**RASPBERRY PI SMART CAR WITH REAL-TIME TRAFFIC SIGNS AND SIGNALS RECOGNITION**

**SISTEME ÎNCORPORATE**

Fermendjin Patricia-Maria-Francesca

Gaube Robert-Gabriel

Murariu Marius-Andrei

2023 – 2024

1. **Project Description**

This project aims to develop an automated image processing system mounted on a smart car, designed to recognize traffic signs and signals using a Raspberry Pi and an attached camera. The car will take the due actions when encountering various supported signs.

An image processing Python script operates in real-time based on a YOLOv8 deep learning model. The smart car utilizes neural networks for the recognition of various traffic signs such as "Stop", "50 speed limit", "70 speed limit”, and red / green traffic light etc.

The functionality will be demonstrated using small-scale models of traffic signs and traffic lights.

For each recognized traffic sign or traffic light colour, a corresponding message will be displayed in the terminal and the appropriate action will be performed. For instance, the smart car will change the speed accordingly when encountering a specific speed limit sign. Also, it will stop when recognizing a red traffic signal and continue forward when the light changes to green. The smart car will halt completely when seeing a stop sign.

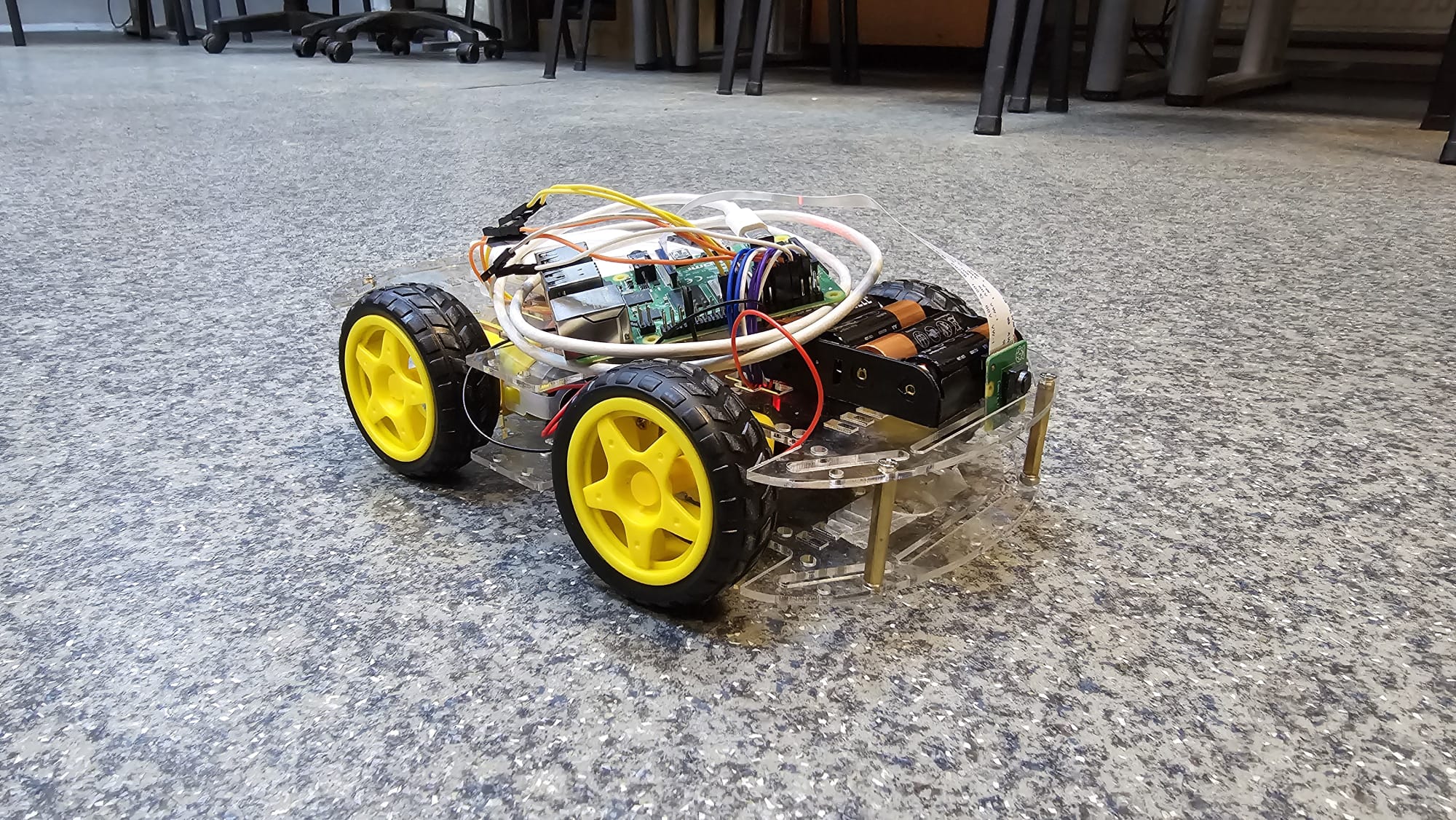
****

Figure 1. Raspberry Pi Smart Car.

