Road Damage Detection: A YOLO-based Approach with Real-Time Mobile Deployment.

By

Md. Tahrin Jahan Emon
221-15-6044

FINAL YEAR DESIGN PROJECT REPORT

This Report Presented in Partial Fulfillment of the Requirements forthe **Degree of Bachelor of Science in Computer Science and Engineering**

Supervised by

Dr. S. M. Aminul Haque
Associate Professor and Associate Head
Department of Computer Science and EngineeringDaffodil International University

Co-Supervised by

Nafiz Ahmed Emon
Lecturer
Department of Computer Science and EngineeringDaffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY
Dhaka, Bangladesh

May 14, 2025

APPROVAL

This Project titled **"Road Damage Detection: A YOLOv Approach with Real-Time Mobile Deployment."**, submitted by Tahrin Jahan Emon, Student ID No: 221-15-6044 to the Department of Computer Science and Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering and approved as to its style and contents. The presentation has been held on **14 May, 2025**.

BOARD OF EXAMINERS

Dr. Naznin Sultana — **Chairman**
Associate Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

Md. Sazzadur Ahamed — **Internal Examiner**
Assistant Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

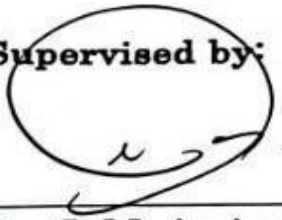
Raja Tariqul Hasan Tusher — **Internal Examiner**
Assistant Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

Shah Md Imran — **External Examiner**
Project Manager, External Member
Smart Leadership Academy and EDGE Project
Bangladesh Computer Council, ICT Division

DECLARATION

We hereby declare that this project has been done by us under the supervision of **Dr. S. M. Aminul Haque, Associate Professor and Associate Head**, Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere for the award of any degree or diploma.

Supervised by:



Dr. S. M. Aminul Haque
Associate Professor and Associate Head
Department of Computer Science and Engineering Daffodil International University

Co-Supervised by:

Nafiz Ahmed Emon
Assistant Professor
Department of Computer Science and Engineering Daffodil International University

Submitted by:

Tahrin 14.05.25

Md. Tahrin Jahan Emon
Student ID: 221-15-6044
Department of Computer Science and Engineering
Daffodil International University

ACKNOWLEDGEMENTS

This work would not have been possible without the support and contributions of many individuals over the past two semesters. We are deeply grateful to everyone who has assisted us in one way or another.

First, we express our heartfelt thanks and gratefulness to the almighty for His divine blessing making it possible for us to complete the **Final Year Design Project (FYDP)** successfully.

We are grateful and wish our profound indebtedness to **Dr. S. M. Aminul Haque , Associate Professor and Associate Head** Department of Computer Science and Engineering, Daffodil International University, Dhaka, Bangladesh. Deep knowledge and keen interest of our supervisor in the field of **Computer Vision** to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts, and correcting themat all stages have made it possible to complete this project.

We would like to express our heartfelt gratitude to the Head of the Department of Computer Science and Engineering for his kind help in finishing our project and also to other faculty members and the staff of the Department of Computer Science and Engineering, Daffodil International University.

We would like to thank our entire course-mates at Daffodil International University, who took part in this discussion while completing the coursework.

Finally, we must acknowledge with due respect the constant support and patience of our parents.

ABSTRACT

Timely detection of road surface damage plays a crucial role in maintaining transportation safety and reducing infrastructure repair costs. Traditional road inspection methods are time-consuming, labor-intensive, and often lack precision, particularly in regions with limited resources. This research proposes a lightweight, real-time road damage detection system using deep learning-based object detection models integrated into a mobile application. The study evaluates and compares three recent YOLO models YOLOv9s, YOLOv10s, and YOLOv12s trained on a custom-annotated dataset of road surface images. Each model is assessed based on detection accuracy (mAP50 and mAP50-95), computational complexity (GFLOPs), and inference speed. Among the three, YOLOv9s demonstrated the best overall performance, achieving 88.2% mAP50 and 52.8% mAP50-95 with an inference speed of 9.8 ms and 26.7 GFLOPs. In contrast, YOLOv10s and YOLOv12s achieved lower accuracy scores but provided faster inference speeds of 6.9 ms and 4.3 ms, respectively, with significantly reduced computational loads. Based on the evaluation, YOLOv9s was selected as the optimal model and exported in TensorFlow Lite format (.tflite) for integration into a Flutter-based Android mobile application. The final system enables users to detect road damage in real-time directly from their smartphones, with results displayed instantly without the need for cloud processing. The proposed solution bridges the gap between academic model development and practical deployment, offering a scalable, cost-effective tool for road maintenance in both urban and rural environments. By ensuring low latency, high detection accuracy, and lightweight design, this study contributes a robust framework for intelligent infrastructure monitoring and supports future smart city applications.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the background and motivation behind the development of a real-time, mobile-compatible road damage detection system using YOLO- based deep learning models. It outlines the key objectives of the research, presents the methodological pipeline involving model evaluation and mobile deployment, and highlights the expected outcomes. The chapter also defines the structure of the report, setting the foundation for the technical discussions in the following chapters.

1.1 Introduction

In the expanding frontier of computer vision, object detection plays a pivotal role in automating tasks that previously required extensive human oversight. Among the many architectures developed in recent years, the You Only Look Once (YOLO) framework has emerged as a dominant force, owing to its remarkable balance between detection accuracy and inference speed. Initially introduced by Redmon et al., YOLO brought a paradigm shift by reframing object detection as a regression problem, predicting bounding boxes and class probabilities in a single evaluation. Over time, this architecture has undergone multiple iterations, giving rise to refined versions like YOLOv5, YOLOv7, and the more recent YOLOv8, v9, v10, and v12 series, each improving upon speed, computational efficiency, and model compactness.

In road damage detection, where identifying potholes, cracks, and surface irregularities with minimal latency is critical, YOLO's real-time capability makes it especially suitable. Researchers such as Wan [1] demonstrated how integrating attention modules into YOLOv5s resulted in a lightweight yet highly effective model (YOLO-LRDD), while Sami [2] applied ECA-Net and label smoothing to enhance YOLOv5 for road pavement monitoring, achieving superior mAP and F1-score compared to its original variant.

The domain of road infrastructure management is essential to public safety,

urban development, and economic efficiency. Regular inspections and early detection of road surface anomalies can significantly reduce the risk of traffic accidents and the long-term cost of maintenance. However, manual inspection is labor-intensive, time-consuming, and often infeasible at large scales especially in low-resource environments.

The adoption of automated object detection models within this domain has transformed traditional inspection pipelines. With advancements in mobile computing and edge AI, there is a pressing need for lightweight, real-time models deployable on portable devices. YOLO-based models have answered this call with encouraging results. For instance, Wu [3] introduced YOLO-LWNet for mobile platforms, while Guo [4] offered EADG-YOLO with enhanced edge-detection capabilities using attention-augmented modules. Furthermore, works like that of Vishwakarma [5] and Irsal [6] explored the integration of CSPNeXt and Swin Transformers into YOLO backbones, addressing scale variance and enhancing visual robustness. These innovations confirm YOLO's central role in building scalable road damage detection systems with real-world applicability. Its adaptability across datasets and architectural modularity enables developers to tune performance across varying computing budgets, camera resolutions, and regional infrastructure conditions.

Yet, a close examination of current literature reveals several critical gaps. Firstly, while many studies propose novel architectures or hybrid enhancements to YOLO, most models remain optimized for benchmark results and lack practical deployment strategies. Few works explicitly focus on model quantization and conversion for real-time use via TensorFlow Lite or other lightweight formats. For example, although Ding [7] introduced SCD-YOLO with remarkable accuracy improvements, the study stopped short of mobile or embedded deployment. Secondly, generalization across diverse road environments remains underexplored. Arya [8] attempted a multi-country dataset, yet many follow-up studies focused on narrow regional datasets like RDD2020 or UAV images from single geographies. This creates a bias in performance generalizability when applied to real-world scenarios. Thirdly, despite the proliferation of YOLO variants, few studies attempt direct comparisons among recent iterations such as YOLOv9s, YOLOv10s, and YOLOv12s under the same training conditions and evaluation criteria.

Moreover, several models rely on raw performance numbers like mAP, neglecting end-user considerations such as app integration, inference speed on mobile GPUs, or UI response time—all of which are vital in deploying practical road maintenance solutions.

To overcome these limitations, this study proposes a robust and comparative approach to road damage detection by evaluating and benchmarking three recent YOLO models: YOLOv9s, YOLOv10s, and YOLOv12s. Each model is trained on a consistently annotated road damage dataset that includes a diverse range of surface anomalies. After training, the best-performing model is exported as a .tflite file (Best_model_float32.tflite) to ensure compatibility with mobile platforms. This model is then integrated into a Flutter-based mobile application that allows end users to detect and visualize road damage in real-time through their smartphones.

1.2 Motivation

The motivation behind this research stems from the growing need for efficient and automated road damage detection systems that can be used in real-world scenarios, particularly on mobile devices. Roads are critical infrastructure, and damage such as cracks and potholes can lead to traffic accidents, vehicle damage, and costly repairs if not detected early. However, in many regions, especially in developing countries, manual road inspections are still the primary method used for monitoring road conditions. These traditional approaches are not only time-consuming and labor-intensive but also limited in coverage and consistency. Therefore, there is a clear demand for a faster, scalable, and cost-effective solution that can detect road damage accurately without the need for expensive equipment or expert intervention.

Recent advancements in deep learning, particularly the YOLO (You Only Look Once) family of models, have shown promising results in object detection tasks due to their high speed and accuracy. Despite this progress, most research has focused on building accurate models without considering real-time deployment challenges, such as model size, inference speed, and hardware limitations. Additionally, many studies use only one YOLO version or test their models on limited datasets without comparing different variants under the same

conditions. As a result, it remains unclear which YOLO model performs best when considering both accuracy and deployment efficiency for mobile-based road damage detection.

This study is motivated by the need to fill this gap. It aims to compare the performance of three recent YOLO models—YOLOv9s, YOLOv10s, and YOLOv12s—using a carefully annotated road damage dataset. The goal is to identify the most suitable model in terms of accuracy, processing speed, and mobile compatibility. The best model is then exported as a .tflite format and deployed in a user-friendly Flutter-based mobile application. This end-to-end system ensures that road damage can be detected directly on a smartphone, enabling real-time reporting and reducing dependence on manual inspections. By building a lightweight yet reliable application, this study contributes a practical solution to a real-world problem, especially useful for road authorities, municipal workers, and even everyday users who wish to report road conditions quickly and efficiently.

1.3 Objectives

The primary aim of this study is to develop an accurate, lightweight, and real-time road damage detection system using advanced YOLO models and integrate it into a mobile application for practical use. With the increasing need for automated infrastructure monitoring, this research focuses on building a solution that not only achieves high detection accuracy but also considers mobile deployment constraints such as model size and inference speed. Building upon the strengths of YOLO architecture and addressing the limitations observed in previous studies, this research sets the following specific objectives:

- To evaluate and compare the performance of three recent YOLO models—YOLOv9s, YOLOv10s, and YOLOv12s—on a road damage dataset in terms of detection accuracy, speed, and resource efficiency, under the same training and validation conditions.
- To select the best-performing model based on overall detection performance and convert it into a lightweight .tflite format using TensorFlow Lite, ensuring suitability for mobile or embedded deployment.

- To build and deploy a mobile application using Flutter that integrates the best model and allows end users to detect road damage in real-time from their smartphones with minimal latency.
- To address existing research gaps, particularly the lack of mobile-focused deployment, comparative evaluation of recent YOLO versions, and unified methodology for real-time road damage detection.

1.4 Methodology

The methodology of this study follows a structured pipeline designed to ensure accurate, real-time road damage detection with practical deployment in a mobile environment. The process begins with the collection and annotation of a road damage image dataset, focusing on various types of surface defects such as cracks and potholes. High-quality annotations are created to ensure precise bounding box representations for training. Once the dataset is prepared, three advanced object detection models—YOLOv9s, YOLOv10s, and YOLOv12s—are trained individually using the same dataset. Each model is evaluated based on key performance metrics including mean Average Precision (mAP), inference speed, and model size. The model that demonstrates the best trade-off between accuracy and efficiency is selected as the final model. This best-performing model is then converted into TensorFlow Lite format (best_model_float32.tflite) to ensure compatibility with mobile devices. Finally, the optimized model is integrated into a Flutter-based mobile application, enabling users to detect road damages in real-time through their smartphones. This deployment ensures that the system is lightweight, responsive, and practical for real-world use, especially in regions where manual inspection is not feasible or cost-effective. The end-to-end approach—from model training to mobile deployment—ensures that the study not only builds a high-performance detection system but also addresses critical real-world challenges in infrastructure monitoring.

1.5 Project Outcome

This project resulted in the successful development of a lightweight, accurate, and real-time road damage detection system that can be deployed on mobile devices. By evaluating three recent YOLO models—YOLOv9s, YOLOv10s, and YOLOv12s—on a custom annotated road damage dataset, the study identified

the model that offers the best balance between detection accuracy, speed, and computational efficiency. The chosen model was optimized and exported as a TensorFlow Lite (.tflite) format to enable smooth performance on low-resource devices. One of the most significant outcomes is the creation of a Flutter-based mobile application that integrates the trained model, allowing users to scan roads in real time and receive instant feedback on detected damages. The application is designed to be user-friendly and functional in practical environments, making it useful for road maintenance authorities, municipal agencies, and even general users. The end-to-end pipeline developed in this project—from dataset preparation and model evaluation to mobile deployment—can serve as a reference framework for future research and real-world solutions in infrastructure monitoring. Ultimately, this study contributes a practical tool that addresses the need for scalable and affordable road condition assessment, particularly in resource-constrained settings.

1.6 Organization of the Report

This report is organized into six chapters, each addressing a key component of the research work conducted in this study titled "Automated Road Damage Detection Using YOLO Models and Mobile Deployment Framework." The structure of the report is designed to reflect the logical flow of the project—from problem identification to implementation and real-world application.

Chapter 1: Introduction
This chapter presents the background of the study, outlines the motivation behind the project, and clearly states the research objectives. It also introduces the methodology followed throughout the research and highlights the key outcomes of the project. The chapter concludes with an overview of the overall structure of the report.

Chapter 2: Background
This chapter covers the theoretical and technical background related to road damage detection and object detection frameworks. A comprehensive literature review is included, analyzing previous research works that applied YOLO and other deep learning models for infrastructure monitoring. The section also highlights existing gaps in the literature which this study aims to address.

Chapter 3: Research Methodology

This chapter provides a detailed explanation of the methodology used in the research. It includes requirement analysis, system design, functional and non-functional requirements, and the proposed detection pipeline using YOLOv9s, YOLOv10s, and YOLOv12s. It also illustrates data flow diagrams, UI layout, and the step-by-step training and deployment process. Additionally, this chapter outlines the overall project planning and task allocation strategies.

Chapter 4: Implementation and Results

This chapter describes the development environment setup, training and evaluation of different YOLO models, and the final selection of the best-performing model. The selected model is then converted into .tflite format and deployed into a Flutter mobile application. The results are presented through accuracy metrics and performance comparisons, followed by a discussion on the practical implications of the findings.

Chapter 5: Engineering Standards and Design Challenges

This chapter focuses on the technical standards and constraints considered during system development. It addresses relevant software, hardware, and communication standards, and discusses the broader impact of the project on society, environment, and ethical considerations. The chapter also maps the project to complex engineering problem-solving and project management practices.

Chapter 6: Conclusion

The final chapter summarizes the entire research work, discusses limitations encountered during the study, and suggests potential future enhancements. It highlights the real-world application of the developed system and how the proposed solution can be scaled or improved for further research and deployment.

Chapter 2

Background

This chapter explores the theoretical and technical foundations related to road damage detection using deep learning and computer vision. It begins with an introduction to the importance of road surface analysis, followed by a literature review of relevant research works, including both similar real-world applications and advanced model development efforts. The chapter concludes with a gap analysis that highlights the limitations in existing studies and establishes the rationale for the methodology proposed in this thesis.

2.1 Introduction

The Road infrastructure plays a vital role in the economic and social development of any nation. Safe and well-maintained roads ensure efficient transportation, reduce vehicle damage, and prevent accidents. However, due to environmental effects, increased vehicle load, and poor maintenance practices, roads often deteriorate over time, leading to issues such as cracks, potholes, and surface deformations. Early detection of such damage is essential for cost-effective repair and timely intervention. Traditional road inspection methods rely heavily on manual labor and visual inspections, which are time-consuming, inconsistent, and often limited in scope, especially in countries with limited infrastructure budgets.

In response to these limitations, recent developments in artificial intelligence (AI) and computer vision have led to the emergence of automated road damage detection systems. Object detection models, particularly those in the YOLO (You Only Look Once) family, have shown remarkable effectiveness due to their ability to perform detection and classification tasks in real-time. These models are designed to balance accuracy and speed, making them highly suitable for applications in road monitoring where real-time feedback is essential. Over successive versions, YOLO models have become faster, lighter, and more accurate, enabling integration into edge devices and mobile platforms.

