Systeme im Fahrzeug, eine Android-App für die LIVE-Erkennung von Verkehrszeichen und Geschwindigkeitsbegrenzungen wäre am Markt noch nicht in großen Stil umgesetzt, und hätte somit Potenzial für Big Data Collection, Datenanalyse und Verkehrstracking im flächendeckenden Anwenderbereich. Für die

Datensammlung müsste aber jedoch auch der Datenschutz berücksichtigt werden. Die häufigsten Methoden zurzeit sind Radarerkennung und eingetragene Verkehrszeichen in Datenbanken und Verkehrskarten.

1.6.3 Wettbewerbsanalyse

Ein führender Anbieter von Bilderkennungstechnologie im Straßenverkehr ist Mobileye mit „Mobileye Supervision“ eine Tochterfirma von Intel, welcher in dieser Diplomarbeit auch für Performancevergleichswerte herangenommen wurde. Ähnliche Anwendungen für künstliche Intelligenz in der Mobilitätsbranche werden in Unternehmen wie Continental oder Aptiv, welche Fahrassistenz und Entfernungssensoren verwenden, angewendet. Eine mobile Echtzeit-Verkehrszeichenerkennung gibt es zurzeit noch nicht und wäre somit am Markt neu. Es gibt jedoch schon bereits Verkehrszeichenerkennungsprogramme, welche in der Lage sind, auch mittels KI, Verkehrszeichen zu erkennen.

2 Grundlagen und Stand der Technik

2.1 Einsatz von Deep Learning und CNNs

Deep Learning-Algorithmen, insbesondere Convolutional Neural Networks (CNNs), haben in den letzten Jahren die klassische Bildverarbeitung weitgehend ersetzt. Frühe Ansätze zur Verkehrszeichenerkennung basierten auf klassischen Bildverarbeitungsmethoden wie Kanten- und Farbsegmentierung. Sie bieten eine viel höhere Erkennungsgenauigkeit und Flexibilität bei der Verarbeitung von Bilddaten.

2.2 Relevante KI-Modelle und Algorithmen

2.2.1 Convolutional Neural Networks (CNNs)

CNNs sind besonders gut für die Analyse von Bildern geeignet, da sie in der Lage sind, hierarchische Merkmale aus den Bilddaten zu extrahieren und zu verarbeiten. Sie bestehen aus mehreren Schichten, die eine detaillierte Analyse der Bildinhalte ermöglichen.

Die folgende Grafik zeigen das Grundprinzip von CNNs:

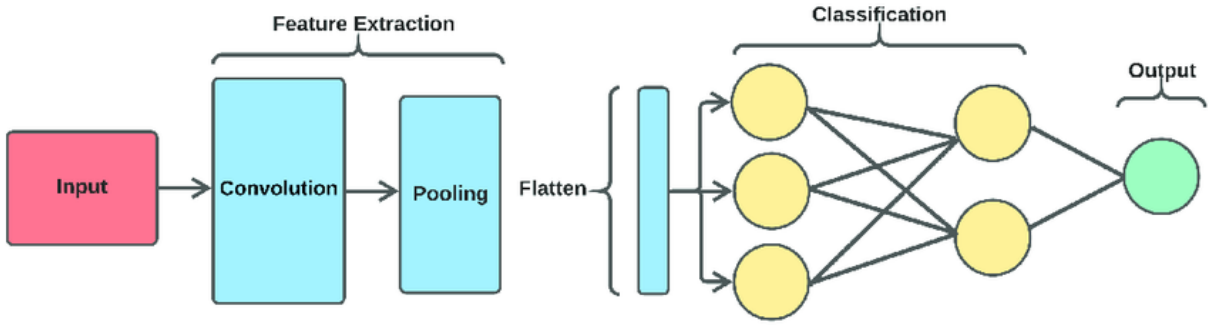


Abbildung 2: Das Grundprinzip von Convolutional Neural Networks

2.2.2 Keras als Framework für Deep Learning

Keras bietet eine einfache API zur Erstellung und zum Training von Deep Learning-Modellen. Die Integration mit TensorFlow ermöglicht eine effiziente Verarbeitung großer Datensätze und eine schnelle Modelloptimierung.

2.3 Vergleich bestehender Ansätze zur Verkehrszeichenerkennung

Klassische Verfahren, die auf Regeln, Heuristiken und fixen Algorithmen, welche man selbst erweitern müsste, basieren, stoßen bei der Verkehrszeichenerkennung an ihre Grenzen, insbesondere in komplexen Umgebungen. Deep Learning-Methoden, wie hier bei TrafficSignDetection eingesetzte CNN-Modelle bieten hier eine viel größere Flexibilität und Robustheit, da sie aus Daten lernen und nicht explizit programmiert werden müssen. Dabei ist es natürlich wichtig die Daten zu klassifizieren (labeln).

2.4 Herausforderungen und offene Fragen

Eine der größten Herausforderungen bei der Verkehrszeichenerkennung ist die Robustheit des Modells gegenüber wechselnden Umgebungsbedingungen. Außerdem war es schwierig aus eigen aufgenommenen Daten einen vollständigen, umfangreichen Datensatz zu generieren und es konnte nur ein gewisser Teil der benötigten Datenmenge abgedeckt werden. In Zukunft könnte die Erkennung von Verkehrszeichen weiter verbessert werden, etwa durch einen internationalen oder europäischen Datensatz mit sehr viel höherer Variation, einfachere Klassifizierungsregeln in der Region-of-Interest Detection um gleich vor der tatsächlich Prediction in Farbkategorie und Formkategorie zu unterteilen und die Integration von Multimodalität (Kombination von Bild- und Sensordaten (LIDAR, Entfernungssensoren)) und fortschrittlicheren Deep Learning-Architekturen, mit besser verständlicheren und auch für Menschen verstehbaren Layern.

3 Projektplanung und Systemarchitektur

3.1 Definition der Projektstruktur

3.1.1 Verwendung von Git als Versionskontrolle

Git (Gitea vom HTL Server) wird zur Versionskontrolle und Codeverwaltung eingesetzt, um eine einfache Zusammenarbeit im Team mit dem Projektleiter zu ermöglichen und die Codebasis laufend zu verbessern.

3.1.2 Aufteilung der Softwarekomponenten

Die folgende Grafik stellt die Systemarchitektur in einer technischen Übersichtsgrafik dar:

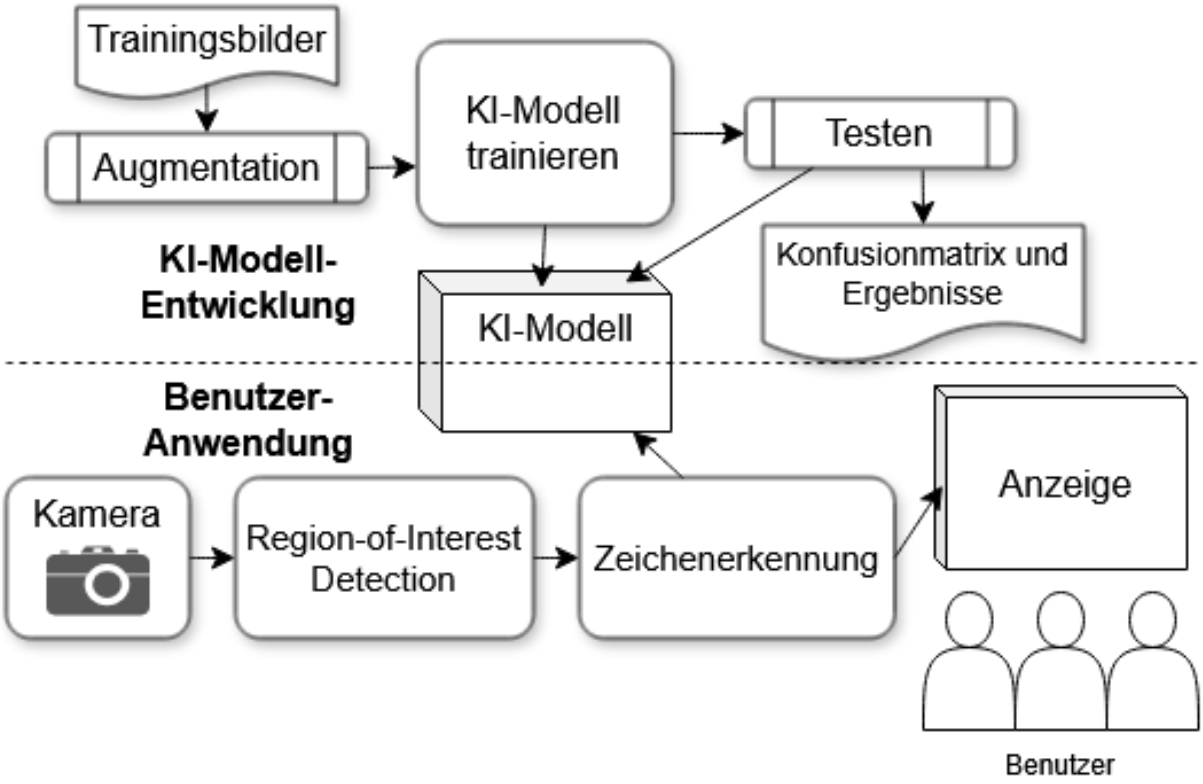


Abbildung: Technische Übersichtsgrafik

Das System wird in dieser Grafik in verschiedene Bereiche unterteilt, wobei man zunächst zwischen KI-Modell-Entwicklung und Benutzer-Anwendung unterscheidet. Beide verwenden als Schlüsselkomponente das KI-Modell, welches bei der Entwicklung das KI-Modell erstellt, trainiert und gespeichert wird. Die Testergebnisse werden im Trainingsprozess immer wieder laufend berücksichtigt und integriert und auch für den Entwickler mittels einer Konfusionmatrix visualisiert. Bei der Benutzer-Anwendung werden über den Kamerazugriff Bilddaten abgerufen, welche mittels der Region-of-Interest Objekt Erkennung weiter an die Zeichenerkennung mit dem trainierten KI-Modell gegeben werden, um die erkannten Verkehrszeichen dem Benutzer anzuzeigen.

3.2 Technologie-Stack und eingesetzte Werkzeuge

3.2.1 Visual Studio Code – Entwicklungsumgebung

Visual Studio Code bietet eine funktionsreiche, benutzerfreundliche und einfache Entwicklungsumgebung, die besonders gut für Python und JavaScript geeignet ist, mit welcher diese Diplomarbeit umgesetzt wurde.

3.2.2 Python & Keras/TensorFlow – Deep Learning Framework

TensorFlow und Keras sind Aufgrund der Flexibilität und breiten Unterstützung perfekt für die Verkehrszeichenerkennung geeignet und bieten sehr gute und selbst einstellbare, customizable Einstellungen der Modellarchitektur an. Sie bieten eine einfache und effiziente Infrastruktur für die Modellierung, den Training- und Validation-Split und das Training des KI-Modells.

3.2.3 JavaScript & React Native – Frontend-Entwicklung

Das Expo Framework, welches auf React Native basiert, wird verwendet, um eine plattformübergreifende Anwendung zu entwickeln, die sowohl für Android als auch für iOS funktioniert. Erste Ansätze zu dieser Cross-Platform-App wurden konzipiert, aber nicht voll funktionsfähig fertiggestellt.

3.2.4 Pandas, NumPy, Matplotlib, Excel – Datenanalyse & Visualisierung

Diese Python Libraries werden verwendet, um die gesammelten Daten zu analysieren und die Ergebnisse der Modelltests zu visualisieren. Excel wird für die Speicherung, Analyse, Berechnung, und Auswertung der Modelltraining-Durchläufe verwendet.

3.3 Aufbau der Softwarekomponenten

Dabei kann man wie die Softwarestruktur in zwei Bereiche unterteilen, zunächst das Training des KI-Modells mit verschiedenen KI-Architekturen und die Benutzeranwendung mit verschiedenen Ausgaben und Benutzeroberflächen.

Die folgende Grafik stellt die Softwarestruktur im Projektordner dar:

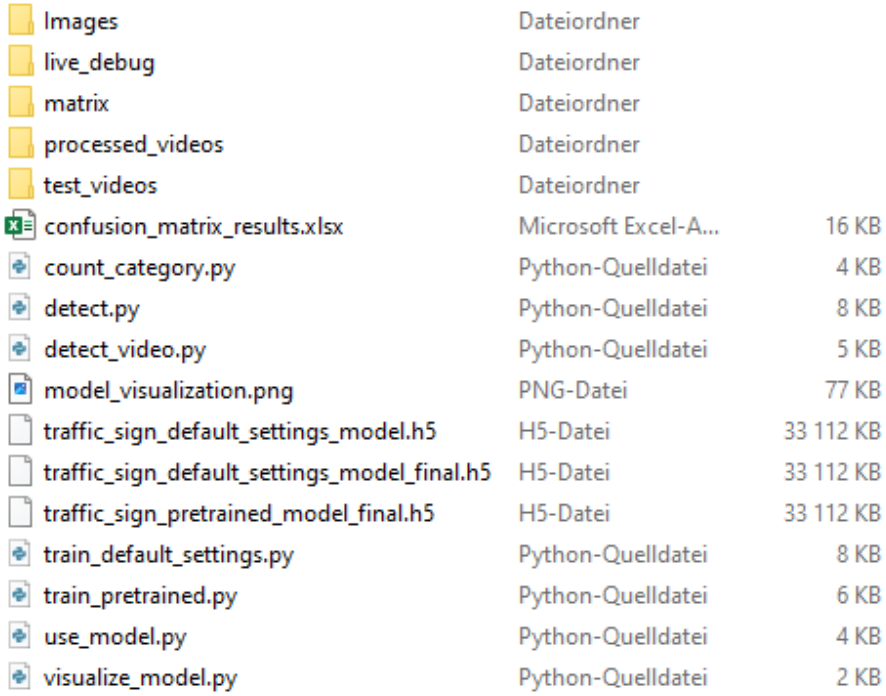


Abbildung 3: Projektstruktur

Für die einzelnen Programme wurde Python verwendet. Die trainierten KI-Modelle werden im .H5-Format gespeichert. Ergebnisse werden als Exceldatei und im Ordner „matrix“ abgelegt. Im Ordner „Images“ finden sich die Daten aus dem GTSRB (German Traffic Sign Recognition Benchmark) Datensatz. Im Ordner „test_videos“ werden die Autofahrtaufnahmen gespeichert und für KI-Modelltests verwendet. Die einzelnen Python-Dateien sind jeweils für das Training und die Anwendung des KI-Modells.

