

# Erkennung deutscher Verkehrszeichen mittels verschiedener Objekterkennungsverfahren

Steven Solleder

Fakultät Informatik

Hochschule für angewandte Wissenschaften Hof

Hof, Deutschland

steven.solleder@hof-university.de

Isabell Waas

Fakultät Informatik

Hochschule für angewandte Wissenschaften Hof

Hof, Deutschland

isabell.waas@hof-university.de

**Zusammenfassung**—Das Ziel dieser Arbeit ist es, eine Android App zu entwickeln, welche in Echtzeit in einer Kameravorschau mittels verschiedener Objekterkennungsverfahren Verkehrszeichen lokalisiert und identifiziert. Zur Objekterkennung werden der ORB-Algorithmus von OpenCV als traditionelle Analysemethode und YOLO als Machine Learning Verfahren verwendet. Dabei werden insbesondere die Objekterkennungsverfahren und deren konkrete Implementierung näher betrachtet. Die wesentlichen Erkenntnisse aus dem anschließenden Vergleich von OpenCV und YOLO sind, dass es mit beiden Verfahren möglich ist, solch eine Anwendungen zu realisieren. Jedoch ist die Objekterkennung mit YOLO wesentlich schneller, genauer und zuverlässiger.

**Index Terms**—Bilderkennungsverfahren, Verkehrszeichenerkennung, Objekterkennungsverfahren, Echtzeiterkennung, Traditionelle Objekterkennung, Feature Detection, ORB, OpenCV, Machine Learning, Transfer Learning, Convolutional Neural Network, YOLO, Client-Server-Anwendung, RESTful Webservice, Android-App

## I. ZIEL DER ARBEIT

Deutschlands treibende Wirtschaftskraft ist schon seit vielen Jahren die Automobilindustrie. [1] Wie eine Umfrage von Statista von November 2023 zeigt, hat das eigene Auto für viele Deutsche einen sehr hohen Stellenwert und dient nicht nur als Fortbewegungsmittel, sondern häufig auch als Hobby. [2] Dementsprechend hoch ist auch die Beteiligung verschiedenster Verkehrsteilnehmer am Straßenverkehr. Um die Verkehrssicherheit dennoch gewährleisten zu können, definiert die Straßenverkehrsordnung (StVO) einen Katalog an Verkehrszeichen, der insbesondere Gefahrzeichen, Richtzeichen und Vorschriftzeichen beinhaltet. Insgesamt sind auf Deutschlands Straßen mehr als 400 verschiedene Schilder zu finden. [3] Mit einer großen Vielfalt an Verkehrszeichen steigt der Bedarf geeigneter Hilfssysteme, um Schilder automatisiert zu erkennen. Diese können beispielsweise sehbehinderten Personen die Orientierung im Straßenverkehr vereinfachen. Darüber hinaus setzen auch moderne Konzepte wie autonom fahrende Kraftfahrzeuge auf derartige Systeme.

Das Ziel des vorliegenden Projektes ist in erster Linie die Erstellung einer flexibel einsetzbaren Komponente zur Erkennung der bekanntesten deutschen Verkehrszeichen. Diese soll die Schilder in Echtzeit und sowohl von Nahem als auch aus der Distanz erfassen können. Während zu Beginn Verfahren zur traditionellen Bildanalyse verwendet werden, soll

im nächsten Schritt Machine Learning zum Einsatz kommen. Die entwickelte Komponente zur Verkehrszeichenerkennung soll schließlich in eine Client-Server Anwendung verpackt werden, damit sie in der Praxis genutzt werden kann. Hierbei wird im Backend ein RESTful Webservice verwendet. Den Client stellt eine mobile Android-App dar. In diese wird als Zusatzfunktion TextToSpeech eingebaut, um sehbehinderten Menschen zu ermöglichen, sich die Namen der erkannten Verkehrszeichen vorlesen zu lassen.

## II. TRADITIONELLE BILDANALYSE

### A. Bibliothek OpenCV



Abbildung 1. Logo von OpenCV [10]

Für die Traditionelle Bildanalyse wird sich der Bibliothek OpenCV bedient. Der Name OpenCV steht für Open Source Computer Vision, was den Zweck jener Bibliothek sehr gut beschreibt. OpenCV ist eine mit Apache 2 [4] lizenzierte Open Source Bibliothek und kann deshalb - unter Verwendung eines Hinweises auf die Lizenz und den Urheber - in privaten und kommerziellen Projekten verwendet werden. Sie beinhaltet eine große Sammlung von über 2500 verschiedenen klassischen und KI-gestützten Algorithmen für verschiedenste Zwecke, darunter das Vergleichen von Bildern oder das Erkennen von Objekten in Bildern. Die Bibliothek ist für viele bekannte Programmiersprachen wie zum Beispiel C++, Java und Python verfügbar und läuft nativ auf allen großen Betriebssystemen wie Windows, macOS, Linux und Android. [5] Alle gerade genannten Aspekte sorgen dafür, dass sich die Computer Vision

Bibliothek OpenCV perfekt für den Einsatz im vorliegenden Projekt eignet.

Im Folgenden soll nun im Groben erklärt werden, wie mithilfe von OpenCV ein Verkehrszeichen (Vorlagenbild) in einem auf der Straße aufgenommenen Bild (Testbild) erkannt wird. Zuerst muss sowohl im Vorlagenbild, als auch im Testbild nach sogenannten Features gesucht werden. Ein Feature besteht aus einem Keypoint und einem Descriptor. Ein Keypoint ist ein konkreter Punkt im Bild. Dieser Punkt ist normalerweise an einer besonders markanten bzw. auffälligen Stelle. Dazu zählen insbesondere Kanten, Ecken und komplexere Muster. Um diesen Punkt auch in anderen Bildern finden zu können, wird zusätzlich eine Beschreibung um den Punkt herum benötigt - die sogenannten Descriptoren. Im Rahmen dieses Projektes wird der besonders schnelle Oriented FAST and Rotated BRIEF (ORB) Algorithmus zur Feature-Erkennung verwendet, um eine Echtzeitanwendung zu ermöglichen. Nachdem in beiden Bildern jeweils eine Menge an Features erkannt wurde, muss die Schnittmenge beider Featuremengen bestimmt werden. Dieser Vorgang wird auch Feature Matching genannt. Ist die Schnittmenge ausreichend groß, kann davon ausgegangen werden, dass das Vorlagenbild tatsächlich im Testbild vorkommt. Die Abbildung 2 stellt den beschriebenen Vorgang beispielhaft an dem Verkehrszeichen „Wildwechsel“ dar. Schlussendlich müssen nur noch die konkrete Bounding Box sowie deren Perspektive berechnet werden. [6] [7]

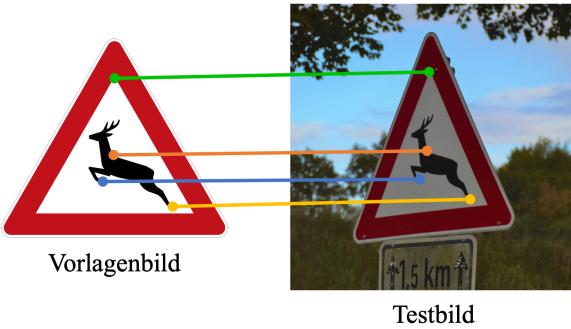


Abbildung 2. Beispiel für Feature Matching anhand des Verkehrszeichens „Wildwechsel“ [8] [9]

### B. Implementierung und Performanceoptimierungen

Im Folgenden wird auf Basis der gerade erläuterten traditionellen Bildanalyse in OpenCV eine Klasse namens TraditionalImageAnalysisDetector implementiert. Diese Klasse soll einen besonderen Fokus auf Effizienz besitzen und darauf optimiert sein, auch im Kontext unserer Echtzeitanwendung verwendet werden zu können. Es muss zusätzlich angemerkt werden, dass die folgenden Erläuterungen die Implementierung der Klasse lediglich konzeptionell und nicht eins zu eins wiedergeben.

Dem Konstruktor bzw. Initializer von der Klasse TraditionalImageAnalysisDetector werden die Vorlagenbilder übergeben, welche später beim Aufruf der Methode detect gesucht werden

sollen. Im Falle dieses Projekts werden dies die einzelnen Verkehrszeichen sein. Zuerst werden diese in Schwarz-Weiß Bilder umgewandelt, um Features besser erkennen zu können. Anschließend werden direkt die Features der Verkehrszeichen berechnet und dauerhaft als Instanzattribut gespeichert. Dadurch muss diese aufwendige Aufgabe nur einmalig beim Erstellen des TraditionalImageAnalysisDetector-Objekts getätigert werden. Zuletzt wird ein später benötigter Threadpool erstellt, welcher so viele Threads enthält, wie es Vorlagenbilder gibt. Dadurch muss nicht jedes Mal aufs Neue ein Thread erstellt werden, was ebenso nicht wenig Leistung bräuchte.

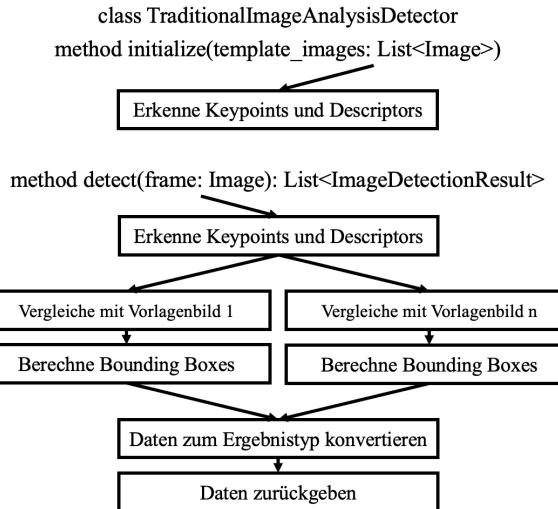


Abbildung 3. Grober Aufbau der TraditionalImageAnalysisDetector Klasse

Der Methode detect kann nun ein Bild übergeben werden. Auch dieses wird aus dem gleichen Grund wie die Vorlagenbilder im Konstruktor zuerst in Schwarz-Weiß umgewandelt. Daraufhin werden die Keypoints und Descriptors einmalig berechnet. Für jedes Vorlagenbild wird nun innerhalb eines Threads aus dem im Konstruktor erstellten Threadpool die Schnittmenge zwischen den Features des übergebenen Bildes und des jeweiligen Vorlagenbildes berechnet. Das heißt in jedem Thread findet ein eigenes Feature Matching statt. Wie viele uneingeschränkt laufenden Threads auf der benutzten Hardware gleichzeitig existieren können und der auf der Hardware verfügbare Arbeitsspeicher stellen somit das einzige Limit bei der Bilderkennung mit dieser Klasse dar. Ist die Schnittmenge ausreichend groß, wird die Bounding Box des Vorlagenbilds im Testbild berechnet. Haben alle Threads ihr Ergebnis an den Hauptthread zurückgegeben, werden diese Ergebnisse zu einer Liste aus ImageDetectionResult-Objekten umgewandelt. Nachstehend ist ein beispielhaftes ImageDetectionResult-Objekt im JSON-Format abgedruckt.

```
{
    "confidence": null,
    "content": "Rechts vorbei",
    "frameCoordinates":
    {

```

```

    "bottomLeft": {"x": 176, "y": 121},
    "bottomRight": {"x": 281, "y": 121},
    "topLeft": {"x": 176, "y": 19},
    "topRight": {"x": 281, "y": 19}
  }
}

```

Anzumerken ist, dass die Confidence nur bei der im nächsten Kapitel folgenden Analyse mit maschinellem Lernen eine Rolle spielt. Da es bei der traditionellen Bildanalyse keine Confidence gibt, ist der Wert im Beispiel null. Die X- und Y-Werte der Koordinaten innerhalb der FrameCoordinates geben positive ganzzahlige Werte in der Einheit Pixel an. Zuletzt wird die Liste aus ImageDetectionResult-Objekten von der detect-Methode zurückgegeben.

