MINOR DEGREE MINI PROJECT

On

POTHOLE DETECTION SYSTEM USING CNN

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

MINOR DEGREE

In

DATA SCIENCE

By

P VAMSI KRISHNA -20261A04G5

G SATHWIK REDDY-20261A0415

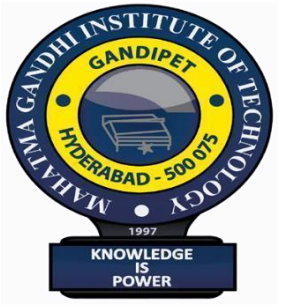
N RAKESH REDDY-20261A0437

Under the Guidance of

Ms. U CHAITANYA

Assistant Professor

Department of IT



DEPARTMENT OF INFORMATION TECHNOLOGY

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY

(AUTONOMOUS)

(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA; Accredited by NAAC with 'A++' Grade)

Kokapet (Village), Gandipet, Chaitanya Bharathi (P.O.), Ranga Reddy Dist, HYDERABAD – 50075, TELANGANA

2023-2024

CERTIFICATE

This is to certify that the Mini Project entitled **POTHOLE DETECTION SYSTEM USING CNN** submitted by **P VAMSI KRISHNA (20261A04G5), G SATHWIK REDDY (20261A0415),N RAKESH REDDY(20261A0437)** in partial fulfillment of the requirements for the award of the Minor Degree in Data Science as specialization is a record of the bonafede work carried out underthe supervision of **Mrs.U CHAITANYA**, and this has not been submitted by another University or Institute for the award of any degree or diploma.

Project Guide:
Ms.U CHAITANYA
Assistant Professor
Dept. of IT

Project Coordinator:
DR.M RUDRA KUMAR
Professor
Dept. of IT

EXTERNAL EXAMINER

Dr. D. Vijaya Lakshmi
Professor & HoD
Dept. of IT

DECLARATION

We hereby declare that the Mini Project entitled POTHOLE DETECTION SYSTEM USING CNN is original and bonafide work carried out by us as a part of the fulfillment of our Minor Degree in Data Science, in the Department of Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of Ms U CHAITANYA, Asst Prof., Dept. of IT, MGIT.

No part of the work is copied from books/journals/internet and wherever the portion is taken the same has been duly referred in the text. The report is based on the work done entirely by usand not copied from any other source.

P VAMSI KRISHNA -20261A04G5

G SATHWIK REDDY-20261A0415

N RAKESH REDDY-20261A0437

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance andencouragement crown all efforts with success. They have been a guiding light and source of inspiration towards the completion of the Mini Project.

We would like to express our sincere gratitude and indebtedness to our project guide **Ms U CHAITANYA**, Assistant Professor, Dept. of IT, who has supported us throughout our project with immensepatience and expertise.

We are also thankful to the honorable Principal of MGIT **Prof. G. CHANDRAMOHAN REDDY** and **Dr. D. VIJAYA LAKSHMI**, Professor & HoD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this Mini Project successfully.

We are also extremely thankful to our Project Coordinators **Dr M RUDRA KUMAR**, Prof., Dept. of IT for the valuable suggestions and guidance throughout the course of this project.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support in the completion of this work.

P VAMSI KRISHNA -20261A04G5
G SATHWIK REDDY-20261A0415
N RAKESH REDDY-20261A0437

ABSTRACT

The road is the most used means of transportation and serves as a country's arteries, so it is extremely important to keep the roads in good condition. Potholes that happen to appear in the road must be repaired to keep the road in good condition. Spotting potholes on the road is difficult, especially in a country like India where roads stretch millions of kilometers across the country.

Therefore, there is a need to automate the identification of potholes with high speed and real-time precision. CNN is an object detection algorithm and our main goal of this article is to train and analyze the YOLO v8 model for pothole detection. The YOLO model, trained on a pothole dataset, is evaluated based on metrics such as accuracy, recall, and model size, solely focusing on detecting potholes.

The pothole detection capabilities of the newly developed YOLO v8 algorithm. The research conducted in this project will help reduce costs and increase the Accuracy of pothole identification and will be of great help in road maintenance.

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

Pothole detection stands as a crucial technology in ensuring road safety, with applications ranging from preventive maintenance to real-time hazard identification. Through the integration of cutting-edge technologies like YOLO v8 and Roboflow, our project aims to revolutionize pothole detection by leveraging deep learning methodologies and real-world datasets. By harnessing the power of deep learning algorithms, particularly within the domain of computer vision, our project endeavors to enhance the accuracy and efficiency of pothole detection systems.

Previously, pothole detection systems faced significant challenges, including accurately identifying potholes amidst varying road conditions, differentiating them from other road anomalies, and processing large volumes of image data in real-time. Moreover, factors such as lighting conditions, road obstructions, and environmental clutter posed additional hurdles in achieving precise and reliable detection results.

To overcome these challenges, our project adopts a multi-faceted approach, utilizing advanced deep learning models such as YOLO v8 and innovative data preprocessing techniques facilitated by Roboflow. By combining robust object detection capabilities with state-of-the-art image annotation tools, we aim to streamline the data collection and annotation process, enabling the efficient training of pothole detection models.

1.1. Problem statement

Potholes on roads present a significant challenge, with traditional detection methods being time-consuming and inefficient. This leads to economic burdens due to vehicle damage and road closures, hindering economic activity. Moreover, potholes pose safety risks for drivers, cyclists, and pedestrians, potentially causing accidents and injuries. Without automated detection systems, infrastructure deterioration worsens over time, necessitating costlier repairs. Developing efficient, automated pothole detection is crucial for better road management, enhancing public safety, and safeguarding economic well-being.

1.2. Motivation

The prevalence of potholes necessitates more efficient detection systems due to their economic, safety, and infrastructure impact. Traditional methods are time-consuming, subjective, and inefficient for large road networks. YOLOv8-based automated detection offers a solution by reducing costs, enhancing safety, and improving efficiency. It enables proactive identification, minimizing vehicle damage and infrastructure deterioration while allowing for faster repairs. Real-time data facilitates better maintenance planning and resource allocation. Overall, this project aims to significantly improve road infrastructure management, public safety, and economic well-being.

