RAJALAKSHMI ENGINEERING COLLEGE
Thandalam, Tamil Nadu 602105

Real-Time Edge-Based Road Anomaly Detection Using YOLOv5 and Statistical Validation on Raspberry Pi 4

Sanji Krishna M P (Team Lead)
Ahamed Faisal A
Subiksha A

Mentor: Dr. S. Chitra, Professor
Department of Electronics and Communication Engineering
Rajalakshmi Engineering College

Abstract

Road surface anomalies such as potholes and cracks significantly impact transportation safety and infrastructure reliability. Manual inspection techniques are labor-intensive and unsuitable for continuous large-scale monitoring. This work presents a real-time edge artificial intelligence (AI) based road anomaly detection system deployed on Raspberry Pi 4 using central processing unit (CPU)-only inference. The system utilizes You Only Look Once version 5 (YOLOv5) converted to Open Neural Network Exchange (ONNX) format and executed using ONNX Runtime (ORT). The architecture integrates structured frame preprocessing, confidence filtering, Intersection over Union (IoU)-based Non-Maximum Suppression (NMS), sliding window temporal voting, and a statistical fallback detection mechanism using Mahalanobis Distance and Z-score analysis. The optimized implementation achieves approximately 7 frames per second (FPS), representing a seven-fold improvement over the baseline configuration. The proposed framework demonstrates that deep learning-based video analytics can be effectively deployed on resource-constrained edge hardware without additional accelerators.

Keywords: Edge Artificial Intelligence, Road Anomaly Detection, YOLOv5, ONNX Runtime, Raspberry Pi Deployment, Non-Maximum Suppression, Temporal Voting, Mahalanobis Distance, Real-Time Inference, Embedded Vision Systems

1 Introduction

Road surface degradation is a major contributor to traffic accidents, vehicle damage, and increased maintenance expenditure. Traditional inspection methods rely on manual surveys, which are subjective and not scalable. Recent advances in deep learning have enabled automated anomaly detection; however, most implementations depend on GPU acceleration. Deploying such systems on low-power embedded devices remains challenging. This work focuses on designing a computationally efficient, CPU-optimized road anomaly detection framework capable of near real-time model execution on Raspberry Pi 4.

2 Related Work

Object detection models such as Faster R-CNN, Single Shot Detector (SSD), and YOLO have been extensively used for surface inspection tasks. Two-stage detectors provide high precision but demand significant computational resources. Single-stage detectors such as YOLOv5 offer faster model execution with acceptable accuracy, making them suitable for edge applications.

Most existing implementations process data in the cloud, which introduces latency and bandwidth constraints. Edge computing reduces response time and improves system autonomy. However, maintaining detection stability under limited computational resources remains an open research challenge. This work addresses these challenges through structured optimization and statistical validation.

3 Objectives

- To achieve model execution performance greater than 5 frames per second on Raspberry Pi 4 using CPU-only deployment.
- To reduce false positive detections using IoU-based Non-Maximum Suppression.
- To enhance temporal stability using sliding window validation.
- To detect unseen anomalies using statistical deviation techniques.
- To design a modular and scalable edge-based monitoring architecture.

4 System Architecture

The system follows a structured multi-stage processing pipeline:

Video Input → Frame Preprocessing → ONNX Model execution → Post-Processing → NMS → Temporal Voting → Statistical Fallback → Logging and Visualization.

Each stage incrementally refines the detection results to ensure accuracy and robustness.