### III. BILDANALYSE MITHILFE VON MACHINE LEARNING

#### A. YOLO



Abbildung 4. Logo von Ultralytics YOLO [11]

Wie in Kapitel II-A beschrieben wurde, werden bei der Traditionellen Bildanalyse manuell vordefinierte Features wie Kanten oder Ecken verwendet, um Objekte zu identifizieren. Maschinelles Lernen (englisch: Machine Learning) erfasst hingegen automatisch anhand Beispieldaten, welche Merkmale eines Objekts wichtig sind. Daher kann Machine Learning aus Erfahrungen lernen und die Fähigkeit, Objekte zu erkennen, mit der Zeit verbessern.

Für die Analyse unserer Verkehrszeichenbilder mit maschinellem Lernen wird YOLO verwendet. YOLO ist ein weit verbreitetes Modell zur Objekterkennung und Bildsegmentierung in Echtzeit. Es wurde im Jahr 2016 von Joseph Redmond, Santosh Divvala, Ross Girshick und Ali Farhadi veröffentlicht. [12] Das amerikanische Unternehmen Ultralytics hat eine Open-Source-Implementierung von YOLO entwickelt. In unserem Projekt wurde YOLOv8 eingesetzt, die neueste Version der Implementierung von Ultralytics. Diese ist äußerst vielseitig nutzbar. So können verschiedenste Bildverarbeitungsaufgaben mithilfe von Künstlicher Intelligenz (KI) umgesetzt werden, darunter Object Detection, Instance Segmentation, Pose Estimation, Multi-Object Tracking sowie Image Classification. Im Rahmen des vorliegenden Projektes wird YOLO zur Objekterkennung genutzt. [13]

Andere Systeme zur Objekterkennung verwenden häufig mehrere Evaluationen eines oder mehrerer neuronaler Netzwerke an verschiedenen Stellen und Skalierungen innerhalb eines Bildes. Dies kann sehr lange dauern und ressourcenintensiv

sein. Im Gegensatz dazu steht YOLO für „You only look once“. Dies bezieht sich auf die Tatsache, dass YOLO eine einzige Evaluation eines neuronalen Netzes verwendet, um das gesamte Bild in einem Durchgang zu analysieren. Dadurch ist YOLO bemerkenswert schnell und auch sehr genau, da die Vorhersagen, wo sich welche Objekte befinden, durch den globalen Kontext des Bildes bestimmt werden.

Der Prozess der Objekterkennung mit YOLO ist in Abbildung 5 dargestellt und läuft wie folgt ab: Zunächst unterteilt YOLO das zu analysierende Bild in ein Gitter. Für jede Zelle dieses Gitters werden nun Bounding Boxen und Confidence Scores vorhergesagt. Ein Confidence Score gibt dabei an, wie sicher YOLO ist, dass eine Bounding Box ein Objekt enthält. Enthält eine Gitterzelle ein Objekt, dann werden ebenso Klassenwahrscheinlichkeiten für dieses Objekt ermittelt. Diese geben an, um welche Objektart es sich höchstwahrscheinlich handelt. Schließlich wird für jede Bounding Box der Confidence Score mit den entsprechenden Klassenwahrscheinlichkeiten multipliziert. Dies ergibt für jede Box einen klassenspezifischen Wahrscheinlichkeitswert, der angibt, wie wahrscheinlich die bestimmte Objektklasse in der Box vorkommt. Anhand der höchsten klassenspezifischen Wahrscheinlichkeitswerte wählt YOLO die relevantesten Bounding Boxen aus. Diese umrahmen die identifizierten und lokalisierten Objekte. [12]

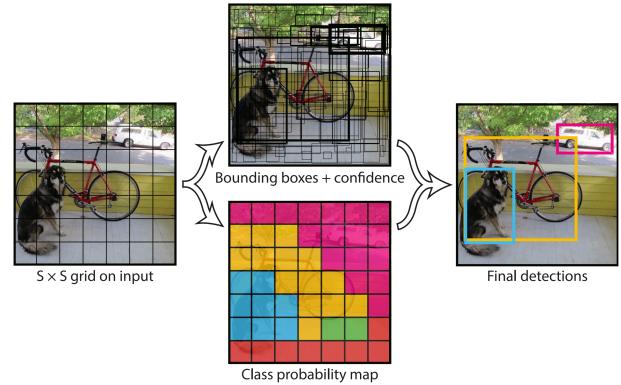


Abbildung 5. Visualisierung der Objekterkennung mit YOLO [12]

#### B. Datensätze

Als nächstes soll betrachtet werden, was für den Einsatz von YOLO im vorliegenden Projekt erforderlich ist. Zunächst wird ein Datensatz im Ultralytics YOLO-Format benötigt. Ein Datensatz besteht aus beschrifteten Bildern im .jpg Format. Das bedeutet, es gibt Bilder sowie für jedes Bild eine gleichnamige Textdatei, ein sogenanntes Label. Ein Label enthält die Klassenkennungen und Koordinaten aller Objekte, die in dem Bild erkannt werden sollen. [18] Der Inhalt des Labels zu dem beispielhaften Foto in Abbildung 6 ist im Folgenden dargestellt. Aus Platzgründen wurden einige Nachkommastellen der Koordinaten abgeschnitten.

17	0.3095588	0.656875	0.0294117647	0.04625
17	0.5772058	0.641875	0.0264705882	0.04125
38	0.58125	0.730625	0.024264705882	0.04125

Jede Zeile des Labels gehört zu genau einem Objekt und besitzt das Format: class x\_center y\_center width height. Dabei beschreiben die letzten vier Werte die Bounding Box, indem zuerst die x- und y-Koordinate des Mittelpunkts der Box und dann die Breite und Höhe der Box angegeben werden. Die Klassenkennung bezieht sich auf die verschiedenen Objektarten, die YOLO identifizieren können soll. Das sind in dem Beispiel zwei „Verbot der Einfahrt“ Schilder und ein „Rechts Vorbei“ Schild. Alle möglichen Klassenkennungen und -namen für den Datensatz werden in einer YAML-Konfigurationsdatei data.yaml festgelegt. Letztere enthält darüber hinaus auch das Stammverzeichnis des Datensatzes sowie die relativen Pfade zu den Verzeichnissen der Bilder. Üblicherweise werden Datensätze in Trainings-, Validierungs- und gegebenenfalls Testbilder unterteilt. Es existiert folglich je ein Ordner train, valid und gegebenenfalls test und jeder dieser enthält wiederum die Ordner images und labels. [18] Der Zweck der verschiedenen Arten von Bildern wird in Kapitel III-C erläutert.



Abbildung 6. Das Bild unseres Datensatzes zu dem Beispiellabel [16]

Da die Erkennung von Verkehrsschildern eine recht häufige Aufgabe im Bereich des maschinellen Lernens ist, wurden bei der Recherche im Internet zwei große Datensätze gefunden. Diese werden vom Institut für Neuroinformatik der Ruhr-Universität Bochum zur Verfügung gestellt. Sie wurden für Wettbewerbe der International Joint Conference on Neural Networks (IJCNN) in den Jahren 2011 und 2013 verwendet. [14] Die IJCNN ist die erste internationale Veranstaltung zu dem Thema neuronale Netze sowie verwandten Gebieten und wird jährlich gemeinsam von der International Neural Network Society und der IEEE Computational Intelligence Society veranstaltet. [15]

### 1) German Traffic Sign Detection Benchmark (GTSDB):

Der erste für das Projekt betrachtete Datensatz ist der German Traffic Sign Detection Benchmark (GTSDB). Er umfasst insgesamt 900 Bilder, darunter 600 für das Training und 300 für die Evaluierung. Die darauf gezeigten Verkehrsschilder werden in vier Klassen eingeteilt: „prohibitory“ (deutsch: verboten), „danger“ (deutsch: gefährlich), „mandatory“ (deutsch: vorgeschrieben) und „other“ (deutsch: andere). [16] Die ersten beiden Klassen spiegeln größtenteils die in der StVO

festgelegten Kategorien Vorschriftzeichen und Gefahrzeichen wider. Das wichtigste Merkmal des GTSDB-Datensatzes ist, dass die Bilder ganze Straßenverkehrssituationen darstellen (siehe Abbildung 7). Hierbei sind die Verkehrszeichen oft sehr weit entfernt, unscharf oder schlecht belichtet. Ebenso sind auf vielen Fotos mehrere von YOLO zu erkennende Schilder gleichzeitig abgebildet. Insgesamt werden auf den Bildern somit sehr realistische, alltagsnahe Situationen gezeigt. Schließlich soll erwähnt werden, dass der Begriff „Detection“ in dem Namen des Datensatzes darauf hindeutet, dass der Datensatz für das Identifizieren und Lokalisieren eines oder mehrerer Verkehrszeichen gleichzeitig gedacht ist. Dies würde auch erklären, warum die Bilder nicht nur die Verkehrszeichen selbst zeigen.



Abbildung 7. Beispielbilder des GTSDB-Datensatzes [16]

Bei den Recherchen wurde der GTSDB-Datensatz bereits in einem dem Ultralytics YOLO-Format sehr ähnlichen Format gefunden. Alle Bilder und ihre zugehörigen Labels befinden sich in einem einzigen Ordner, wobei in zwei separaten Textdateien angegeben ist, welche Bilder für das Training und welche für das Testen gedacht sind. Basierend darauf wird die für das vorliegende Projekt benötigte Ordnerstruktur manuell erstellt, indem die Ordner train und test mit ihren jeweiligen Unterordnern images und labels erzeugt und die Bilder und Labels entsprechend eingesortiert werden. Da YOLO jedoch auch Bilder für die Validierung benötigt, wird ein Teil der für das Testen vorgesehenen Bilder sowie ihre zugehörigen Labels in die Unterordner images und labels in einem Ordner valid verschoben. Zuletzt wird die data.yaml Datei erstellt, indem die Pfade zu den benötigten Ordnern mit Bildern angegeben und die Klassennamen aus einer Datei classes.names entnommen werden. [19]

### 2) German Traffic Sign Recognition Benchmark (GTSRB):

Als nächstes soll der zweite betrachtete Datensatz beschrieben werden, welcher wesentlich umfangreicher ist. Der German Traffic Sign Recognition Benchmark (GTSRB) umfasst mehr als 50.000 Bilder mit Verkehrszeichen, welche 43 verschiedenen Klassen zugeordnet werden. Letztere sind in Abbildung 8 dargestellt und beinhalten nicht alle, aber die wichtigsten deutschen Verkehrszeichen. [17]



Abbildung 8. Darstellung der 43 Klassen des GTSRB-Datensatzes [20]

Im Gegensatz zu dem GTSDB-Datensatz ist auf den Bildern des GTSRB-Datensatzes stets nur ein einziges Verkehrszeichen und sehr wenig Hintergrund zu sehen. Allerdings variieren auch hier die Schärfe und die Lichtverhältnisse, wie die Beispielbilder in Abbildung 9 erkennen lassen. Zudem fällt der namentliche Unterschied zwischen dem zuvor und dem aktuell betrachteten Datensatz auf, welcher in dem Wort „Recognition“ statt dem Begriff „Detection“ besteht. Dies weist darauf hin, dass der GTSRB-Datensatz nur für das Identifizieren, jedoch nicht für das Lokalisieren von Verkehrszeichen dient. Ebenfalls würde dies zu der Tatsache passen, dass auf den Fotos des aktuell betrachteten Datensatzes direkt die Schilder im Fokus liegen und kaum Hintergrund abgebildet ist.



Abbildung 9. Beispielbilder des GTSRB-Datensatzes [17]

Leider wurde der GTSRB-Datensatz im Internet weder im Ultralytics YOLO-Format noch in einem ähnlichen Format gefunden und muss daher vollständig manuell in ersteres konvertiert werden.