Despite these advancements, a majority of existing research still focuses on improving model performance within controlled environments or benchmark datasets, without extending their findings into practical, deployable applications. Several studies have developed high-performing YOLO-based models or enhanced them using attention mechanisms, yet very few have fully addressed real-time mobile deployment or converted models to lightweight formats like TensorFlow Lite (.tflite). Additionally, while some research compares a few YOLO variants, there is limited work systematically evaluating the latest models such as YOLOv9s, YOLOv10s, and YOLOv12s under a unified experimental setup.

This chapter presents a detailed background of the field by exploring similar real-world applications, state-of-the-art deep learning-based damage detection models, and relevant technical developments. It also examines research that contributed to foundational understanding and feature optimization. Through this, the chapter establishes the knowledge base required for understanding the scope and novelty of the proposed research work while identifying the existing gaps in literature that motivate the solution proposed in this thesis.

2.2 Literature Review

Wan [1] introduced YOLO-LRDD, a lightweight road damage detection model based on an improved YOLOv5s framework. By incorporating the Efficient Channel Attention (ECA) module into ShuffleNetV2, they created a novel backbone called Shuffle-ECANet. The integration of BiFPN improved feature fusion, and the Focal-EIOU loss function enhanced anchor box quality during training. Evaluated on the RDD2020 dataset, the model achieved a 22.3% increase in speed and a 28.8% reduction in size compared to YOLOv5s, all while maintaining accuracy, making it highly suitable for mobile deployment.

Li [9] proposed RDD-YOLO, an enhanced version of YOLOv8 that integrates the SimAM attention mechanism and GhostConv layers to boost precision and reduce computational load. Bilinear interpolation was employed for improved upsampling. The RDD2022 dataset was used, and the model achieved mAP50 and mAP50-95 scores of 62.5% and 36.4%, respectively, with an F1-score of 69.6%, outperforming the baseline by 2.8%. This study demonstrates an efficient

solution for accurate road condition monitoring. Wang [10] addressed the challenges of small-scale and irregular road damage detection by proposing YOLO-RD. The framework integrated the Star Operation Module (SOM), Multi-dimensional Auxiliary Fusion (MAF), and Wavelet Transform Convolution (WTC) for detailed feature extraction and robustness. On the RDD2022 Japanese dataset, YOLO-RD achieved 25.75% accuracy with a 4.93% improvement for small object detection over the YOLOv8 baseline, showcasing its suitability for complex, real-world road environments.

Jeong and Kim [11] leveraged multi-source data collected from six countries to create a robust ensemble detection framework using YOLOv5x models and image tiling techniques. Their model focused on four road damage types and achieved an average F1-score of 0.6744 and a real-time performance of 1 FPS, highlighting the benefit of ensemble modeling in high-resolution multi-source road damage imagery. Wu [3] developed YOLO-LWNet, a lightweight road damage detection algorithm optimized for mobile devices. The proposed LWC module, along with custom-designed backbone and feature fusion networks, enabled efficient inference. Two versions—small and tiny—were tested on the RDD2020 dataset, and the model outperformed YOLOv5 and YOLOv6 in terms of accuracy, scale, and computational complexity, proving effective in resource-constrained environments.

Zhang [12] enhanced YOLOv3 using a Multi-Level Attention Block (MLAB) to improve feature focus. Tested on UAV-captured images, the modified network achieved a mAP of 68.75%, significantly outperforming the original YOLOv3's 61.09%. The model showed high accuracy in detecting various crack types, particularly improving the detection of alligator and oblique cracks, affirming its utility for aerial road surveillance. Xie [13] proposed YOLO-SFT, a road damage detection algorithm that employs the StarNet-context anchor attention (Star-CAA), focusing diffusion cross-stage (FDCS) pyramid network, and task-align dynamic (TAD) detection head. These architectural innovations led to a 5.6% improvement in mAP and a 4.0% boost in F1-score. The model also reduced computational load by 16.9% while maintaining real-time capability with a speed of 64.5 FPS, making it a practical option for intelligent transportation systems. Ding [7] introduced SCD-YOLO, a novel crack detection model based on

YOLOv8n. By integrating the Shift-Wise Convolutional Feature module (SWC2f), Cross-Scale Attention Fusion Module (CCAFM), and the Dynamic Head detection module (Dyhead), the model effectively handled multi-scale and complex road crack patterns. Evaluated on the RDD2022 dataset, SCD-YOLO achieved a 4.1% mAP improvement, affirming its robustness and efficiency in diverse geographic scenarios.

Sami [2] modified YOLOv5 by integrating ECA-Net, label smoothing, K- means++, Focal Loss, and additional prediction layers for enhanced real-time pavement damage detection. The improved model, tested on RDD2022, achieved a 1.9% mAP and 1.29% F1-score improvement over YOLOv5s and even surpassed YOLOv8s with fewer parameters and reduced computational cost, making it both accurate and resource-efficient. Guo [4] developed EADG-YOLO, an improved YOLOv8-based model for edge-aware road damage detection. The model used EIEStem for edge feature extraction, multi-scale attention to suppress irrelevant information, DySample for refined upsampling, and a lightweight detection head. It achieved an mAP@0.5 of 87.3% on the RDD2022 dataset while reducing computational cost by 7%, showing promise for deployment in embedded systems and mobile platforms. Wang [14] employed Faster R-CNN to detect road damage using smartphone-captured images. To handle class imbalance, they used augmentation techniques such as brightness and contrast adjustments. After optimizing model parameters based on damage size and aspect ratios, their approach achieved a Mean F1-score of 0.6255 in a road damage challenge, proving the viability of traditional object detectors in practical applications.

Samma [15] addressed the complexity of pre-trained CNNs like VGG-19 by developing a two-layer optimizer that prunes filters based on SVM-guided accuracy. Applied to a drone-based image dataset with 529 images, their evolved lightweight model reduced filter usage by 52% and achieved a high F1-score of 96.4%. This optimizer also outperformed several metaheuristic algorithms, showcasing a novel way to simplify and adapt deep models for aerial road inspections. Xu [16] proposed CrdNet, a cascade CNN-based framework tailored for road damage detection. The model integrates LrNet as a backbone to preserve weak semantic information and applies multi-aspect ratio anchor mechanisms to capture irregular damage shapes. By employing adaptive proposal assignment

and cascade predictions, CrdNet reached a remarkable mAP of 90.92% on a custom dataset, demonstrating its effectiveness in detecting complex road anomalies. Kulambayev [17] introduced a real-time road surface damage detection system utilizing the Mask R-CNN architecture, particularly known for its superior instance segmentation capabilities. This framework was specifically fine-tuned to handle diverse lighting and environmental conditions across urban, suburban, and rural roads. Their comprehensive dataset and tailored model enabled accurate classification and segmentation of road anomalies like potholes and rutting. The study emphasized scalability, real-time performance, and adaptability for integration into smart maintenance systems and vehicular aids, making it a highly practical contribution to infrastructure monitoring.

Chen [18] further explored the potential of Mask R-CNN by enhancing it with DenseNet as a backbone for more effective feature extraction. Their pipeline included a feature pyramid network (FPN), a region proposal network (RPN), and a fully convolutional head for segmentation and classification. This method allowed not only detection and classification but also precise damage masking, achieving better results than existing conventional models. Their work addressed challenges related to complex road backgrounds and structural variation in damage patterns. Ale [19] assessed the speed-accuracy tradeoff in road damage detection, comparing two-stage detectors like Faster R-CNN with one-stage models. Their findings favored RetinaNet—a one-stage detector—which offered a good balance of speed and accuracy for practical implementation. Despite being lighter and faster than its counterparts, RetinaNet maintained robust detection capabilities, making it ideal for real-time infrastructure monitoring in constrained environments. Arya [8] addressed the lack of cross-country applicability in road damage models by creating a large-scale, heterogeneous dataset comprising over 26,000 images from India, Japan, and the Czech Republic. Their deep learning-based framework assessed model generalization across geographies and offered a publicly available benchmark through the Global Road Damage Detection Challenge (GRDDC) 2020. Their work laid the foundation for adaptable road monitoring systems in regions lacking high-end hardware and expertise.

Vishwakarma [5] leveraged RTMDet—a one-stage object detection model—for real-time and efficient road damage detection. The model was enhanced through the CSPNeXt backbone, dynamic augmentation switching, and effective transfer learning. RTMDet outperformed YOLOv8 by 5% in mAP50 on the CRDDC 2022 dataset. Notably, it achieved F1-scores of 0.57 (RTMDet-Large) and 0.58 (RTMDet-Medium), while maintaining inference speeds under 0.07 seconds/image, establishing a benchmark for high-accuracy, low-latency detection in real-world deployments. Irsal [6] introduced the YOLOv7-Swin model, which integrates Swin Transformer modules into the CSP component of YOLOv7 to boost detection accuracy under diverse scenarios. They evaluated various YOLO versions (YOLOv5l, YOLOv6l, YOLOv7-tiny, YOLOv7, YOLOv7x) and found that YOLOv7-Swin struck an optimal balance with mAP@0.50 of 0.47 and mAP@0.5:0.95 of 0.232. The model retained relatively fewer parameters while outperforming heavier configurations, proving efficient and powerful for complex detection tasks. Desman [20] evaluated the performance of a Vision Transformer-based architecture (YOLOS) trained on RDD2022, focusing on the role of data cleaning. Their ViT-based model showed strong performance in bounding box precision and object consistency under variable lighting conditions. It achieved AP scores of 62.1% (IoU=0.5), 37.1% (IoU=0.75), and 36.2% (IoU=0.5:0.95), along with AR values of 42.1% (Large), 60.3% (Medium), and 75.4% (Small), indicating excellent scalability across object sizes and complexity levels. Roy and Bhaduri [21] proposed DenseSPH-YOLOv5, a high-performance detection model that integrates DenseNet blocks for feature reuse and CBAM for attention optimization. A Swin Transformer prediction head further enhanced detection across multiple object scales. Trained on RDD-2018, the model achieved a mAP of 85.25%, F1-score of 81.18%, and precision of 89.51% at 62.4 FPS. These results outperformed contemporary models and offered robust detection in noisy environments, supporting real-time deployment in field conditions.

Anzum [22] developed a road crack classification system based on the Data-Efficient Image Transformer (DeiT). Using the Kaggle Concrete dataset, their DeiT-based approach outperformed well-known models like YOLOv5,

YOLOv8, Xception, ResNet50, and MobileNetV2. It achieved an outstanding accuracy of 99.75%, confirming the potential of Transformer-based models for lightweight, high-accuracy applications in binary crack classification. Jeong [23] contributed to the ORDDC 2024 by evaluating a range of deep learning models on a multi-country dataset. Their ensemble model based on Faster R-CNN achieved an F1-score of 71.65% but with slower inference (0.3012 sec/image), while a YOLOv9 model offered a faster alternative with 62.02% F1-score and a much lower inference time (0.0340 sec/image). This study highlighted the trade-off between speed and accuracy and emphasized the viability of YOLOv9 for real-time tasks. Ashraf [24] proposed a comprehensive pavement crack detection framework using multi-scale feature aggregation and transformer-based attention. Their model, trained on 10,000 images, significantly improved across all metrics after incorporating attention mechanisms. Detection precision rose from 88.7% to 94.3%, IoU from 78.8% to 93.2%, and segmentation Dice score reached 94.7%. These results confirm the value of transformers and multi-scale design in complex surface analysis and segmentation.

Lastly, Yang [25] applied the Vision Transformer (ViT) architecture to classify road cracks, emphasizing its superiority in handling long-distance feature correlations over CNNs. Histogram equalization and data augmentation were used to enhance image quality. The ViT model outperformed ResNet, DenseNet, and EfficientNet across classification metrics such as accuracy, F1-score, and recall, confirming its ability to handle transverse and longitudinal cracks effectively.

Table 2.1: Research Matrix of Potato Leaf Disease Detection Studies

Author	Year	Accuracy	Model Used	Contribution
Wan et al.	2022	Comparable to YOLOv5s with 22.3% speed improvement	YOLO-LRDD (YOLOv5s + Shuffle-ECANet)	Lightweight road damage detection model for mobile use
Li et al.	2024	mAP50: 62.5%, mAP50-95: 36.4%, F1: 69.6%	RDD-YOLO (YOLOv8 + SimAM, GhostConv)	Improved attention and feature simplification for better accuracy

Wang et al.	2025	25.75% accuracy, +4.93% small object detection	YOLO-RD (YOLOv8 + SOM, MAF, WTC)	Improved small damage and irregular shape detection
Jeong & Kim	2022	F1-score: 0.6744	YOLOv5x Ensemble	Image tiling and multi-source data for robust detection
Wu et al.	2023	Not specified	YOLO-LWNet	Lightweight detector for mobile devices using LWC module
Zhang et al.	2022	mAP: 68.75%	YOLOv3 + MLAB	Improved detection with UAV images using multi-level attention
Xie et al.	2025	F1 +4%, mAP +5.6%	YOLO-SFT	Feature diffusion, Star-CAA, FDCS for better accuracy and speed
Ding et al.	2024	+4.1% mAP	SCD-YOLO (YOLOv8n + SWC2f, Dyhead)	Improved crack detection using shift-wise attention modules
Sami et al.	2023	+1.9% mAP, +1.29% F1	YOLOv5 + ECA-Net	Real-time road damage detector with fewer parameters than YOLOv8
Guo et al.	2025	mAP@0.5: 87.3%	EADG-YOLO (YOLOv8 + EIEStem, DySample)	Edge-aware lightweight detection for embedded systems
Wang et al.	2018	F1-score: 0.6255	Faster R-CNN	Data augmentation and bounding box optimization for road detection
Samma et al.	2021	F1-score: 96.4%	VGG-19 + Two-layer optimizer	Filter pruning with SVM for drone image damage detection
Xu et al.	2021	mAP: 90.92%	CrdNet (CNN + Cascade Detection)	Length-aware high-quality detection using custom LrNet backbone
Ale et al.	2018	Relatively high	RetinaNet	One-stage fast model with balanced performance
Arya et al.	2021	Not specified	YOLO/Faster R-CNN	Cross-country dataset and evaluation model for global use
Vishwakarma	2024	F1: 0.57–0.58	RTMDet	Augmentation strategy + pretrained

				models for fast accurate detection
Irsal et al.	2024	mAP@0.5: 0.47	YOLOv7-Swin	Swin Transformer added to YOLOv7 for balanced accuracy & complexity
Desman et al.	2025	AP@IoU=0.5: 62.1%, AR Small: 75.4%	YOLOS (Vision Transformer)	Light-aware detection with ViT
Roy & Bhaduri	2023	mAP: 85.25%, F1: 81.18%	DenseSPH-YOLOv5	DenseNet + CBAM + SwinTransformer for real-time detection
Anzum et al.	2024	Accuracy: 99.75%	DeiT	Transformer-based binary crack classifier
Jeong et al.	2024	F1: 71.65% (Faster R-CNN), 62.02% (YOLOv9)	Faster R-CNN, YOLOv9	Trade-off between speed and accuracy evaluated
Ashraf et al.	2024	Precision: 94.3%, IoU: 93.2%	Transformer + Multi-scale Aggregation	Segmentation, classification, detection with high performance

2.2.1 Similar Applications

Several recent studies have explored real-time road damage detection systems with practical or field-oriented deployment goals, similar to the application proposed in this thesis. These works generally combine lightweight object detection models with mobile-friendly or real-time implementations, making them highly relevant to this study.

Wan [1] developed YOLO-LRDD, a lightweight model based on YOLOv5s with a modified ShuffleNetV2 backbone integrated with an ECA attention module. The model was specifically designed to enhance inference speed and reduce model size, achieving a 22.3% faster recognition rate than standard YOLOv5s while maintaining comparable accuracy. Their work directly aligns with mobile deployment needs. Wu [3] proposed YOLO-LWNet, another lightweight model aimed at mobile terminal devices. With a custom LWC module and reduced model complexity, their study focused on balancing detection performance with low computational demand, a goal that mirrors the mobile-focused motivation of the current research.

Guo [4] extended this application-centric approach by proposing EADG-YOLO, an edge-aware YOLOv8-based model, which was optimized for embedded deployment using a reduced detection head and dynamic upsampling. Their study reported a strong mAP of 87.3% while reducing computation by 7% compared to baseline YOLOv8, highlighting both accuracy and resource efficiency. Vishwakarma [5] developed RTMDet, evaluated on the CRDDC2022 dataset, and optimized it for fast deployment in cloud and edge containers. Their best models achieved F1 scores of 0.57–0.58 and inference speeds under 0.07 seconds per image, reinforcing the effectiveness of lightweight and fast detectors for real-world use. Similarly, Sami [2] enhanced YOLOv5 by incorporating efficient attention modules and performance-boosting techniques like label smoothing and K-means++ clustering. Their model outperformed YOLOv8 in accuracy while reducing parameter count and computational cost, directly supporting mobile or embedded application needs. Lastly, the study by Jeong [11] explicitly targeted optimized deployment and benchmarked YOLOv9 against Faster R-CNN, achieving a much faster inference time (0.034s) with reasonable accuracy (F1-score: 62.02%), thus validating its suitability for real-time mobile applications.

