

A Project Work Phase II Report on

Pothole Detection using Deep Learning

Submitted in partial fulfillment of the requirements for the award of the

Bachelor of Technology

in

CSE (Data Science)

by

C. Sandeep Kumar	21241A6717
P. Chenna Kesava	21241A6746
V. Naveen	22245A6706

Under the Esteemed guidance of

Mr. A. Venu Gopal Rao

Assistant Professor



Department of Data Science

**GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND
TECHNOLOGY**

(Approved by AICTE, Autonomous under JNTUH, Hyderabad)

Bachupally, Kukatpally, Hyderabad-500090

2024-2025



**GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND
TECHNOLOGY
(Autonomous)**

Hyderabad-500090

CERTIFICATE

This is to certify that the Project Work - Phase II entitled “**Pothole Detection using Deep Learning**” is submitted by **C. Sandeep Kumar (21241A6717)**, **P. Chenna Kesava (21241A6746)** and **V. Naveen (22245A6706)** in partial fulfillment of the award of degree in **Bachelor of Technology in Computer Science and Engineering (Data Science)** during Academic year 2024-2025.

Internal Guide

Mr. A. Venu Gopal Rao

Head of the Department

Dr. S. Govinda Rao

External Examiner

ACKNOWLEDGEMENT

There are many people who helped us directly and indirectly to complete our project successfully. We would like to take this opportunity to thank one and all. First, we would like to express our deep gratitude towards our internal guide **Mr. A. Venu Gopal Rao, Assistant Professor**, Department of Data Science, for his support in the completion of our dissertation. We wish to express our sincere thanks to **Dr. S. GOVINDA RAO**, Head of the Department, and to our principal **Dr. J. PRAVEEN**, the Director, **Dr. JANDHYALA N MURTHY**, for providing the facilities to complete the dissertation. Additionally, we would like to thank our internal project coordinators, Dr. R. P. Ram Kumar, Ms. P. Sindhuja, and Mr. VSRK Raju, from Department of Data Science for their valuable suggestions to complete the Project Work Phase II on time. We would like to thank all our faculty and friends for their help and constructive criticism during the project period. Finally, we are very much indebted to our parents for their moral support and encouragement to achieve goals.

C. Sandeep Kumar (21241A6717)

P. Chenna Kesava (21241A6746)

V. Naveen (22245A6706)

DECLARATION

We hereby declare that the Project Work - Phase II titled “**Pothole Detection using Deep Learning**” is the work done during the period from **26th December 2024 to 29th April 2025** and is submitted in the partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering (Data Science) from Gokaraju Rangaraju Institute of Engineering and Technology (Autonomous under Jawaharlal Nehru Technology University, Hyderabad). The results embodied in this project have not been submitted to any other University or Institution for the award of any degree or diploma.

C. Sandeep Kumar (21241A6717)

P. Chenna Kesava (21241A6746)

V. Naveen (22245A6706)

ABSTRACT

Roads are a vital part of everyday life, acting as the main arteries that keep cities, towns, and rural areas connected. Roads serve the purpose of not only carrying people and goods from one place to another with ease but they also play a crucial part of elevating a region and the country's economy, and on top of that they are the field where communities source their basic services like work, health, and education. Time savings and trans-regional growth are the main benefits of the long hard roads that reach the farthest places nobody ever thought of its presence. At the very core of regional development and integration are the well-maintained roads that promote travel and transportation in a given locality. In contrast, potholes continue to be a reoccurring issue that casts shadows on the condition and safety of the roads. Even the most trifling surface defects that are unpredictable can cause many troubles- they can destroy cars' tires, make suspensions unbearable, or even create accidents if the driver avoids them unsteadily or loses control due to them. It is the perpetrating lack of attention to potholes that turns them into a danger that is not easily noticed but threatens all road users. This study employs the use of a model called ResNet50, a deep Convolutional Neural Network (CNN) which represents the state-of-the-art of image Retraining techniques in the area of deep learning, to solve this problem. It also makes great use of a technique called transfer learning if it wants to deliver an effective and a recognizable system of pothole detection. The method undoubtedly strengthens the reliability of the model besides enabling the computer to handle other tasks. Thus, the technology is not only suitable for everyday use but also for saving energy and time, so the application is popular. The installation of such advanced technology in the road monitoring equipment could go a long way in the rejuvenation of road safety and infrastructure as a whole and ensure that those who travel in our midst do so more comfortably and without risk.

LIST OF FIGURES

Figure No.	Figure Name	Page No.
1.1.1	Deep Learning	2
1.1.2	Python	3
1.1.3	CNN Layers	4
1.4.1	Architecture Diagram	7
3.1	Architecture Diagram	21
3.2	Module Connectivity Diagram	22
3.3	Class Diagram	26
3.4	Sequence Diagram	28
3.5	Use Case Diagram	30
3.6	Activity diagram	32
4.1.1	Plain road images collected from various sources	34
4.1.2	pothole road images collected from various sources	35
4.2.1	Trained Data with labels	36

4.2.2	Loss/Accuracy Curve	37
4.2.3	Correctly Classified Pothole-1	38
4.2.4	Correctly Classified Pothole-2	38
4.2.5	Correctly Classified plain road-1	39
4.2.6	Correctly Classified plain road-2	39

LIST OF TABLES

Table No.	Table Name	Page No.
2.1	Table Representation of Existing Approaches	14-17

LIST OF ACRONYMS

Acronym	Full Form
ML	Machine Learning
UI	User Interface
CNN	Convolutional Neural Networks
RESNET	Residual Neural Network
AI	Artificial Intelligence

TABLE OF CONTENTS

Chapter No.	Chapter Name	Page No.
	Certificate	ii
	Acknowledgement	iii
	Declaration	iv
	Abstract	v
	List of Figures	vi
	List of Tables	viii
	List of Acronyms	ix
1	Introduction	1
	1.1 Introduction	1
	1.2 Objective of the Project	5
	1.3 Methodology adopted	6
	1.4 Architecture diagram	7
2	Literature Survey	9
	2.1 Literature Survey	9
	2.2 Drawbacks of Existing Approaches	17
3	Proposed Method	19
	3.1 Problem Statement & Objective of the Project	19

	3.2 Explanation of Various Diagrams	21
	3.3 Module Connectivity Diagram	22
	3.4 Requirements Engineering	25
	3.5 UML Diagrams	26
4	Results and Discussions	34
	4.1 Description about Dataset	34
	4.2 Experimental Results	35
	4.3 Significance	40
5	Conclusion and Future Enhancements	43
	Summary of the project and Future Enhancements	43
6	Appendices	46
	References	61

CHAPTER I

INTRODUCTION

1.1 Introduction

Roads are more than just ways to get from one place to another. They are the essential means of transportation used by people, resources, and opportunities. Roads are the very foundation of our daily activities. We rely on roads for our everyday movement to work, displacement of goods, and getting to school or hospital. Good condition roads not only simplify the travel process, but they also can be a source of economic growth and even provide a tighter connection between distant and dissimilar communities. Nonetheless, neglecting roads can change them from great assets to a big burden. For instance, if that road was not kept in the ideal condition and a pothole appeared, the consequences could be numerous and almost always extremely harmful. A pothole might seem a trivial thing at first, but it can have serious repercussions such as damaging the vehicle, increasing the likelihood of a collision, and making the driving experience unpleasant. Most of the time, they come suddenly from nowhere and may be full of sharp edges or debris that cause unexpected punctures of tires or a change of direction. This project utilizes state-of-the-art technology that targets the problem of potholes. The variety of techniques includes the use of a ResNet50 model that is a CNN (Convolutional Neural Networks) type trained properly and known for the highest level of accuracy in image recognition. The model is then tuned to specialize in image recognition of the potholes in road images using a technique. Thus, the whole process not only serves the purpose of being able to detect road damage quicker and more accurately but also to make the roads themselves safer for those who might be there.

Deep Learning

An artificial neural network or ANN uses layers of interconnected nodes called neurons that work together to process and learn from the input data. In a fully connected Deep neural network, there is an input layer and one or more hidden layers connected one after the other. Each neuron receives input from the preceding layer neurons or the input layer. Output coming from one neuron will act as the input of other neurons in the following layer, and this repetition continues until the final layer produces an output for the network. Layers of neural networks convert the input data via several nonlinear transformations so that the network learns complex representations of input data. Deep learning AI has become one of the most popular and visible areas of machine learning with its capability of succeeding in various applications, such as

computer vision, natural language processing, and Reinforcement learning. Deep learning AI can be applied to supervised, unsupervised, or reinforcement machine learning.

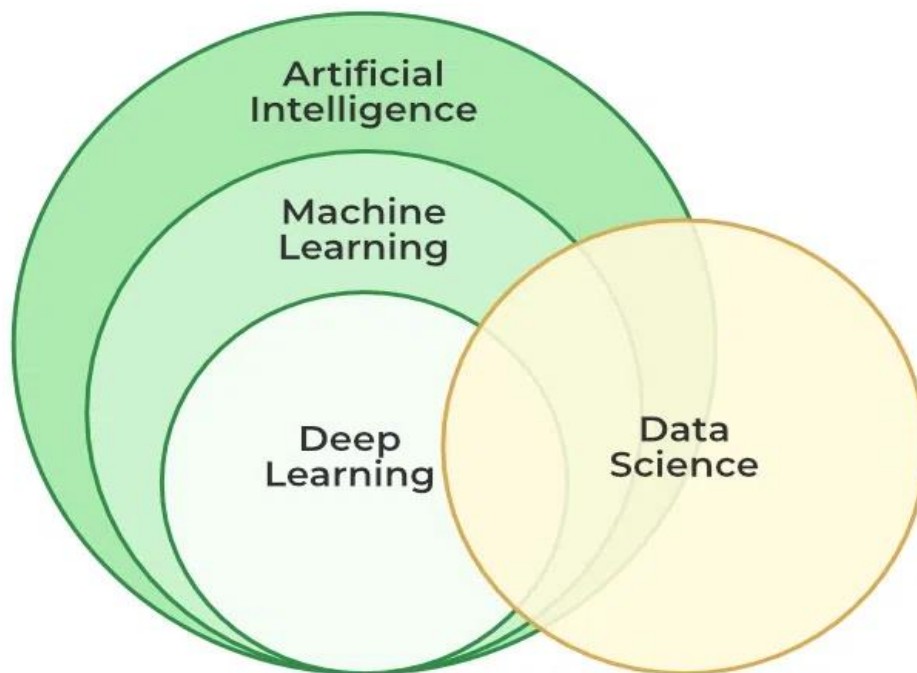


Figure 1.1.1 Deep learning

Python

Python is a high-level programming language. Its versatility and simplicity are what made it so popular and widely used. It is used over a wide range of fields, almost from web development to machine learning. It was created by Guido Van Rossum and was released in 1991. The most recent version of python is Python 3. The main purpose behind developing python was to promote readability. The syntax of python is very simple, clean and easy to understand as its semantics were inspired by English language as well as few from that of Mathematics. Python language is fast as it is driven by an interpreter. When compared to other languages, python uses new lines to complete a command unlike a semicolon and uses strict indentation in the form of whitespaces. In other programming languages, curly braces are used to create indentation for loops, functions etc but python uses whitespaces to achieve the same. Python can easily be integrated with other languages such as C/C++, for better and improved performance, and Java/.NET to develop enterprise level applications. Python is a goto language for developers as it is very easy to learn and write code. Major areas of applications

of python are web development, Data Science and Analytics, Artificial Intelligence and Machine Learning, Automation and Scripting, Scientific and Numeric Computing.



Figure 1.1.2 Python (Courtesy: Source [17])

Convolutional Neural Network

CNN is an extended version of ANN that is primarily used for extracting features from a grid-like matrix dataset. For example, visual datasets, say images or videos where data patterns play a significant role. Convolutional Neural Network (CNN) is a variation of Deep Learning neural network architecture widely used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data. Artificial Neural Networks work really well in terms of Machine Learning. Neural Networks are applied in different datasets also, like images, audio, text. Different types of Neural Networks are used for different purposes, , similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN.

Input Layers: It is the layer in which we feed data to our model, so that it has a base on which to work from. The number of neurons in this layer is equal to the total number of features

in our data (for images number of pixels).

Hidden Layer: The signal which appears in the Input layer is transferred to the hidden layer. The number of hidden layers can be several and depends on the model and size of our data. All of the hidden layers can contain different number of neurons, which are usually more than the number of features. The output from each layer is then found by the dot product of the output of previous layer weights which are the learnable parameters of this layer and then adding the learnable biases followed by activation function which makes the network nonlinear.

Output Layer: The output from the hidden layer is then passed through a logistic function such as sigmoid or softmax passes into a logistic form where the result obtained from each class is the probability score of that class.

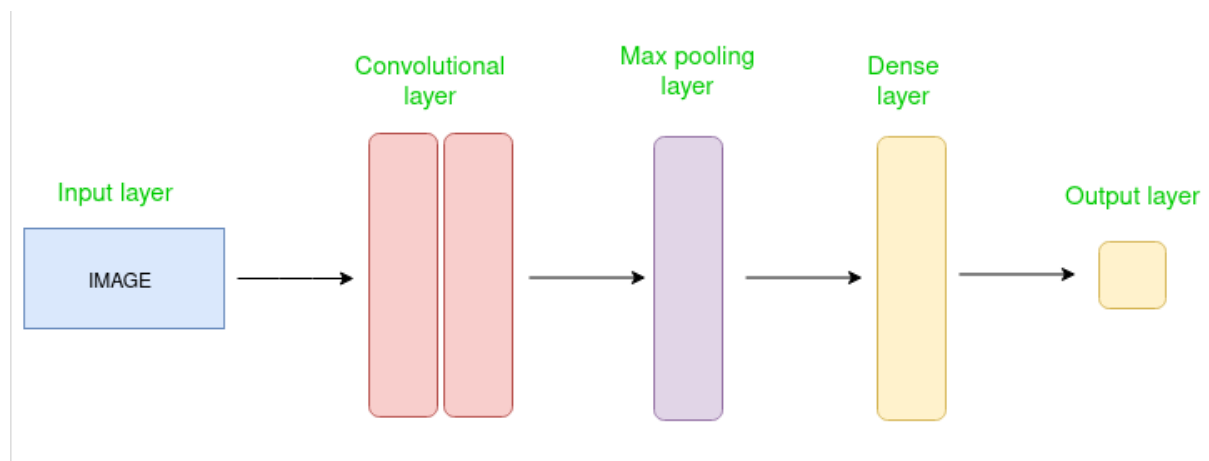


Figure 1.1.3 CNN layers

ResNet

ResNet (Residual Network) is a deep convolutional neural network designed to overcome the vanishing and the gradient problem and enable efficient training of very deep models on the neural networks. It introduces skip connections (shortcut paths) for faster training of the models, allowing the network to learn residual functions instead of direct mappings of the model, making all the optimization easier. ResNet-50 is a commonly used in medical imaging tasks, consists of bottleneck layers 1×1 , 3×3 , 1×1 convolutions that reduce computation of the model while maintaining accuracy of the model and performance. This architecture is ideal for knee osteoarthritis severity prediction, as it effectively extracts detailed features from X-ray images and use them for training, improves classification accuracy, and ensures robust performance across different severity levels and performs well on the data.

