TECHNICAL REPORT ON REAL-TIME ROAD ANOMALY DETECTION ON ARM-BASED EDGE DEVICE

SUBMITTED BY
PAUL ROJER R
BRINDHA S
SRIRAM A

Real-Time Road Anomaly Detection on ARM-Based Edge Device

Deployment Platform: Raspberry Pi 4 (ARM Cortex-A72)

1. Abstract

This report presents the end-to-end development of a real-time road anomaly detection system optimized for ARM-based edge deployment.

The objective is to design a lightweight yet accurate object detection system capable of running on Raspberry Pi 4 without GPU acceleration.

The system detects potholes, cracks, speed bumps, pedestrians, and unexpected obstacles using a YOLOv8 Nano deep learning architecture.

Model optimization, ONNX export, CPU-based inference tuning, and benchmarking strategies are detailed extensively.

2. Introduction

Edge AI has emerged as a transformative technology enabling real-time decision-making without cloud dependency.

Road anomaly detection is a critical safety application for smart transportation systems.

Deploying deep learning models on constrained ARM processors presents challenges including limited compute, thermal constraints, and memory limits.

This project addresses those challenges through architecture selection, optimization strategies, and efficient inference design.

3. Literature Review

Traditional road inspection relies on manual surveys or expensive LiDAR-based systems.

Cloud-based AI systems introduce latency and require continuous internet connectivity.

Recent research focuses on lightweight CNN models for embedded inference.

YOLO-based detectors have demonstrated strong real-time performance in embedded systems.

4. System Architecture Overview

The system architecture consists of video acquisition, preprocessing, inference engine, post-processing, and visualization modules.

Dashcam video is captured and converted into frames.

Frames are resized and normalized before being passed into the ONNX inference engine.

Bounding boxes are rendered and FPS is calculated in real time.