1. **Raspberry Pi 4B**

Raspberry Pi is a single-board computer designed as a low-cost, credit-card sized computer that can be used for a variety of applications, including robotics projects.

* 1. **General Specifications**
     1. Hardware
* Quad core 64-bit ARM-Cortex A72 running at 1.5 GHz
* 8 GB LPDDR4 RAM
* H.265 (HEVC) hardware decode (up to 4Kp60)
* H.264 hardware decode (up to 1080p60)
* VideoCore VI 3D Graphics
* Supports dual HDMI display output up to 4Kp60.
  + 1. Interfaces
* 802.11 b/g/n/ac Wireless LAN
* Bluetooth 5.0 with BLE
* 1x SD Card
* 2x micro-HDMI ports supporting dual displays up to 4Kp60 resolution
* 2x USB2 ports
* 2x USB3 ports
* 1x Gigabit Ethernet port (supports PoE with add-on PoE HAT)
* 1x Raspberry Pi camera port (2-lane MIPI CSI)
* 1x Raspberry Pi display port (2-lane MIPI DSI)
* 28x user GPIO supporting various interface options: up to 6x UART, up to 6x I2C, up to 5x SPI, 1x SDIO interface, 1x DPI (Parallel RGB Display), 1x PCM, up to 2x PWM channels, up to 3x GPCLK outputs.
  + 1. Software
* ARMv8 instruction set
* Raspberry Pi OS Lite, 64 bits, 12.03.2024 release.
  + 1. Power Requirements

The Pi4B requires a USB-C power supply capable of delivering 5V at 3A.

* 1. **Board Schematics**

A blueprint of a computer

Description automatically generated

Figure 2.1. Raspberry Pi 4B physical specifications.

* 1. **GPIO Interface**

The Pi4B makes 28 BCM2711 GPIOs available via a standard Raspberry Pi 40-pin header. This header is backwards compatible with all previous Raspberry Pi boards with a 40-way header.

A green circuit board with different colors of text

Description automatically generated

Figure 2.2. GPIO pinout.

As well as being able to be used as straightforward software-controlled input and output (with programmable pulls), GPIO pins can be switched into various other modes backed by dedicated peripheral blocks such as I2C, UART and SPI.

* 1. **Raspberry Pi Camera**

The Pi4B has 1x Raspberry Pi 2-lane MIPI CSI Camera connector.

The Raspberry Pi board has been fitted with a Raspberry Pi Camera V2  
8-megapixel camera able to take photographs of up to 3280 x 2464 pixels and video at 1080p30, 720p60 and 480p90 resolutions. This camera contains a Sony IMX219 image sensor featuring a fixed focus lens.

A computer screen shot of a computer screen

Description automatically generated

Figure 2.2. Raspberry Pi Camera V2 specifications.

1. **Raspberry Pi Smart Car**

The Raspberry Pi Smart Car is composed of a double-layered chassis,  
4 wheels, 4 DC motors, 2 L298N dual h-bridge motor drivers, a 4 AA battery holder, a Raspberry Pi 4B, an 8-megapixel Raspberry Pi Camera V2, and a 10000 mAh power bank. The hardware schematic is presented in Appendix B.

* 1. **DC Plastic Gearbox Motors**

The smart car is fitted with 4 DC plastic gearbox motors (TT motors), each powering one of the wheels.

Technical specifications:

* Rated Voltage: 3 ~ 6 V
* Continuous No-Load Current: 150mA ± 10%
* Min. Operating Speed (3V): 90 ± 10% RPM
* Min. Operating Speed (6V): 200 ± 10% RPM
* Stall Torque (3V): 0.4 kg x cm
* Stall Torque (6V): 0.8 kg x cm
* Gear Ratio: 1:48
* Body Dimensions: 70 x 22 x 18 mm
* Wires Length: 200mm & 28 AWG
* Weight: 30.6g

A drawing of a mechanical device

Description automatically generated

Figure 2.3. DC plastic gearbox motor dimension diagram.