3.4 Fahrtenaufnahmen und Kamera

Für die Aufnahme der Autofahrten wurden Videos mithilfe einer Handy-Kamera Haltung aufgenommen.

Die folgende Grafik zeigt den Aufbau der Handyhalterung in der Verwendung:

Abbildung 4: Handyhalterung

3.4.1 Herausforderungen bei der Videoaufnahme

Bei der Aufnahme ist es wichtig das gesamte Fahrgeschehen aufzuzeichnen. Hierbei gab es jedoch Probleme, zunächst ganz grundsätzlich mit der Halterung des Smartphones, weil diese gegen die Fahrbahnunebenheiten nicht robust war und es deswegen während der Videoaufnahme zum Ruckeln kam. Außerdem hatte die Handykamera manchmal Probleme mit dem Verhältnis, dem Licht, dem Zoomfaktor und dem Autofokus, was auch zu Problemen während der Autofahrt führte.

4 KI-Modell-Entwicklung

4.1 Datenerfassung und Vorverarbeitung

4.1.1 Auswahl und Verwendung von Trainingsdatensätzen

Der GTSRB (German Traffic Sign Recognition Benchmark) Datensatz wird verwendet, um das Modell zu trainieren. Der Datensatz enthält eine Vielzahl von Verkehrszeichen in unterschiedlichen Umgebungen, Lichtverhältnissen und Perspektiven. Im GTSRB-Datensatz werden 43 verschiedene Verkehrszeichenklassen kategorisiert.

Die Struktur des Datensatzes in Python:

```
# ====================
# 1. Labels definieren
# ====================

class_labels = {
    0: "Geschwindigkeitsbegrenzung (20km/h)",
    1: "Geschwindigkeitsbegrenzung (30km/h)",
    2: "Geschwindigkeitsbegrenzung (50km/h)",
    3: "Geschwindigkeitsbegrenzung (60km/h)",
    4: "Geschwindigkeitsbegrenzung (70km/h)",
    5: "Geschwindigkeitsbegrenzung (80km/h)",
    6: "Ende der Geschwindigkeitsbegrenzung (80km/h)",
    7: "Geschwindigkeitsbegrenzung (100km/h)",
    8: "Geschwindigkeitsbegrenzung (120km/h)",
    9: "Überholverbot",
    10: "Überholverbot für Fahrzeuge über 3,5t",
    11: "Vorfahrt an der nächsten Kreuzung",
    12: "Vorfahrtstraße",
    13: "Vorfahrt gewähren",
    14: "Stop",
    15: "Verbot für Fahrzeuge aller Art",
    16: "Verbot für Fahrzeuge über 3,5t",
    17: "Einfahrt verboten",
    18: "Allgemeine Gefahrstelle",
    19: "Kurve nach links",
    20: "Kurve nach rechts",
    21: "Doppelkurve",
    22: "Unebene Fahrbahn",
    23: "Schleudergefahr bei Nässe oder Schmutz",
    24: "Verengte Fahrbahn",
    25: "Baustelle",
    26: "Ampel",
    27: "Fußgänger",
    28: "Kinder",
    29: "Radfahrer",
    30: "Achtung Schnee oder Eisglätte",
    31: "Wildwechsel",
    32: "Ende aller Geschwindigkeits- und Überholverbote",
    33: "Rechts abbiegen",
    34: "Links abbiegen",
    35: "Geradeaus fahren",
    36: "Geradeaus oder rechts abbiegen",
    37: "Geradeaus oder links abbiegen",
```

```
44.     38: "Rechts vorbei",
45.     39: "Links vorbei",
46.     40: "Kreisverkehr",
47.     41: "Ende des Überholverbots",
48.     42: "Ende des Überholverbots für Fahrzeuge über 3,5t",
49. }
50.
51. def interpret_sign(class_index):
52.     return class_labels.get(class_index, "Unbekanntes Schild")
53.
```

Code-Listing 1: Datensatzstruktur des GTSRB in Python

Hierbei ist zu beachten, dass die Datenmenge unterschiedlich verteilt ist.

Die folgende Grafik stellt die Verteilung der Datenmenge pro Kategorie dar:

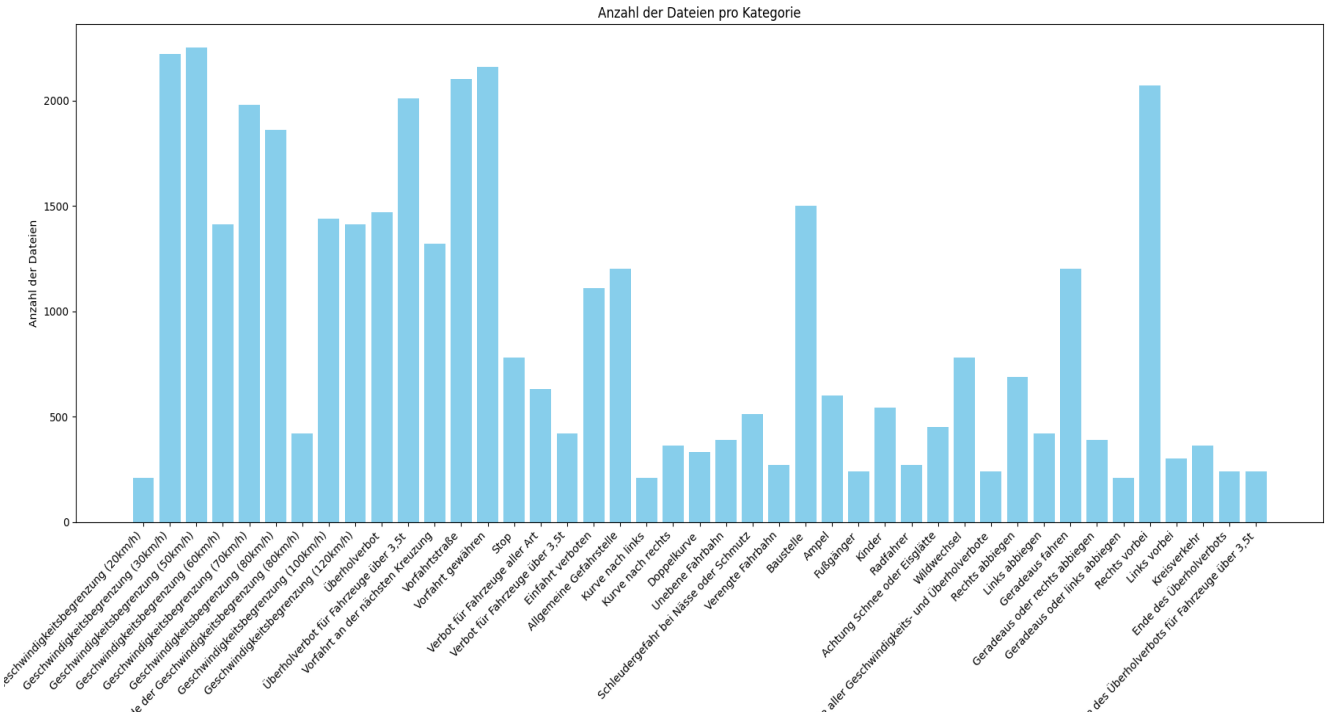


Abbildung 5: Verteilung der Datenmenge pro Kategorie (ungeordnet)

Hier wird die Verteilung noch einmal nach der Anzahl der Dateien pro Kategorie der Größe nach geordnet dargestellt:

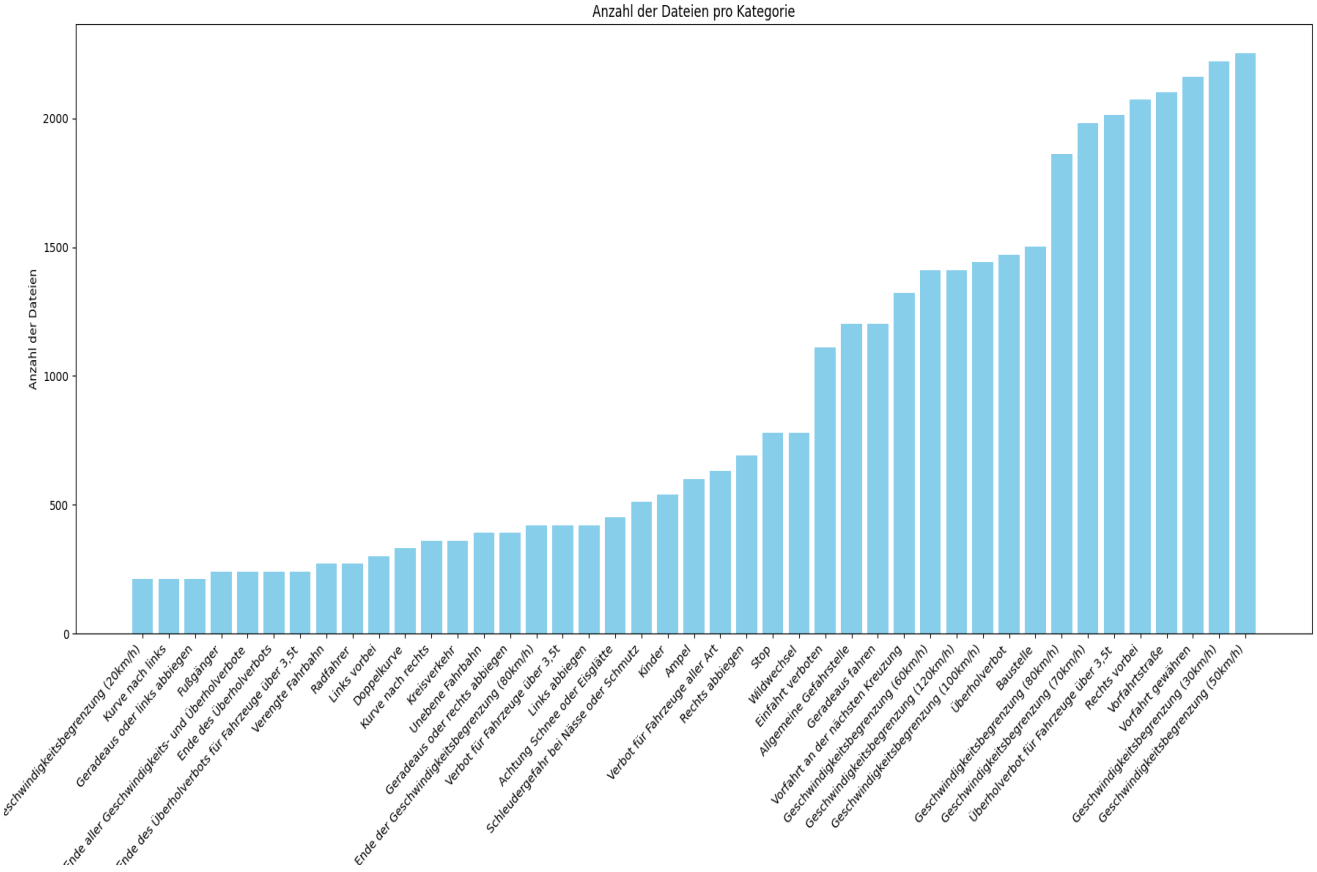


Abbildung 6: Verteilung der Datenmenge pro Kategorie (geordnet)

Diese Verteilungsanalyse ist hilfreich beim Modelltraining und der Auswertung der Modelltestergebnisse, weil sich daraus auch unteranderem erschließen lässt, dass Verkehrszeichenkategorien, die unterbewertet sind, nur einen geringeren Pool aus Trainingsbilder pro Kategorie zur Verfügung hat, was eigene Effekte auf die Genauigkeit, Belastbarkeit und Verallgemeinerbarkeit des Modells haben kann.

Eine ungleichmäßige Verteilung der Trainingsdaten kann dazu führen, dass das Modell bestimmte Kategorien schlechter erkennt, da es weniger Beispiele zum Lernen hat. Dies kann insbesondere bei seltenen Verkehrszeichen problematisch sein, da das Modell möglicherweise nicht robust genug ist, um sie zuverlässig zu klassifizieren.

Eine sorgfältige Analyse der Datenverteilung ist daher wichtig, um mögliche Verzerrungen zu erkennen und gegebenenfalls durch Datenaugmentation oder eine gezieltere Datensammlung auszugleichen. Deswegen wurde auch experimentiert, grundsätzlich eine maximale Bilderanzahl pro Kategorie zu verwenden, um die Verteilung und Ungleichheiten anzupassen. Dies wurde jedoch im finalen Modell und der Anwendung nicht berücksichtigt.