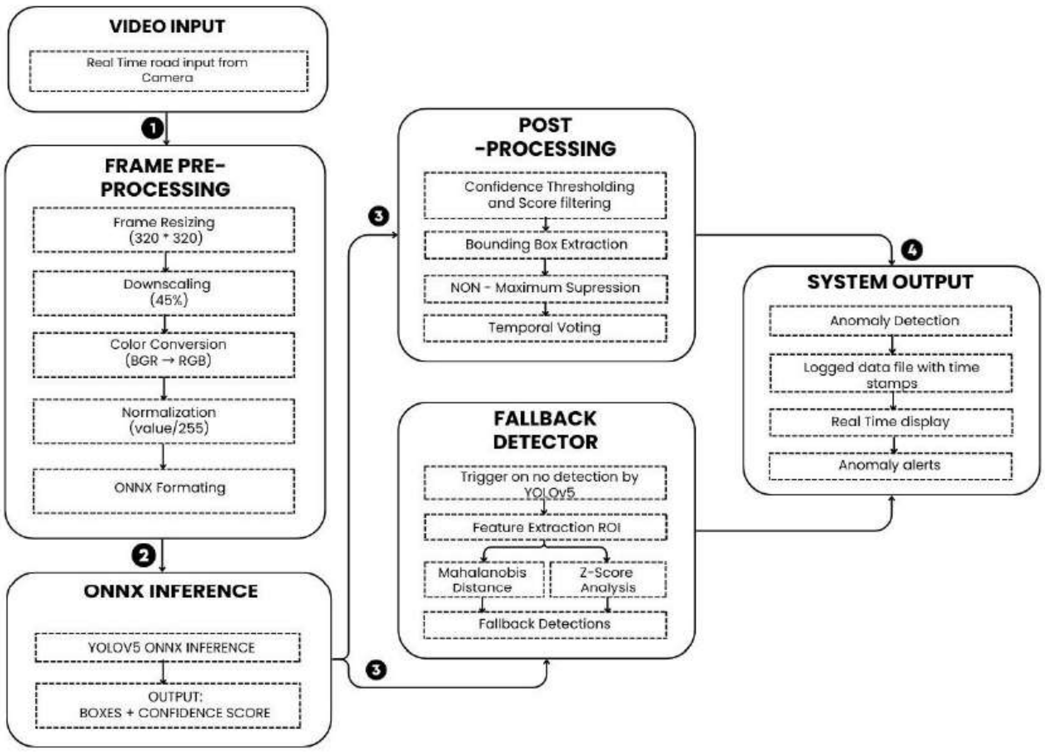


Figure 1: Proposed Edge-Based Road Anomaly Detection Architecture

5 Methodology

5.1 Frame Preprocessing

To reduce computational complexity, each frame is downscaled to 45% of its original resolution and resized to 320×320 pixels. Pixel intensity normalization is performed as:

$$x_{norm} = \frac{x}{255} \tag{1}$$

where x represents the original pixel value and x_{norm} is the normalized value between 0 and 1. This normalization ensures numerical stability during neural network model execution.

5.2 Bounding Box Representation

The YOLOv5 detector outputs bounding boxes represented as:

$$B = \{(x, y, w, h, c)\} \tag{2}$$

Here, (x, y) denote the center coordinates, w and h represent width and height, and c denotes the confidence score associated with the detection.

5.3 Intersection over Union

Duplicate bounding boxes are removed using IoU:

$$IoU = \frac{Area_{intersection}}{Area_{union}} \tag{3}$$

If the overlap between two boxes exceeds a predefined threshold, the box with lower confidence is suppressed.

5.4 Temporal Voting

Detection validation across frames is performed using:

$$Detection_{valid} = \begin{cases} True & \text{if } votes \geq 2 \\ False & \text{otherwise} \end{cases} \tag{4}$$

This ensures that only persistent detections across consecutive frames are confirmed.

5.5 Mahalanobis Distance

Statistical deviation is computed as:

$$D_M(x) = (x - \mu)^T \Sigma^{-1} (x - \mu) \tag{5}$$

where μ represents the mean feature distribution and Σ^{-1} represents the inverse covariance matrix. Larger values indicate deviation from normal road texture.

5.6 Z-Score Analysis

Intensity deviation is measured using:

$$Z = \frac{|x - \mu|}{\sigma} \tag{6}$$

where σ is the standard deviation and μ is the mean intensity value. A high Z-score suggests abnormal surface characteristics.

6 Software Architecture

The system is implemented using modular Python scripts.

runvid.py handles video capture, preprocessing, inference execution, visualization, and timestamp logging.

mahalanobis_detector.py performs feature extraction and statistical distance computation.

fallback_anomaly.py implements Z-score based anomaly confirmation.

7 Hardware Platform

Target Hardware: Raspberry Pi 4

CPU: Quad-core Cortex-A72

RAM: 4GB

Inference Mode: CPU-only

The absence of GPU acceleration emphasizes computational optimization strategies.

8 Optimization Techniques

Frame downscaling reduces pixel-level operations, achieving approximately $2\times$ improvement in processing speed. ONNX Runtime graph optimization enables operator fusion and reduces redundant computation. Memory optimization through rolling temporal buffers minimizes allocation overhead. Combined, these strategies resulted in performance improvement from 1 FPS to 7 FPS.

9 Experimental Evaluation

Table 1: Evaluation on Different Devices

Device	Initial FPS	Optimized FPS
Laptop	25–45	25–45
Raspberry Pi 4	1	7

The optimized deployment achieved 7 FPS, meeting the near real-time target while maintaining detection stability.

10 Limitations

Performance may degrade under extreme low-light conditions. CPU-only model execution restricts scalability to larger models. Quantization was not implemented and remains a potential area for enhancement.

11 Results and Discussion

11.1 Experimental Setup

The performance of the proposed Edge AI road anomaly detection system was evaluated across multiple stages of development, transitioning from a high-performance laptop environment to a resource-constrained Raspberry Pi 4 edge device.