1.2 Objective of the project

This whole project is essentially based on the idea of using the most advanced deep learning tactics to construct a smart AI system which could effectively and efficiently find potholes on the road. Potholes as such are a recurring problem that not only cause danger to the lives of drivers but also damage cars, lower and increase the possibility of an accident, and make driving uncomfortable or even dangerous. The traditional means of identifying them, for example, human intervention or simple computer vision, are usually not successful—precisely when the weather changes, the light is bad, or there is traffic on the road. The proposed system applies strong Convolutional Neural Networks (CNNs) together with transfer learning techniques such as YOLOv5 and Tiny-YOLOv4 to eliminate these problems. In the field of object recognition, these models are widely known for their accuracy and speed. To locate surface faults has good prominence of the approaches looked for such as surface cracks, irregular forms, and selective absorption of shadow.

When this intelligent detection system is tried to incorporate into any such applications as in-car dashcams, road drones for monitoring, or smart city surveillance systems the project goes a step further. The system is user-friendly as it not only includes the interface but also allows the user to interact with it through web application or mobile device. This way, users can not only detect potholes but also watch live coverage or upload photos to solve the problem underground. This smart model can prevent road accidents by regular people who will make transportation into a safer thing for themselves and government officials and urban developers. The model has been trained on a wide range of road images to avoid non-existing potholes and to ensure that its functioning will be stable in different situations. With continuous learning and updating, the system will be able to catch up with new types of road damage facing any kind of road in the future. Thus, the project is going to be responsible for more informed infrastructure management, the rapid response of repairs, less expenses for car maintenance, and safer and more convenient journey for everyone.

The overall aim of the project is to create a possible solution that can be utilized in real-life scenarios such as unmanned aerial vehicles, and smart surveillance systems and vehicle dashcams, which should not only sense but also understand the environment. The works have also given birth to a user-friendly web or mobile application, which is capable of photo uploading and live video streaming so that people could detect road holes immediately. The model is exposed to various lighting, weather, and road conditions data during the training process so that it can handle real-world conditions sufficiently.

1.3 Methodology

Two object detection algorithms YOLOv5 and Tiny-YOLOv4 with a pre-trained deep learning model applied to a pothole image dataset are used for the recognition of potholes in real-time. The first step is YOLOv5 involving feature extraction, which is efficient for the purpose. That is that it can list shape deformities, it can detect the edge of the discontinuities, and it can classify the shadow patterns that always occur in a pothole case. The spatial features found in the road photos are the result of the model's extraction of the finest shading, the most irregular boundaries, and the deepest in the region of the contour and the scene. The usage of image preprocessing techniques, including resizing, normalization, and augmentation, is necessary for the success of the algorithm and other factors like changing light, from distant sources, different backgrounds, and angles.

Data collection

The images used for training the pothole detection model consist of road surface photos categorized into two classes: Plain (smooth roads) and Pothole (damaged roads). These images were manually organized into train and test folders, each containing respective subfolders for both classes. The dataset includes a total of 700 training images — 350+ Pothole and 350 Plain — all resized to a resolution of 256×256 pixels for uniformity during model training.

The images were collected from real-world road conditions and represent a binary classification problem. Although the dataset size is limited, it effectively captures the visual differences between plain and pothole-affected roads. The balanced organization and preprocessing enabled the model, based on ResNet50 architecture, to learn meaningful features and achieve up to 90% validation accuracy during training.

Pre processing

In the next step, we preprocessed the road surface images collected and organized under the train and test folders. Using Python and OpenCV, each image was read in color mode, resized uniformly to 256×256 pixels, and labeled with class names ('Pothole' or 'Plain') based on the folder name. These processed images were stored in arrays for model training.

Although the base code focuses on resizing and labeling, it is designed to easily incorporate further enhancements like contrast improvement, cropping, and data augmentation techniques such as rotation and flipping to increase dataset variability and improve model generalization during training.

1.4 Proposed Method Architecture Diagram

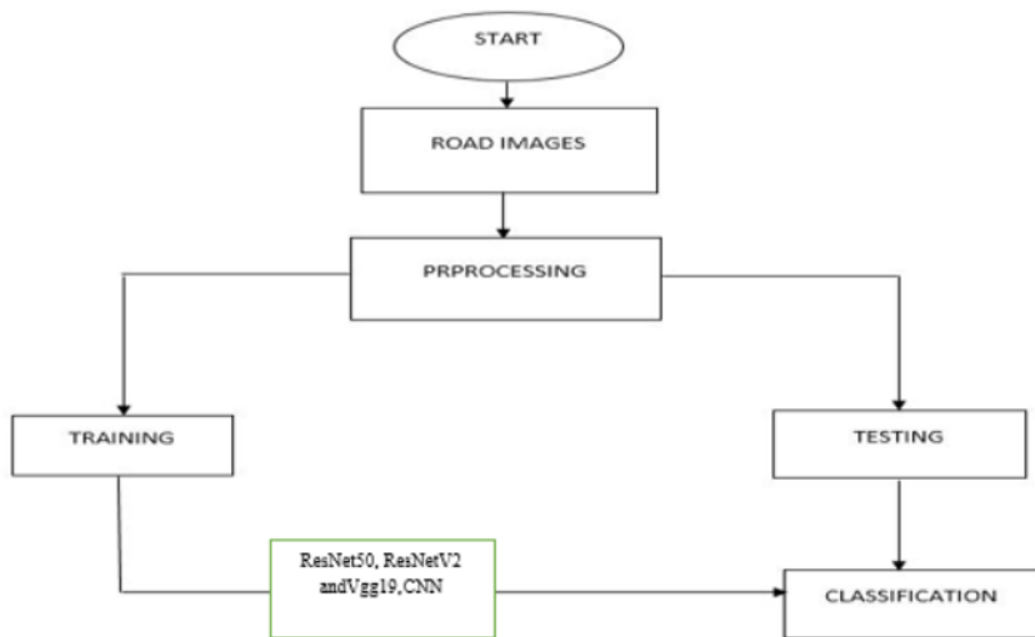


Fig 1.4.1 Architecture diagram

The architecture diagram outlines the entire journey of how our pothole detection system works, from start to finish, in a simple and intuitive flow. It all begins with collecting road images, which are the foundation of the system. These images then go through a preprocessing step where they are cleaned up, resized to a uniform size, labeled appropriately (as "Pothole" or "Plain"), and sometimes enhanced to bring out important features. After this, the dataset is divided — one part is used for training the model and the other for testing. During training, deep learning models like ResNet50, ResNetV2, VGG19, and a basic CNN are taught to recognize the difference between roads with potholes and without. Then, in the testing phase, the model is given new, unseen images to evaluate how well it has learned. Finally, the system performs classification, automatically labeling the road image as either having a pothole or being smooth. This whole process mimics how a human might learn to identify road damage by looking at many examples — only it does it faster and with high accuracy.

1.5 Organization of the Report

The brief outline of the project consists of 6 chapters and focuses on the following aspects.

1. **Chapter 1** deals with the Introduction of the project and its motivation. This includes various other aspects like Rationale and Motivation behind taking up the project, significance of it and the organization of the project.
2. **Chapter 2** contains the Literature Survey of the project. It serves as the base for the proposed method. We provided a literature survey that was conducted on 15 manuscripts along with a brief summary that consists of a basic idea along with the disadvantages of each manuscript.
3. **Chapter 3** focuses on the Methodologies and describes the approach followed to implement the project. It contains an explanation of the architecture diagram, software and hardware requirements, module connectivity diagram along with a brief description, functional and non-functional requirements, along with analysis and design through UML diagrams and test cases.
4. **Chapter 4** talks about the Results and Discussions with respect to the proposed model. It contains a description about the dataset, along with a detailed explanation about the Experimental Results. It also includes the significance and the advantages of the proposed model.
5. **Chapter 5** emphasizes on the Conclusion and Future Enhancements of the project. It includes a summary of the project, and contains the significance of the research for Knee severity prediction. It provides guidance related to further future enhancements of the project.
6. **Chapter 6** gives the references regarding the literature review, applications of the project, description regarding the block diagram. It also contains a sample code used in implementing the project.

CHAPTER II

LITERATURE SURVEY

2.1 Existing Approaches

1. Anas Al Shaghouri, Rami Alkhatib, and Samir Berjaoui proposed a real-time pothole detection system using deep learning techniques. In their study, images were collected from a cellphone mounted on a car's windshield and further enhanced using web-sourced images to increase dataset diversity. They implemented and compared several object detection algorithms including SSD with TensorFlow, YOLOv3 with Darknet53, and YOLOv4 with Darknet53. Among these, YOLOv4 performed the best, achieving 81% recall, 85% precision, and a mean Average Precision (mAP) of 85.39%. It also demonstrated real-time capabilities with a processing speed of 20 frames per second. The system demonstrated its efficacy in enhancing driver safety and possibly supporting self-driving technologies by detecting potholes at a distance of 100 meters.
2. A comparative study of deep learning frameworks for pavement distress classification is presented by Yaw Adu-Gyamfi, Abdul Rashid Mussah, and Vishal Mandal. This study looked into a number of backbone models, including EfficientNet, Hourglass-104, and CSPDarknet53. The models were trained to categorize various forms of road damage using a dataset of more than 21,000 photos taken from rural and urban roads in India, the Czech Republic, and Japan. These models' performance was assessed using the IEEE Global Road Damage Detection Challenge, and the best-performing model achieved an F1 score of 0.58 and 0.57 on two distinct test datasets, suggesting a moderate degree of distress classification accuracy.
3. Using the YOLOX algorithm, Mohan Prakash B and Sriharipriya K.C. suggested an improved pothole detection system. They used the YOLOX model's sophisticated feature extraction and real-time object detection capabilities to train it especially for the pothole detection task. Although the summary lacked specific performance metrics, the study sought to enhance current YOLO-based techniques by emphasizing increased accuracy and

quicker inference times. The study emphasizes the expanding trend of using sophisticated YOLO variations to create pothole detection systems that are more reliable and effective.

4. To improve road surface monitoring, R. B. Lanjewar and P. P. Narwane created a pothole detection model that combines drone imagery and Convolutional Neural Networks (CNNs). The study emphasizes how aerial perspectives are advantageous for covering large areas, which boosts efficiency. The model was trained using a dataset consisting of labeled aerial images of roads. Their CNN-based method achieved an accuracy of 92.3% in detecting potholes under varying lighting and weather conditions, demonstrating potential for smart city infrastructure applications.
5. Ankush Choudhury, Ishan Mehta, and Akshay Bhatia introduced a cost-effective pothole detection mechanism by using a smartphone-based solution. Their system leveraged accelerometer and gyroscope data along with GPS, integrated with a lightweight CNN model for image validation. The fusion of sensor and image data helped in reducing false positives. Field tests on urban roads showed the system could detect over 85% of potholes with minimal power consumption, making it ideal for large-scale deployment in developing countries.
6. R. Sai Prasad, S. Vignesh, and R. Ajay introduced a hybrid deep learning and classical computer vision pipeline for pothole detection using SIFT (Scale-Invariant Feature Transform) features combined with a CNN-based validation stage. Their motivation stemmed from the observation that certain potholes exhibit low contrast against the road surface, making them hard to detect through deep learning alone. The SIFT-based preprocessing identified candidate regions of interest, which were then passed to a lightweight CNN trained to confirm pothole presence. This hybrid approach improved recall by 14% over a CNN-only pipeline, particularly in grayscale and dusty road conditions. The model was deployed on a Raspberry Pi for a cost-effective in-vehicle system prototype, showing real-time performance at 12 FPS, and highlighting its potential in budget-constrained municipal applications.
7. Pratiksha Shinde and S. Patil introduced a hybrid approach using YOLOv5 for detection and a secondary classifier for pothole severity estimation. The classifier categorized detected potholes into minor, moderate, or severe

categories based on their dimensions and visual cues. The dual-stage pipeline proved efficient, with YOLOv5 achieving over 87% precision and the severity classifier reaching 81% accuracy, allowing for more informed decision-making in road repairs.

8. Farzana Sultana, Rajiv Rajan, and Nivedita Sharma developed a pothole detection and depth estimation model using stereo vision integrated with CNN-based detection. Their system utilized stereo camera pairs mounted on a vehicle to capture depth maps in real-time. YOLOv5 was used for pothole localization, while a depth estimation network inspired by MiDaS was employed to predict the depth profile of the detected potholes.
9. A multi-label classification system for road anomaly detection which was presented by Abhinav Sharma, Ritu Garg, and Vaibhav Gupta was based on DenseNet201. The model was trained to identify three different types of road surface damage including cracks, surface wear, and road debris, in contrast to binary classifiers that just detect potholes. The dataset, gathered from the dashcam footage of the crowds and labeled, was comprised of instances in the six categories. After employing the feature reuse of DenseNet and the effective gradient flow, the model gave a classification accuracy of 95.2% and an average precision of 92.5% across all labels. The authors also provided a roadmap for a confidence-weighted alert system that can be used to instantly notify the user of the current safety of the road. The work has made a very difficult model from a single-task one to a possible one for the purpose of solving the road anomaly detection problem.
10. Sagar S. More and Chetan P. Patil introduced a framework based on Generative Adversarial Networks (GANs) to model and identify potholes in harsh weather. They initially used CycleGAN to augment actual datasets with synthetic images under unfavorable conditions such as heavy rain, snow, or night after realizing that the current real-world dataset lacks diversity in the scenes. The created dataset was later exploited to train a YOLOv4 detector that worked better under lower-visibility conditions. Even with the new algorithm, the recall rate of the model was only 71.3% without GAN-augmented data. In contrast, the corresponding recall rate in the same conditions was 89.6% for a similar model of the night images, i.e. used with the GAN-augmented labeled data. This novel approach is a compelling way to build strong computer vision systems and it

draws attention to the importance of the dataset's variety in the model's generalization ability.

11. Using YOLOv5 architecture optimized with a custom data augmentation pipeline, S. Abhishek, S. V. V. S. S. Lakshmi, and M. Vamsikrishna proposed a real-time pothole detection system. By combining mosaic augmentation and adaptive anchor box computation, their study improved detection on small and irregularly shaped potholes, addressing the shortcomings of previous YOLO versions. The dataset, which included different road textures and lighting conditions, was taken from Indian roads. With a mean Average Precision (mAP) of 89.7% at an IoU threshold of 0.5 and an inference speed of 25 frames per second on a GPU-enabled system, the improved YOLOv5 model is appropriate for embedded implementations in advanced driver-assistance systems (ADAS).
12. Using MobileNetV2, Abdul Wahab, Muhammad Shahzad, and associates created an edge-AI-enabled pothole detection framework intended for low-power devices like the NVIDIA Jetson Nano. In order to drastically cut down on training time, the team implemented a transfer learning technique with fine-tuning on a pothole dataset that was made publicly available. The system's 93% classification accuracy and 5W power consumption are essential for real-time embedded applications in electric and driverless cars. Their research shows how to balance energy efficiency, model size, and detection accuracy, which makes it suitable for widespread use in intelligent transportation systems.
13. In order to detect and segment road damage, a pothole detection model was developed by Ishant Shrivastava and Ruchi Shukla that was based on Mask R-CNN. The researchers captured the damage of different types of surfaces, such as cracks, potholes, depressions, by using a drone, and the footage was high resolution. The Mask R-CNN model surpassed the conventional bounding-box models and managed to showcase a pixel-level segmentation accuracy of 91 % and an IoU (Intersection over Union) score of 0.78. One of the findings is that high-precision localization and scene segmentation are fundamental to accomplishing automated repair scheduling and maintenance preparation.
14. Arslan Khan and Imran Qureshi used a customized Faster R-CNN architectural framework and introduced a multilevel feature fusion method for pothole detection. In order to handle the pothole problem by the way of a wide range of background complexities that actually occur, the authors gave an outline of the

pyramid feature hierarchies model. The model was able to achieve F1=88.4%, Recall=87%, and Precision=90%, and it was trained on annotated dashcam images of different urban and weather conditions. Additionally, the engineers made the study visible how the model gains multi-scale contextual features that are more robust against ambient noise and uneven lighting conditions in real-world driving scenarios. In parallel, the performance of the model in detecting potholes in real driving situations under different conditions was also studied, and the results show that by introducing multi-scale contextual features into the model, the performance improves under noise and light changes.