Since the DC motors don’t possess an inbuilt speed and rotation direction control system, they must be controlled programmatically using pulse-width modulation (PWM), thereby controlling the motor’s behaviour using the input voltage, which is directly proportional to the motor’s rotation speed. Likewise, the rotation direction depends on the input voltage polarity.

PWM is a type of modulation in which the width of the pulse is changed without changing the signal’s frequency. Consequently, the motor’s speed can be changed by altering the duty cycle of the PWM signal due to the proportionality between the speed and the on-time of the pulse. This allows for an instant analog control of a DC motor via a digital signal generated by the Raspberry Pi computer.

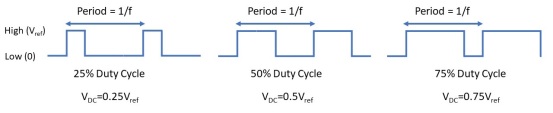


Figure 2.4. DC value (VDC) of a PWM signal for various duty cycle values.  
The signal’s period remains constant.

* 1. **L298N Dual H-bridge DC Motor Driver**

As stated above, the behaviour of a DC motor can be controlled by using a pulse-width modulated signal. However, this cannot be done directly since the Raspberry Pi GIPO pin voltage is too low to drive the motor.

Still, the desired control voltage can be achieved by interposing an H-bridge between the Raspberry Pi and the DC motor.

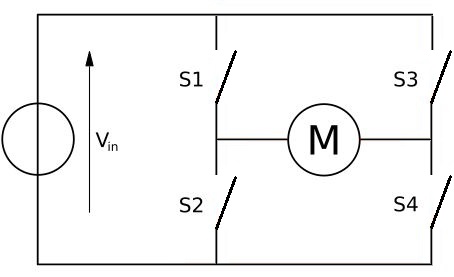


Figure 2.5. H-bridge motor connection.

The direction of the motor’s rotation can be changed by individually controlling the four switches. For instance, if S1 and S4 were closed, the motor would turn in a particular direction, and if S2 and S3 were closed, the direction would be opposite.

This effect can be obtained by using the L298 Dual H-Bridge dual bidirectional motor driver, an integrated circuit able to independently control two DC motors both in speed and direction.

L298N dual h-bridge motor driver technical specifications:

* Input Voltage: DC 3.2 V - 40 V
* Power Supply: DC 5 V - 35 V
* Peak current: 2 Amp
* Operating current range: 0 ~ 36 mA
* Control signal input voltage range: low: -0.3V ≤ Vin ≤ 1.5V; high: 2.3V ≤ Vin ≤ Vss.
* Enable signal input voltage range (active-high): low: -0.3 ≤ Vin ≤ 1.5V; high: 2.3V ≤ Vin ≤ Vss.
* Maximum power consumption: 20W (for T = 75 ℃).
* Storage temperature: -25 ℃ ~ +130 ℃.
* Size: 3.4 cm x 4.3 cm x 2.7 cm

