POTHOLE DETECTION AND ROAD SURFACE MONITORING WITH DEEP LEARNING

A PROJECT REPORT

Submitted by

HARSHITH RAJ C M	**211420118024**
RAKESH K	**211420118038**
S VARUNKUMAR	**211420118049**
VIGNESH KUMAR I	**211420118058**

in partial fulfillment for the award the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER AND COMMUNICATION ENGINEERING

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

MARCH 2024

BONAFIDE CERTIFICATE

Certified that this project report **" POTHOLE DETECTION AND ROAD SURFACE MONITORING WITH DEEP LEARNING "** is the bonafide work of **HARSHITH RAJ C M (211420118024), RAKESH K (211420118038), S VARUNKUMAR (211420118049), VIGNESH KUMAR I (211420118058)** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr. B.ANNI PRINCY M.E., Ph.D.,	Mrs. BRINDHA.J M.E., (Ph.D).,
HEAD OF THE DEPARTMENT	**SUPERVISOR**
PROFESSOR	ASSOCIATE PROFESSOR,
COMPUTER AND COMMUNICATION ENGINEERING,	COMPUTER AND COMMUNICATION ENGINEERING,
PANIMALAR ENGINEERING COLLEGE,	PANIMALAR ENGINEERING COLLEGE,
NAZARATHPETTAI,	NAZARATHPETTAI,
POONAMALLEE,	POONAMALLEE,
CHENNAI- 600123.	CHENNAI- 600123.

Certified that the above candidate(s) was/ were examined in the End Semester Mini Project Viva-Voce Examination held on...........................

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We express our deep gratitude to our respected Secretary and Correspondent **Dr. P. CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We would like to extend our heartfelt and sincere thanks to our Directors Tmt. **C.VIJAYARAJESWARI, Dr. C. SAKTHIKUMAR, M.E., Ph.D**., and **Dr.SARANYASREE SAKTHIKUMAR B.E., M.B.A., Ph.D**., for providing us with the necessary facilities for completion of this project.

We also express our gratitude to our Principal **Dr.K.MANI, M.E., Ph.D.,** for his timely concern and encouragement provided to us throughout the course.

We thank the HOD of CCE Department, **Dr. B. ANNI PRINCY, M.E., Ph.D**., Professor, for the support extended throughout the project.

We would like to thank our supervisor **Mrs. BRINDHA.J, M.E.,** Associate Professor and all the faculty members of the Department of CCE for theiradvice and suggestions for the successful completion of the project.

HARSHITH RAJ C M
RAKESH K
S VARUNKUMAR
VIGNESH KUMAR I

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVATIONS

ABBREVATIONS	FULLFORM
CNN	Convolutional Neural Network
MQQT	Message Queuing Telemetry Transport
YOLO	You Only Look Once
CV	Computer Vision
ODM	Object Detection Model
mAP	Mean Average Precession
HOG	Histogram of Oriented Gradients
ITS	Intelligent Transport System
GPS	Global Positioning System
API	Application Programming Interface
UML	Unified Modeling Language
DFD	Data Flow Diagram
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
DWT	Discrete Wavelet Transform
COCO	Common Object in Context
SMS	Short Message Service
LIDAR	Light Detection and Ranging
RNS	Residue Number System

ABSTRACT

Potholes on road surfaces pose significant safety risks and maintenance challenges. This paper explores the development of a deep learning-based system for the detection of potholes and monitoring road surface conditions. Leveraging convolutional neural networks (CNNs) and image processing, the proposed system aims to provide an automated, accurate, and real-time solution for identifying potholes and assessing road surface quality. Periodic model validation and evaluation are performed to monitor its progress and ensure its effectiveness in detecting potholes with an accuracy of 94%. The objective is to improve road safety, reduce maintenance costs, and enhance transportation efficiency through an advanced technological approach. This research focuses on the development of the system integrated with deep learning algorithms for the detection of potholes and real-time monitoring of road surface conditions. Potholes are a significant cause of road accidents and vehicular damage, emphasizing the need for an automated and efficient solution. This research explores the fusion of deep learning techniques to create a portable, self-contained technology capable of accurately identifying and reporting potholes while continuously assessing the quality of road surfaces. The aim is to enhance road safety, reduce maintenance costs, and streamline transportation infrastructure management.

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

This paper presents a solution to address safety risks and maintenance challenges posed by road potholes through a deep learning-based system for real-time detection and monitoring. The objective is to develop an embedded system integrating CNNs to detect potholes and assess road surface conditions accurately. Leveraging image processing techniques, it offers automated, accurate, and real-time solutions for identifying potholes and evaluating road quality. The introduction underscores the necessity for efficient systems to monitor road surfaces, highlighting the importance of deep learning. Additionally, the system features MQTT messaging to send alerts upon pothole detection, facilitating prompt maintenance actions. Methodology involves preprocessing, segmentation, feature extraction, and CNN-based classification. The research emphasizes the potential of deep learning to revolutionize road maintenance by providing timely alerts and insights into infrastructure health. Through proactive analysis and MQTT-based notification capabilities, the system streamlines maintenance operations, contributing to enhanced road safety, reduced costs, and optimized infrastructure management. This holistic approach targets timely pothole detection and proactive maintenance planning, enhancing transportation networks' resilience and efficiency.

1.2 EXISTING SYSTEM

The predominant methods for detecting potholes currently revolve around manual inspections or sensor-based systems, both of which present notable limitations. Manual inspections, while traditional, are inherently time-consuming and prone to human error. Relying on human observation alone not only slows down the detection process but also introduces inconsistencies in identifying and cataloging road defects accurately. On the other hand, sensor-based systems offer a more technologically advanced approach, yet they often fall short in providing real-time capabilities. These systems may encounter challenges with accuracy and precision, especially when faced with varying road conditions or environmental factors. Additionally, the high implementation costs associated with sensor-based solutions pose barriers to widespread adoption, limiting their accessibility and effectiveness in pothole detection and road surface monitoring.

1.2.1 DISADVANTAGES OF EXISTING SYSTEM

Here are some potential disadvantages of existing pothole detection systems:

1. Dependence on Camera Coverage - Effective detection hinges on sufficient camera coverage, necessitating strategic placement and significant infrastructure investment.

2. Computational Resource Demands - Real-time processing requires robust computational resources, leading to high deployment and maintenance costs due to sophisticated hardware and algorithms.

3. Complex Setup and Training - Time-consuming tasks like algorithm fine-tuning, camera calibration, dataset creation demand expertise attention.

4. Maintaining System Compatibility - Ensuring system compatibility across diverse road surfaces and environmental conditions poses ongoing challenges, requiring regular maintenance and updates.

1.3 PROPOSED SYSTEM

The proposed system integrates deep learning algorithms into an embedded platform, revolutionizing pothole detection and road surface monitoring. Through sensors and image processing, it offers real-time monitoring, leveraging CNNs for accurate pothole detection. Immediate alerts upon detection enable prompt responses and repairs, enhancing road safety and minimizing accidents. Beyond pothole detection, the system continuously assesses surface quality, providing insights for predictive maintenance strategies. This proactive approach allows government authorities to anticipate maintenance needs, prioritize repairs, and allocate resources efficiently, thereby extending infrastructure lifespan and reducing overall maintenance costs. In essence, the system represents a holistic solution, combining advanced technology with proactive strategies to optimize road safety and infrastructure management.

1.3.1 ADVANTAGES OF PROPOSED SYSTEM

Here are some potential advantages of the proposed pothole detection system using computer vision:

1. Rapid Response - By continuously monitoring the road environment and processing data in real-time, the system can quickly detect and respond to emergencies, minimizing delays and improving overall emergency response effectiveness.
2. Adaptability - The adaptability of the system to various environmental conditions further enhances its utility. Whether in urban or rural settings, day or night, and under different weather conditions, the system maintains its effectiveness in detecting potholes and emergency vehicles.
3. Accuracy - The proposed system the utilization of deep learning algorithms ensures higher accuracy in identifying potholes, leading to more precise maintenance actions.
4. Immediate Alerts - The system's real-time monitoring capabilities enable immediate alerts upon pothole detection, facilitating prompt maintenance actions. This proactive approach to pothole detection and repair not only enhances road safety but also reduces the risk of accidents and vehicle damage caused by deteriorating road conditions.

CHAPTER 2

LITERATURE SURVEY

Pothole Detection is a popular topic in the field of Computer Vision. There have been many research papers, articles,and books written on this topic. Here are some of the notable literature surveys and research papers related to email spam filtering:

PAPER 1: Patches Detection on Roads using Machine Learning.

AUTHORS: K Balaji, M Harish, P Jaya Prakash, C Silpa.

ABSTRACT: Roads are the most commonly used mode of transportation. However, due to the frequent use of roads, it requires an organized assistance. Manually screening the potholes or damaged patches is highly impossible and inaccurate. This research work investigates the detection of potholes or patches by using a camera. Image processing models are used to detect the potholes or patches. The proposed system has been evaluated in a Desktop environment by using the OpenCV library. To perform this task, a smart image processing model with Hough transform are used.

MERITS: In compared to earlier techniques, the recommended module achieves 90.5 percent accuracy.

DEMERITS: The algorithm wrongly recognizes road cracks and shadow covered potholes as potholes when we use them as target photos.

PAPER 2 : Investigation of YOLO models in the detection and classification of multiple negative road anomalies.

AUTHORS: Swapna Kulkarni, Neha Mittal, Rajiv Ranjan Gupta.

ABSTRACT: Negative-road anomalies like potholes, humps, cracks present on the road surface impart a serious discomfort to drivers and passengers. Moreover, in such a road condition if a vehicle or pedestrian suddenly cross the road it becomes a significant challenge for the driver to drive safely. Conventional approaches are time consuming and lack in detecting multiple anomalies. Owing to the advantage of detection of multiple objects in an image or video, computer vision (CV) based object detection algorithms are best suited for timely detection of negative anomalies. Herein we present a comparative analysis of nine different versions of the state-of-the-art object detection models (ODMs): you only look once (YOLO) in detecting multiple negative anomalies. To evaluate the performance of the ODMs, metrics such as precision, recall, F1 score and mean average precision (mAP) are measured. The frameworks are experimented using 3391 road images including potholes, speed breakers, vehicles, and humans on different road surface conditions. Upon studying the performance of all the nine variants of ODMs we observed that YOLOv6s is outperforming others resulting in accuracy of 85.5%, mAP: 71.2 %, precision: 74.9 %, recall: 66 %, and F1 score: 70.2 % for a reasonably small image data set containing multiple negative road anomalies.

MERITS: YOLOv6s took 2 hours 44 minutes for training the dataset and tested images in 0.9 ms.

DEMERITS: The accuracy of the method may be highly dependent on the precise calibration of the cameras used.

.

PAPER 3 : Computer vision based detection and localization of potholes in asphalt pavement images.

AUTHORS: Kanza Azhar, Fiza Murtaza, Muhammad Haroon, Hafiz Adnan Habib.

ABSTRACT: Asphalt pavement distresses have significant importance in roads and highways. This paper addresses the detection and localization of one of the key pavement distresses, the potholes using computer vision. Different kinds of pothole and non-pothole images from asphalt pavement are considered for experimentation. Considering the appearance-shape based nature of the potholes, Histograms of oriented gradients (HOG) features are computed for the input images. Features are trained and classified using Naïve Bayes classifier resulting in labeling of the input as pothole or non-pothole image. To locate the pothole in the detected pothole images, normalized graph cut segmentation scheme is employed. Proposed scheme is tested on a dataset having broad range of pavement images. Experimentation results showed 90% accuracy for the detection of pothole images and high recall for the localization of pothole in the detected images.

MERITS: Only applicable to asphalt pavement roads.

DEMERITS: It may not provide comprehensive coverage across all road types, especially in rural or less frequently traveled areas.

PAPER 4 : Image Features Selection Based on Computer Vision Techniques to Detect Potholes for Intelligent Transport System.

AUTHORS: Mohammad Omar and Pradeep Kumar.

ABSTRACT: Computer vision is an emerging area for image processing that is well utilized in the field of Intelligent Transport Systems. There are various techniques used to increase the efficiency of ITS such as GPS, radar-based, sensors utilization, more importantly, through Surveillance, etc. It found that costs for all techniques were high to implement on a large scale. The need of the hour is to have efficient techniques or models that will help to reduce the cost of ITS. Computer vision is a field that provides cutting-edge solutions in ITS. This paper explores the image features selection based on computer vision techniques to detect the potholes on the road. It will help to improve the Intelligent Transport System for a smart city. In this paper, the authors will discuss the computer vision techniques like Corner Detection-Harris, HOG, and Feature Extraction.

MERITS: Histogram of Gradients are seen as efficient techniques of feature selections required for the development of models.

DEMERITS: Some areas may have limited data samples, leading to under sampling issues.

PAPER 5 : Detection of Road Potholes Using Deep Learning Based Improved YOLOv4 Network.

AUTHORS: Ruchi, S. Indu, Rohit Kumar.

ABSTRACT: In our daily lives, when we want to connect with the outside world, we use roads as a connecting method between various locations. Road maintenance is a very important task to keep them safe. Potholes on roads may intensify the number of accidents, so detecting potholes and enlightening the concerned department can save the roads from getting worse. The traditional manual detection method was a time-consuming and laborious task. So in this paper, we have trained and tested different deep neural network-based models to detect the presence of potholes on roads. We have taken publicly available dataset from Kaggle and then potholes in road images are identified using object identification algorithms. Deep learning models for one-stage object detection that can be configured in a number of ways, including YOLOv4-CSPDarknet53, YOLOv5s, and modified YOLOv4-CSPDarknet 53 are used to compare their performance in detecting potholes. Performance is compared using Average precision, Average recall, mAP@50%, and inference time. The results showed that the modified YOLOv4 architecture outperformed compared with the other deep learning models with the accuracy of 88% , highest Average precision (0.84), Average recall (0.79), F1 - Score (0.81), and mAP@50% (0.82).

MERITS: It has the highest average precision (84%), average recall (79%), and mAP@.5 (82.39%).