Folgender Code ermöglicht die Datenmenge zuerst im Ordner zu zählen und danach mittels Matplotlib in Python zu visualisieren:

```
 1. def count_files_in_subfolders(directory):
 2.     category_counts = {label: 0 for label in class_labels.values()}
 3.
 4.     for category in os.listdir(directory):
 5.         if category.isdigit() and int(category) in class_labels:
 6.             category_name = class_labels[int(category)]
 7.             category_path = os.path.join(directory, category)
 8.             if os.path.isdir(category_path):
 9.                 num_files = len([f for f in os.listdir(category_path) if
os.path.isfile(os.path.join(category_path, f))])
10.                 category_counts[category_name] = num_files
11.
12.     return category_counts
13.
14. def plot_category_counts(category_counts, sorted_order=True):
15.     categories = list(category_counts.keys())
16      counts = list(category_counts.values())
17.
18.     if sorted_order:
19.         sorted_indices = sorted(range(len(counts)), key=lambda i:
counts[i])
20.         categories = [categories[i] for i in sorted_indices]
21.         counts = [counts[i] for i in sorted_indices]
22.
23.     plt.figure(figsize=(10, 5))
24.     plt.bar(categories, counts, color='skyblue')
25.     plt.xlabel('Kategorien')
26.     plt.ylabel('Anzahl der Dateien')
27.     plt.xticks(rotation=45, ha='right')
28.     plt.title('Anzahl der Dateien pro Kategorie')
29.     plt.show()
30.
31. if __name__ == "__main__":
32.     directory = "./Images"  # Ordner anpassen
33.     category_counts = count_files_in_subfolders(directory)
34.
35.     # 1. Sortierte Darstellung
36.     plot_category_counts(category_counts, sorted_order=True)
37.
38.     # 2. Unsortierte Darstellung
39.     plot_category_counts(category_counts, sorted_order=False)
40.
```

Code-Listing 2: Datensatzvisualisierung

Die Funktion count_files_in_subfolders durchsucht ein Verzeichnis mit numerisch benannten Unterordnern (entsprechend den Klassenlabels) und zählt die darin enthaltenen Bilddateien pro Kategorie. Sie gibt ein Dictionary mit den Klassennamen als Schlüssel und den jeweiligen Dateianzahlen als Werte zurück. Die Funktion plot_category_counts visualisiert diese Verteilung als Balkendiagramm, wobei die Option sorted_order die Balken nach Größe sortiert anzeigt. Das Hauptprogramm ruft beide Funktionen auf - erst für eine sortierte, dann für eine unsortierte Darstellung der Datenverteilung.

4.1.2 Datenaugmentation

Durch Datenaugmentation wird die Robustheit des Modells verbessert (Rotation, Zoom, Flip), indem verschiedene Varianten der Bilddaten generiert werden.

Folgender Code ermöglicht die Datenaugmentation in Python:

```
# =====================
# 3. Datenaugmentation
# =====================
data_augmentation = tf.keras.Sequential(
    [
        tf.keras.layers.experimental.preprocessing.RandomFlip(
             "horizontal", input_shape=(224, 224, 3)
        ),
        tf.keras.layers.experimental.preprocessing.RandomRotation(0.1),
        tf.keras.layers.experimental.preprocessing.RandomZoom(0.2),
    ]
)

```

Code-Listing 3: Data-Augmentation in Python

4.1.3 Vorbereitung der Daten

Die Bilder werden vor dem Modelltraining skaliert und normalisiert, um die Performance des Modells zu optimieren.

Folgender Code ermöglicht die Skalierung und Normalisierung der Daten im Trainingsprozess in Python:

```
# =====================
# 2. Dataset laden
# =====================
dataset_path = r"Images"
img_size = (224, 224)

# Train/Test-Split aus dem Verzeichnis
train_ds = image_dataset_from_directory(
    dataset_path, validation_split=0.2, subset="training", seed=123,
image_size=img_size
)
val_ds = image_dataset_from_directory(
    dataset_path,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=img_size,
)

```

Code-Listing 4: Skalierung und Normalisierung

4.2 Entwicklung des KI-Modells

4.2.1 Architektur des CNN-Modells

Das verwendete CNN-Modell basiert auf mehreren Convolutional und Pooling-Schichten, um hierarchische Merkmale zu extrahieren und schließlich die Verkehrszeichen zu klassifizieren.

Die folgende Grafik stellt die Modellarchitektur des CNNs grafisch dar:

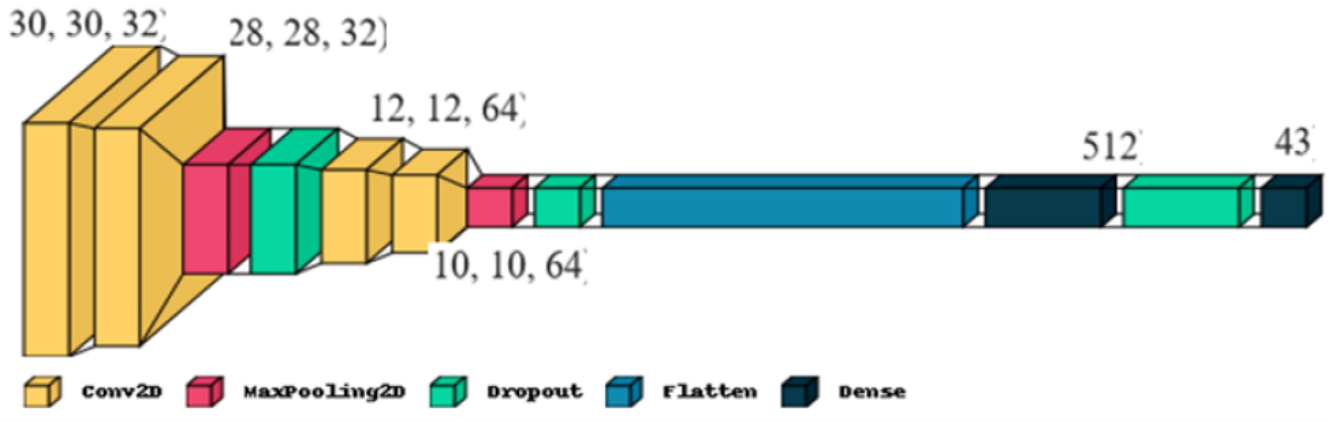


Abbildung 7: grafische Darstellung der Modellarchitektur

Die Architektur beginnt mit mehreren Convolutional-Layern (Conv2D), die mit Filtergrößen wie 30x30 (32 Filter) und 12x12 (64 Filter) Merkmale aus den Eingabebildern extrahieren. Darauf folgen MaxPooling-Layer, die durch Downsampling die räumlichen Dimensionen reduzieren und gleichzeitig die wichtigsten Merkmale erhalten. Ein Dropout-Layer verhindert Overfitting, indem er zufällig Neuronen während des Trainings deaktiviert. Die Flatten-Schicht wandelt die mehrdimensionalen Features in einen Vektor um, den abschließende Dense-Layer (vollvernetzte Schichten) für die Klassifikation nutzen - wobei die letzte Schicht mit 43 Neuronen der Anzahl der Verkehrszeichenklassen entspricht. Diese schrittweise Verarbeitung von groben zu feinen Merkmalen ermöglicht eine robuste Erkennung der 43 verschiedenen Verkehrszeichen.

4.2.2 Das KI-Modell

Implementation vom CNN-KI-Modell in Python:

```
# =====================
# 4. Modell erstellen
# =====================
model = Sequential(
    [
        data_augmentation,
        Rescaling(1.0 / 255),
        Conv2D(128, (3, 3), activation="relu"),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation="relu"),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation="relu"),
```

```
13.         MaxPooling2D((2, 2)),
14.         Conv2D(256, (3, 3), activation="relu"),
15.         MaxPooling2D((2, 2)),
16.         Flatten(),
17.         Dense(64, activation="relu"),
18.         Dropout(0.2),
19.         Dense(128, activation="relu"),
20.         Dense(len(class_labels), activation="softmax"),
21.     ]
22. )
23.
24. model.summary()
25.
```

Code-Listing 5: KI-Modelle in Python mittels Keras und Tensorflow

4.2.3 Trainingsprozess und Optimierung

Das Modell wird mit einer Vielzahl von Hyperparametern trainiert, und EarlyStopping wird verwendet, um Überanpassung zu vermeiden.

Folgender Code ermöglicht EarlyStopping und Trainingsparameter:

```
 1. # =====================
 2. # 5. Modell kompilieren
 3. # =====================
 4. model.compile(
 5.     loss=tf.keras.losses.SparseCategoricalCrossentropy(),
 6.     optimizer="adam",
 7.     metrics=["accuracy"],
 8. )
10.
11. # =====================
12. # 6. Modell trainieren
13. # =====================
14. callbacks = [EarlyStopping(monitor="val_loss", patience=5)]
15. history = model.fit(train_ds, validation_data=val_ds, epochs=10,
callbacks=callbacks)
16.
17.
18.
19. # =====================
20. # 7. Modell speichern
21. # =====================
22. model.save("traffic_sign_default_settings_model.h5")
23. print("Modell gespeichert.")
24.
```

Code-Listing 6: EarlyStopping und Trainingsparameter

Der Code konfiguriert den Trainingsprozess des KI-Modells durch die Kompilierung mit der Sparse Categorical Crossentropy Loss-Funktion und dem Adam-Optimizer, wobei die Genauigkeit (accuracy) als Metrik verfolgt wird. Das Training erfolgt über maximal 10 Epochen mit einem EarlyStopping-Callback, das den Prozess automatisch abbricht, wenn

der Validierungsverlust (val_loss) sich über 5 Epochen nicht verbessert, wodurch Overfitting verhindert wird. Abschließend wird das trainierte Modell im HDF5-Format gespeichert, um es für spätere Inferenzaufgaben verfügbar zu machen. Diese Implementierung gewährleistet ein effizientes Training mit automatischer Optimierung der Stoppbedingung.

5 Benutzeranwendung

5.1 Integration der Kamera-Funktionalität

Die Kamera wird in der Anwendung eingebunden, um kontinuierlich die Umgebung zu erfassen. Die Echtzeit-Bilddaten werden zuerst mittels der Grafikverarbeitung gefiltert und dann werden die Daten an das KI-Modell gesendet, welches die Verkehrszeichen analysiert und entsprechende Informationen anzeigt.

Folgender Code ermöglicht Kamera-Funktionalität in Python:

```
cap = cv2.VideoCapture(0)
cv2.namedWindow("Traffic Sign Detection", cv2.WINDOW_NORMAL)

# ----------------------------------#
# Debug mode aktivieren/deaktivieren #
# ----------------------------------#
debug_mode = True

# ----------------------------------#
# Video Capture Application          #
# ----------------------------------#
while True:
    ret, frame = cap.read()
    if not ret:
        break

    traffic_signs = find_traffic_signs(frame, model, debug_mode)

    for x, y, w, h in traffic_signs:
        roi = frame[y : y + h, x : x + w]

        predicted_class, confidence = predict_sign(roi, model, debug_mode)

        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(
            frame,
            f"{predicted_class}: {confidence*100:.2f}%",
            (x, y - 10),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.7,
            (255, 255, 255),
            2,
        )

    # Fenstergröße abrufen
    height, width = frame.shape[:2]
    win_width = cv2.getWindowImageRect("Traffic Sign Detection")[2]
```

```
38.     win_height = cv2.getWindowImageRect("Traffic Sign Detection")[3]
39.
40.     # Frame auf Fenstergröße skalieren
41.     resized_frame = cv2.resize(frame, (win_width, win_height))
42.
43.     cv2.imshow("Traffic Sign Detection", resized_frame)
44.
45.     if cv2.waitKey(1) & 0xFF == ord("q"):
46.         break
47.
48. cap.release()
49. cv2.destroyAllWindows()
50.
```

Code-Listing 7: Kamera-Funktionalität in Python

Der Code implementiert eine Live-Erkennung von Verkehrszeichen durch Kameraaufnahmen, wobei er zunächst den Videostream der Webcam (Device 0) initialisiert und ein anpassbares Anzeigefenster erstellt. In einer kontinuierlichen Schleife werden einzelne Frames verarbeitet, wobei die Funktion find_traffic_signs() potenzielle Verkehrszeichen lokalisiert und predict_sign() diese mittels des trainierten KI-Modells klassifiziert. Das System ermöglicht Debugging durch ROI-Speicherung, skaliert die Ausgabe dynamisch an die Fenstergröße und kann durch Drücken der Taste "q" beendet werden.

5.1.1 Überlappende Region of Interest Rechtecke

Eine dazugehörige Grafikverarbeitung für die Bilddaten der Kamera in der Region of Interest Detection wird in Python mittels der Non-Maximum Suppression (NMS) realisiert, damit überlappende erkannte Region of Interest Rechtecke reduziert werden können.

Folgender Code ermöglicht Grafikverarbeitung in Python:

```
 1.  # Non-Maximum Suppression (NMS) - Überlappende Rechtecke reduzieren
 2. def non_maximum_suppression(boxes, confidences, overlap_thresh=0.3):
 3.     if len(boxes) == 0:
 4.         return []
 5.
 6.     boxes = np.array(boxes)
 7.     confidences = np.array(confidences)
 8.
 9.     x1 = boxes[:, 0]
10.     y1 = boxes[:, 1]
11.     x2 = boxes[:, 0] + boxes[:, 2]
12.     y2 = boxes[:, 1] + boxes[:, 3]
13.
14.     areas = (x2 - x1 + 1) * (y2 - y1 + 1)
15.
16.     idxs = np.argsort(confidences)
17.
18.     selected_boxes = []
19.     while len(idxs) > 0:
20.         i = idxs[-1]
21.         selected_boxes.append(boxes[i])
22.
```

```
23.         xx1 = np.maximum(x1[i], x1[idxs[:-1]])
24.         yy1 = np.maximum(y1[i], y1[idxs[:-1]])
25.         xx2 = np.minimum(x2[i], x2[idxs[:-1]])
26.         yy2 = np.minimum(y2[i], y2[idxs[:-1]])
27.
28.         w = np.maximum(0, xx2 - xx1 + 1)
29.         h = np.maximum(0, yy2 - yy1 + 1)
30.         overlap_area = w * h
31.
32.         iou = overlap_area / (areas[i] + areas[idxs[:-1]] - overlap_area)
33.
34.         idxs = idxs[np.where(iou <= overlap_thresh)[0]]
35.
36.     return selected_boxes
37.
```

Code-Listing 8: Grafikverarbeitung in Python

Die Funktion non_maximum_suppression() filtert überlappende Bounding-Boxen basierend auf ihren Konfidenzwerten und berechnet deren Intersection-over-Union (IoU), um nur die Boxen mit der höchsten Konfidenz beizubehalten, wenn sie einen vordefinierten Überlappungsschwellenwert überschreiten, was redundante Detektionen eliminiert und die präziseste Auswahl von Verkehrszeichen-Positionen sicherstellt.

