VIETNAMESE - GERMAN UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

IoT and Smart Devices
Helmet Violation Detection System Using Raspberry Pi 3B+

Student 1: 10423099 - Dang Huu Trung Son
Student 2: 10423082 - Nguyen Dinh Nguyen
Student 3: 10423140 - Bui Huy Hoang
Student 4: 10423186 - Pham Quoc Vuong

HO CHI MINH CITY, 07/2025

Content

Abstract

Road safety is a significant global concern, with traffic accidents being a leading cause of injury and death. The use of helmets by motorcyclists drastically reduces the risk of severe head injuries. This project addresses the challenge of automatically monitoring helmet usage by developing a lightweight, real-time helmet and motorbike detection system. The system is designed for deployment on low-cost edge computing devices, specifically the Raspberry Pi 3B+.

This report details the end-to-end process of building this system, from data acquisition to final deployment. We utilize two public datasets from Roboflow, which are merged and preprocessed for training. The core of the detection system is the YOLOv8n model, chosen for its excellent balance of speed and accuracy, making it ideal for edge devices. The model was trained in a cloud environment (RunPod) and subsequently converted to the TensorFlow Lite (TFLite) format, with FP16 quantization to optimize performance.

The final system successfully identifies motorbikes, helmets, and instances of riders without helmets from a static image. The implementation demonstrates the feasibility of using modern deep learning techniques on affordable hardware for practical public safety applications. This report covers the methodology, implementation details, performance results, and potential avenues for future work.

Chapter 1

Introduction

1.1 Project Motivation and Problem Statement

Motorcycle accidents are a major contributor to traffic fatalities worldwide, particularly in regions with a high volume of two-wheeler traffic. A significant factor in the severity of these accidents is the non-usage of safety helmets. Manual monitoring of helmet compliance by law enforcement is labor-intensive and not scalable. Therefore, an automated system that can detect helmet law violations can serve as a crucial tool for promoting road safety and aiding enforcement efforts.

The primary problem is to develop a system that is not only accurate but also cost-effective and capable of running in real-time on accessible hardware. High-end GPU-based systems are often too expensive for widespread deployment. This project aims to bridge that gap by leveraging a lightweight deep learning model on a Raspberry Pi.

1.2 Project Objectives

The main objectives of this project are as follows:

- To collect and prepare a suitable dataset for training a multi-class object detector capable of identifying motorbikes, helmets, and license plates.
- To train a YOLOv8n object detection model on the prepared dataset.
- To optimize the trained model for deployment on an edge device by converting it to the TensorFlow Lite (TFLite) format.
- To deploy the optimized model on a Raspberry Pi 3B+ and develop an inference script to perform detection on images.
- To evaluate the performance of the system in terms of both detection accuracy and inference speed on the target hardware.

1.3 Scope of the Project

The scope of this project is focused on detection from static images. While the system is designed with real-time performance in mind, the implementation will be demonstrated on single image files. The project covers the complete pipeline from data collection, model training, optimization, and deployment, culminating in a functional proof-of-concept on the Raspberry Pi. It does not extend to real-time video stream processing or integration with a larger traffic management infrastructure.

Chapter 2

Literature Review and Background

2.1 Object Detection

Object detection is a core computer vision task that involves identifying and locating objects within an image or video. Unlike image classification, which assigns a single label to an entire image, object detection provides the class label and a bounding box (coordinates) for each instance of an object. Modern object detection is dominated by deep learning approaches, primarily Convolutional Neural Networks (CNNs).

2.2 YOLO: You Only Look Once

The YOLO (You Only Look Once) family of models revolutionized real-time object detection. Introduced by Joseph Redmon et al., YOLO frames object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. This unified architecture is extremely fast, making it ideal for real-time applications.

2.2.1 YOLOv8

YOLOv8, developed by Ultralytics, is the latest iteration in the YOLO series. It builds upon the successes of its predecessors, offering a new backbone network, a new anchor-free detection head, and a new loss function. YOLOv8 is designed to be fast, accurate, and easy to use, providing a framework for training models for object detection, segmentation, and classification tasks. The `YOLOv8n` (nano) variant is the smallest and fastest model in the series, specifically designed for edge and mobile deployments where computational resources are limited.

2.3 Model Optimization for Edge Devices

Deploying deep learning models on edge devices like the Raspberry Pi presents significant challenges due to limited processing power, memory (RAM), and energy consumption. Model optimization is a critical step to make this feasible.