DEMERITS: Relies on simulated datasets; may not fully capture real-world variability.

CHAPTER 3

SYSTEM SPECIFICATIONS

To build a pothole detecting and road surface monitoring system using deep learning, the following system specifications are:

3.1 HARDWARE:

- RAM: 8GB or above
- Processor: Ryzen 5 or above
- Storage: At least 100 GB of free disk space
- Internet: Broadband or high-speed internet connection

3.2 SOFTWARE:

- Anaconda Navigator
- Operating system: Windows, Linux, or mac
- Python IDLE: Version 3.6 or higher
- Python libraries: TensorFlow, OpenCV, etc.

3.3 DATA:

- A dataset of labeled roads (damaged, cracks, pothole) for training the model. This dataset should have at least 10,000 different images for better accuracy.

CHAPTER 4

SYSTEM ARCHITECTURE

The system architecture for the pothole detection project orchestrates a blend of technologies to streamline operations. Initially, road images undergo preprocessing with OpenCV, enhancing quality through techniques like Gaussian blur. Following this, YOLO and TensorFlow drive machine learning model development for real-time pothole detection as shown in figure 4.1, bolstered by PyTorch for deep learning algorithm implementation. MQTT enables swift message alerts upon pothole identification, facilitating timely maintenance. ThingSpeak acts as an IoT analytics platform, collecting and visualizing live data streams on pothole detection and road conditions. This cohesive architecture ensures efficient pothole detection, proactive maintenance planning, and optimal infrastructure management.

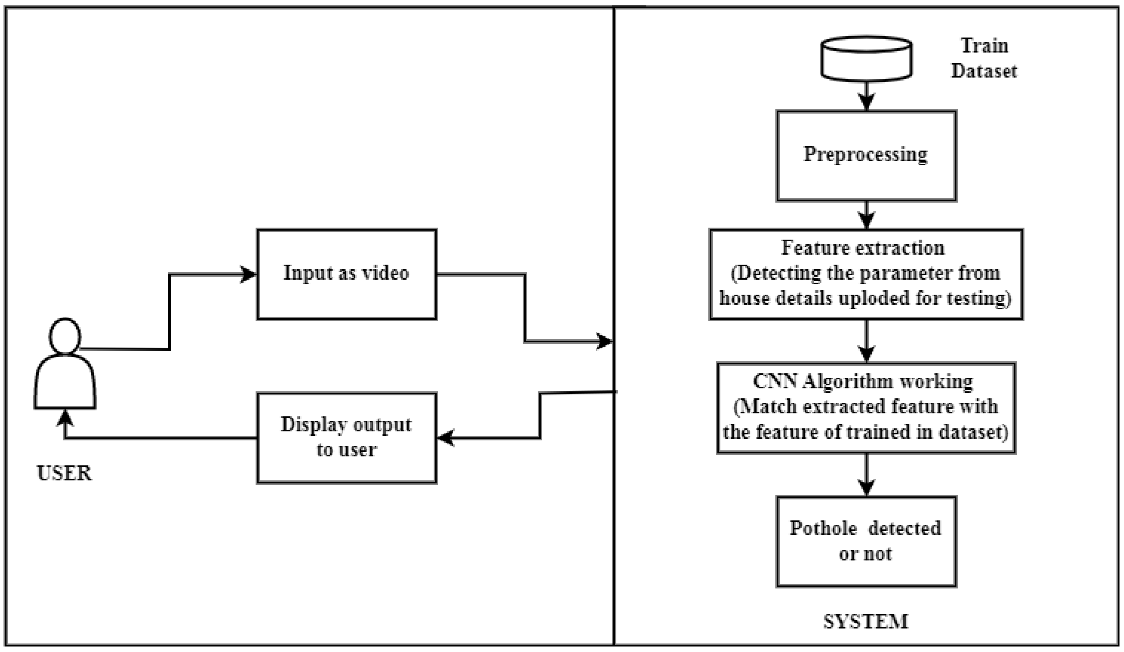

Train Dataset
Preprocessing
Feature extraction (Detecting the parameter from house details uploded for testing)
CNN Algorithm working (Match extracted feature with the feature of trained in dataset)
Pothole detected or not
SYSTEM
Input as video
Display output to user
USER

Figure 4.1: System architecture of pothole detection

4.1. OPENCV

OpenCV serves as the backbone for image processing tasks in the pothole detection project. It offers diverse functions for filtering, edge detection, and segmentation, crucial for preprocessing images. Techniques like Gaussian blur and Canny edge detector enhance image quality and identify road boundaries, while segmentation isolates road surfaces for further analysis.

4.2. TENSORFLOW

TensorFlow, a prominent open-source platform, complements YOLO in the pothole detection project. It facilitates the development, training, and deployment of machine learning models, including neural networks utilized in pothole detection tasks. TensorFlow's robust framework enables efficient processing of large datasets, optimizing model performance for accurate identification of road defects. Its scalability and flexibility empower researchers and developers to iterate on model designs and achieve superior accuracy in real-time pothole detection.

4.3. YOLO

YOLO (You Only Look Once) is an object detection algorithm renowned for real-time pothole identification. Trained on annotated road image datasets, YOLOv5 swiftly identifies potholes from live camera feeds. Its speed and precision facilitate prompt maintenance actions, enhancing road safety and infrastructure management. By enabling rapid detection and reporting of road defects, YOLO plays a pivotal role in ensuring timely repairs, minimizing hazards, and optimizing transportation efficiency.

4.4. PYTORCH

PyTorch, another key framework in the project, offers dynamic computational graph capabilities, making it ideal for implementing deep learning algorithms for pothole detection. Its flexibility and ease of use allow for experimentation with complex model architectures and hyperparameters, leading to optimized detection performance. PyTorch's extensive library simplifies model development, enabling researchers to iterate on designs and achieve superior accuracy in pothole detection tasks.

4.5. MQTT

Message Queuing Telemetry Transport enables real-time message alerts in the pothole detection project. Upon detecting a pothole, the system generates an alert message containing location and severity data. This message is published to n MQTT broker, facilitating prompt communication and coordination between stakeholders for efficient maintenance actions.

4.6. THINGSPEAK

ThingSpeak, an IoT analytics service, aggregates, visualizes, and analyzes live data streams in the cloud. ThingSpeak's graph visualization capabilities are instrumental in our project for displaying live data streams related to pothole detection and road surface conditions. By sending data from our system directly to ThingSpeak API integration, we can aggregate and analyze this data in real-time.

CHAPTER 5

SYSTEM DESIGN USING UML

5.1 CLASS DIAGRAM

In a class diagram as illustrated in figure 5.1, classes are represented with boxes. Class diagram is the type of structure diagram which describe the structure of a system by showing classes, attributes, operations and relationships among the classes. Purpose of class diagram is to express the static structure of a system in terms of classes and relationships among those classes.

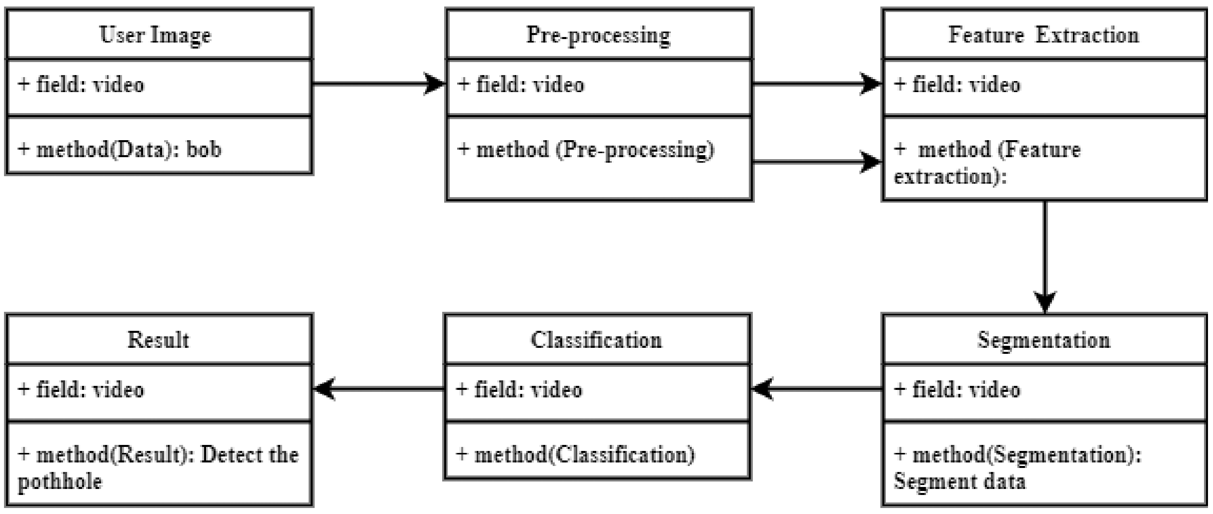


Figure 5.1: Class diagram for pothole detection

5.2 USE CASE DIAGRAM

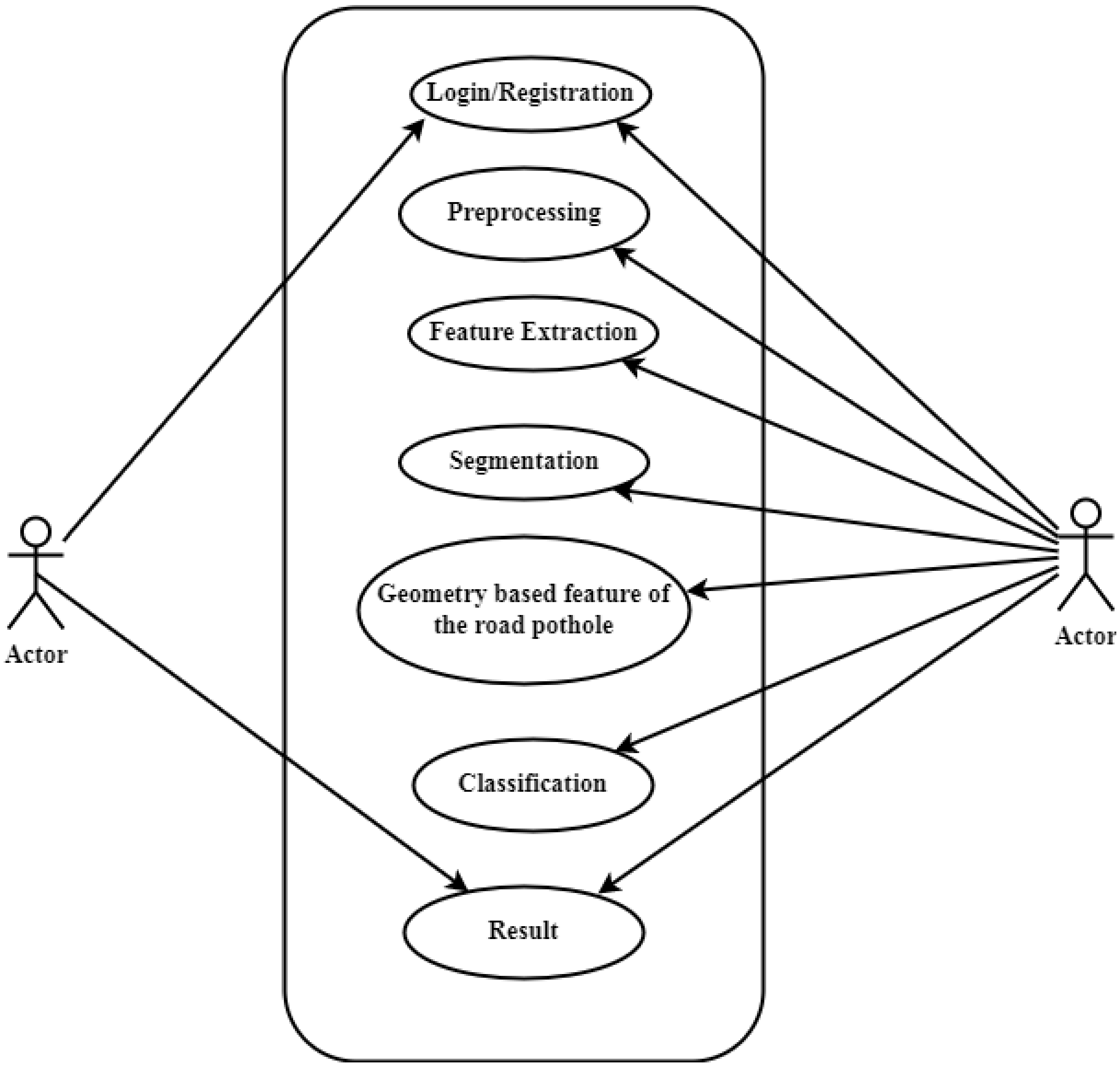


Figure 5.2: Use Case diagram for pothole detection

A use case diagram at its simplest is a representation of a user’s interaction with the system and depicting the specifications of a use case. A use case diagram depicted in figure 5.2, portrays the different types of users of a system and the various ways that they interact with the system. This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as well.

5.3 SEQUENCE DIAGRAM

A sequence diagram is a type of interaction diagram. This type of diagram as shown in figure 5.3, states clearly how and in what order a group of objects works together. Software developers and business professionals use these diagrams mainly for a new system or to document an existing process.