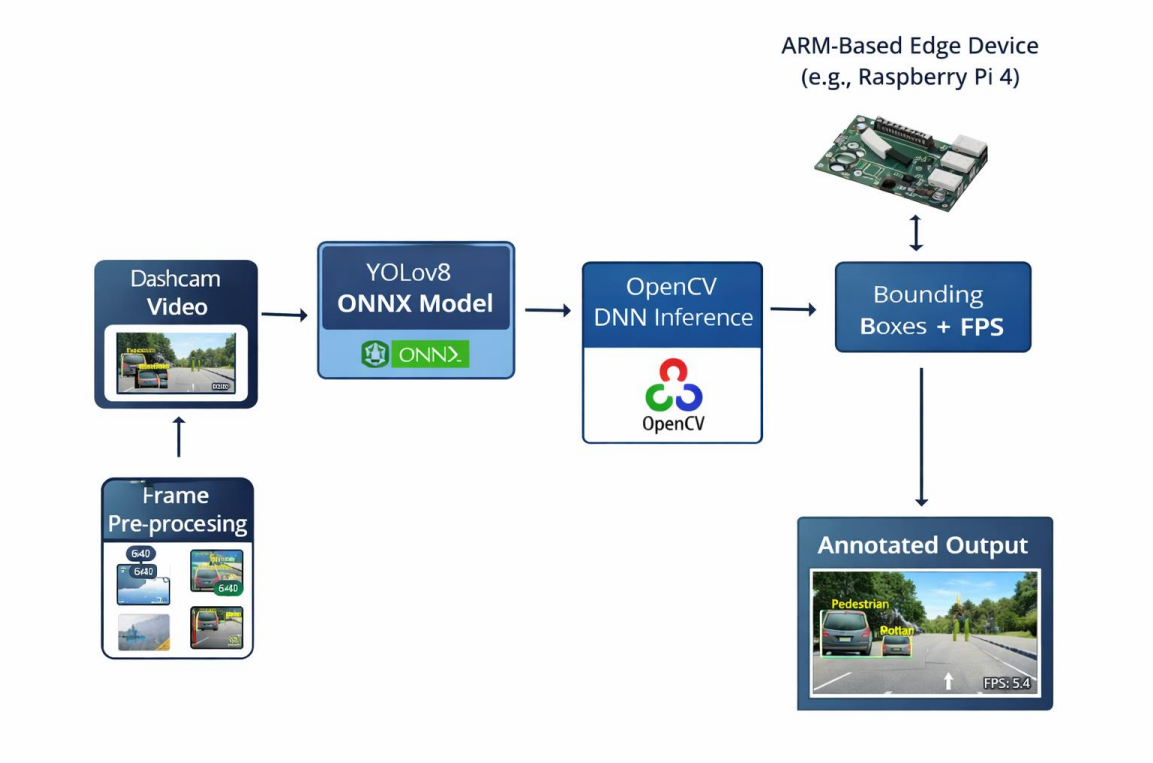


Fig 4.1: System Architecture of the ARM-Based Real-Time Road Anomaly Detection Pipeline Deployed on Raspberry Pi 4

5. Dataset Preparation

Total dataset size: 13,500 annotated images.

Classes include potholes, surface cracks, speed bumps, pedestrians, and obstacles.

Annotation format follows YOLO labeling standards.

Data augmentation techniques: HSV augmentation, mosaic augmentation, scaling, translation, horizontal flipping.

Dataset split: 80% training, 10% validation, 10% testing.