All edge experiments were conducted on Raspberry Pi 4 with CPU-only model execution using ONNX Runtime execution provider with input resolution of 320 × 320 pixels and a frame scale of 45%. Performance was measured in Frames Per Second (FPS), representing real-time model execution capability.

11.2 Analysis of Performance Degradation on Edge

When the model was directly deployed onto the Raspberry Pi without modification, performance dropped from 25 - 45 FPS (Figure 2) to 1 FPS (Figure 3). This significant degradation can be attributed to:

- CPU-only model execution without GPU acceleration
- PyTorch runtime overhead
- High computational complexity of convolutional layers
- Inefficient preprocessing pipeline
- Memory access latency

This stage confirmed that deep learning models trained on high-performance hardware require systematic optimization before edge deployment.

11.3 Impact of Optimization Phase

The significant increase in FPS from 1 FPS in the initial edge deployment to 7 FPS in the final optimized system (Figure 5) can be attributed to systematic architectural and runtime-level optimizations tailored for resource-constrained hardware. Initially, the model was executed using the full PyTorch runtime (Figure 3), which introduced considerable framework overhead and inefficient CPU utilization. By restructuring the preprocessing pipeline through frame downscaling (45%) and resizing inputs to 320×320, the computational load per frame was substantially reduced and achieved 4 FPS (Figure 4). Further improvements were achieved by enabling graph-level optimizations, operator fusion, and reduced runtime latency through ONNX Runtime's CPU execution provider. Memory handling was streamlined using contiguous arrays and lightweight buffering, minimizing data transfer overhead. Beyond FPS improvement, system robustness was enhanced by using efficient post-processing and structured Non-Maximum Suppression reduced redundant computations, while temporal voting improved detection stability without adding significant computational cost to achieve . Additionally, Statistical Anomaly Detection (Mahalanobis and Z-score) provided fallback detection capability when the primary model produced null outputs.

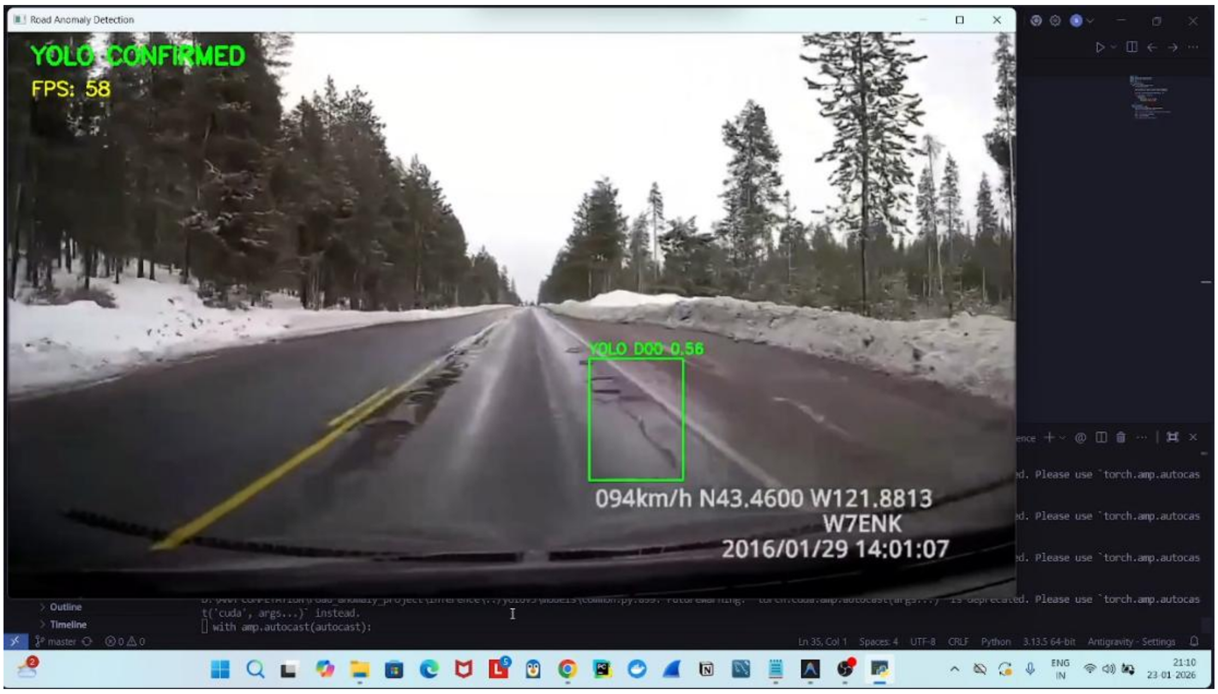
Collectively, these optimizations transformed the system from an unoptimized CPU-bound deployment at 1 FPS to a stable 7 FPS near-real-time Edge AI solution (Figure 5), representing a 7-fold performance improvement suitable for practical field deployment (Figure 6).