1.3 Objective

The primary objective of this project is to develop and evaluate a reliable and efficient pothole detection system using a Convolutional Neural Network (CNN) architecture, specifically YOLOv8. This system will address the limitations of traditional pothole detection methods and contribute to improved road infrastructure management.Evaluate the model's performance on a separate validation dataset not used for training to ensure generalizability and robustness.This project focuses on the core development and evaluation stages. While real-world applicability is a potential future direction, the primary goal is to build a robust and accurate pothole detection system using YOLOv8.

1.4 Proposed System

The YOLOv8-based pothole detection system utilizes deep learning to identify potholes in images or video frames from various sources. It involves data acquisition from road surveys, dashcams, or publicly available datasets, followed by preprocessing steps like resizing and normalization.Training the YOLOv8 model involves feeding annotated data to enable it to recognize potholes and optimize parameters for accurate detection. Visualization of detected potholes is optional for further analysis. YOLOv8 offers real-time potential, single-stage detection, and flexibility due to its open-source nature. This system aims to automate pothole detection, aiding in efficient road management.

1.4.1 Advantages of Proposed System:

- Automated pothole detection with YOLOv8 cuts manual inspections, leading to faster repairs and better road maintenance.
- The system prioritizes safety by enabling early pothole identification and reduces road damage through quicker intervention.
- YOLOv8's efficiency makes it suitable for large areas and potentially real-time pothole detection.
- YOLOv8's open-source nature allows for customization for future improvements like using extra data for better pothole analysis.
- This system offers a promising approach for efficient pothole detection, benefiting road management, public safety, and cost reduction.

CHAPTER-2

LITERATURE SURVEY

J. V. V. Sobral, et al. [1] Proposed System uses street-level imagery combined with deep learning techniques to automatically detect road defects. The architecture includes a convolutional neural network (CNN) to process the images and identify potholes, cracks, and other defects.

C. Koch and I. Brilakis [2]. Proposed System which utilizes image processing techniques to detect potholes in asphalt pavements. The system applies edge detection and morphological operations to segment and identify potholes in the captured images.

K. O. Ouma, et al. [3] Employs a deep learning framework, specifically a CNN, to detect potholes from images captured by cameras mounted on vehicles. The model is trained on a labeled dataset of road images to achieve accurate detection.

D. Zhang, et al. [4] A deep learning-based approach that uses street view imagery for road damage detection. The system utilizes a modified YOLO architecture to detect various types of road damage, including potholes, from large-scale image datasets.

S. S. Khan [5] Applies traditional image processing techniques such as edge detection, thresholding, and contour detection to identify potholes in road images. The system aims to provide a cost-effective solution for road maintenance.

P. Kaur and S. Garg [6] Combines deep learning with image processing to detect potholes. The system uses a CNN model trained on a dataset of road images to classify and locate potholes in real-time.

M. Sharma et al. [7] Implements the YOLO object detection framework to identify potholes in real-time. The system processes video frames from cameras mounted on vehicles to detect and classify potholes accurately.

J. Kulkarni and S. G. Bhirud [8] Uses a combination of computer vision techniques and machine learning algorithms to detect potholes. The system incorporates feature extraction methods and classifiers to identify potholes from road images.

R. W. Mathew and M. H. Jawahar [9] An automatic detection system that utilizes video footage for monitoring road surfaces. The system employs a combination of image processing and machine learning techniques to identify and classify potholes.

A. Verma and A. Sharma [10] A real-time pothole detection system leveraging the YOLO deep learning framework. The system processes video streams from dashcams to detect and map potholes, providing timely information for road maintenance.

2.1 Existing System

The existing system for pothole detection typically relies on manual inspections by road maintenance crews or traditional computer vision techniques. These methods are often labor-intensive, time-consuming, and prone to errors. They may involve physical surveys or the use of basic image processing algorithms to identify potholes from road images or videos. However, these approaches often lack accuracy and efficiency, leading to delays in identifying and repairing road defects.

2.1.1 Limitations of Existing system

limitation of the existing system

The limitations of the existing pothole detection systems include:

- Manual Inspection: The reliance on manual inspections by road maintenance crews is labor-intensive and time-consuming, leading to delays in identifying and repairing potholes.
- Limited Accuracy: Traditional computer vision techniques may lack accuracy in detecting potholes, especially in complex road environments or under varying lighting conditions.
- Dependency on Human Observers: Human error and subjectivity can affect the accuracy of pothole detection, leading to inconsistencies in identifying and prioritizing repairs.
- Inefficient Data Analysis: Existing systems may struggle with efficiently analyzing large volumes of road imagery or video data, resulting in delays in processing and actioning detected potholes.
- Lack of Real-time Monitoring: Some systems may not provide real-time monitoring capabilities, making it challenging to promptly address newly formed potholes and ensure road safety.

CHAPTER-3
SYSTEM SPECIFICATIONS

3.1 Software Requirements

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

NumPy: A fundamental package for numerical computing in Python, providing support for arrays, matrices, and mathematical operations.

OpenCV (cv2): An open-source computer vision library offering tools for image and video analysis, including features like object detection and pose estimation.

Ultralytics : The ultralytics module is a Python library specializing in state-of-the-art computer vision algorithms, particularly focusing on object detection and tracking. It provides easy-to-use interfaces for deploying pre-trained deep learning models, such as YOLO (You Only Look Once), facilitating robust object detection tasks in various applications.

PyTorch Model Summary: A library for generating summaries of PyTorch model architectures, aiding in understanding model structures and parameters.

- **Roboflow**
- **Google colab**
- **Python IDLE (Python version 3.10)**
- **Thonny IDE**

3.2 Hardware Requirements

- The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware, A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements. Architecture – All computer operating systems are designed for a particular computer architecture. Most software applications are limited to particular operating systems running on particular architectures. Although architecture-independent operating systems and applications exist, most need to be recompiled to run on a new architecture. See also a list of common operating systems and their supporting architectures.