Als erstes werden die Trainingsbilder des Datensatzes in dem Ordner GTSRB\_Final\_Training\_Images betrachtet. Für die 43 Verkehrszeichenklassen existiert darin jeweils ein Ordner, dessen Name die Klassenkennung ist, die mithilfe führender Nullen auf fünf Ziffern gebracht wurde, z.B. 00012. Jeder dieser Ordner enthält Bilder sowie eine .csv Datei mit den

Klassenkennungen und Bounding Box Koordinaten der Objekte in den Bildern. Da die Bilder im .ppm Format vorliegen, werden sie zunächst in das gewünschte .jpg Format konvertiert. Hierfür wird das Python Skript convertImages.py genutzt, welches in dem unten stehenden Listing abgebildet ist.

```
import os
from PIL import Image

input_folder = "images"

output_folder = "converted_images"
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

for filename in os.listdir(input_folder):
    if filename.endswith(".ppm"):
        image = Image.open(os.path.join(input_folder, filename))
        image.save(os.path.join(output_folder,
                               os.path.splitext(filename)[0] + ".jpg"))

Listing 1. Skript convertImages.py
```

Dann werden die Labels für die Bilder mit einem zweiten Python Skript convertLabels.py aus der .csv Datei generiert.

```
import csv
import os

def convert_to_yolo_format(width, height, x1, y1, x2, y2):
    x_center = (x1 + x2) / (2 * width)
    y_center = (y1 + y2) / (2 * height)
    box_width = (x2 - x1) / width
    box_height = (y2 - y1) / height
    return x_center, y_center, box_width, box_height

csv_file_path = "dataset-information-class-0.csv"
class_id = "0"

output_folder = "labels"
os.makedirs(output_folder, exist_ok=True)

with open(csv_file_path, 'r') as csvfile:
    csv_reader = csv.DictReader(csvfile, delimiter=';')

    for row in csv_reader:
        filename = row['Filename']
        width = float(row['Width'])
        height = float(row['Height'])
        x1 = float(row['Roi.X1'])
        y1 = float(row['Roi.Y1'])
        x2 = float(row['Roi.X2'])
        y2 = float(row['Roi.Y2'])

        x_center, y_center, box_width, box_height =
            convert_to_yolo_format(width, height, x1, y1, x2, y2)

        output_file_path = os.path.join(output_folder,
                                         f"{filename.split('.')[0]}.txt")

        with open(output_file_path, 'w') as output_file:
            output_file.write(f"{class_id} {x_center}
{y_center} {box_width} {box_height}\n")

Listing 2. Skript convertLabels.py
```

Anschließend sollen die Ordner für die einzelnen Klassen aufgelöst werden, sodass alle Bilder in einem Ordner images und alle Labels in einem Ordner labels liegen. Jedoch treten hierbei mehrere doppelte Namen auf, da die Bilder und Labels jeweils innerhalb der Ordner für die Klassen nummeriert sind. Dieses Problem kann mit dem Python Skript renamer.py behoben werden, indem jedes Bild und jedes Label die Klassenkennung des darin enthaltenen Verkehrszeichens sowie einen Unterstrich als Präfix erhält. Ein Bild mit einem Verkehrsschild der Klasse 1 heißt somit beispielsweise 1\_00000\_00000.jpg und das zugehörige Label 1\_00000\_00000.txt.

```

import os

def add_prefix_to_files(folder_path, prefix):
    for filename in os.listdir(folder_path):
        if os.path.isfile(os.path.join(folder_path, filename)):
            old_path = os.path.join(folder_path, filename)
            new_path = os.path.join(folder_path,
                                   f"{prefix}_{filename}")
            os.rename(old_path, new_path)

prefix = "1"
add_prefix_to_files("images", prefix)
add_prefix_to_files("labels", prefix)

```

Listing 3. Skript renamer.py

Zuletzt wird noch der Ordner GTSRB\_Final\_Training\_Images in train umbenannt.

Nun sollen die Testbilder in dem Ordner GTSRB\_Final\_Test\_Images in das korrekte Format gebracht werden. Dieser beinhaltet Bilder im .ppm Format mit verschiedensten Verkehrszeichen sowie eine .csv Datei. Grundsätzlich kann bei der Konvertierung ähnlich wie bei den Trainingsbildern vorgegangen werden. Der entscheidende Unterschied ist jedoch, dass in der .csv Datei keine Klassenkennungen angegeben sind und die mithilfe des convertLabels.py Skriptes erstellten Labels somit keine Klassenkennungen enthalten. Da auf den Bildern verschiedenste Verkehrsschilder vorkommen und deren Klassenkennungen in keinster Weise aus dem Namen der Bilder bzw. Labels abgeleitet werden können, ist die einzige Lösung, manuell jedes Bild zu betrachten und die Klassenkennungen in das jeweilige Label zu schreiben. Da dieses Vorgehen für die mehr als 12.000 Testbilder unverhältnismäßig viel Zeit in Anspruch nehmen würde, wird es nur für etwa 200 Bilder und zugehörige Labels durchgeführt. Dementsprechend werden auch nur diese ca. 200 Bilder bzw. Labels als Testbilder in den GTSRB-Datensatz für das vorliegende Projekt aufgenommen. Schließlich wird auch hier der Ordner von GTSRB\_Final\_Test\_Images in test umbenannt und die darin enthaltenen Bilder und Labels werden in Ordner namens images und labels einsortiert.

Aufgrund der Tatsache, dass auch hier Validierungsbilder fehlen, jedoch wegen eben beschriebener Problematik bereits wenige Testbilder vorliegen, wird das erste Trainingsbild bzw. -label jeder Verkehrsschildklasse in die Unterordner images und labels in einem Ordner valid verschoben. Die betroffenen Bilder und Labels können leicht anhand ihres Namens identifiziert werden, da sie stets mit einer Klassenkennung von 0 bis 42 beginnen und mit „\_00000\_00000“ enden.

Zum Schluss wird die data.yaml Datei erstellt. Sie beinhaltet die Pfade zu den verschiedenen Ordnern mit Bildern sowie die Klassennamen der 43 Verkehrsschildarten. Die Klassennamen sind nicht in dem Datensatz enthalten und werden daher selbst recherchiert. Dabei werden größtenteils die offiziellen Bezeichnungen der StVO, jedoch insbesondere bei langen Namen auch manchmal kürzere, alltagsgebräuchliche Begriffe für die Schilder verwendet.

**3) Finaler Datensatz:** Im Rahmen des vorliegenden Projektes sollen eines oder mehrere Verkehrsschilder zugleich in alltäglichen Situationen identifiziert und lokalisiert werden können. Dies soll auch bei nicht ganz optimalen Bedingungen, beispielsweise bei schlechten Lichtverhältnissen, bestmöglich funktionieren. Auch sollen die Verkehrszeichen sowohl aus der Ferne als auch aus der Nähe erkannt werden.

Wie bereits erwähnt beinhaltet der GTSRB-Datensatz lediglich Bilder, in denen Verkehrsschilder aus kurzer Distanz fotografiert wurden und im Fokus liegen. Daher kann YOLO mithilfe dieses Datensatzes auch nur genau dann ein Verkehrszeichen in einem Bild korrekt einordnen, wenn dieses ebenfalls sehr groß und mit wenig Hintergrund gezeigt wird. Genau umgekehrt verhält es sich mit dem GTSDB-Datensatz, welcher für die Erkennung der Schilder in komplexeren Straßensituationen und aus größerer Entfernung ausgelegt ist. Aus diesem Grund wurde entschieden, die beiden betrachteten Datensätze zu kombinieren.

Der GTSRB-Datensatz liegt nach den in Kapitel III-B2 erläuterten Anpassungen bereits im Ultralytics YOLO-Format vor. Er verwendet die 43 Klassen, welche in Abbildung 8 dargestellt sind. Der Einfachheit halber und da die wichtigsten deutschen Verkehrszeichen enthalten sind, soll unser finaler Datensatz ebenfalls genau diese 43 Klassen verwenden.

Was den GTSDB-Datensatz betrifft, so besitzt auch dieser nach den in Kapitel III-B1 beschriebenen Änderungen das gewünschte Format. Allerdings werden die zu erkennenden Objekte hier nur in vier Verkehrszeichenkategorien eingeteilt, sodass YOLO mit diesem Datensatz den genauen Namen des jeweiligen Schildes nicht ermitteln könnte. Aus diesem Grund werden die 43 Klassen des GTSRB-Datensatzes verwendet, um den GTSDB-Datensatz neu zu klassifizieren. Hierzu müssen die Klassenkennungen in den Labels aller Bilder angepasst werden. Dies geschieht manuell, indem jedes Bild zunächst betrachtet wird und die zu erkennenden Verkehrszeichen entdeckt werden. Dann wird in dem zugehörigen Label anhand der Koordinaten der Bounding Box für jedes Schild die korrekte Textzeile gefunden und die alte durch die neue Klassenkennung ersetzt. So besaß das Verkehrszeichen „Gefahrstelle“ beispielsweise zuvor die Klassenkennung 1 für die Kategorie „danger“ und nun hat es die Klassenkennung 18, welche auf die konkrete Verkehrsschildart „Gefahrstelle“ hinweist.

Nachdem nun beide Datensätze das Ultralytics YOLO-Format und die korrekten Klassen besitzen, können sie zu einem neuen Datensatz kombiniert werden. Hierfür kann die data.yaml Datei des GTSRB-Datensatzes nahezu unverändert genutzt werden, indem lediglich die Pfade zu den Bildern angepasst werden. Dann werden die Ordner test, train und valid mit ihren jeweiligen Unterordnern images und labels erstellt. Anschließend werden alle Bilder und Labels beider Datensätze in diese Ordner einsortiert. Der finalen Datensatz umfasst schließlich fast 40.000 Bilder für das Training und ebenso viele zugehörige Labels. Für die Validierung sind es ca. 80 und für das Testen etwa 270 Bilder sowie Labels.



Abbildung 10. Beispielbilder des Finalen Datensatzes [16] [17]

### C. Modell

Nachdem nun ein geeigneter Datensatz vorliegt soll erläutert werden, wie dieser genutzt wird, um die Objekterkennung mit YOLO in dem vorliegenden Projekt umzusetzen. Hierzu wird ein sogenanntes Modell benötigt.