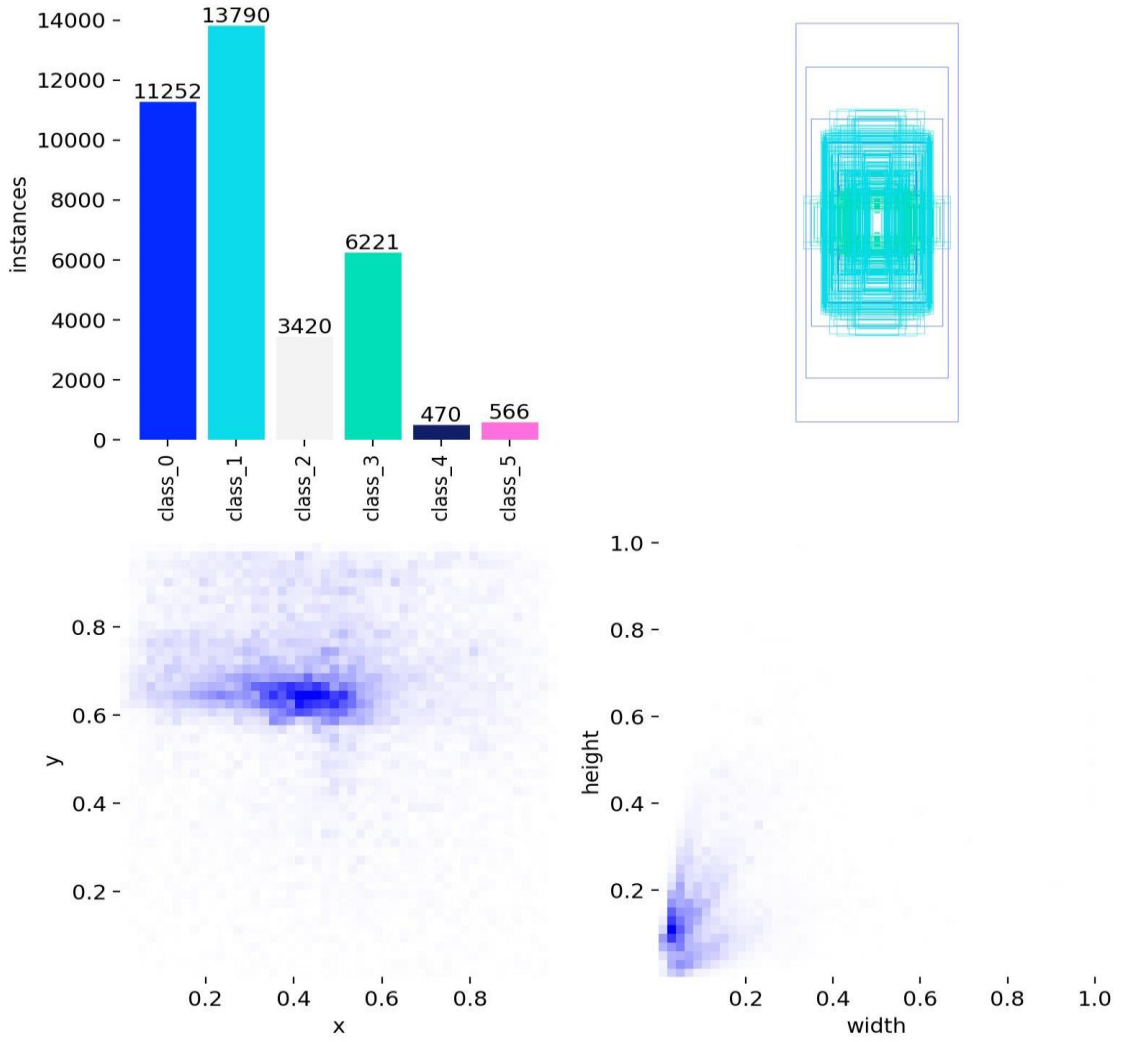


Fig 5.1: Distribution of annotated object classes in the training dataset, illustrating the frequency of each road anomaly category.

6. YOLOv8 Architecture Explanation

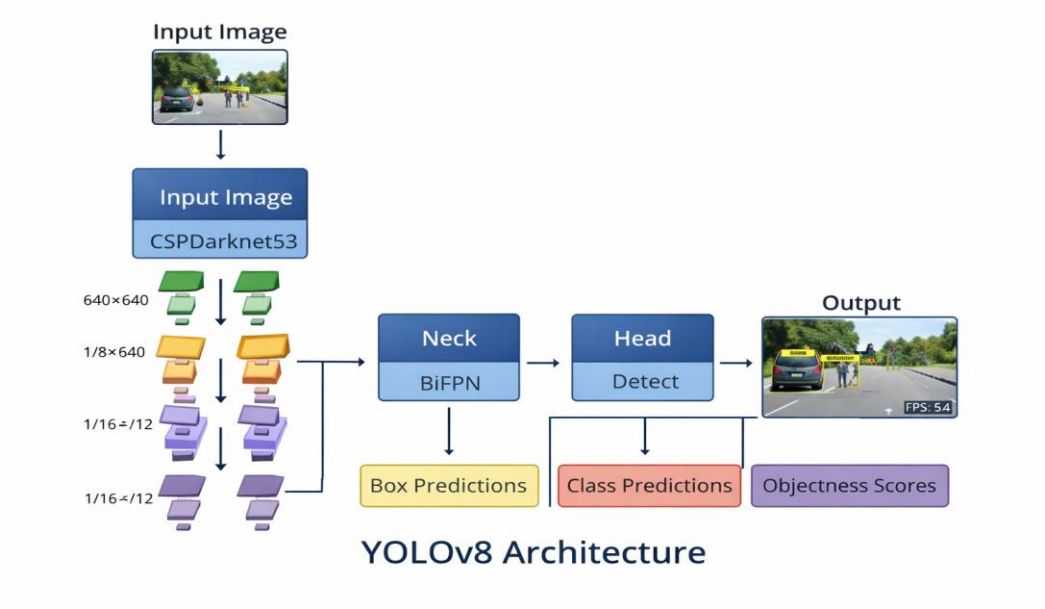
YOLOv8 Nano consists of three main components: backbone, neck, and detection head.

The backbone extracts spatial features using convolutional layers.

The neck performs feature aggregation using PAN-like connections.

The detection head predicts bounding boxes, objectness scores, and class probabilities.

Loss functions include bounding box regression loss, objectness loss, and classification loss.



7. Mathematical Background

Bounding box regression minimizes localization error using IoU-based loss.