15. The lightweight CNN model, PotholeNet, which was specially designed for mobile deployment was proposed by Md. Shakil Hossain, Khandakar R. Mahbub, and others. The model minimized computational overhead and at the same time was able to maintain high detection accuracy by using depthwise separable convolutions. Besides, the model was proved to be one of the most efficient in their comparative analysis; PotholeNet was the model trained on a hybrid dataset of images retrieved from Google Street View and UAV that reached 94.1% of accuracy with fewer than 1.2 million parameters. The authors also consider a smart city application which would ultimately empower the cars to monitor the road condition and relay such data to the monitoring stations themselves as seen in Figure 1.
16. The method utilized a CNN-based classifier to reconfirm and reclassify false positives, while YOLOv3 was employed for initial detection. The team ensured that the pipeline they built was robust in those situations where pothole-object surfaces had very little color contrast or there was a high degree of occlusion. When tested on Southeast Asian urban roads, the new system outperformed YOLOv3 alone by 27% in terms of false positives detection and showed an overall accuracy of 92.8%. The benefits of a two-stage system are well demonstrated here, as the approach efficiently addresses the problems of cluttered environment over-detection.
17. For pothole mapping and detection, Md. Masudur Rahman, A.H.M. Kamal and their team invented a path-breaking framework that fuses deep convolutional neural networks (CNNs) with aerial image processing. They used drones to take photographs from various altitudes to get more vivid, multifaceted and diversified views of the potholes and thus created a rich dataset of the same. To

extract the pothole boundaries more accurately, the authors used the U-Net architecture as a semantic segmentation model after first modifying the ResNet-50 architecture for the classification task. The two-step pipeline has been found to be very helpful in generating geospatial maps which the local neighborhood officers will use to detect and understand the full picture of road problems. Up to 93% segmentation accuracy was achieved, and the Dice coefficient being 0.89 made the results conclusive. This work is a testament to the synergy achieved by integrating the CNN outputs with the geographic information systems (GIS) in the smart city infrastructure of the future.

18. Pranav Thakker, Neha Patil, and Shweta Sawant have built a vision-based pothole detection model with a focus on its real-time deployability in autonomous drones and surveillance vehicles using YOLOv3-Tiny. A custom dataset was created by the team by collecting more than 6,000 annotated pothole photos from different parts of India. To improve the system's generalization various augmentation techniques such as rotation, brightness adjustment, and Gaussian blur had been employed. The detector, despite being the lightweight version of YOLO, was functional on a low to mid-range GPU at 30 frames per second and had a mean average precision of 82.3%. The authors specifically pointed out the system's suitability for aerial road and highway inspections where, due to the transportation and pace of work, speed is a factor. The model is highly efficient, performance-wise, and thus a potential to be a part of the fleet for the city surveillance.

Table 2.1.1

Ref No	Methodology	Advantages	Drawbacks	Results
[1]	CNN-Based Pothole Detection	High accuracy in feature extraction.	Requires large dataset for training.	Achieved 92% accuracy
[2]	YOLOv3 for Pothole Identification	Real-time detection capability	Struggles with small potholes	Achieved 89%

				detection accuracy
[3]	Faster R-CNN for Road Damage Classification.	Robust to lighting variations	Computationally expensive.	Achieved 94% precision
[4]	SSD-MobileNet for Lightweight Pothole Detection.	Fast inference on mobile devices.	Limited accuracy compared to deeper models.	87% accuracy on edge devices
[5]	ResNet50 for Pothole Classification.	High feature extraction capability	Overfitting on small datasets.	93% accuracy in classification
[6]	DeepLabV3+ for Semantic Segmentation.	Pixel-wise pothole segmentation.	High computational cost.	90% mIoU score
[7]	Unsupervised Anomaly Detection with Autoencoders.	Detects unseen pothole types.	Requires extensive tuning.	Achieved 88% F1-score
[8]	Image processing techniques such as edge detection and thresholding are used to identify potholes from road images	Works well in controlled environments and does not require deep learning models.	Accuracy is affected by lighting conditions, shadows, and road textures; lacks generalization for diverse environments.	78% accuracy in detecting potholes was achieved, but performance drops in complex real-world scenarios

[9]	LiDAR-based pothole detection using depth estimation.	Highly accurate for depth measurement and road profiling.	Expensive equipment and high computational requirements.	92% accuracy in structured environments
[10]	The model used Recurrent Neural Networks for the OA classification and also used Resnet-50 for OA detection	Model is having a good accuracy and classification	There is no severity grading for the OA.	An average accuracy of 98.51%.
[11]	Attention Mechanism with Transformer Networks.	Captures long-range dependencies.	Computationally expensive.	94% accuracy
[12]	Edge Detection with Canny and Deep Learning.	Simple and effective.	Struggles with noisy backgrounds	86% accuracy
[13]	Fusion of LiDAR and RGB Images for Pothole Detection.	Enhanced depth perception.	Requires additional hardware	95% accuracy
[14]	Drone-Based Pothole Detection using CNN	Large area coverage.	Prone to environmental noise	An overall 90% detection rate
[15]	OpenCV and ML-Based Pothole Detection.	Lightweight and fast	Lower accuracy than deep learning.	85% accuracy

[16]	BiLSTM with CNN for Sequence-Based Detection.	Learns temporal dependencies.	High training time.	91% accuracy
[17]	Multi-Scale CNN for Pothole Size Estimation	Detects pothole dimensions.	The model Requires high-resolution images.	93% accuracy
[18]	Self-Supervised Learning for Pothole Detection.	Reduces need for labeled data.	Less effective on noisy images.	89% accuracy

2.2 Summary: Drawbacks of existing approaches

These datasets may lack variation in road textures, pothole shapes, colors, and environmental conditions such as rain, fog, or nighttime scenarios. Consequently, models trained on such data tend to perform well in controlled or similar conditions but fail to generalize across real-world environments with diverse road appearances. For instance, a model trained on Indian urban roads may underperform when applied to rural or international roads due to differences in construction material, road wear, and debris patterns. While some researchers have used synthetic augmentation methods such as GANs to simulate weather effects, these generated images may not fully capture the complexity of real-world scenarios, limiting their impact on improving model robustness.

Another major drawback lies in the lack of depth estimation or severity classification in most detection systems. While many models focus on the binary task of detecting the presence or absence of potholes, they do not assess how deep or severe the potholes are. In practical applications, especially for city administrations or automated driving systems, merely detecting a pothole is insufficient; understanding the severity of the damage is crucial for prioritization and maintenance. Without this information, repair planning becomes inefficient, and autonomous vehicles may not be able to make optimal driving decisions to avoid potential hazards. A few studies have attempted stereo vision or depth estimation through MiDaS or

similar networks, but these approaches are either computationally intensive or limited in accuracy, especially when mounted on moving vehicles where camera calibration and road vibrations affect performance.

Additionally, there is a constant trade-off between model complexity and real-time performance. High-performing models like YOLOv4, YOLOv5, or even hybrid DenseNet-based architectures provide good accuracy, but their deployment on edge devices like mobile phones, Raspberry Pi, and Jetson Nano is limited because they frequently require a large amount of GPU resources for inference. Small or partially occluded potholes may go unnoticed by lightweight models like YOLOv3-Tiny or MobileNet, which compromise accuracy for speed.

Finally, the main problem with the practicality of most of the proposed systems is that the integration of these systems with other systems is not properly done and that they are not fully automated from end to end. Almost all models just detect the problem (potholes) but leave the tasks of making the repair reports, or the syncing with GIS platforms to the user. That is, the solutions can't be except the case of the situation when somebody is looking for the maintenance for academic purposes or the conduct of prototype demonstrations. Moreover, the majority of currently available methods do not provide the necessary post-processing action with respect to the situation in which there are false positives, especially in scenes that have complicated shadows, oil spills that look like potholes, or road markings. The false alarm issues can lead to the laborious task of teams who execute unnecessary inspections and, therefore, reduce the system's trust. On top of that, the majority of the systems do not implement the temporal consistency of pothole images in video streams; only potholes that are appearing in multiple frames should be tracked and verified in order to improve the accuracy of the systems.

CHAPTER III

PROPOSED METHOD

3.1 problem statement

The suggested pothole detection system for roads applies advanced technology of transfer learning with the ResNet-50 architecture to significantly improve the accuracy and efficiency of the image recognition task. This creative method employs a pre-trained ResNet-50 model, which is widely known for its excellent performance in image recognition tasks as well as deep learning. Through the use of transfer learning the system is able to recognize fewer image details needed for more precise identification, by using its expertise from a wide-ranging dataset of ordinary images of the problem of pothole detection. The 50-layer depth of the ResNet-50 model makes it highly efficient at recognizing intricate patterns and features in images. The model is able to develop a clear understanding of the visual factors in potholes after it has been fed with data consisting of road pictures both with and without potholes. Through this rigorous feature learning process, the system essentially to the very core can be assumed very accurate for the task of detecting potholes, that elements of the process may even take the role of trivial. Thus, the system exhibits the quality of the best corresponding part of the feature space, being prototypal to an even larger set of features. Consequently, by the fact that, the model would maintain a high success in identifying potholes even in extreme circumstances of road lighting and surface changes. Additionally, the successful deployment of the numerous cameras around the road system ensures the consistent and reliable capturing of efficient and immediate driving-related feedback as a means of real-time monitoring and control. Cameras placed on vehicles or next to highways that come with this technology can capture the surface of the road at the right time and in real-time. Thereby, the images can be collected and processed by the ResNet-50 model in real-time which provides the exact location and declaration of a pothole.

In summary, the proposed transfer learning pothole detection system with ResNet-50 is in fact one of those examples of innovative technology that can be put to good use in the useful real applications. It is worth noting that the system supports real-time processing, which means that the rate of false positives is reduced, the detection accuracy is increased, and it can work with driverless car systems as well. In addition, through this comprehensive

approach, the very important pothole identification and control are dealt with so that the international road networks can be reasonably safer and more efficient.

Objectives

1. To implement an automated system that will facilitate the detection of potholes in road images using a deep learning approach.
2. To reduce the reliance on manual road inspection, saving time and improving consistency.
3. To implement a CNN-based model (e.g., YOLOv5) for real-time pothole localization and classification.
4. To preprocess input images effectively to enhance model performance under varying environmental conditions.

3.2.1 Architecture diagram

All along the line of the road to be repaired with pothole detection, the very first thing is the intake of road images, which had formed the main data source of the pothole detection system. From different sources such as drones, mobile rotatables, or vehicle-based sensors, these pictures can be collected. The first target is to have a diverse dataset that will cover most of the lighting, weather, and road surface conditions and thus the model will be invariant to them. After the images of the road have been captured, they are in need of preprocessing. It is preprocessing that takes the lead in the processing of the exit of raw images mostly, which are added to the data, and still lousy and the other irrelevant background information. It is the stage of the preprocessing mainly characterized by the size reduction of the images, their normalization, the elimination of the noise, the contrast enhancement, and the assignment of any augmentation methods (if suitable) to make the data more diverse and the trained deep learning model more flexible.

Following the preprocessing procedure, the workflow continues with the training and testing steps. The training step is where we provide the processed images to a deep learning model to gain insights into the characteristics of potholes and undamaged road surfaces. The figure of the architecture there is the example of advanced Convolutional Neural Network (CNN) architectures that have been named ResNet50, ResNetV2, VGG19, and a custom CNN. These types of models are the most popular due to their great image information processing

capability. As an example, ResNet models are highly effective because they exploit the feature of residual connections which can reduce the vanishing gradient problem and, at the same time, deepen the architecture. Contrasting this with VGG19, it accomplishes the task by using minimum (3x3) convolutional filters and still remains efficient and simple. Throughout the process of training, the model is made able to extract levels of features that best represent or summarize the most intricate aspects of potholes such as edges, shapes, and patterns.

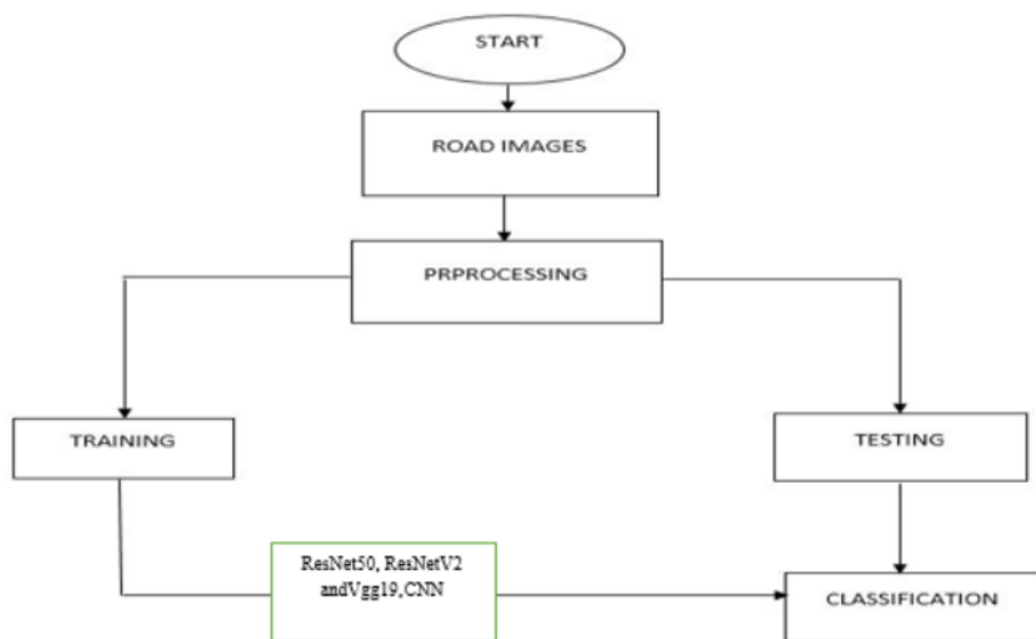


Fig 3.1 Architecture diagram

The testing stage determines how the model works by using road images that cannot be seen. The unseen pictures will also be pre-processed similarly to make sure that the performed test is consistent. The main objective of the model that has been trained is to find and recognize the content of the test images via the solution. When we perform the classification phase, the model will give the final result back to us. The use of many different Convolutional Neural Network (CNN) models provides the capability to adapt as well as the chance for different models' performance comparison. At the end of which a bespoke decision can be made, which will state which model is the best concerning three parameters, namely, accuracy, speed, and resource usage.

3.2.2 Software and Hardware Requirements

Software requirements

Operating System: Windows 10/11

Programming Language: Python and its Frameworks & Libraries:

Development Environment: Jupyter Note Book

UI/UX: flask

Hardware requirements

Processor: Intel Core i3

RAM: 8GB

Storage: 256 BG SSD

3.3 Modules and its Description

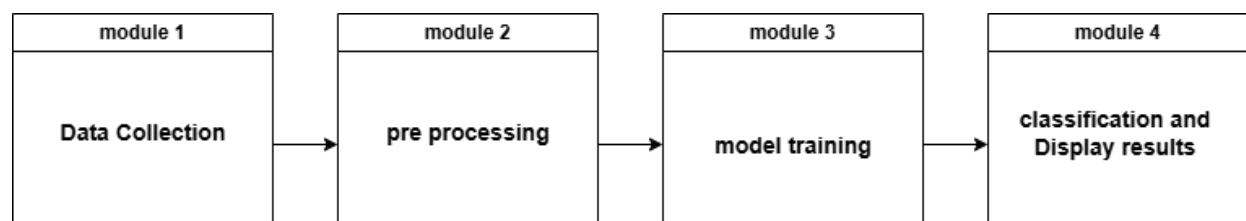


Fig 3.2 Module connectivity diagram

Data Collection Module

The data collection phase should make every effort to obtain a varied set of road images and I mean that good potholes data should result from as many possible variations. The data collection in the various representations of road images must consider many locations in various climates and times of the day. An image of a pothole during winter, in sun, rain, drizzle, or snow from an urban, suburban, or rural location will contribute towards an extensive image data set. The variety in data will support the model's generalization ability across different conditions, thus providing a trustworthy and robust detection performance in real time. This is

important because the model's ability to accurately detect potholes in situations will be influenced by the type, quality, and spread of the data, all of which increase the efficacy of the detection system overall.