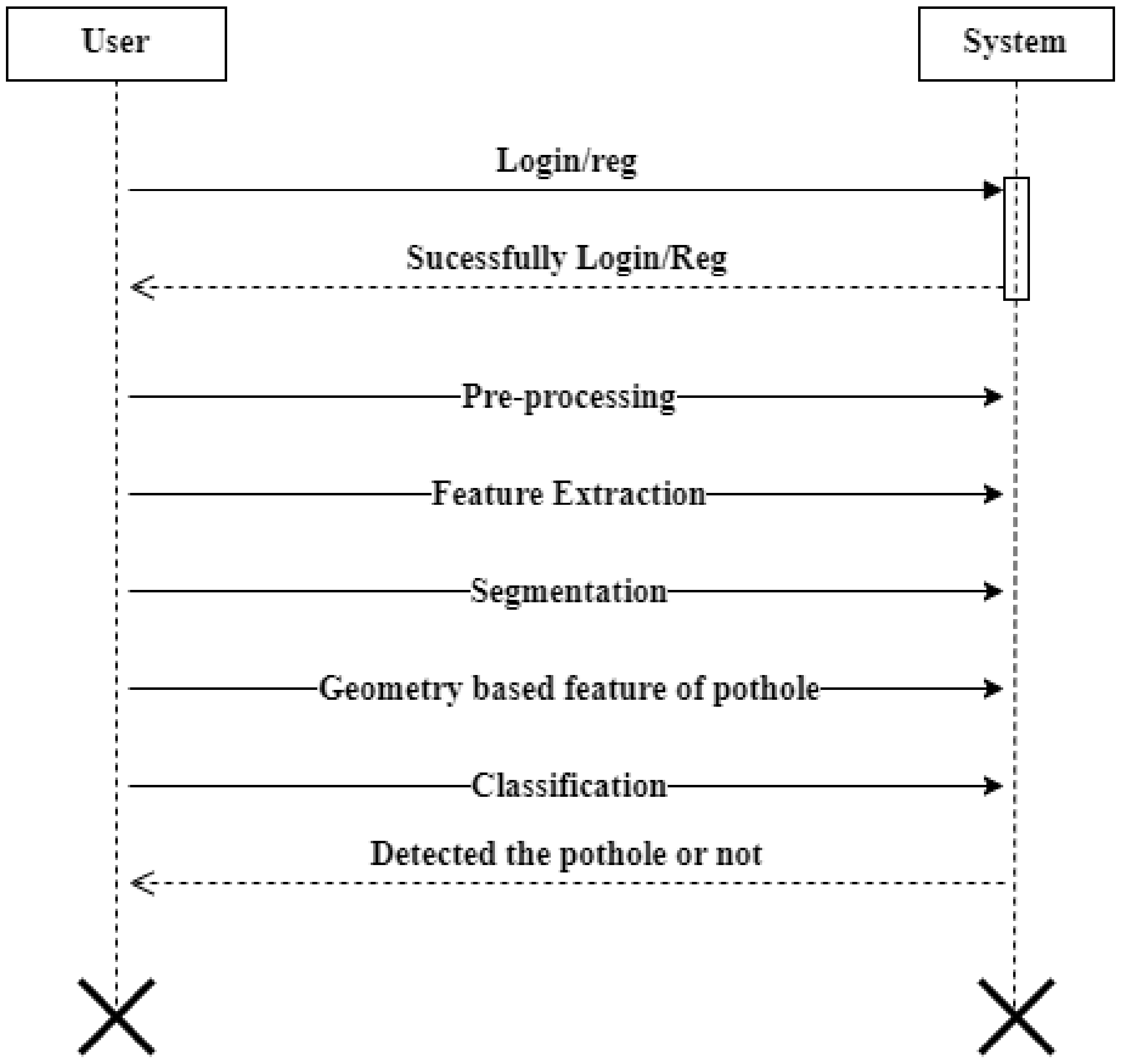


Figure 5.3: Sequence diagram for pothole detection

5.4 ACTIVITY DIAGRAM

From figure 5.4, Activity Diagram can be used to describe the dynamic aspects of the system. It is a flow chart which represents a flow from one activity to another. So, activity diagram is considered as a flow chart. Main element used in this diagram in this diagram is activity itself.

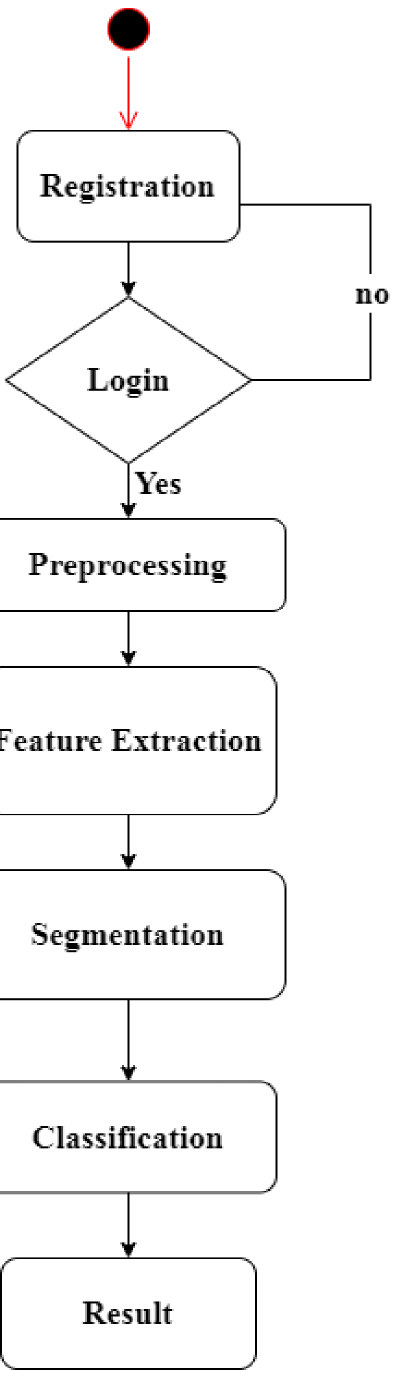


Figure 5.4: Activity diagram for pothole detection

5.5 COMPONENT DIAGRAM

A component diagram, also known as a UML component diagram as exemplified in figure 5.5, describes the organization and wiring of the physical components in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development.

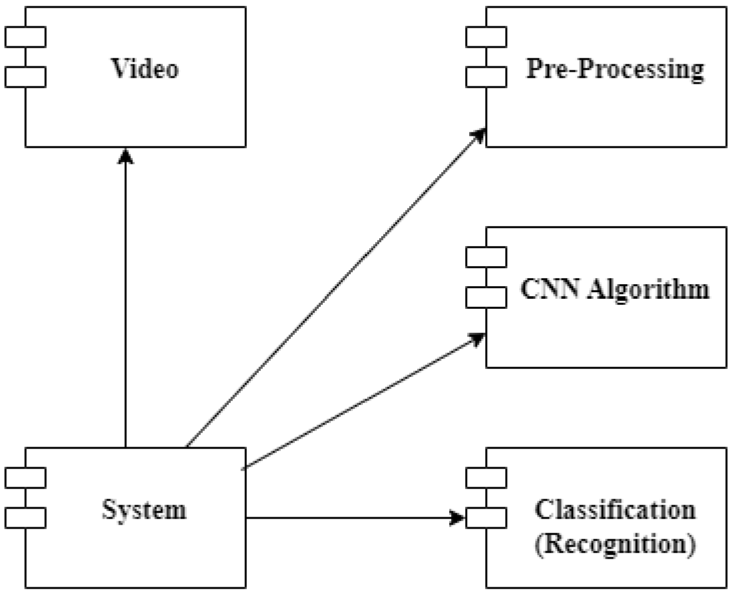


Figure 5.5: Component diagram for pothole detection

CHAPTER 6

MODULES

6.1. Data Acquisition Module

The Data Acquisition Module is pivotal in pothole detection, utilizing diverse devices like vehicle-mounted cameras and drones to capture road surface images. This module ensures adaptability to different scenarios, offering dynamic perspectives. Geographical diversity is achieved by capturing images from various locations, enriching the dataset for robust performance. Temporal variability, accounting for different times and weather conditions, creates a representative dataset. The module's impact lies in its ability to enrich the dataset with diverse, real-world scenarios, enhancing adaptability and overall effectiveness in identifying potholes across varied environments.

6.2. Data Pre-processing Module

The Data Pre-processing Module optimizes pothole detection by resizing, normalizing, and reducing noise in captured images. Resizing ensures uniformity for consistent feature extraction, while normalization adjusts pixel values for standardized analysis. Noise reduction techniques, such as filtering, improve the signal-to-noise ratio, enhancing the overall quality of the dataset and preparing it for subsequent computer vision algorithms.

6.3 Feature Extraction Module

The Feature Extraction Module identifies potholes by analysing texture, colour, and shape features. Texture analysis captures road surface patterns, colour features detect anomalies indicative of degradation, and shape characterization pinpoints potential pothole locations. Dimensional considerations and hierarchical feature selection optimize the dataset, contributing to accurate pothole identification and enhancing overall system performance.

6.4 Object Detection Module

The Object Detection Module employs advanced computer vision algorithms, particularly convolutional neural networks (CNNs), to identify and locate potential potholes within captured images. Leveraging CNNs enables the module to recognize intricate patterns associated with potholes. The output provides precise localization of regions of interest, streamlining subsequent analysis. Pattern recognition, enhanced accuracy with deep learning, and the presentation of output as regions of interest contribute to the effectiveness of the Object Detection Module in accurately identifying potholes.

6.5 Classification Module

The Classification Module determines whether the detected segments contain actual potholes or are false positives. Utilizing machine learning models trained on labelled datasets, it enhances the system's ability to accurately classify and distinguish road anomalies.

CHAPTER 7

PROJECT IMPLEMENTATION

This project presents the implementation of a real-life pothole detection system, which follows a systematic workflow to effectively address the issue. Beginning with data acquisition from various sources like images, videos, or live streams of road surfaces, the captured data undergoes pre-processing steps including resizing and basic normalization to ensure standardized analysis. Utilizing the YOLOv5 object detection algorithm, the system identifies and precisely localizes potential potholes within the captured imagery without the need for segmentation. Subsequently, post-processing techniques are applied to refine detection results by eliminating noise, ensuring accurate identification of potholes. Finally, visualization methods are employed to represent detected potholes visually, facilitating further interpretation and action.

7.1 DIGITAL IMAGE PROCESSING

Digital image processing refers to the manipulation and analysis of digital images using various algorithms and techniques to enhance, interpret, or extract information from visual data. Digital image processing plays a pivotal role in the pothole detection project, offering a robust framework for analyzing and interpreting visual data captured from road surfaces. Through a variety of techniques and algorithms, digital image processing enhances the quality of captured images, standardizes their format, and extracts valuable information crucial for pothole detection.

Preprocessing techniques such as resizing, normalization, and noise reduction ensure that images are optimized for analysis, improving the accuracy of subsequent detection algorithms. The object detection algorithm, a cornerstone of the project, heavily relies on digital image processing to accurately identify and localize potential potholes within the images. Additionally, segmentation and post-processing techniques further demonstrates the indispensable role of digital image processing in the project's success.

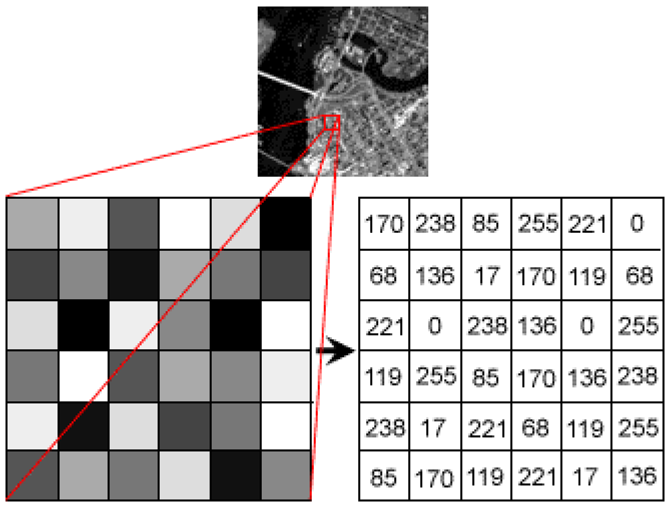
7.1.1 FUNDAMENTALS OF DIGITAL IMAGE

IMAGE

An image refers to a two-dimensional array of pixel values representing visual data captured by cameras. Image represents a visual depiction of a road surface captured by cameras which is shown in figure 7.1.

Figure 7.1: Pothole Road

The pixels that constitute an image are ordered as a grid (columns and rows) like in figure 7.2. Each pixel consists of numbers representing magnitudes of brightness and color.



170	238	85	255	221	0
68	136	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136

Figure 7.2: Pixels ordered as grids

As shown in figure 7.3, each pixel has a color represented by a 32-bit integer, where the first eight bits determine the redness of the pixel, the next eight bits the greenness, the next eight bits the blueness, and the remaining eight bits the transparency of the pixel.

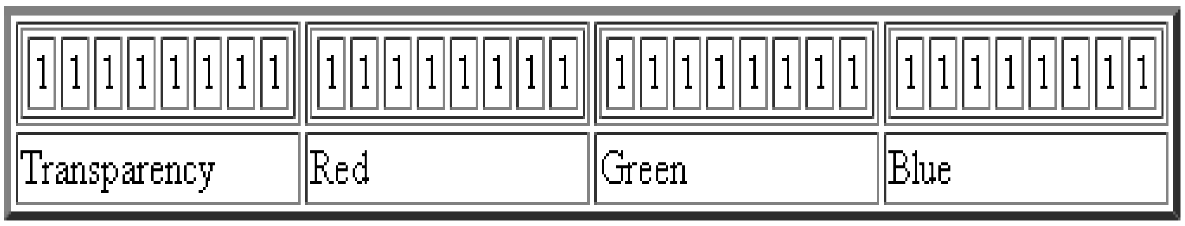


Figure 7.3: Color palette of the input image

Additionally, the project involves the monitoring of road surfaces to assess their quality and identify areas prone to damage, such as potholes. Digital image processing techniques are applied to these images to extract features, monitor road surfaces, identify objects such as potholes, and enhance interpretability. Ultimately, images play a fundamental role in the project, enabling the system to detect and address potholes and monitor road surfaces effectively.

IMAGE FILE SIZE

Image sizes play a crucial role in various aspects of the system's functionality. Image sizes refer to the dimensions of the captured images, typically represented as height and width in pixels. These dimensions directly impact the computational requirements, accuracy, and efficiency of the image processing pipeline.