Classification loss is computed using cross-entropy.

Total loss = Localization Loss + Objectness Loss + Classification Loss.

Non-Maximum Suppression (NMS) removes redundant overlapping detections using IoU thresholding.

8. Training Configuration

Base Model: YOLOv8n pretrained weights.

Epochs: 80

Image size: 640x640

Batch size: 8

Learning rate: 0.003

Training performed on CPU environment.

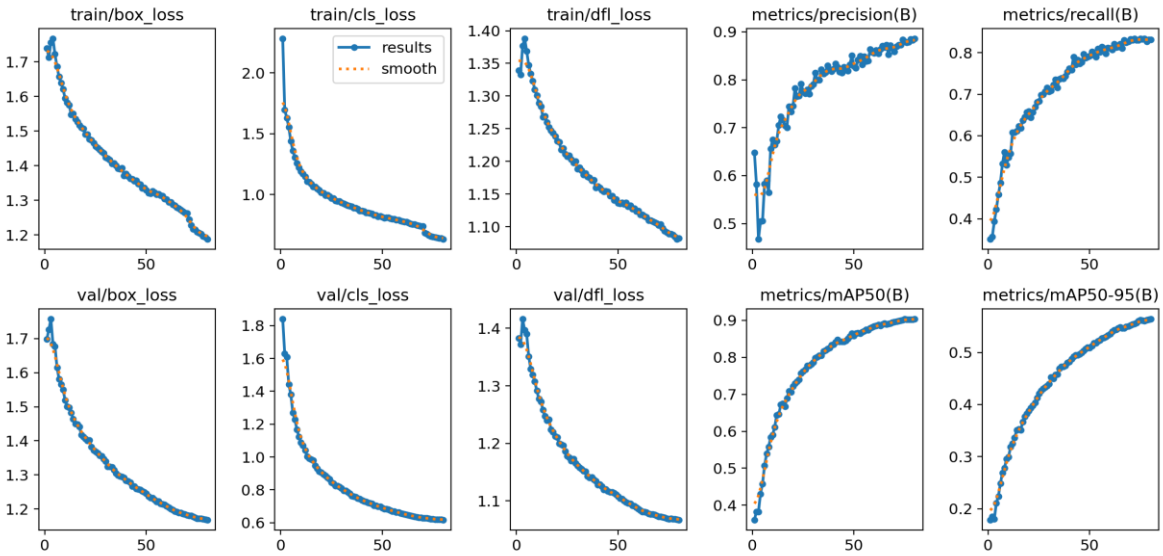
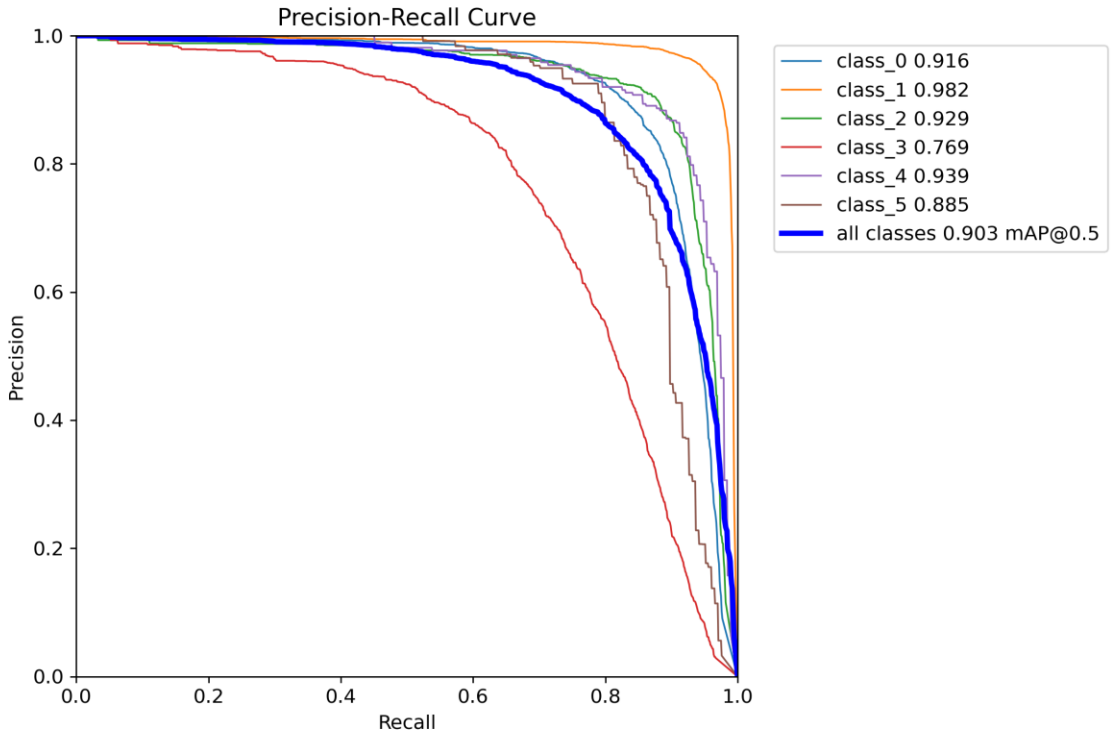
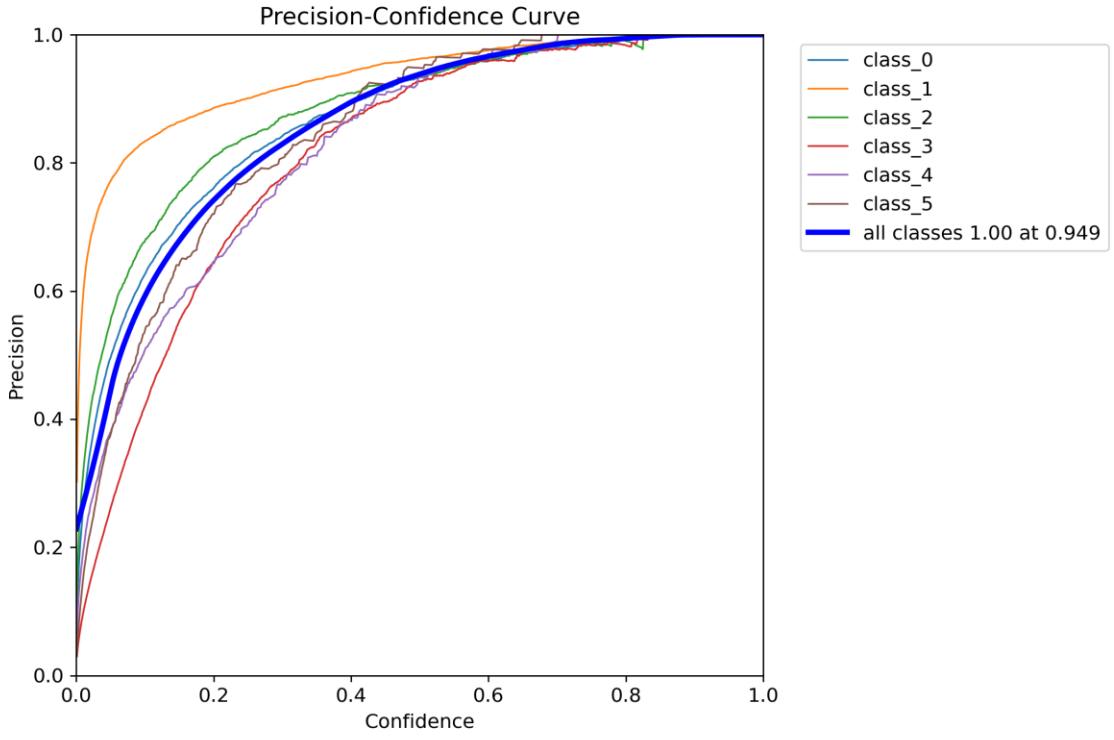


Fig 8.1 : Training results showing loss curves and evaluation metrics (e.g., precision, recall, mAP) across epochs using the configured hyperparameters.