Bei dem in unserem Fall benötigten Modell handelt es sich um eine spezielle Form eines neuronalen Netzes, nämlich um ein sogenanntes Convolutional Neural Network (CNN). Diese Art von Deep-Learning-Modell arbeitet mit mathematischen Prinzipien zur Mustererkennung und eignet sich besonders dann, wenn Daten verarbeitet werden sollen, die ein Gittermuster aufweisen. Beispiele hierfür wären Anwendungen in den Bereichen Bild-, Video- und Spracherkennung. Ein CNN besteht in der Regel aus verschiedenen Layern. Dabei gibt es stets mindestens ein Convolutional Layer, ein Pooling Layer und ein oder mehrere Fully-connected Layers. Der erste Typ von Schichten dient hauptsächlich dem Finden von Mustern und dem Erlernen von Merkmalen aus den Eingabedaten. Das Pooling Layer soll anschließend die durch das beziehungsweise die Convolutional Layer gewonnenen Informationen reduzieren, sodass nur die wesentlichen Informationen verbleiben. Dies verhindert unter anderem Overfitting, was bedeuten würde, dass das Modell zu spezifische Details der Trainingsdaten lernt. Somit würden irrelevante Merkmale als wichtig einstuft werden und das Modell würde bei unbekannten Daten schlechte Ergebnisse liefern. Zuletzt wandelt das beziehungsweise wandeln die Fully-connected Layer die erkannten Merkmale in eine endgültige Ausgabe um. Diese kann beispielsweise in einer Klassifizierung bestehen. [24]

Die ursprüngliche Architektur des neuronalen Netzwerks von YOLO aus der originalen Publikation zeigt Abbildung 11. Darin ist zu sehen, dass die erste Version von YOLO insgesamt 24 Convolutional Layers verwendet hat. Auf diese folgten 4 Pooling Layers und schließlich 2 Fully-connected Layers. Mit den Jahren wurde YOLO immer weiter verbessert, wobei sich die anfängliche Architektur auch weiterentwickelt hat, und es folgten mehrere neue Versionen. Die neueste und in dem vorliegenden Projekt genutzte Version ist YOLOv8. Diese verwendet ein wesentlich komplexer aufgebautes neuronales

Netz. Allein dessen Hauptteil, das sogenannte Backbone, nutzt 53 Convolutional Layers, da er auf dem CNN CSP-Darknet53 basiert. [25] [26]

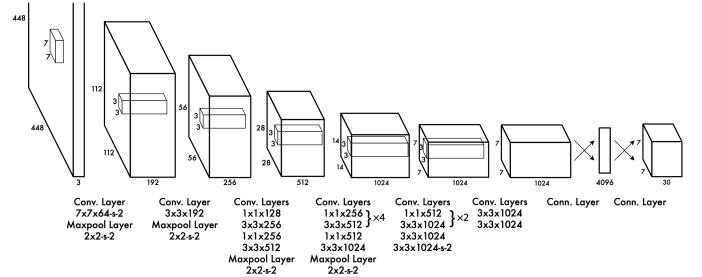


Abbildung 11. Architektur des neuronalen Netzwerks der ursprünglichen Version von YOLO [12]

*1) Auswahl eines vorab trainierten Modells:* Die verwendete YOLO-Version, YOLOv8, bietet bereits eine Vielzahl von Modellen, die auf bestimmte Aufgaben spezialisiert sind. Zu diesen gehören beispielsweise die Objekt-Erkennung oder Posen-Schätzung. Die für uns relevanten Modelle für die Objekt-Erkennung werden auf dem COCO-Datensatz trainiert. [21] Dieser besteht aus 80 Kategorien an gewöhnlichen, häufig zu identifizierenden Objekten wie Personen, Autos oder verschiedene Tierarten.

Auch wenn der COCO-Datensatz abgesehen von Stoppschildern keine Verkehrszeichen beinhaltet, lohnt es sich, eines der beschriebenen Modelle als Grundlage für das weitere Training zu verwenden. [22] Das Vorgehen, ein vortrainiertes Modell für einen neuen, jedoch verwandten Anwendungsfall zu nutzen, wird als Transfer Learning (TL) bezeichnet. Daraus ergeben sich verschiedene Vorteile gegenüber dem Trainieren eines neuen Modells von Grund auf. Zunächst ist TL sehr viel effizienter. So wird zum Beispiel sehr viel Zeit gespart, da das vorhandene Modell bereits ein elementares Wissen über Objekteigenschaften, Aufgaben, Gewichte und Funktionen erworben hat, auf dem weiter aufgebaut werden kann. Dies führt darüber hinaus auch dazu, dass ein wesentlich kleinerer Datensatz sowie weniger Rechenleistung benötigt werden, um das vorab trainierte Modell für die neue Aufgabe zu erweitern. Abgesehen davon führt TL oft auch zu leistungsstärkeren Modellen. Dies ist dadurch begründet, dass das Modell in seinem Trainingsprozess verschiedenste Situationen mit unterschiedlichen realistischen Störfaktoren kennengelernt hat. Somit kann es auch unter nicht optimalen Bedingungen ausreichend gut funktionieren. [23]

Um nun TL im vorliegenden Projekt zu verwenden, muss eines der Modelle für Objekt-Erkennung von YOLOv8 ausgewählt werden. Gemeinsam haben alle fünf Modelle, dass sie standardmäßig für das Verarbeiten von Bildern in der Auflösung 640 mal 640 Pixel entworfen sind. Ebenso besitzen alle eine ähnliche Architektur, jedoch unterscheiden sie sich in der Komplexität der verwendeten Layers. Die nachfolgende Tabelle I gibt eine Übersicht über die relevantesten Unterschiede zwischen den Modellen. Was deren Namen betrifft so weist der letzte Buchstabe stets auf die Größe des Modells hin. Das

bedeutet, dass zum Beispiel bei YOLOv8n das „n“ für „Nano“ steht, bei YOLOv8l hingegen das „l“ für „Large“. [21] [27]

Modell	mAPval 50-95	Geschwindigkeit CPU ONNX (ms)	Geschwindigkeit A100 TensorRT (ms)
YOLOv8n	37.3	80.4	0.99
YOLOv8s	44.9	128.4	1.20
YOLOv8m	50.2	234.7	1.83
YOLOv8l	52.9	375.2	2.39
YOLOv8x	53.9	479.1	3.53

Tabelle I

DIE MODELLE ZUR OBJEKT-ERKENNUNG VON YOLOV8 [21]

In der Tabelle sind für jedes Modell verschiedene Werte angegeben. Diese dienen als Entscheidungsgrundlage bei der Auswahl des Modells.

Als erstes soll der mAPval 50-95 betrachtet werden. Die Abkürzung mAP steht dabei für mean Average Precision und val weist wahrscheinlich darauf hin, dass die Bewertung anhand eines Datensatz zur Validierung stattgefunden hat. Um die Bedeutung des Werts erklären zu können, müssen zunächst ein paar andere Fachbegriffe erläutert werden. So misst Precision, wie oft das Modell richtig liegt, wenn es etwas als positiv vorhersagt. Der Recall gibt dagegen an, wie gut das Modell alle tatsächlich positiven Fälle herausgefunden hat. Das bedeutet, wenn das Modell zum Beispiel alles als positiv einstuft, wäre der Recall hoch, die Precision jedoch niedrig. Da Precision und Recall folglich miteinander im Konflikt stehen können, sollte ein ausgewogenes Verhältnis zwischen beiden angestrebt werden. Ein weiterer zu betrachtender Wert ist IoU, was für Intersection over Union steht. Der IoU misst, wie sehr die vorhergesagte und die tatsächliche Box um das zu erkennende Objekt überlappen. Der IoU kann Werte zwischen 0 und 1 annehmen. Je größer der IoU ist, umso mehr stimmen die beiden Boxen überein, das heißt, umso genauer wurde das Objekt lokalisiert. Der mAP wird nun verwendet, um zu bewerten, wie gut und konsistent das Modell Objekte klassifiziert und lokalisiert. Dazu berechnet es den Durchschnitt der Precision-Werte bei verschiedenen Recall-Werten für jede Objektklasse. Genauer ist mAP der Durchschnitt der Bereiche unter den Precision-Recall-Kurven jeder Klasse. Der IoU dient dabei dazu, zu bestimmen, ob eine Vorhersage als korrekt gilt. Die Angabe 50-95 hinter mAPval in der Tabelle bedeutet, dass IoU-Werte von 0,5 bis 0,95 mit einer Schrittweite von 0,05 verwendet werden und schließlich der Durchschnitt all dieser Werte genutzt wird. Insgesamt heißt ein hoher mAP, dass das Modell sehr genau ist. [28] Da die Tabellenwerte in der dazugehörige Spalte von oben nach unten steigen, erreicht somit das YOLOv8x Modell die präzisesten Ergebnisse.

Nun soll als zweites ein Blick auf die Geschwindigkeit der Modelle geworfen werden. Mit dieser beschäftigen sich die dritte und vierte Spalte der Tabelle. Beide geben an, wie viele Millisekunden das Modell gebraucht hat, um Eingabedaten zu verarbeiten und eine Inferenz, das heißt z.B. eine Vorhersage, auszugeben. Der Unterschied zwischen den

Spalten ist die verwendete Hardware. Bei der dritten Spalte wird ein Prozessor (CPU) gemeinsam mit ONNX genutzt. ONNX steht für Open Neural Network Exchange. Es handelt sich dabei um ein offenes Ökosystem, mit dem neuronale Netze in verschiedenen Tools und Bibliotheken interoperabel verwendet werden können. [31] Dagegen kommt bei der letzte Spalte ein speziell für Aufgaben im Bereich KI optimierter Grafikprozessor (GPU) zum Einsatz, die NVIDIA A100 Tensor Core-GPU. Diese nutzt TensorRT, eine Bibliothek von NVIDIA, die die Inferenz von neuronalen Netzwerken beschleunigt. Aus diesem Grund sind die Zeiten hier um ein Vielfaches geringen als in der vorherigen Spalte. [29] [30] Dennoch ist unabhängig von der konkret verwendeten Hardware eindeutig zu sehen, dass die Geschwindigkeit der Modelle von oben nach unten sinkt. Folglich nimmt die Objekterkennung mit dem YOLOv8n Modell am wenigsten Zeit in Anspruch.

Im vorliegenden Projekt sollen Verkehrszeichen unter verschiedensten Bedingungen möglichst genau erkannt werden. Gleichermaßen wichtig ist jedoch, dass die Objekterkennung so schnell wie möglich, im Idealfall in Echtzeit erfolgt. Um beide Anforderung bestmöglich zu erfüllen, scheint es zunächst am sinnvollsten, das YOLOv8m Modell zu wählen. Dieses stellt die mittlere Zeile in der Tabelle und damit den besten Kompromiss zwischen Präzision und Geschwindigkeit dar. Jedoch wird bei den ersten Trainingsversuchen schnell festgestellt, dass die Klassifizierungsergebnisse mithilfe dieses Modells bei Weitem nicht gut genug sind. Wird hingegen das YOLOv8x Modell herangezogen, welches die zuverlässigste, jedoch langsamste Objekterkennung ermöglicht, verbessert sich die Genauigkeit um ein Vielfaches. Was die Geschwindigkeit betrifft, so ist diese für unseren Anwendungszweck noch immer ausreichend. Aus diesem Grund wird final entschieden, das YOLOv8x Modell als Grundlage für das weitere Training zu verwenden.

2) *Trainieren des Modells:* Wie bereits beschrieben wurde, wird beim Transfer Learning ein vortrainiertes Modell für einen verwandten Anwendungsfall genutzt. Um nun das YOLOv8x Modell, welches bereits die 80 verschiedenen Objektarten des COCO-Datensatzes erkennen kann, zur Identifizierung und Lokalisierung von unseren 43 Verkehrszeichenklassen verwenden zu können, muss es mithilfe unseres Datensatzes weiter trainiert werden.