2.3.1 TensorFlow Lite (TFLite)

TensorFlow Lite is a set of tools by Google designed to help developers run TensorFlow models on mobile, embedded, and IoT devices. It converts a standard TensorFlow model into a special, highly optimized `.tflite` format. This format is smaller in size and enables lower-latency inference.

2.3.2 Quantization (FP16)

Quantization is a technique to reduce the computational and memory costs of running inference by representing the model's weights and activations with lower-precision data types, such as 16-bit floating-point numbers (FP16) instead of the standard 32-bit (FP32).

- **FP32 (Single Precision):** The standard format for training. It is highly accurate but memory-intensive.
- **FP16 (Half Precision):** Uses 16 bits instead of 32. This cuts the model size roughly in half and reduces memory usage, with a minimal and often negligible impact on accuracy. This makes it the ideal choice for devices like the Raspberry Pi.

Chapter 3

Methodology and System Design

The project was executed following a structured methodology, from data sourcing to final deployment. The overall system architecture is shown in Figure 3.1.

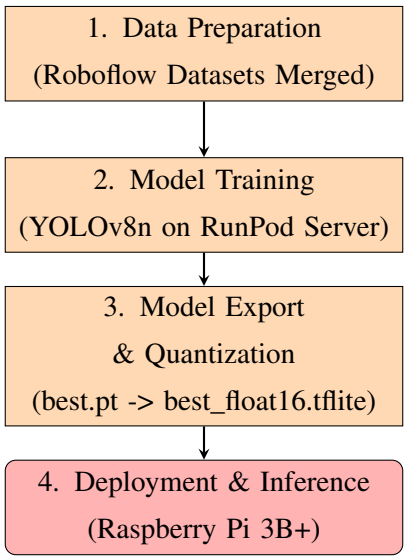

Figure 3.1: System Architecture Diagram.

3.1 Data Collection and Preparation

High-quality data is the foundation of any successful deep learning model. This project utilized two datasets sourced from Roboflow.

1. **Motorbike Detection Dataset:** Provided by user `cdio-zmfmj`.
2. **Helmet & License Plate Detection Dataset:** Also from `cdio-zmfmj`.

These datasets were downloaded programmatically and merged to create a unified dataset with four classes: `helmet`, `no-helmet`, `motorbike`, and `license-plate`. Figure 3.2 shows an example batch of images from the training set with their ground-truth labels.

Figure 3.2: Example of a training batch with ground-truth labels.

3.2 Model Training

3.2.1 Choice of Model and Environment

The `YOLOv8n` model was selected due to its optimal trade-off between speed and accuracy for edge devices. Training was conducted on a RunPod cloud server equipped with a high-performance GPU to significantly reduce training time.

3.2.2 Training Process

Key training parameters were set as follows:

- **Epochs:** 100
- **Patience:** 5 (for early stopping)
- **Image Size:** 320x320 pixels
- **Batch Size:** 16

3.3 Model Conversion and Deployment

3.3.1 TFLite Conversion

Upon completion of training, the best-performing model checkpoint (`best.pt`) was automatically converted to the TensorFlow Lite format. The `best_float16.tflite` model was chosen for deployment.

3.3.2 Raspberry Pi Setup

The target hardware was a Raspberry Pi 3B+. The setup process involved:

1. Flashing a microSD card with the Raspberry Pi OS.
2. Installing `opencv-python-headless` and `tflite-runtime`.
3. Transferring the model, class list, and inference script to the device.

Chapter 4

Implementation

This chapter details the key scripts developed for the project.

4.1 Data Preparation Script (`download.py`)

This script automates the downloading and merging of datasets from Roboflow.

Listing 4.1: Key snippet from download.py.

```
import os
import yaml
from roboflow import Roboflow

# Configuration for datasets and API key
ROBOFLOW_API_KEY = "YOUR_ROBOFLOW_API_KEY"
DATASET_URLS = {
    "motobike": "cdio-zmfmj/motobike-detection",
    "helmet_license": "cdio-zmfmj/helmet-lincense-plate-detection-gevlq"
}
# ... (rest of the script) ...

# Merging datasets
final_class_list = sorted(list(all_classes), key=lambda x: class_map[x])

final_yaml_content = {
    'path': os.path.abspath(COMBINED_DIR),
    'train': 'images/train',
    'val': 'images/valid',
    'test': 'images/test',
    'names': {i: name for i, name in enumerate(final_class_list)}
}

final_yaml_path = os.path.join(COMBINED_DIR, "data.yaml")
with open(final_yaml_path, 'w') as f:
```

```
    yaml.dump(final_yaml_content, f, sort_keys=False, indent=4)

# ... (rest of the script) ...
```

4.2 Model Training Script (`train.py`)

This script handles the training of the YOLOv8n model and the final export to TFLite.