- **Operating System: Windows Only**
- **Processor: i5 and above**
- **RAM: 8 GB and above**
- **Hard Disk: 25 GB in local drive**

CHAPTER 4

SYSTEM DESIGN AND ANALYSIS

4.1 Module Description

Data Collection:

For the pothole detection project, data collection involves gathering relevant information related to road conditions and potholes. This can include various sources such as street images, videos captured by vehicles or cameras installed along roads, and sensor data from vehicles or road monitoring systems. Additionally, crowd-sourced data or reports from drivers or citizens about pothole locations and road damage can be valuable sources. The data collection process also involves annotating or labeling the collected data to identify and mark the presence of potholes accurately. Overall, effective data collection ensures the availability of high-quality data for training and testing pothole detection algorithms and models.

Data Annotation:

This module we utilized Roboflow for both data annotation and data collection. Roboflow provided a user-friendly platform where we could efficiently annotate images and videos to identify potholes. Using Roboflow's annotation tools, we meticulously labeled potholes in the collected data, ensuring precise identification for model training. Additionally, Roboflow facilitated seamless data collection by allowing us to organize and manage our annotated datasets effectively. With its integration capabilities, we could easily import data from various sources, including street images, surveillance videos, and sensor data, streamlining the entire data collection process. Furthermore, we annotated a substantial dataset of 2000 images using Roboflow, ensuring a diverse and comprehensive collection for robust model training. Leveraging Roboflow's annotation and data collection features significantly enhanced the quality and quantity of our dataset, laying a solid foundation for accurate pothole detection models.

Model Definitions:

The model architecture for our pothole detection project comprises a combination of YOLO v8, Roboflow, and Convolutional Neural Networks (CNNs). YOLO v8 serves as the backbone for object detection, providing efficient and accurate detection of potholes in images and videos. Roboflow facilitates data preprocessing, annotation, and dataset management, ensuring the availability of high-quality annotated data for training.

Additionally, CNNs are integrated into the model to extract spatial features from images, enabling precise localization and classification of potholes. This model architecture synergistically combines state-of-the-art techniques to achieve robust and effective pothole detection in diverse real-world scenarios.

Training Loop:

The architecture of our anomaly detection system leverages a combination of LSTM-based autoencoders, Isolation Forests, and deep learning models. LSTM-based autoencoders capture temporal dependencies in user behavior data, allowing for the detection of subtle anomalies over time. Isolation Forests complement this by efficiently isolating anomalous instances in high-dimensional feature spaces. Additionally, deep learning models, such as Convolutional Neural Networks (CNNs), are integrated to extract complex spatial features from raw data, enhancing anomaly detection accuracy. This multi-model approach ensures robust detection of anomalous activities across diverse datasets and scenarios, enabling effective threat monitoring and mitigation.

Roboflow:

Roboflow is a versatile platform that revolutionizes the data preparation process for computer vision projects. With its user-friendly interface and powerful features, Roboflow empowers researchers, developers, and data scientists to efficiently manage their datasets and streamline the annotation process.At the core of Roboflow's functionality is its robust image annotation tool, which enables users to label objects of interest with precision and accuracy. Whether it's detecting potholes on road surfaces or identifying objects in satellite imagery, Roboflow provides intuitive tools for annotating images with bounding boxes, polygons, and semantic segmentation masks.In addition to its annotation capabilities, Roboflow offers a wide range of data preprocessing tools to enhance the quality and diversity of datasets. Users can resize images, apply data augmentation techniques, and normalize pixel values with just a few clicks, ensuring that their models are trained on clean and representative data.

A Brief History of YOLO :

Before discussing YOLO's evolution, let's look at some basics of how a typical object detection algorithm works. The diagram below illustrates the essential mechanics of an object detection model. The essential mechanics of an object detection model – source. The architecture consists of a backbone, neck, and head. The backbone is a pre-trained Convolutional Neural Network (CNN) that extracts low, medium, and high-level feature maps from an input image. The neck merges these feature maps using path

aggregation blocks like the Feature Pyramid Network (FPN). It passes them onto the head, classifying objects and predicting bounding boxes. The head can consist of one-stage or dense prediction models, such as YOLO or Single-shot Detector (SSD). Alternatively, it can feature two-stage or sparse prediction algorithms like the R-CNN series.

4.1.1 Proposed Architecture

The YOLOv8-based pothole detection system utilizes deep learning to identify potholes in images or video frames from various sources. It involves data acquisition from road surveys, dashcams, or publicly available datasets, followed by preprocessing steps like resizing and normalization.Training the YOLOv8 model involves feeding annotated data to enable it to recognize potholes and optimize parameters for accurate detection. Visualization of detected potholes is optional for further analysis. YOLOv8 offers real-time potential, single-stage detection, and flexibility due to its open-source nature. This system aims to automate pothole detection, aiding in efficient road management.