Table 2.2: Similar Applications Matrix

Author(s)	Year	Model Used	Focus/Contribution	Deployment Focus
Wan et al.	2022	YOLO-LRDD (YOLOv5s + ECA + BiFPN)	Lightweight model with 22.3% speed boost, 28.8% size reduction	Mobile/real-time
Wu et al.	2023	YOLO-LWNet	Designed for mobile use with lightweight LWC modules	Mobile device
Guo et al.	2025	EADG-YOLO (YOLOv8 + EIEStem)	Achieved 87.3% mAP with 7% less computation	Embedded/mobile
Vishwakarma	2024	RTMDet (CSPNeXt backbone)	F1: 0.57–0.58, fast inference (0.049–0.070s/image)	Cloud/mobile container
Sami et al.	2023	YOLOv5 + ECA + label smoothing	Outperformed YOLOv8 with fewer parameters and better mAP	Lightweight deployment

Jeong et al.	2024	YOLOv9 vs Faster R-CNN	YOLOv9 had fast inference (0.034s/image), suitable for real-time	Real-time optimization

2.2.2 Related Research

While many recent studies have targeted practical applications, several others have significantly contributed to the underlying research in model design, attention mechanisms, feature enhancement, and dataset development for road damage detection. These foundational works provide valuable insights into the strengths and limitations of various deep learning approaches, especially those involving the YOLO family and Transformer-based models.

Wang [10] introduced YOLO-RD, an advanced framework built on YOLOv8 and designed to improve small-scale damage detection and adapt to irregular surface patterns. The model integrated modules such as the Star Operation Module (SOM) and Multi-dimensional Auxiliary Fusion (MAF), achieving a 4.93% improvement in small object detection. Similarly, Xie [13] proposed YOLO-SFT by replacing the default backbone and neck structure with custom modules like Star-CAA and FDCS, improving mAP by 5.6% and increasing the F1-score by 4%. These enhancements contributed to better gradient flow, context diffusion, and real-time performance. Roy and Bhaduri [21] designed DenseSPH-YOLOv5 by integrating DenseNet blocks with CBAM attention and a Swin Transformer head. Their model achieved a high mAP of 85.25% and F1-score of 81.18% on RDD2018, with notable improvements in multiscale detection accuracy. Likewise, Ding [7] presented SCD-YOLO, which embedded Shift-Wise Convolution and a dynamic head mechanism into YOLOv8, resulting in a 4.1% increase in mAP while maintaining lightweight efficiency.

Desman [20] investigated the use of Vision Transformers (ViTs) for road damage detection through a model called YOLOS, trained on the RDD2022 dataset. The study demonstrated strong performance across object sizes and lighting conditions, showing AP scores of 62.1% (IoU=0.5) and high recall for small and medium objects. Meanwhile, Yang [23] also explored the application of Vision Transformers in road crack classification, highlighting their superior ability to

capture long-range dependencies when compared to CNNs. Additionally, foundational works like that of Samma [15] introduced a two-layer optimizer to prune VGG-19 filters for drone-based image analysis, achieving 96.4% F1-score with reduced complexity. Xu [16] contributed CrdNet, a CNN-based cascade detection model that incorporated multi-aspect anchor mechanisms and achieved 90.92% mAP, demonstrating the power of tailored architectures for complex detection challenges. Ashraf [24] further pushed the boundary by integrating multi-scale feature aggregation with Transformer-based attention for crack segmentation, achieving over 94% precision and IoU.

Table 2.3: Related Research Matrix

Author(s)	Year	Model Used	Key Contribution	Accuracy / Performance
Wang et al.	2025	YOLO-RD (YOLOv8 + SOM + MAF)	Enhanced detection of small and irregular damages	+4.93% small object detection
Xie et al.	2025	YOLO-SFT (Star-CAA, FDCS)	Improved gradient flow and feature diffusion	+5.6% mAP, +4.0% F1-score
Roy & Bhaduri	2023	DenseSPH-YOLOv5	DenseNet + Swin + CBAM attention for better multiscale detection	mAP: 85.25%, F1: 81.18%
Ding et al.	2024	SCD-YOLO (SWC2f, DyHead)	Shift-wise convolution + dynamic head for efficiency	+4.1% mAP
Desman et al.	2025	YOLOS (ViT)	Vision Transformer for size-consistent road damage detection	AP@0.5: 62.1%, AR Small: 75.4%
Yang et al.	2023	Vision Transformer vs CNNs	Long-range crack classification via ViT	ViT > ResNet, DenseNet, EfficientNet
Samma et al.	2021	VGG-19 + Two-layer optimizer	Reduced filters, high drone-based crack detection	F1-score: 96.4%
Xu et al.	2021	CrdNet (Cascade CNN)	Length-aware detection, adaptive anchor strategy	mAP: 90.92%
Ashraf et al.	2024	Transformer + Multi-scale Attention	Crack segmentation, detection, and classification	Precision: 94.3%, IoU: 93.2%

2.3 Gap Analysis

The existing literature on road damage detection has shown significant progress in developing object detection models using deep learning, particularly the YOLO family. However, several important limitations remain unaddressed, which create gaps in the current state of research and justify the need for this study. Most of the works reviewed, including those by Wan [1], Wu [3], and Guo [4], successfully developed lightweight YOLO-based models, but they often stopped short of full mobile deployment or practical real-time implementation for end users.

Another critical gap lies in the lack of comparative evaluation between the latest YOLO variants. Works such as those by Sami [2] and Jeong [11] evaluated YOLOv5 and YOLOv9 in isolation or in comparison to older models like Faster R-CNN, but no existing study has conducted a systematic comparison of YOLOv9s, YOLOv10s, and YOLOv12s under the same dataset and training conditions. This restricts our understanding of which model offers the best trade-off between accuracy and efficiency, especially when targeting real-time mobile applications. Moreover, many Transformer-based models, including YOLOS (Desman [20] and DenseSPH-YOLOv5 (Roy & Bhaduri, [21]), focus on enhancing attention mechanisms or improving feature extraction, yet they do not address lightweight conversion or deployment. Similarly, studies using ViT (Yang [25]) or multi-scale aggregation (Ashraf [24]) show excellent accuracy but remain in the theoretical or experimental domain without demonstrating usability on edge or mobile platforms. Additionally, while Arya [8] attempted to build a global dataset, most works still train on single-region datasets and do not account for diverse visual environments. The gap in generalized performance across varying lighting, background noise, and road textures makes the models less robust in real-world use.

Table 2.4: Gap Summary Table

Identified Gap	Supporting Studies	How This Study Addresses It
Lack of real mobile deployment despite claims of lightweight models	Wan et al. (2022), Wu et al. (2023), Guo et al. (2025)	Converts the best YOLO model to .tflite and deploys it in a Flutter-based mobile app

No unified comparison between YOLOv9s, YOLOv10s, and YOLOv12s	Sami et al. (2023), Jeong et al. (2024)	Compares all three YOLO versions under identical conditions to identify the best performer
Strong detection models not tested for edge/mobile or real-time use	Roy & Bhaduri (2023), Ding et al. (2024), Desman et al. (2025)	Evaluates both accuracy and inference speed, focusing on real-time performance
Transformer-based models focus on accuracy, ignore deployment	Yang et al. (2023), Ashraf et al. (2024)	Prioritizes lightweight model export and on-device usability rather than lab-only results
Limited geographic generalization and data diversity	Arya et al. (2021)	Uses a custom dataset annotated for realistic road environments and visual variation

2.4 Summary

This chapter reviewed the key technological advancements and research contributions in road damage detection using deep learning models. Several studies have proposed lightweight and high-accuracy models for damage detection, particularly using YOLO-based architectures and attention-enhanced feature extraction techniques. While many works showed promising results in terms of precision and efficiency, most lacked real-world deployment strategies such as mobile integration and lightweight format conversion. Furthermore, only a few studies attempted comparative evaluation between the latest YOLO variants or prioritized generalization across varying visual environments. Through a comprehensive gap analysis, this chapter established that there remains a significant need for a unified, end-to-end pipeline that combines model evaluation, real-time deployment, and usability on mobile devices—precisely the direction taken by this thesis.

Chapter 3

Research Methodology

This chapter presents the detailed methodology and design specifications followed to develop the proposed road damage detection system. It outlines the system architecture, requirement analysis, model selection strategy, and mobile application development process. The chapter also includes functional and non-functional requirements, UI design, context and data flow diagrams, and task allocation. These components collectively describe how the YOLO models were trained, evaluated, and deployed into a real-time Flutter-based mobile application, aligning the system design with the project's goals and real-world applicability.

3.1 Methodology

3.1.1 Overview

This research proposes a robust road damage detection framework leveraging the YOLO (You Only Look Once) family of object detection models for real-time and mobile-friendly deployment. The methodology is structured into several key phases, starting with the use of a publicly available dataset consisting of 2,208 annotated images categorized into three damage classes: crack, patch, and pothole. The dataset undergoes annotation-based preprocessing before being divided into training (1,980 images), validation (145 images), and test (83 images) sets. Three YOLO variants—YOLOv9s, YOLOv10s, and YOLOv12s— are trained and evaluated to identify the most effective model based on detection accuracy and inference speed. The best-performing model is then converted into a lightweight float32.tflite format and integrated into a Flutter-based mobile application, enabling real-time road damage detection for end-users through smartphone cameras.

3.1.2 Proposed Methodology

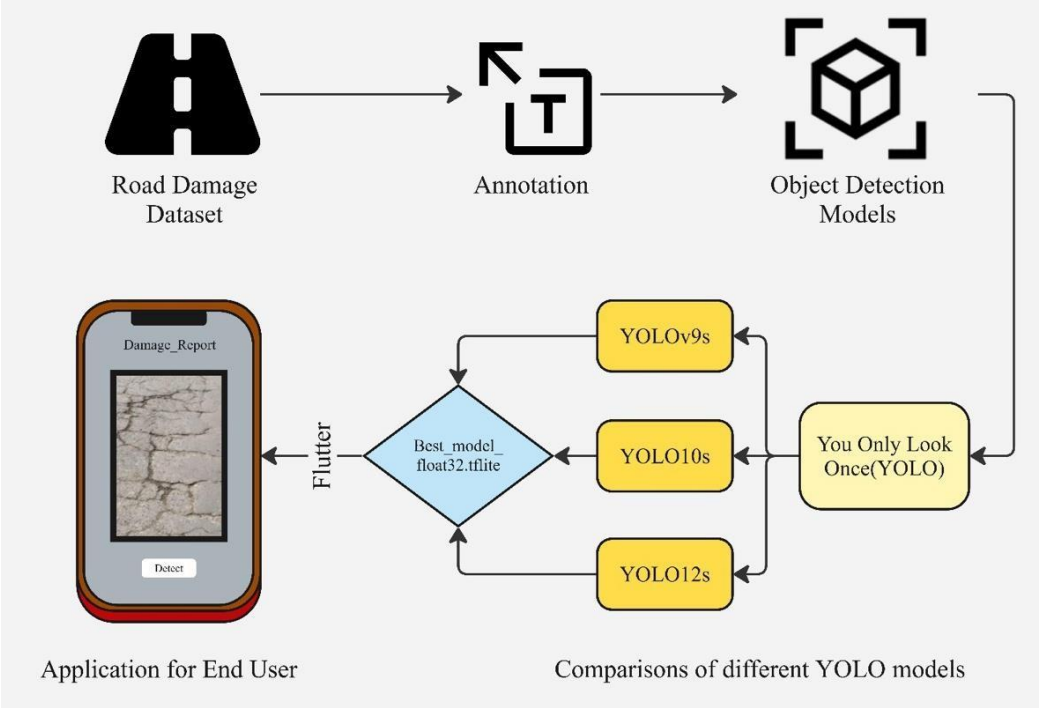


Figure 3.1: The Methodological Flowchart

Figure 3.1 illustrates the complete methodological workflow of the proposed road damage detection system. The process begins with acquiring a road damage image dataset, which is then annotated to mark the regions corresponding to different damage types—crack, patch, and pothole. These annotated images train three YOLO-based object detection models: YOLOv9s, YOLOv10s, and YOLOv12s. After evaluating the performance of each model, the best-performing one is selected and converted into a TFLite format (best_model_float32.tflite) for lightweight deployment. This optimized model is then integrated into a Flutter-based mobile application that allows end-users to detect and report road damage in real-time using their smartphones.

3.1.3 Functional and Nonfunctional Requirements

To ensure the development of a reliable and efficient road damage detection system, both functional and nonfunctional requirements were carefully identified and addressed. These requirements guided the implementation of the object detection model, system deployment, and user interaction through a mobile application.

Functional Requirements

Functional requirements define the core features and operations of the system. The key functional requirements include:

- Image Input and Preprocessing: The system must accept road images captured by the user or pre-collected images from the dataset. These images undergo preprocessing through annotation to label the specific damage types: crack, patch, and pothole.
- Damage Detection and Classification: The core functionality of the system is to detect road surface damage in images using trained YOLO object detection models. It should accurately localize and classify multiple damage types within a single frame.
- Model Comparison and Evaluation: The system must support training and evaluation of multiple YOLO variants (YOLOv9s, YOLOv10s, YOLOv12s) on a common dataset, allowing comparison based on metrics such as precision, recall, F1-score, and inference time.
- Model Export and Optimization: The best-performing model must be converted into a TensorFlow Lite (.tflite) format to ensure compatibility with mobile devices while maintaining accuracy and speed.
- Mobile Application Integration: A Flutter-based mobile application must integrate the optimized model, enabling real-time road damage detection through the user's smartphone camera. Users should be able to capture or upload an image and receive instant detection results.
- User Interaction and Feedback: The app should allow users to interact with the detection interface, view the type and location of damage, and potentially log or report detected damage for infrastructure management purposes.

Nonfunctional Requirements

Nonfunctional requirements focus on the system's quality attributes, such as performance, scalability, and usability. The key nonfunctional requirements include:

- Performance: The system must exhibit high detection accuracy with minimal false positives and false negatives. It should also ensure low latency during inference to support real-time detection on mobile devices.

- Scalability: The architecture should be designed in a modular fashion to allow the integration of future model versions or additional damage types without major re-engineering.
- Portability: The system should be portable across Android devices by using the Flutter framework and TensorFlow Lite, which ensures cross-platform compatibility and efficient performance.
- Usability: The user interface of the mobile application should be intuitive and accessible to non-technical users, especially field workers or citizens reporting road damage.
- Maintainability: The system should be easy to maintain, with well-documented code, modular structure, and support for future updates, such as model retraining or new dataset integration.
- Efficiency: The application and model must be optimized to consume minimal device resources such as memory, battery, and processing power, ensuring smooth operation on low-end to mid-range smartphones.

By fulfilling these functional and nonfunctional requirements, the proposed road damage detection system aims to provide a scalable, accurate, and user-friendly solution for real-time infrastructure monitoring and maintenance planning.

3.1.4 Data Flow Diagram Level 1

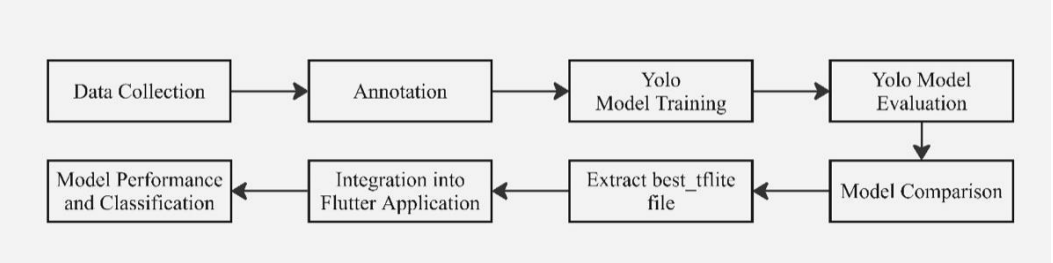


Figure 3.2: Data Flow Diagram Level 1

Figure 3.2 represents the Level 1 Data Flow Diagram of the proposed road damage detection system. The process begins with data collection, where road surface images are gathered for analysis. These images are then passed through an annotation phase, labeling each image with relevant damage types such as cracks, patches, or potholes. The annotated dataset is used to train various YOLO models, which are subsequently evaluated to measure their accuracy, speed, and robustness in

detecting damages. Following model evaluation, a comparison is conducted among different YOLO variants (YOLOv9s, YOLOv10s, YOLOv12s) to determine the most suitable deployment model. The best-performing model is then extracted in. tflite format for mobile compatibility and is integrated into a Flutter application. Finally, the system undergoes a thorough performance validation and classification test to ensure real-time usability and effectiveness in road damage detection for end users.

