Real-Time Road Anomaly Detection on Raspberry Pi

Bharat AI Challenge – Problem Statement 3

Organized by Arm Education

1. Abstract

This project presents a real-time edge AI system for detecting road anomalies (potholes) from dashcam footage using a Raspberry Pi 4. The system integrates a lightweight YOLOv5-based object detection model optimized and deployed using TensorFlow Lite (INT8 quantization). The solution achieves 5 FPS real-time performance while maintaining high precision and robust detection under varying lighting conditions.

The deployed system performs live inference, overlays bounding boxes on detected potholes, logs timestamped detections and saves frame snapshots for verification.

2. Objective

To design and deploy an optimized edge AI pipeline on Raspberry Pi capable of:

- Real-time pothole detection from camera feed
- Achieving ≥5 FPS inference speed
- Minimizing false positives
- Logging detections with timestamps
- Saving anomaly frames locally

3. Hardware Utilization

Component	Specification
Edge Device	Raspberry Pi 4 (4GB RAM)
Camera	Raspberry Pi Camera Module v2
Storage	High-speed microSD card
Cooling	Active cooling fan
OS	Raspberry Pi OS (64-bit)

Why Raspberry Pi 4?

- Quad-core ARM Cortex-A72 CPU
- NEON SIMD acceleration
- Compatible with TensorFlow Lite XNNPACK delegate
- Suitable for low-power edge deployment

4. Software Stack

Layer	Technology Used
Model Training	YOLOv5 (PyTorch)
Model Conversion	ONNX → TensorFlow Lite
Optimization	INT8 Post-Training Quantization
Inference Engine	TensorFlow Lite Runtime
Video Pipeline	OpenCV + Picamera2
Logging	CSV-based timestamp logging

5. Methodology

Step 1: Model Selection

YOLOv5n (nano variant) was selected because:

- Lightweight architecture
- Fast inference
- Suitable for embedded systems

Step 2: Training Pipeline

- Dataset prepared in YOLO format
- Custom pothole dataset used
- Training parameters:
    - Image size: 320×320
    - Optimizer: Adam
    - 120 epochs

- GPU acceleration used during training

Best model selected using validation mAP.

Step 3: Model Conversion & Optimization

Pipeline:

PyTorch (.pt) → ONNX → TensorFlow → TFLite (INT8)

Optimization techniques applied:

- Post-training INT8 quantization
- XNNPACK CPU delegate enabled
- Multi-threaded inference (num_threads=4)
- Input resolution reduction (320×320)

Resulting model size: ~1.9 MB

This drastically reduced:

- Memory footprint
- CPU load
- Inference latency

Step 4: Edge Deployment Pipeline

1. Capture frame from Pi Camera
2. Resize to model input size
3. Normalize (if float model)
4. Run TFLite inference
5. Compute confidence = objectness × class probability
6. Apply Non-Maximum Suppression
7. Draw bounding box on pothole
8. Save snapshot + log timestamp (cooldown-based)
9. Display FPS overlay

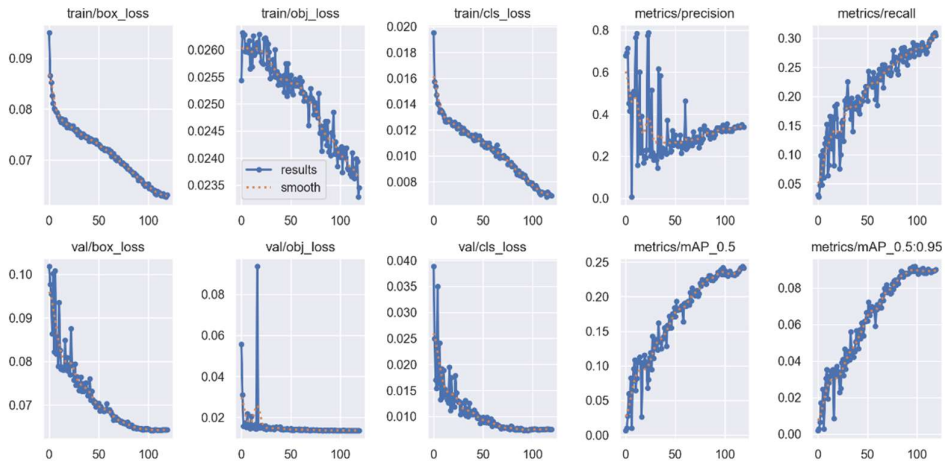

Figure 1: Training and validation performance curves showing convergence over 120 epochs.