5.2 Region of Interest Detection

Die Region of Interest Detection wird in der Anwendung eingebunden, um kontinuierlich die Verkehrszeichen im Frame als interessante Region im Bild zu erfassen. Die Echtzeit-Bilddaten werden mit dem Canny-Kanten-Algorithmus und Graustufen analysiert, danach werden die Grafikdaten mittels der Grafikverarbeitung gefiltert und dann werden die Daten an das KI-Modell gesendet, welches die Verkehrszeichen klassifizieren soll.

Folgender Code ermöglicht Region of Interest Detection in Python:

```
 1. def find_traffic_signs(frame, model, debug_mode=False):
 2.     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
 3.     blurred = cv2.GaussianBlur(gray, (5, 5), 0)
 4.
 5.     edges = cv2.Canny(blurred, 100, 200)
 6.
 7.     contours, _ = cv2.findContours(edges, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
 8.
 9.     boxes = []
10.     confidences = []
11.
12.     for contour in contours:
13.         if cv2.contourArea(contour) < 1000:
14.             continue
15.
16.         x, y, w, h = cv2.boundingRect(contour)
17.         aspect_ratio = w / float(h)
18.
```

```
19.         if aspect_ratio > 0.5 and aspect_ratio < 2.0:
20.             approx = cv2.approxPolyDP(
21.                 contour, 0.02 * cv2.arcLength(contour, True), True
22.             )
23.             num_vertices = len(approx)
24.
25.             if num_vertices == 4 or num_vertices > 4:
26.                 roi = frame[y : y + h, x : x + w]
27.
28.                 predicted_class, confidence = predict_sign(roi, model,
debug_mode)
29.
30.                 boxes.append((x, y, w, h))
31.                 confidences.append(confidence)
32.
33.     final_boxes = non_maximum_suppression(boxes, confidences,
overlap_thresh=0.3)
34.
35.     return final_boxes
36.
```

Code-Listing 9: Region of Interest Detection

Die Funktion find_traffic_signs() analysiert ein Kamerabild, um potenzielle Verkehrszeichen zu lokalisieren, indem sie das Bild in Graustufen konvertiert, mit dem Canny-Algorithmus Kanten detektiert und Konturen mit geeignetem Seitenverhältnis (z.B. rechteckig/quadratisch) filtert. Für jede passende Kontur wird die ROI an das KI-Modell übergeben, um die Klassifikationskonfidenz zu prüfen, und mittels Non-Maximum Suppression überlappende Rechtecke zu reduzieren, sodass nur die relevantesten Funde zurückgegeben werden.

5.3 KI-Modellanwendung

Für die Umsetzung der KI-Modellanwendung wurde eine eigene Funktion geschrieben, die das Bildmaterial richtig aufbereitet und ans Modell weitergibt. Anfangs gab es Probleme mit der Datenübergabe und dadurch wurden falsche Ergebnisse ausgegeben, was durch die Konvertierung der Bilddaten gelöst werden konnte. Die Modellanwendung läuft jetzt stabil und ist der zentrale Teil der Verkehrszeichenerkennung im Live-System.

Folgender Code ermöglicht KI-Modellanwendung in Python:

```
 1.  def predict_sign(roi, model, debug_mode=False):
 2.      # ROI-Bild in ein PIL-Image (Pillow-Image-Library) umwandeln um keine
Buffer zu verwenden
 3.      roi_pil = Image.fromarray(cv2.cvtColor(roi, cv2.COLOR_BGR2RGB))
 4.
 5.      # Bild in einen Speicherpuffer schreiben (statt auf die Festplatte zu
speichern)
 6.      img_buffer = io.BytesIO()
 7.      roi_pil.save(img_buffer, format="PNG") # PNG, um keine Verluste zu
haben
 8.      img_buffer.seek(0) # Speicherpuffer an den Anfang setzen
 9.
```

```
10.     # Bild mit keras `load_img` laden, genau wie in use_model.py
11.     img = tf.keras.preprocessing.image.load_img(img_buffer,
target_size=(224, 224))
12.
13.     # In ein numpy-Array umwandeln
14.     img_array = tf.keras.preprocessing.image.img_to_array(img)
15.
16.     # Umwandlung in Batch-Dimension (Modell erwartet einen Batch)
17.     img_array = np.expand_dims(img_array, axis=0)
18.
19.     # Vorhersage des Modells
20.     preds = model.predict(img_array)
21.
22.     # Index der Klasse mit der höchsten Wahrscheinlichkeit
23.     class_idx = np.argmax(preds, axis=1)[0]
24.     confidence = preds[0][class_idx]
25.
26.     if debug_mode:
27.         print(f"Predicted Class: {class_labels[class_idx]}")
28.         print(f"Confidence: {confidence:.4f}")
29.         print(f"Shape of input image: {roi.shape}")
30.         print(f"ROI (cropped image) saved to 'live_debug/' with
timestamp.")
31.
32.         # Bild speichern mit einem Zeitstempel
33.         if not os.path.exists("live_debug"):
34.             os.makedirs("live_debug")
35.
36.         timestamp = time.strftime("%Y%m%d-%H%M%S")
37.         file_name =
f"live_debug/{timestamp}_{class_labels[class_idx]}.jpg"
38.         cv2.imwrite(file_name, roi)  # Bild speichern
39.
40.     return interpret_sign(class_idx), confidence
41.
```

Code-Listing 10: KI-Modellanwendung in Python

Die Funktion predict_sign() klassifiziert eine Bildregion (ROI) mit einem trainierten KI-Modell, indem sie das Bild zunächst in ein RGB-Format konvertiert, als PNG im Speicher puffert und auf 224×224 Pixel skaliert, um es batch-kompatibel für das Modell vorzubereiten. Sie nutzt model.predict(), um die Klasse mit der höchsten Wahrscheinlichkeit und deren Konfidenzwert zu ermitteln, gibt im Debug-Modus detaillierte Infos (wie vorhergesagte Klasse, Konfidenz und ROI-Speicherung mit Zeitstempel) aus und liefert schließlich die interpretierte Klassenzuordnung (z.B. "Stop-Schild") samt Konfidenz zurück, was die Echtzeit-Erkennung von Verkehrszeichen ermöglicht.

5.4 Echtzeit-Verkehrszeichenerkennung

Die Anwendung verarbeitet in Echtzeit den Video-Stream der Kamera, und erkennt Verkehrszeichen.

Folgender Code ermöglicht Echtzeit-Verkehrszeichenerkennung in Python:

```
 1.  while True:
 2.      ret, frame = cap.read()
 3.      if not ret:
 4.          break
 5.
 6.      traffic_signs = find_traffic_signs(frame, model, debug_mode)
 7.
 8.      for x, y, w, h in traffic_signs:
 9.          roi = frame[y : y + h, x : x + w]
10.
11.          predicted_class, confidence = predict_sign(roi, model, debug_mode)
12.
13.          cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
14.          cv2.putText(
15.              frame,
16.              f"{predicted_class}: {confidence*100:.2f}%",
17.              (x, y - 10),
18.              cv2.FONT_HERSHEY_SIMPLEX,
19.              0.7,
20.              (255, 255, 255),
21.              2,
22.          )
23.
```

Code-Listing 11: Echtzeit-Verkehrszeichenerkennung in Python

Der Echtzeit-Erkennungscode liest kontinuierlich Kamerabilder ein, nutzt find_traffic_signs() zur ROI-Detektion und predict_sign() zur Klassifikation, markiert erkannte Verkehrszeichen mit grünen Rechtecken und beschriftet sie mit Klasse und Konfidenz (z.B. "50 km/h: 96.2%"), was eine visuelle Live-Ausgabe der Ergebnisse ermöglicht.

5.5 Anzeige der Ergebnisse der KI

Die Ergebnisse des KI-Modells werden grafisch angezeigt. Es werden die wichtigsten Kennzahlen wie Confidence in Prozent und erkannte Verkehrszeichen präsentiert.

Folgender Code ermöglicht die Anzeige der Ergebnisse in Python:

```
1. # =====================
2. # 8. Trainingsergebnisse visualisieren
3. # =====================
4. # Verlust (Loss) anzeigen
5. plt.plot(history.history["loss"], label="Train Loss")
6. plt.plot(history.history["val_loss"], label="Validation Loss")
7.
```

```
 8. plt.legend()
 9. plt.show()
10.
11. # Genauigkeit (Accuracy) anzeigen
12. plt.plot(history.history["accuracy"], label="Train Accuracy")
13. plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
14.
15. plt.legend()
16. plt.show()
17.
18. # =====================
19. # 9. Konfusionsmatrix berechnen
20. # =====================
21. def generate_confusion_matrix(model, val_ds):
22.     # Wahrheitswerte und Vorhersagen sammeln
23.     true_labels = []
24.     predictions = []
25.
26.     for images, labels in val_ds:
27.         # Vorhersagen des Modells
28.         preds = model.predict(images)
29.         predicted_class = np.argmax(preds, axis=1)
30.
31.         # Labels der Validierungsbilder
32.         true_labels.extend(labels.numpy())
33.         predictions.extend(predicted_class)
34.
35.     # Konfusionsmatrix berechnen
36.     cm = confusion_matrix(true_labels, predictions)
37.
38.     # Genauigkeit pro Kategorie berechnen
39.     accuracy_per_class = cm.diagonal() / cm.sum(axis=1)
40.
41.     # Speichern der Konfusionsmatrix und der Genauigkeit in eine Excel-
Datei
42.     df_cm = pd.DataFrame(
43.         cm, index=list(class_labels.values()),
columns=list(class_labels.values())
44.     )
45.     df_accuracy = pd.DataFrame(
46.         {"Category": list(class_labels.values()), "Accuracy":
accuracy_per_class}
47.     )
48.
49.     # Excel-Datei speichern
50.     with pd.ExcelWriter("confusion_matrix_results.xlsx") as writer:
51.         df_cm.to_excel(writer, sheet_name="Confusion Matrix")
52.         df_accuracy.to_excel(writer, sheet_name="Accuracy per Category")
53.
54.     # Konfusionsmatrix visualisieren
55.     plt.figure(figsize=(10, 8))
56.     sns.heatmap(
57.         cm,
58.         annot=True,
59.         fmt="d",
60.         cmap="Blues",
61.         xticklabels=list(class_labels.values()),
62.         yticklabels=list(class_labels.values()),
```

```
63.     )
64.     plt.xlabel("Vorhergesagte Labels")
65.     plt.ylabel("Wahre Labels")
66.     plt.title("Konfusionsmatrix")
67.     plt.show()
68.
69. # Konfusionsmatrix anzeigen und Excel-Datei speichern
70. generate_confusion_matrix(model, val_ds)
71.
```

Code-Listing 12: Anzeige der Ergebnisse in Python

Dieser Python-Code dient zur Auswertung des trainierten KI-Modells. Zuerst werden der Trainingsverlauf (Loss und Accuracy) für Trainings- und Validierungsdaten als Kurvendiagramme visualisiert, um die Modellperformance zu analysieren. Anschließend wird eine Konfusionsmatrix erstellt, die zeigt, wie häufig welche Klassen korrekt oder falsch klassifiziert wurden. Die Matrix wird sowohl als farbige Heatmap dargestellt, als auch in eine Excel-Datei exportiert. Dies ermöglicht eine detaillierte Bewertung der Erkennungsgenauigkeit des Modells, insbesondere für die Verkehrszeichenklassifikation.

5.6 App-Konzept

5.6.1 Cross-Platform-App-Konzept

Erste Ansätze zu dieser Cross-Platform-App wurden konzipiert, aber nicht voll funktionsfähig fertiggestellt. Es wurde ein Expo Projekt angelegt welches im Browser, als auch auf iOS und Android Geräten, welche mit Android Studio als virtuelle Devices erstellt wurden, grundsätzlich funktioniert. Die Kamerafunktion funktionierte bereits, ein Region of Interest Detection Algorithmus konnte in Javascript jedoch noch nicht umgesetzt werden, wofür Python hingegen deutlich besser geeignet war.

5.6.2 Herausforderungen und Probleme

Um eine vollwertige Implementation in Expo zu realisieren, müsste eine gleichwertige Region of Interest Integration in Javascript programmiert werden. Da Javascript eigentlich nur eine Frontend-Skriptsprache ist, ist die Umsetzung eines Region of Interest Algorithmus in Python deutlich einfacher, und konnte in Javascript noch nicht umgesetzt werden. Außerdem verlangt Javascript das fertige Modell im Format json, welches grundsätzlich mit tensorflowjs konvertiert werden kann, aber da die Region of Interest Integration fehlt, fehlen dem Modell die benötigten grafischen Inputdaten.