3.1.5 UI Design

The user interface (UI) design plays a pivotal role in ensuring accessibility, usability, and smooth interaction between the user and the backend functionalities of the road damage detection system. The mobile application developed using the Flutter framework presents a clean and intuitive interface where users can capture or upload road images and receive instant feedback on detected damage types. The UI is designed to be minimalistic yet functional, making it suitable for both technical and non-technical users such as field inspectors or road maintenance workers. The design incorporates a two-tier architecture to clearly separate the frontend presentation layer from the backend processing logic, ensuring modularity, maintainability, and scalability.

1. Application Setup

The mobile application for road damage detection is developed using the Flutter framework, which allows for cross-platform compatibility, particularly on Android and iOS devices. The application setup is designed to ensure simplicity, usability, and direct functionality for end users, especially those involved in fieldwork such as road inspectors, engineers, or public utility staff.

Upon launching the app, the user is greeted with the “Damage Detector” interface. This screen includes two primary input options: “Capture Image”, which activates the device’s camera for real-time image capture, and “Import from Library”, which allows the user to select an existing image from the gallery. These functionalities enable flexibility in data input based on user context and need.

Once the image is selected or captured, the “Detect” button becomes active. Pressing triggers the backend inference engine, which loads the optimized .tflite model and performs object detection to identify the damage types. The results are then visualized in the “Damage Report” screen, where the image is displayed along with

bounding boxes highlighting the detected damages. Each box is labelled with the damage type (e.g., "Crack") and a corresponding accuracy score or confidence level to reflect the model's certainty in its prediction.

2. Two-Tier Architecture

The proposed road damage detection system employs a Two-Tier Architecture that distinctly separates the Frontend (Presentation Layer) from the Backend (Processing Layer), facilitating both user interaction and computational efficiency.

The Frontend Tier is responsible for user engagement and input management. It consists of a graphical interface built with Flutter that allows users to capture or select an image and initiate the detection process. The frontend ensures a user-friendly experience, featuring clearly labeled buttons, a modern layout, and minimalistic design to keep interactions straightforward and efficient. Once the image is selected, the user interface communicates with the backend for processing and subsequently presents the results visually.

The Backend Tier handles the heavy-lifting tasks related to image processing and object detection. When the frontend sends an image, the backend initiates a series of operations, starting with Image Pre-processing, where the input image is resized, normalized, and reshaped to fit the model's required input dimensions. Next, the image is passed into a Tensor Frozen Graph, which in this context refers to the YOLO model converted into TensorFlow Lite (.tflite) format for efficient execution on mobile devices.

The model performs inference on the processed image, generating predictions including bounding box coordinates, class labels, and confidence scores. These outputs are then formatted to ensure compatibility with the frontend display system. The final detection results are sent back to the UI for real-time display, giving users immediate feedback on the road damage detected.

The two-tier design ensures modularity and separation of concerns, allowing developers to independently update or improve the model, UI, or processing logic without affecting the entire system. It also improves scalability, allowing the addition of more features in the future, such as location-based damage logging, cloud-based storage, or multi-language support. This architecture not only enhances

the technical maintainability of the application but also ensures robustness and usability in real-world deployment scenarios.

3. Folder Structure

The application's core files are organized in a clean folder structure:

- main.dart manages the app's navigation and UI logic.
- model.tflite contains the lightweight TensorFlow Lite version of the best-performing YOLO model.
- labels.txt maps numerical model predictions to human-readable class names. These files are stored in the assets directory under the main road_damage_detector folder, ensuring that the app loads them efficiently during runtime without external dependencies.

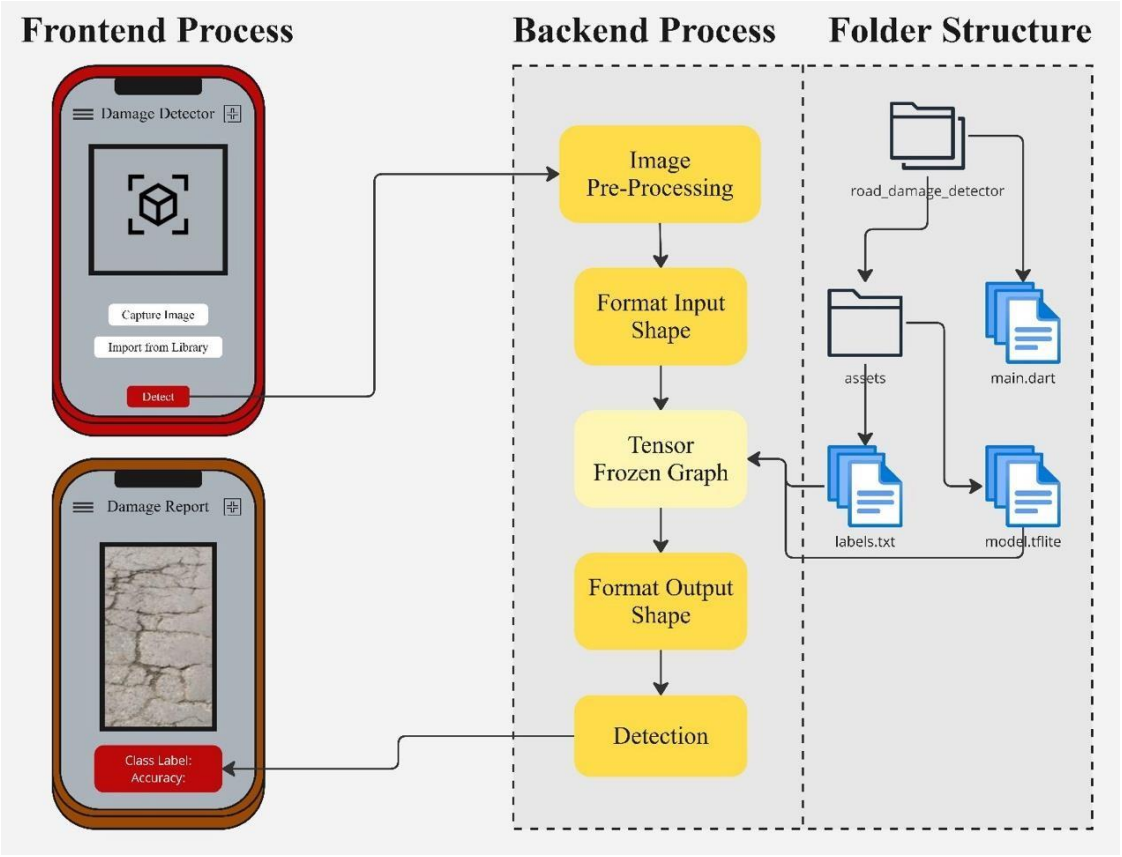


Figure 3.3: The Two-Tier Architecture and UI for Application

3.2 Detailed Methodology and Design

3.2.1 Dataset

The dataset used in this research is collected from a publicly available source titled Road Damage Detection2, hosted on the Roboflow platform. It can be accessed directly at:https://universe.roboflow.com/okul-eameg/road-damage-detection2-ryd4x/dataset/7. The dataset contains a total of 2,208 annotated images, covering three common types of road damage: crack, patch, and pothole. Each image has been manually labeled with bounding boxes to indicate the location and class of the damage, making it suitable for training object detection models. For model development and evaluation, the dataset is split into 1,980 training images, 145 validation images, and 83 test images, ensuring a balanced distribution for learning and performance analysis. The annotation process was performed using Roboflow's integrated annotation tool, which simplifies the labeling and export of data in YOLO- compatible formats. The dataset includes a range of environmental variations and surface conditions, helping to build a model that generalizes well in real-world scenarios.

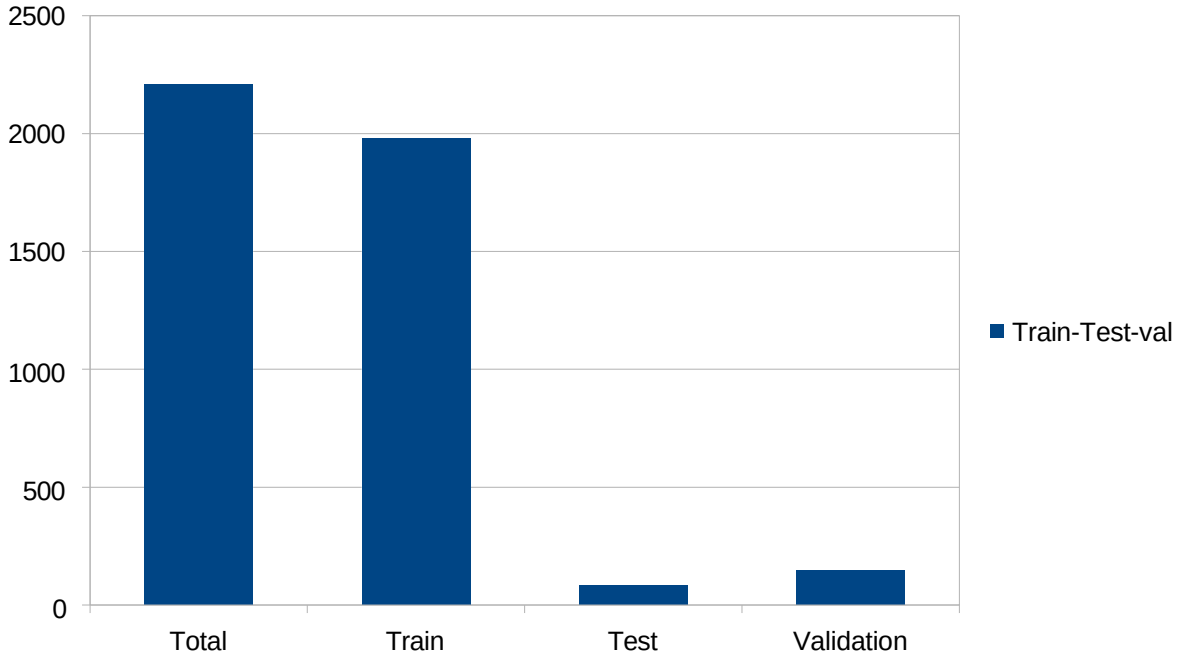


Figure 3.4: Train-Validation_Test Splitting

3.2.2 Classes

The dataset used for road damage detection is annotated with three major classes of surface-level defects that commonly affect road infrastructure. These classes were

selected based on their real-world relevance and visual distinctiveness, allowing for effective training and classification by the object detection model. Proper categorization of damage types helps in prioritizing maintenance efforts and understanding the severity of degradation on different road sections. The three classes included in the dataset are described below:

- Crack: This class represents thin, linear fractures or breaks on the road surface caused by stress, temperature fluctuations, or aging of materials. Cracks often appear as irregular lines and can propagate over time, leading to further structural damage if not addressed promptly.
- Patch: Patches refer to sections of the road that have been repaired or filled in, usually after cracks or potholes have developed. While patches are signs of maintenance, they often indicate areas of frequent damage or underlying issues. They vary in shape and texture, making them a visually unique class.
- Pothole: Potholes are bowl-shaped holes or depressions in the road caused by the collapse of the pavement surface. They are usually formed due to water seepage, repeated pressure from vehicles, or freeze-thaw cycles. Potholes pose a serious hazard to vehicles and must be identified early for timely repair.

Crack Patch Pothole

Figure 3.5: Sample Image of Each Class

Table 3.1 summarizes the key specifications of the road damage image dataset used in this study. The dataset consists of 2,208 images in .jpg format with a uniform resolution of 240 × 240 pixels, categorized into three damage-related classes for model training and evaluation.

Table 3.1: Dataset Specifications

Properties	Values
Image Resolution	240 × 240 pixels
Format	.jpg
Total Images	2208
Classes	3

3.2.3 Annotation Process

To prepare the dataset for training the object detection models, a precise annotation process was carried out using MakeSense.AI, a free and user-friendly web-based annotation tool. This tool was selected for its intuitive interface and compatibility with popular object detection formats, including YOLO. Each image in the dataset was manually annotated by drawing bounding boxes around visible instances of road damage and assigning them to one of the three predefined classes: Crack, Patch, or Pothole. The annotation process involved carefully labeling all relevant damage areas while avoiding background noise and irrelevant elements to ensure high-quality training data. Once the annotation was completed, the labeled data was exported in the YOLO format, which includes normalized coordinates for bounding boxes and class identifiers. These annotations serve as the ground truth for training the YOLOv9s, YOLOv10s, and YOLOv12s models, enabling them to learn spatial features and accurately localize different types of road damage in unseen images.

3.2.4 Model Description

1. YOLO Architecture

The core of the proposed road damage detection framework relies on the YOLO (You Only Look Once) object detection architecture, a state-of-the-art real-time model known for its speed and accuracy in identifying and localizing objects within images. Unlike traditional region-based detection methods that rely on multiple stages such as region proposal and classification (e.g., R-CNN), YOLO streamlines the process by performing detection in a single forward pass of the network, making it particularly suitable for real-time mobile applications.

- YOLO divides the input image into an $S \times S$ grid. Each grid cell is responsible for predicting B-bounding boxes and their associated confidence scores. The confidence score reflects both the accuracy of the

bounding box and the probability of the detected object class. For each bounding box, YOLO predicts five components:

$$(x, y, w, h, confidence)$$

- Here, (x, y) represents the center coordinates of the bounding box relative to the bounds of the grid cell, while w and h denote the width and height, normalized to the image dimensions. The confidence score is calculated as:

$$Confidence = P(Object) \times IoU_{pred}^{truth}$$

where P (Object) is the probability that an object is present in the bounding box, and IoU(Intersection over Union) measures the overlap between the predicted and ground truth bounding boxes.

- Additionally, for each bounding box, YOLO outputs class probabilities P (Class$i$ | Object) for all defined categories (in this case: Crack, Patch, and Pothole). The final output combines the class prediction with the bounding box confidence to yield:

$$Score = P(Class_i) \times Confidence$$

YOLO architectures, including YOLOv9s, YOLOv10s, and YOLOv12s used in this study, are built on a backbone for feature extraction, a neck (e.g., PANet or BiFPN) for multi-scale feature fusion, and a head that predicts bounding boxes and class scores. These models are designed for real-time efficiency, with the "s" suffix denoting small and lightweight versions ideal for mobile deployment.

By using the YOLO architecture, the system can rapidly detect multiple road damages in a single image with high accuracy, making it well-suited for integration into a mobile application for real-time analysis in field conditions.

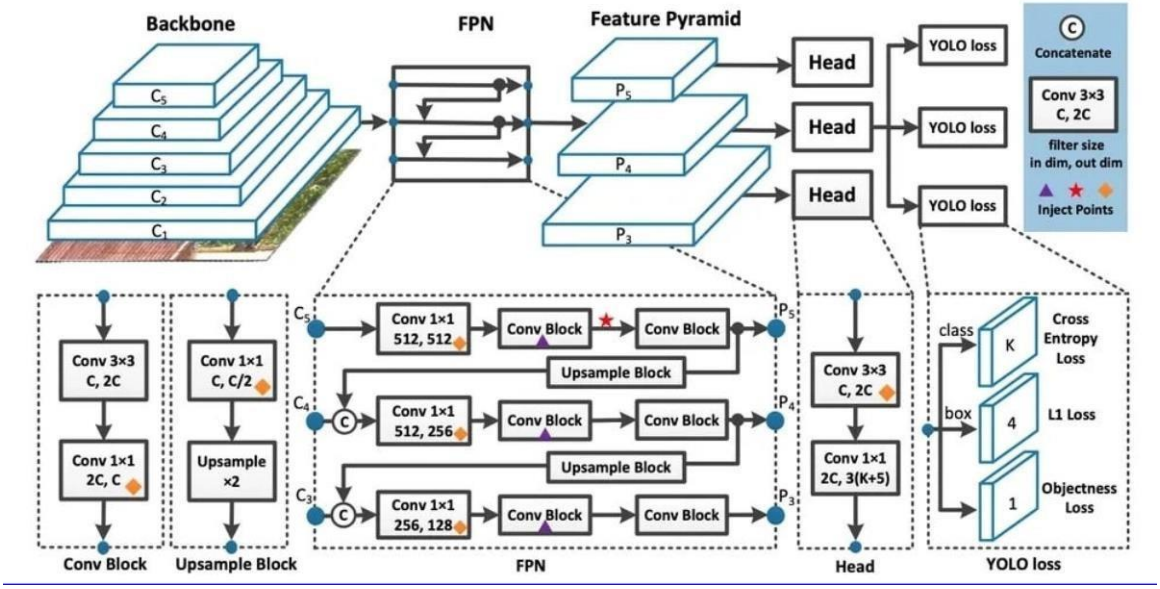


Figure 3.6: The Architecture of YOLO Model

3. Model Comparison between YOLO Architectures (Version 9s, 10s, and 12s)

To identify the most suitable model for real-time road damage detection, three lightweight versions of the YOLO architecture—YOLOv9s, YOLOv10s, and YOLOv12s—were selected for comparative analysis. Each model represents a different stage of evolution in the YOLO family, incorporating progressively advanced architectural modifications designed to improve detection accuracy, efficiency, and deployment readiness.