Precision-Recall Curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
class_0 0.916
class_1 0.982
class_2 0.929
class_3 0.769
class_4 0.939
class_5 0.885
all classes 0.903 mAP@0.5


Precision-Confidence Curve
Precision
Confidence
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
class_0
class_1
class_2
class_3
class_4
class_5
all classes 1.00 at 0.949

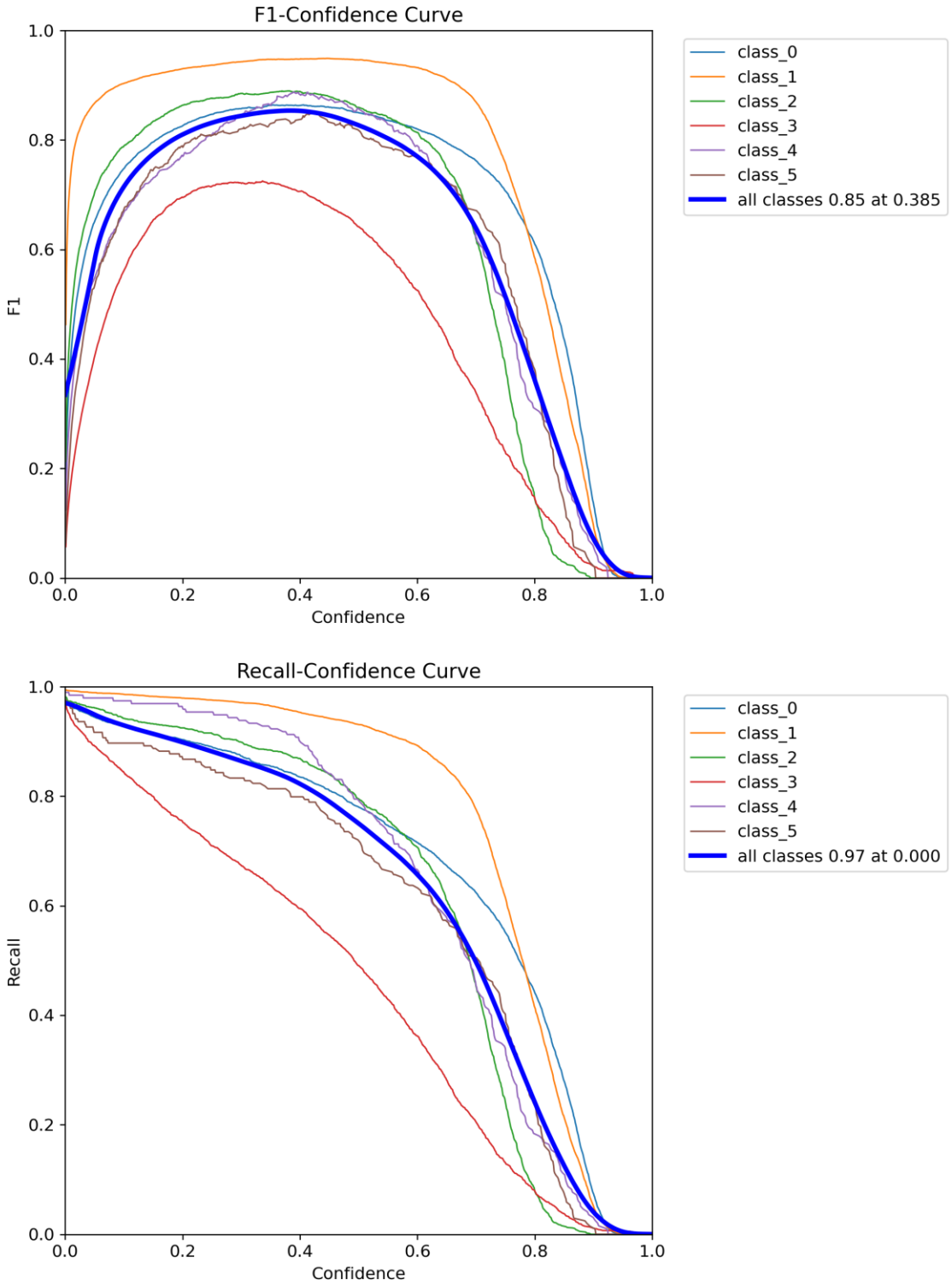


Fig 8.1: Comprehensive training results of the YOLOv8 model showing loss components, Precision, Recall, F1-score, and Precision–Recall (PR) curve, illustrating the model's detection performance and convergence behavior over training epochs.