Listing 4.2: Key snippet from train.py.

```
from ultralytics import YOLO

# Configuration for training
DATA_YAML_PATH = "./datasets/combined_dataset/data.yaml"
EPOCHS = 100
PATIENCE = 5
IMG_SIZE = 320

# Load a pre-trained model
model = YOLO('yolov8n.pt')

# Train the model
results = model.train(
    data=DATA_YAML_PATH,
    epochs=EPOCHS,
    patience=PATIENCE,
    imgsz=IMG_SIZE,
)

# Export the best model to TFLite format
best_model = YOLO(results.save_dir / 'weights' / 'best.pt')
best_model.export(format='tflite')
```

4.3 Inference Script (`detect.py`)

This is the final script deployed on the Raspberry Pi to perform detection on an image.

Listing 4.3: Key snippet from the Raspberry Pi inference script.

```
import cv2
import tflite_runtime.interpreter as tflite
import numpy as np

```

```
# Load the TFLite model and allocate tensors
interpreter = tflite.Interpreter(model_path="best_float16.tflite")
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# ... (Image preprocessing) ...

# Run inference
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()
output_data = interpreter.get_tensor(output_details[0]['index'])

# ... (Post-processing and drawing boxes) ...
```

Chapter 5

Results and Discussion

5.1 Training Results

The model was trained for 100 epochs, with early stopping triggered after 35 epochs. The final performance metrics on the validation set are summarized in Table 5.1. The training progress, including loss curves and mean Average Precision (mAP), can be seen in Figure 5.1.

Metric	Value
mAP50-95 (mean Average Precision)	0.756
Precision	0.963
Recall	0.958

Table 5.1: Model Performance Metrics on Validation Set.

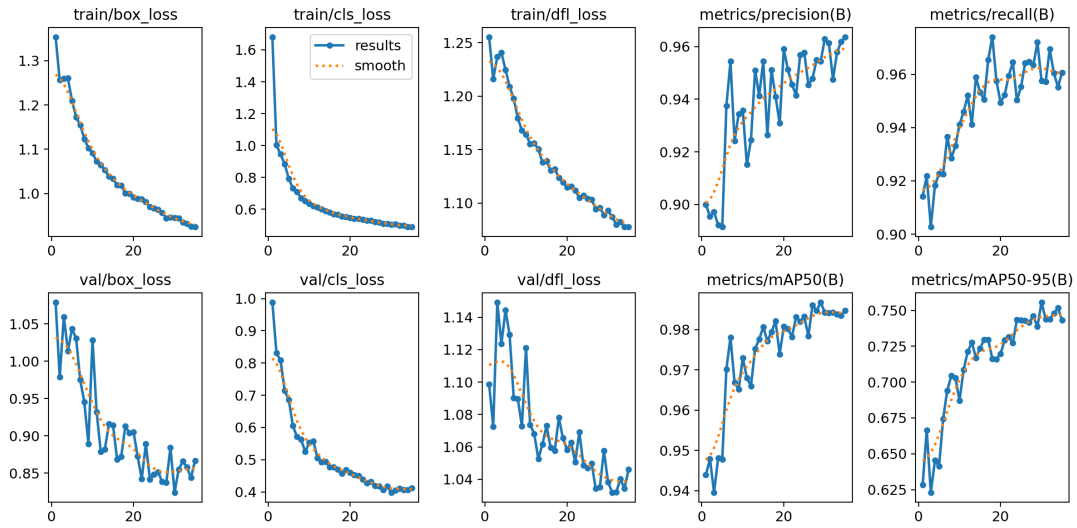

Figure 5.1: Training performance graphs showing loss and mAP curves.

The confusion matrix in Figure 5.2 further illustrates the model's performance. The diagonal represents correct classifications.