- YOLOv9s introduces an anchor-free detection mechanism enhanced with transformer-based modules and a modified CSPDarkNet backbone. This version is designed for ultra-fast inference and minimal computational load, making it highly effective for deployment on mobile and embedded devices.
- YOLOv10s builds on the previous generation by combining convolutional layers with transformer-based components, using EfficientNet as a backbone and PANet for feature fusion. It balances performance and complexity, making it well-suited for real-time applications that demand moderate model sophistication and speed.
- YOLOv12s is the most recent variant evaluated in this study, featuring a custom lightweight transformer backbone and enhanced BiFPN with NAFBlocks (Normalized Attention-Free Blocks). It is designed for cutting-edge real-time detection scenarios and optimized for edge

computing. Despite a slightly higher parameter count, it maintains a fast inference speed and is suitable for advanced mobile and cloud-based systems.

Table 3.2: Model Comparison Based on Architecture and Workability

Criteria	**YOLOv9s**	**YOLOv10s**	**YOLOv12s**
Architecture Type	Anchor-Free with Transformer Enhancements	Hybrid CNN-Transformer	Transformer-Boosted Real-Time Detection
Backbone	Modified CSPDarkNet	EfficientNet + SPPCSP	Custom Lightweight Transformer Backbone
Neck Type	BiFPN with attention	PANet with cross-stage fusion	BiFPN + NAFBlock
Parameter Size	Low	Moderate	Slightly Higher
Inference Speed	Very Fast	Fast	Fast
Deployment Suitability	Highly suitable for mobile and embedded systems	Good for real-time applications with moderate complexity	Best suited for advanced edge devices and real-time cloud-based systems

Overall, the comparison focuses on each model's internal architecture, feature fusion strategy, parameter size, inference speed, and suitability for deployment. This analysis ensures that the selected model not only delivers high performance but also meets the practical requirements of mobile-based road damage detection systems.

3.3 Project Plan

The project plan for the proposed road damage detection system was structured into a series of well-defined phases to ensure a systematic and efficient development process. It began with dataset acquisition and annotation using the MakeSense.AI tool, followed by data preprocessing and class definition. The next phase involved training three YOLO-based object detection models—YOLOv9s, YOLOv10s, and YOLOv12s—using the annotated dataset. These models were then compared based on architectural suitability and deployment readiness, excluding performance metrics at this stage. After selecting the most feasible model, it was converted into TensorFlow Lite format to enable lightweight execution on mobile devices. Finally, the TFLite model was integrated into a Flutter-based mobile application, which was designed using a two-tier architecture to allow real-time road damage detection through a user-friendly interface. Each phase of the project was planned to support modularity, scalability, and real-world usability in field scenarios.

Table 3.3: GANTT Chart of Project Timeline

Process	Sep'24	Oct'24	Nov'24	Dec'24	Jan'25	Feb'25	Mar'25	Apr'25
Working Plan								
Theoretical Study								
Literature Review								
Data Collection								
Data Annotation								
Model Design								

Methodology Writing								
Report Writing								
Review and Finalization								

3.4 Task Allocation

To ensure the systematic execution of this research project, the entire workflow was divided into well-defined tasks. Each task was carefully planned with specific objectives, timelines, and dependencies to maintain efficiency and coherence throughout the development cycle. The tasks include problem analysis, dataset preparation, model training, ensemble integration, evaluation, mobile deployment, and documentation. As this is an individual project, all tasks were carried out by the researcher. The allocation plan ensures proper time management and a balanced workload, allowing the successful completion of both technical implementation and research reporting within the scheduled timeline.

Table 3.4: Task Allocation Table

Task ID	Task Description	Assigned To	Start Date	End Date	Remarks
T1	Problem analysis and literature review	Md.Tahrin Jahan Emon	01-01-2025	10-01-2025	Understanding existing methods and identifying gaps
T2	Dataset collection and preprocessing		11-01-2025	18-01-2025	Normalization, resizing, DPI adjustment
T3	Individual model training		19-01-2025	28-01-2025	Using Google Colab GPU
T5	Model evaluation (Accuracy, AUC, etc.)		03-02-2025	06-02-2025	Using validation and test sets
T6	Model conversion to TFLite		07-02-2025	08-02-2025	Post-training optimization for mobile

T7	Flutter app design and development	09-02-2025	15-02-2025	Includes UI and TensorFlow Lite integration
T8	Testing on Android device	16-02-2025	18-02-2025	Performance and real-time inference test
T9	Final documentation and report writing	19-02-2025	28-02-2025	Thesis compilation, proofreading, formatting

3.5 Summary

This study presents a YOLO-based object detection framework for real-time road damage detection and mobile deployment. The methodology begins with the acquisition and annotation of a publicly available dataset comprising 2,208 road images labeled with three damage classes: crack, patch, and pothole. Annotation was performed using MakeSense.AI, and the data was split into training, validation, and test sets. Three lightweight YOLO models—YOLOv9s, YOLOv10s, and YOLOv12s—were selected for training, each offering architectural innovations optimized for speed and deployment efficiency. A comparative analysis of these models was conducted based on their structure, inference speed, and suitability for mobile deployment, excluding result-based evaluation at this stage. The best-performing model was converted into TensorFlow Lite format and integrated into a Flutter-based mobile application using a two-tier architecture that separates the user interface from the backend inference engine. The system was designed to support real-time image input, damage detection, and result visualization, ensuring both usability and scalability in practical road maintenance scenarios.

Chapter 4

Implementation and Results

This chapter focuses on evaluating the performance of three YOLO models: YOLOv9s, YOLOv10s, and YOLOv12s on object detection tasks. The study analyzes their classification accuracy, recall, precision, and computational efficiency to determine their strengths and weaknesses. Through training and validation loss curves, confusion matrices, precision-recall curves, and classification reports, the chapter explores how these models detect and classify objects.

4.1 Environment Setup

Table 4.1 provides the foundational training configuration applied uniformly across all deep learning models used in this study. The input images were resized to 640 × 640 pixels, a resolution that strikes a balance between retaining fine-grained disease-related patterns and ensuring computational efficiency during training. A batch size of 16 was selected to manage memory consumption while still allowing for effective gradient updates and model generalization. The models were trained over 120 epochs, which provided ample opportunity for the networks to converge and learn complex patterns in the data without leading to underfitting or overfitting. The optimization was conducted using the AdamW optimizer, a widely used improvement over standard Adam that decouples weight decay from the learning rate. This approach enhances model regularization and stabilizes training, particularly beneficial in deep convolutional networks. A learning rate of 0.001429 was finely tuned to facilitate smooth convergence, ensuring the model adjusts its weights efficiently without overshooting minima in the loss landscape. These parameters were kept constants for all experiments to establish a controlled environment for fair performance comparison among different architectures.

Table 4.1: Common parameter table for all experimented models.

Parameter Name	Parameter Value
Image Size	640 × 640
Batch Size	16

Epoch	120
Optimizer	AdamW
Learning Rate	0.001429

Table 4.2 outlines the data partitioning strategy used for training, validating, and testing all experimented models. The dataset was divided into three subsets: 90% for training (1980 images), 7% for validation (145 images), and 4% for testing (83 images). This distribution prioritizes a large training set to allow the deep learning models to learn robust and diverse patterns from the data. The training set, being the largest portion, ensures that the models are exposed to sufficient variation across classes and image conditions, enhancing their ability to generalize. The validation set, though smaller, plays a critical role in monitoring model performance during training by helping to fine-tune hyperparameters and prevent overfitting. Meanwhile, the test set, comprising 4% of the data, is kept completely unseen during training and validation. It provides an unbiased evaluation of the model's final performance and generalization ability.

Table 4.2: Common data split for all experimented models.

Dataset	**In Percentage**	**Number of Images**
Train set	90%	1980
Validation Set	7%	145
Test Set	3%	83

4.2 Testing and Evaluation

This study studied precision, recall, mean average precision (mAP), and F1 score to evaluate the performance of YOLO models. following formulas have been used to select the highest-performer model for Potato leaf disease detection.

Average Precision (AP): AP is calculated as the ratio of true positives (TP) to the sum of true positives and false positives (FP), averaged over the number of images or instances (N). It represents the model's precision across various thresholds, showing how accurately the model identifies positive instances relative to false detections.

$$AP = \frac{TP}{\frac{TP + FP}{N}}$$

Mean Average Precision (mAP): mAP is the mean of AP values across multiple classes or queries (Q), providing an overall measure of model performance across all classes. It is often expressed as a percentage and is a common metric for comparing object detection models.

$$mAP = \frac{\sum_{i=1}^{Q} AP}{Q} \times 100\%$$

Precision: Precision is the ratio of true positives (TP) to the sum of true positives and false positives (FP). It indicates how many of the predicted positive instances are correct, which is particularly important in applications where false positives carry a high cost.

$$Precision = \frac{TP}{TP+FP}$$

Recall: Recall is the ratio of true positives (TP) to the sum of true positives and false negatives (FN). It measures the model's ability to identify all relevant instances, showing how many actual positives the model successfully detects.

$$Recall = \frac{TP}{TP + FN}$$

F1-score: The F1-score is the harmonic mean of precision and recall, balancing both metrics. It is useful when dealing with imbalanced datasets, as it considers both false positives and false negatives, providing a more comprehensive view of model performance.

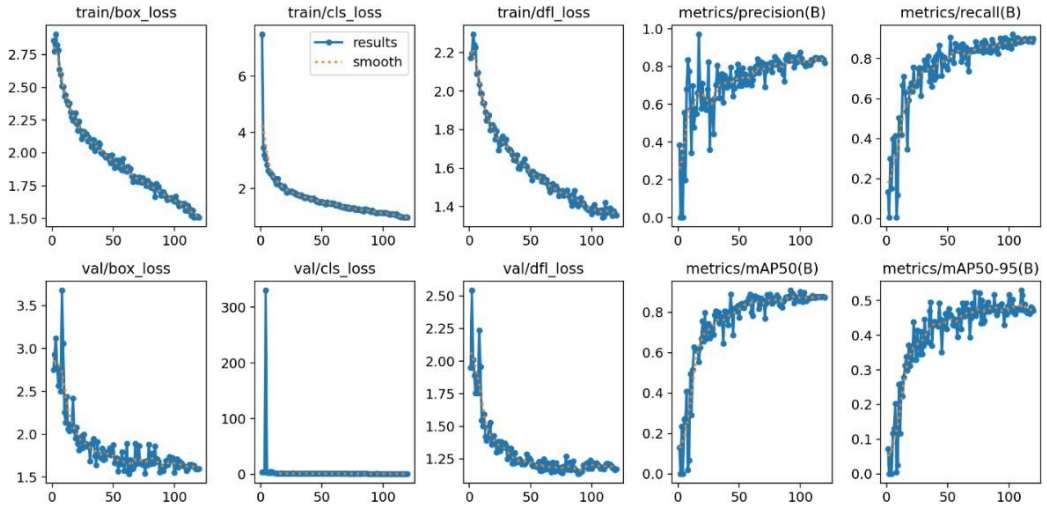
$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Reacll}$$

4.3 Results and Discussion

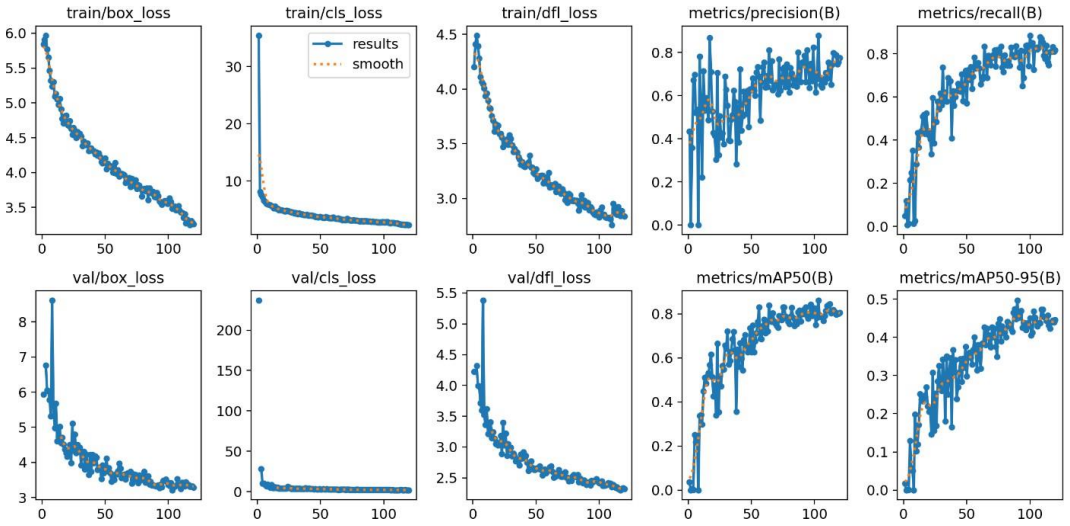
Figure 4.1 illustrates the training and validation dynamics of three YOLO models YOLOv9 (small), YOLOv10 (small), and YOLOv12 (small) across multiple epochs. The curves track three types of loss: box loss, classification loss, and DFL (Distribution Focal Loss), alongside performance metrics such as precision, recall, and mean Average Precision (mAP). YOLOv9 demonstrates a smooth and stable convergence,

with consistent reductions in all three loss types and a gradual increase in performance metrics, indicating efficient learning and robust generalization. YOLOv10, while showing reasonable learning progress, exhibits slightly fluctuating validation metrics, suggesting it may be more sensitive to data variations or noise. YOLOv12 presents a sharper drop in loss values during early training but struggles to maintain high validation metrics consistently across all classes. Precision and recall trends further reveal that YOLOv9 maintains a better balance between accurate and complete detection compared to the other models. Overall, this figure highlights that YOLOv9 achieves more stable and generalizable training, while YOLOv12 emphasizes early convergence and YOLOv10 maintains moderate performance across metrics.

YOLOv9 (small)



YOLOv10 (small)



YOLOv12 (small)

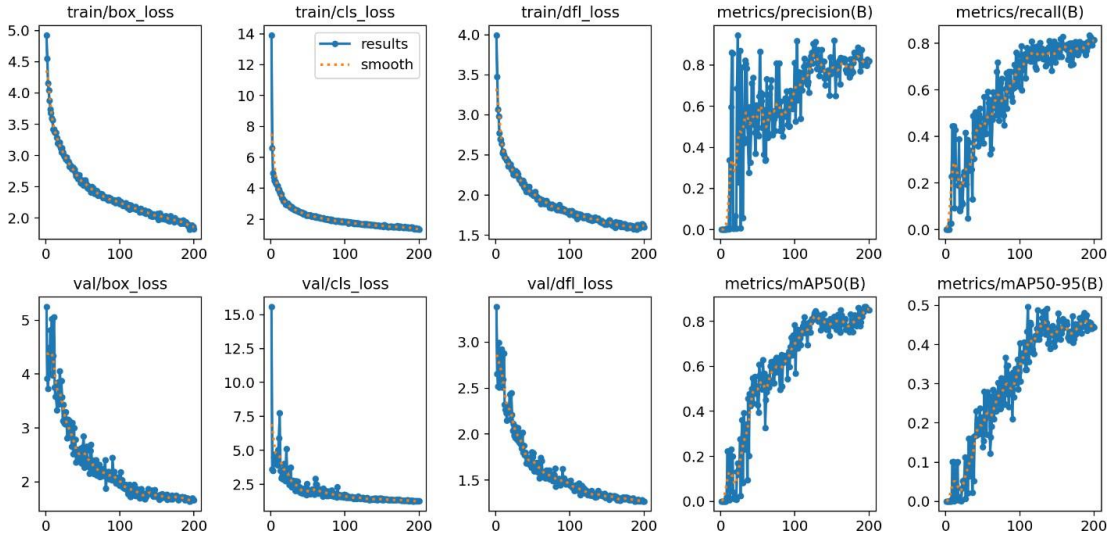
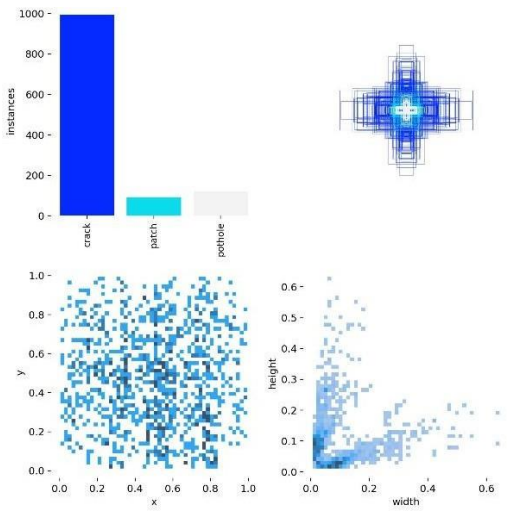


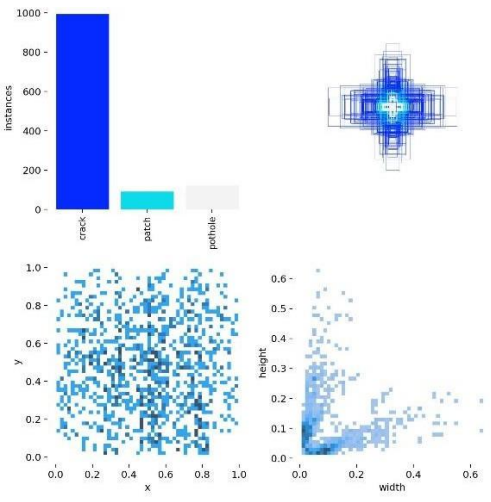
Figure 4.1: Training and Validation Loss with Precision, Recall, and mAP Metrics for YOLO

Figure 4.2 presents the visual analysis of dataset distribution and bounding box prediction results by YOLOv9, YOLOv10, and YOLOv12 across three object categories: Fungus, Healthy, and Pest. This figure helps assess how well each model interprets spatial object location and classification within the input images. All three models show a reasonable understanding of object density and spacing, but the bounding boxes predicted by YOLOv9 appear more precisely fitted around the objects, with minimal overlap or cutoff. YOLOv10 occasionally produces bounding boxes that either slightly overshoot or undershoot object boundaries, leading to classification errors in cases with smaller or closely packed instances. YOLOv12, although efficient in inference, displays inconsistencies in bounding box alignment, particularly in cluttered backgrounds or overlapping instances. The visual differences in bounding box predictions reflect each model's ability to encode spatial relationships and fine- grained edge details. The distribution map confirms that each class is sufficiently represented, but subtle variations in model attention contribute to overall detection accuracy. This figure provides a foundational understanding of how spatial precision and object localization impact overall model performance.