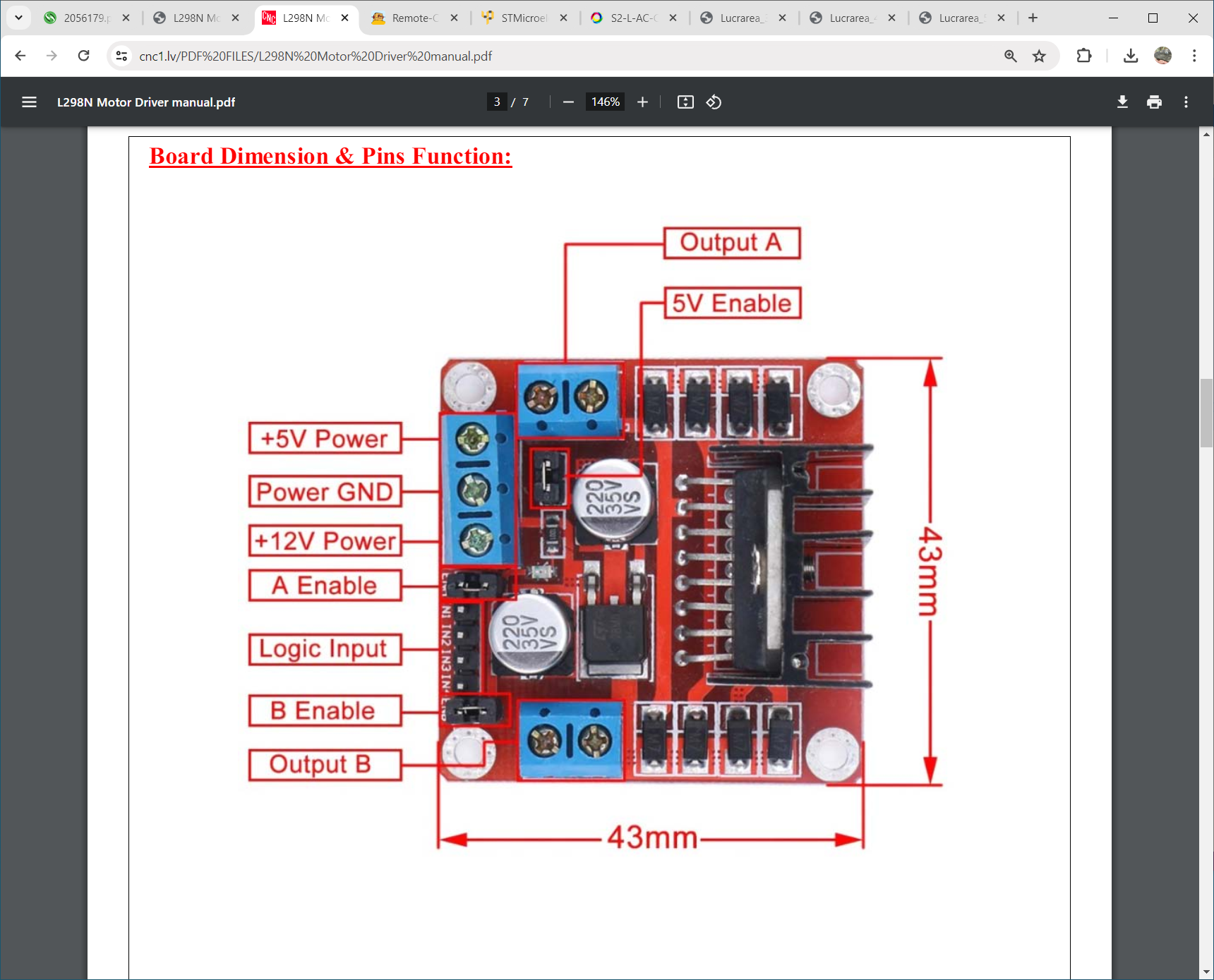


Figure 2.6. L298N board and pins.

1. **Real-time Traffic Signs and Signals Detection**
   1. **Computer Vision**

Computer vision is a field of artificial intelligence (AI) that enables computers to process visual inputs such as digital images and videos. This is achieved by automatically analysing vast amounts of data, recognizing certain patterns and distinctions, and ultimately obtaining a model able to extract and classify various features.

The model’s ability to distinguish different objects is acquired through technologies such as deep learning (a subset of machine learning) and convolutional neural networks.

A convolutional neural network (CNN) is a deep learning algorithm that receives an image as input and differentiates various aspects of the image by assigning importance degrees through learnable weights and biases. This process generally involves multiple convolution layers that identify the dependencies between pixels in order to successively create more meaningful representations of the data. A CNN may at first translate pixels into lines, which are then combined to form features, such as a traffic light’s visor, and finally combined to create the more complex signal head.

A yellow and black logo

Description automatically generated

Figure 3.1. A CNN identifies increasingly complex features with each layer.

* 1. **YOLO**

YOLO (You Only Look Once) is a deep learning model based on CNNs designed for real-time object detection in computer vision applications. Essentially, a convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. Also, YOLO trains on full images and directly optimizes detection performance.

YOLOv8 incorporates various object detection methods, including classification, object detection, and image segmentation, providing pixel-level information about each object. The model’s architecture comprises a backbone, a neck and a head. The backbone focuses on feature extraction, the neck aggregates and refines the features, while the head produces bounding boxes, class probabilities, and confidence scores.

The YOLOv8 architecture (fig. 3.2) utilizes a Feature Pyramid Network to gradually reduce the spatial resolution of the input image while increasing the number of feature channels, allowing the detection of objects with different scales and resolutions.

A diagram of a computer

Description automatically generated

Figure 3.2. YOLOv8 Architecture.

* 1. **Model Training**

The purpose of our computer vision application is to correctly identify traffic signs and signals in a video stream collected in real time from the Raspberry Pi’s camera. To satisfy these requirements, we have trained a custom deep learning model, thus obtaining a reliable road sign detection solution using YOLOv8.

The dataset that the model has been trained and tested on is available at https://shorturl.at/fuvBL, consisting of 5771 images and their associated labels. For training purposes, the dataset has been split in the following manner: 3520 images for training (61.15%), 1603 images for validation (27.77%), and 638 images for testing (6.37%).

After a comparative analysis of the model’s performance in different scenarios, a satisfactory precision rate has been achieved by training the model in 100 epochs with batches of 64 images. Additionally, the experimentally determined hyperparameters are an initial learning rate of 0.0001 and a dropout rate of 15%. The resulting YOLOv8 model is composed of 168 layers and contains 3,008,573 parameters.