In der Dokumentation von Ultralytics YOLOv8 werden dazu verschiedene sogenannte Modi aufgelistet. Neben Predict, dem Modus, mit dem später das trainierte Modell die Objekte in Bildern erkennt, wird dabei auch Train genannt. Dieser Modus dient dazu, das Modell mit benutzerdefinierten Datensätzen für die gewünschte Aufgabe zu optimieren. Die genutzten Hyperparameter können dabei angepasst werden, um die Performanz des Modells zu verfeinern. Zu diesen gehört unter anderem die Größe des Bildes, auf dem Vorhersagen getroffen werden, oder die Anzahl der Epochen, welche im nächsten Absatz erläutert werden. YOLOv8 zeichnet sich durch besondere

Effizienz aus, da es die verfügbaren Hardware-Ressourcen so effektiv wie möglich einsetzt. Das bedeutet zum Beispiel, dass sowohl CPUs als auch GPUs für das Training nutzbar sind. Ebenso werden moderne Komponenten wie die M1 und M2 Chips der Firma Apple unterstützt, welche unter anderem Aufgaben im Bereich der Bildverarbeitung sehr performant ausführen können. Darüber hinaus ist das Training über Kommandozeilen- und Python-Schnittstellen möglich. Ebenfalls werden währenddessen zum Beispiel Ausgaben in der Kommandozeile sowie Grafiken erstellt, mit denen genau analysiert werden kann, wie das Training des Modells verlaufen ist. [32] [33]

Als nächstes soll darauf eingegangen werden, wie das Training konkret abläuft. An dieser Stelle wird ebenfalls erklärt, welchen Zweck die Aufteilung des Datensatzes in Trainings- und Validierungsbilder hat. Ein Trainingsprozess erstreckt sich in der Regel über mehrere Durchläufe, sogenannte Epochen. In jeder Epoche wird das Modell mit den Trainingsbildern unseres Datensatzes gefüttert und YOLO passt seine Parameter an, um die Vorhersagen genauer zu machen. Betrachtet man die 43 Klassen von Verkehrszeichen, so fällt zum Beispiel auf, dass die Schilder zur Begrenzung der Höchstgeschwindigkeit sowie die dreieckigen Schilder mit einem schwarzen Symbol in der Mitte sich untereinander stark ähneln. Sie besitzen immer dieselbe Form und Farben, lediglich der Bereich in der Mitte ist unterschiedlich. Es könnte also sein, dass festgestellt wird, dass das Modell die Verkehrszeichen zur Begrenzung der Höchstgeschwindigkeit oder die dreieckigen Schilder mit schwarzem Symbol untereinander verwechselt. Daher ist anzunehmen, dass das Modell sich zu stark auf die Form und Farben des Schildes konzentriert. YOLO würde dementsprechend die Parameter anpassen, die sich auf den interessanten Bereich beziehen, das heißt, auf die Mitte des Verkehrszeichen. Auf diese Weise wird das Modell empfindlicher für die Merkmale, die die Zahlen bzw. Symbole auf den verschiedenen Verkehrszeichen unterscheiden. Nach Abschluss jeder Epoche wird die Leistung des Modells anhand der Validierungsbilder bewertet. Diese Bilder wurden beim Training nicht verwendet. Dies hat zur Folge, dass sie sich dazu eignen, um zu beurteilen, wie gut das Modell auf neue Bilder verallgemeinert werden kann. Schließlich gilt das Modell als ausreichend trainiert, wenn seine Leistung bei den Validierungsbildern ein zufriedenstellendes Niveau erreicht. [33]

Bevor das Training in dem vorliegenden Projekt betrachtet wird, sollen ein paar für das Verständnis wichtige Fachbegriffe sowie die genutzte Hardware erläutert werden. Bei den Fachbegriffen handelt es sich um die drei Arten von spezifischen Losses, die es bei YOLO gibt. Der erste ist der `box_loss`. Er misst, wie genau die Bounding Boxen vorhergesagt werden. Als nächstes gibt es den `cls_loss`, was eine Abkürzung für `classification loss` ist. Er bewertet, wie gut die Art der Objekte innerhalb der Bounding Boxen klassifiziert wird. Als drittes muss noch der `dfl_loss` genannt werden, was für `distribution focal loss` steht. Dieser soll die Modelleistung bei unausgewogenen Trainingsdaten verbessern. Das heißt, er bezieht sich auf die ungleiche Verteilung von Klassen in einem Datensatz. [35]

Da der Trainingsprozess je nach verwendeter Hardware sehr lange dauern kann, wird ein möglichst leistungsstarker Jupyter Notebook-Server des iisys (Institut für Informationssysteme) der Hochschule Hof genutzt. Dieser verfügt über eine GPU mit 40 Gigabyte (GB) Speicherkapazität sowie einen gemeinsam genutzten Arbeitsspeicher (englisch: Shared Memory) von ca. 8 GB. Es wird zudem die Open-Source-Deep-Learning-Bibliothek PyTorch Version 2 (Stable), die CUDA-Bibliothek in der Version 11.8 sowie die Programmiersprache Python in der Version 3.10 eingesetzt. [34]

Um nun das Modell für das Projekt zu trainieren, werden lediglich die beiden Zeilen benötigt, die in dem folgenden Listing gezeigt werden. In der oberen Zeile wird dabei festgelegt, von welchem vortrainierten Modell ausgegangen werden soll. Wie bereits erläutert, wird hier das YOLOv8x Modell eingesetzt. Dieses wird durch die erste Zeile in dem Skript automatisch heruntergeladen. Mit dem Aufruf von `train` wird dann in der zweiten Zeile das Training gestartet. Über Parameter wird der Pfad zu der `data.yaml` Datei des Datensatzes sowie die Anzahl der zu durchlaufenden Epochen definiert. Für dieses Projekt wurde entschieden, über 50 Epochen zu trainieren.

```
import ultralytics
model = ultralytics.YOLO('yolov8x.pt')
model.train(data='/Mixed.yolov8/data.yaml', epochs=50)
```

Listing 4. Skript `train.py`

Die untenstehende Tabelle bildet nun ab, wie sich die Verluste und Genauigkeit während des Trainingsprozesses entwickelt haben. Aus Platzgründen werden nicht alle, sondern nur ausgewählte Epochen als Tabellenzeilen mit aufgenommen.

Epoche	box_loss	cls_loss	dfl_loss	mAP50-95
1	0,7178	1,798	1,256	0,38
2	0,5744	0,9383	1,107	0,483
3	0,5682	0,923	1,098	0,369
...	...	...	...	...
38	0,2803	0,3576	0,9715	0,726
39	0,2772	0,3533	0,9711	0,726
40	0,2735	0,3465	0,9687	0,728
41	0,3197	0,1608	1,044	0,73
42	0,3101	0,1545	1,035	0,732
...	...	...	...	...
48	0,2755	0,129	1,007	0,742
49	0,268	0,1245	0,9996	0,744
50	0,2615	0,1204	0,9956	0,74

Tabelle II  
VERLUSTE UND GENAUIGKEIT FÜR AUSGEWÄHLTE EPOCHEN WÄHREND DES TRAININGS

Zunächst soll sich mit den für YOLO spezifischen Verlusten `box_loss`, `cls_loss` und `dfl_loss` beschäftigt werden. Diese werden zusätzlich zu den numerischen Werten in der Tabelle auch in der Grafik 12 visuell abgebildet. Die oberen 3 Diagramme zeigen die Entwicklung dieser Verluste für die Trainingsbilder. Dagegen wird sie in den unteren Diagramme für die Validierungsbilder dargestellt. In den oberen Diagrammen ist zu erkennen, dass die Graphen während des Trainings bis zur 40. Epoche immer weiter sinken. Dann gibt es einen

plötzlichen Sprung. Dieser besteht bei dem `box_loss` und dem `dfl_loss` in einem Anstieg, bei dem `cls_loss` in einem Abfall. Danach nimmt die Kurve bei allen drei Verlusten wieder einen gleichmäßigen Abwärtstrend an. Wirft man nun einen Blick in die Tabelle, so kann man dieselbe Beobachtung feststellen. In den unteren Diagrammen ist hingegen zu sehen, dass die Graphen während der Validierung nach anfänglichen Unregelmäßigkeiten konstant fallen. Das bedeutet, dass die Verluste stetig abnehmen und das Modell immer besser wird. Aus diesem Grund scheint der leichte Anstieg in der 40. Epoche in den oberen Graphen nur ein lokales Minimum zu sein, das ignoriert werden kann.

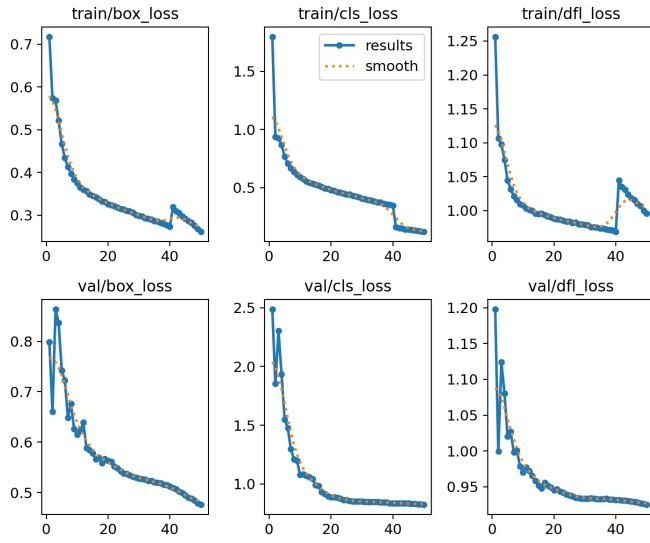


Abbildung 12. Visualisierung der Verluste während des Trainingsprozesses über 50 Epochen

Als zweites soll nun die Präzision des Modells betrachtet werden. Dazu kann erneut die Tabelle II sowie die Abbildung 13 verwendet werden. In Letzterer sind Grafen zu Precision und Recall zu sehen. Zur Bewertung der Genauigkeit von YOLO-Modellen wird jedoch üblicherweise der mAP herangezogen, zu dem ebenfalls Kurven dargestellt sind. Der mAP hat dabei den IoU-Wert 50 sowie 50 bis 95. Es fällt auf, dass die beiden mAP Grafen nahezu identisch aussehen. Daher wird an dieser Stelle wieder hauptsächlich der mAP50-95 betrachtet, der auch eine Spalte in der Tabelle darstellt. Wie zuvor in Kapitel III-C1 erläutert wurde, ist das Modell umso besser im Klassifizieren und Lokalisieren von Objekten, je höher der mAP ist. Da sowohl der Grafik als auch der Tabelle entnommen werden kann, dass der mAP50-95 anfänglich sehr stark und dann schwächer, jedoch weiterhin zunimmt, kann festgestellt werden, dass das Modell mit jeder Epoche immer präziser wird.

Zuletzt soll gesagt werden, dass letztendlich entschieden wird, das Training nach der 50. Epoche nicht weiter fortzusetzen. Der Hauptgrund hierfür ist, dass es von der 40. bis zur 50. Epoche nur noch geringe Verbesserungen bei den Verlusten sowie bei der Genauigkeit gibt. Daher ist nach der 50. Epoche keine signifikante Verbesserung mehr zu erwarten, sodass ein

weiteres Training, welches sehr viel Zeit und Ressourcen in Anspruch nehmen würde, nicht gerechtfertigt ist.

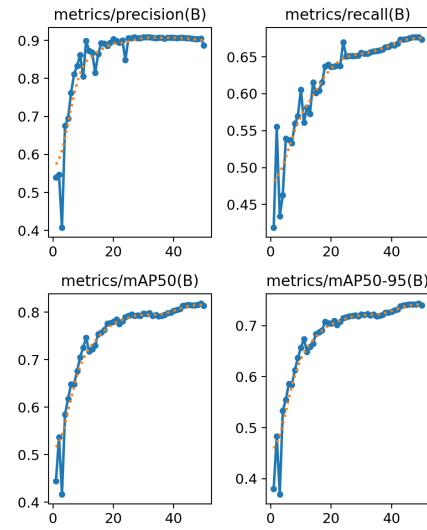


Abbildung 13. Visualisierung der Genauigkeit während des Trainingsprozesses über 50 Epochen

#### D. Implementierung

```
class MachineLearningImageAnalysisDetector
    method initialize(model: TrainedModel)
```

Lade Model

```
method detect(frame: Image):List<ImageDetectionResult>
```

Komprimieren

Vorhersagen

Daten zum Ergebnistyp konvertieren

Daten zurückgeben

Abbildung 14. Grober Aufbau der MachineLearningImageAnalysisDetector Klasse

Als nächstes soll die Implementierung der Klasse `MachineLearningImageAnalysisDetector` betrachtet werden. In deren Initializer muss das trainierte YOLO-Modell übergeben werden. Wie auch schon die Klasse `TraditionalImageAnalysisDetector`, hat die Klasse eine Methode namens `detect`, welche ein Bild annimmt und eine Liste an `ImageDetectionResults` zurückgibt. In dieser wird zuerst das übergebene Bild komprimiert. Im Rahmen dieser Arbeit stellt das Komprimieren kein größeres Problem dar, da die wesentliche Form der Verkehrszeichen auch noch erkennbar ist, wenn die Bilder klein und in