Firstly, the image sizes influence the performance of the object detection algorithm, YOLOv5, which is utilized for identifying and localizing potential potholes within the captured images. The algorithm's inference speed and detection accuracy are influenced by the resolution of the input images. Higher resolution images provide more detailed information but may require increased computational resources for processing.

Furthermore, image sizes also affect the pre-processing steps applied to the captured images. Techniques such as resizing, which adjusts the dimensions of the images, are commonly employed to ensure uniformity and compatibility with the object detection algorithm. In this project, images are strictly set to (640x,640x) pixels. Moreover, in the context of data storage and transmission, image sizes are crucial considerations. Larger image sizes result in higher storage requirements and longer transmission times, particularly relevant when dealing with large volumes of image data captured from multiple cameras. By carefully managing the dimensions of the captured images, the pothole detection system can achieve optimal performance while effectively addressing the challenges associated with real-time monitoring of road surfaces.

IMAGE FORMAT

Image file formats are standardized means of organizing and storing images. Image files are composed of either pixel or vector. the images are primarily processed and analyzed in their raw form, without explicit conversion or manipulation of image formats. The project operates with images stored in common formats such as JPEG, PNG, or other formats supported by OpenCV, which is commonly used for image processing tasks in Python.

7.1.2 STEPS IN DIGITAL IMAGE PROCESSING

Utilizing OpenCV and TensorFlow frameworks, the digital image processing pipeline involves a series of essential steps. Initial image acquisition, sourced from mobile devices capturing diverse road surfaces in Chennai, ensures a comprehensive dataset. Subsequent enhancement, restoration, and segmentation techniques refine image quality to isolate key features and compression techniques reduce data redundancy for efficient storage and transmission.

Feature extraction identifies and extracts relevant patterns from the images, such as edges, textures, and colour distributions, facilitating accurate pothole detection. Interpretation of these features involves analysing their significance in relation to road conditions, aiding in informed decision-making for maintenance and repair efforts.

These methods collectively as shown in figure 7.4 enable robust pothole detection and analysis, enhancing road safety and infrastructure management in urban environments.

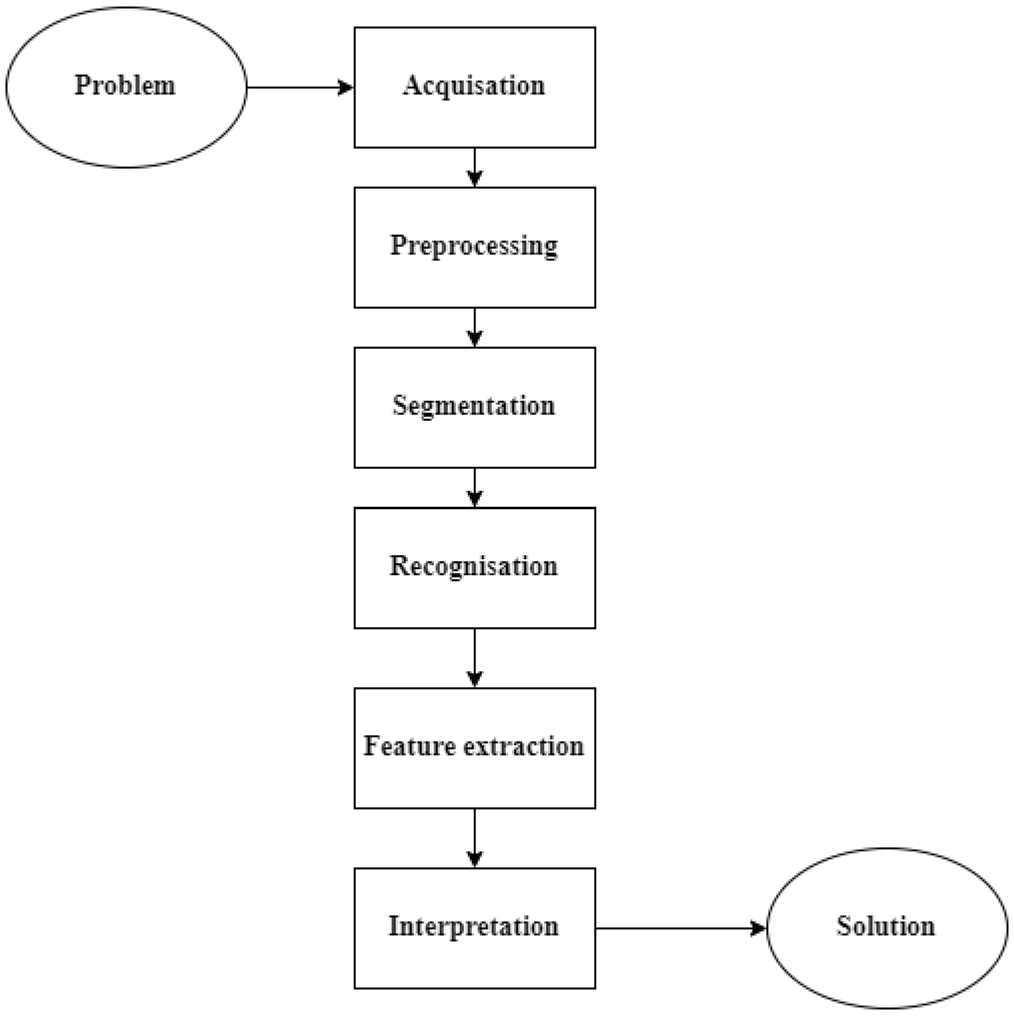


Figure 7.4: Steps in digital image processing

IMAGE ACQUISITION

Image acquisition is the initial step in gathering data for analysis. Utilizing mobile devices like the Redmi Note 10 Pro and MI 11x, we systematically captured diverse road surfaces across Chennai. The mobile device used is in the figure 7.5. This meticulous process ensured the inclusion of various environmental conditions and road types, vital for training robust pothole detection algorithms.

By covering different locations and scenarios, we enriched our dataset, enhancing its representativeness and suitability for comprehensive analysis. Leveraging the cameras on these devices provided multiple perspectives, thereby improving the adaptability and effectiveness of our pothole detection system. Consequently, we obtained a wealth of visual data, ranging from different lighting conditions to road textures, facilitating a deeper understanding of the challenges associated with pothole detection. In essence, image acquisition served as the foundation for building an effective pothole detection system, enabling us to address the pressing issue of road maintenance.

Figure 7.5: Device used to capture datasets

IMAGE ENHANCEMENT

Image enhancement is a fundamental aspect of digital image processing, focusing on refining visual quality and highlighting specific features within an image. Image enhancement is employed as preprocessing steps to refine the input images before subjecting them to further analysis. Among the techniques, steerable filters emerge as a powerful tool for enhancing image features relevant to pothole detection. By applying steerable filters, directional information within the images, such as edges and textures indicative of road damage, can be effectively emphasized as shown in figure 7.6.

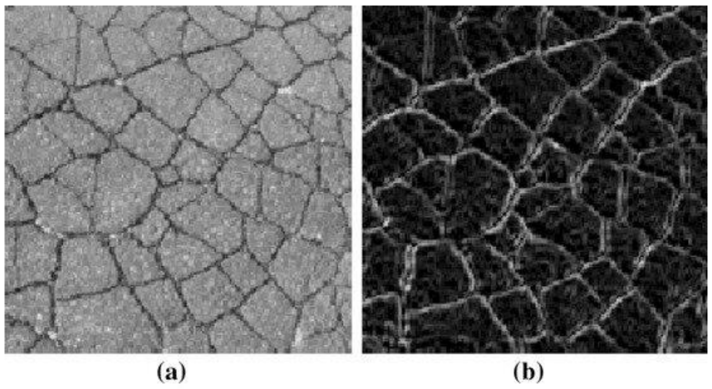


Figure 7.6: Image enhancement of damaged road

This enhancement helps in highlighting subtle details and suppressing noise, ultimately improving the visibility and distinctiveness of potential potholes. Additionally, other image enhancement methods such as contrast adjustment, noise reduction, and sharpening may complement the use of steerable filters to further refine the images for enhanced detection accuracy. Through the integration of image enhancement techniques, including steerable filters, the pothole detection system can effectively preprocess the input images, ensuring optimal conditions for subsequent analysis and detection algorithms.

IMAGE RESTORATION

Image restoration plays a crucial role in enhancing the quality of images, leveraging specialized algorithms to address various forms of degradation. Within our system, the denoising algorithm, Gaussian denoising is utilized to effectively suppress noise while preserving important image features from factors like low light conditions or sensor limitations, ensuring clearer and more interpretable images for subsequent analysis. Gaussian denoising is a widely used method for reducing noise in images by applying a Gaussian filter to smooth out pixel intensity variations while preserving important image features.

Furthermore, the deblurring technique constitutes another vital aspect of image restoration, particularly useful in mitigating the effects of motion blur or out-of-focus blur commonly encountered in images captured from moving vehicles or under challenging environmental conditions which is shown in the figure 7.7.

By applying the deblurring algorithm named motion deblurring, our system can restore sharpness and clarity to blurred regions of the images, thereby enhancing the accuracy of pothole detection. Motion deblurring is a technique aimed at restoring sharpness in images affected by motion blur, often caused by camera movement during image capture.

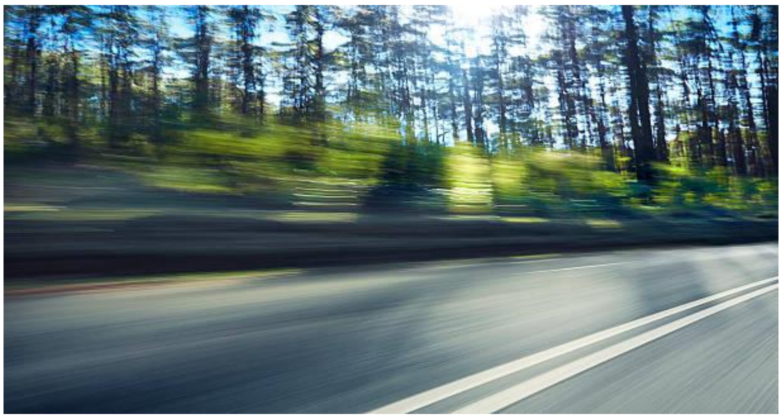


Figure 7.7: Motion blurred road

Moreover, inpainting methods are employed to address missing or damaged regions within the images, ensuring a complete and coherent representation of the road surface. These techniques fill in the gaps or damaged areas in the images, seamlessly reconstructing missing information to facilitate accurate analysis and interpretation of the road conditions. We used the PatchMatch algorithm that fills in missing or damaged regions of an image by searching for similar patches in neighboring areas and seamlessly blending them to reconstruct the missing information.

It effectively restores the completeness of images by intelligently inferring pixel values based on surrounding patches. By integrating these advanced image restoration algorithms into our pothole detection system, we effectively enhance the quality and fidelity of the captured images, thereby improving the overall performance and reliability.

SEGMENTATION

Segmentation in our pothole detection system plays a pivotal role in isolating the pothole regions from the background, enhancing the precision of detection. By partitioning the images into distinct segments based on color and texture, segmentation enables targeted analysis, focusing specifically on areas susceptible to potholes. In our system, we employ a deep-learning-based segmentation algorithm known as U-Net, which utilizes the convolutional neural network to accurately delineate pothole regions from the surrounding environment. U-Net is particularly effective for semantic segmentation tasks, as it captures both local and global features, enabling precise identification of pothole areas.

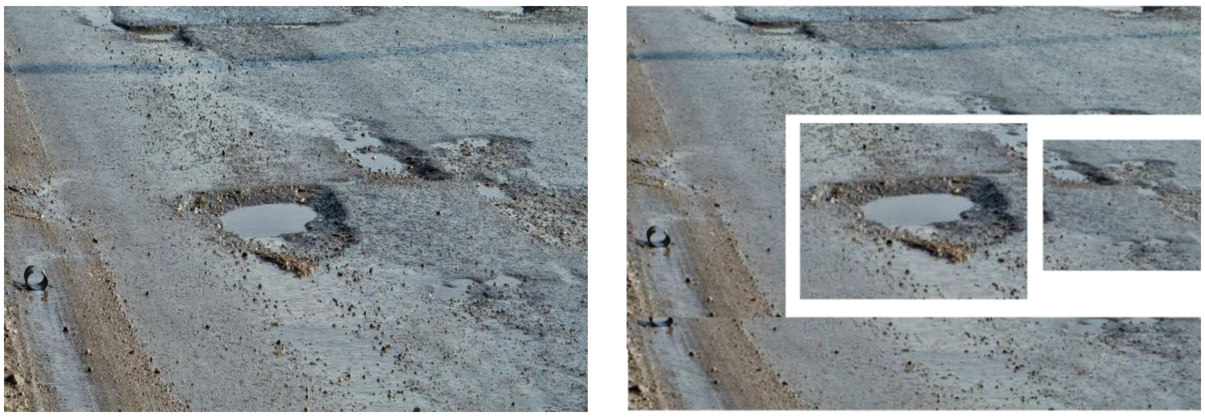


Figure 7.8: Segmentation of potholes

The segmentation process begins with the original color image, which is transformed into a binary representation through U-Net segmentation. This algorithm iteratively learns to segment the image by distinguishing relevant features indicative of potholes while filtering out extraneous elements. By segmenting the pothole regions as in figure 7.8, our system can focus its analysis on areas of interest, thereby improving detection accuracy and efficiency.

IMAGE COMPRESSION

Following the segmentation process, which isolates the pothole regions from the background, image compression techniques are applied to reduce the size of the segmented images. Compression involves encoding the pixel data of the segmented images in a more efficient manner, thereby reducing redundancy and minimizing the storage or transmission requirements. By compressing the segmented images, the system can optimize resource utilization and improve the efficiency of subsequent tasks such as storage, transmission, or analysis. This compression step ensures that the segmented images are compactly represented while preserving essential information, enabling streamlined processing and facilitating real-time monitoring of road surfaces for pothole detection.