11.4 Real-World Field Validation

Field testing was conducted by mounting the Raspberry Pi system on a moving bike and driving over roads containing potholes (Figure 6).
Real-world challenges included:

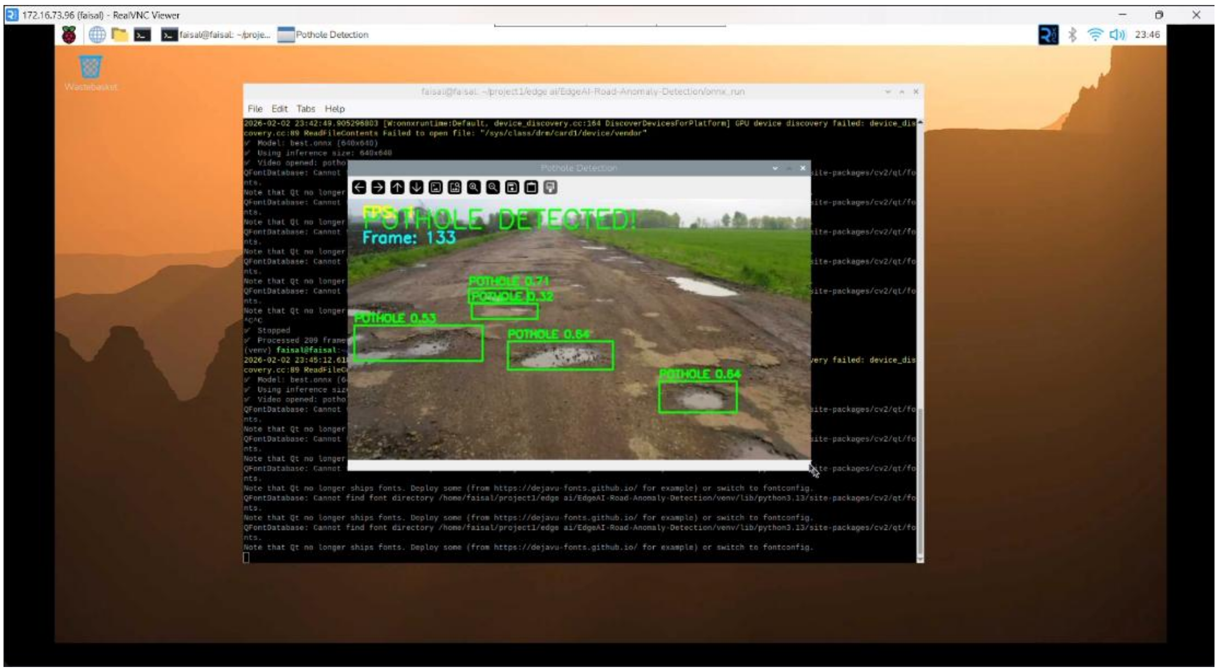
- Motion blur
- Camera vibration
- Dynamic lighting variation
- Complex road textures

Despite these challenges, the system maintained approximately 7–8 FPS and successfully detected potholes with temporal stability.
This confirms that the proposed architecture is not only computationally efficient but also robust under real-world operating conditions.