6 Testphase und Auswertung

6.1 Erste Tests und Debugging-Prozesse

Erste Tests wurden durchgeführt, um die Funktionsweise der Kamera und die Qualität der Bildaufnahmen in verschiedenen Umgebungen zu überprüfen und auszuwerten. Fehler und unerwartete Ergebnisse wurden während des Debugging-Prozesses identifiziert und

behoben. Die Testergebnisse wurden für die Weiterentwicklung des Modells und des Modelltrainings kontinuierlich verwendet.

6.2 Modell-Validierung und Performance-Analyse

Zur Validierung des Modells werden die üblichen Performance-Metriken wie Genauigkeit, Präzision und Recall verwendet. Eine Confusion-Matrix hilft dabei, die Klassifizierungsfehler des Modells zu analysieren.

Die folgende Grafik stellt die Ergebnisse mittels einer Confusion-Matrix grafisch dar:

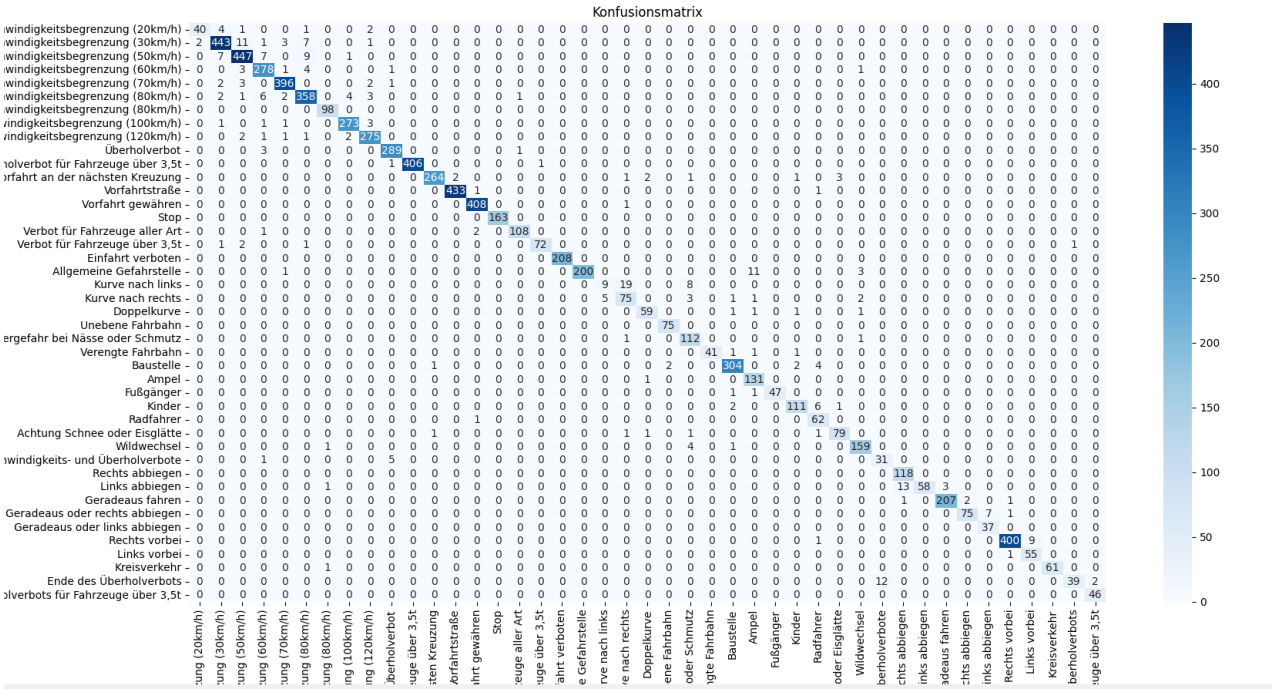


Abbildung 8: grafische Darstellung mittels Konfusionmatrix

Hierbei lässt sich auch wieder die ungleiche Verteilung der Daten widerspiegeln, weil einige Klassen im Train/Validation Split deutlich überrepräsentiert sind (in der dunkelblau). Außerdem lassen sich vereinzelt bemerkbare Abweichungen feststellen. Im besten Fall sollte es sich um eine farbige diagonale Gerade handeln komplett ohne Abweichungen. Dieser Ergebnis-Run wurde letztendlich für die praktische Demonstration und die Weiterverwendung herangenommen, also wurde mit dieser Genauigkeit weitergearbeitet.

6.3 Verbesserungspotenziale und Optimierungsstrategien

Es wurden verschiedene Strategien zur Modelloptimierung und -verbesserung untersucht, wie z.B. eine erweiterte Datenaugmentation, die Anpassung der Architektur mit anderen Schichtenmodellen oder die Einbindung zusätzlicher Modelleinstellungen zur Verbesserung der Genauigkeit der Erkennung. Dafür wurde auch die Epochenanzahl beim Training auf 20 Epochen erhöht. Dies führte zu einer höheren Trainingszeit, aber auch zu besseren und genaueren Ergebnissen. Es wurde auch testweise auf ein Pretrained-Modell (vortrainiertes Modell) umgestiegen, um zu testen ob Pretrained-Modelle bessere Ergebnisse liefern.

7 Ergebnisse und Diskussion

7.1 Zusammenfassung der wichtigsten Ergebnisse

Die Ergebnisse des KI-Modells werden in den folgenden Schritten zusammengefasst. Die wichtigsten Kennzahlen wie Genauigkeit, Zeit zur Erkennung und die Anzahl der korrekt erkannten Verkehrszeichen werden hier präsentiert.

7.1.1 Die Genauigkeit des KI-Modells

Die folgende Grafik stellt die Genauigkeit des KI-Modells beim Trainings- und Validationsprozess grafisch dar:

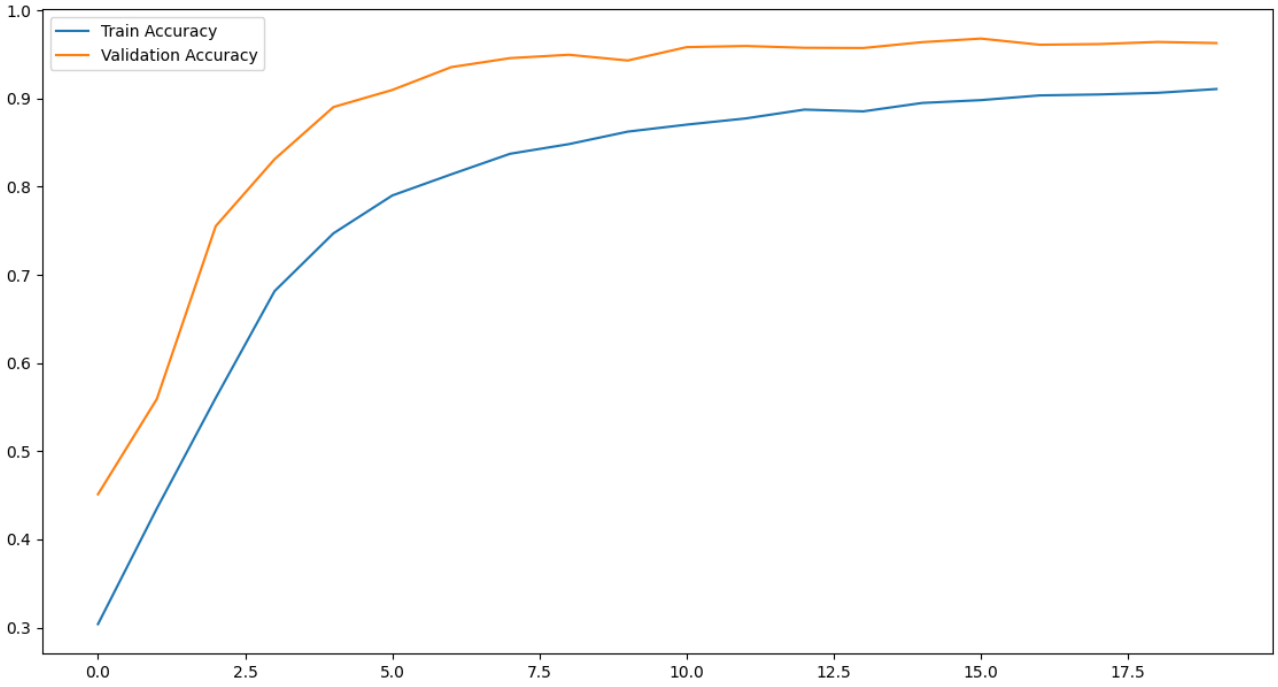


Abbildung 9: grafische Darstellung der Modellgenauigkeit

Die folgende Grafik stellt den Trainingsverlust und den Validierungsverlust beim Trainings- und Validationsprozess grafisch dar:

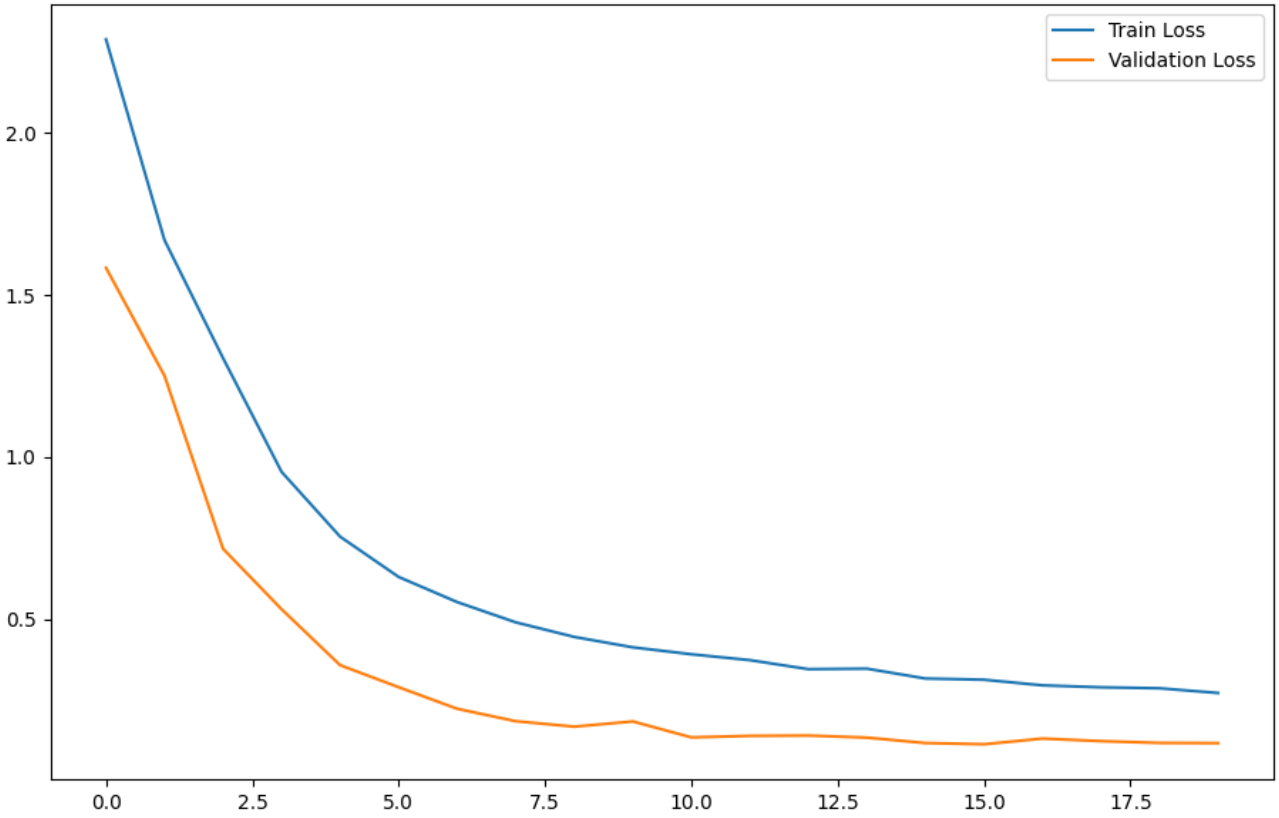


Abbildung 10: grafische Darstellung von Train- und Validationloss

7.1.1.1 Erkenntnisse

Idealerweise sollte sowohl der Trainingsverlust und er Validierungsverlust im Verlauf des Trainings sinken. Hierbei lassen sich auch Effekte wie Overfitting oder Underfitting beschreiben, wobei man die Entwicklung des Validierungsverlust und des Trainingsverlust betrachtet.

7.1.2 100 zufällige Testbilder

Die folgende Grafik zeigt die Ergebnisse der Erkennung des KI-Modells für 100 Testbilder aus dem Datensatz:

Abbildung 11: 100 Testbilder

Dabei ist Truth (Wahrheit) immer die tatsächliche Kategorie des Bildes, und Pred (Prediction, Vorhersage) die Kategorie, zu welcher Kategorie die trainierte KI das Bild zuordnen würde.

In diesem Durchlauf lässt sich erkennen, dass das KI-Modell von den 100 Bildern 95 Bilder richtig erkannt, und 5 Bilder falsch erkannt hat.

7.1.3 Ausschnitte aus den aufgenommenen Fahrten

Im Zuge der Diplomarbeit wurden insgesamt 27 Autofahrten mit einer Gesamtlänge von über 5 Stunden in den Bezirken Hollabrunn, Korneuburg, Horn und Wien aufgenommen. Aus diesen Videos wurden Snippets erstellt, welche teilweise das gesamte Straßenbild abbilden, in der Regel jedoch die manuell positionierten, herausgecutteten und interessanten Regionen mit dem Verkehrsschild. Dies war dazu da zunächst zu ermitteln, wie das KI-Modell auf vollständige Straßenbilder reagiert, und danach welche Ergebnisse die einzelnen Verkehrsschilder liefern.