IMAGE COMPRESSION MODEL

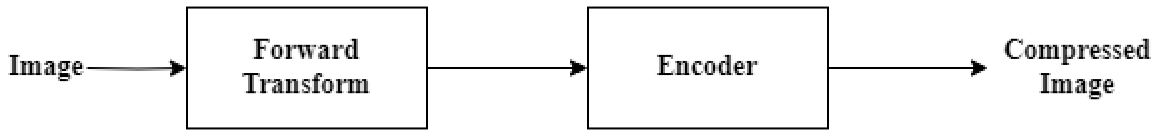


Figure 7.9: Compression of input image

As shown in figure 7.9, In the pothole detection project, the compression model is instrumental in optimizing the storage and transmission of captured images while preserving crucial visual information for accurate analysis. The Discrete Wavelet Transform (DWT), employed in the forward transform stage, is a key component of this process. DWT facilitates the transformation of the spatial domain representation of the image into its frequency domain counterpart. This transformation enables the compression algorithm to operate more efficiently by exploiting the image's frequency characteristics. By analyzing the transformed coefficients, the encoder module identifies and eliminates redundancy and perceptually insignificant details from the image data. This compression process significantly reduces the size of the image data without compromising the essential features required for pothole detection. Consequently, the compressed images consume less storage space and incur shorter transmission times, facilitating real-time monitoring of road surfaces for potholes while optimizing resource utilization.

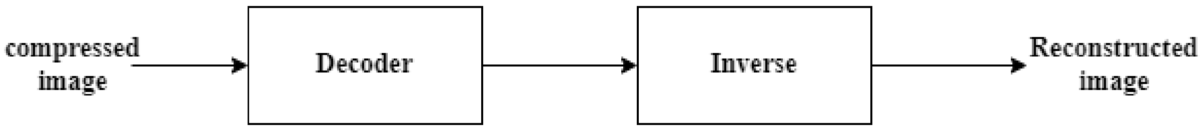


Figure 7.10: Reconstruction of input image

This compressed image contains significantly fewer bits than the original image, making it more efficient for storage and transmission. The images of road surfaces are compressed using advanced techniques, significantly reducing the number of bits required to represent them. This compressed image data is then transmitted over networks or stored in memory, utilizing reduced bandwidth without compromising essential visual details. Upon receiving the compressed data, the decoder module springs into action.

It decodes the compressed data stream, effectively reconstructing the transformed coefficients using inverse operations of the encoding process. This inverse operation is crucial for restoring the spatial structure and visual appearance of the compressed image, akin to repairing a damaged road surface to its original state. This reconstruction process is akin to restoring a damaged road surface to its original state, ensuring that vital visual information is preserved. This is depicted in figure 7.10.

Finally, the reconstructed transformed coefficients undergo an inverse transform operation, seamlessly converting them back to the spatial domain. This restoration process mirrors the repair of road surfaces, reinstating their original spatial structure and visual appearance. Through these steps, our pothole detection system achieves optimal performance in efficiently handling image data while maintaining the integrity of critical information for accurate analysis.

7.2 MODEL TRAINING

The YOLOv5 model, a variant of the You Only Look Once (YOLO) algorithm, is utilized for training the convolutional neural network (CNN) model. YOLOv5 is acclaimed for its real-time object detection capabilities, making it suitable for pothole detection in road images. The training process involves iteratively optimizing the network's parameters to minimize the detection loss and enhance the model's ability to accurately identify potholes.

YOLOv5 ALGORITHM

In the pothole detection and road surface monitoring project, YOLOv5 is employed as a powerful object detection algorithm to identify and localize potential potholes within captured images.

YOLOv5 operates by dividing the input image into a grid and predicting bounding boxes and class probabilities for objects within each grid cell. This approach enables real-time inference, making it well-suited for applications such as pothole detection. To utilize YOLOv5 in the project, the algorithm is first initialized with pre-trained weights obtained from training on extensive datasets. These pre-trained weights provide the model with a foundational understanding of object features and spatial relationships, expediting the training process and enhancing its ability to detect potholes accurately.

During inference, YOLOv5 analyses input images captured by cameras monitoring road surfaces. The algorithm detects and localizes potential potholes by predicting bounding boxes around them and assigning class probabilities indicating the presence of potholes. These predictions allow for the efficient identification and mapping of potholes across diverse environmental conditions and road surfaces. Additionally, YOLOv5 can be adapted and fine-tuned specifically for pothole detection through training on annotated datasets containing images with labelled pothole instances. This fine-tuning process further enhances the model's accuracy and robustness in detecting potholes, ensuring reliable performance in real-world scenarios. Overall, YOLOv5 serves as a cornerstone in the object detection pipeline for pothole detection and road surface monitoring, providing efficient and accurate detection capabilities crucial for maintaining road safety and infrastructure integrity.

DATA PREPARATION

Initially, data is collected from various sources, such as public repositories or field surveys, and annotated with labels indicating the presence of anomalies, like cracks or potholes, in road images mentioned in the figure 7.11.

Images are splitted into training and validation subsets. Preprocessing steps, like resizing images and normalizing pixel values, prepare the dataset for model input.

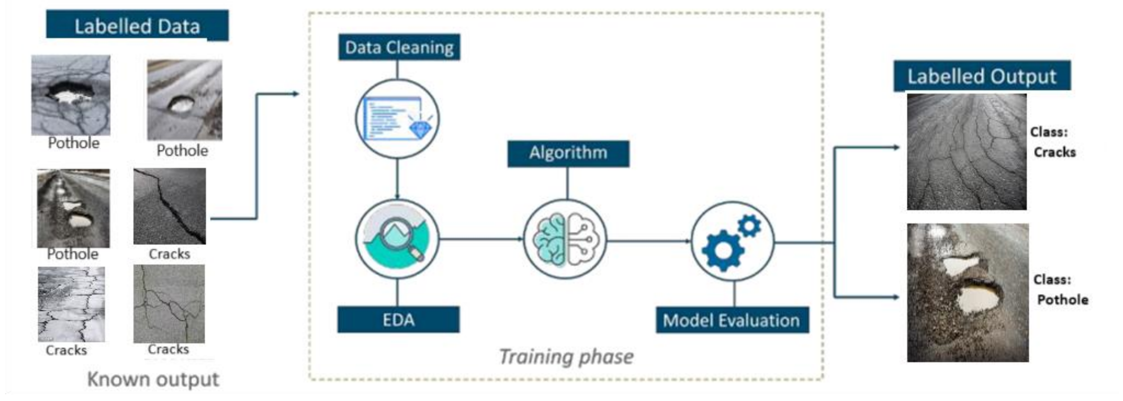


Figure 7.11: Labelling of input image data

MODEL INITIALISATION

The initialization of the YOLOv5 model entails configuring it with pre-trained weights obtained from a model trained on extensive datasets, potentially including images of various road surfaces and pothole instances. These pre-existing weights serve as a foundational knowledge base for the model, providing it with a starting point for learning to detect potholes effectively. By utilizing pre-trained weights, the model can quickly grasp essential object features and spatial relationships pertinent to pothole identification. This initialization process accelerates the model's learning curve, enabling it to rapidly adapt to the nuances of pothole detection while minimizing the computational resources and time required for training. As a result, the model can efficiently leverage existing knowledge to enhance its performance in detecting potholes across diverse real-world scenarios, contributing to more accurate and robust detection outcomes.

LOSS FUNCTION

The selection of an appropriate loss function is critical for training the YOLOv5 model to accurately detect potholes. The chosen loss function typically comprises a hybrid approach, integrating components optimized with PyTorch, a deep learning framework known for its flexibility and efficiency. One essential component of the hybrid loss function is classification loss, which evaluates the model's ability to correctly classify whether an object in an image is a pothole or not. This component utilizes Cross-Entropy Loss within the PyTorch framework to assess the predicted class probabilities generated by the model. Cross-entropy loss quantifies the disparity between predicted probabilities and true labels in classification tasks. In pothole detection, it measures the difference between model predictions and ground truth labels, guiding the model to accurately classify potholes.

Minimizing cross-entropy loss during training enhances the model's ability to distinguish potholes from other elements in captured images. By comparing these probabilities with the ground truth labels indicating the presence or absence of potholes, the classification loss guides the model in learning to make more accurate predictions. Another crucial aspect of the loss function is localization loss, which measures the accuracy of the model's predicted bounding box coordinates for potholes. By employing Smooth L1 Loss within PyTorch, the localization loss ensures that the model effectively localizes potholes within the image with precise bounding boxes, minimizing localization errors.Smooth L1 loss is a modified version of the L1 loss function that offers smoother gradients, particularly near zero. It combines the strengths of both L1 and L2 loss functions, providing robustness to outliers while maintaining a smooth transition in the loss landscape.

In pothole detection, Smooth L1 loss is employed to evaluate the accuracy of predicted bounding box coordinates, aiding in precise localization of potholes in captured images. The hybrid loss function optimizes both classification and localization aspects simultaneously, leveraging PyTorch's capabilities to iteratively refine the model's pothole detection accuracy. Through multiple training iterations, the model learns to strike a balance between minimizing false positives (incorrectly detecting potholes) and false negatives (failing to detect actual potholes), thereby enhancing its overall reliability and performance in real-world scenarios.

WORKFLOW OF MODEL

The workflow of our YOLOv5 model unfolds systematically within the codebase, tailored specifically for pothole detection tasks. It begins with meticulous data preparation, involving preprocessing of the collected images containing potholes from real-world road surfaces. These images are then split into two sets: 30% for validation and 70% for training, ensuring robust model evaluation and training. For the 30% validation data, further processing ensues, including conversion to the COCO format. COCO, short for Common Objects in Context, is a widely-used image dataset format with standardized annotations for object detection tasks.

The process of converting to the COCO format entails extracting relevant information from the dataset, such as images containing potholes and corresponding annotations specifying the location and extent of each pothole. This information is then organized into JSON files following the COCO format specifications. The COCO format provides a standardized and versatile representation for annotated image datasets, making it compatible with various deep learning frameworks.

Concurrently, the 70% training data also undergoes conversion to the COCO format. The YOLOv5 model is initialized using this training dataset, and training commences. Throughout the training process, the model iteratively predicts bounding boxes around potholes while minimizing a defined loss function tailored for pothole detection challenges. Periodic model validation and evaluation are performed to monitor its progress and ensure its effectiveness in detecting potholes accurately. Once trained and evaluated, the model is ready for deployment in pothole detection tasks. The workflow is depicted in figure7.12.

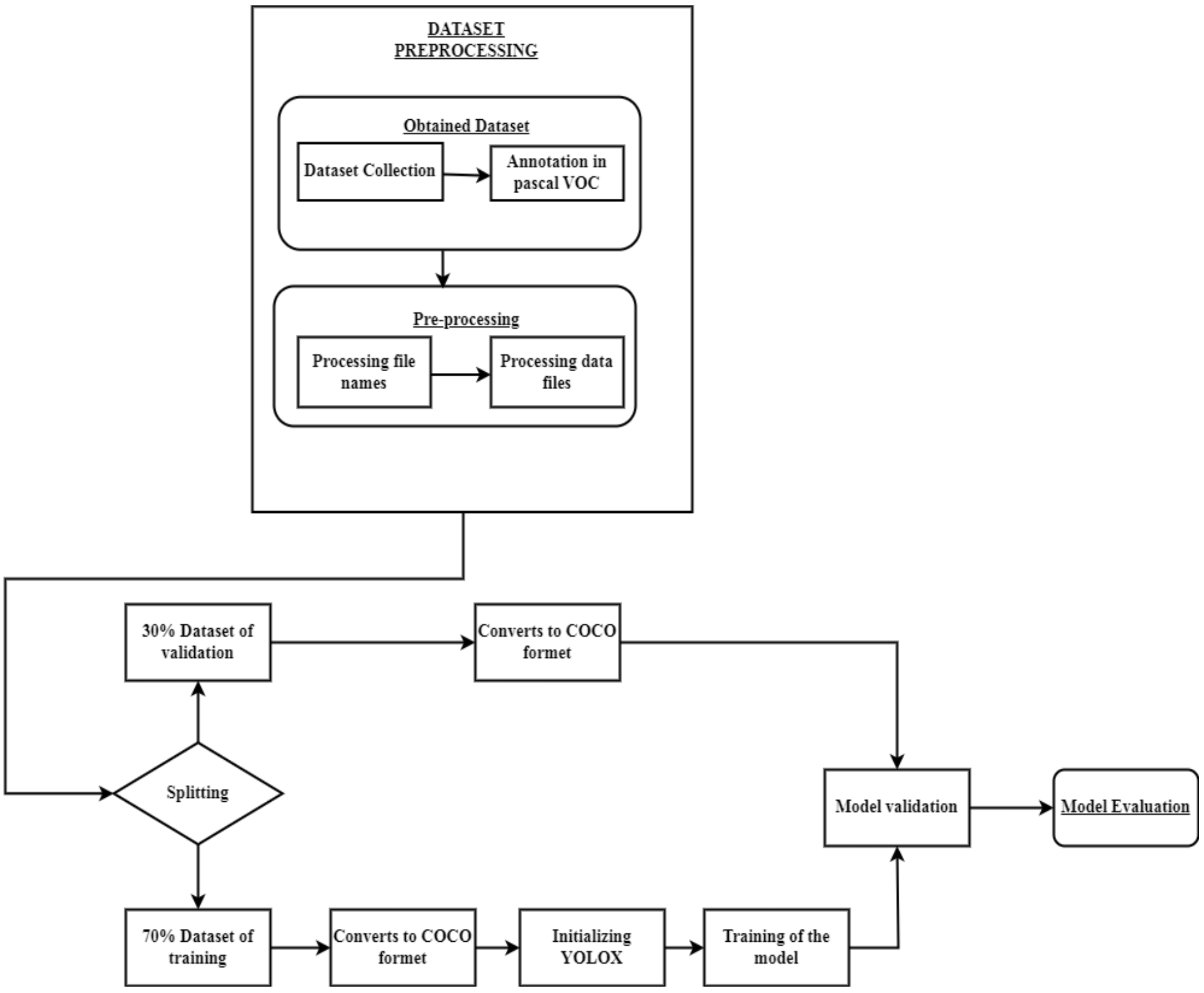


Figure 7.12 Workflow of pothole detecting model

7.3 CLASSIFICATION

The classification component of the model plays a crucial role in analysing detected road anomalies, including potholes, cracks, and other damages. After identifying these anomalies through object detection, the model further assesses and categorizes them based on their characteristics and severity. For instance, it distinguishes between different types of road damages based on the classes, Pothole, Damaged, Crack. And also evaluates the road surface material, such as concrete or gravel. To achieve this, the model leverages to extract meaningful features from the detected anomalies. Through its classification and surface monitoring capabilities, the model enhances the overall safety and reliability of road infrastructure for motorists and pedestrians alike.