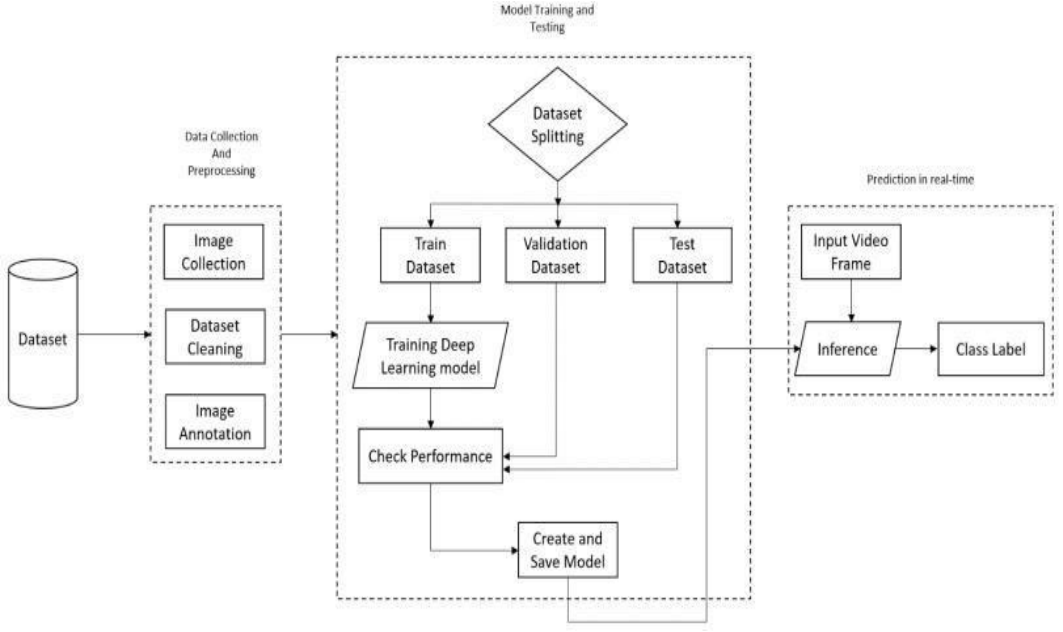


Fig 4.1.1. Architecture diagram of Pothole detection

4.1.2 Workflow Diagram

The diagram typically consists of actors, represented as stick figures or system entities, and use cases, depicted as ovals. Actors are external entities that interact with the system, while use cases represent specific functionalities or tasks the system performs. Lines connecting actors and use cases indicate relationships, such as associations, inclusions, or extensions.

Associations show which actors are involved in specific use cases, while inclusion relationshipsdenote that one use case encompasses another. Extensions indicate optional or conditional behaviour. The use case diagram provides a high-level overview of the system's functionalities,showcasing the scope of user interactions and system responses. It serves as a valuable tool forcommunication between stakeholders, allowing for a clear understanding of the system's behaviour and requirements.

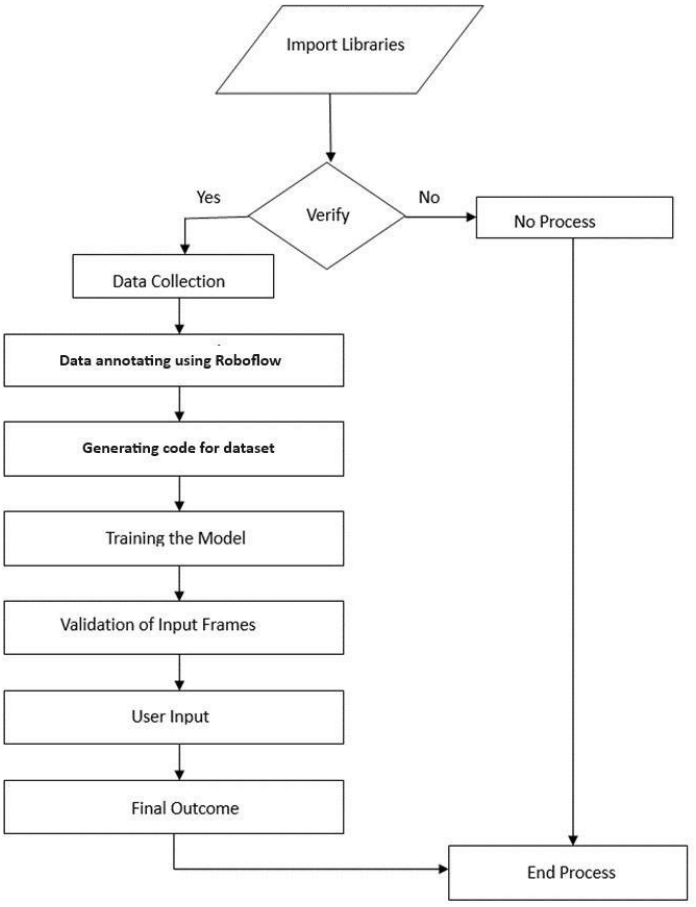


Fig 4.1.2 Work flow diagram of Pothole detection

4.2 UML Diagrams

The Unified Modeling Language (UML) is a language used in software engineering to represent systems. It provides a range of notations that help model the structure, behavior and architecture of software systems. By using UML, software developers, analysts, designers and stakeholders can effectively. Comprehend the design, flow and relationships within a system. UML diagrams are categorized into types each representing aspects of a system;

- Structural Diagrams: These diagrams offer a view of the systems structure by showcasing classes, components and their relationships. Examples include Class Diagrams, Object Diagrams and Component Diagrams.
- Behavioral Diagrams: These diagrams depict the behavior and interactions, among components within the system. They include Sequence Diagrams, Activity Diagrams,State Diagrams and Use Case Diagrams.
- Deployment Diagrams: These diagrams illustrate how software components are physically deployed on hardware nodes, servers,or devices.
- Overall UML serves as a tool, for designing, analyzing, and documenting software systems. It aids in representing systems throughout the entire software development lifecycle.

The Primary goals of UML are as follows:

- Establishing a unified notation and semantics for software artifacts.
- Creating graphical representations to illustrate software systems.
- Depicting complex systems with different levels of detail.
- Facilitating effective communication among stakeholders.
- Supporting integration with diverse modelling tools.
- Adapting to various software development methodologies and contexts.

4.2.1 Use case diagram

A use case diagram is a visual representation of the functional requirements and interactions within a system, illustrating how users (actors) interact with the system through various use cases. The diagram typically consists of actors, represented as stick figures or system entities, and use cases, depicted as ovals. Actors are external entities that interact with the system, while use cases represent specific functionalities or tasks the system performs. Lines connecting actors and use cases indicate relationships, such as associations, inclusions, or extensions. Associations show which actors are involved in specific use cases, while inclusion relationships denote that one use case encompasses another. Extensions indicate optional or conditional behaviour. The use case diagram provides a high-level overview of the system's functionalities, showcasing the scope of user interactions and system responses. It serves as a valuable tool for communication between stakeholders, allowing for a clear understanding of the system's behaviour and requirements.