9. Performance Metrics

mAP50: ~0.90

mAP50-95: ~0.56

Precision: ~0.89

Recall: ~0.83

These metrics indicate strong detection reliability with balanced precision and recall.

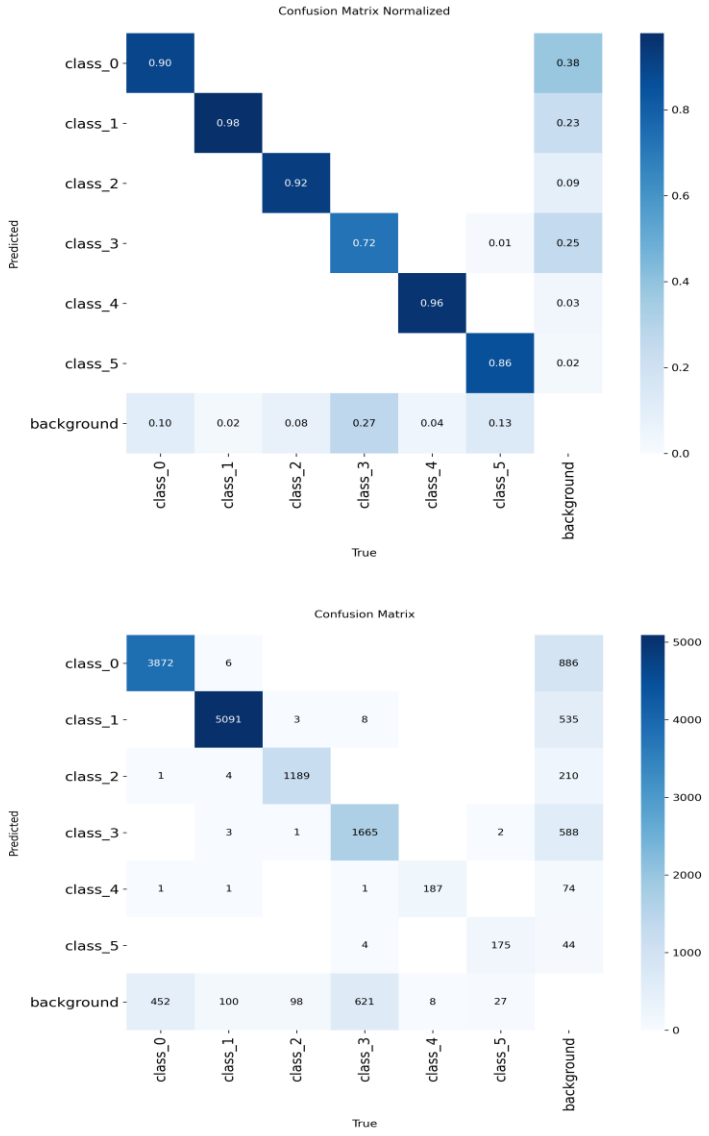


Fig 9.1: Confusion matrices of the trained YOLOv8 model, including the standard and normalized versions, illustrating class-wise

10. Model Export and ONNX Optimization

The trained PyTorch model was exported to ONNX format for cross-platform compatibility.

ONNX enables hardware-agnostic deployment.

OpenCV DNN backend was selected for ARM CPU inference.

Blob preprocessing reduces runtime overhead.