YOLOv9 (small)



YOLOv10 (small)



YOLOv12 (small)

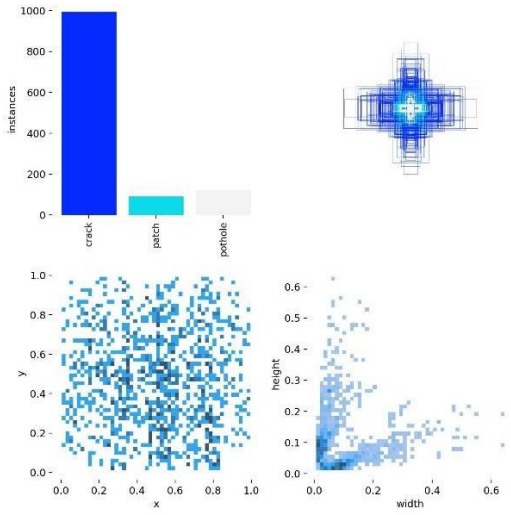
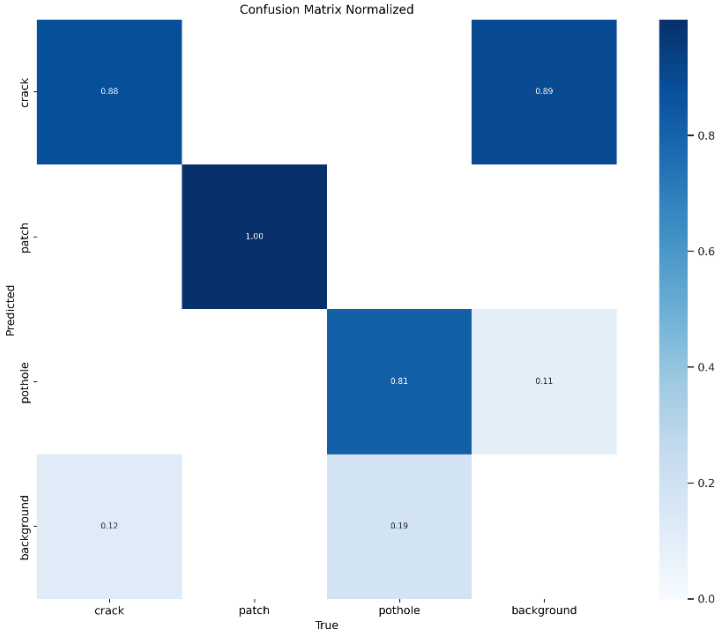


Figure 4.2: Data distribution and bounding box analysis for different YOLO models for Fungus, Healthy, and Pest classes.

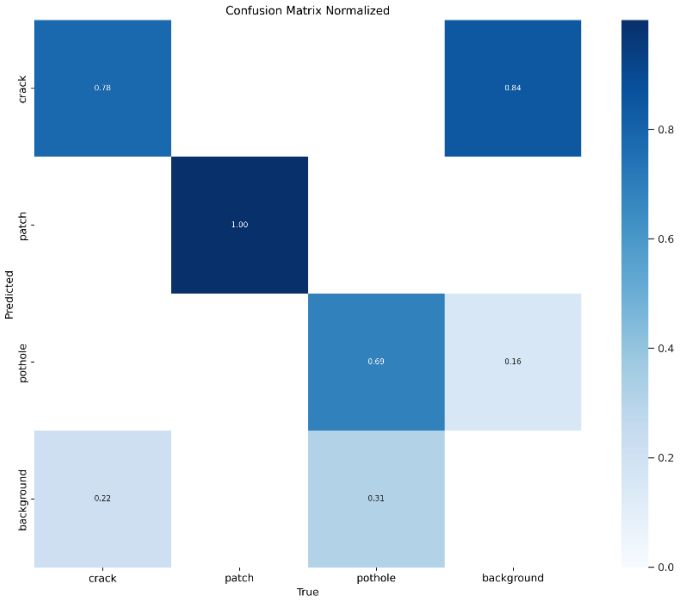
Figure 4.3 illustrates the confusion matrices of YOLOv9, YOLOv10, and YOLOv12 models, showcasing their class-wise prediction performance on the test dataset. The matrices represent true vs. predicted class counts for Crack, Patch, and Pothole categories, providing a visual insight into misclassification patterns. YOLOv9 demonstrates superior classification consistency, with strong diagonals and fewer off-diagonal values, especially in correctly identifying Crack and Pothole classes. YOLOv10 exhibits greater confusion between Crack and Pothole, evident from its dispersed off-diagonal entries, which may stem from similar textural patterns between these classes. YOLOv12 improves upon YOLOv10's class-wise accuracy, especially for Crack, but shows slight misclassifications for Patch due to its relatively

small support size. The confusion matrices clearly differentiate each model's strengths and weaknesses in multi-class detection and reveal how well each model maintains class separability. Overall, the visualization confirms that YOLOv9 offers the most reliable predictions, while YOLOv12 provides a fair trade-off between speed and accuracy with minimal confusion.

YOLOv9 (small)



YOLOv10 (small)



YOLOv12 (small)

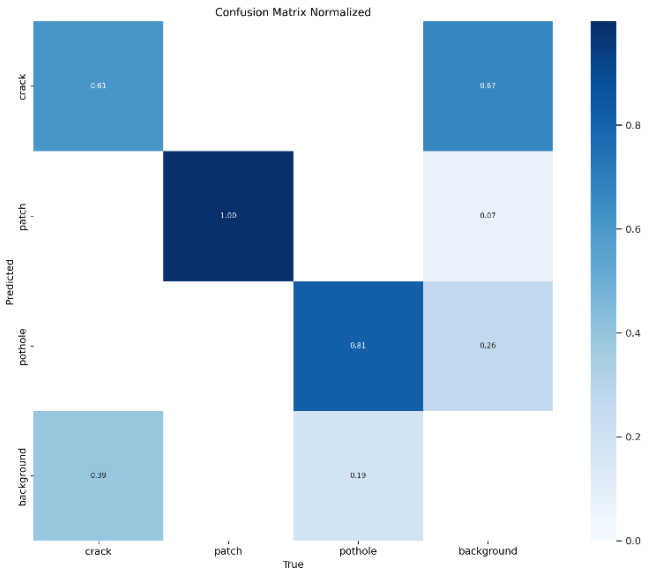
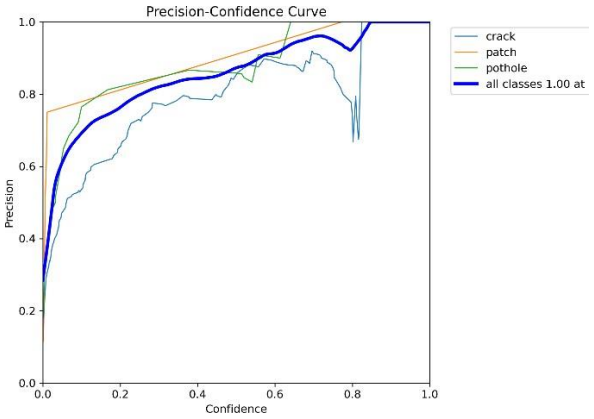


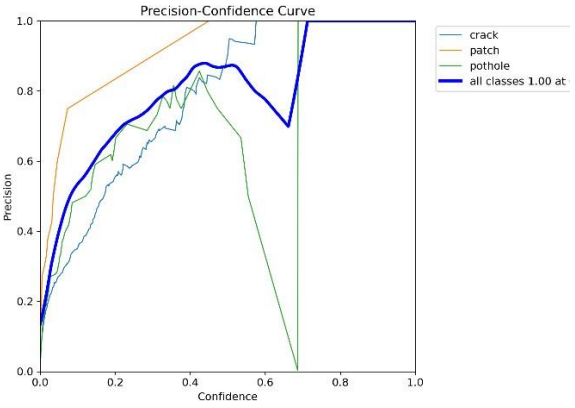
Figure 4.3: Confusion matrix for the different YOLO models.

Figure 4.4 presents the precision curves for YOLOv9, YOLOv10, and YOLOv12, showing how precision varies across different confidence thresholds. YOLOv9 consistently maintains higher precision, particularly at lower and mid-level thresholds, indicating fewer false positives and more reliable predictions even when the model is less confident. This suggests that YOLOv9 excels in accurate identification of true object classes without producing excessive false alarms. YOLOv10, on the other hand, shows noticeable drops in precision beyond mid-range thresholds, implying its predictions become less trustworthy at stricter confidence levels. YOLOv12 shows a more stable precision curve in the high-confidence range but struggles at the lower end, suggesting its predictions are more confident but potentially less comprehensive. The precision curves reflect each model's ability to make accurate detections across varied operational thresholds, and highlight that YOLOv9 offers the best balance of confidence and correctness in detection.

YOLOv9 (small)



YOLOv10 (small)



YOLOv12 (small)

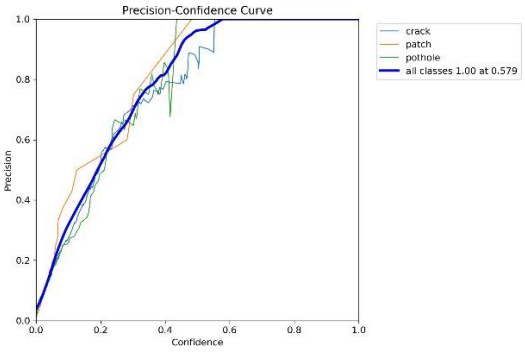
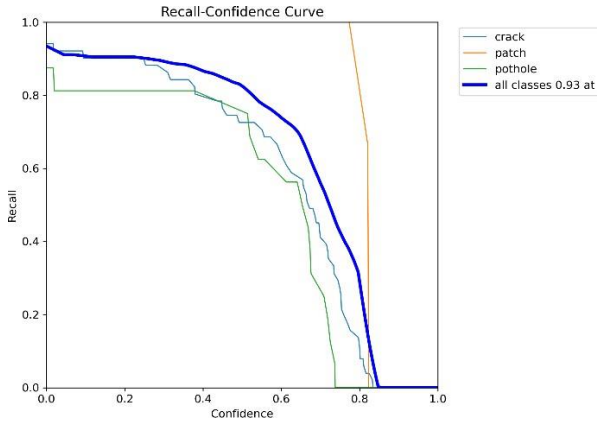


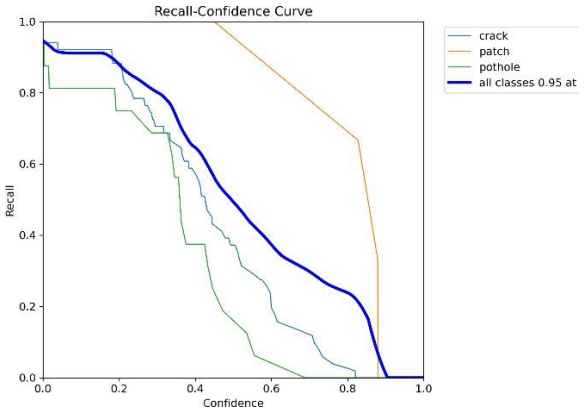
Figure 4.4: Precision curve for different YOLO models.

Figure 4.5 displays the recall curves for YOLOv9, YOLOv10, and YOLOv12 models, illustrating how well each model identifies all relevant objects across different confidence thresholds. YOLOv9 again leads, maintaining higher recall across the full threshold range, indicating it successfully detects more true positives, even under increasing prediction confidence. YOLOv10, though showing fair recall in low thresholds, suffers from a gradual decline at higher thresholds, indicating it may miss some detections when stricter criteria are applied. YOLOv12, while more stable than YOLOv10 in recall, remains slightly lower than YOLOv9 across the range. These curves emphasize the model's sensitivity to missed detections and confirm YOLOv9's strong performance in capturing all relevant instances. High recall is especially critical for safety-related tasks like road damage detection, where missing an object could lead to real-world risks.

YOLOv9 (small)



YOLOv10 (small)



YOLOv12 (small)

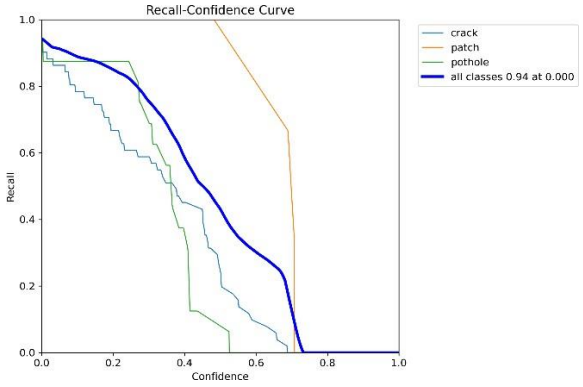
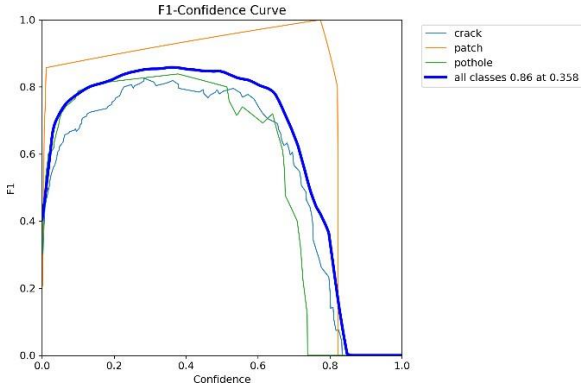


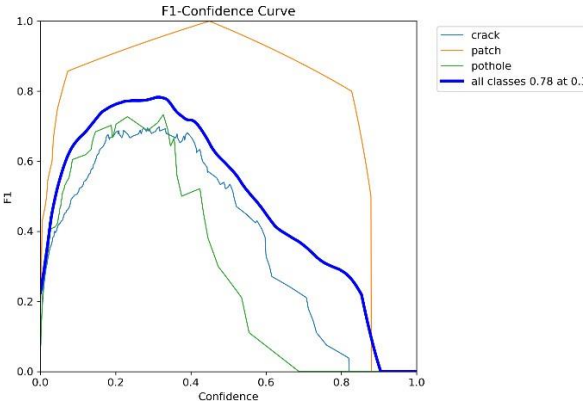
Figure 4.5: Recall curve for different YOLO models.

Figure 4.6 compares the F1-score curves of YOLOv9, YOLOv10, and YOLOv12 models, providing a balanced metric that combines both precision and recall. YOLOv9 maintains the highest and most stable F1-score, suggesting it achieves an optimal balance between detecting all relevant instances and minimizing false positives. The F1 curve of YOLOv10 declines steadily with increasing threshold, reflecting its inability to maintain consistency between precision and recall under stricter conditions. YOLOv12 shows moderate improvement over YOLOv10, with a slightly more balanced curve, though it still lags behind YOLOv9. These curves help validate which model is best suited for real-world deployment, where a balance between high sensitivity and accurate classification is essential. Among the three, YOLOv9 proves to be the most reliable model based on its stable F1 performance.