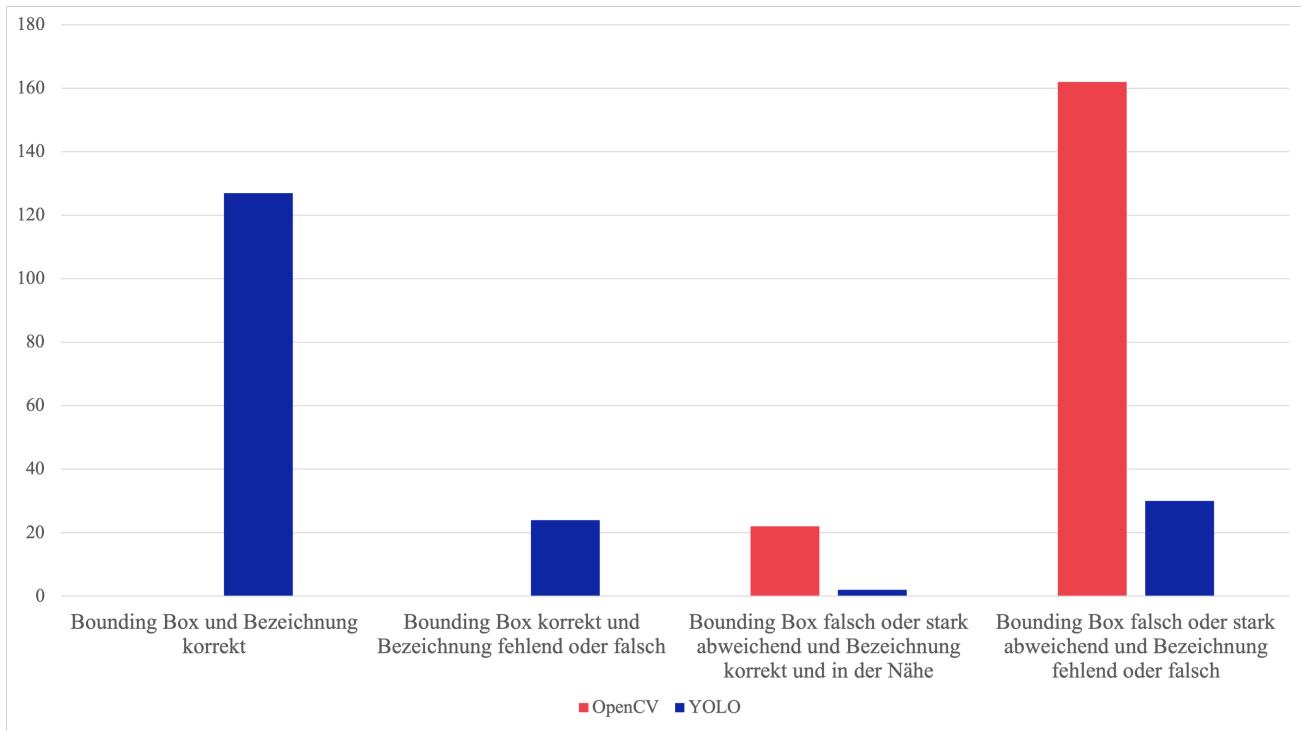


Abbildung 15. Visualisierung des Vergleichs von OpenCV und YOLO mit 139 Testbildern

niedrigerer Auflösung vorliegen. Dies ist dem Fakt geschuldet, dass auch der Gesetzgeber ein großes Interesse daran hat, dass die Verkehrsteilnehmer die verschiedenen Verkehrszeichen aus einer großen Entfernung - also wenn diese noch sehr klein sind - sehen können. Anschließend findet der in dem Kapitel III-A näher erläuterte Vorhersagevorgang mithilfe von YOLO statt. Die erhaltenen Daten werden nun in das Ergebnisformat, nämlich die Liste aus ImageDetectionResult-Objekten, umgewandelt und an den Aufrufer der Methode detect zurückgegeben.

#### IV. VERGLEICH BEIDER ANALYSEMETHODEN

Nachdem die Verkehrszeichenerkennung sowohl mit OpenCV, einem Verfahren zur traditionellen Bildanalyse, als auch mit YOLO, einem Machine Learning Verfahren, umgesetzt wurde, sollen beide Analysemethoden miteinander verglichen werden.

In Kapitel III-B wurde bereits davon gesprochen, dass Datensätze neben Trainings- und Validierungsbildern häufig auch Testbilder beinhalten. Diese dienen dazu, nach Abschluss des Trainingsprozesses anhand neuer, nicht gesehener Daten zu prüfen, wie gut das Modell tatsächlich funktioniert. Darüber hinaus wurde im Verlauf desselben Kapitels erläutert, wie der finale Datensatz zusammengestellt wurde. Dabei wurde auch erwähnt, dass die Datensätze GTSDB und GTSRB, aus denen der finale Datensatz besteht, Testbilder enthielten. Grundsätzlich stehen etwa 270 beiden Analysemethoden noch unbekannte Testbilder zur Verfügung, die nahe und ferne Verkehrszeichen in unterschiedlichen Lichtbedingungen und Schärfegraden zeigen und sich somit für einen realistischen

Vergleich eignen. Problematisch ist, dass ca. 200 der Bilder aus dem GTSRB Datensatz stammen. Für den Vergleich sollen allerdings gleich viele Fotos mit nahen wie mit fernen Schildern genutzt werden. Um ein gleichmäßiges Verhältnis zwischen Bildern aus dem GTSRB und dem GTSDB Datensatz zu erhalten, werden von den ca. 270 Testbildern nur etwa 140 ausgewählt.

Zur Durchführung des Vergleichs sollen beide Objekterkennungsverfahren separat in alle Testbilder einzeichnen, welche Bounding Boxen und welche Verkehrsschildbezeichnungen zu sehen sind. Das Ergebnis sind zwei eigenständige Ordner: Einer enthält die von OpenCV und einer die von YOLO mit Anmerkungen versehenen Testbilder. Die Auswertung erfolgt schließlich manuell, indem jedes annotierte Bild betrachtet und aufgeschrieben wird, inwieweit das jeweilige Analyseverfahren die Verkehrsschilder darin erkennt. Zu erwähnen ist, dass OpenCV für den Vergleich je ein Vorlagenbild jeder der 43 Verkehrsschildklassen erhält, auf die das YOLO Modell trainiert wurde, um zu ermöglichen, dass jede Verkehrszeichenart von beiden Analysemethoden erkannt wird.

Insgesamt wird bei dem Vergleich sehr schnell deutlich, dass Machine Learning die traditionelle Bildanalyse um ein Vielfaches übertrifft. In den 139 Testbildern befinden sich insgesamt 183 Verkehrszeichen der 43 beiden Objekterkennungsverfahren bekannten Klassen. Das Säulendiagramm in Abbildung 15 stellt jeweils für beide Verfahren dar, bei wie vielen Schildern Bounding Box und Verkehrszeichenbezeichnung korrekt erkannt werden, bei wie vielen nur jeweils eines richtig bestimmt wird und bei wie vielen sowohl die Bounding Box als auch

die Bezeichnung falsch ist. Während OpenCV kein einziges Verkehrszeichen lokalisieren kann und nur bei 22 Objekten die korrekte Bezeichnung herausfindet, liegt YOLO bei ungefähr 70 % der Objekte sowohl mit der Bounding Box als auch der Bezeichnung richtig. Bei 24 Objekten erkennt YOLO hingegen lediglich die Bounding Box und zwei Objekte werden zwar richtig klassifiziert, aber nicht lokalisiert. Würde das YOLO Modell noch um einige, beispielsweise 200 Epochen, länger trainiert werden oder würde ein größerer, noch vielseitigerer Datensatz genutzt werden, so würde es vermutlich noch mehr Schilder fehlerfrei lokalisieren und identifizieren. Der Unterschied zwischen den beiden Analyseverfahren würde dann noch deutlicher ausfallen.

Im Folgenden soll auf den Vergleich etwas detaillierter eingegangen werden, indem die häufigsten Fehler beider Verfahren erläutert werden. Zuerst soll die traditionelle Bildanalyse betrachtet werden. Abbildung 16 zeigt dazu eines der Testbilder mit den von OpenCV eingezeichneten Anmerkungen.



Abbildung 16. Das Bild 00740 des GTSDB Datensatzes mit Anmerkungen durch OpenCV

Es kann zuerst allgemein festgestellt werden, dass OpenCV bei keinem der Testbilder auch nur eine konkrete Bounding Box einzeichnet. Stattdessen sind, wie die angesprochene Abbildung zeigt, lediglich manchmal schräge Linien zu sehen, welche jedoch nie ein konkretes Objekt umrahmen. Was die Bezeichnungen der Schilder betrifft, so sind diese meist über das gesamte Foto verstreut. Es fällt jedoch bei nicht wenigen Bildern auf, dass in der Nähe, manchmal direkt unter den zu erkennenden Schildern oft besonders viele – wenn auch häufig fehlerhafte – Bezeichnungen stehen. Daraus kann geschlossen werden, dass OpenCV diese Verkehrszeichen zumindest als relevante Objekte erfasst. Auch in der Abbildung 16 ist zu sehen, dass sich die meisten Verkehrsschildnamen im Bereich um die beiden „Höchstgeschwindigkeit 80“ Schilder befinden. Des Weiteren wechselt OpenCV sehr häufig bestimmte, sich in Form, Farbe oder genutzten Symbolen ähnelnde Schilder untereinander, sodass die Bezeichnungen nicht selten fast richtig sind. Das betrifft insbesondere alle Verkehrszeichen zur Begrenzung der Höchstgeschwindigkeit, alle Schilder, die sich auf ein Überholverbot beziehen, und alle dreieckigen, rot-weißen Verkehrszeichen, die in der Mitte ein schwarzes Sym-

bol aufweisen. Auch diese zweite Beobachtung kann mit dem Bild 16 bewiesen werden, da sich links oberhalb und rechts unterhalb des linken „Höchstgeschwindigkeit 80“ Schildes mehrere Bezeichnungen von Verkehrszeichen zur Begrenzung der Höchstgeschwindigkeit befinden, jedoch leider nicht die exakte.

Zuletzt soll darauf hingewiesen werden, dass OpenCV bei jedem einzelnen Testbild, das aus dem GTSRB-Datensatz stammt, weder die Bounding Box noch die Bezeichnung des gezeigten Objektes bestimmen kann. Dies scheint zunächst verwunderlich zu sein, da die Vorlagenbilder, die OpenCV zur Verfügung stehen, ebenfalls ein einziges Schild aus der Nähe zeigen. Jedoch herrschen bei den Vorlagenbildern stets optimale Bedingungen, was die Schärfe und Lichtverhältnisse betrifft, während die Testbilder fast alle etwas unscharf, verwackelt oder über- bzw. unterbelichtet sind. Somit kann der Schluss gezogen werden, dass die traditionelle Bildanalyse nur bei perfekten Bedingungen Chancen hat, ein Objekt richtig zu erkennen.

Als Nächstes soll nun genauer betrachtet werden, wie maschinelles Lernen bei dem Vergleich abgeschnitten hat. Hierzu soll zuerst auf Abbildung 17 eingegangen werden. Diese zeigt dasselbe Testfoto wie Abbildung 16, jedoch nun mit den von YOLO eingezeichneten Bounding Boxen und Bezeichnungen. Im direkten Vergleich ist klar zu sehen, dass OpenCV im Grunde kein einziges Objekt in dem Foto erkennt, wohingegen YOLO alle drei Schilder eindeutig und fehlerfrei lokalisieren und benennen kann.