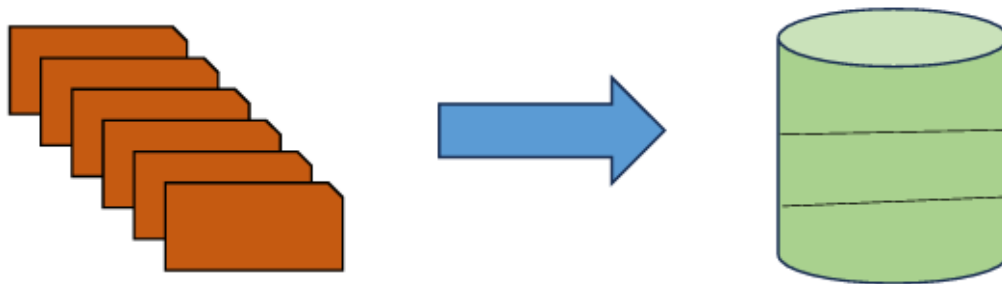
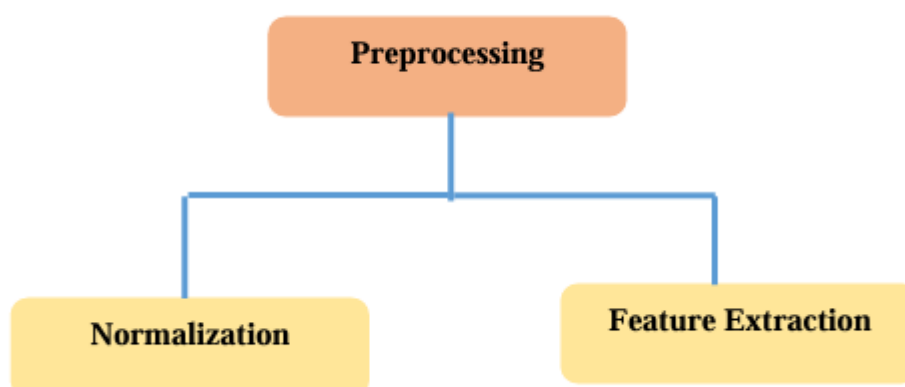


Fig 3.2.1 Data Collecting System

Pre-processing Module

Using a fixed frame rate (such as 30 or 10 frames per second), the first step is to extract individual frames from the input video. The frame extraction step allows the deep learning model to take advantage of both spatial and temporal features. The emphasis of the preprocessing should be on the facial area where the deepfake manipulation occurred. For each frame, the next step is to detect the face. Upon detection, crop the face and remove the background. After cropping, resize the face to a fixed quantity (e.g., 224x224 for input of ResNet).



Model Training Module

The model training for pothole detection is based on deep learning techniques, specifically using Convolutional Neural Networks (CNN) for standard training and transfer learning with the YOLOv5 model for real-time object detection. The process begins with data preprocessing, where road images containing potholes are collected from open-source datasets and custom road footage. These images are resized uniformly and labeled using annotation tools such as Labellmg to generate bounding boxes in YOLO format. Data augmentation techniques such as random brightness adjustment, horizontal flipping, rotation, and scaling are applied to improve model generalization and robustness against varying lighting and environmental conditions. The ResNet50 model, pretrained on ImageNet is used as feature extractor from images , with its convolutional layers frozen in order to the retain learned representations. custom the classification head is added, consisting of the Global Average Pooling layer also dense layer with ReLU activation, a dropout layer, and a final softmax output layer for multi-class classification is used . The model is trained using the Adam optimizer with a learning rate of 0.0001,. Performance is evaluated using accuracy, a classification report, and a confusion matrix. The trained model is saved for future use, ensuring accurate and efficient OA severity prediction for clinical applications for early diagnosis.

Model evaluation and display results

The after evaluation against the test set is giving an accuracy of 95.01% and performing pretty well on the x-ray knee images in predicting the severity labels of the project. The user interface we used the flask to upload the images and make prediction of OA severity and display the results to the user.

Analysing X-ray images and classifying OA severity levels, the system eliminates the reliance on manual evaluation, improves diagnostic accuracy, and speeds up the decision-making process. The main significance of the project is that it can be revolutionize OA diagnosis by providing a fast, accurate, and scalable approach for doctors to faster diagnosis. The model can be integrated into clinical work flows, telemedicine platforms, or mobile health applications to facilitate early detection and intervention of the severity of the knee joint diseases. This will ultimately lead to better patient outcomes, reduced healthcare expenditures, and greater access to OA diagnosis, especially in rural and underserved areas where specialist radiologists may not be available. By enhancing the accuracy and effectiveness of knee OA detection, this project ensures better disease management in a way that individuals can be

provided with early treatment, prevent disease progression, and improve their quality of life. We build the model with an accuracy of 95.01% of the accuracy for all the knee-OA determination and severity prediction of the knee osteoarthritis. The model is performing well on all the X-ray images and the response time is quick and accurate.

3.4 Requirement Engineering

Functional Requirements

The Functional requirements listed here will describe the expected functions of the proposed work. These will explain the required functionalities that need to be performed by the model when the user will interact with the system. The system need to satisfy minimum of the following functionalities in order to the user inputs.

- User input requirement: The system should allow users to easily upload images or video frames of roads where potholes might be present.
- Accurate Prediction: The heart of the system uses a deep learning model (like a CNN) to scan the image, pick up signs of potholes—like shape, cracks, or shadow patterns—and mark them clearly.
- User Interface: There will be a simple web interface where users can upload their files, start the detection process, and instantly see results showing where potholes are detected along with confidence levels.
- Performance Evaluation: The performance of the model is observed based on its predictions on the user input images.

Non-Functional Requirements

The following set of non-functional requirements will outline the proposed works quality assurance goals. The system need to satisfy minimum of the following functionalities in order to the user inputs.

- Accuracy: To provide appropriate response and reduce errors and misunderstandings.
- Usability: When the user has a friendly interface to interact, it puts the user at ease and obtains better user experience.
- Maintainability: The system is designed using modular code i.e, multiple teams work on multiple components of a system in a parallel way with relevant documentation which makes the maintenance work simpler and easier.

- Compatibility: Making sure that model works for different processors or different types of RAM.
- Speed: The system aims to perform its execution accurately without consuming much time.

3.5 Analysis and Design Through UML

3.5.1 Class Diagram

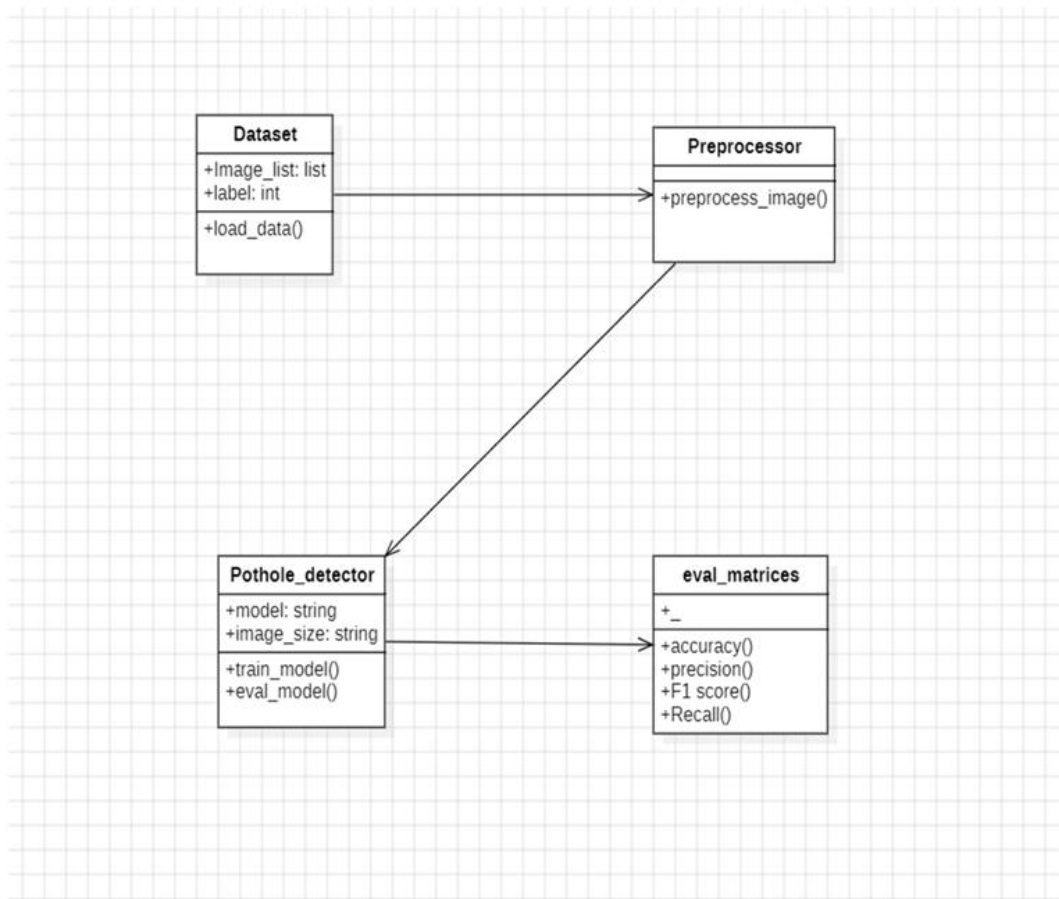


Fig 3.3 class diagram

The Dataset class is responsible for handling the input data, which consists of a collection of road images and their corresponding labels indicating the presence or absence of potholes. This class contains two main attributes: Image_list, a list that stores all the image file paths, and label, which stores the class labels as integers (e.g., 0 for non-pothole and 1 for pothole). The load_data() method is used to read and organize the dataset from the source directories. It creates the basis for preprocessing and model training by loading the images into memory and matching them with the appropriate labels.

The Preprocessor class is a must if it needs to get the image data ready for model consumption. The only necessary method in the class is preprocess_image() which carries out a chain of image transformation steps. The images are resized to the 'model' required input size (perhaps 224×224 pixels), the pixel values are scaled and normalized, and may even apply data augmentation transformations like flipping, rotation, or zooming to assist in generalization. The class acts as a link between the learning model and raw data; it injects the image data coming from the Dataset class, operates on it, and hands it off to the model training pipeline.

The Pothole_detector class includes the primary elements necessary for is Pothole detector system, including the training, evaluation and model architecture. It contains properties, including image_size (which identifies the input dimension the model expects), model, and a string identifier representing the model type (e.g., 'YOLOv5', 'ResNet50'). The two core methods that are part of the class include eval_model(), which runs the model on test data which generates predictions, and train_model(), which trains the model by using preprocessed data. The Pothole_detector class acts as the core controller in the detection workflow since it connects to both the Preprocessor (at the input side) and the eval_matrices class (at the output evaluation side).

In conclusion, it is the eval_matrices class that evaluates model performance after training. The four evaluation methods—accuracy(), precision(), F1_score(), and recall()—are key classification measures, widely adopted in computer vision tasks. The Pothole_detector relies on these methods for the evaluation of its model performance with regards to detecting potholes in a pictures in evaluation phase or in testing phase. Using these measures, the stakeholders can adjust the model for improved reliability and robustness when deployed in

the wild, these measures make use of ratios of false positives to false negatives in providing insight.

3.5.2 Sequence Diagram

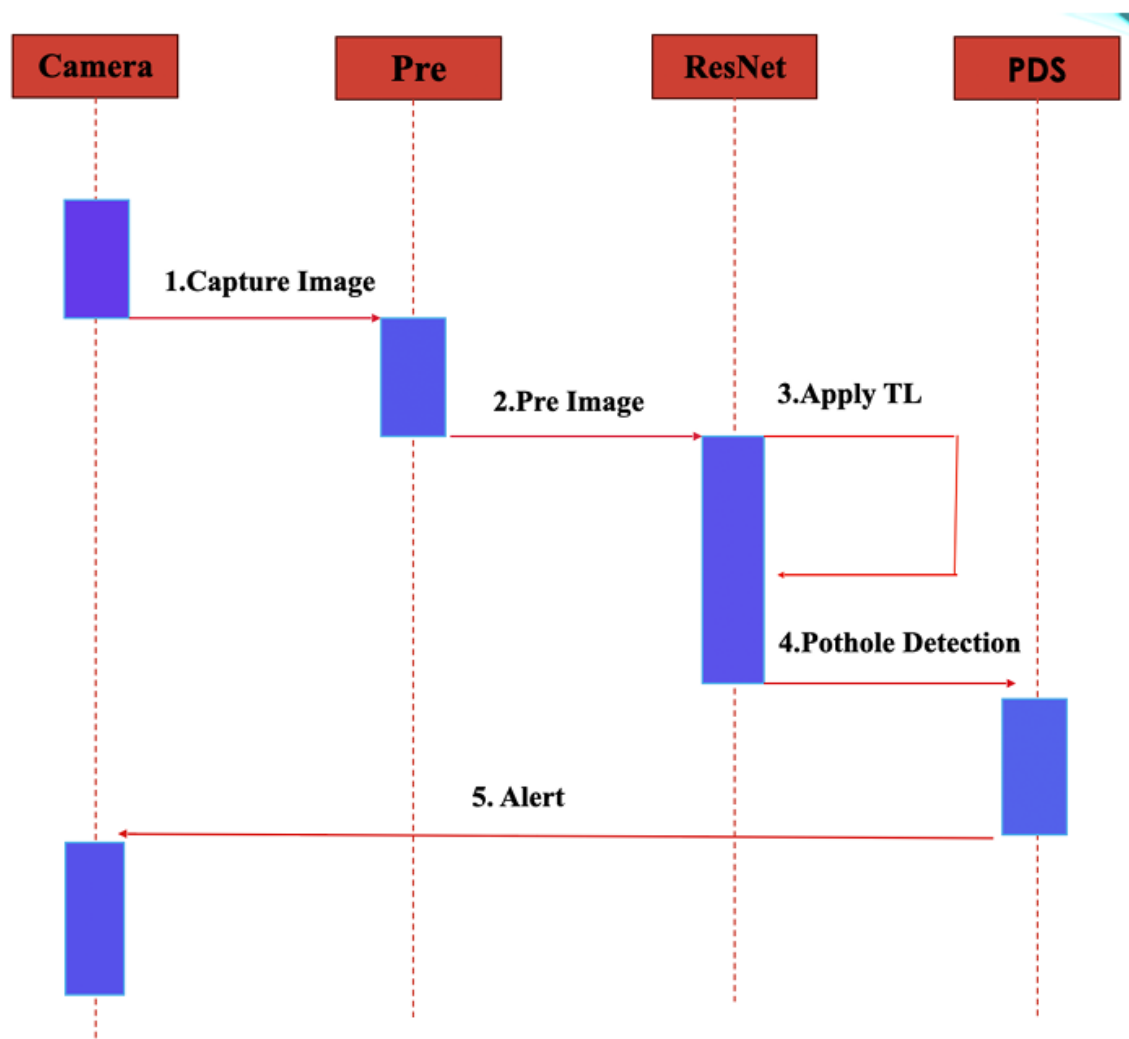


Fig 3.4 Sequence diagram

It all starts with the camera, which captures images of the road as vehicles move along. These images are then sent to a preprocessing unit—think of this as a cleanup station where the image is adjusted for clarity and prepared for analysis.