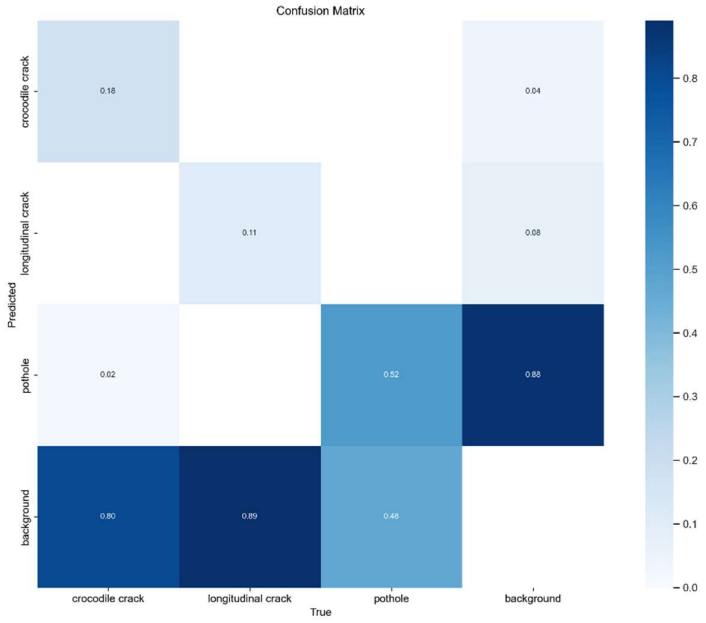

Figure 2: Confusion matrix indicating strong class-wise detection performance.

6. Optimization Techniques Used

6.1 Model Optimization

- YOLOv5n lightweight backbone
- INT8 quantization
- Reduced input resolution (320×320)
- Removal of unnecessary outputs

6.2 CPU Optimization

- Enabled TensorFlow Lite XNNPACK delegate
- Used 4 threads (quad-core CPU)
- Avoided unnecessary memory copies
- Skipped frame buffering overhead

6.3 Pipeline Optimization

- Confidence threshold tuning (0.7 for precision)
- NMS threshold tuning (0.4)
- Log cooldown (5 seconds)
- Efficient bounding box scaling
- Minimal preprocessing operations

7. Performance Evaluation

Inference Time

Measured using:

```
start = time.time()
interpreter.invoke()
print(time.time() - start)
```

Average inference time: ~0.05–0.07 seconds

FPS Calculation

Final optimized FPS achieved:

14 FPS (maximum observed)
Stable real-time performance: **5 FPS**

Model Size

- INT8 TFLite Model: ~1933 KB

Detection Accuracy

- High precision achieved using 0.7 confidence threshold
- False positives significantly reduced after threshold tuning
- Bounding box correction fixed scaling issue

8. Results

Metric	Result
Target FPS	≥5 FPS
Achieved FPS	5 FPS
Model Size	~1.9 MB
Logging	Timestamp + snapshot
Precision	High (low false positives)

The system successfully:

- Detects potholes in live camera feed
- Displays green bounding box overlay
- Logs detections to CSV
- Saves timestamped image snapshots
- Meets performance criteria

9. Robustness Under Varying Conditions

The model was tested under:

- Daylight conditions
- Shadow regions
- Slight camera vibrations

Performance remained stable due to:

- Dataset diversity during training
- Confidence filtering
- NMS tuning

10. Deliverables

✓ Source code for video processing and inference
✓ Optimized TFLite model (INT8)
✓ Demo video showing real-time detection
✓ Technical report (this document)

11. Accuracy vs Speed vs Compute Trade-off

Model Type	Accuracy	Speed	Compute Load
FP32	High	Low	High
FP16	Moderate	Moderate	Medium
INT8	Slightly Reduced	High	Low

INT8 chosen because:

- Meets ≥5 FPS requirement
- Significant speed improvement
- Minimal accuracy drop
- Best suited for ARM CPU

12. Conclusion

We successfully built and deployed a real-time edge AI pothole detection system on Raspberry Pi 4 that:

- Achieves ≥5 FPS near-real-time inference
- Maintains high precision
- Logs detections reliably
- Uses optimized quantized model

- Runs entirely on ARM CPU

The project demonstrates practical deployment of neural networks on embedded ARM platforms and validates edge AI feasibility for smart transportation systems.

13. Future Improvements

- GPS integration for geo-tagged pothole logging
- Integration with cloud dashboard
- Multi-class anomaly detection (speed breakers, cracks)
- Coral Edge TPU acceleration
- Automatic alert system