The following objects are currently supported by the model: red light, green light, stop sign, speed limit 10, speed limit 20, speed limit 30, speed limit 40, speed limit 50, speed limit 60, speed limit 70, speed limit 80, speed limit 90, speed limit 100, speed limit 110, and speed limit 120.

A collage of different signs

Description automatically generated

Figure 3.3. Image samples from the training dataset.

The following Python libraries were used: matplotlib, numpy, opencv-python, pandas, seaborn, ultralytics.

The complete code is provided in Appendix A (chapter 6.1).

The hardware used for model training is a laptop with an Intel Core i9-13980HX CPU and an NVIDIA GeForce RTX 4090 graphics card.

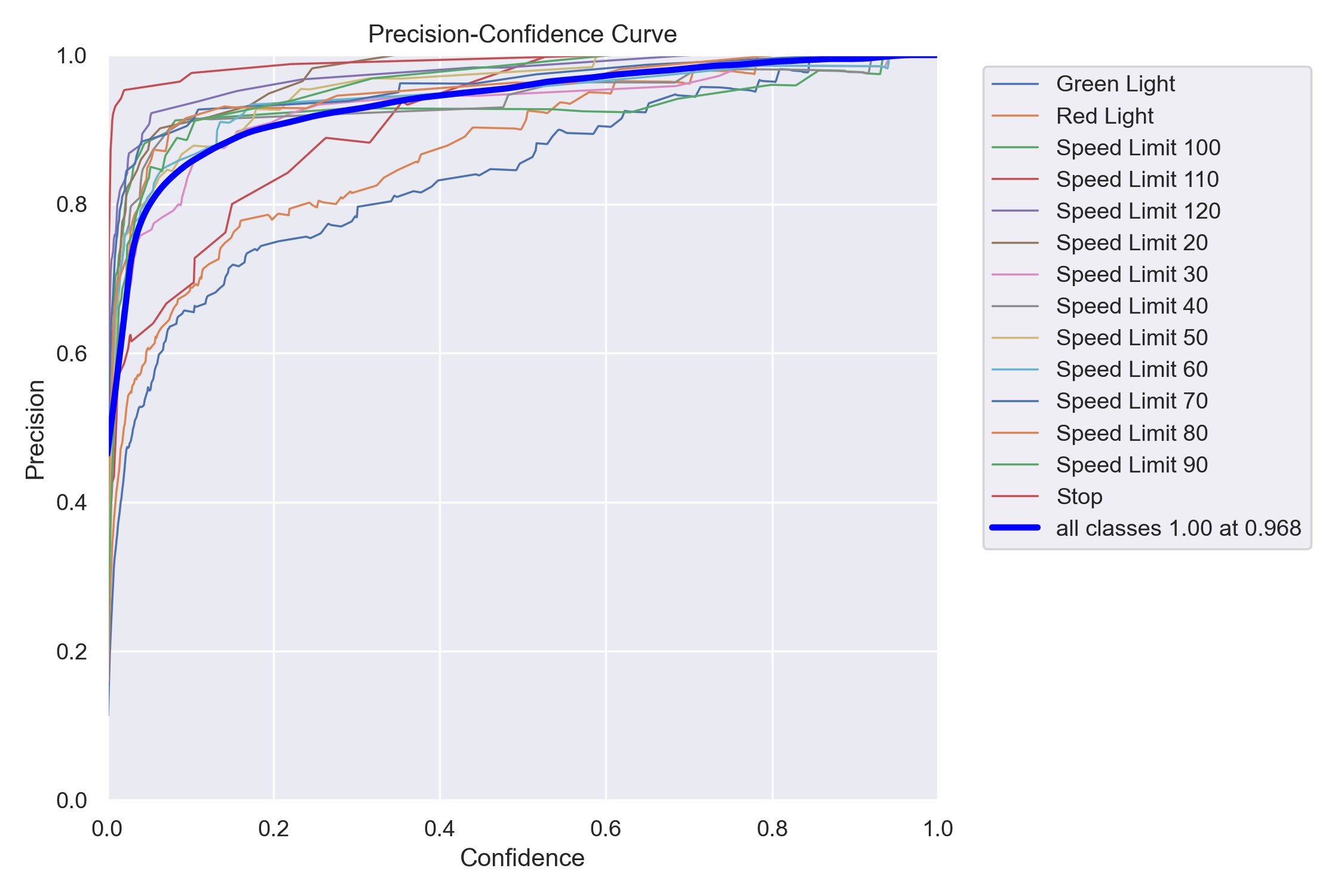
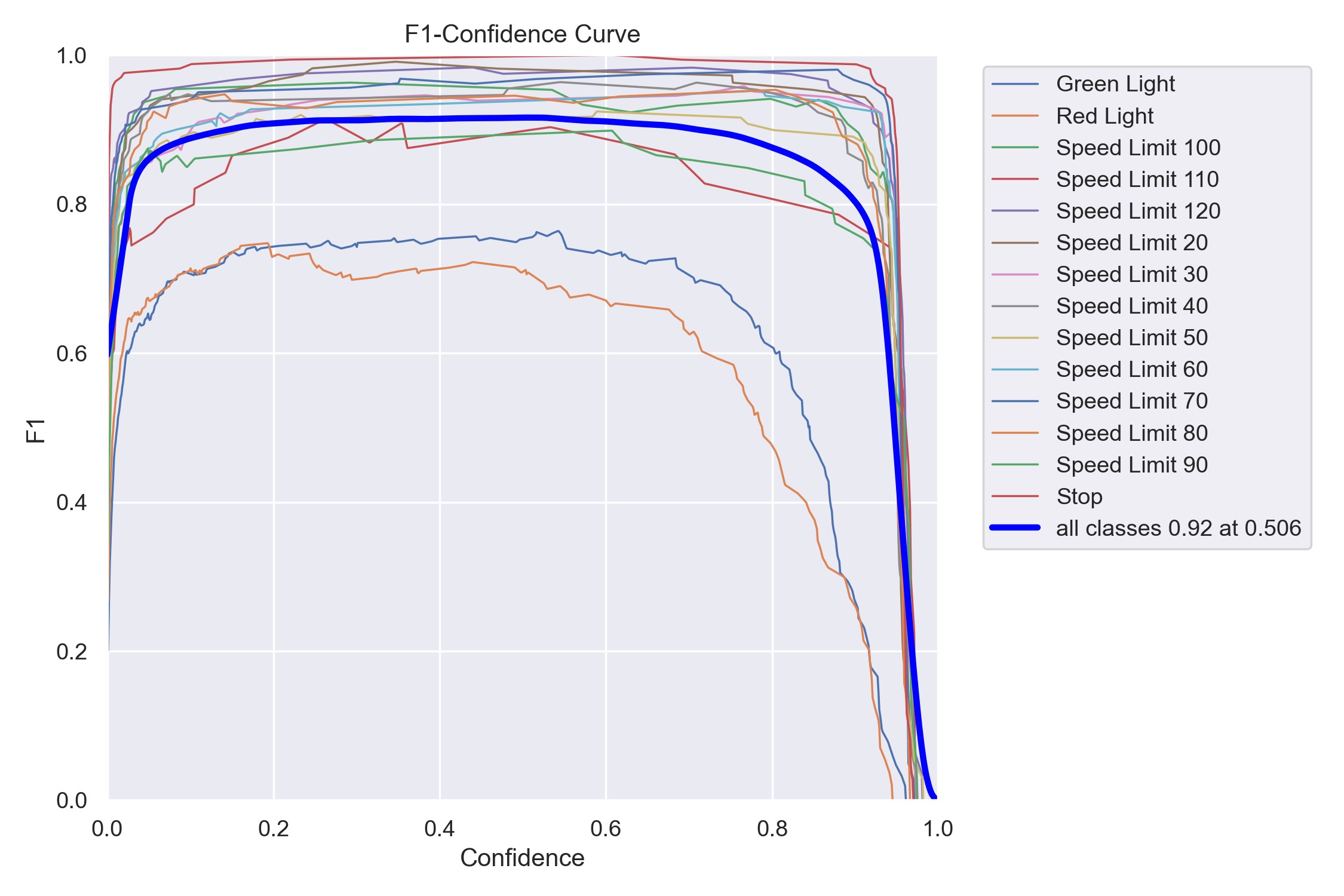
* 1. **Model Testing**

Based on the testing results, the overall precision of the model is 96.1213%.

A graph of a graph of a function

Description automatically generated with medium confidence

Figure 3.4. The model’s precision and loss values on the training and validation sets.



(a) F1 Confidence Curve. (b) Precision-Confidence Curve.

Figure 3.5. Model confidence metrics.

A screenshot of a video

Description automatically generated

Figure 3.6. Sample test output with boxed traffic signs and signals.