Abbildung 17. Das Bild 00740 des GTSDB Datensatzes mit Anmerkungen durch YOLO

Jedoch muss eingeräumt werden, dass auch die Objekterkennung mit YOLO in dem Vergleich nicht einwandfrei funktioniert. So treten auch hier oft Verwechslungen zwischen bestimmten Schilderarten auf. Dazu gehören insbesondere alle Verkehrszeichen zur Begrenzung der Höchstgeschwindigkeit, die Schilder „Arbeitsstelle“ und „Schnee- und Eisglätte“ sowie die Verkehrszeichen „Vorfahrt an der nächsten Kreuzung“, „Kinder“ und „Radverkehr“. Ein Beispiel hierfür stellt Abbildung 18 dar. Wie zuvor in Kapitel III-C2 erläutert wurde, wäre vermutlich ein etwas längerer Trainingsprozess notwendig, um die Mitte der Verkehrsschilder in den Parametern des Modells mehr zu gewichten und das Problem der Irrtümer zu

beheben. Hervorzuheben ist allerdings, dass die Objekte in den Testbildern, die ursprünglich aus dem GTSRB-Datensatz stammen, von dem Machine Learning Verfahren bis auf 2 Schilder korrekt erkannt wurden. Das zeigt, dass das Modell bei nahen Verkehrszeichen bereits sehr gut und nahezu ohne Verwechslungen funktioniert.



Abbildung 18. Das Bild 00802 des GTSDB Datensatzes mit Anmerkungen durch YOLO

## V. UMSETZUNG IN EINER CLIENT-SERVER-ANWENDUNG

### A. Verwendete Technologien

Wie bereits in der Einleitung erwähnt, soll die Verkehrszeichenerkennung mit OpenCV und YOLO nun in eine praktisch nutzbare Client-Server-Anwendung verpackt werden. Bevor diese näher erläutert wird, soll auf die eingesetzten Technologien eingegangen werden.

Für das gesamte Projekt werden ausschließlich die Programmiersprache Python und in Python geschriebene Bibliotheken genutzt. Davon ausgenommen ist die Android-App, für welche die in Java-Bytecode kompilierbare Programmiersprache Kotlin verwendet wird.

### B. Systemarchitektur

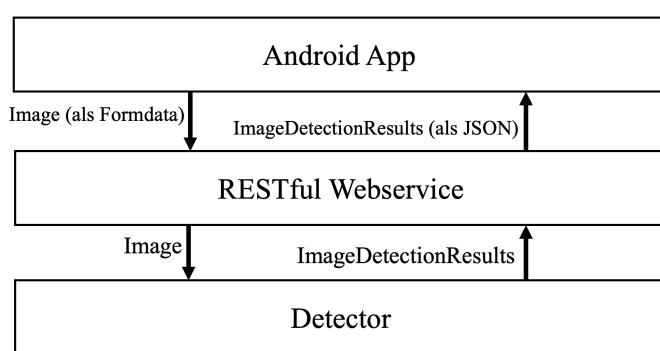


Abbildung 19. Schichten der Systemarchitektur

Im Folgenden soll nun auf die einzelnen Schichten der Systemarchitektur und dabei insbesondere auf deren Schnittstellen eingegangen werden. Zum besseren Verständnis

gibt die nachfolgende Abbildung 19 einen Überblick über die Systemarchitektur. Diese besteht in einer klassischen Schichtenarchitektur und erleichtert somit das Verständnis des Programms sowie das Finden von Fehlern.

**1) Detector-Klassen:** Aus beiden Objekterkennungsverfahren, die im Rahmen dieser Arbeit erläutert wurden, geht jeweils eine Detector-Klasse hervor. Beide haben eine Methode namens detect, welche ein Bild als Übergabeparameter erwartet. Dieses Bild muss in Form eines Objektes des Typs ndarray (aus dem Package numpy) übergeben werden. Wichtig zu erwähnen ist, dass dieses ndarray-Objekt ein zweidimensionaler Array sein muss, wobei an jedem Index ein Array mit der Länge drei sein muss. Diese Arrays stellen die Pixel des übergebenen Bildes im BGR (Blue Green Red) Format ohne Alphawerte dar. Zurück gibt die Methode detect eine Liste aus ImageDetectionResult-Objekten, wobei deren Aufbau bereits in Kapitel II-B gezeigt wurde. Zur Verdeutlichung dieser einheitlichen Schnittstelle erben beide Klassen von der abstrakten Klasse AbstractDetector, welche gerade genannte detect-Method als Implementierungsvorschrift in Form einer abstrakten Methode beinhaltet. Durch diese einheitliche Schnittstelle, können die Detector-Klassen ohne Weiteres an der gleichen Stelle verwendet werden. Diese Detectoren stellen die unterste Schicht der Systemarchitektur dar.

**2) RESTful Webservice:** Auf der eben erläuterten Schicht baut nun ein kleiner RESTful Webservice auf. Dieser beinhaltet die folgenden zwei Endpunkte zur Verkehrszeichenerkennung in einem Bild, wobei der erste OpenCV und der zweite YOLO nutzt.

`POST /traditional-image-analysis /  
image-detection-results`

`POST /machine-learning-image-analysis /  
image-detection-results`

Beiden Endpunkten muss das Bild, in dem die deutschen Verkehrszeichen erkannt werden sollen, unter dem Schlüssel image als FormData geschickt werden. Das Bild als FormData zu schicken ist der performanteste Weg, da keine weitere Zeit beispielsweise durch die Konvertierung in Base64 verloren geht. Als Antwort wird von den Endpunkten eine Liste an ImageDetectionResult-Objekten unter dem Schlüssel imageDetectionResults zurückgesendet.

Nebenbei soll angemerkt werden, dass die Detector-Klassen im vorliegenden Projekt auch in anderen Skripten verwendet werden. Dazu zählt zum Beispiel ein Skript zum Einzeichnen von Bounding Boxen in auf dem Computer gespeicherten Bildern, was insbesondere für den im vorherigen Kapitel beschriebenen Vergleich beider Objekterkennungsverfahren zum Einsatz kommt. Auf diese Skripte soll in dieser Arbeit nicht weiter eingegangen werden, jedoch können diese bei näherem Interesse im Quellcode-Ordner angesehen werden.

3) *Android App*: Auf dem RESTful Webservice baut zuletzt die Android App auf. Beim Senden des Bildes an den RESTful Webservice, muss nun folgendes beachtet werden: Da die Kameravorschau den ganzen Bildschirm ausfüllen soll und nicht jeder Smartphone-Bildschirm das gleiche Seitenverhältnis besitzt, besteht das Problem, dass es sich bei dem Vorschaubild in der App nur um einen Ausschnitt des tatsächlich an den Webservice gesendeten Bildes handelt. Dies führt dazu, dass die vom RESTful Webservice erhaltenen Bounding Boxen an den falschen Stellen im Vorschaubild eingezeichnet werden. Um dies zu beheben, wird jedes zu schickende Bild vor dem Senden an den RESTful Webservice auf den in der App zu sehenden Ausschnitt zugeschnitten.

Beim Erhalt der ImageDetectionResult-Objekte muss darüber hinaus an einen weiteren Aspekt gedacht werden. Sowohl die x- als auch die y-Koordinate der vier Punkte des FrameCoordinates-Objektes befinden sich in der Einheit Pixel. Android arbeitet jedoch ausschließlich mit der pixelunabhängigen Einheit DP (Density-independent Pixels). Bei dem Einzeichnen der Bounding Boxen im Vorschaubild der App müssen die x- und y-Koordinaten folglich zuerst in die Einheit DP umgerechnet werden. [36]

### C. Benutzeroberfläche und Funktionen

Als nächstes sollen die Benutzeroberfläche und die konkrete Funktionalität der Android-App näher betrachtet werden.



Abbildung 20. Startbildschirm der Android-App

In der App findet der Nutzer genau einen Screen vor, welcher direkt eine Kameravorschau zeigt. Werden Verkehrszeichen in dieser erkannt, zeichnet die App um jedes einzelne einen Rahmen und schreibt die erkannte Bezeichnung unter den Rahmen. Zu erwähnen ist, dass bei dem ersten Öffnen der Anwendung erst die Berechtigung zur Verwendung der Kamera des Gerätes gegeben werden muss. Dazu zeigt das Betriebssystem einen eigenen Dialog an, der den Nutzer nach der Erlaubnis zur Nutzung der Kamera fragt. Solange der App noch keine Kameraberechtigung erteilt wurde, ist das Vorschaubild schwarz.

Die Benutzeroberfläche der App ist eher minimalistisch gehalten, um nicht zu sehr von der Kameravorschau abzulenken. Unten rechts auf der Vorschau befinden sich zwei sogenannte Floating Action Buttons (FAB).

Der untere Button beinhaltet ein Icon mit einem Bildschirm, auf dem ein Lautsprecher abgebildet ist. Er dient dazu, mithilfe der TextToSpeech Funktion von Android die erkannten Bezeichnungen der aktuell in der Kameravorschau befindlichen Verkehrsschilder laut vorzulesen. So können visuell beeinträchtigte Menschen mithilfe ihres Smartphones leichter den Alltag im Straßenverkehr bewältigen.



Abbildung 21. Kontextmenü für weitere Optionen

Über dem Button zum Vorlesen der Verkehrszeichen befindet sich ein weiterer Button zum Ändern bestimmter Einstellungen. Das Zahnrad-Icon soll seine Funktionalität verdeutlichen. Er öffnet das in Abbildung 21 zu sehende Kontextmenü. Dessen unterste Option zeigt Informationen über

die Entwickler an. Die Option darüber ermöglicht es, die Adresse des Servers mithilfe eines Dialogs zu ändern (siehe Abbildung 22). Dabei muss die Adresse die IP-Adresse inklusive dem Port sein, auf welchem der RESTful Webservice zu Erkennung der Verkehrszeichen läuft. An diesen Server wird jedes aufgenommene Bild geschickt. Auf Basis der Antworten des Servers werden zum einen die Bounding Boxen in der Kameravorschau eingezeichnet und zum anderen die Schilder-Bezeichnungen bei Klick auf den TextToSpeech-FAB vorgelesen. Die Einstellung der korrekten IP-Adresse und des richtigen Ports ist somit unerlässlich für die fehlerfreie Funktionsweise der Anwendung. Ist die Adresse falsch oder kann aus anderweitigen Gründen keine Verbindungen zum Server hergestellt werden - weil zum Beispiel kein Netz vorhanden ist - werden in der App keine Verkehrszeichen erkannt.

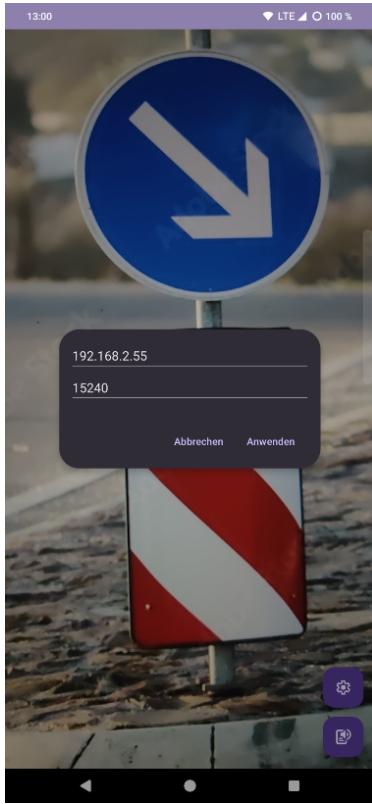


Abbildung 22. Textfelder-Dialog zur Eingabe der Server-Adresse

Die oberste Option des Kontextmenüs ermöglicht es, während des laufenden Betriebs der App über Choice-Boxen in einem Dialog den Objekterkennungsmodus zu ändern (siehe Abbildung 23). Dabei kann zwischen „Traditional Image Detection“ und „Machine Learning Image Detection“ gewechselt werden. Standardmäßig ist Zweitere ausgewählt, da YOLO, wie der Vergleich gezeigt hat, wesentlich besser funktioniert. Je nachdem, für welche Choice-Box der Nutzer sich entscheidet, wird entweder der Endpunkt POST /machine-learning-image-analysis/image-detection-results oder der Endpunkt POST /traditional-image-analysis/image-detection-results des RESTful Webservices angesteuert.