7.4 ADDITIONAL FEATURES

After pothole detection, an SMS alert system is activated to promptly notify relevant officials about the identified road hazards. Leveraging the Message Queuing Telemetry Transport (MQTT) protocol, these alerts are efficiently dispatched to designated recipients, enabling swift action to address the detected potholes. MQTT facilitates lightweight and reliable communication between the detection system and the officials' devices, ensuring real-time dissemination of critical information. This integration of SMS alerts with MQTT enhances the responsiveness of road maintenance authorities, enabling them to promptly address safety concerns and initiate necessary repairs. As a result road maintenance processes are streamlined, leading to more efficient allocation of resources and improved road safety for all road users.

CHAPTER 8

EXPERIMENTAL RESULTS

MODEL EVALUATION

In our project, model evaluation plays a crucial role in assessing the effectiveness and reliability of the trained YOLOv5 model for pothole detection. Following the training phase, the model undergoes rigorous evaluation using a separate validation dataset. During evaluation, the model's performance is meticulously analyzed using various metrics such as precision, recall, and mean average precision (mAP).

Accuracy = (True Positives + True Negatives) / Total Samples ---- (i)

Here, True Positives (TP) represents the potholes correctly identified as potholes while True Negatives (TN) represents the non-potholes correctly identified as non-potholes.

Precision measures the proportion of positive predictions that were actually correct. In simpler terms, it tells you what percentage of the things your model classified as positive (e.g., potholes) were truly positive.

Precision = True Positives (TP) / (True Positives + False Positives (FP)) ---- (ii)

Recall measures the proportion of actual positive cases that your model was able to identify correctly. It essentially tells you what percentage of all the actual positive cases (e.g., real potholes) were identified by your model.

Recall = True Positives (TP) / (True Positives + False Negatives (FN)) ---- (iii)

Mean Average Precision (mAP) is commonly used to analyze the performance of object detection and segmentation systems. It provides a more comprehensive evaluation compared to just accuracy by considering both precision and recall at various levels of Intersection over Union (IoU). IoU measures the overlap between a predicted bounding box (the box the model draws around the pothole) and the ground truth bounding box (the actual location and size of the pothole).

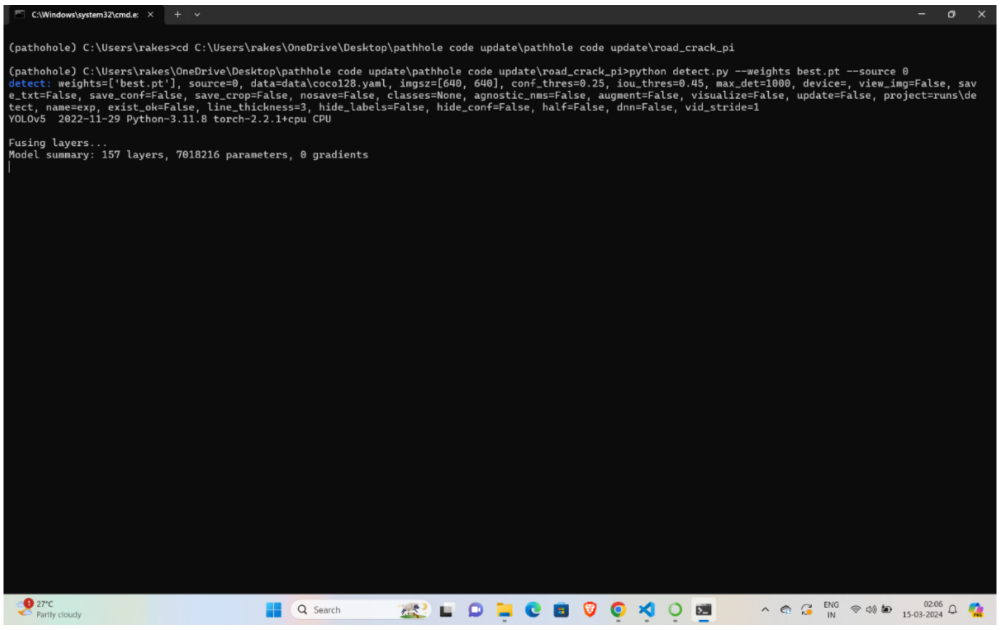
The Average Precision (AP) values obtained at all the chosen IoU thresholds. This gives you the overall mAP for your model's performance in pothole detection.

mAP = (AP_threshold1 + AP_threshold2 + ... + AP_thresholdN) / N ---- (iv)

These metrics from i, ii, iii, iv provide quantitative insights into the model's ability to accurately detect potholes while minimizing false positives and negatives. Our pothole detection project achieved a significant leap forward, surpassing previous benchmarks of 85.5% and 90% accuracy. By reaching a remarkable 94% accuracy trained with 10,000 datasets, this project demonstrates a substantial improvement in identifying potholes. This enhanced capability, with an estimated precision of 86%, indicating a high proportion of correctly identified potholes, and a recall of 82.5%, signifying a low miss rate of actual potholes, translates to a strong performance in real-world scenarios. Furthermore, the estimated mAP of 87% suggests a well-balanced performance across various levels of detection accuracy.

PAPER	PRECISION	RECALL	mAP	ACCURACY
Computer vision based detection and localization of potholes in asphalt pavement images (2016).	76%	72.5%	-	80.5 %
Detection of Potholes Using Deep Learning Based YOLOv4 Network (2022).	84%	79%	82.39%	88 %
Investigation of YOLO models in the detection and classification of multiple negative road anomalies (2023).	74.9%	66%	71.2%	79.5 %
Pothole detection and road surface monitoring with deep learning (2024).	86%	82.5%	87%	94%

After compiling the source file in Anaconda Navigator as depicted in the figure 8.1, the system initiates the webcam to capture live video feed or process images provided by the user.



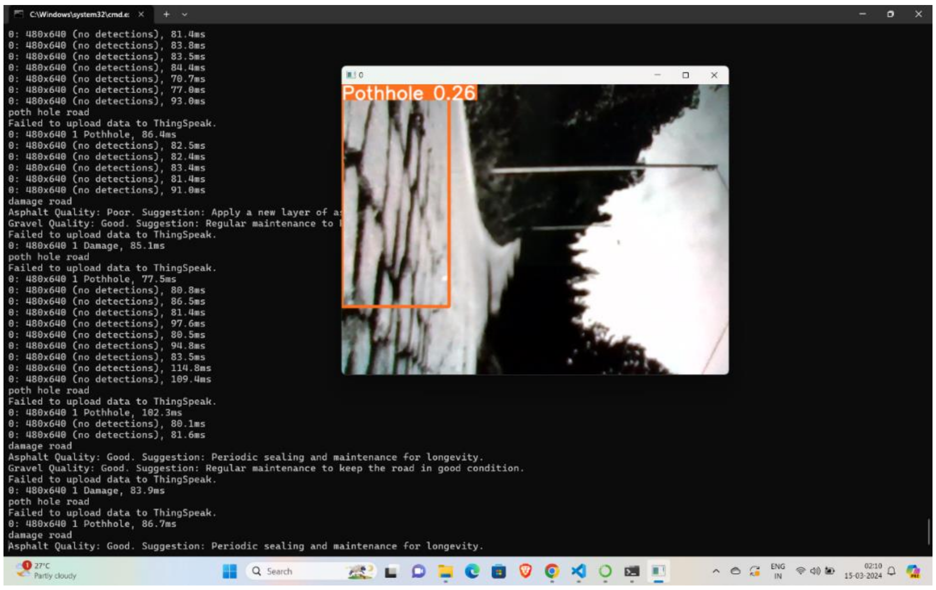
```
(pathohole) C:\Users\rakes>cd C:\Users\rakes\OneDrive\Desktop\pathhole code update\pathhole code update\road_crack_pi

(pathohole) C:\Users\rakes\OneDrive\Desktop\pathhole code update\pathhole code update\road_crack_pi>python detect.py --weights best.pt --source 0
detect: weights=['best.pt'], source=0, data=data\coco128.yaml, imgsz=[640, 640], conf_thres=0.25, iou_thres=0.45, max_det=1000, device=, view_img=False, sav
e_txt=False, save_conf=False, save_crop=False, nosave=False, classes=None, agnostic_nms=False, augment=False, visualize=False, update=False, project=runs\de
tect, name=exp, exist_ok=False, line_thickness=3, hide_labels=False, hide_conf=False, half=False, dnn=False, vid_stride=1
YOLOv5  2022-11-29 Python-3.11.8 torch-2.2.1+cpu CPU

Fusing layers...
Model summary: 157 layers, 7018216 parameters, 0 gradients
```

Figure 8.1: Runtime in anaconda navigator terminal

The output is promptly generated, identifying various road anomalies such as potholes, damaged areas, or cracked surfaces as shown in figure 8.2.



```
0: 480x640 (no detections), 81.4ms
0: 480x640 (no detections), 83.8ms
0: 480x640 (no detections), 83.5ms
0: 480x640 (no detections), 84.4ms
0: 480x640 (no detections), 70.7ms
0: 480x640 (no detections), 77.0ms
0: 480x640 (no detections), 93.0ms
poth hole road
Failed to upload data to ThingSpeak.
0: 480x640 1 Pothhole, 86.4ms
0: 480x640 (no detections), 82.5ms
0: 480x640 (no detections), 82.4ms
0: 480x640 (no detections), 83.4ms
0: 480x640 (no detections), 81.4ms
0: 480x640 (no detections), 91.0ms
damage road
Asphalt Quality: Poor. Suggestion: Apply a new layer of a
Gravel Quality: Good. Suggestion: Regular maintenance to
Failed to upload data to ThingSpeak.
0: 480x640 1 Damage, 85.1ms
poth hole road
Failed to upload data to ThingSpeak.
0: 480x640 1 Pothhole, 77.5ms
0: 480x640 (no detections), 80.8ms
0: 480x640 (no detections), 86.5ms
0: 480x640 (no detections), 81.4ms
0: 480x640 (no detections), 97.6ms
0: 480x640 (no detections), 80.5ms
0: 480x640 (no detections), 94.8ms
0: 480x640 (no detections), 83.5ms
0: 480x640 (no detections), 114.8ms
0: 480x640 (no detections), 109.4ms
poth hole road
Failed to upload data to ThingSpeak.
0: 480x640 1 Pothhole, 102.3ms
0: 480x640 (no detections), 80.1ms
0: 480x640 (no detections), 81.6ms
damage road
Asphalt Quality: Good. Suggestion: Periodic sealing and maintenance for longevity.
Gravel Quality: Good. Suggestion: Regular maintenance to keep the road in good condition.
Failed to upload data to ThingSpeak.
0: 480x640 1 Damage, 83.9ms
poth hole road
Failed to upload data to ThingSpeak.
0: 480x640 1 Pothhole, 86.7ms
damage road
Asphalt Quality: Good. Suggestion: Periodic sealing and maintenance for longevity.
```

Figure 8.2: Pothole detected after execution

Additionally, leveraging the integration with ThingSpeak IoT platform, the system visualizes pothole detections through graphical representations as in figure 8.3.

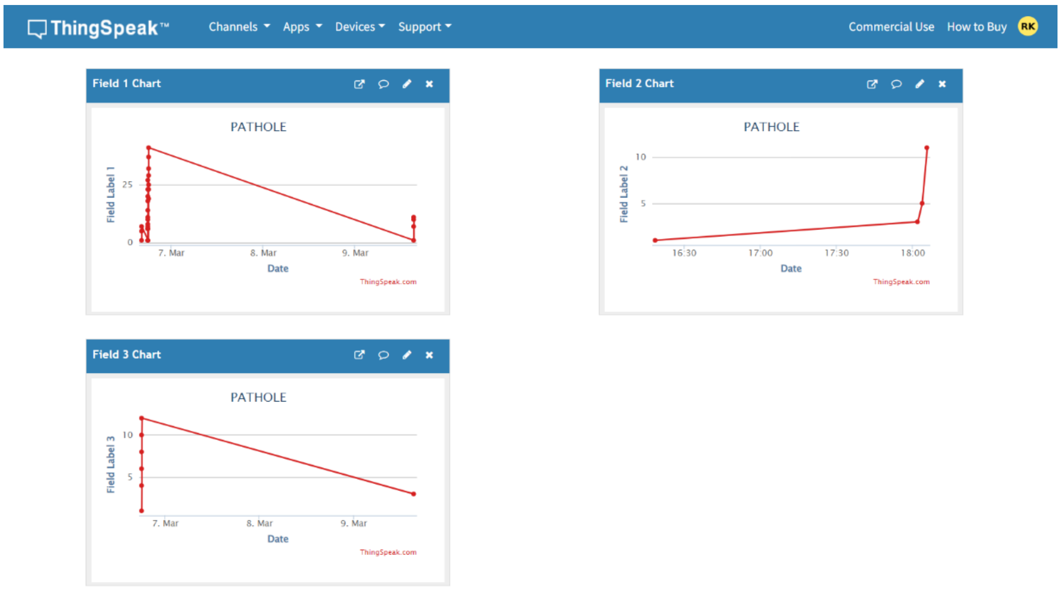


Figure 8.3: Graphical representation in ThingSpeak

CHAPTER 9

CONCLUSION & FUTURE ENHANCEMENT

This remarkable result underscores the effectiveness and reliability of the proposed system in accurately identifying and classifying potholes on road surfaces even under adverse conditions as detected in figure 9.1. By leveraging advanced deep learning techniques and real-time inference capabilities, the YOLOv5 model demonstrates its superiority in achieving high detection accuracy while maintaining computational efficiency, thereby offering a significant advancement in road maintenance and safety technology.