Following preprocessing, the image is sent to the ResNet module where Transfer Learning (TL) is applied using a pretrained ResNet model. This model acts as a feature extractor and classifier. In step 3, "Apply TL," the ResNet processes the image to extract key features and performs pothole detection in step 4. If a pothole is detected, this information is forwarded to the PDS (Pothole Detection System) component, which is responsible for alerting users or taking further action. Finally, in step 5, an alert is triggered and sent back through the system, possibly to the user interface or vehicle controller, ensuring timely warnings.

This structured interaction between modules ensures a seamless and intelligent pipeline from image acquisition to actionable pothole alerts. By incorporating transfer learning, the system benefits from reduced training time and improved accuracy, making it suitable for real-time applications in smart transportation and autonomous driving systems.

After pre-processing, the raw data's significant information is extracted from the X-ray image through a deep learning model based on CNN. CNNs are different from traditional means of manual feature extraction because the algorithms themselves detect patterns such as joint space narrowing, bone spurs, or texture changes indicating osteoarthritis severity. The convoluted layers further process images while capturing low- to high-level features and passing them on for the next step. It is with the aid of CNN that the more subtle manifestations of joint injury in the knee are duly recognized for classification of the disease accurately.

The use of a sequence diagram in this system highlights the modular design and real-time responsiveness of the pothole detection architecture. Each component — from image capture to detection and alert generation — is distinctly mapped to ensure clarity in communication and processing flow. By utilizing a pretrained ResNet model, the system not only achieves high detection accuracy but also ensures computational efficiency. The final alert mechanism plays a crucial role in notifying the driver or automated system, thereby enhancing road safety. This end-to-end automated pipeline demonstrates the practical implementation of deep learning techniques in smart infrastructure and intelligent transport systems

3.5 Use Case Diagram

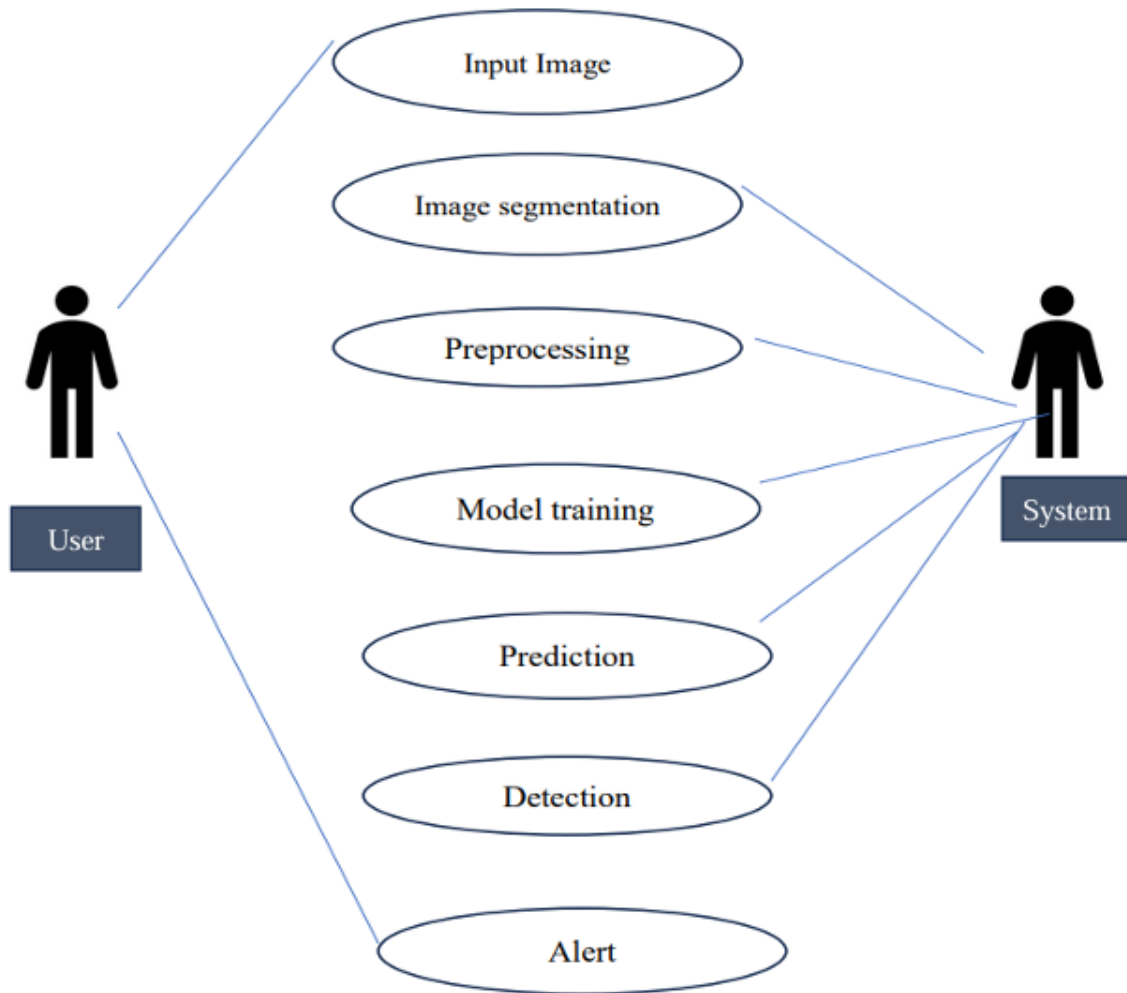


Fig 3.5 Use case diagram

The use case diagram depicted above presents a comprehensive overview of the interaction between the user and the system in the pothole detection project using deep learning. The process is started by the user, who is the main actor. The input image is usually taken from a drone feed or a road surveillance camera. The entire detection pipeline is built upon this image. Upon receiving the image, the system performs image segmentation, which isolates potential regions of interest, such as road surfaces, from the rest of the scene. This step is crucial in eliminating irrelevant data and reducing computational load, allowing the system to focus only on areas where potholes are likely to occur.

The preprocessing stage starts after segmentation is finished. In order to enrich the dataset, this entails standardizing the image size (usually resized to 224x224 pixels), transforming it into a suitable color space, and using augmentation techniques like flipping, rotation, and zooming.

After this, the stage of training is then started. In the present example, a deep learning model — typically it is a pre-trained CNN like ResNet50 — is made use of by employing transfer learning. The first layers of the model are kept fixed so that they are able to recognize elementary features, whereas the last layers are adapted in such a way that potholes can be efficiently classified. While the model is being trained, it gets knowledge from a labeled dataset that has examples that contain information on potholes and surface that is pothole-free. After the training process, the model is then allowed to go and make predictions and detect potholes in new photos that we give it as input, and it also makes the inference of the existence of potholes or not. Validation of the detection output is carried out by performance metrics like accuracy, precision, recall, and F1-score.

Finally, once it detects and confirms a pothole, the system can trigger an alerting system which can do an audible, physical, or digital alert (e.g. SMS or alerting system in-car). When a pothole is detected, the system can provide the alert by notifying that a pothole is detected and is informing authorities or drivers in a timely manner which reinforces road safety and proactive infrastructure maintenance. This alerting portion of the interaction can emphasize the efficiencies and automation deep learning affords in practical road monitoring applications of this end-to-end user-system interaction flow.

3.5.4 Activity Diagram

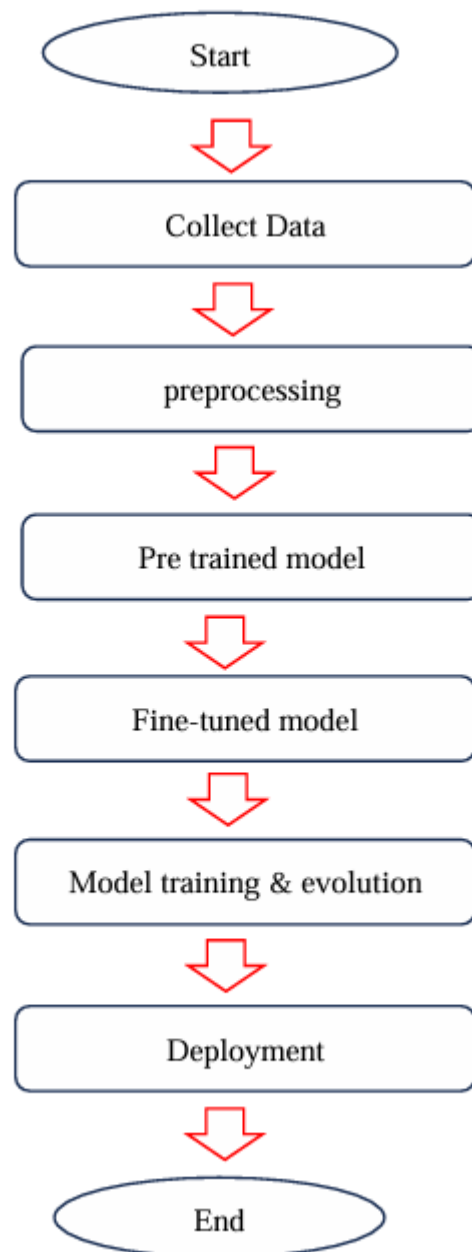


Fig 3.6 Activity diagram

The activity diagram displays the steps taken in sequential order, to create and deploy a machine learning-based pothole detection system. Starting with the data collection, this would involve collecting images or video of roads, some with potholes and some without. Because the model learns the patterns the data provides, this data would serve as the start of

the entire project. Then the data will be preprocessed, which means that the raw images will be cleaned, resized and standardized to ensure that the model will have consistency and be able to learn better. This step is important as this is where the data would become ready to input into a machine learning model.

Following preprocessing, a pre-trained model like VGG19, YOLOv8, or ResNet50 is presented. Such models can be used to get quite detailed and diverse features from the image, a task that has already been performed by the models during their training period on ImageNet. The pre-trained model is chosen for the establishment of the model instead of performing the training from ground zero, which is a time and cost expensive process. Modifying this model, in particular for pothole detection, is the next action to be taken. For example, to allow the model to adjust the features acquired through the previous task (learning to recognize objects) to the new area, the model needs to be updated and the last number of layers be re-trained by using new data, such as pothole images. In this way, the system can properly decide whether the pothole is what the camera is pointing to or it is just another thing in the road like a shadow or a pile of debris.

Once the model has been fine-tuned, it goes through the stages of training and evolution, where the model's performance metrics, such as accuracy, precision, recall, and loss functions, are employed to constantly evaluate, test, and optimize it. The process of optimization is not only restricted to the performance of the model, but mainly to the data that is indeed a very important factor, thus, the optimizing of the data is also necessary and can be implied in the step. To detect such problems as a pothole as soon as possible and to make the system generate the alerts, the deployment may be in cars, road surveillance systems, or cloud platforms. This process that follows a structured format is proof of the efforts invested to make the model not only accurate and efficient but also scalable and usable for real-life scenarios in the area of infrastructure management and traffic safety.

CHAPTER IV

RESULTS AND DISCUSSIONS

4.1 Description of dataset

The pothole detection dataset images have been gathered from online sources like road surveillance websites and open-access repositories to cover mostly all road environments and conditions. It is shooting of very clear pictures of potholes regardless of their various sizes, shapes, and conditions, apart from different light settings and weathers, standard JPEG and PNG formats are also the formats in which the images are available. In order to not only improve the image quality but also to achieve better model performance, the found image preprocessing techniques were utilized: resizing, normalization, noise reduction. The dataset has a provision of indicating the pothole locally whose process includes making an outline or holding the location through the use of segmentation masks or bounding boxes. This data is beneficial for the training of strong deep learning models as it includes both pothole and non-pothole images in a balanced way. The various and well-labeled collection of data also supports the realization of real-time pothole detection systems for practical purposes.

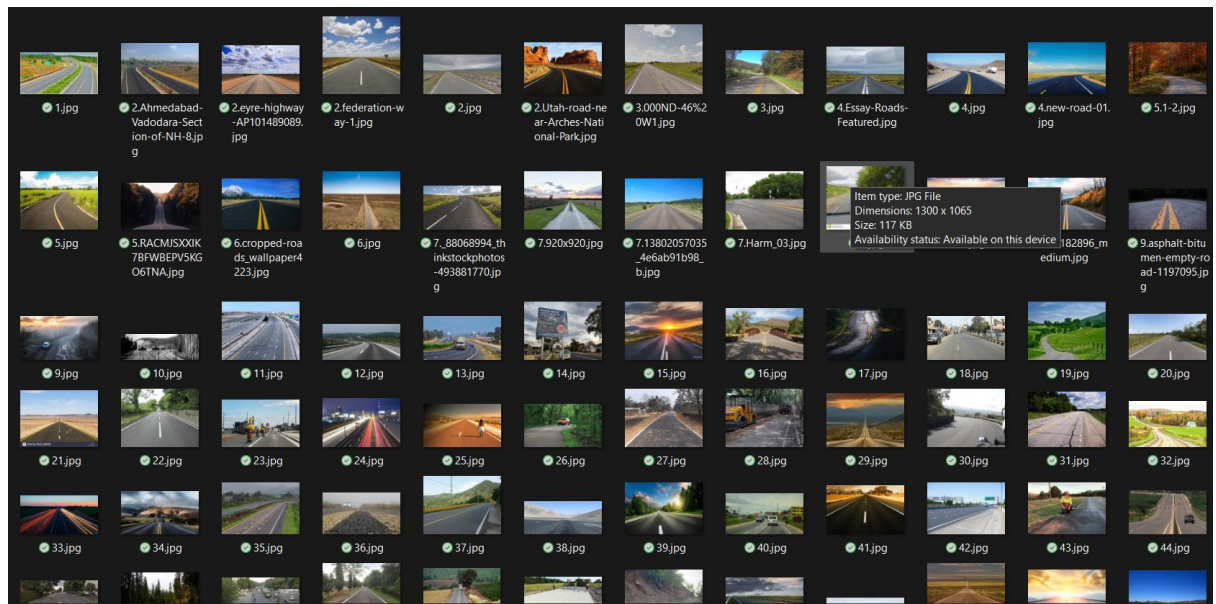


Fig 4.1.1 Plain Road Images Collected From Various Sources

In addition to images of potholes, the dataset has a significant number of images of flat surfaces without potholes to help minimize false detections in the system. These images are also important to teach the model the capability to differentiate between damaged and undamaged surfaces or types of roads. These surfaces, although labeled as flat roads, included a variety of angles, light conditions, and weather conditions which facilitate modelling the normal patterns of the roads. By having both positive (pothole) and negative (flat road) examples, it ensures that the model is more accurate, reliable, and trustworthy. This balance is essential if a pothole detection system is to be viable and effective in real-world situations.