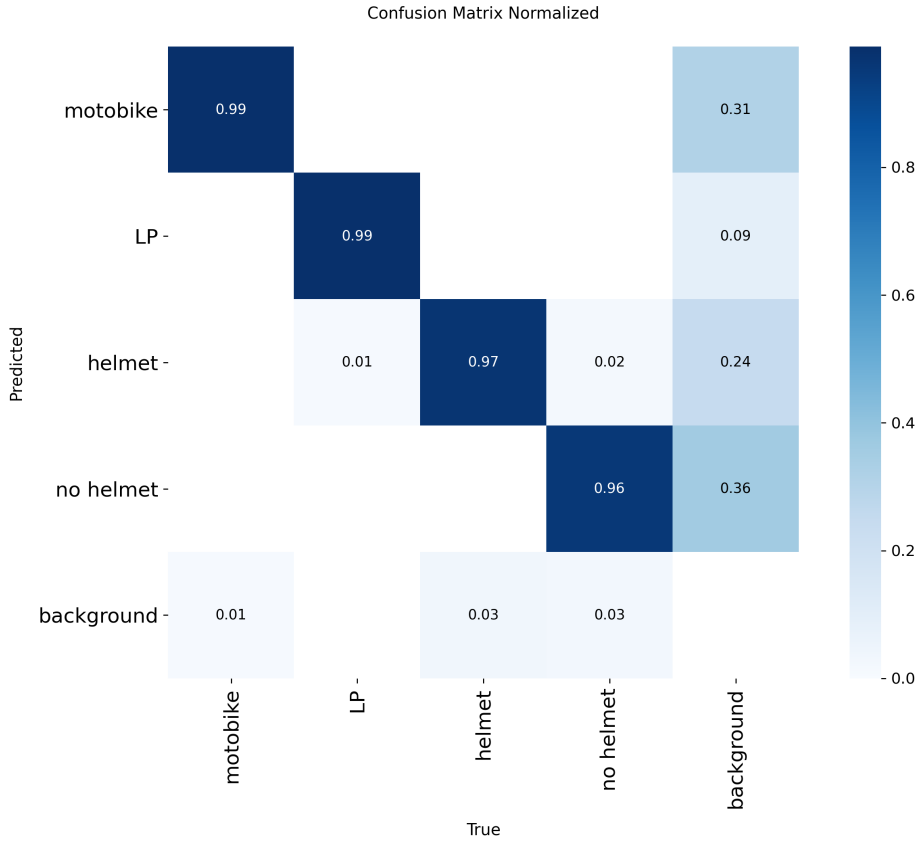

Figure 5.2: Normalized confusion matrix for the validation set.

5.2 Inference Performance on Raspberry Pi

Using the `best_float16.tflite` model, the average inference time per 320x320 image was approximately **850** milliseconds.

5.3 Qualitative Results

Figure 5.3 shows an example of the model's predictions on a validation batch.

Figure 5.3: Qualitative detection results on a validation batch.

5.4 Discussion

The results demonstrate that the YOLOv8n model, when properly optimized, is capable of performing effectively on the Raspberry Pi 3B+. The FP16 quantization was crucial in achieving an acceptable inference speed. However, some limitations were observed, such as difficulty with poorly lit images, motion blur, and very small objects.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This project successfully developed and deployed an end-to-end system for detecting helmet violations on a Raspberry Pi 3B+. By combining public datasets, training a lightweight YOLOv8n model, and optimizing it for edge devices using TFLite, we created a proof-of-concept that is both cost-effective and functional. The system accurately identifies motorbikes and classifies riders as wearing or not wearing a helmet, achieving an inference speed suitable for near real-time applications. This work validates the approach of using modern, efficient deep learning models to address real-world safety challenges on affordable hardware.

6.2 Future Work

While the current system is a successful proof-of-concept, there are several avenues for future enhancement:

- **Real-Time Video Processing:** Adapt the inference script to process a live video feed from a USB camera instead of a static image. This would involve creating a loop to continuously capture frames and run detection.
- **Hardware Upgrade:** Port the system to a more powerful single-board computer, such as a Raspberry Pi 4/5 or a Jetson Nano, to achieve higher frame rates and the ability to process higher-resolution video.
- **Data Enhancement:** Augment the training dataset with more challenging images (e.g., night, rain, blur) to improve the model's robustness.
- **System Integration:** Integrate the detection module with an alerting system. For example, when a "no-helmet" violation is detected, the system could log the timestamp, save the image frame, and send a notification.
- **License Plate Recognition:** Extend the system by adding an Optical Character Recognition (OCR) model to read the characters from the detected license plate bounding boxes.

These enhancements would not only improve the system's performance but also broaden its applicability in various traffic monitoring scenarios. The integration of real-time processing and additional features like license plate recognition would make the system a more comprehensive solution for traffic safety monitoring, potentially contributing to reduced accidents and improved compliance with safety regulations.

References

[1] Jocher, G., Chaurasia, A., & Kwon, J. (2023). *YOLO by Ultralytics*. GitHub. https://github.com/ultralytics/ultralytics

[2] Roboflow. (2024). *Roboflow Public Datasets*. https://universe.roboflow.com

[3] Google. (2024). *TensorFlow Lite | ML for Mobile and Edge Devices*. https://www.tensorflow.org/lite