Die folgende Grafik zeigt die Ergebnisse der Erkennung des KI-Modells für 100 selbst ausgeschnittene Ausschnitte relevanter Frames aus diesen Videos:

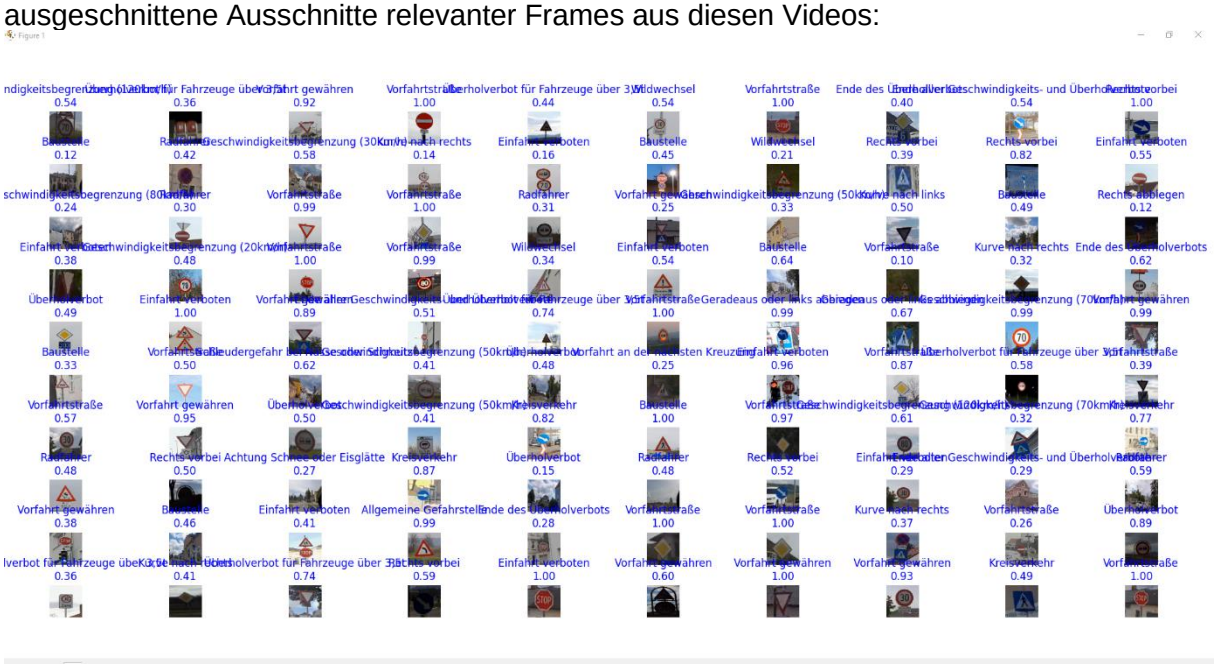


Abbildung 12: Ausschnitte aus Videos

Hierbei lässt sich erkennen, dass sich das KI-Modell deutlich unsicherer ist, als mit den Bildern aus dem Datensatz, welcher auch zum Training herangenommen wurde, obwohl sich die Bilder grundsätzlich nicht extrem stark unterscheiden. Hierbei handelt es sich im Grunde einfach um die eigen aufgenommenen Bilder. Hierbei konnten im Gegensatz zu den Bildern aus dem Datensatz 39/100 Bildern korrekt erkannt werden, hierbei ist auch zu berücksichtigen das auch die Gesamtstraßenabbildungen dabei sind, welche diese Ergebnisse verzerren, denn von den 100 Bildern sind 18 Gesamtstraßenbilder, welche so gut wie gar nicht erkannt werden können, da noch keine Region of Interest Detection erfolgt ist.

7.1.4 Echtzeit-Verkehrszeichenerkennung

Hier wird das Python-Programm für die Echtzeiterkennung über die Gerätekamera veranschaulicht.

Die folgenden Grafiken stellen die Ergebnisse diese Echtzeiterkennungs-Python-Applikation dar:

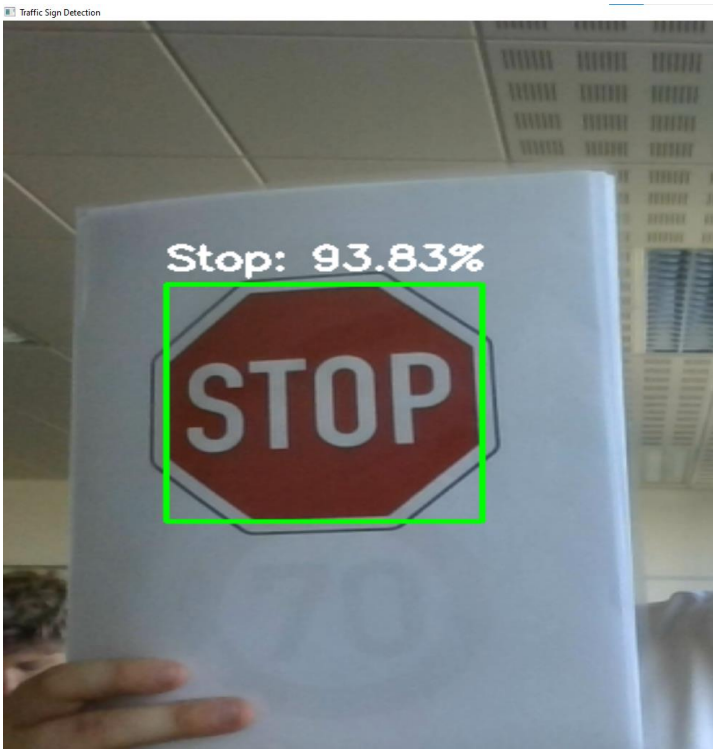


Abbildung 13: Echtzeiterkennung Beispiel 1

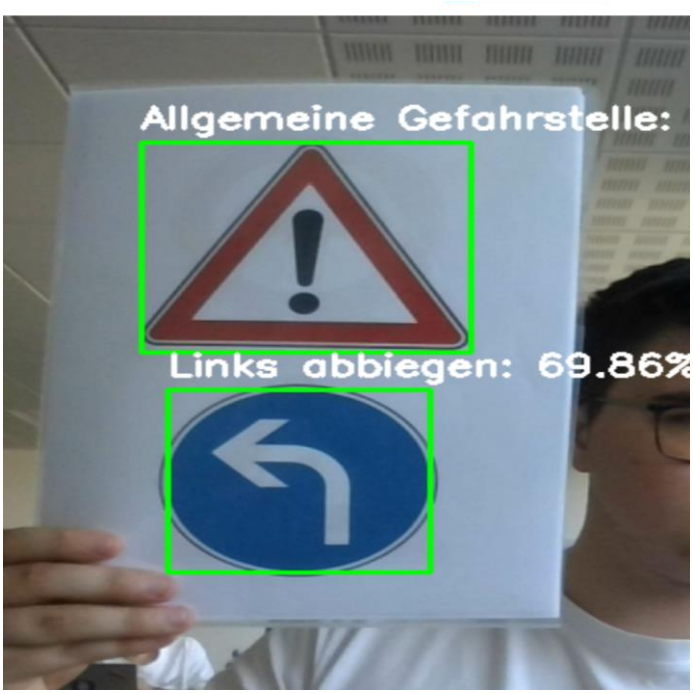


Abbildung 14: Echtzeiterkennung Beispiel 2

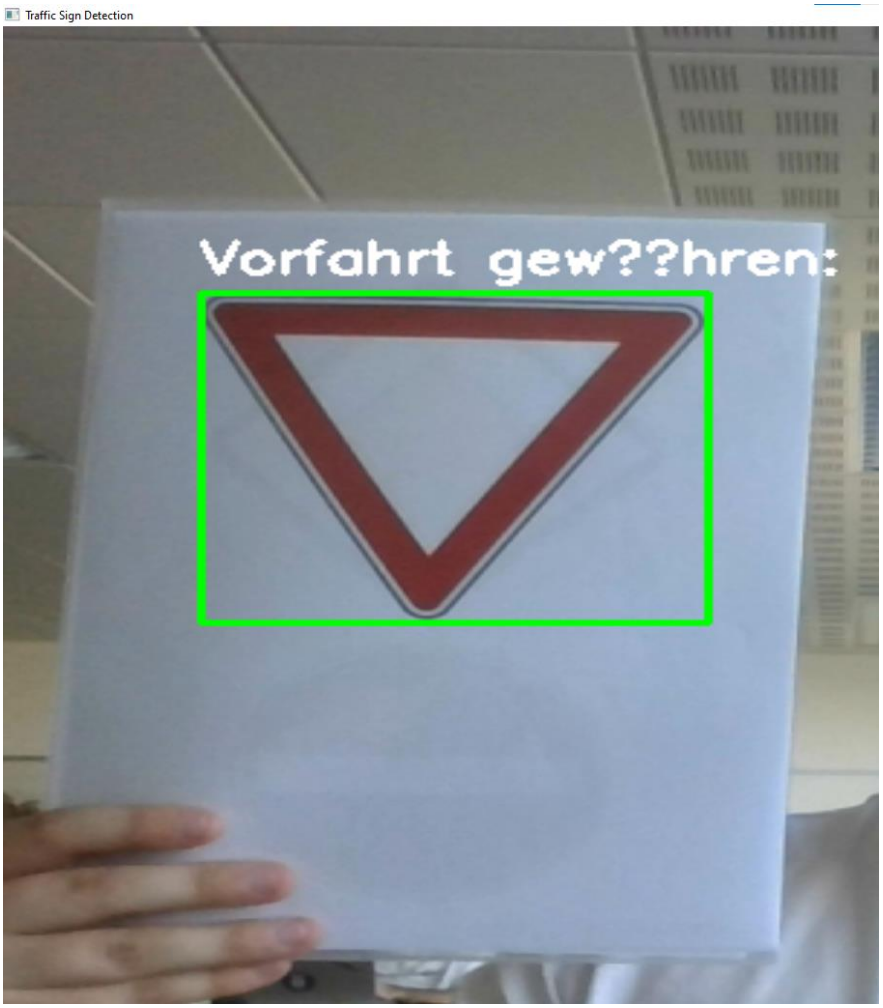


Abbildung 15: Echtzeiterkennung Beispiel 3

7.1.4.1 Herausforderungen

Bei den ersten Durchläufen mit der Echtzeiterkennung waren zwar die Bilddaten der interessanten Regionen im Frame richtig, jedoch wurde diese falsch an das KI-Modell übergeben und waren somit für das KI-Modell unbrauchbar. Das konnte mit einer kurzen Konvertierung zu python-internen Images und dann später auf PNG-Bilder gelöst werden. Obwohl eine ASCII Konvertierung des Textes stattfindet, können Umlaute noch nicht richtig verarbeitet werden. Außerdem kommt es noch teilweise zu Fehlern bei der Region-of-Interest-Filterung, sodass es teilweise zum Flackern kommt. Wenn ein Verkehrszeichen aktiv verarbeitet und erkannt wird sinkt auf die Bildwiederholungsrate und führt zum Effekt des Laggings. Hieraus lassen sich auch wertvolle Schlüsse zur Smartphone-Entwicklung schließen, wie z.B. andere Hardwareleistung und potenziell noch stärkere Lagging-Effekte auf mobilen Geräten, weil diese leistungsschwächer sind und außerdem noch mit Javascript gearbeitet wurde, was den Prozess noch weiter verlangsamt. Da das Modell nur 33 MB groß ist, ist die Speicherung kein Problem, auch auf Smartphones.

7.2 Echtzeiterkennung im Straßenverkehr

Für die praktische Anwendung und Demonstration des KI-Modells wurde es LIVE getestet in der echten, realen Anwendung in echten Verkehrssituationen.

7.2.1 Aufbau

Für die praktische Anwendung wurde das System mit mehreren Komponenten umgesetzt. Es wurde zunächst über den DroidCam Client eine Verbindung über das Netz vom mobilen Hotspot des Smartphones zum System hergestellt. DroidCam ermöglicht die kabellose Echtzeitübertragung der Kamera über WLAN-Netze. Nach dem die Verbindung aufgebaut wurde, wurde die Verkehrszeichenerkennung im Auto gestartet. Die Verkehrszeichenerkennung erkannte über die kabellose Echtzeitkameraübertragung des Smartphones auch erfolgreich Verkehrszeichen LIVE im Straßenverkehr. Die Fahrten wurden dabei LIVE mit dem Bildschirmaufzeichnungstool OBS-Studio aufgenommen.