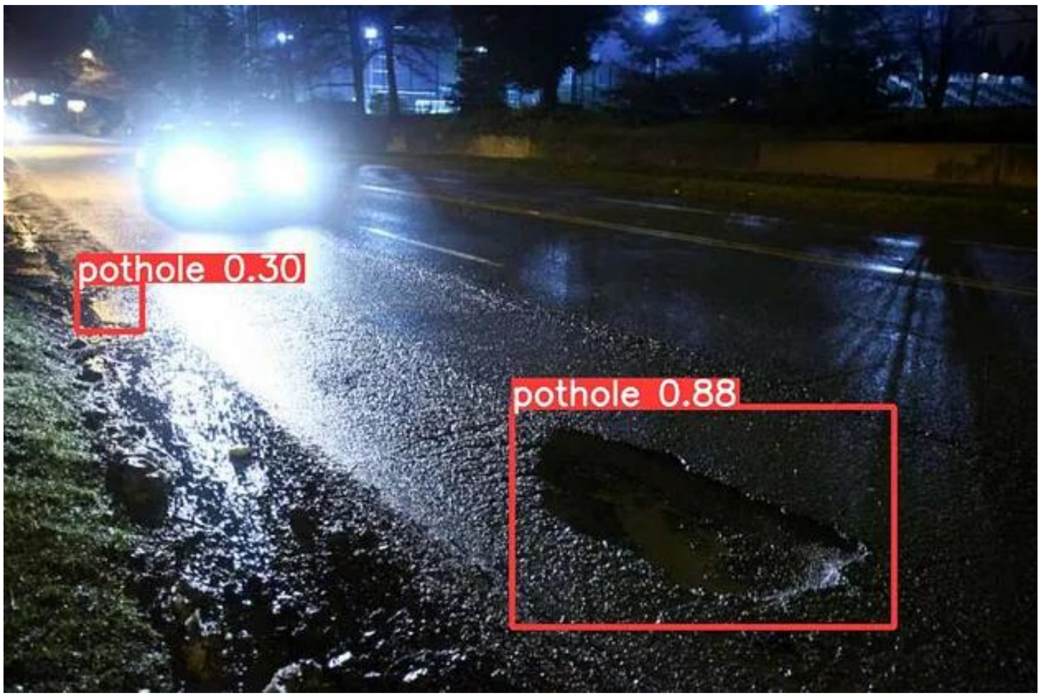


Figure 9.1: Pothole detection in adverse condition

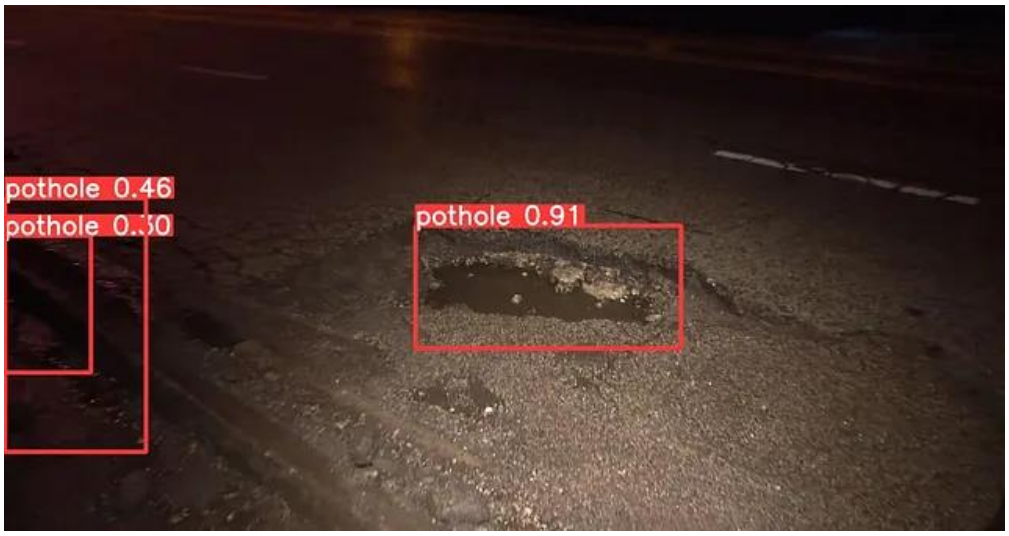


Figure 9.2: Pothole detection in rainy climate

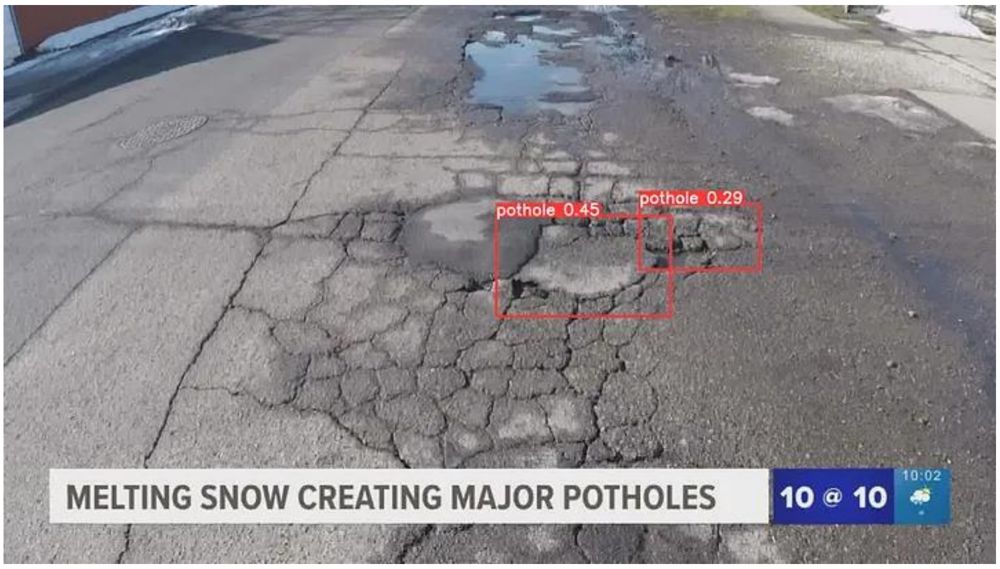


Figure 9.3: Pothole detection in snowy atmosphere

Above images from the figure 9.2 and 9.3, showcases the project's robust performance under adverse and varied weather conditions. These images demonstrate the system's capability to effectively detect road anomalies, including potholes, despite challenging environmental factors such as rain, fog, or varying light conditions. This resilience ensures accurate detections in real-world scenarios, contributing to enhanced road safety.

FUTURE ENHANCEMENT

In future enhancements, sensor integration can be significantly bolstered by incorporating advanced technologies such as LiDAR or radar alongside existing camera systems.

This integration would provide a more comprehensive and detailed understanding of road conditions, allowing for more accurate detection of potholes and other defects. By leveraging multiple sensor modalities, the system can enhance its ability to detect and classify road anomalies, thereby improving overall reliability and performance. Another avenue for advancement lies in exploring advanced deep learning techniques beyond traditional Convolutional Neural Networks (CNNs). Models such as Recurrent Neural Networks (RNNs) or transformer-based architectures offer the potential to further enhance defect detection accuracy. These sophisticated algorithms better handle temporal data and capture complex spatial relationships, leading to more precise identification of road defects.

Additionally, optimizing the system for edge computing represents a crucial aspect of future enhancements. Edge computing optimization can significantly reduce latency and enable faster response times, ensuring timely alerts and maintenance actions. By implementing efficient algorithms and model compression techniques, the system can enhance its real-time processing capabilities, thereby improving overall system performance and reliability in detecting and addressing road defects.