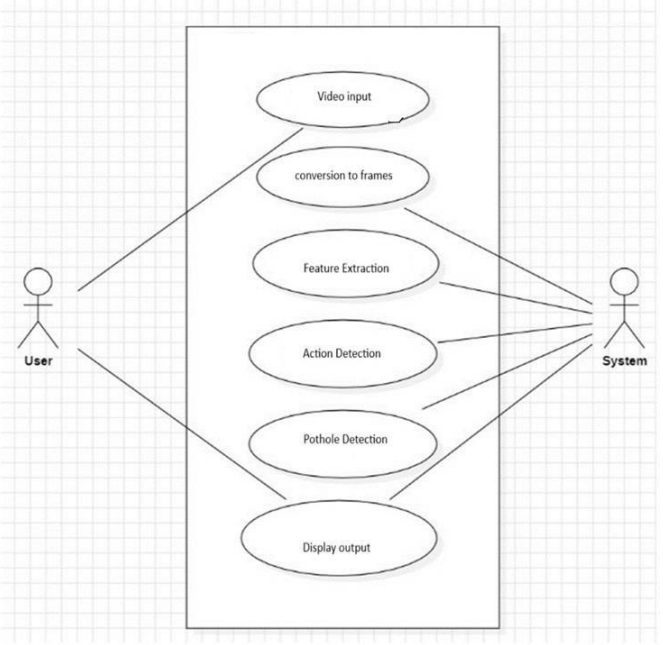


Fig 4.2.1. Use case diagram of Pothole detection

4.2.2 Sequence diagram

A sequence diagram is a type of UML (Unified Modeling Language) diagram that illustrates the interactions and order of messages between objects or components within a system over a specific period. It provides a dynamic view of a system by showing the sequence of messages exchanged among different entities, typically representing objects or instances of classes. The diagram is presented in a chronological order from top to bottom, with vertical lines representing the lifelines of the involved entities. Arrows and horizontal lines depict the flow of messages between these entities, showing the progression of the interactions. Sequence diagrams are valuable for visualizing the runtime behavior of a system, aiding in the understanding of the temporal aspects of the system's processes and communication flow. They are particularly useful during the design and analysis phases of software development to model and communicate the dynamic aspects of a system's functionality.

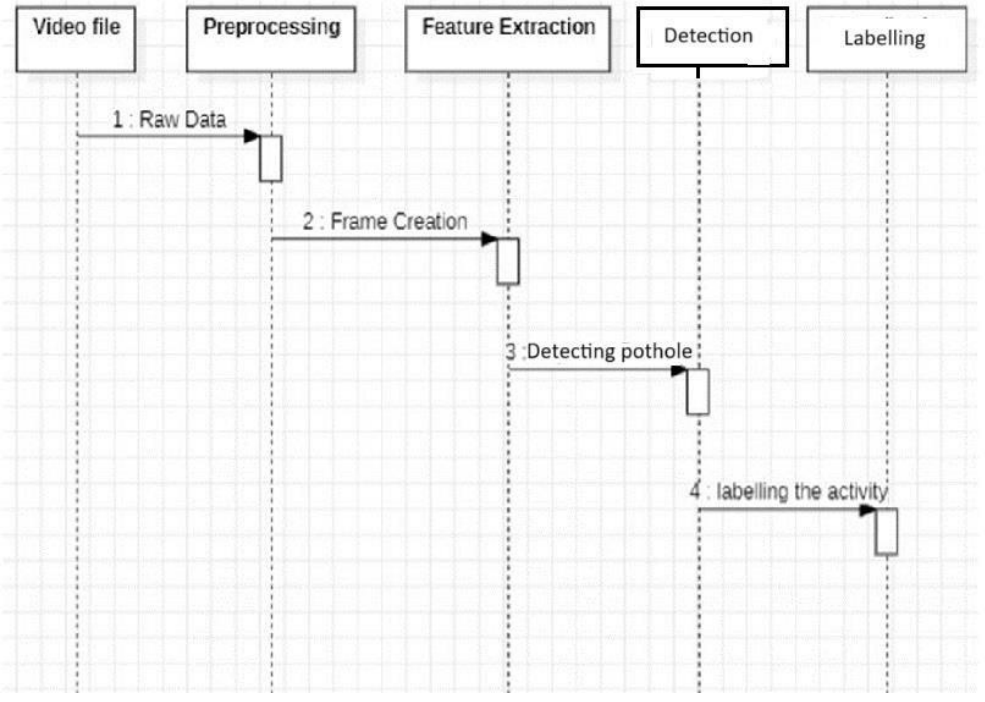


Fig 4.2.2. Sequence diagram of Pothole detection

4.2.3 Activity diagram

An activity diagram is a UML (Unified Modeling Language) diagram that visually represents the flow of activities in a system or process. It is particularly useful for modeling business processes, workflows, or the dynamic aspects of a system. Nodes in the diagram represent activities, and arrows indicate the flow of control from one activity to another. Decision points, parallel activities, and synchronization bars can be included to depict complex scenarios. Activity diagrams help stakeholders understand the sequential and parallel aspects of a process, making them valuable tools for both business and system analysts.

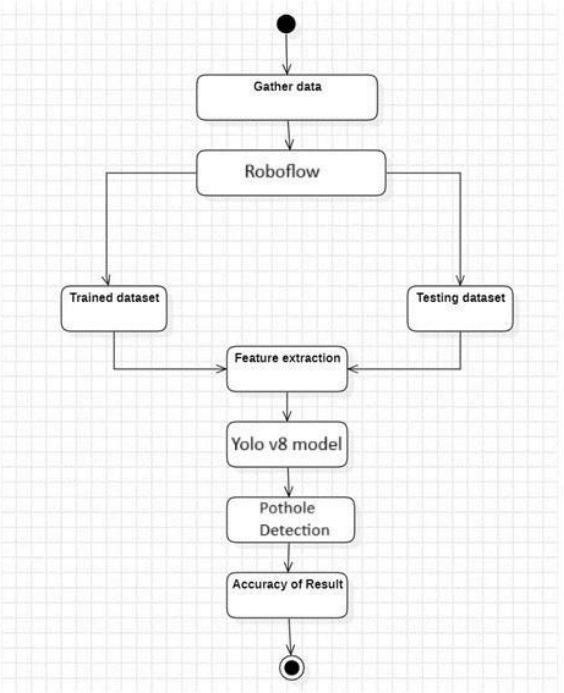


Fig 4.2.3. Activity diagram of Pothole detection

4.2.4 Class diagram

A class diagram is a static UML diagram that illustrates the structure of a system by representing classes, their attributes, relationships, and methods. Classes are depicted as boxes, and associations between classes are shown by connecting lines. Attributes and methods are listed within each class box. Class diagrams are fundamental in object-oriented modeling, offering a blueprint for software development. They facilitate communication among developers, ensuring a shared understanding of the system's structure and providing a basis for coding and design decisions.