11. Edge Deployment Strategy

Deployment performed on Raspberry Pi 4 with ARM Cortex-A72 CPU.

Inference executed using cv2.dnn.readNetFromONNX().

Frame resizing reduces computation load.

Inference timing measured separately from rendering.

12. Optimization Techniques

Lightweight YOLOv8 Nano architecture selection.

Reduced input resolution where necessary.

Avoided dynamic memory allocation inside inference loop.

Separated inference timing measurement.

Reduced logging overhead.

Potential INT8 quantization for future improvement.

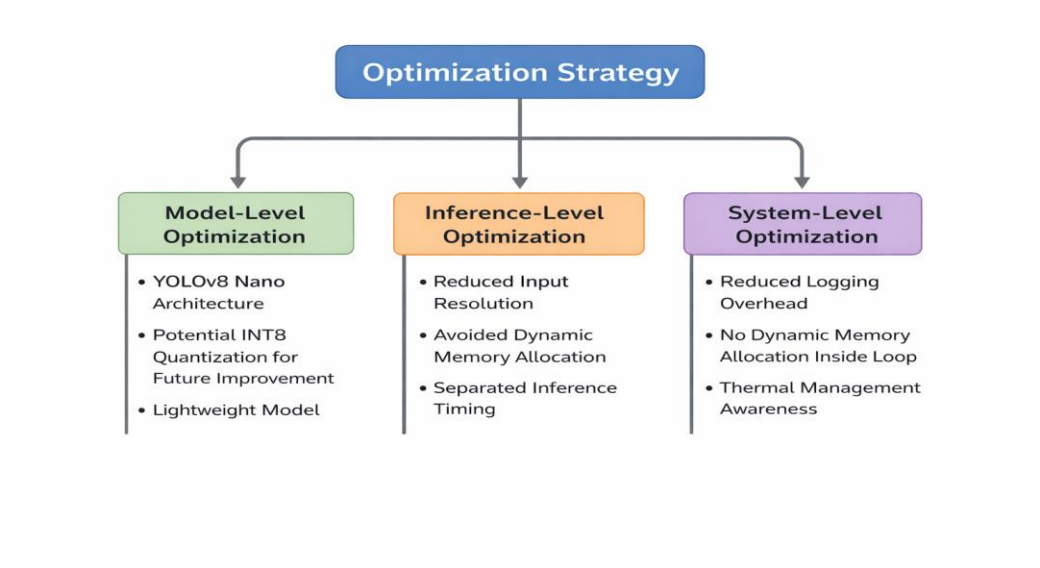


Fig 12.1 : Overview of optimization techniques applied to enhance the efficiency and real-time performance of the YOLOv8 model on embedded hardware.

13. Hardware Profiling and Benchmarking

Metrics collected: FPS, inference time, CPU usage, temperature.

Thermal monitoring ensures no throttling under sustained load.

Table 13.1: Performance benchmarking results measured on Raspberry Pi 4 during real-time inference.

Note: Performance benchmarking results measured on Raspberry Pi 4 during real-time end-to-end inference.

Metric	**Measured Value**
Average FPS	6 FPS
Minimum FPS	4 FPS
Average Inference Time (ms)	45
CPU Usage (Average %)	74
CPU Usage (Peak %)	300
CPU Temperature (Average °C)	47.8
CPU Temperature (Peak °C)	52
Model Format	ONNX (FP32)
Input Resolution	320 x 320

14. Limitations

FP32 inference increases CPU utilization.

Performance may degrade under low lighting.

Thermal throttling possible during extended operation.

15. Future Scope

INT8 Quantization using TensorRT or OpenVINO.

FP16 optimization if hardware supported.

Integration with vehicle ADAS systems.

Cloud synchronization for anomaly reporting.

16. Conclusion

The project demonstrates successful deployment of deep learning on ARM-based edge hardware.

The system achieves real-time detection while maintaining high accuracy.

This work validates the feasibility of edge AI for smart transportation applications.