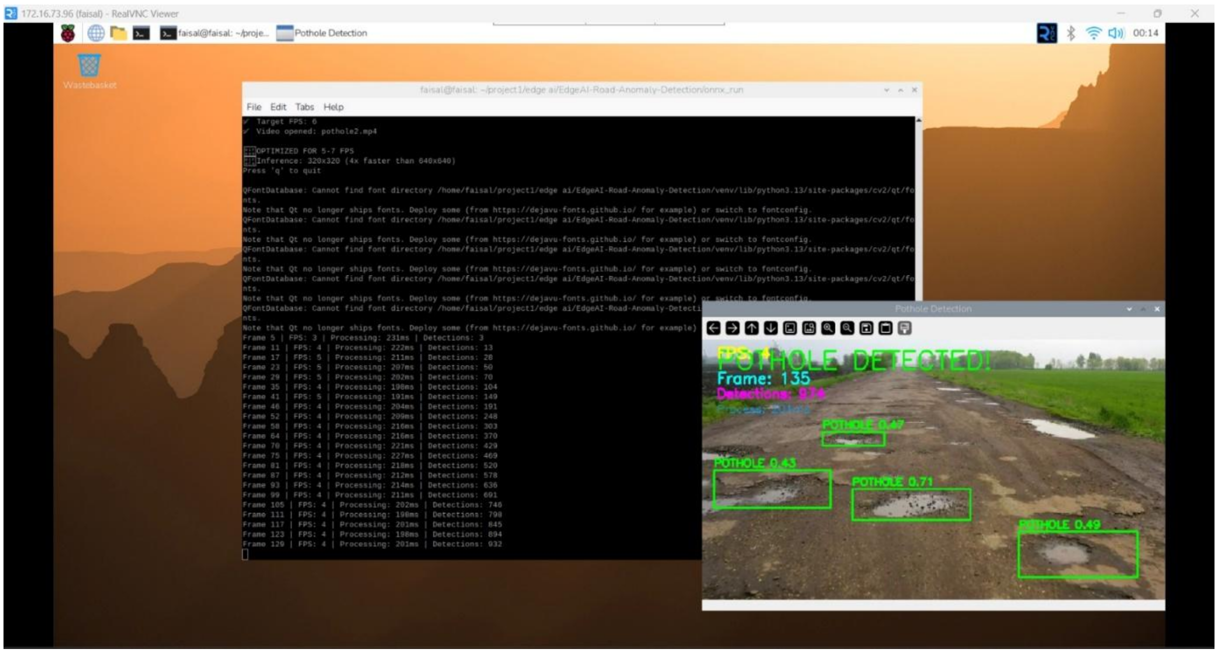
```
d. Please use `torch.amp.autocas
d. Please use `torch.amp.autocas
d. Please use `torch.amp.autocas
t('cuda', args...)` instead.                                     ed. Please use `torch.amp.autocas
with amp.autocast(autocast):
```

Figure 1: Baseline performance in Laptop (Average 25-45 FPS)



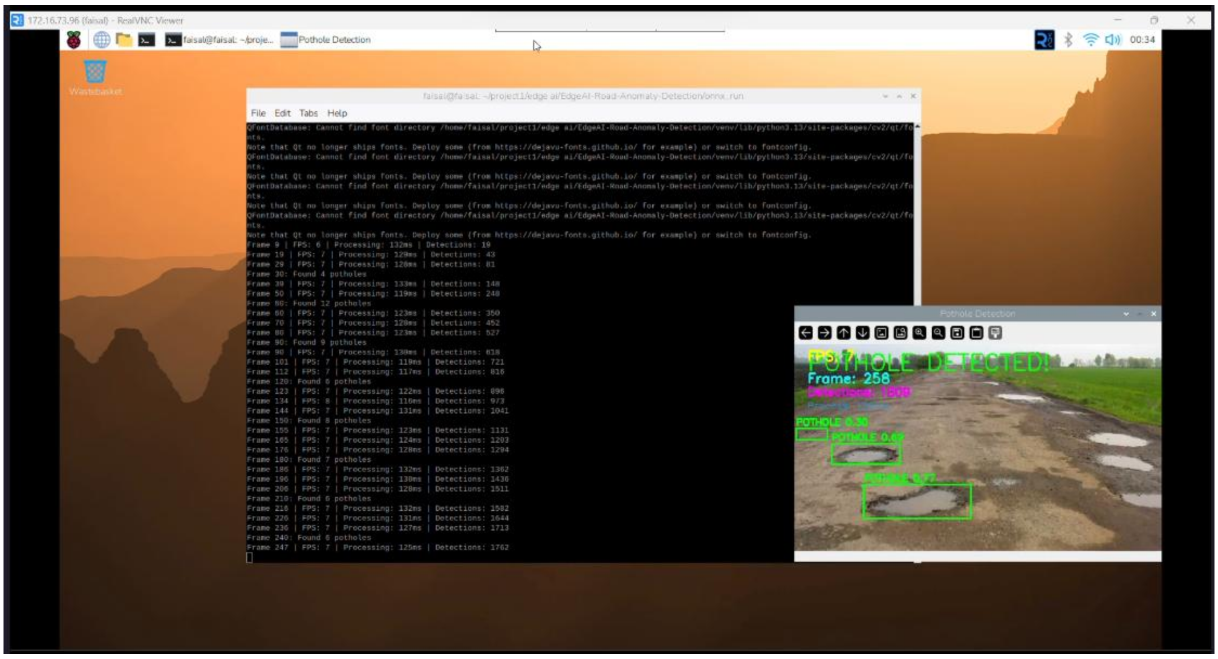
```
2026-02-02 23:42:49.905296803 [W:onnxruntime:Default, device_discovery.cc:164 DiscoverDevicesForPlatform] GPU device discovery failed: device_dis
covery.cc:89 ReadFileContents Failed to open file: "/sys/class/drm/card1/device/vendor"
✓  Model: best.onnx (640x640)
✓  Using inference size: 640x640
✓  Video opened: potho
QFontDatabase: Cannot                                                                                  site-packages/cv2/qt/fo
nts.
Note that Qt no longer
QFontDatabase: Cannot                                                                                  site-packages/cv2/qt/fo
nts.
Note that Qt no longer
QFontDatabase: Cannot                                                                                  site-packages/cv2/qt/fo
nts.
Note that Qt no longer
QFontDatabase: Cannot                                                                                  site-packages/cv2/qt/fo
nts.
Note that Qt no longer
QFontDatabase: Cannot                                                                                  site-packages/cv2/qt/fo