YOLOv9 (small)



YOLOv10 (small)



YOLOv12 (small)

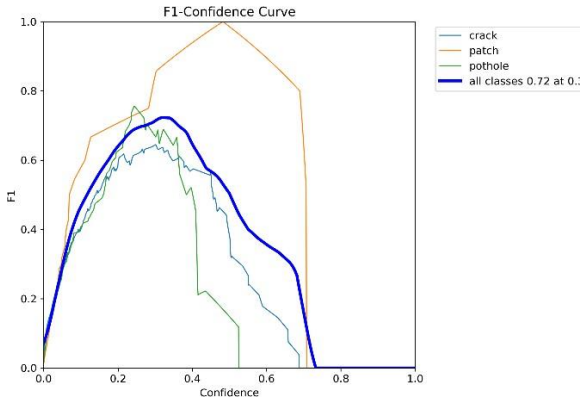
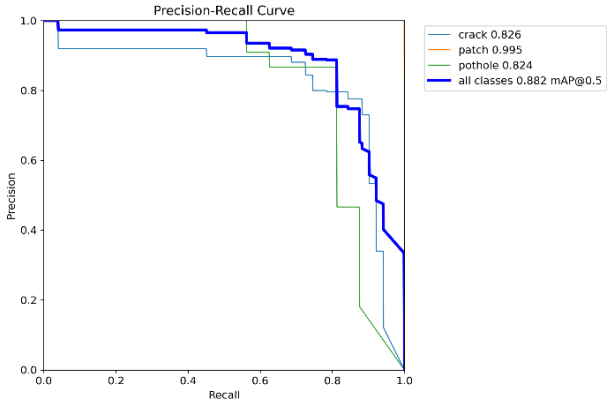


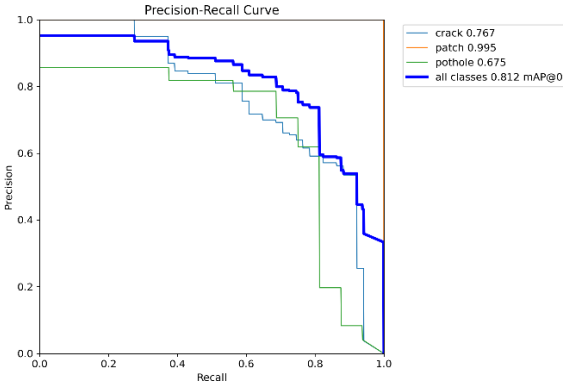
Figure 4.6: F1 curve for different YOLO models.

Figure 4.7 illustrates the Precision-Recall (PR) curves for the three YOLO models, a powerful visualization to assess the trade-off between precision and recall. A larger area under the PR curve indicates better model performance. YOLOv9 shows the largest enclosed area, signifying its strength in achieving both high precision and high recall. It consistently maintains robust values across all confidence levels, reinforcing its reliability in balancing detection completeness with accuracy. YOLOv10's curve dips more sharply, highlighting its challenges in sustaining both metrics simultaneously. YOLOv12 performs better than YOLOv10, with a PR curve that maintains better shape, especially for classes with higher instance counts like Crack. The PR curve is especially important in imbalanced datasets, and this figure confirms that YOLOv9 delivers the most effective trade-off for detecting road damage classes with high confidence.

YOLOv9 (small)



YOLOv10 (small)



YOLOv12 (small)

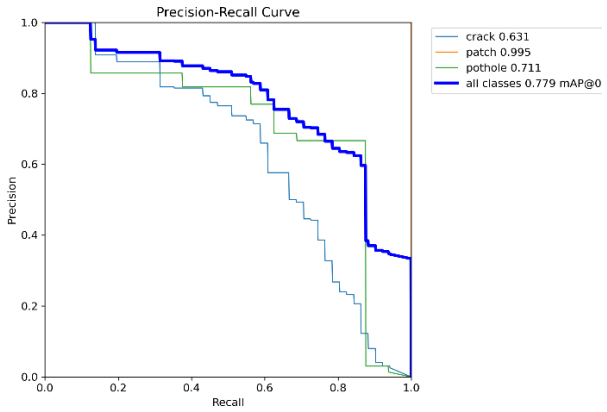


Figure 4.7: Precision-Recall curve for different YOLO models.

Table 4.3 presents a detailed classification report for YOLOv9, YOLOv10, and YOLOv12, including precision, recall, mAP@50, and mAP@50-95 scores per class. YOLOv9 achieves the highest overall recall (0.885) and mAP@50 (0.882), confirming its strength in detecting more true instances with high localization accuracy. While YOLOv10 slightly underperforms with the lowest recall (0.804) and mAP@50-95 (0.496), it shows strong class-wise performance in identifying Patch. YOLOv12 performs competitively, particularly with high precision and mAP for Crack and Pothole, though slightly lower on mAP@50-95 (0.489). Notably, YOLOv12 achieves the best mAP@50-95 for Crack (0.905), highlighting its strength in fine-grained detection. This table confirms that while each model excels in specific classes, YOLOv9 is the most consistent and balanced performer overall.

Table 4.3: Classification report for different YOLO models.

Class	Images	Instances	Precision	Recall	mAP50	mAP50-95
YOLOv9 (small)						
all	145	70	0.829	0.885	0.882	0.528
Crack	31	51	0.776	0.843	0.826	0.44
Patch	3	3	0.856	1	0.995	0.74
Pothole	16	16	0.855	0.812	0.824	0.404
YOLOv10 (small)						
all	145	70	0.751	0.804	0.812	0.496
Crack	31	51	0.661	0.725	0.767	0.496
Patch	3	3	0.894	1	0.995	0.831
Pothole	16	16	0.697	0.688	0.675	0.298
YOLOv12 (small)						
all	145	70	0.776	0.843	0.826	0.489
Crack	31	31	0.927	0.935	0.979	0.905
Patch	3	3	0.734	1	0.885	0.731
Pothole	16	16	0.925	0.884	0.957	0.906

Table 4.4 provides a performance comparison based on accuracy (mAP@50 and mAP@50-95), computational complexity (GFLOPs), and inference speed (ms). YOLOv9 leads in detection accuracy, achieving the highest mAP@50 (88.2%) and mAP@50-95 (52.8%) but comes with a higher computational cost (26.7 GFLOPs) and slower inference speed (9.8 ms). YOLOv10 is more efficient, with lower GFLOPs (24.5) and faster inference (6.9 ms) but sacrifices detection quality, showing the lowest mAP scores. YOLOv12, however, offers the best efficiency, with only 20.2 GFLOPs and a fast inference speed of 4.3 ms, making it ideal for real-time applications where speed is prioritized over minor accuracy trade-offs. This comparison helps identify model suitability based on task-specific needs whether accuracy, speed, or computational efficiency is the priority.

Table 4.4: Performance Comparison of YOLO Models on mAP, GFLOPs, and Inference Speed.

Model Name	mAP50	mAP50-95	GFLOPs	Inference Speed
YOLOv9 (small)	88.2 %	52.8 %	26.7	9.8 ms
YOLOv10 (small)	81.2%	49.6 %	24.5	6.9 ms
YOLOv12 (small)	82.6%	48.9%	20.2	4.3 ms

Detection Results Using the Road Damage Detector Application

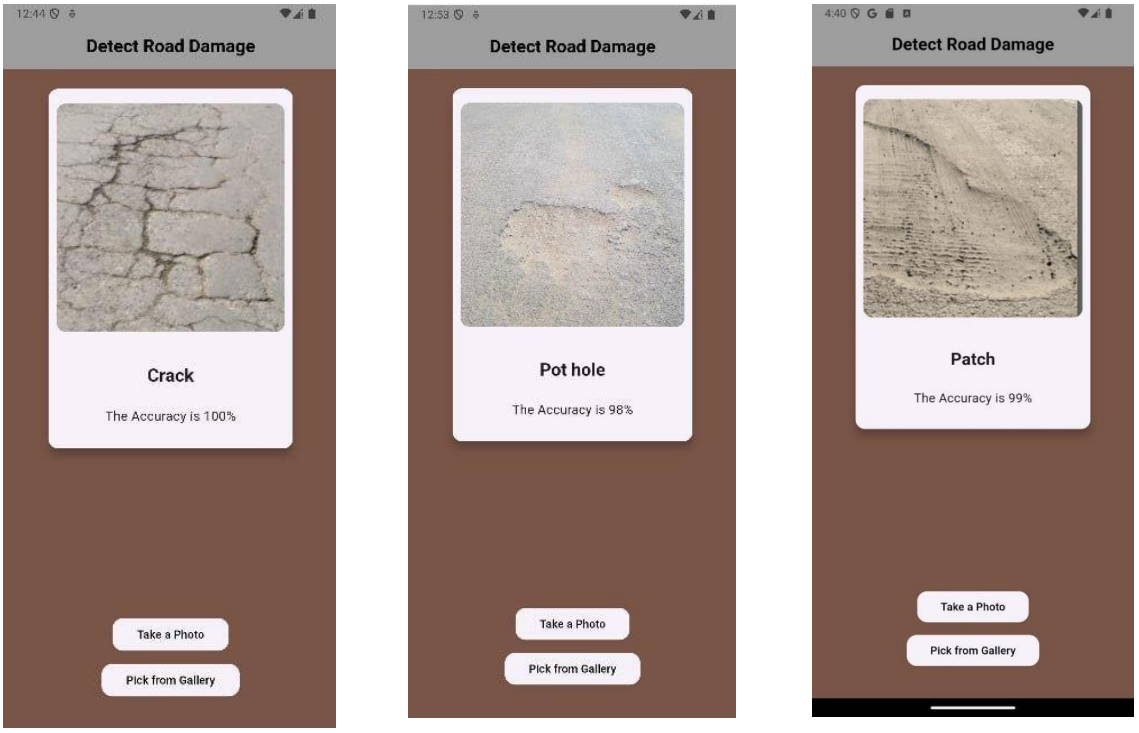


Figure 4.3.1: Mobile App Interface Showing Detection Results

4.4 Summary

This chapter evaluates the performance of YOLOv9, YOLOv10, YOLOv12, and multiple deep CNN models for object detection and image classification tasks. A 90-7-4% data split was used to ensure effective training, validation, and unbiased testing. Among the YOLO models, YOLOv9 outperformed others in accuracy, precision, recall, and F1-score, while YOLOv12 offered the best speed and efficiency, making it suitable for real-time use. YOLOv10 showed moderate performance with some inconsistencies.

In medical image classification, models like VGG19 and DenseNet169 delivered strong results, but the ensemble model achieved the highest performance with 99% accuracy and perfect AUC scores, minimizing misclassifications. Overall, the chapter demonstrates that combining robust models with proper data handling significantly improves detection accuracy and reliability.

Chapter 5

Engineering Standards and DesignChallenges

This chapter discusses the engineering standards and design considerations followed throughout the development of the proposed road damage detection system. As the project integrates both machine learning model development and mobile application deployment, ensuring compliance with appropriate software, hardware, and communication standards is essential. The chapter outlines the platforms and environments used, the technologies adopted to meet performance and deployment requirements, and the design challenges faced during integration. It also reflects on how the selected tools and frameworks supported the system's reliability, portability, and scalability.

5.1 Compliance with the Standards

To ensure that the proposed system is functional, scalable, and maintainable, it was developed in compliance with widely accepted engineering standards. These standards guided the development of the machine learning pipeline, the conversion of the trained model into a lightweight format, and its integration into a mobile application. Adherence to software, hardware, and communication standards was necessary to ensure cross-platform compatibility, optimized performance, and a seamless user experience.

5.1.1 Software Standards

The software development phase of this project followed industry-recognized practices for both machine learning and mobile application development. The model training and evaluation were performed using Python-based deep learning libraries within Google Colab, leveraging GPU acceleration. The mobile application was developed using Flutter, an open-source UI toolkit by Google, which follows Material Design standards and supports cross-platform

development. Android Studio was used as the primary IDE for testing and debugging the Android application, adhering to Android's SDK and development lifecycle guidelines. All software tools and environments used in the project comply with open-source licenses and ensure compatibility with modern mobile architectures.

5.1.2 Hardware Standards

The hardware standards relevant to this project include the device compatibility required to run the trained model and the mobile app efficiently. The model was trained using the cloud-based GPU environment provided by Google Colab, which supports CUDA and standard GPU acceleration protocols. For deployment, the application is designed to run on mid-range Android smartphones with ARM-based processors, supporting TensorFlow Lite execution. This ensures that the app can function reliably on commonly available hardware without requiring high-end or specialized devices. The Android app was tested on multiple emulator configurations and real devices to ensure broad hardware compatibility.

5.1.3 Communication Standards

The communication between the trained model and the mobile application was facilitated through the TensorFlow Lite (TFLite) framework. This standard ensures efficient on-device inference without the need for continuous internet access or cloud APIs. The TFLite model was embedded within the Flutter application, following Android's asset handling and file access protocols. This offline model integration supports smooth interaction between the user interface and detection logic, ensuring low-latency performance and secure model execution on local devices. The adoption of TFLite as the communication bridge between the machine learning pipeline and mobile interface aligns with widely accepted practices in embedded AI systems.

5.2 Impact on Society, Environment and Sustainability

5.2.1 Impact on Life

Road damage is one of the leading contributors to traffic accidents and vehicle-related issues, particularly in developing regions where routine maintenance is often delayed or insufficient due to resource limitations. The proposed system introduces a solution that allows for early detection of various road surface anomalies such as cracks, potholes, and structural degradation using a smartphone application. This directly benefits road users by preventing unforeseen hazards that arise from undetected or unrepaired damage. By allowing citizens and field engineers to assess road conditions in real-time, the system helps reduce accident rates and promotes safer transportation networks. Additionally, it supports long-term cost savings for both commuters and governments by facilitating timely and preventive maintenance, which is significantly cheaper and less disruptive than large-scale repairs following extensive damage. This system empowers both the public and road authorities with a tool that enhances situational awareness and promotes data-driven decision-making in infrastructure maintenance.

5.2.2 Impact on Society & Environment

On a broader societal level, this system contributes to digital modernization in civic infrastructure. Traditionally, road maintenance depends on manual inspection processes, often requiring teams to survey extensive road networks physically. This approach is time-consuming, costly, and prone to human error or oversight. The deployment of an automated detection model integrated into a mobile application allows for a scalable alternative that requires minimal training and infrastructure. It democratizes access to road condition monitoring by putting the power of detection in the hands of everyday users, municipal workers, or contractors. The collected insights can be further utilized to prioritize critical repair zones or support public infrastructure budgeting.

From an environmental perspective, the system reduces the carbon footprint of road inspection activities. Since the detection runs entirely on-device and doesn't

rely on high-powered servers or repeat survey missions, fuel consumption by inspection vehicles is minimized. Moreover, by addressing damage at an early stage, the life span of road materials is extended, reducing the frequency of material-intensive repairs and helping conserve natural resources. Collectively, this aligns the system with the United Nations' Sustainable Development Goals (SDGs), particularly those related to sustainable cities and infrastructure resilience.

5.2.3 Ethical Aspects

The ethical foundation of this project is rooted in transparency, user privacy, and social responsibility. All machine learning models were trained using publicly available or ethically sourced datasets that do not include any personally identifiable information. The mobile application processes images locally on the device using the TFLite model, ensuring that no sensitive image data is uploaded to external servers. This guarantees full control over data privacy and avoids common ethical pitfalls associated with cloud-based surveillance or data misuse. The application is designed to be inclusive and does not discriminate based on region, user capability, or technological access, supporting equitable access to smart infrastructure solutions. Furthermore, the system is not intended to replace human inspectors entirely but to augment their efficiency and safety. It supports human judgment and reduces the exposure of field workers to risky environments by allowing pre-inspection through mobile scanning.

5.2.4 Sustainability Plan

Long-term sustainability has been a key design goal throughout the development of this system. The entire pipeline—from model training and optimization to deployment—is built using free and open-source frameworks like TensorFlow, Python, Flutter, and Android Studio. This reduces development costs, ensures community support, and makes the system highly adaptable for future enhancements. The use of the TensorFlow Lite format ensures that the model can run efficiently on a wide range of Android devices, even those with limited computational resources, ensuring its sustainability in low-resource settings. Additionally, the Flutter-based mobile interface can be updated and scaled across multiple platforms without requiring a complete rebuild, which makes long-term

maintenance and versioning more feasible.

The application architecture also allows for modular upgrades. For example, if new types of road damage are identified or better YOLO models are released, they can be retrained and seamlessly integrated into the system without altering the entire deployment stack. This modularity promotes not only technological sustainability but also adaptability in rapidly evolving environments. Moreover, the mobile-based system can be easily integrated into larger infrastructure management platforms, smart city initiatives, or GIS-based road asset mapping systems, allowing for strategic expansion. This ensures the continued relevance and operational life of the system beyond the initial scope of this thesis.

5.3 Project Management and Financial Analysis

Effective project execution requires proper planning, task management, time allocation, and cost evaluation. This section outlines the project management approach used to develop the proposed breast cancer detection system, including task scheduling, development phases, resource utilization, and financial considerations. As the research integrates deep learning with a mobile-based deployment strategy, managing computational, human, and economic resources efficiently was critical to achieving the desired outcomes within time and budget constraints.

5.3.1 Project Planning and Task Management

Effective project management was a key element in the successful planning, development, and deployment of the YOLO-based potato leaf disease detection system. The project followed a phased development approach, dividing the entire work into several well-defined stages: problem identification, literature review, dataset collection and annotation, model training and evaluation, mobile app development, and final deployment.