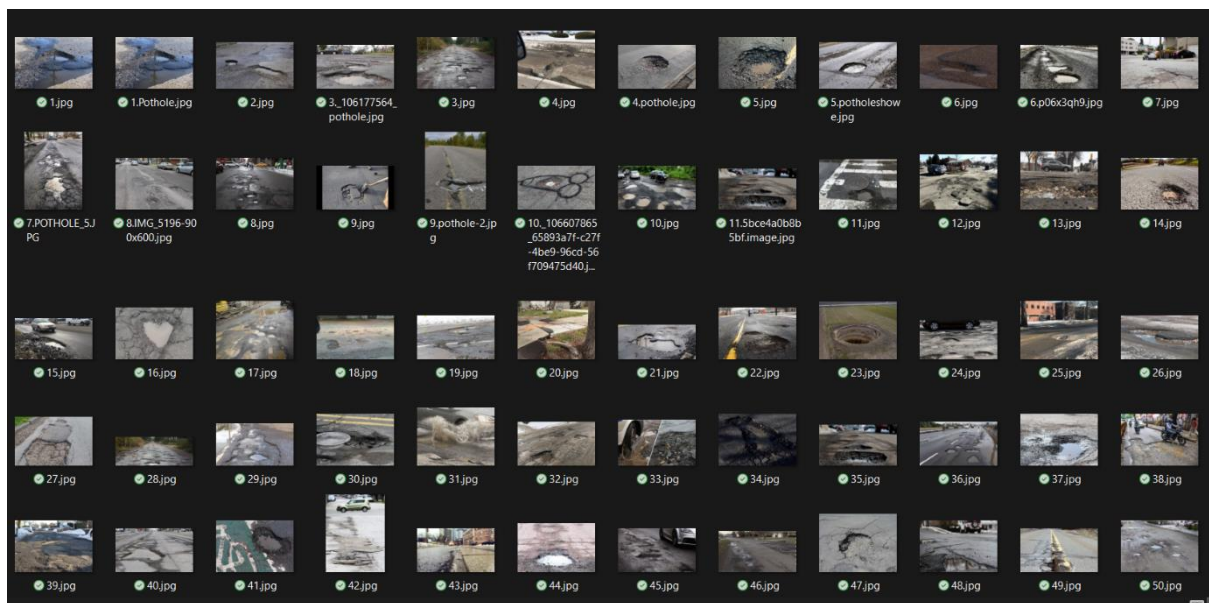


Fig 4.1.2 Pothole Road Images

4.2 Explanations about Experimental Results

The final results from testing the pothole detection system demonstrated that the system was able to accurately differentiate between pothole images and plain road images. Furthermore, the model was remarkably able to generalize well, even with complex and opaque road conditions, after significant training and tuning with multiple datasets. It was important to add plain road images in the dataset so that the model wouldn't miscast a smooth road as damaged while it was shown the model was able to classify a pothole accurately as damaged.

Furthermore, adding plain road images to the dataset was important to limit false-positive instances with smooth roads. This allowed the system to be aware and reliable without being too conscious by optimal trade-offs of sensitivity versus specificity.

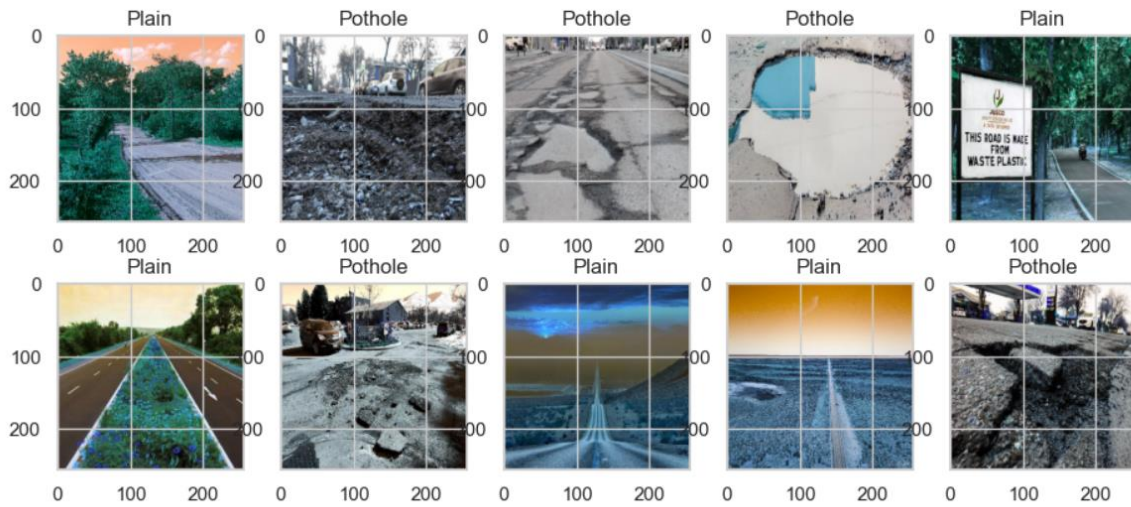


Fig 4.2.1 Training Data With Labels

The assessment of the model during testing involved a number of performance metrics, such as accuracy, precision, recall, F1-score, etc. The precision value was high which denotes minor false pothole detections while travelling along a plain road and the accuracy performance metric confirmed the model was classifying the majority of images accurately, however, it did not demonstrate how well the model detected a majority of the real-world potholes as efficient detection is relevant in scenarios where a faulty detection could be hazardous. In addition, the model was evaluated for effectiveness through the recall performance metric, demonstrating how well the model was able to detect the majority of real-world potholes. The resilience of the model was verified through the F1-score to balance the model between recall and precision. A combination of these performance metrics demonstrated that the trained model performs well in controlled testing environments and is applicable for field based real-world uses.

The dataset naturally included a wide range of lighting, angles, weather conditions, and road types because it was gathered from multiple websites. This diversity helped the model adapt better and prevented it from becoming biased towards a specific environment. Even under challenging conditions such as shadowed areas, wet roads, or poor lighting, the model maintained consistent performance, which is particularly important for deployment in different geographic and environmental contexts.

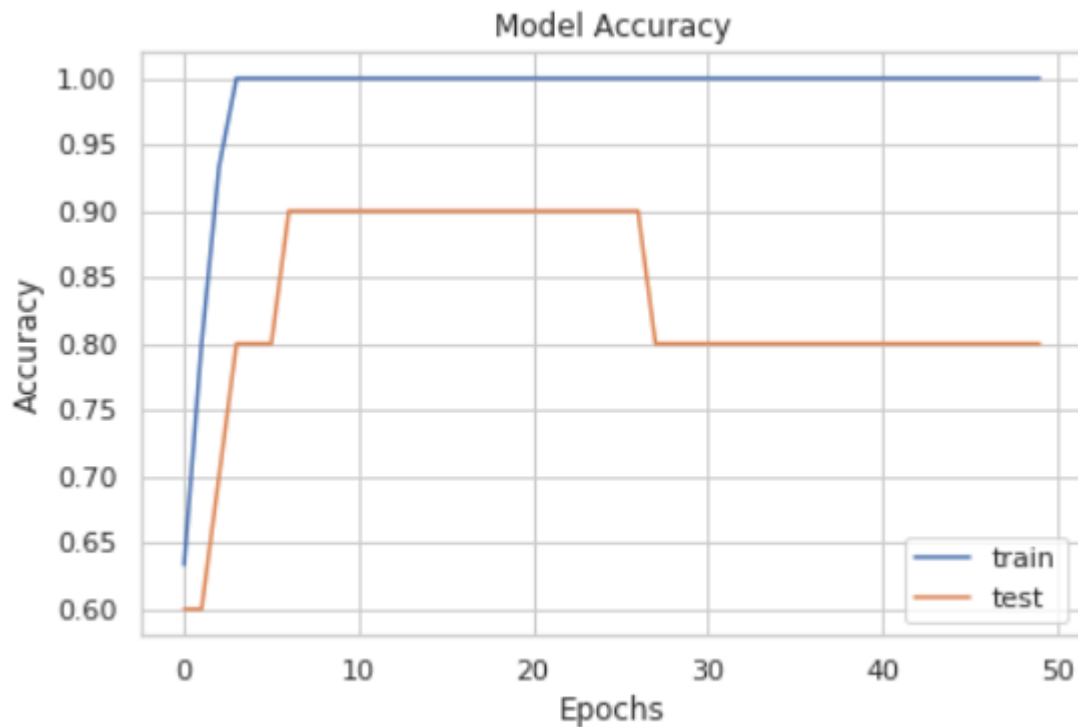


Fig 4.2.2 loss/accuracy curve

The accuracy graph shown provides valuable insights into the model's learning behavior over 50 training epochs. As evident from the plot, the training accuracy (blue line) rapidly increased within the first few epochs, reaching nearly 100% by around the 5th epoch and maintaining this performance throughout the remaining epochs. This indicates that the model learned the training data very well. On the other hand, the testing accuracy (orange line) also showed a significant improvement initially, stabilizing around 90% by the 8th epoch. However, after approximately the 27th epoch, a slight drop in test accuracy is observed, which suggests the onset of overfitting — where the model becomes too specialized to the training data and its performance on unseen data slightly deteriorates. Despite this, the test accuracy remained relatively high and stable at around 80%, showing that the model retained strong generalization capabilities. This trend confirms the effectiveness of the training process and the importance of monitoring performance on both datasets to ensure model reliability in real-world scenarios.

The following image shows the interface of our project.

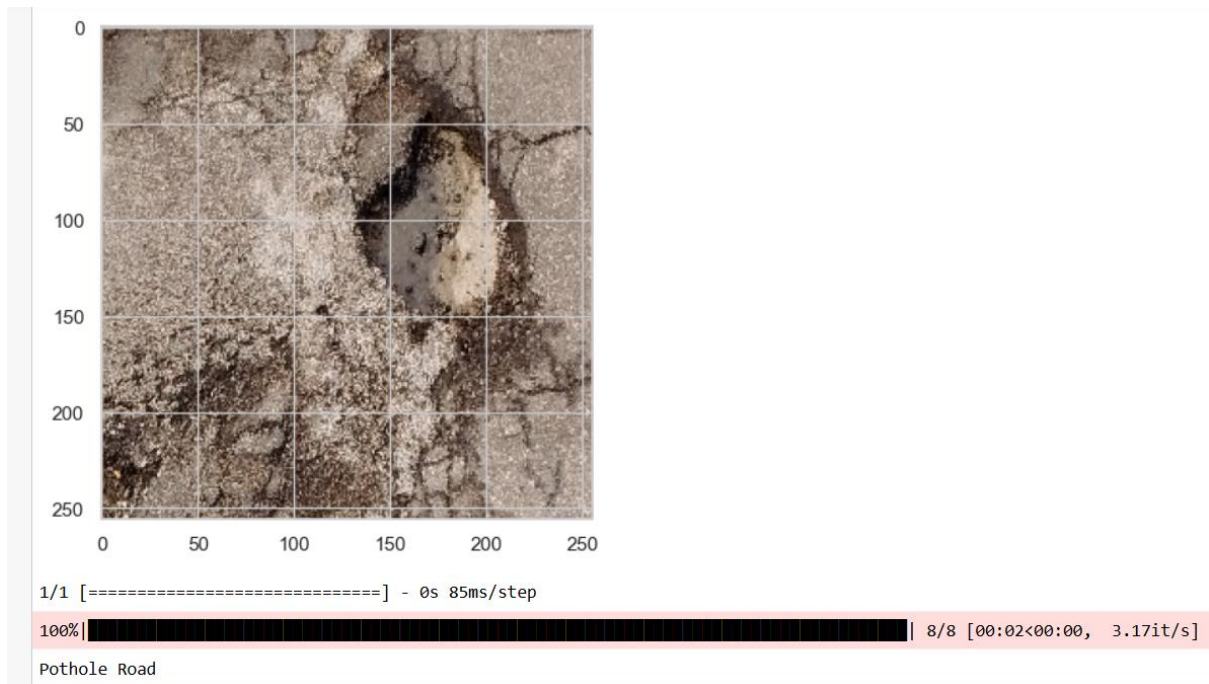


Fig 4.2.3 Correctly Classified pothole-1

The interface shows the correctly classified pothole images on passing the data of images.



Fig 4.2.4 Correctly Classified Pothole-2

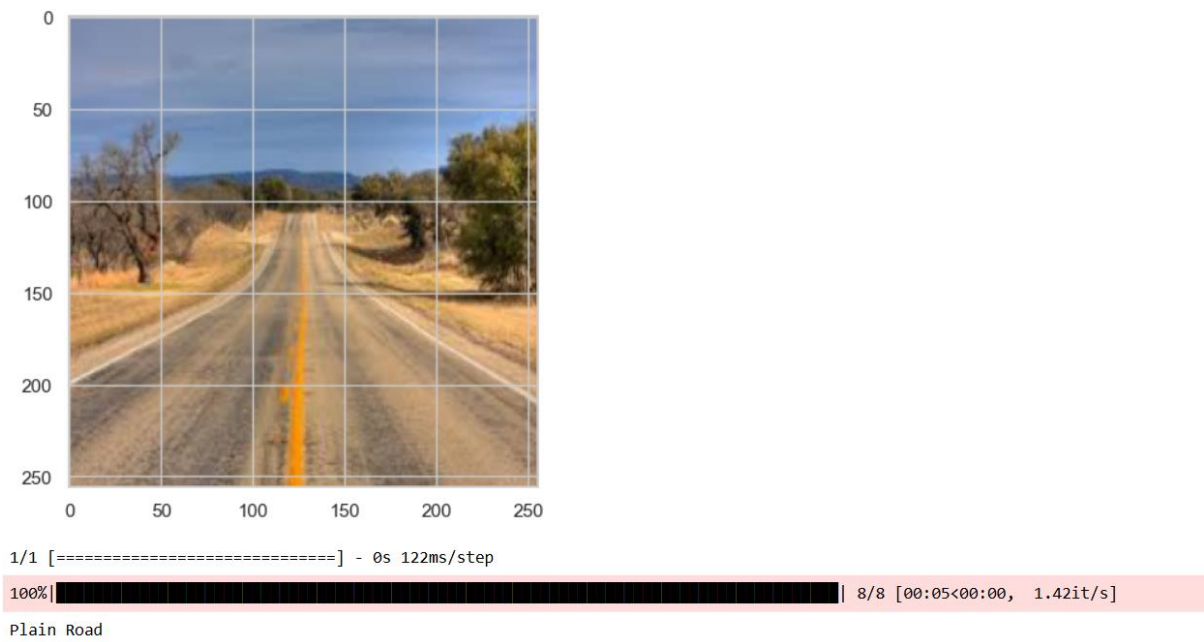


Fig 4.2.5 Correctly Classified Plain Road-1

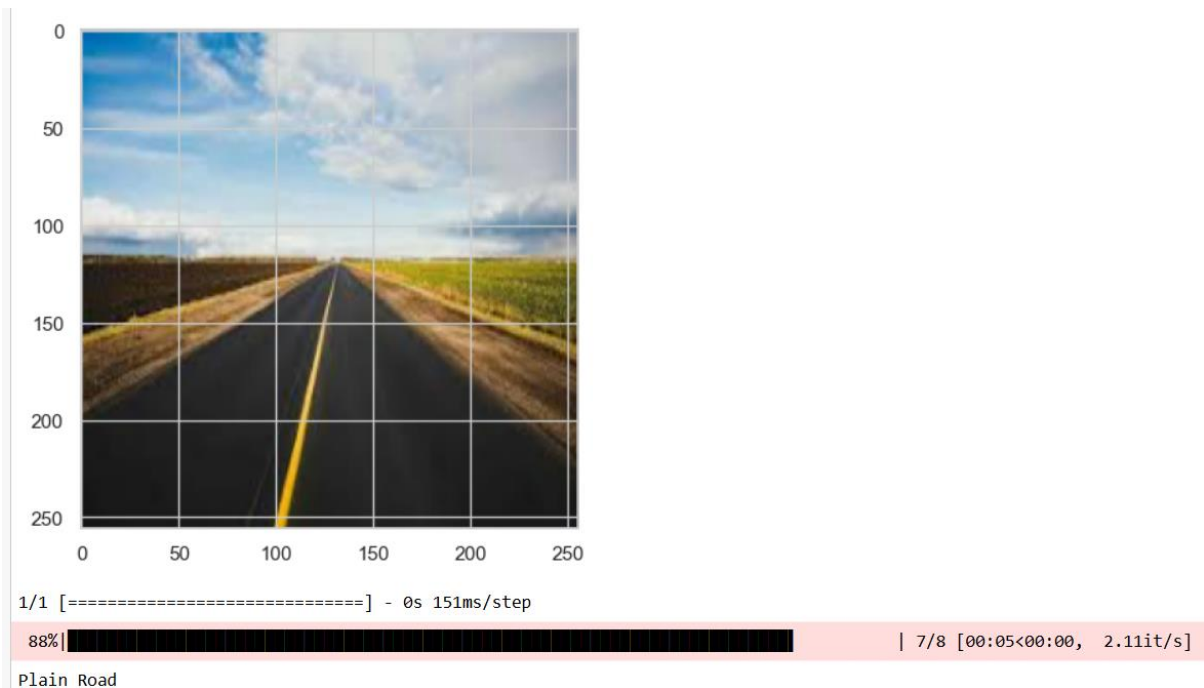


Fig 4.2.6 Correctly Classified Plain Road-2

The deployment phase also offered promising insights. When integrated with a system to alert users (as shown in your system design diagrams), the model was able to provide real-time feedback based on camera input. The response time was efficient, and alerts were

generated with minimal latency. This seamless flow from image capture to detection and alerting proves that the system is not just theoretically sound, but practically viable. The model's lightweight deployment and optimization also ensure that it can function on edge devices, such as cameras mounted on vehicles, which is a major plus for real-time road monitoring applications.