1. **References**
2. J. Redmon, S. Divvala, R. Girshick and A. Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*, IEEE Xplore, 2016.
3. D. Reis, J. Kupec, J. Hong, A. Daoudi, *Real-Time Flying Object Detection with YOLOv8*, Journal of Imaging, 2023.
4. S. Saha, *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*, Medium, 2018. Available: <https://towardsdatascience.com/a-comprehensive-guide-toconvolutional-neural-networks-the-eli5-way-3bd2b1164a53>. [Accessed: 22/04/2024]
5. *What is computer vision?*, IBM. Available: <https://www.ibm.com/topics/computer-vision>. [Accessed: 22/04/2024]
6. J. Sawtell-Rickson, *What Is Computer Vision*, Builtin, 2022. Available: <https://builtin.com/machine-learning/computer-vision>. [Accessed: 22/04/2024]
7. *Under the Hood: YOLOv8 Architecture Explained*, Keylabs, 2023. Available: <https://keylabs.ai/blog/under-the-hood-yolov8architecture-explained> [Accessed: 22/04/2024]
8. <https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf> [Accessed: 27/04/2024]
9. <https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-mechanical-drawing.pdf> [Accessed: 27/04/2024]
10. <https://randomnerdtutorials.com/raspberry-pi-pinout-gpios/> [Accessed: 27/04/2024]
11. <https://www.farnell.com/datasheets/2056179.pdf> [Accessed: 27/04/2024]
12. <https://www.farnell.com/cad/2623733.pdf> [Accessed: 27/04/2024]
13. <https://www.adafruit.com/product/3777> [Accessed: 27/04/2024]
14. <https://blog.upverter.com/2019/11/21/using-pwm-for-dc-motor-control/> [Accessed: 27/04/2024]
15. **Appendix A – Code**
    1. **YOLOv8 Training and Testing Script**

# Import Essential Libraries

import os

import random

import cv2

from ultralytics import YOLO

import matplotlib.pyplot as plt

import seaborn as sns

sns.set(style='darkgrid')

import warnings

warnings.filterwarnings('ignore')

# Build from YAML and transfer weights

Final\_model = YOLO('yolov8n.yaml').load('yolov8n.pt')

# Training the model

Result\_Final\_model = Final\_model.train(

    data="/Users/Marius/Desktop/images/data.yaml",epochs=100,

    imgsz = 416, batch = 64, lr0=0.0001, dropout= 0.15,

    device=0)

# Display the model's metrics

print(Result\_Final\_model)

# Display the model's metrics, such as precision and losses

# on the train and validation sets

import df

# Read the results.csv file as a pandas dataframe

Result\_Final\_model.columns = df.columns.str.strip()

# Create subplots

fig, axs = plt.subplots(nrows=5, ncols=2, figsize=(15, 15))

# Plot the columns using seaborn

sns.lineplot(x='epoch', y='train/box\_loss', data=df, ax=axs[0,0])

sns.lineplot(x='epoch', y='train/cls\_loss', data=df, ax=axs[0,1])

sns.lineplot(x='epoch', y='train/dfl\_loss', data=df, ax=axs[1,0])

sns.lineplot(x='epoch', y='metrics/precision(B)', data=df, ax=axs[1,1])

sns.lineplot(x='epoch', y='metrics/recall(B)', data=df, ax=axs[2,0])

sns.lineplot(x='epoch', y='metrics/mAP50(B)', data=df, ax=axs[2,1])

sns.lineplot(x='epoch', y='metrics/mAP50-95(B)', data=df, ax=axs[3,0])

sns.lineplot(x='epoch', y='val/box\_loss', data=df, ax=axs[3,1])

sns.lineplot(x='epoch', y='val/cls\_loss', data=df, ax=axs[4,0])

sns.lineplot(x='epoch', y='val/dfl\_loss', data=df, ax=axs[4,1])

# Set titles and axis labels for each subplot

axs[0,0].set(title='Train Box Loss')

axs[0,1].set(title='Train Class Loss')

axs[1,0].set(title='Train DFL Loss')

axs[1,1].set(title='Metrics Precision (B)')

axs[2,0].set(title='Metrics Recall (B)')

axs[2,1].set(title='Metrics mAP50 (B)')

axs[3,0].set(title='Metrics mAP50-95 (B)')

axs[3,1].set(title='Validation Box Loss')

axs[4,0].set(title='Validation Class Loss')

axs[4,1].set(title='Validation DFL Loss')

plt.suptitle('Training Metrics and Loss', fontsize=24)

plt.subplots\_adjust(top=0.8)

plt.tight\_layout()

plt.show()

# Model testing

# Loading the best performing model

Valid\_model = YOLO('/Users/Marius/Desktop/runs/detect/train6/weights/best.pt')

# Evaluating the model on the testset

metrics = Valid\_model.val(split = 'test')