nts.
Note that Qt no longer
^C^C
✓  Stopped
✓  Processed 209 frame
(venv) faisal@faisal:~
2026-02-02 23:45:12.61                                                                                 very failed: device_dis
covery.cc:89 ReadFileCo
✓  Model: best.onnx (6
✓  Using inference siz
✓  Video opened: potho
QFontDatabase: Cannot                                                                                  site-packages/cv2/qt/fo
nts.
Note that Qt no longer
QFontDatabase: Cannot                                                                                  site-packages/cv2/qt/fo
nts.
Note that Qt no longer
QFontDatabase: Cannot                                                                                  site-packages/cv2/qt/fo
nts.
Note that Qt no longer ships fonts. Deploy some (from https://dejavu-fonts.github.io/ for example) or switch to fontconfig.
QFontDatabase: Cannot find font directory /home/faisal/project1/edge ai/EdgeAI-Road-Anomaly-Detection/venv/lib/python3.13/site-packages/cv2/qt/fo
nts.
Note that Qt no longer ships fonts. Deploy some (from https://dejavu-fonts.github.io/ for example) or switch to fontconfig.
QFontDatabase: Cannot find font directory /home/faisal/project1/edge ai/EdgeAI-Road-Anomaly-Detection/venv/lib/python3.13/site-packages/cv2/qt/fo
nts.
Note that Qt no longer ships fonts. Deploy some (from https://dejavu-fonts.github.io/ for example) or switch to fontconfig.
```