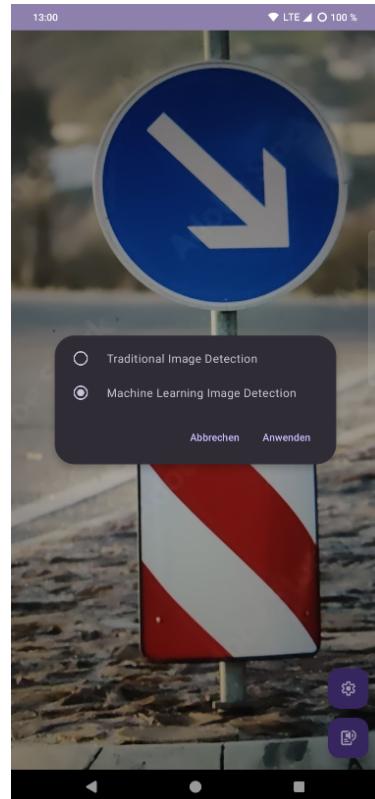


Abbildung 23. Choice-Box-Dialog für den Objekterkennungsmodus

Zuletzt soll erwähnt werden, dass alle in der App eingestellten Werte dauerhaft gespeichert werden. Daher können sie bei dem erneuten Öffnen der App voreingestellt werden und der Nutzer muss sie nicht immer wieder neu eingeben.

#### D. Struktur des Quellcode-Ordners

Dieser Arbeit ist ein Quellcode-Ordner beigefügt. Er beinhaltet unter anderem die zwei Detectoren, aber auch verschiedene weitere Skripte. Zu diesen zählen zum Beispiel die Android App oder die Erkennung der ausgewählten deutschen Verkehrszeichen in der Webcam.

In jeder Datei mit groß geschriebenem Namen befindet sich jeweils eine gleichnamige Klasse. Die Dateien mit klein geschriebener Bezeichnung stellen Skripte dar und nutzen die Klassen aus den Dateien mit groß geschriebenem Namen. Die Android App besteht zwar zwangsläufig aus mehreren Dateien, kann jedoch als Datei mit klein geschriebener Bezeichnung - also als Skript - gesehen werden. Darüber hinaus gibt es den Quellcode-Ordner Ressourcendateien, der z. B. das trainierte Modell für YOLO umfasst. Jede Datei mit groß geschriebenem oder klein geschriebenem Namen und jede Ressourcendatei ist in einem von drei Ordnern eingesortiert. Die Ordner „TraditionalImageAnalysis“ und „MachineLearningImageAnalysis“ sind selbsterklärend. Der Ordner „Commons“ beinhaltet alles, was sowohl „TraditionalImageAnalysis“ als auch „MachineLearningImageAnalysis“ betrifft.

## VI. FAZIT

Im Rahmen dieses Projekts konnte sich ein grundlegender Eindruck darüber verschafft werden, wie Bilder und insbesondere Objekte in diesen analysiert und in neuen Fotos wiedererkannt werden können. Dabei war zunächst interessant zu sehen, wie traditionelle Analyseverfahren wie OpenCV anhand spezifischer Merkmale, z.B. Ecken oder Kanten, Objekte identifizieren können. Unter sehr guten Bedingungen bezüglich Schärfe, Licht und Größe des Objekts im Bild funktioniert dies sogar dann, wenn das Objekt in anderen Perspektiven gezeigt wird. Dennoch war es beeindruckend bei dem durchgeführten Vergleich zu sehen, dass maschinelles Lernen deutlich schneller und zuverlässiger funktioniert. So können oft selbst mehrere, äußerst klein abgebildete Schilder in einem Foto gleichzeitig erkannt werden. Der Grund dafür ist nicht zuletzt, dass YOLO das gesamte Bild auf einmal analysiert, statt schrittweise verschiedene Bereiche, wodurch die Objekte im Kontext und den Beziehungen zueinander betrachtet werden.

Insgesamt kann gesagt werden, dass die im Verlauf des Projektes geschaffene Verkehrszeichenerkennung – insbesondere unter Verwendung von maschinellem Lernen – bereits sehr vielversprechende Ergebnisse erzielt. Diese könnten durch ein umfassenderes Training des Modells, beispielsweise über 300 statt nur 50 Epochen sowie mit einem größeren Datensatz, vermutlich noch weiter optimiert werden.

## LITERATUR

- [1] J. Rudnicka. "Umsatz der wichtigsten Industriebranchen in Deutschland von 2011 bis 2021". Statista. <https://de.statista.com/statistik/daten/studie/203580/umfrage/umsaetze-der-wichtigsten-industriebranchen-in-deutschland/> (abgerufen 14.01.2024).
- [2] U. Bashir. "Einstellungen zu Autos und Mobilität in Deutschland im Jahr 2023". Statista. <https://de.statista.com/prognosen/999887/deutschland-einstellungen-zu-autos-und-mobilitaet> (abgerufen 14.01.2024).
- [3] "Verkehrszeichen und ihre Bedeutung". ADAC. <https://www.adac.de/verkehr/recht/verkehrszeichen/> (abgerufen 14.01.2024).
- [4] "APACHE LICENSE, VERSION 2.0". Apache. <https://www.apache.org/licenses/LICENSE-2.0> (abgerufen 30.01.2024).
- [5] "About". OpenCV. <https://opencv.org/about/> (abgerufen 30.01.2024).
- [6] "Understanding Features". OpenCV. [https://docs.opencv.org/4.x/d5/tutorial\\_py\\_features\\_meaning.html](https://docs.opencv.org/4.x/d5/tutorial_py_features_meaning.html) (abgerufen 30.01.2024).
- [7] "ORB (Oriented FAST and Rotated BRIEF)". OpenCV. [https://docs.opencv.org/3.4/d1/d89/tutorial\\_py\\_orb.html](https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html) (abgerufen 03.02.2024).
- [8] "Bildtafel der Verkehrszeichen in der Bundesrepublik Deutschland seit 2017". Wikipedia. [https://de.wikipedia.org/wiki/Bildtafel\\_der\\_Verkehrszeichen\\_in\\_der\\_Bundesrepublik\\_Deutschland\\_seit\\_2017#](https://de.wikipedia.org/wiki/Bildtafel_der_Verkehrszeichen_in_der_Bundesrepublik_Deutschland_seit_2017#) (abgerufen 03.02.2024).
- [9] "Verkehrsschild 142 – Achtung Wildwechsel". Landesjagdverband Schleswig-Holstein. <https://ljv-sh.de/shop/schilder-poster-plakate/verkehrsschild-142-achtung-wildwechsel/> (abgerufen 03.02.2024).
- [10] "OpenCV". Github. <https://github.com/opencv> (abgerufen 03.02.2024).
- [11] "ultralytics/functions-matlab". Github. <https://github.com/ultralytics/functions-matlab> (abgerufen 03.02.2024).
- [12] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.
- [13] "Home". Ultralytics YOLOv8 Docs. <https://docs.ultralytics.com/> (abgerufen 16.01.2024).
- [14] "Welcome to the INI Benchmark Website!". INI. <https://benchmark.ini.rub.de/> (abgerufen 17.01.2024).
- [15] "International Joint Conference on Neural Networks (IJCNN)". International Neural Network Society. <https://www.inns.org/ijcnn-home> (abgerufen 17.01.2024).
- [16] "The German Traffic Sign Detection Benchmark". INI. [https://benchmark.ini.rub.de/gtsdb\\_news.html](https://benchmark.ini.rub.de/gtsdb_news.html) (abgerufen 17.01.2024).
- [17] "The German Traffic Sign Recognition Benchmark". INI. [https://benchmark.ini.rub.de/gtsrb\\_news.html](https://benchmark.ini.rub.de/gtsrb_news.html) (abgerufen 17.01.2024).
- [18] "Object Detection Datasets Overview". Ultralytics YOLOv8 Docs. <https://docs.ultralytics.com/datasets/detect/> (abgerufen 18.01.2024).
- [19] "Traffic Signs Dataset in YOLO format". Kaggle. <https://www.kaggle.com/datasets/valentynsichkar/traffic-signs-dataset-in-yolo-format/> (abgerufen 18.01.2024).
- [20] "Wikimedia Commons". Wikimedia Commons. <https://commons.wikimedia.org/> (abgerufen 20.01.2024).
- [21] "Objekt-Erkennung". Ultralytics YOLOv8 Docs. <https://docs.ultralytics.com/de/tasks/detect/> (abgerufen 20.01.2024).
- [22] T.Y. Lin et al., "Microsoft COCO: Common Objects in Context", Computer Vision – ECCV 2014, Zurich, Switzerland, 2014, vol. 8693, Lecture Notes in Computer Science, pp. 740-755, doi: 10.1007/978-3-319-10602-1\_48.
- [23] "Was ist Transfer Learning?". aws. <https://aws.amazon.com/de/what-is/transfer-learning/> (abgerufen 20.01.2024).
- [24] R. Yamashita, M. Nishio, R. K. G. Do and K. Togashi, "Convolutional neural networks: an overview and application in radiology", Insights into Imaging, vol. 9, pp. 611–629, Aug. 2018, doi: 10.1007/s13244-018-0639-9.
- [25] VK. "YoloV8 Architecture & Cow Counter With Region Based Dragging Using YoloV8". Medium. [https://medium.com/@VK\\_Venkatkumar/yolov8-architecture-cow-counter-with-region-based-dragging-using-yolov8-e75b3ac71ed8](https://medium.com/@VK_Venkatkumar/yolov8-architecture-cow-counter-with-region-based-dragging-using-yolov8-e75b3ac71ed8) (abgerufen 20.01.2024).
- [26] "Ultralytics YOLOv5 Architektur". Ultralytics YOLOv8 Docs. [https://docs.ultralytics.com/de/yolov5/tutorials/architecture\\_description/](https://docs.ultralytics.com/de/yolov5/tutorials/architecture_description/) (abgerufen 20.01.2024).
- [27] "What are the differences between YOLOv8-det and YOLOv8-seg of network structures". Github. <https://github.com/ultralytics/ultralytics/issues/6224> (abgerufen 20.01.2024).
- [28] J. Hui, "mAP (mean Average Precision) for Object Detection". Medium. <https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173> (abgerufen 23.01.2024).
- [29] "NVIDIA A100 Tensor Core-GPU". NVIDIA. <https://www.nvidia.com/de-de/data-center/a100/> (abgerufen 23.01.2024).
- [30] "NVIDIA TensorRT". NVIDIA DEVELOPER. <https://developer.nvidia.com/tensorrt> (abgerufen 23.01.2024).
- [31] "What is ONNX?". UbiOps. <https://ubiops.com/what-is-onnx/> (abgerufen 23.01.2024).
- [32] "Ultralytics YOLOv8 Modes". Ultralytics YOLOv8 Docs. <https://docs.ultralytics.com/modes/> (abgerufen 24.01.2024).
- [33] "Model Training with Ultralytics YOLO". Ultralytics YOLOv8 Docs. <https://docs.ultralytics.com/modes/train/> (abgerufen 24.01.2024).
- [34] "Server Options". Jupyterhub. <https://jupyterhub.kiawz.iisys.de/hub/spawn> (abgerufen 24.01.2024).
- [35] "Are class and box losses calculated the same in YoloV8 and YoloV5?". Github. <https://github.com/ultralytics/ultralytics/issues/2789> (abgerufen 24.01.2024).
- [36] "Support different pixel densities". Android Developers. <https://developer.android.com/training/multiscreen/screendensities#TaskUseDP> (abgerufen 03.02.2024).