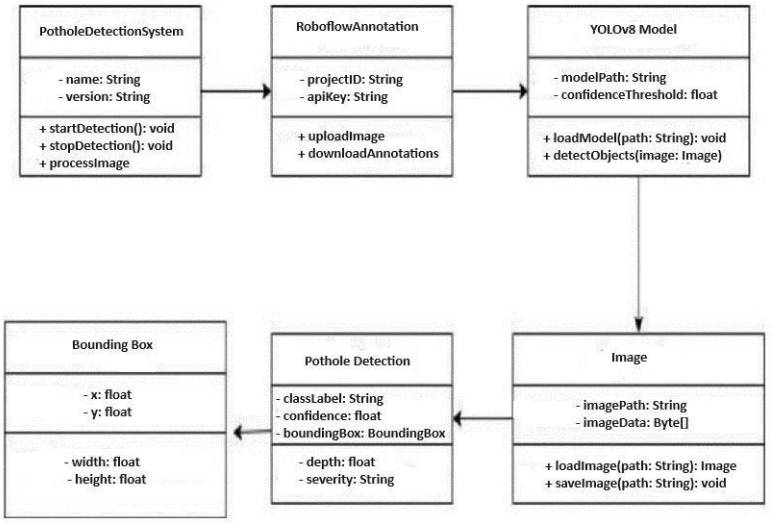


Fig 4.2.4. Class diagram of Pothole detection

4.2.5 Component diagram

A component diagram is a visual representation in UML that illustrates the structural organization of a software system by depicting components and their relationships. Components represent modular units, such as classes or packages, and connections show dependencies or interfaces between them. It provides a high-level view of a system's architecture, aiding in understanding the organization and interactions of its parts.

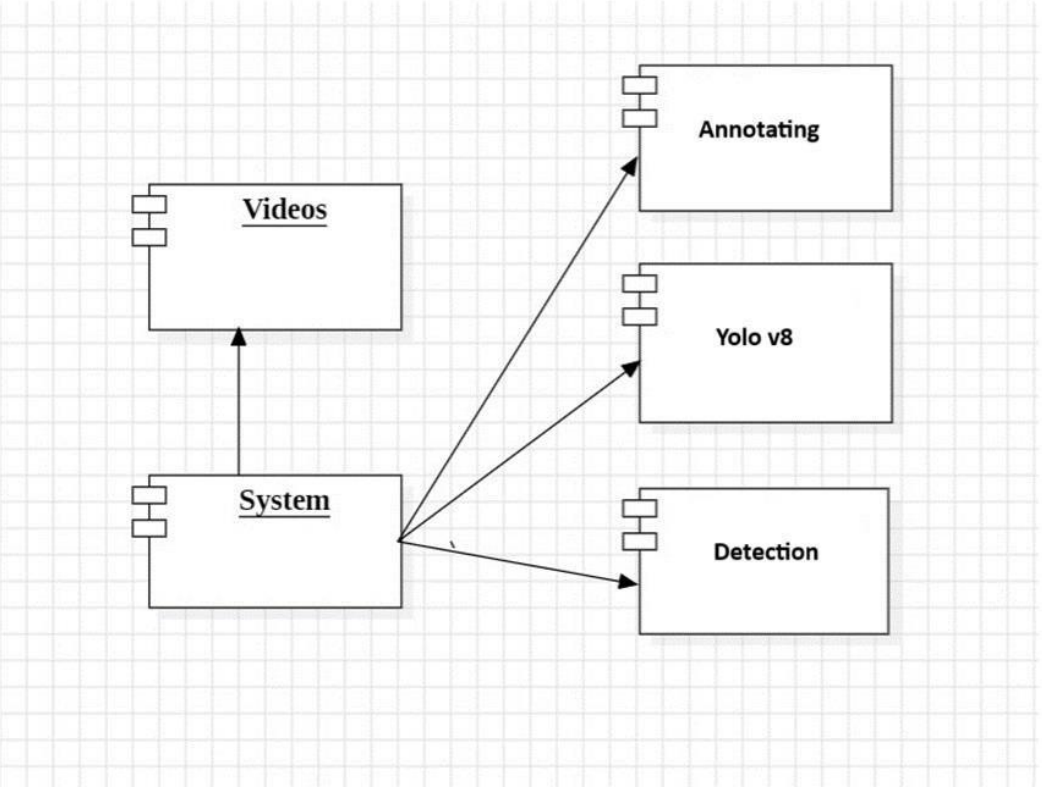


Fig 4.2.5. Component diagram of Pothole detection

4.2.6 Deployment diagram

A deployment diagram in UML focuses on the hardware and software infrastructure of a system. It depicts how software components are distributed across hardware nodes and shows the connections between them. Nodes, representing hardware devices or software execution environments, are connected by communication pathways, indicating the deployment configuration. Deployment diagrams are valuable in understanding the physical deployment of a system, assisting system architects and administrators in planning, configuring, and maintaining the infrastructure that supports the software.