Figure 2: First Edge Deployment - Raspberry Pi (1 FPS)


172.16.73.96 (faisal) - RealVNC Viewer
faisal@faisal: ~/proje...
Pothole Detection
00:14
Wastebasket
faisal@faisal: ~/project1/edge ai/EdgeAI-Road-Anomaly-Detection/onnx_run
File Edit Tabs Help
POTHOLE DETECTED!
Frame: 135
POTHOLE 0.47
POTHOLE 0.43
POTHOLE 0.71
POTHOLE 0.49

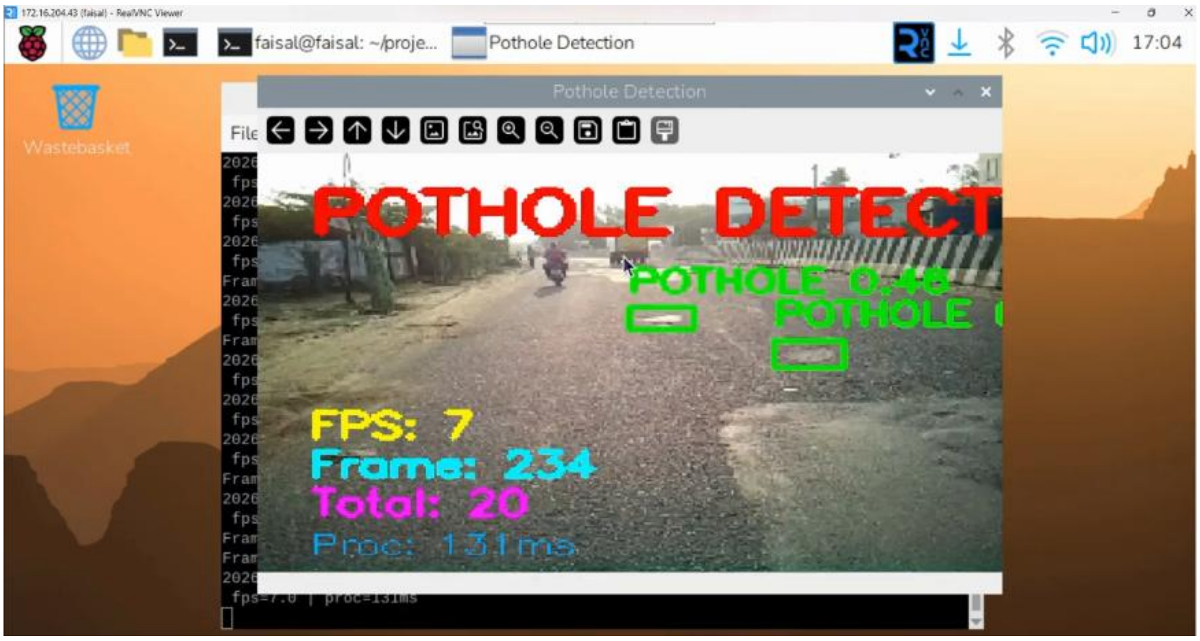
```
✓ Target FPS: 6
✓ Video opened: pothole2.mp4

OPTIMIZED FOR 5-7 FPS
Inference: 320x320 (4x faster than 640x640)
Press 'q' to quit

QFontDatabase: Cannot find font directory /home/faisal/project1/edge ai/EdgeAI-Road-Anomaly-Detection/venv/lib/python3.13/site-packages/cv2/qt/fo
nts.
Note that Qt no longer ships fonts. Deploy some (from https://dejavu-fonts.github.io/ for example) or switch to fontconfig.
QFontDatabase: Cannot find font directory /home/faisal/project1/edge ai/EdgeAI-Road-Anomaly-Detection/venv/lib/python3.13/site-packages/cv2/qt/fo
nts.
Note that Qt no longer ships fonts. Deploy some (from https://dejavu-fonts.github.io/ for example) or switch to fontconfig.
QFontDatabase: Cannot find font directory /home/faisal/project1/edge ai/EdgeAI-Road-Anomaly-Detection/venv/lib/python3.13/site-packages/cv2/qt/fo
nts.
Note that Qt no longer ships fonts. Deploy some (from https://dejavu-fonts.github.io/ for example) or switch to fontconfig.
QFontDatabase: Cannot find font directory /home/faisal/project1/edge ai/EdgeAI-Road-Anomaly-Detection/venv/lib/python3.13/site-packages/cv2/qt/fo
nts.
Note that Qt no longer ships fonts. Deploy some (from https://dejavu-fonts.github.io/ for example) or switch to fontconfig.
QFontDatabase: Cannot find font directory /home/faisal/project1/edge ai/EdgeAI-Road-Anomaly-Detecti
nts.
Note that Qt no longer ships fonts. Deploy some (from https://dejavu-fonts.github.io/ for example)
Frame 5 | FPS: 3 | Processing: 231ms | Detections: 3
Frame 11 | FPS: 4 | Processing: 222ms | Detections: 13
Frame 17 | FPS: 5 | Processing: 211ms | Detections: 28
Frame 23 | FPS: 5 | Processing: 207ms | Detections: 50
Frame 29 | FPS: 5 | Processing: 202ms | Detections: 70
Frame 35 | FPS: 4 | Processing: 198ms | Detections: 104
Frame 41 | FPS: 5 | Processing: 191ms | Detections: 149
Frame 46 | FPS: 4 | Processing: 204ms | Detections: 191
Frame 52 | FPS: 4 | Processing: 209ms | Detections: 248
Frame 58 | FPS: 4 | Processing: 216ms | Detections: 303
Frame 64 | FPS: 4 | Processing: 216ms | Detections: 370
Frame 70 | FPS: 4 | Processing: 221ms | Detections: 429
Frame 75 | FPS: 4 | Processing: 227ms | Detections: 469
Frame 81 | FPS: 4 | Processing: 218ms | Detections: 520
Frame 87 | FPS: 4 | Processing: 212ms | Detections: 578
Frame 93 | FPS: 4 | Processing: 214ms | Detections: 636
Frame 99 | FPS: 4 | Processing: 211ms | Detections: 691
Frame 105 | FPS: 4 | Processing: 202ms | Detections: 746
Frame 111 | FPS: 4 | Processing: 198ms | Detections: 798
Frame 117 | FPS: 4 | Processing: 201ms | Detections: 845
Frame 123 | FPS: 4 | Processing: 198ms | Detections: 894
Frame 129 | FPS: 4 | Processing: 201ms | Detections: 932
```

Figure 3: First Optimization - Improved Edge Performance (4 FPS)