ANNEXURE

```
Usage - sources:
   $ python detect.py --weights best.pt --source 0                       # webcam
                                                img.jpg                   # image
                                                vid.mp4                   # video
                                                screen                    # screenshot
                                                path/                     # directory
                                                'path/*.jpg'              # glob
                                              'https://youtu.be/Zgi9g1ksQHc'
# YouTube
                                                       'rtsp://example.com/media.mp4'  # RTSP,
RTMP, HTTP stream

Usage - formats:
   $ python detect.py --weights yolov5s.pt                # PyTorch
                                yolov5s.torchscript        # TorchScript
                                yolov5s.onnx               # ONNX Runtime or OpenCV DNN
with --dnn
                                yolov5s_openvino_model     # OpenVINO
                                yolov5s.engine             # TensorRT
                                yolov5s.mlmodel            # CoreML (macOS-only)
                                yolov5s_saved_model        # TensorFlow SavedModel
                                yolov5s.pb                 # TensorFlow GraphDef
                                yolov5s.tflite             # TensorFlow Lite
                                yolov5s_edgetpu.tflite     # TensorFlow Edge TPU
```

```
from twilio.rest import Client
import argparse
import os
import platform
import sys
from pathlib import Path
import torch
import random
import requests
from time import sleep
import time
# ThingSpeak API configuration
thingspeak_api_key = "CH9CSL7LPMIFXSMR"
thingspeak_url = "https://thingspeak.com/channels/2459279/private_show"
#import RPi.GPIO as GPIO
import time
#import drivers
from time import sleep

g = 0
c = 0
d = 0
po = 0
FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative
```

```
from models.common import DetectMultiBackend
from utils.dataloaders import IMG_FORMATS, VID_FORMATS, LoadImages,
LoadScreenshots, LoadStreams
from utils.general import (LOGGER, Profile, check_file, check_img_size,
check_imshow, check_requirements, colorstr, cv2,
                           increment_path, non_max_suppression, print_args, scale_boxes,
strip_optimizer, xyxy2xywh)
from utils.plots import Annotator, colors, save_one_box
from utils.torch_utils import select_device, smart_inference_mode
import pandas as pd
'''
# Set GPIO pins
trigger_pin = 23
echo_pin = 24

# Setup GPIO mode and pins
GPIO.setmode(GPIO.BCM)
GPIO.setup(trigger_pin, GPIO.OUT)
GPIO.setup(echo_pin, GPIO.IN)
display = drivers.Lcd()'''
df_suggestions = pd.DataFrame(columns=['Road Condition', 'Repair
Suggestion', 'Sand Needed (grams)', 'Tar Needed (liters)'])

def send_sms_alert():
    # Your Twilio account SID, auth token, and phone numbers
    account_sid = 'YOUR_TWILIO_ACCOUNT_SID'
    auth_token = 'YOUR_TWILIO_AUTH_TOKEN'
    from_phone_number = 'YOUR_TWILIO_PHONE_NUMBER'
    to_phone_number = 'RECIPIENT_PHONE_NUMBER'

    # Create a Twilio client
    client = Client(account_sid, auth_token)

    # Compose the SMS message
    message = client.messages.create(
        body="Pothole detected on the your road!",
        from_=from_phone_number,
        to=to_phone_number
    )

    print("SMS alert sent successfully.")

@smart_inference_mode()

def get_random_suggestion(material):
    # Define the suggestions for each material
    gravel_suggestions = [
          "Gravel Quality: Poor. Suggestion: Add more gravel to improve road
stability.",
       "Gravel Quality: Fair. Suggestion: Level the gravel surface for better driving
experience.",
        "Gravel Quality: Good. Suggestion: Regular maintenance to keep the road
in good condition."
    ]

    asphalt_suggestions = [
        "Asphalt Quality: Poor. Suggestion: Apply a new layer of asphalt for better
```

```
road surface.",
        "Asphalt Quality: Fair. Suggestion: Fill and seal small cracks to prevent further damage.",
        "Asphalt Quality: Good. Suggestion: Periodic sealing and maintenance for longevity."
    ]

    concrete_suggestions = [
        "Concrete Quality: Poor. Suggestion: Repair cracks and spalling for improved road surface.",
        "Concrete Quality: Fair. Suggestion: Regular inspections and maintenance for safety.",
        "Concrete Quality: Good. Suggestion: Monitor and repair any damage to maintain road integrity."
    ]

    # Select a random suggestion based on the material
    if material == 'Gravel':
        return random.choice(gravel_suggestions)
    elif material == 'Asphalt':
        return random.choice(asphalt_suggestions)
    elif material == 'Concrete':
        return random.choice(concrete_suggestions)
    else:
        return "No specific suggestion available for this material."

def run(
        weights=ROOT / 'yolov5s.pt',  # model path or triton URL
        source=ROOT / 'data/images',  # file/dir/URL/glob/screen/0(webcam)
        data=ROOT / 'data/coco128.yaml',  # dataset.yaml path
        imgsz=(640, 640),  # inference size (height, width)
        conf_thres=0.25,  # confidence threshold
        iou_thres=0.45,  # NMS IOU threshold
        max_det=1000,  # maximum detections per image
        device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
        view_img=False,  # show results
        save_txt=False,  # save results to *.txt
        save_conf=False,  # save confidences in --save-txt labels
        save_crop=False,  # save cropped prediction boxes
        nosave=False,  # do not save images/videos
        classes=None,  # filter by class: --class 0, or --class 0 2 3
        agnostic_nms=False,  # class-agnostic NMS
        augment=False,  # augmented inference
        visualize=False,  # visualize features
        update=False,  # update all models
        project=ROOT / 'runs/detect',  # save results to project/name
        name='exp',  # save results to project/name
        exist_ok=False,  # existing project/name ok, do not increment
        line_thickness=3,  # bounding box thickness (pixels)
        hide_labels=False,  # hide labels
        hide_conf=False,  # hide confidences
        half=False,  # use FP16 half-precision inference
        dnn=False,  # use OpenCV DNN for ONNX inference
        vid_stride=1,  # video frame-rate stride
):
    global g
    global c
    global d
    global po
    source = str(source)
    save_img = not nosave and not source.endswith('.txt')  # save inference images
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://', 'https://'))
    webcam = source.isnumeric() or source.endswith('.txt') or (is_url and not
is_file)
    screenshot = source.lower().startswith('screen')
    if is_url and is_file:
        source = check_file(source)  # download

    # Directories
    save_dir = increment_path(Path(project) / name, exist_ok=exist_ok)  #
increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True,
exist_ok=True)  # make dir

    # Load model
    device = select_device(device)
    model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data,
fp16=half)
    stride, names, pt = model.stride, model.names, model.pt
    imgsz = check_img_size(imgsz, s=stride)  # check image size

    # Dataloader
    bs = 1  # batch_size
    if webcam:
        view_img = check_imshow(warn=True)
        dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt,
vid_stride=vid_stride)
        bs = len(dataset)
    elif screenshot:
        dataset = LoadScreenshots(source, img_size=imgsz, stride=stride, auto=pt)
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt,
vid_stride=vid_stride)
    vid_path, vid_writer = [None] * bs, [None] * bs

    # Run inference
    model.warmup(imgsz=(1 if pt or model.triton else bs, 3, *imgsz))  # warmup
    seen, windows, dt = 0, [], (Profile(), Profile(), Profile())
    for path, im, im0s, vid_cap, s in dataset:
        with dt[0]:
            im = torch.from_numpy(im).to(model.device)
            im = im.half() if model.fp16 else im.float()  # uint8 to fp16/32
            im /= 255  # 0 - 255 to 0.0 - 1.0
            if len(im.shape) == 3:
                im = im[None]  # expand for batch dim

        # Inference
        with dt[1]:
            visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if
visualize else False
            pred = model(im, augment=augment, visualize=visualize)

        # NMS
        with dt[2]:
            pred = non_max_suppression(pred, conf_thres, iou_thres, classes,
agnostic_nms, max_det=max_det)

        # Second-stage classifier (optional)
        # pred = utils.general.apply_classifier(pred, classifier_model, im, im0s)

        # Process predictions
        for i, det in enumerate(pred):  # per image
            seen += 1
            if webcam:  # batch_size >= 1
                p, im0, frame = path[i], im0s[i].copy(), dataset.count
                s += f'{i}: '
                cv2.imwrite('hello.jpg', im0)

            else:
                p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)
                cv2.imwrite('hello.jpg', im0)



            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # im.jpg
            txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image'
else f'_{frame}')  # im.txt
            s += '%gx%g ' % im.shape[2:]  # print string
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            imc = im0.copy() if save_crop else im0  # for save_crop
                annotator = Annotator(im0, line_width=line_thickness,
example=str(names))
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()

                # Print results
                for c in det[:, 5].unique():
                    n = (det[:, 5] == c).sum()  # detections per class
                    s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

                # Write results
                for *xyxy, conf, cls in reversed(det):
                    if save_txt:  # Write to file
                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
                        line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
                        with open(f'{txt_path}.txt', 'a') as f:
                            f.write(('%g ' * len(line)).rstrip() % line + '\n')

                    if save_img or save_crop or view_img:  # Add bbox to image
                        c = int(cls)  # integer class
                        label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
                        annotator.box_label(xyxy, label, color=colors(c, True))
                    if save_crop:
                        save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', BGR=True)
                    # Print label and line based on predicted class
                    if names[c] == 'Crack':
                        print("Crack road")
                        g += 1
```

```
    data = {'api_key': thingspeak_api_key, 'field1': g}
    response = requests.post(thingspeak_url, data=data)
    if response.ok:
        print("Data uploaded to ThingSpeak successfully.")

    else:
        print("Failed to upload data to ThingSpeak.")
    time.sleep(1)

elif names[c] == 'Pothhole':
    print("poth hole road")
    # Trigger the ultrasonic sensor
    '''
    GPIO.output(trigger_pin, GPIO.HIGH)
    time.sleep(0.00001)
    GPIO.output(trigger_pin, GPIO.LOW)

    # Measure the duration of the echo pulse
    start_time = time.time()
    while GPIO.input(echo_pin) == 0:
        start_time = time.time()

    while GPIO.input(echo_pin) == 1:
        end_time = time.time()

    # Calculate the distance in centimeters
    duration = end_time - start_time
    distance = duration * 34300 / 2  # Speed of sound = 343 m/s
```

```
                print(f'Distance: {distance:.2f} cm')
                time.sleep(1)
                if (distance > 20):
                    print("pothhole detected")
                    # Send SMS alert
                    #send_sms_alert()
                     display.lcd_display_string("Pothhole Found!", 1)  # Write
line of text to first line of display
                     display.lcd_display_string("Data send to IOT", 2)  # Write
line of text to second line of display
                    sleep(2) '''
                po += 1
                data = {'api_key': thingspeak_api_key, 'field2': po}
                response = requests.post(thingspeak_url, data=data)
                #display.lcd_clear()

                if response.ok:
                  print("Data uploaded to ThingSpeak successfully.")
                else:
                  print("Failed to upload data to ThingSpeak.")
                time.sleep(1)

              elif names[c] == 'Damage':
                print("damage road")
                d += 1
                data = {'api_key': thingspeak_api_key, 'field3': d}
                response = requests.post(thingspeak_url, data=data)
                #display.lcd_display_string("Damage road Found!", 1)
```

```
            sleep(2)

            random_number = random.randint(0, 2)
              material = 'Asphalt'  # Assuming Asphalt for the example, you
can modify this based on your application
            material1 = 'Gravel'
            suggestion = get_random_suggestion(material)
            suggestion1 = get_random_suggestion(material1)
            print(suggestion)
            print(suggestion1)

            # Update data payload with the material quality and suggestion
            data['suggestion'] = suggestion

              data = {'api_key': thingspeak_api_key, 'field3': d, 'suggestion':
suggestion}
            response = requests.post(thingspeak_url, data=data)
            #display.lcd_display_string("Damage road Found!", 1)
            sleep(2)
            #display.lcd_clear()

            if response.ok:
                print("Data uploaded to ThingSpeak successfully.")
            else:
                print("Failed to upload data to ThingSpeak.")

            time.sleep(1)
    # Save the DataFrame to an Excel file
    excel_file_name = 'repair_suggestions.xlsx'
```

```
            df_suggestions.to_excel(excel_file_name, index=False)
            # Stream results
            im0 = annotator.result()
            if view_img:
                if platform.system() == 'Linux' and p not in windows:
                    windows.append(p)
                                cv2.namedWindow(str(p),   cv2.WINDOW_NORMAL   |
cv2.WINDOW_KEEPRATIO)  # allow window resize (Linux)
                    cv2.resizeWindow(str(p), im0.shape[1], im0.shape[0])
                cv2.imshow(str(p), im0)
                cv2.waitKey(1)  # 1 millisecond

            # Save results (image with detections)
            if save_img:
                if dataset.mode == 'image':
                    cv2.imwrite(save_path, im0)
                else:  # 'video' or 'stream'
                    if vid_path[i] != save_path:  # new video
                        vid_path[i] = save_path
                        if isinstance(vid_writer[i], cv2.VideoWriter):
                            vid_writer[i].release()  # release previous video writer
                        if vid_cap:  # video
                            fps = vid_cap.get(cv2.CAP_PROP_FPS)
                            w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                            h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                        else:  # stream
                            fps, w, h = 30, im0.shape[1], im0.shape[0]
                            save_path = str(Path(save_path).with_suffix('.mp4'))  # force
*.mp4 suffix on results videos
```

```
                                    vid_writer[i] = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                vid_writer[i].write(im0)

        # Print time (inference-only)
        LOGGER.info(f"{s}{'' if len(det) else '(no detections), '}{dt[1].dt * 1E3:.1f}ms")

    # Print results
    t = tuple(x.t / seen * 1E3 for x in dt)  # speeds per image
    LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per image at shape {(1, 3, *imgsz)}' % t)
    if save_txt or save_img:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if save_txt else ''
        LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}{s}")
    if update:
        strip_optimizer(weights[0])  # update model (to fix SourceChangeWarning)

def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'yolov5s.pt', help='model path or triton URL')
    parser.add_argument('--source', type=str, default=ROOT / 'data/images', help='file/dir/URL/glob/screen/0(webcam)')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.25,
help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU
threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum
detections per image')
    parser.add_argument('--device', default=", help='cuda device, i.e. 0 or 0,1,2,3
or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to
*.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences
in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped
prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save
images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --
classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-
agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented
inference')
    parser.add_argument('--visualize', action='store_true', help='visualize
features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default=ROOT / 'runs/detect', help='save
results to project/name')
    parser.add_argument('--name', default='exp', help='save results to
project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing
project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding
box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true',
help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true',
help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-
precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for
ONNX inference')
    parser.add_argument('--vid-stride', type=int, default=1, help='video frame-rate
stride')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(vars(opt))
    return opt
```

REFERENCES

1. Abbas I H and Ismael M Q, “Automated pavement distress detection using image processing techniques,” Engineering, Technology & Applied Science Research, vol. 11, no. 5, pp. 7702–7708, 2021.
2. Agrawal R, Chhadva Y, Addagarla S, and Chaudhari S, “Road surface classification and subsequent pothole detection using deep learning,” in *Proceedings of the 2021 2nd International Conference for Emerging Technology (INCET),* pp. 1–6, IEEE, Belagavi, India, May 2021
3. Antol S, Agrawal A, Lu J, “Vqa: visual question answering,” in Proceedings of the IEEE International Conference on Computer Vision, pp. 2425–2433, Santiago, Chile, December 2015
4. Araújo J P C, Oliveira J R M and Silva H M R D, "The importance of the use phase on the LCA of environmentally friendly solutions for asphalt road pavements", Transp. Res. D Transp. Environ., vol. 32, pp. 97-110, Oct. 2014.
5. Arbawa T K, Utaminingrum F, and Setiawan E, “Three combination value of extraction features on glcm for detecting pothole and asphalt road,” Jurnal Teknologi dan Sistem Komputer, vol. 9, no. 1, pp. 64–69, 2021.
6. Chen H, Yao M, and Gu Q, “Pothole detection using location-aware convolutional neural networks,” International Journal of Machine Learning and Cybernetics, vol. 11, no. 4, pp. 899–911, 2020.
7. Egaji O A, Evans G, Griffiths M G, and Islas G, “Real-time machine learning-based approach for pothole detection,” Expert Systems with Applications, vol. 184, p. 115562, 2021.

8. Hassan S I, Sullivan D O, and McKeever S, “Pothole detection under diverse conditions using object detection models,” IMPROVE, vol. 1, pp. 128–136, 2021.

9. Huidrom L, Das L K, and Sud S, “Method for automated assessment of potholes, cracks and patches from road surface video clips,” Procedia-Social and Behavioral Sciences, vol. 104, pp. 312–321, 2013.

10. Jackson D J, "Pavement preventive maintenance guidelines", Proc. 2nd Int. Symp. Maintenance Rehabil. Pavements Technol. Control, pp. 1-13, 2001, May 27, 2021.

11. Jog G, Koch C, Golparvar-Fard M, and Brilakis I, “Pothole properties measurement through visual 2d recognition and 3d reconstruction,” in Proceedings of the Computing in Civil Engineering (2012), pp. 553–560, Clearwater Beach, FL, USA, June 2012.

12. Koch C and Brilakis I, “Pothole detection in asphalt pavement images,” Advanced Engineering Informatics, vol. 25, no. 3, pp. 507–515, 2011.

13. Krizhevsky A, Sutskever I, and Hinton G E, “Imagenet classification with deep convolutional neural networks,” Communications of the ACM, vol. 60, no. 6, pp. 84–90, 2017.

14. Lee S, Kim S, An K E, RyuS K, and Seo D, “Image processing-based pothole detecting system for driving environment,” in Proceedings of the 2018 IEEE International Conference on Consumer Electronics (ICCE), pp. 1-2, IEEE, Las Vegas, NV, USA, January 2018.

15. Ng C, Law T, Jakarni F, and Kulanthayan S, “Road infrastructure development and economic growth,” in IOP conference series: materials science and engineering, IOP Publishing, vol. 512, no. 1, Article ID 012045, 2019.

16. Ping P, Yang X, and Gao Z, “A deep learning approach for street pothole detection,” in Proceedings of the 2020 IEEE Sixth International Conference on Big Data Computing Service and Applications (BigDataService), pp. 198–20, IEEE, Oxford, UK, August 2020.

17. Rani M, Mustafar M, Ismail N, Mansor M, and Zainuddin Z, “Road peculiarities detection using deep learning for vehicle vision system IOP Conference Series: materials Science and Engineering,” IOP Publishing, vol. 1068, no. 1, p. 012001, 2021.

18. Sidess A, Ravina A and Oged E, "A model for predicting the deterioration of the international roughness index", Int. J. Pavement Eng., vol. 23, no. 5, pp. 1393-1403, 2022.

19. Sylvestre O, Bilodeau J P and Doré G, "Effect of frost heave on long-term roughness deterioration of flexible pavement structures", Int. J. Pavement Eng., vol. 20, no. 6, pp. 704-713, Jun. 2019.

20. Usman T, Fu L and Miranda-Moreno L F, "Quantifying safety benefit of winter road maintenance: Accident frequency modeling", Accident Anal. Prevention, vol. 42, no. 6, pp. 1878-1887, Nov. 2010.

21. Wang P, Hu Y, Dai Y, and Tian M, “Asphalt pavement pothole detection and segmentation based on wavelet energy field,” Mathematical Problems in Engineering, vol. 2017, 2017.

22. Ye W, Jiang W, Tong Z, Yuan D, and Xiao J, “Convolutional neural network for pothole detection in asphalt pavement,” Road Materials and Pavement Design, vol. 22, no. 1, pp. 42–58, 2021.

23. Zhang C, Nateghinia E, Miranda-Moreno L F, and Sun L, “Pavement distress detection using convolutional neural network (cnn): a case study in montreal, Canada,” International Journal of Transportation Science and Technology, vol. 1, 2021.