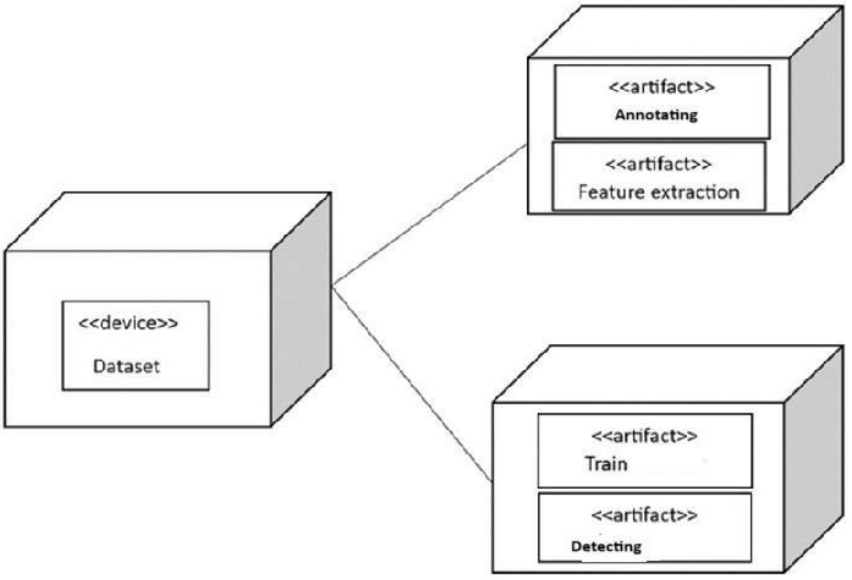


Fig 4.2.6. Deployment diagram of Pothole detection

CHAPTER 5

IMPLEMENTATION

Implementing a pothole detection system involves a meticulous process, starting with the collection and annotation of road images containing potholes. Leveraging tools like Roboflow, these images are annotated to precisely mark the locations of potholes, facilitating accurate model training. Following annotation, the dataset undergoes preprocessing to ensure compatibility with YOLOv8 and CNN models. Roboflow offers versatile preprocessing capabilities, enabling tasks such as resizing, augmentation, and dataset organization for optimal training efficiency.

Model selection plays a pivotal role, with YOLOv8 serving as the cornerstone due to its efficiency and accuracy in object detection, including potholes. Complementing YOLOv8, a CNN-based classifier is employed to further enhance detection accuracy, particularly in distinguishing potholes from other objects or road anomalies.

1. Data Preparation:

For our pothole detection project, we sourced data from multiple sources, including Google and YouTube, to create a diverse and comprehensive dataset. These platforms provided us with a wide range of images and videos capturing various road conditions and environments, including urban streets, highways, and rural roads.

After annotating the dataset using Roboflow, we generated the necessary code and metadata files required for training our models. Roboflow's code generation capabilities allowed us to seamlessly export the annotated dataset in various formats compatible with popular deep learning frameworks such as TensorFlow, PyTorch, and YOLO (You Only Look Once).

By collecting data from diverse sources and annotating it with Roboflow, we ensured that our dataset encompasses a wide range of road conditions and pothole types, enabling our models to generalize well to real-world scenarios. This annotated dataset serves as the foundation for training and evaluating our pothole detection algorithms, helping us develop robust and accurate solutions for identifying and mitigating road hazards.

5.1 Sample code and Implementation Details

DATA COLLECTION:

In Roboflow, data can be collected by uploading images or videos directly to the platform or by importing data from various sources such as Google Drive, Dropbox, or URLs. Additionally, Roboflow provides integration with popular data labeling tools like LabelImg, making it easier to annotate and preprocess the collected data efficiently.

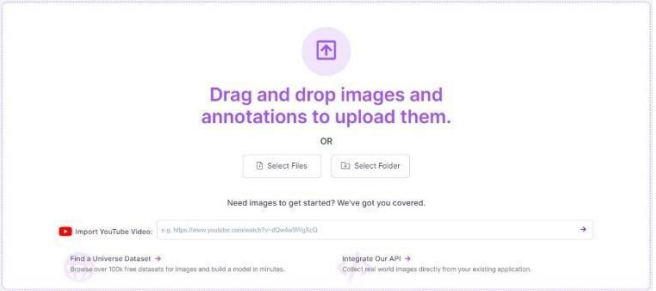


Fig 5.1.1 Data Collection Image of Roboflow

Fig 5.1.2 Sample Image

ANNOTATION:

In Roboflow, potholes or other objects of interest can be annotated directly on the images using bounding boxes or segmentation masks, allowing for precise labeling of the relevant features for training machine learning models.

Fig 5.1.3 Annotated Image

DATA SET:

In Roboflow, we diligently annotated a substantial dataset comprising 2000 images, ensuring meticulous labeling of potholes and pertinent features crucial for training our machine learning models.

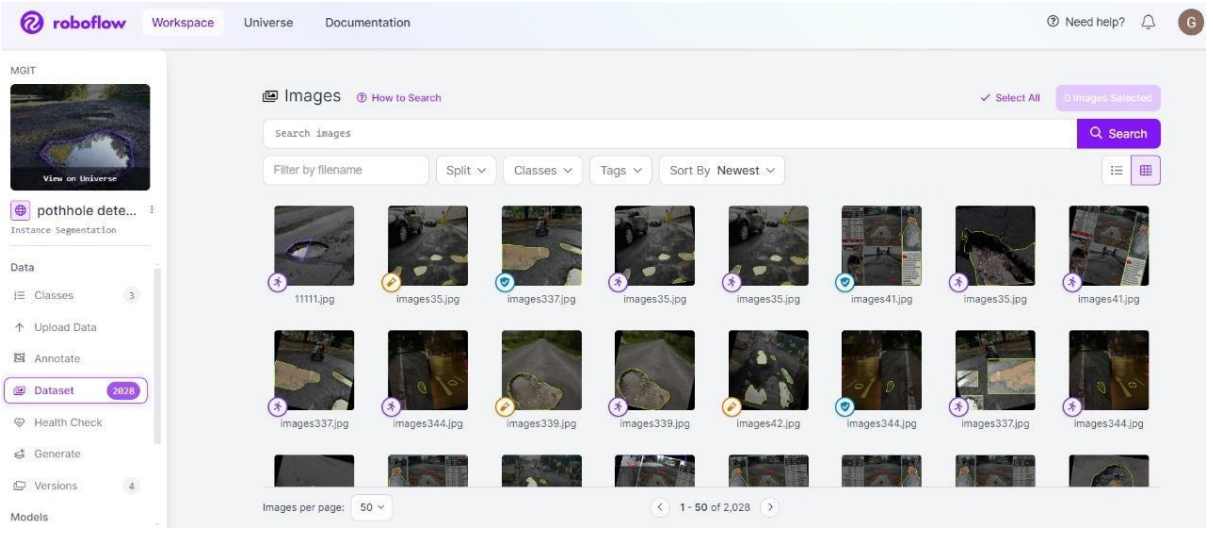


Fig 5.1.4 Data Set

CODE GENERATION:

We gathered a dataset of 2000 images from various sources, including Google and YouTube, and annotated potholes using Roboflow. After annotation, we generated code tailored to our dataset to facilitate efficient training of our pothole detection model. This data preparation process ensured that our machine learning system could effectively identify potholes in real-

world scenarios.