172.16.73.96 (faisal) - RealVNC Viewer
faisal@faisal: ~/proje...
Pothole Detection
00:34
Wastebasket
faisal@faisal: ~/project1/edge ai/EdgeAI-Road-Anomaly-Detection/onnx_run
File Edit Tabs Help
POTHOLE DETECTED!
Frame: 258
POTHOLE 0.30
POTHOLE 0.69

```
QFontDatabase: Cannot find font directory /home/faisal/project1/edge ai/EdgeAI-Road-Anomaly-Detection/venv/lib/python3.13/site-packages/cv2/qt/fo
nts.
Note that Qt no longer ships fonts. Deploy some (from https://dejavu-fonts.github.io/ for example) or switch to fontconfig.
QFontDatabase: Cannot find font directory /home/faisal/project1/edge ai/EdgeAI-Road-Anomaly-Detection/venv/lib/python3.13/site-packages/cv2/qt/fo
nts.
Note that Qt no longer ships fonts. Deploy some (from https://dejavu-fonts.github.io/ for example) or switch to fontconfig.
QFontDatabase: Cannot find font directory /home/faisal/project1/edge ai/EdgeAI-Road-Anomaly-Detection/venv/lib/python3.13/site-packages/cv2/qt/fo
nts.
Note that Qt no longer ships fonts. Deploy some (from https://dejavu-fonts.github.io/ for example) or switch to fontconfig.
QFontDatabase: Cannot find font directory /home/faisal/project1/edge ai/EdgeAI-Road-Anomaly-Detection/venv/lib/python3.13/site-packages/cv2/qt/fo
nts.
Note that Qt no longer ships fonts. Deploy some (from https://dejavu-fonts.github.io/ for example) or switch to fontconfig.
Frame 9 | FPS: 6 | Processing: 132ms | Detections: 19
Frame 19 | FPS: 7 | Processing: 129ms | Detections: 43
Frame 29 | FPS: 7 | Processing: 128ms | Detections: 81
Frame 30: Found 4 potholes
Frame 39 | FPS: 7 | Processing: 133ms | Detections: 148
Frame 50 | FPS: 7 | Processing: 119ms | Detections: 248
Frame 60: Found 12 potholes
Frame 60 | FPS: 7 | Processing: 123ms | Detections: 350
Frame 70 | FPS: 7 | Processing: 128ms | Detections: 452
Frame 80 | FPS: 7 | Processing: 123ms | Detections: 527
Frame 90: Found 9 potholes
Frame 90 | FPS: 7 | Processing: 130ms | Detections: 618
Frame 101 | FPS: 7 | Processing: 119ms | Detections: 721
Frame 112 | FPS: 7 | Processing: 117ms | Detections: 816
Frame 120: Found 6 potholes
Frame 123 | FPS: 7 | Processing: 122ms | Detections: 898
Frame 134 | FPS: 8 | Processing: 116ms | Detections: 973
Frame 144 | FPS: 7 | Processing: 131ms | Detections: 1041
Frame 150: Found 8 potholes
Frame 155 | FPS: 7 | Processing: 123ms | Detections: 1131
Frame 165 | FPS: 7 | Processing: 124ms | Detections: 1203
Frame 176 | FPS: 7 | Processing: 128ms | Detections: 1294
Frame 180: Found 7 potholes
Frame 186 | FPS: 7 | Processing: 132ms | Detections: 1362
Frame 196 | FPS: 7 | Processing: 130ms | Detections: 1436
Frame 206 | FPS: 7 | Processing: 128ms | Detections: 1511
Frame 210: Found 6 potholes
Frame 216 | FPS: 7 | Processing: 132ms | Detections: 1582
Frame 226 | FPS: 7 | Processing: 131ms | Detections: 1644
Frame 236 | FPS: 7 | Processing: 127ms | Detections: 1713
Frame 240: Found 6 potholes
Frame 247 | FPS: 7 | Processing: 125ms | Detections: 1762
```

Figure 4: Final Optimization - Production Edge Deployment (7 FPS)



```
fps=7.0 | proc=131ms
```

Figure 5: Real-World Field Testing - Bike Pothole Detection

11 Conclusion & Future Work

This work presents a real-time edge artificial intelligence-based road anomaly detection system successfully deployed on Raspberry Pi 4 using CPU-only Through structured preprocessing, ONNX optimization, Non-Maximum Suppression, temporal voting, and statistical fallback mechanisms, the system achieves approximately 7 frames per second with stable detection performance. model execution The proposed system supports sustainable infrastructure development and contributes toward intelligent transportation systems. Future improvements include applying integer quantization and model pruning to further increase model execution speed and reduce memory usage on edge hardware. Deploying the system on hardware accelerators such as Edge TPUs or Jetson platforms can significantly improve FPS beyond CPU-only limits. Expanding the dataset with more diverse road conditions will enhance model generalization and robustness. Integration of GPS-based geo-tagging and cloud connectivity can enable large-scale road monitoring and analytics. Additionally, incorporating lightweight model architectures and adaptive resolution scaling can further optimize performance for real-world deployment. Finally, exploring self-supervised or continual learning mechanisms could allow the model to adapt over time to new road conditions without full retraining.