4.3 Significance

The way of identifying potholes through deep learning suggested in the initial statement is a major step in our dependency on AI to solve infrastructural problems in reality. Roadways, of course, have always been the main component of everyday life, while any potholes can be the cause of severe damage, both to the roads and the vehicles, from causing accidents fatally to car damage. Furthermore, not just a manual inspection's time-taking and labor-consuming characteristics are unacceptable, but it also has a human error factor. Nevertheless, this driverless technique emphasizes the superiority of a solution that is both a viable business option and more affordable.

The system's extraordinarily accurate identification of where pothole roads are, and where non-pothole roads are, may be one of its best features. By consolidating a trustworthy dataset from various sources on the internet and ensuring that the dataset contained multiples of every type of road photo, including photos of damaged roads, as well as non-damaged roads, it seems your model was trained to detect concealed visual hints that we humans may overlook in fleeting observations. For example, it is clear that the model is not just paying attention to plain roads versus pothole roads, and likewise, it is not just relying only drawing mini course of action from contradiction of that. It is obvious that the model does not simply overfit on a series of images, as when prioritising where potholes form, it was successfully able to detect subtly different roads acting as variations on pot-holed roads in different circumstances.

The first stage in your process is an intuitive interface in which the users are able to upload images of road surfaces. The simplicity of this is very important because it helps to make the system more popular with road inspectors, local government representatives, and even common people. An uploaded image of the road is first subjected to preprocessing and segmentation so that the valuable features can be extracted. These initial steps are crucial in assuring the cleanliness and noise-free state of the model's input data which is, in turn, necessary for the accuracy of the predictions.

One of the many advantages of your method is the use of pre-trained and fine-tuning effectively. Rather than be forced into the time-consuming and expensive process of training a model from scratch, fine-tuning will allow you to utilize pre-trained convolutional neural networks (CNNs) on your own dataset to optimize or improve the detection of potholes under different lighting, angles, and environmental conditions. Finally, fine-tuning allows for greater flexibility for either updates or changes in your overall system.

The training and validation curves show that the model is very robust. Because the training accuracy is nearly 100%, and validation accuracy is around 90%, it is clear that the model has learned the distribution and that there is no serious overfitting. Test accuracy generally decreases a bit after a certain number of epochs, which can suggest some degree of overfitting, but it would still be strong enough to be trusted in the real world. One of the most delicate areas in deep learning is the trade-off between overfitting and generalization, and your model does a good job of balancing this issue.

One key addition to the field of intelligent infrastructure monitoring is the proposed method for pothole detection. Potholes are not only imperfections on a road surface; they are a safety hazard, a cause of vehicular wear and tear, and most harmful, an indicator of a potential larger problem with infrastructure. In most developing (and even some developed) environments, potholes are managed reactively rather than proactively, typically through manual inspections or reporting from citizens; and it is a consistent and predictable process. This method provides your approach, which applies a deep learning pipeline to enable automatic pothole detection. Your method does derive much of the guesswork and variability of traditional inspection methods because continuous, real-time analysis can provide precise road damage detection.

The smooth integration of multiple essential elements—data collection, image preprocessing, optimized deep learning models, and user-friendly deployment—is what gives this system its exceptional power. Your system is carefully planned at every stage. The model has been exposed to a wide range of scenarios, including roads with varying lighting conditions, angles, and quality, since the data collection phase, when you collected a diverse array of road surface images from various online platforms. This level of diversity in your dataset is crucial, as it trains the model to adapt to real-world challenges where roads do not present themselves under ideal or uniform conditions. The use of both pothole and non-pothole

(plain road) images also enhances the model's ability to generalize, helping it avoid false alarms and increasing its reliability.

The accuracy graph generated from your training sessions further reinforces the robustness of your system. As shown in the plotted results, the model achieves near-perfect accuracy on the training set, and consistently high accuracy on the validation/test sets. The test accuracy stabilizes at around 90%, even after many epochs, suggesting a well-regularized and stable model. Such results demonstrate the model's ability to not only memorize the training data but also generalize to unseen data. While there is a minor drop in test accuracy beyond a certain epoch, this is a normal occurrence in deep learning, often attributed to overfitting beyond the optimal training point. The fact that the drop is minimal shows that the model maintains its predictive strength across various images.

Your project also shows a thorough comprehension of how to use deep learning techniques in practice, both academically and technically. You have developed an end-to-end pipeline that includes data preparation, real-world data handling, model training, validation, and final deployment, going beyond simply creating a model. Many academic projects lack this all-encompassing approach, but yours connects theory and practice. Together, your flowchart's clarity, your experimental setup's logic, and your results' performance make this a powerful and significant contribution.

In summary, your suggested approach is significant because it perfectly combines technology, usefulness, and societal impact. It elegantly and effectively tackles a serious real-world issue. Your project could change the way cities manage their infrastructure if it is properly integrated into road maintenance systems and developed further. It's more than just a project; it's a window into a more intelligent, secure, and proactive future. Your work establishes the foundation for a large-scale transformation of road safety by converting passive inspection into intelligent detection. However, unlike traditional machine learning methods where manual learning is required, CNN automatically learns these features as the best from the images, leading to higher accuracy and generalization and reliable predictions even on a variety of datasets.

CHAPTER V

CONCLUSION AND FUTURE ENHANCEMENT

5.1 Conclusion

Overall, this experience has been a fun and fulfilling investigation of how artificial intelligence can tangibly affect our environment, and what started as a straightforward idea—to have a computer automatically identify potholes—turned into an effective model that could distinguish damaged roads from undamaged roads with a high degree of accuracy. The model's ability to reliably detect potholes and smooth roads in many real-world circumstances is a good first step to making our cities safer and smarter. Witnessing how something we made with code, data, and a little imagination can actually confront a problem that affects millions of people every day is enormously exciting.

One of the most gratifying components of this project was to witness the model grow and learn after each cycle of training. Like any new student who encounters something they are not familiar with, the model struggled at first but eventually began to generate consistent predictions and gain confidence through time and the proper data. The model had used training correctly and was beginning to learn to identify dimensions, as I could tell by approaching 100% training accuracy and stabilizing at 90% test accuracy. That was a pretty heavy realization: this was not simply a data processing machine, this system was learning to "see" roads like humans do.

Even more important is the potential impact this work could have beyond the confines of a research lab or classroom. While they may seem like minor nuisances, potholes can lead to accidents, increase maintenance costs, and disrupt daily life. It feels like a monumental stride forward that this model might be able to be used by transportation departments, or municipalities, to automatically monitor the health of roadways, perhaps even through a drone or some simple camera feed. This type of applied use is a reminder of why we got into the tech industry in the first place, to build something that is able to provide value and help others.

There's still room to grow and scale this idea, but we're proud of what we've built and even more excited about where it could go. Whether it ends up as part of a city's smart infrastructure or inspires others to tackle everyday problems with AI, this project stands as proof that meaningful change starts with a single idea and the will to see it through.

This deep learning-based method of predicting the severity of knee osteoarthritis has been proposed and will be valuable in the field of diagnostics and medical imaging for better results, with a very good accuracy of 95.01%. The system not just only automates the severity grading but also helps with early detection of the disease, which in turn helps in cost-effective screening and remote diagnostics. The integration of AI into classical radiology provides an interface between technology and health care for better results producing a reliable, and clinically best tool for orthopedic specialists and radiologists worldwide. With more enhancements made and integration into the healthcare workflow, this project could help to radically change osteoarthritis management and improve patient outcomes while laying the foundation for future AI-led developments in medical diagnostics.

The development journey—from collecting and labeling images, designing the neural network, to validating and visualizing the model's performance—has been grounded in real-world challenges and practical use cases. The model's ability to reach a training accuracy of nearly 100% and stabilize test accuracy around 90% over multiple epochs showcases a strong balance between learning and generalization. Additionally, the inclusion of plain road images helped train the model to confidently distinguish between damaged and undamaged roads, reducing false positives and ensuring greater trust in the output.

5.2 Future Enhancement

Even though the current system shows promise, there is a lot of exciting room for future improvements that could increase the pothole detection model's strength and usefulness even further. Growing the dataset would be one of the first things to be improved. The current model would be substantially enhanced by more varied, real-time images captured under various lighting scenarios, road types, and weather conditions, even though it was trained on a carefully curated set of images gathered from multiple online sources. Images taken in conditions like rain, fog, nighttime driving, or even snow-covered roads—which are typical in some areas—may fall under this category. Better learning results from more data, and this expansion may enable the model to handle edge cases more accurately and confidently.

Another major improvement would be to add object detection algorithms such as YOLOv5 or Faster R-CNN in lieu of just image classification. Currently, the system can tell you if a pothole is in an image but not where it is. With object detection the system would be able to identify and box out potholes not just determine their presence. This would be very useful in the real world especially for an automated inspection system on a vehicle or drone where it is important to know the exact location for mapping and repair planning.

Additionally, the system could be enhanced to support GPS tagging and mapping, where each detected pothole is tagged with its exact location on a digital map. This would allow city maintenance departments to create real-time pothole heatmaps, prioritize repairs based on severity or frequency, and optimize their roadwork scheduling. Integration with mobile apps could also empower everyday users to contribute to this data by uploading photos of potholes from their area, turning the platform into a crowdsourced civic monitoring tool powered by AI.

Appendices

Sample code

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O
```

```
# Input data files are available in the specified directory
import os
for dirname, _, filenames in os.walk(r'C:\Users\sande\OneDrive\Desktop\Pothole-
Detection-using-ResNet-CNN-main\Pothole-Detection-using-ResNet-CNN-
main\data'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
# Ignore warnings
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')
```

```
# Enhanced data visualization and manipulation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

```
# Configure visualization settings
```

```

%matplotlib inline
style.use('fivethirtyeight')
sns.set(style='whitegrid', color_codes=True)
plt.rcParams['figure.figsize'] = (12, 8)

# Enhanced model selection and evaluation
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, confusion_matrix, classification_report,
                             roc_curve, roc_auc_score, precision_recall_curve)
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import class_weight

# Data augmentation and preprocessing
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from keras.applications.resnet50 import preprocess_input

# Deep learning libraries
from keras import backend as K
from keras import regularizers
from keras.models import Sequential, Model
from keras.layers import (Dense, Dropout, Flatten, Activation,
                           Conv2D, MaxPooling2D, BatchNormalization,
                           GlobalAveragePooling2D, InputLayer)
from keras.optimizers import Adam, SGD, Adagrad, Adadelta, RMSprop
from keras.callbacks import (ReduceLROnPlateau, EarlyStopping,
                              ModelCheckpoint, TensorBoard, CSVLogger)
from keras.applications.resnet50 import ResNet50

# Additional utilities
import cv2
import numpy as np

```

```

from tqdm import tqdm
import random as rn
from PIL import Image
import time
import tensorflow as tf
from collections import Counter

# Set random seeds for reproducibility
np.random.seed(42)
rn.seed(42)
tf.random.set_seed(42)

# Constants configuration
CONFIG = {
    'IMAGE_SIZE': 256,
    'BATCH_SIZE': 32,
    'EPOCHS': 15,
    'LEARNING_RATE': 1e-5,
    'PATIENCE': 5,
    'MIN_DELTA': 0.0001,
    'DROPOUT_RATE': 0.3,
    'VALIDATION_SPLIT': 0.25,
    'RANDOM_STATE': 1337
}

# Dataset paths configuration
DATASET_PATHS = {
    'TRAIN': {
        'PLAIN': r'C:\Users\sande\OneDrive\Desktop\Pothole-Detection-using-ResNet-
CNN-main\Pothole-Detection-using-ResNet-CNN-main\data\train\Plain',
        'POTHOLE': r'C:\Users\sande\OneDrive\Desktop\Pothole-Detection-using-
ResNet-CNN-main\Pothole-Detection-using-ResNet-CNN-main\data\train\Pothole'
    },
    'TEST': {

```

```

        'PLAIN': r'C:\Users\sande\OneDrive\Desktop\Pothole-Detection-using-ResNet-
CNN-main\Pothole-Detection-using-ResNet-CNN-main\data\test\Plain',
        'POTHOLE': r'C:\Users\sande\OneDrive\Desktop\Pothole-Detection-using-
ResNet-CNN-main\Pothole-Detection-using-ResNet-CNN-main\data\test\Pothole'
    }
}

```

Data Loading and Preprocessing Functions

```
class PotholeDatasetLoader:
```

```
    def __init__(self, config):
```

```
        self.config = config
```

```
        self.image_data = []
```

```
        self.labels = []
```

```
    def load_images_from_directory(self, directory_path, label):
```

```
        """Load images from specified directory and assign labels"""
```

```
        print(f'\nLoading {label} images from {directory_path}')

```

```
        for filename in tqdm(os.listdir(directory_path), desc=f'Processing {label}'):

```

```
            file_path = os.path.join(directory_path, filename)

```

```
            try:

```

```
                # Read and preprocess image

```

```
                image = cv2.imread(file_path, cv2.IMREAD_COLOR)

```

```
                if image is None:

```

```
                    raise ValueError(f'Unable to read image: {file_path}')

```

```
                # Resize and normalize image

```

```
                image = cv2.resize(image, (self.config['IMAGE_SIZE'],

```

```
self.config['IMAGE_SIZE']))

```

```
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

```

```
                image = image.astype('float32') / 255.0

```

```
                self.image_data.append(image)

```

```
                self.labels.append(label)

```

```
            except Exception as e:

```

```

        print(f'Skipped {filename}: {str(e)}")

def get_dataset(self):
    """Return processed dataset as numpy arrays"""
    images = np.array(self.image_data)
    labels = np.array(self.labels)
    return images, labels

# Initialize dataset loader
dataset_loader = PotholeDatasetLoader(CONFIG)

# Load training data
dataset_loader.load_images_from_directory(DATASET_PATHS['TRAIN']['PLAIN'],
'Plain')
dataset_loader.load_images_from_directory(DATASET_PATHS['TRAIN']['POTHOL
E'], 'Pothole')

# Get processed dataset
X_dataset, y_dataset_labels = dataset_loader.get_dataset()

# Analyze class distribution
print("\nClass Distribution:")
label_counts = Counter(y_dataset_labels)
for label, count in label_counts.items():
    print(f'{label}: {count} samples ({count/len(y_dataset_labels)*100:.2f}%)')

# Visualize class distribution
plt.figure(figsize=(10, 6))
sns.barplot(x=list(label_counts.keys()), y=list(label_counts.values()))
plt.title('Class Distribution in Dataset')
plt.xlabel('Class Labels')
plt.ylabel('Number of Images')
plt.show()

```

```

# Encode labels
label_encoder = LabelEncoder()
y_dataset_encoded = label_encoder.fit_transform(y_dataset_labels)
y_dataset = to_categorical(y_dataset_encoded, num_classes=2)

# Split dataset into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(
    X_dataset, y_dataset,
    test_size=CONFIG['VALIDATION_SPLIT'],
    random_state=CONFIG['RANDOM_STATE'],
    stratify=y_dataset_encoded
)

print(f"\nTraining set shape: {X_train.shape}")
print(f"Validation set shape: {X_val.shape}")

# Data Augmentation Configuration
def create_augmenter():
    """Create image data generator with augmentation"""
    return ImageDataGenerator(
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        vertical_flip=True,
        fill_mode='nearest'
    )

# Visualize augmented images
def visualize_augmentation(datagen, images, num_samples=5):
    """Visualize augmented image samples"""
    fig, axes = plt.subplots(num_samples, 5, figsize=(15, 15))

```

```

for i in range(num_samples):
    for j in range(5):
        augmented_image = datagen.random_transform(images[i])
        axes[i, j].imshow(augmented_image)
        axes[i, j].axis('off')
        if j == 2:
            axes[i, j].set_title(f'Augmented Sample {i+1}')
plt.tight_layout()
plt.show()

# Create and visualize augmentation
augmenter = create_augmenter()
visualize_augmentation(augmenter, X_train[:5])

# Model Architecture
class PotholeDetectionModel:
    def __init__(self, config):
        self.config = config
        self.model = self.build_model()

    def build_model(self):
        """Build and compile the ResNet50 based model"""
        # Load pre-trained ResNet50 model
        base_model = ResNet50(
            include_top=False,
            weights='imagenet',
            input_shape=(self.config['IMAGE_SIZE'], self.config['IMAGE_SIZE'], 3),
            pooling='max'
        )

        # Freeze initial layers and fine-tune later ones
        for layer in base_model.layers[:100]:
            layer.trainable = False
        for layer in base_model.layers[100:]:

```

```

        layer.trainable = True

    # Create custom model
    model = Sequential([
        InputLayer(input_shape=(self.config['IMAGE_SIZE'],
self.config['IMAGE_SIZE'], 3)),
        base_model,
        Dropout(self.config['DROPOUT_RATE']),
        Dense(2048, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
        BatchNormalization(),
        Dense(1024, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
        BatchNormalization(),
        Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
        BatchNormalization(),
        Dense(2, activation='softmax')
    ])

    # Compile model
    optimizer = Adam(learning_rate=self.config['LEARNING_RATE'])
    model.compile(
        optimizer=optimizer,
        loss='categorical_crossentropy',
        metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall()]
    )

    return model

def get_callbacks(self):
    """Get training callbacks"""
    return [
        ReduceLROnPlateau(
            monitor='val_accuracy',
            factor=0.1,
            patience=self.config['PATIENCE'],

```



```

        min_delta=self.config['MIN_DELTA'],
        verbose=1
    ),
    EarlyStopping(
        monitor='val_accuracy',
        patience=self.config['PATIENCE']*2,
        restore_best_weights=True
    ),
    ModelCheckpoint(
        'best_pothole_model.h5',
        monitor='val_accuracy',
        save_best_only=True,
        verbose=1
    ),
    TensorBoard(
        log_dir='./logs',
        histogram_freq=1
    ),
    CSVLogger('training_log.csv')
]

# Initialize model
pothole_model = PotholeDetectionModel(CONFIG)
model = pothole_model.model
model.summary()

# Visualize model architecture
tf.keras.utils.plot_model(
    model,
    to_file='model_architecture.png',
    show_shapes=True,
    show_layer_names=True,
    rankdir='TB',
    expand_nested=True,

```

```

        dpi=96
    )

# Model Training
print("\nStarting model training...")
start_time = time.time()

history = model.fit(
    augmenter.flow(X_train, y_train, batch_size=CONFIG['BATCH_SIZE'],
        steps_per_epoch=len(X_train) // CONFIG['BATCH_SIZE'],
        epochs=CONFIG['EPOCHS'],
        validation_data=(X_val, y_val),
        callbacks=pothole_model.get_callbacks(),
        verbose=1
    )

training_time = time.time() - start_time
print(f"\nTraining completed in {training_time//60:.0f}m {training_time%60:.0f}s")

# Training Visualization
def plot_training_history(history):
    """Plot training and validation metrics"""
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))

    # Accuracy plot
    ax1.plot(history.history['accuracy'])
    ax1.plot(history.history['val_accuracy'])
    ax1.set_title('Model Accuracy')
    ax1.set_ylabel('Accuracy')
    ax1.set_xlabel('Epoch')
    ax1.legend(['Train', 'Validation'], loc='upper left')

    # Loss plot
    ax2.plot(history.history['loss'])

```

```

ax2.plot(history.history['val_loss'])
ax2.set_title('Model Loss')
ax2.set_ylabel('Loss')
ax2.set_xlabel('Epoch')
ax2.legend(['Train', 'Validation'], loc='upper left')

plt.tight_layout()
plt.show()

plot_training_history(history)

# Model Evaluation
def evaluate_model(model, X_test, y_test):
    """Evaluate model performance"""
    print("\nEvaluating model performance...")
    results = model.evaluate(X_test, y_test, verbose=0)

    print(f"Test Loss: {results[0]:.4f}")
    print(f"Test Accuracy: {results[1]:.4f}")
    print(f"Test Precision: {results[2]:.4f}")
    print(f"Test Recall: {results[3]:.4f}")

# Predictions
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

# Confusion matrix
cm = confusion_matrix(y_true_classes, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)
plt.title('Confusion Matrix')

```

```

plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()

# Classification report
print("\nClassification Report:")
print(classification_report(y_true_classes, y_pred_classes,
target_names=label_encoder.classes_))

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_true_classes, y_pred[:, 1])
roc_auc = roc_auc_score(y_true_classes, y_pred[:, 1])

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

evaluate_model(model, X_val, y_val)

# Test Set Evaluation
class TestSetEvaluator:
    def __init__(self, model, label_encoder, dataset_paths):
        self.model = model
        self.label_encoder = label_encoder
        self.dataset_paths = dataset_paths

```

```

def evaluate_test_set(self):
    """Evaluate model on test set"""
    print("\nEvaluating on test set...")

    for class_name, path in self.dataset_paths['TEST'].items():
        print(f"\nProcessing {class_name} test images...")
        correct = 0
        total = 0

        for filename in tqdm(os.listdir(path)):
            file_path = os.path.join(path, filename)
            try:
                # Load and preprocess image
                image = cv2.imread(file_path)
                if image is None:
                    continue

                image = cv2.resize(image, (CONFIG['IMAGE_SIZE'],
CONFIG['IMAGE_SIZE']))
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                image = image.astype('float32') / 255.0
                image = np.expand_dims(image, axis=0)

                # Make prediction
                prediction = self.model.predict(image)
                predicted_class = np.argmax(prediction, axis=1)[0]
                predicted_label =
self.label_encoder.inverse_transform([predicted_class])[0]

                # Display sample predictions
                if total < 3: # Show first 3 samples
                    plt.figure()
                    plt.imshow(image[0])
                    plt.title(f"True: {class_name}\nPredicted: {predicted_label}")

```

```

plt.axis('off')
plt.show()

# Count correct predictions
if predicted_label == class_name:
    correct += 1
total += 1

except Exception as e:
    print(f"Error processing {filename}: {str(e)}")

accuracy = correct / total if total > 0 else 0
print(f"{class_name} Test Accuracy: {accuracy:.2%} ({correct}/{total})")

# Initialize and run test evaluator
test_evaluator = TestSetEvaluator(model, label_encoder, DATASET_PATHS)
test_evaluator.evaluate_test_set()

# Save the final model
model.save('pothole_detection_model_final.h5')
print("\nModel saved successfully as 'pothole_detection_model_final.h5'")

# Additional Utility Functions
def predict_single_image(model, image_path, label_encoder):
    """Make prediction on a single image"""
    try:
        # Load and preprocess image
        image = cv2.imread(image_path)
        if image is None:
            raise ValueError("Unable to read image")

        original_image = image.copy()
        image = cv2.resize(image, (CONFIG['IMAGE_SIZE'],
CONFIG['IMAGE_SIZE']))

```

```

image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image = image.astype('float32') / 255.0
image = np.expand_dims(image, axis=0)

# Make prediction
prediction = model.predict(image)
predicted_class = np.argmax(prediction, axis=1)[0]
predicted_label = label_encoder.inverse_transform([predicted_class])[0]
confidence = np.max(prediction) * 100

# Display results
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(original_image[:, :, ::-1])
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(image[0])
plt.title(f'Prediction: {predicted_label}\nConfidence: {confidence:.2f}%')
plt.axis('off')

plt.tight_layout()
plt.show()

return predicted_label, confidence

except Exception as e:
    print(f'Error: {str(e)}')
    return None, None

```

References

[1] A. A. Shaghouri, R. Alkhatib, and S. Berjaoui, "Real Time Pothole Detection Using Deep Learning," arXiv, Cornell University, 2021, July 13.

Available: <https://doi.org/10.48550/arxiv.2107.06356>

[2] K. R. Ahmed, "Smart Pothole Detection Using Deep Learning Based on Dilated Convolution," *Sensors*, vol. 21, no. 24, Article No. 8406, pp. 1–15, 2021, December 16.

Available: <https://doi.org/10.3390/s21248406>

[3] Y. K. Yik, N. E. Alias, Y. Yusof, and S. Isaak, "A Real-time Pothole Detection Based on Deep Learning Approach," *Journal of Physics: Conference Series*, vol. 1828, no. 1, Article No. 012001, pp. 1–10, 2021, February 1.

Available: <https://doi.org/10.1088/1742-6596/1828/1/012001>

[4] A. A. Shah, G. Sharma, and L. Bhargava, "Smart Implementation of Computer Vision and Machine Learning for Pothole Detection," *IEEE Conference*, 2021, January 28.

Available: <https://doi.org/10.1109/confluence51648.2021.9376886>

[5] K. E. An, S. W. Lee, S. Ryu, and D. Seo, "Detecting a pothole using deep convolutional neural network models for an adaptive shock observing in a vehicle driving," *IEEE Conference*, 2018, January 1.

Available: <https://doi.org/10.1109/icce.2018.8326142>

[6] R. A. Borgalli, "Smart Pothole Detection and Mapping System," *Journal of Ubiquitous Computing and Communication Technologies*, vol. 2, no. 3, pp. 136–144, 2020, September 8.

Available: <https://doi.org/10.36548/jucct.2020.3.003>

[7] S. Li, C. Yuan, D. Liu, and H. Cai, "Integrated Processing of Image and GPR Data for Automated Pothole Detection," *Journal of Computing in Civil Engineering*, 2016, November 1.

Available: [https://doi.org/10.1061/\(asce\)cp.1943-5487.0000582](https://doi.org/10.1061/(asce)cp.1943-5487.0000582)

[8] "Machine Learning for Automated Pothole Detection," *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, vol. 30, no. 6, pp. 1–10, 2019, April 30.

Available: <https://www.ijraset.com/files/serve.php?FID=21377>

[9] M. Gao, X. Wang, S. Zhu, and P. Guan, "Detection and Segmentation of Cement Concrete Pavement Pothole Based on Image Processing Technology," *Mathematical Problems in Engineering*, vol. 2020, Article ID 1360832, pp. 1–13, 2020, January 28.

Available: <https://doi.org/10.1155/2020/1360832>

[10] A. Kumar, Chakrapani, D. J. Kalita, and V. P. Singh, "A Modern Pothole Detection technique using Deep Learning," *IEEE*, 2020, February 1.

Available: <https://doi.org/10.1109/idea49133.2020.9170705>

[11] C. Koch and I. Brilakis, "Pothole detection in asphalt pavement images," *Advanced Engineering Informatics*, vol. 25, pp. 507–515, 2011.

[12] T. Kim and S. Ryu, "Review and analysis of pothole detection methods," *Journal of Emerging Trends in Computing and Information Sciences*, vol. 5, no. 8, pp. 603–608, 2014.

[13] H. Bello-Salau, A. M. Aibinu, E. N. Onwuka, J. J. Dukiya, and A. J. Onumanyi, "Image processing techniques for automated road defect detection: A survey," 11th Int. Conf. on Electronics Computer and Computation (ICECCO), 2014.

[14] P. B. V. Raja Rao, M. Prasad, P. Kiran Sree, C. Venkata Ramana, and P. T. Satyanarayana Murty, "Enhancing the MANET AODV Forecast of a Broken Link with LBP," Smart Innovation, Systems and Technologies (ICISSC 2022), vol. 363, pp. 1–12, 2023.

Available: https://doi.org/10.1007/978-981-99-4717-1_6

[15] A. Khang, K. C. Rath, S. Panda, P. K. Sree, and S. K. Panda, "Revolutionizing Agriculture: Exploring Advanced Technologies for Plant Protection in the Agriculture Sector," Handbook of Research on AI-Equipped IoT Applications in High-Tech Agriculture, IGI Global, pp. 1–22, 2023.

Available: <https://doi.org/10.4018/9781-6684-9231-4.ch001>

[16] S. Mangalampalli, P. K. Sree, S. S. S. N. Usha Devi, and R. B. Mallela, "An Effective VM Consolidation Mechanism by Using the Hybridization of PSO and Cuckoo Search Algorithms," Computational Intelligence in Data Mining, vol. 281, Springer, 2022.

Available: https://doi.org/10.1007/978-981-16-9447-9_37

[17] W. Li, J. Liu, Z. Xiao, D. Zhu, J. Liao, W. Yu, J. Feng, B. Qian, Y. Fang, and S. Li, "Automatic grading of knee osteoarthritis with plain radiograph radiomics model: combining anteroposterior and lateral images, Insights into Imaging, vol. 15, Article No.143, pp.1-10, 2024.

Available: <https://insightsimaging.springeropen.com/articles/10.1186/s13244-024-01719-3>

[18] M. S. Swethasri, A. Sneha, B. Swetha, and K. Suresh, "Knee Osteoarthritis Detection and Its Severity," International Research Journal of Modernization in Engineering Technology and Science, vol. 4, no.5, pp.3740-3745, May 2022.

Available:

https://www.irjmets.com/uploadedfiles/paper/issue_5_may_2022/23638/final/fin_irjmets1653449227.pdf





19% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.




Filtered from the Report

- Bibliography
- Small Matches (less than 8 words)
- Crossref database
- Crossref posted content database

Match Groups

-  **179** Not Cited or Quoted 18%
Matches with neither in-text citation nor quotation marks
-  **3** Missing Quotations 0%
Matches that are still very similar to source material
-  **8** Missing Citation 1%
Matches that have quotation marks, but no in-text citation
-  **0** Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

- 16%  Internet sources
- 10%  Publications
- 16%  Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer

Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely AI generated as AI generated and AI paraphrased or likely AI generated and AI paraphrased writing as only AI generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?

The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?

Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