# Final results

print("precision(B): ", metrics.results\_dict["metrics/precision(B)"])

print("metrics/recall(B): ", metrics.results\_dict["metrics/recall(B)"])

print("metrics/mAP50(B): ", metrics.results\_dict["metrics/mAP50(B)"])

print("metrics/mAP50-95(B): ", metrics.results\_dict["metrics/mAP50-95(B)"])

* 1. **Raspberry Pi Car Operations**

import cv2

import RPi.GPIO as GPIO

from time import sleep, time

from ultralytics import YOLO

# Config variables

speed = 10

# Setup GPIO settings

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

# Give ports to every motor

(Ena1, In11, In12) = (2, 3, 4)

(Ena2, In21, In22) = (17, 27, 22)

(Ena3, In31, In32) = (10, 9, 11)

(Ena4, In41, In42) = (14, 15, 18)

# Create a list for ports from the same category

enables = [Ena1, Ena2, Ena3, Ena4]

lefts = [In11, In21, In31, In41]

rights = [In12, In22, In32, In42]

pwms = []

# Setup all ports as out

for (enable, left, right) in zip(enables, lefts, rights):

    GPIO.setup(enable, GPIO.OUT)

    GPIO.setup(left, GPIO.OUT)

    GPIO.setup(right, GPIO.OUT)

# Setup the enables

for enable in enables:

    pwm = GPIO.PWM(enable, speed)

    pwm.start(0)

    pwms.append(pwm)

going\_forward = False

# Smooth transition going and stopping

def set\_motor\_speed(speed):

    for pwm in pwms:

        pwm.ChangeDutyCycle(speed)

def gradient\_descent(step):

    global speed

    for i in range(7):

        set\_motor\_speed(speed)

        speed += step

        sleep(0.05)

# Setup electricity flow

def set\_direction():

    for left in lefts:

        GPIO.output(left, GPIO.HIGH)

    for right in rights:

        GPIO.output(right, GPIO.LOW)

def forward():

    set\_direction()

    global going\_forward

    if not going\_forward:

        gradient\_descent(10)

    going\_forward = 1

def stop():

    global going\_forward

    if going\_forward:

        gradient\_descent(-10)

    going\_forward = 0

    set\_motor\_speed(0)

# Stop before running the program

stop()

# Loading model and defining labels

model = YOLO('weights/best.pt')

labels = {

    0: 'Green Light',

    1: 'Red Light',

    2: 'Speed Limit 10',

    3: 'Speed Limit 100',

    4: 'Speed Limit 110',

    5: 'Speed Limit 120',

    6: 'Speed Limit 20',

    7: 'Speed Limit 30',

    8: 'Speed Limit 40',

    9: 'Speed Limit 50',

    10: 'Speed Limit 60',

    11: 'Speed Limit 70',

    12: 'Speed Limit 80',

    13: 'Speed Limit 90',

    14: 'Stop',

    }

# Getting access to the camera

vid = cv2.VideoCapture(0)

print('Camera ready')

# Doing a dummy read and infer

(ret, frame) = vid.read()

flipped = cv2.flip(cv2.flip(cv2.resize(frame, dsize=(416, 416)), 0), 1)

results = model.predict(flipped, verbose=False, stream=True)

is\_at\_red\_light = False

while True:

    if not is\_at\_red\_light:

        forward()

    # Read image and infer

    (ret, frame) = vid.read()

    flipped = cv2.flip(cv2.flip(cv2.resize(frame, dsize=(416, 416)), 0), 1)

    results = model.predict(flipped, half=False, verbose=False)

    # Process results

    for result in results:

        box = result.boxes

        if len(box.cls) != 0 and len(box.xyxy.tolist()[0]) == 4 and float(box.conf[0]) > 0.75:

            label = labels[int(box.cls[0])]

            x1, y1, x2, y2 = [int(res) for res in

                                box.xyxy.tolist()[0]]

            # Result printing

            print('Found ', label, ' at ', x1, y1, x2, y2, ' and confidence ', float(box.conf[0]))

            print('Procent area covered ', ((x2 - x1) \* (y2 - y1)) / (416 \* 416), '\n')

            # Sign based behaviour

            if label == 'Stop':

                while True:

                    stop()

            elif label == 'Red Light':

                is\_at\_red\_light = True

                stop()

            elif label == 'Green Light':

                is\_at\_red\_light = False

            elif label == 'Speed Limit 70' and not is\_at\_red\_light:

                set\_motor\_speed(100)

            elif label == 'Speed Limit 50' and not is\_at\_red\_light:

                set\_motor\_speed(80)

1. **Appendix B – Hardware schematics**