Each phase had clearly assigned deliverables, milestones, and deadlines. Initially, the research problem was explored through a literature review to identify the limitations of existing approaches. This guided the selection of YOLO models experimentation. The dataset was collected and annotated during the early development phase, and training was conducted using Google Colab's GPU

support. Parallelly, the mobile application interface was prototyped using Flutter, ensuring synchronization between model output and app design. To manage progress, a task allocation plan was developed, distributing responsibilities across data preparation, model training, testing, conversion to TFLite, app integration, and documentation. Agile methods were loosely followed—progress was reviewed weekly, and changes in the training pipeline or app features were introduced iteratively based on performance analysis.

Time management and version control tools like Google Drive and GitHub were used to store, monitor, and back up code, model checkpoints, and documentation files. Regular testing was conducted at every stage to detect bugs or integration issues. This structured workflow allowed the team to keep development on track, meet academic deadlines, and ensure quality output.

5.3.2 Tools and Platforms Used

Table 5.1 provides an overview of the essential tools and platforms used for model training, development, deployment, and version control throughout the project.

Table 5.1: Details of tools and platform used

Category	Tool/Platform	Purpose
Model Training	Google Colab (GPU)	Training models
Programming Language	Python	Core DL development and preprocessing
DL Framework	TensorFlow / Keras	Model building, training, and TFLite conversion
Mobile Development	Flutter	Cross-platform mobile app development
IDE	Android Studio	Testing and packaging the Android app
Version Control	Git / GitHub	Code versioning and collaboration

5.3.3 Financial Analysis

This project was designed to be cost-effective and accessible for students or

independent researchers. Below is an approximate cost breakdown:

Table 5.2: Estimated Cost and Financial Analysis.

Resource/Item	Estimated Cost (BDT)	Remarks
Google Colab Pro (optional)	3500	Optional for extended GPU time and faster training
Computer (personal or lab use)	Existing Resource	Used for app development and testing
Internet Access	4200	Required for cloud training and resource access
Flutter and Android Studio (free tools)	0	Open-source and community-supported development tools
Cloud Storage/Backup (optional)	300	Google Drive or similar platforms
Total	8000 BDT	

5.4 Complex Engineering Problem

This section highlights the complex nature of the engineering problem addressed in this project. The task of designing a highly accurate, efficient, and mobile-compatible road damage detection system involves integrating multiple domains of knowledge, resolving conflicting design constraints, and ensuring deployment in real-world environments. It requires the combination of advanced deep learning models, software engineering principles, and mobile application development, all while maintaining ethical and performance standards. The following subsections provide detailed mappings to problem-solving categories, knowledge profiles, and engineering activities, demonstrating the depth and interdisciplinary nature of the project.

5.4.1 Complex Problem Solving

To successfully complete this project, several complex problem-solving aspects had to be addressed—from selecting the right model architecture to real-time mobile deployment. The table below maps the engineering problem with standard complexity attributes:

Table 5.3: Mapping with Complex Problem Solving

EP1	EP2	EP3	EP4	EP5	EP6	EP7
✓		✓	✓			✓

Justifications:

- EP1 – Depth of Knowledge: This project required a deep understanding of object detection using YOLO (v9s, v10s, v12s), model evaluation, TensorFlow Lite conversion, and cross-platform mobile app development using Flutter.
- EP3 – Depth of Analysis: A comparative analysis of multiple YOLO models was conducted using performance metrics like mAP and inference speed, followed by model optimization and lightweight conversion for deployment.
- EP4 – Familiarity of Issues: Challenges such as model quantization, on-device execution, and annotation complexities were addressed using prior knowledge gained from academic research and technical training.
- EP7 – Interdependence: The modular architecture of the system allows integration with future transportation or smart city frameworks, making it a flexible and scalable solution.

Mapping with Knowledge Profile for EP1

Table 5.4: Mapping with Knowledge Profile

K3	K4	K5	K6	K8
✓		✓		✓

Justifications:

- K3 – Engineering Fundamentals: Applied core knowledge of computer vision, deep learning, algorithm tuning, and dataset preprocessing.
- K5 – Engineering Design: Designed and compared multiple YOLO models, selected the best one, and integrated it into a functional mobile application.
- K8 – Research Literature: The system design and methodology were deeply informed by recent literature on YOLO variants, mobile deployment, and road condition analysis.

5.4.2 Engineering Activities

The system required a series of engineering activities including data preprocessing, model training, mobile deployment, performance tuning, and app- user integration. The mapping below reflects how the project relates to complex engineering activity indicators:

Table 5.5: Mapping with Complex Engineering Activities

EA1	EA2	EA3	EA4	EA5
✓		✓	✓	✓

Justifications:

- EA1 – Range of Resources: Utilized multiple tools and platforms including Google Colab (for training), TensorFlow (for model optimization), Flutter (for cross-platform mobile development), and Android Studio (for testing).
- EA3 – Innovation: Compared and deployed the latest YOLO models (v9s, v10s, v12s) to identify the best-performing model for on-device deployment.
- EA4 – Societal & Environmental Consequences: By enabling early road damage detection using smartphones, the system contributes to safer roads and sustainable maintenance.
- EA5 – Familiarity: Skills and concepts applied throughout this project were developed through academic experience, personal learning, and earlier model deployment experiments.

5.5 Summary

This chapter highlighted the engineering standards, design considerations, and interdisciplinary problem-solving approaches undertaken throughout the development of the proposed road damage detection system. By aligning the system with established software, hardware, and communication standards, the project ensured compatibility, efficiency, and portability across platforms. The

use of open-source tools like Google Colab, TensorFlow Lite, Flutter, and Android Studio allowed the system to be built cost-effectively while still adhering to engineering best practices.

The chapter also examined the societal and environmental impacts of the project, emphasizing how the mobile deployment of a lightweight detection model can contribute to safer road conditions, reduce manual inspection burdens, and support more sustainable infrastructure management. Ethical concerns and long-term sustainability goals were addressed by focusing on offline functionality, data privacy, and the reusability of the system architecture.

Additionally, a detailed project cost analysis revealed that the system can be developed and deployed at a student or prototype level with minimal expenses. Through structured mappings to complex engineering problems, knowledge domains, and engineering activities, the chapter illustrated how the project integrates technical depth, innovation, real-world applicability, and academic rigor. The insights presented here underscore the feasibility and impact of the system in both academic and practical environments.

Chapter 6

Conclusion

This chapter concludes the research by summarizing the key findings, highlighting the contributions made through the development of model for road damage detection. It also discusses the limitations faced during implementation and outlines potential directions for future improvements. The chapter reflects on the practical impact of the system and how it can evolve to become a more comprehensive, interpretable, and scalable solution for real-time medical diagnosis using mobile technology.

6.1 Summary

This research focused on developing a lightweight, accurate, and real-time road damage detection system using advanced object detection techniques. The core objective was to evaluate and compare the performance of three recent YOLO models—YOLOv9s, YOLOv10s, and YOLOv12s—on a road damage image dataset and identify the most optimal model for mobile deployment. After training and evaluating these models under identical conditions, the best-performing model was exported in TensorFlow Lite (.tflite) format and deployed through a Flutter-based Android mobile application.

The entire pipeline, starting from dataset annotation to real-time mobile deployment, was designed to be practical, cost-effective, and adaptable. It was implemented using open-source tools like Google Colab, TensorFlow, Flutter, and Android Studio. This project addressed multiple real-world challenges in road infrastructure monitoring by offering a scalable, mobile-first solution that reduces dependency on manual inspections, increases detection speed, and allows for broad public engagement in road safety. The system empowers end-users—whether individuals, municipal workers, or inspection teams—to detect road damage on-site using a smartphone and instantly visualize the results.

Beyond just model performance, the project emphasized deployment feasibility, making a meaningful contribution by bridging the gap between academic deep learning research and field-ready applications. The results confirm that such systems can be implemented affordably and effectively in both urban and rural contexts, especially in developing countries where infrastructure maintenance is often resource constrained.

6.2 Limitation

While the project achieved its intended objectives, several limitations were observed that warrant further exploration and improvement. First, the dataset used for training and testing was region-specific and relatively limited in terms of geographic diversity, road texture variations, and weather conditions. As a result, the model may not generalize well to unseen environments without additional training on a more diverse dataset.

Second, although TensorFlow Lite conversion enabled the model to run on Android smartphones, inference speed and consistency can vary depending on the device's hardware capabilities. Lower-end devices may experience latency or heating issues when running the model continuously. Furthermore, while the model performs well in detecting the presence of damage, it currently does not classify the damage into specific categories such as potholes, longitudinal cracks, or transverse cracks. This limits its usefulness in prioritizing maintenance tasks based on severity or type.

Additionally, the mobile application functions fully offline and lacks features such as GPS-based tagging, centralized damage reporting, or data syncing with cloud storage. This restricts its potential integration with municipal management systems or smart city infrastructure. Lastly, user interaction is limited to detection only, and there is no mechanism for user feedback, error reporting, or collaborative verification of detected damage.

6.3 Future Work

Future iterations of this project can explore multiple directions to enhance the system's robustness, scope, and usability. A key area of improvement is the expansion and diversification of the dataset. Incorporating images from different regions, weather conditions, lighting environments, and road types will help improve the model's generalization and resilience. Fine-tuning the model using transfer learning techniques on diverse datasets can further increase its accuracy across multiple deployment scenarios.

Another important enhancement is the integration of GPS-based geotagging, which would allow the detected damage to be mapped and reported automatically. This would make the application highly valuable for city planners, road authorities, and maintenance contractors. Additionally, enabling cloud connectivity could help build a centralized road damage database, offering insights into damage frequency, location-based trends, and maintenance history.

Incorporating classification of damage types would be a major step forward, allowing the model not only to detect damage but to identify what kind of defect it is. This would enable more informed maintenance planning and prioritization. Adding a feedback mechanism within the app—where users can validate or correct the model's predictions—can help continuously improve detection accuracy through user-generated data.

Finally, exploring the deployment of this system on edge devices such as Raspberry Pi or drones could extend its use cases beyond smartphones, enabling large-scale automated road inspection in both urban and remote areas. Such enhancements would significantly improve the system's impact and adaptability, aligning it with modern smart city and infrastructure monitoring goals.

References

[1] Wan, F., Sun, C., He, H., Lei, G., Xu, L., & Xiao, T. (2022). YOLO-LRDD: A lightweight method for road damage detection based on improved YOLOv5s. EURASIP Journal on Advances in Signal Processing, 2022(1), 98.

[2] Sami, A. A., Sakib, S., Deb, K., & Sarker, I. H. (2023). Improved YOLOv5-based real-time road pavement damage detection in road infrastructure management. Algorithms, 16(9), 452.

[3] Wu, C., Ye, M., Zhang, J., & Ma, Y. (2023). YOLO-LWNet: A lightweight road damage object detection network for mobile terminal devices. Sensors, 23(6), 3268.

[4] Guo, N., Zhang, R., Dong, Q., Wang, H., Xu, S., & He, G. (2025). EADG-YOLO: road damage detection by improving of YOLOv8n. Journal of Electronic Imaging, 34(1), 013026-013026.

[5] Vishwakarma, R. (2024, December). Exploring Real-Time Model Augmentation and Pretraining for Optimized Road Damage Detection. In 2024 IEEE International Conference on Big Data (BigData) (pp. 8469-8478). IEEE.

[6] Irsal, R. B. P., Utaminingrum, F., & Ogata, K. (2024). Swin transformer adaptation into YOLOv7 for road damage detection. Bulletin of Electrical Engineering and Informatics, 13(4), 2527-2536.

[7] Ding, K., Ding, Z., Zhang, Z., Yuan, M., Ma, G., & Lv, G. (2024). SCD-YOLO: a novel object detection method for efficient road crack detection. Multimedia Systems, 30(6), 1-16.

[8] Arya, D., Maeda, H., Ghosh, S. K., Toshniwal, D., Mraz, A., Kashiyama, T., & Sekimoto, Y. (2021). Deep learning-based road damage detection and classification for multiple countries. Automation in Construction, 132, 103935.

[9] Li, Y., Yin, C., Lei, Y., Zhang, J., & Yan, Y. (2024). RDD-YOLO: Road damage detection algorithm based on improved You Only Look Once version 8. Applied Sciences, 14(8), 3360.

[10] Wang, W., Yu, X., Jing, B., Tang, Z., Zhang, W., Wang, S., ... & Yang, L. (2025). YOLO-RD: A Road Damage Detection Method for Effective Pavement Maintenance. Sensors, 25(5), 1442.

[11] Jeong, D., & Kim, J. (2022, December). Road damage detection using yolo with image tiling about multi-source images. In 2022 IEEE International Conference on Big Data (Big Data) (pp. 6401-6406). IEEE.

[12] Zhang, Y., Zuo, Z., Xu, X., Wu, J., Zhu, J., Zhang, H., ... & Tian, Y. (2022). Road damage detection using UAV images based on multi-level attention mechanism. Automation in construction, 144, 104613.

[13] Xie, Y., Du, D., Wang, Z., Liu, Y., & Bi, M. (2025). YOLO-SFT: Road Damage Detection Algorithm Based on Feature Diffusion. Journal of Transportation Engineering, Part B: Pavements, 151(2), 04025004.

[14] Wang, W., Wu, B., Yang, S., & Wang, Z. (2018, December). Road damage detection and classification with faster R-CNN. In 2018 IEEE international conference on big data (Big data) (pp. 5220-5223). IEEE.

[15] Samma, H., Suandi, S. A., Ismail, N. A., Sulaiman, S., & Ping, L. L. (2021). Evolving pre-trained CNN using two-layers optimizer for road damage detection from drone images. Ieee Access, 9, 158215-158226.

[16] Xu, H., Chen, B., & Qin, J. (2021). A CNN-based length-aware cascade road damage detection approach. Sensors, 21(3), 689.

[17] Kulambayev, B., Nurlybek, M., Astaubayeva, G., Tleuberdiyeva, G., Zholdasbayev, S., & Tolep, A. (2023). Real-time road surface damage detection framework based on mask r-cnn model. International Journal of Advanced Computer Science and Applications, 14(9).

[18] Chen, Q., Gan, X., Huang, W., & Feng, J. (2020). Road Damage Detection and Classification Using Mask R-CNN with DenseNet Backbone. Computers, Materials & Continua, 65(3).

[19] Ale, L., Zhang, N., & Li, L. (2018, December). Road damage detection using RetinaNet. In 2018 IEEE International Conference on Big Data (Big Data) (pp. 5197-5200). IEEE.

[20] Desman, A., Muchtar, K., Oktiana, M., Fitria, M., Away, Y., Fardian, F., & Riza, H. (2025, January). Evaluating the Impact of Data Cleaning for Improving Road Damage Detection Through Vision Transformer. In 2025 IEEE International Conference on Consumer Electronics (ICCE) (pp. 1-6). IEEE.

[21] Roy, A. M., & Bhaduri, J. (2023). DenseSPH-YOLOv5: An automated damage detection model based on DenseNet and Swin-Transformer prediction head-enabled YOLOv5 with attention mechanism. Advanced Engineering Informatics, 56, 102007.

[22] Anzum, H., Sammo, M. N. S., & Akhter, S. (2024, March). Leveraging Data Efficient Image Transformer (DeIT) for Road Crack Detection and Classification. In 2024 International Conference on Advances in Computing, Communication, Electrical, and Smart Systems (iCACCESS) (pp. 1-6). IEEE.

[23] Jeong, J., Cho, J., & Lee, J. G. (2024, December). Optimized Road Damage Detection Using Enhanced Deep Learning Architectures for Improved Inference Speed and Accuracy. In 2024 IEEE International Conference on Big Data (BigData) (pp. 8453-8459). IEEE.

[24] Ashraf, A., Sophian, A., & Bawono, A. A. (2024). Crack detection, classification, and segmentation on road pavement material using multi-scale feature aggregation and transformer-based attention mechanisms. Construction Materials, 4(4), 655-675.

[25] Yang, Y., Guo, C., Zuo, C., & Yang, B. (2023, November). A Detection and Classification Method of Asphalt Pavement Crack based on Vision Transformer. In Proceedings of the 2023 3rd International Conference on Big Data, Artificial Intelligence and Risk Management (pp. 1125-1131).

221-15-6044

ORIGINALITY REPORT

11%	8%	5%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Daffodil International University Student Paper	3%
2	www.mdpi.com Internet Source	1%
3	Submitted to United International University Student Paper	<1%
4	arxiv.org Internet Source	<1%
5	dspace.daffodilvarsity.edu.bd:8080 Internet Source	<1%
6	www.ncbi.nlm.nih.gov Internet Source	<1%
7	Irina Nizovtseva, Pavel Mikushin, Ilya Starodumov, Ksenia Makhaeva, Simon Kraev, Dmitrii Chernushkin. "Bubble Detection in Multiphase Flows Through Computer Vision and Deep Learning for Applied Modeling", Mathematics, 2024 Publication	<1%