Folgende Code-Erweiterungen ermöglichen Verbindungen zwischen dem Verkehrszeichensystem und der Smartphonekamera in Python:

```
 1. def find_traffic_signs(frame, model, debug_mode=False):
 2.     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
 3.     blurred = cv2.GaussianBlur(gray, (5, 5), 0)
 4.     edges = cv2.Canny(blurred, 100, 200)
 5.
 6.     contours, _ = cv2.findContours(edges, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
 7.
 8.     boxes = []
 9.     confidences = []
10.
11.     for contour in contours:
12.         area = cv2.contourArea(contour)
13.         if area < 700:  # Kleinere Schilder zulassen
14.             continue
15.
16.         x, y, w, h = cv2.boundingRect(contour)
17.         aspect_ratio = w / float(h)
18.
19.         if 0.6 <= aspect_ratio <= 1.6:
20.             approx = cv2.approxPolyDP(contour, 0.02 *
cv2.arcLength(contour, True), True)
21.             if len(approx) >= 4:
22.                 roi = frame[y:y+h, x:x+w]
23.                 predicted_class, confidence = predict_sign(roi, model,
debug_mode)
24.
25.                 # Neue Schwelle: schon ab 0.6 anzeigen
26.                 if confidence >= 0.6:
27.                     boxes.append((x, y, w, h))
28.                     confidences.append(confidence)
29.
30.     return non_maximum_suppression(boxes, confidences, overlap_thresh=0.2)
31.
32. # ----------------------------
```

```
33. # Kameraquelle
34. # DroidCam URL "http://192.168.0.100:4747/video"
35. # ----------------------------
36. CAMERA_URL = 1  # Für interne Kamera
37.
38. cap = cv2.VideoCapture(CAMERA_URL)
49. cv2.namedWindow("Traffic Sign Detection", cv2.WINDOW_NORMAL)
40. debug_mode = False
41.
```

Code-Listing 13: Erkennung im Straßenverkehr

Hierbei ist zu beachten das die Funktion find_traffic_signs() angepasst wurde, damit Verkehrsschilder auch schon ab einer Confidence von 0.6 erkannt werden sollen und auch das Schilder mit einer kleineren Fläche erkannt werden können. Die Bilddatenquelle kommt jetzt auch nicht mehr vom Stream 0, sonder vom Stream 1 welcher vom DroicCam Client erstellt wird und auch hier in Python verwendet werden kann. Alternativ könnte man auch direkt die IP und den Port des DroidCam Clients vom Smartphone verwenden.

Die folgenden Grafiken stellen die Ergebnisse dieser Echtzeiterkennungs-Python-Applikation im Straßenverkehr dar:

Abbildung 16: Straßenverkehr 1

Abbildung 17: Straßenverkehr 2

Abbildung 18: Straßenverkehr 3

Abbildung 19: Straßenverkehr 4

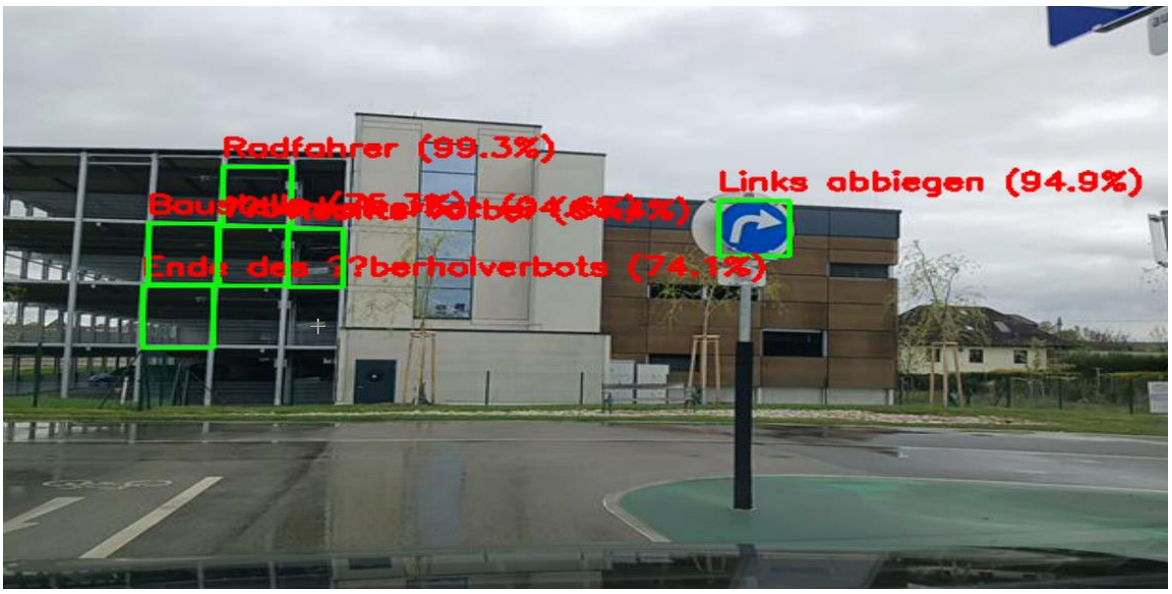


Abbildung 20: Straßenverkehr 5

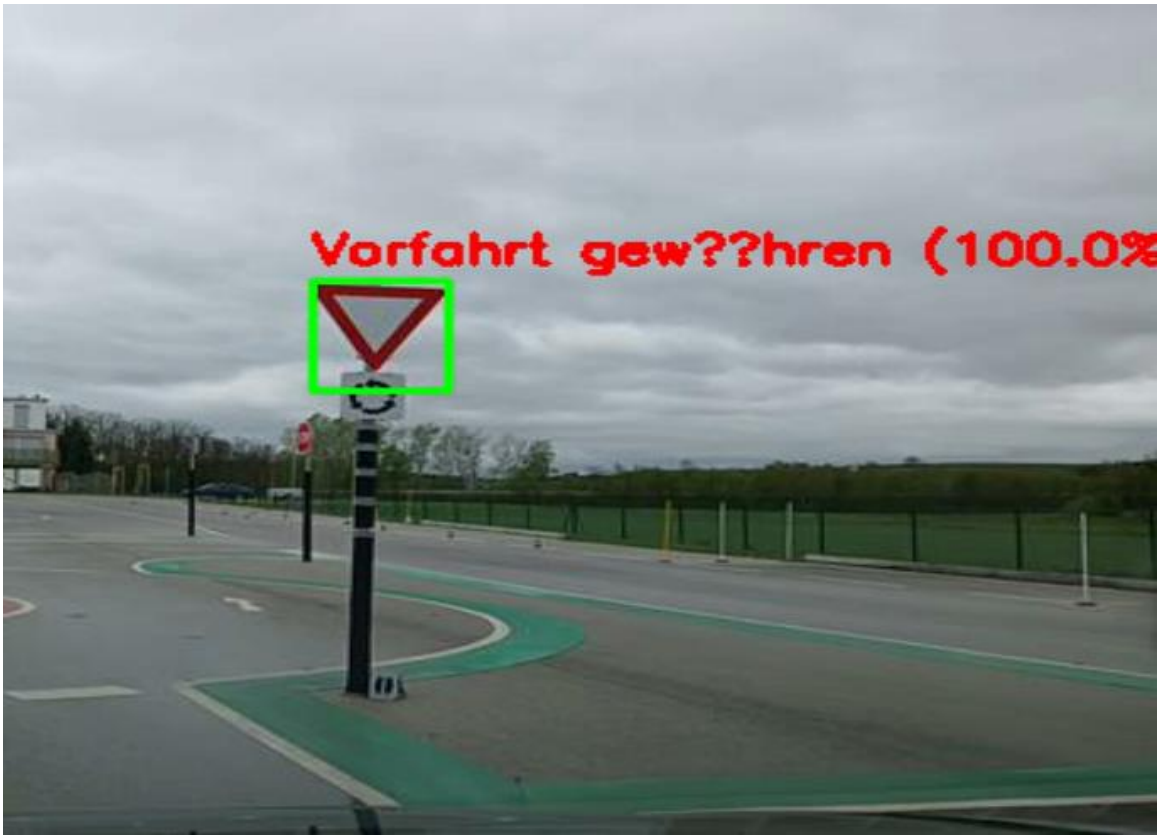


Abbildung 21: Straßenverkehr 6

Abbildung 22: Straßenverkehr 7

Abbildung 23: Straßenverkehr 8

Abbildung 24: Straßenverkehr 9

Abbildung 25: Straßenverkehr 10

Abbildung 26: Straßenverkehr 11

Abbildung 27: Straßenverkehr 12

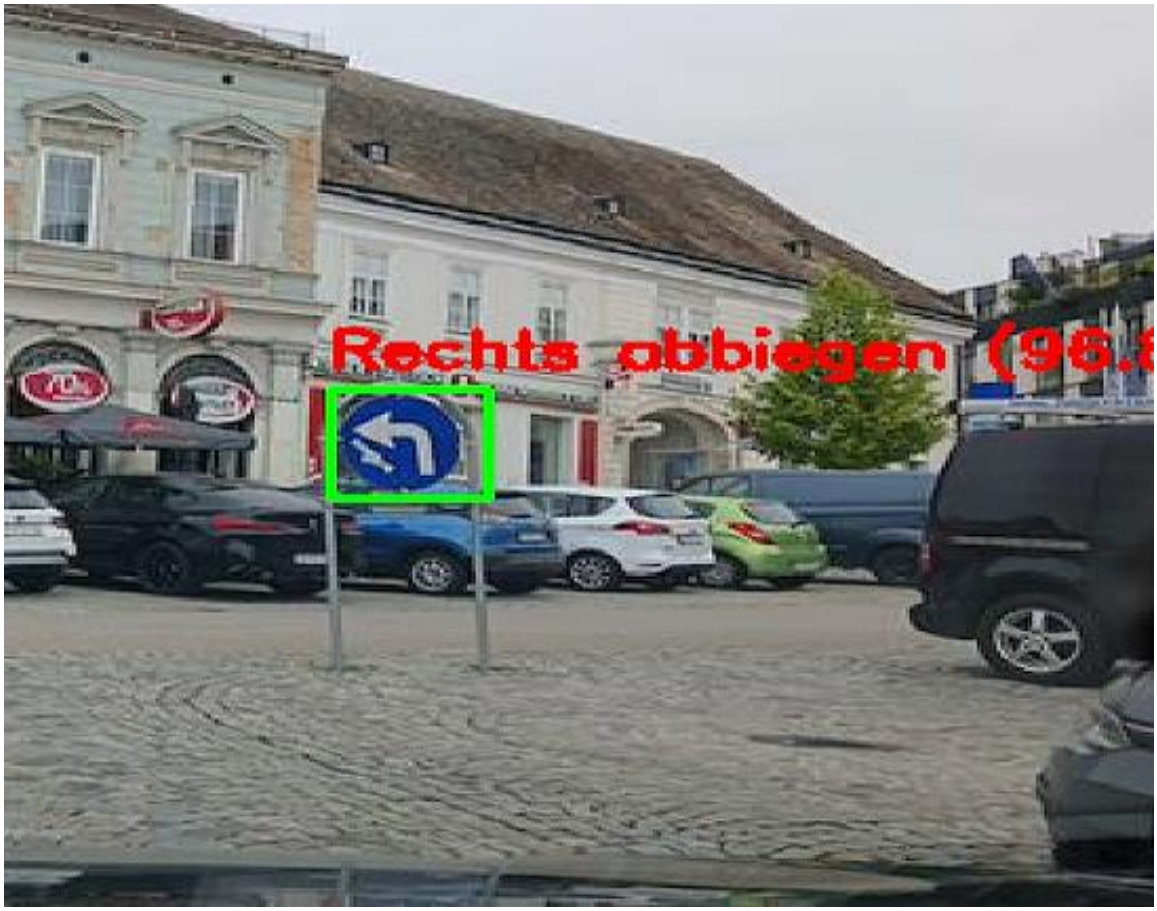


Abbildung 28: Straßenverkehr 13

Abbildung 29: Straßenverkehr 14

Abbildung 30: Straßenverkehr 15

Abbildung 31: Straßenverkehr 16

7.3 Interpretation der Testergebnisse

Aus diesen Testfahrten mit der Echtzeiterkennung der Verkehrszeichen im realen Straßenverkehr lassen sich sehr wertvolle Erkenntnisse und Erfahrungen schließen. Hierbei sind einige verschiedene Aspekte zu berücksichtigen. Zunächst war die Anwendung und der Aufbau sehr einfach, das Setup mit der Handyhalterung konnte übernommen werden. Verkehrszeichen könnten mühelos vom System erfasst werden und es wurden auch fehlerhafte Region of Interest Erkennungsfälle berücksichtigt, damit die Fehlerquote gering ist. Nur ab einer bestimmten Confidence, welche das KI-Modell liefert, wurden die Bilddaten auch eingerahmt und weiterverarbeitet. Verkehrszeichen wurden erfolgreich LIVE unterschieden, grafisch markiert, und erkannt. Die interessanten Bildausschnitte aus den aufgenommenen Fahrten geben eine sehr hohe Genauigkeit für erkannte Verkehrszeichen wieder, wobei auch die Varianz der Verkehrsschilder sehr gut gewählt und abwechslungsreich ist. Bekannte Fehler lassen sich sehr gut in *„Abbildung 21: Straßenverkehr 5“* abbilden und erklären. Hierbei kam es dazu, dass die jeweils auch rechteckigen Parkgaragenabteile fehlerhaft gekennzeichnet und erkannt wurde und, dass auch das „Rechts abbiegen“ Schild mit „Links abbiegen“ vertauscht wurde. Zusammengefasst bietet das Programm die gewünschte Qualität und Ergebnisse der Verkehrszeichenerkennung.

7.4 Herausforderungen

Bei den Fahrten ließ sich sehr schnell auch das vorherrschende Problem mit dem Lagging feststellen, weshalb das Bild bei der Verwendung und der Aufnahme ruckelt. Außerdem muss auch unbedingt die Framerate und die Geschwindigkeit des Autos berücksichtigt werden, denn dieser Prozess wurde auch mit schnelleren Geschwindigkeiten getestet, hierbei sinkt jedoch die Genauigkeit und Erkennungsrate, weil die Verkehrszeichen nicht so lange auf dem Bild bleiben und somit nur einige Frames Zeit bleibt diese zu erkennen. Bis ~70 km/h ließen sich gute Ergebnisse liefern. Wichtig ist auch zu beachten, dass das KI-Modell anders auf wechselnde Lichtverhältnisse reagiert. In Abbildung 31 und Abbildung 32 wurden die Verkehrszeichen auch bei Nacht vollständig richtig erkannt. Jedoch waren die Ergebnisse bei höheren Geschwindigkeiten, auf Landstraßen ohne Stadtbeleuchtung nicht reproduzierbar, weil einfach schon die Kamera nicht die nötige Kameraqualität liefern konnte, weil unter anderem die Autolichter oder auch das Fernlicht diese Schilder zu stark beleuchtet haben, und die Schilder für die Kamera immer zu grell waren. Fairerweise müsste man auch erwähnen ich als Mensch diese grellen Verkehrsschilder, bedingt durch die Lichter, auf dem Bildschirm auch nicht erkennen konnte.

7.5 Zusammenfassung der Testergebnisse

Die Analyse der Testergebnisse zeigt, dass das entwickelte KI-Modell eine Gesamtgenauigkeit von 85% - 95% (~90%) bei der Klassifikation von Verkehrszeichen erreicht. Dabei handelt es sich vor allem um Bilder, mit denen das Modell auch wirklich trainiert hat. Handelt es sich um andere Bilder sinkt diese Genauigkeit ab. Für die Genauigkeit wurden als Referenz aber auch bei allen anderen KI-Modellen und Architekturen immer Bilder aus dem Datensatz genommen, wie auch bei EfficientNetB0 oder Mobileye Supervision.

7.6 Vergleich mit bestehenden Lösungen

Das entwickelte Modell im Vergleich mit anderen bestehenden Verkehrsschilderkennungssystemen:

Kriterium	Unser System	Mobileye Supervision (Kommerzielle Nutzung)	Standard-CNN (EfficientNetB0)
Genauigkeit (GTSRB)	85%-95%	98%	78%-85%
Echtzeitfähigkeit	15 FPS (CPU)	30 FPS (ASIC)	8 FPS (CPU)
Modellgröße	33 MB	5 MB (komprimiert)	33 MB
Energieverbrauch	Hoch (CPU-basiert)	Sehr niedrig	Sehr hoch
Hardwareabhängigkeit	CPU/GPU	Dedizierte Hardware	GPU erforderlich

Die Vergleichsdaten wurden vom selbst-entwickeltem KI-Modell mit der eigenen Architektur für die TrafficSignDetection herangenommen. Die Performancedaten für Mobileye beziehen sich auf das Mobileye EyeQ®5 Datasheet (Intel). Für die Performancedaten von Standard-, Pretrained-Modell (EfficientNetB0) wurde eine demonstrative Implementation zum Vergleichen programmiert.

Folgender Code stellt für Vergleichszwecke die Implementation von so einem pretrained Standard-CNN-Modell gekürzt dar, wobei alle anderen Parameter und Trainingsdaten gleichbleiben:

```
# =====================
# 4. Modell erstellen (mit EfficientNetB0)
# =====================
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.2),
    tf.keras.layers.RandomZoom(0.2),
])

base_model = tf.keras.applications.EfficientNetB0(input_shape=(224, 224, 3), include_top=False, weights='imagenet')
base_model.trainable = False  # Wir frieren das Basis-Modell ein

model = tf.keras.Sequential([
    data_augmentation,
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(len(class_labels), activation='softmax')
])

```

Code-Listing 14: Pretrained Modell

7.6.1 Stärken unseres Ansatzes

- Vollständig open-source Implementierung
- Anpassbare Architektur für Forschungszwecke
- Keine Hardware-Abhängigkeiten
- Gute Ergebnisse mit vergleichsweise einfacher CNN-Architektur

7.6.2 Schwächen

- Geringere Robustheit bei extremen Bedingungen
- Keine temporale Integration (Daten werden nicht über die Zeit hinweg verknüpft)
- Begrenzte Mobile-Tauglichkeit durch Rechenbedarf

8 Fazit und Ausblick

8.1 Erreichte Ziele und Projekterfolge

Die Arbeit hat alle wesentlichen Zielsetzungen erfolgreich umgesetzt:

1. Entwicklung eines CNN-basierten Klassifikators mittels künstlicher Intelligenz mit hoher Genauigkeit
2. Implementierung einer vollständigen Verarbeitungspipeline von der Datenerfassung bis zur Echtzeiterkennung mittels ROI (Region of Interest) Detection
3. Erfolgreiche Demonstration der Verkehrszeichenerkennung
4. Ergebnisauswertung mittels Konfusionsmatrix und Metriken

8.2 Weitere Zielsetzungen

Weitere Ziele, welche teilweise bereits erfolgreich umgesetzt wurden, inkludieren:

1. Cross-Platform-App mittels dem Framework Expo Go, hierbei ist zu beachten das eine Region of Interest Detection im Frontend noch nicht umgesetzt wurde
2. Ortstafeln wurden im Fahrtaufnahmeprozess inkludiert und auch klassifiziert, jedoch nicht eingelernt und es gibt auch keine aktuelle Darstellung von Geschwindigkeitsbegrenzungen im Gültigkeitsbereich

8.3 Herausforderungen und Lösungsansätze während der Umsetzung

Die größten Herausforderungen und deren gewählte Lösungsansätze:

1. Datenungleichgewicht:
    - Problem: bis zu 5-fache Unterschiede in Klassenverteilung bzw. ungleiche Verteilung von Trainingsbildern pro Verkehrszeichenkategorie
    - Lösungsansatz: Gezielte und kalkulierte Anzahl an Datenmengen pro Kategorie
2. Modellarchitektur, Genauigkeit, Eigene CNNs, Pretrained Models
    - Problem: Richtige Auswahl der KI-Modell Struktur und Leistung der einzelnen Modellkonzeptionen
    - Lösungsansatz: komplett eigen definierte Modell Struktur ohne Abhängigkeit von bereitgestellten KI-Lösungen
3. Echtzeit-Performance:
    - Problem: Lags bei ROI-Detektion
    - Lösungsansatz: Optimierung durch Non-Maximum Suppression
4. Grafikverarbeitung:
    - Problem: Die ROI-Daten werden bei der Verarbeitung nicht im richtigen Format an das Modell übergeben
    - Lösungsansatz: Die ROI-Daten werden zuerst in das Format PNG konvertiert und erst danach an das Modell übergeben
5. Mobile Integration:
    - Problem: JS-Umsetzung der Bildverarbeitung
    - Lösungsansatz: Prototyp für spätere Implementierung wurde erstellt
6. Umlaut-Anzeigefehler:
    - Problem: Kodierungsfehler in der Ausgabe
    - Lösungsansatz: ASCII-Konvertierungsansatz

8.4 Zukünftige Weiterentwicklungen

Für die praktische Anwendung sollte das System als mobile App mit Expo Go umgesetzt werden, um die Performance auf Smartphones zu testen. Wichtig wäre die Optimierung der Bildwiederholrate, um das aktuelle Lagging-Problem zu beheben. Der Datensatz müsste um häufige Verkehrszeichen wie Ortstafeln erweitert und durch automatische Datensammlung kontinuierlich verbessert werden. Semi-supervised Learning in Kombination von Sensordaten und Entfernungsdaten könnte helfen, den Datensatz mit einer automatisierten Auswertung mit neuen Beispielen zu ergänzen.

9 Quellen

Community, React Native. (2025). *React Native Documentation*. Von https://reactnative.dev/docs/getting-started abgerufen

Corporation, Intel. (2025). *Mobileye EyeQ®5 Datasheet*. Von https://www.intel.com/content/ abgerufen

Foundation, Python Software. (2025). *Python Documentation*. Von https://docs.python.org/ abgerufen

Stallkamp, J. (2012). *German Traffic Sign Recognition Benchmark (GTSRB)*. Von https://benchmark.ini.rub.de/gtsrb_dataset.html abgerufen

Team, Expo. (2025). *Expo Documentation*. Von https://docs.expo.dev/ abgerufen

Team, Matplotlib Development. (2025). *Matplotlib Documentation*. Von https://matplotlib.org/stable/contents.html abgerufen

Team, OpenCV. (2025). *OpenCV Documentation*. Von https://docs.opencv.org/ abgerufen

Team, Pandas Development. (2025). *Pandas Documentation*. Von https://pandas.pydata.org/docs/ abgerufen

Team, TensorFlow. (2025). *Keras Documentation*. Von https://keras.io/ abgerufen

10 Verzeichnis der Abbildungen

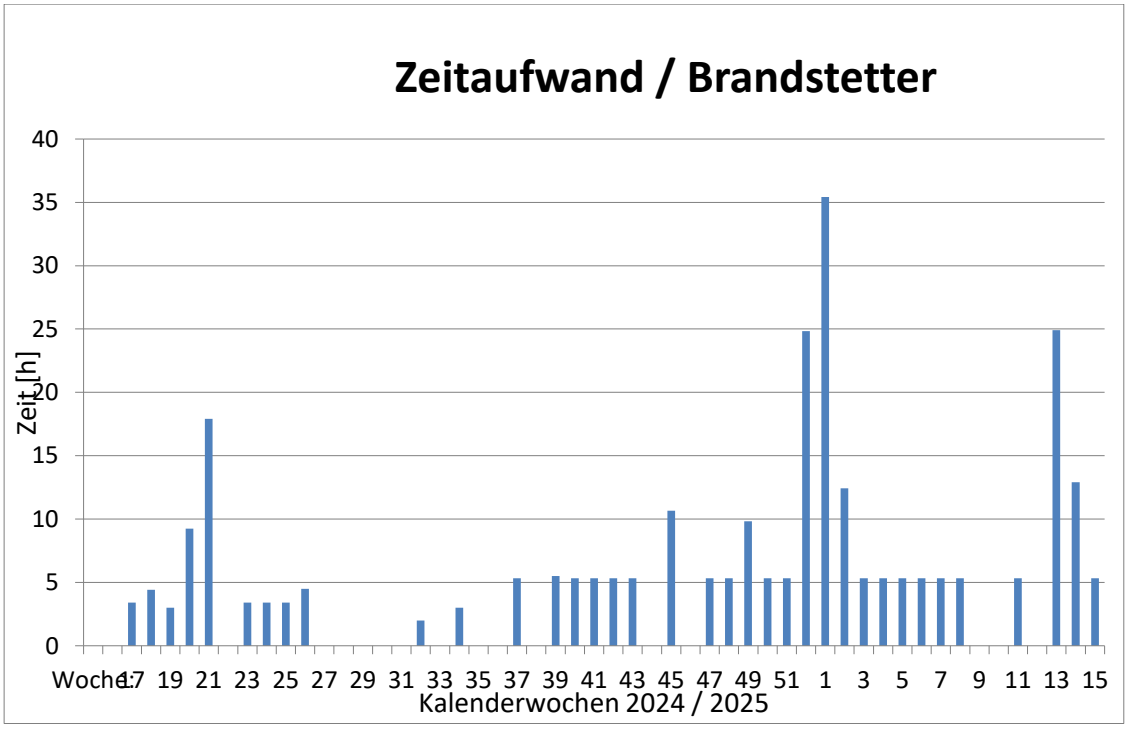
Abbildung 1: Logo 13
Abbildung 2: Das Grundprinzip von Convolutional Neural Networks 14
Abbildung 3: Technische Übersichtsgrafik 16
Abbildung 4: Projektstruktur 18
Abbildung 5: Handyhalterung 19
Abbildung 6: Verteilung der Datenmenge pro Kategorie (ungeordnet) 21
Abbildung 7: Verteilung der Datenmenge pro Kategorie (geordnet) 22
Abbildung 8: grafische Darstellung der Modellarchitektur 25
Abbildung 9: grafische Darstellung mittels Konfusionmatrix 35
Abbildung 10: grafische Darstellung der Modellgenauigkeit 36
Abbildung 11: grafische Darstellung von Train- und Validationloss 37
Abbildung 12: 100 Testbilder 38
Abbildung 13: Ausschnitte aus Videos 39
Abbildung 14: Echtzeiterkennung Beispiel 1 40
Abbildung 15: Echtzeiterkennung Beispiel 2 40
Abbildung 16: Echtzeiterkennung Beispiel 3 41
Abbildung 17: Straßenverkehr 1 43
Abbildung 18: Straßenverkehr 2 43
Abbildung 19: Straßenverkehr 3 44
Abbildung 20: Straßenverkehr 4 44
Abbildung 21: Straßenverkehr 5 45
Abbildung 22: Straßenverkehr 6 45
Abbildung 23: Straßenverkehr 7 46
Abbildung 24: Straßenverkehr 8 46
Abbildung 25: Straßenverkehr 9 46
Abbildung 26: Straßenverkehr 10 47
Abbildung 27: Straßenverkehr 11 47

Abbildung 28: Straßenverkehr 12 48
Abbildung 29: Straßenverkehr 13 48
Abbildung 30: Straßenverkehr 14 49
Abbildung 31: Straßenverkehr 15 49
Abbildung 32: Straßenverkehr 16 49

11 Verzeichnis der Code-Listings

Code-Listing 1: Datensatzstruktur des GTSRB in Python 21
Code-Listing 2: Datensatzvisualisierung 23
Code-Listing 3: Data-Augmentation in Python 24
Code-Listing 4: Skalierung und Normalisierung 24
Code-Listing 5: KI-Modelle in Python mittels Keras und Tensorflow 26
Code-Listing 6: EarlyStopping und Trainingsparameter 26
Code-Listing 7: Kamera-Funktionalität in Python 28
Code-Listing 8: Grafikverarbeitung in Python 29
Code-Listing 9: Region of Interest Detection 30
Code-Listing 10: KI-Modellanwendung in Python 31
Code-Listing 11: Echtzeit-Verkehrszeichenerkennung in Python 32
Code-Listing 12: Anzeige der Ergebnisse in Python 34
Code-Listing 13: Erkennung im Straßenverkehr 43
Code-Listing 14: Pretrained Modell 51

12 Begleitprotokoll



Gesamt: 314,25h
Freizeit: 210,84h