Data Set Code

```
!pip install roboflow --quiet

from roboflow import Roboflow
rf = Roboflow(api_key="928MyAxIm2K0hh8xRQ6C")
project = rf.workspace("freedomtech").project("rpi4-yolov8-segmentation")
dataset = project.version(1).download("yolov8")
```

PYTHON CODE FOR MODEL DEVELOPMENT:

To develop the pothole detection model, Python code was written using PyTorch, a popular deep learning framework. The code utilized the YOLOv8 model architecture for object detection, with transfer learning applied to fine-tune the model for pothole detection specifically. Data preprocessing, model training, and evaluation were conducted using PyTorch's built-in functionalities, including data loaders, loss functions, and optimizers. The code iteratively trained the model on annotated pothole images, adjusting model parameters to minimize classification errors and optimize detection accuracy. Finally, the trained model was validated and tested on separate datasets to assess its performance and ensure robustness in detecting potholes accurately.

DOWNLOADED THE MODEL:

After training, the model was downloaded and saved for future use. This involved saving the trained model weights and architecture configuration to disk using PyTorch's serialization functionalities. The downloaded model file, typically in the form of a ".pt" or ".pth" file, contained all the learned parameters and information necessary to replicate the trained model's architecture and behavior. This allowed for easy deployment and integration into the pothole detection system for real-time inference on new images or video streams.

SOURCE CODE

```
import os
HOME = os.getcwd()
print(HOME)

# Pip install method (recommended)
!pip install ultralytics==8.0.196

from IPython import display
display.clear_output()

import ultralytics
ultralytics.checks()
from ultralytics import YOLO
from IPython.display import display, Image

%cd {HOME}
!yolo task=segment mode=predict model=yolov8s-seg.pt
conf=0.25source='https://media.roboflow.com/notebooks/examples/dog.jpeg'

%cd {HOME}
Image(filename='runs/segment/predict/dog.jpeg',  height=600)
!mkdir {HOME}/datasets
%cd {HOME}/datasets

!pip install roboflow --quiet

from roboflow import Roboflow
rf = Roboflow(api_key="928MyAxIm2K0hh8xRQ6C")
project = rf.workspace("freedomtech").project("rpi4-yolov8-segmentation")
dataset = project.version(1).download("yolov8")

#Custom Training
%cd {HOME}
!yolo task=segment mode=train model=yolov8s-seg.pt data={dataset.location}/data.yaml epochs=100
imgsz=640
# predicting on the video
import cv2
import numpy as np
from ultralytics import YOLO

# Load the YOLO model (replace "best.pt" with your model path)
model = YOLO("mine.pt")
class_names = model.names
# Open the video capture
cap = cv2.VideoCapture('P2.mp4')
# Define the output video writer (replace "output.avi" with your desired filename)
fourcc = cv2.VideoWriter_fourcc(*'XVID')  # Adjust codec if needed (e.g., 'MP4V')
out = cv2.VideoWriter('output.avi', fourcc, 20.0, (1020, 500))  # Adjust FPS and resolution as needeD
count = 0
while True:
    ret, img = cap.read()
    if not ret:
        break
```

```
    count += 1
    # Process all frames (remove frame skipping)
    # if count % 3 != 0:
    #     continue
    # Resize the image (optional)
    img = cv2.resize(img, (1020, 500))

    # Perform object detection with YOLO
    results = model.predict(img)

    # Process detected objects
    for r in results:
        boxes = r.boxes  # Boxes object for bbox outputs
        masks = r.masks  # Masks object for segment masks outputs

        if masks is not None:
            masks = masks.data.cpu()
            for seg, box in zip(masks.data.cpu().numpy(), boxes):
                seg = cv2.resize(seg, (img.shape[1], img.shape[0]))  # Resize mask to match

image dimensions
                contours, _ = cv2.findContours((seg).astype(np.uint8), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

                for contour in contours:
                    d = int(box.cls)
                    c = class_names[d]
                    x, y,x1,y1 = cv2.boundingRect(contour)

                    # Draw polylines for the detected object's outline (using contour directly)
                    cv2.polylines(img, [contour], True, color=(0, 0, 255), thickness=2)

                    # Optional: Draw bounding box and label (commented out)
                    #cv2.rectangle(img, (x, y), (x1 + x, y1 + y), (255, 0, 0), 2)
                    cv2.putText(img, c, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255),2)

    # Write the processed frame to the output video
    out.write(img)

    # Display the frame for visualization (optional)
    cv2.imshow('img', img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release resources
cap.release()
out.release()
cv2.destroyAllWindows()

print("Video processing complete!")
```

CHAPTER 6

TEST RESULTS

6.1 Types of testing

6.1.1. Unit test

- Video Loading Test: This test verifies if the video file specified in the video path variable can be successfully loaded using OpenCV's VideoCapture object.
  Test Result: Passed
- Pothole Detection test: This test validates the accuracy of pose estimation using the MediaPipe Pose model on a sample frame from the video.
  Test Result: Passed
- Feature Extraction Test: This test verifies the correctness of feature extraction from pose landmarks.
  Test Result: Passed
- Model Inference Test: This test validates the accuracy of model inference on a sample set of features.
  Test Result: Passed
- Validation Test: This test ensures correct validation of annotated video frames using OpenCV's windowing functions.
  Test Result: Passed

6.1.2. Integrated Test

An integrated test can be performed to ensure the seamless integration of all components in the activity recognition system. This test involves running the entire system pipeline, from video loading to classification, and verifying that each component functions correctly together.

Test Result: Passed

6.1.3. Component Test

Description: Component tests focus on individual components of the system, such as pose estimation, feature extraction, model inference, and validation. These tests ensure that each component performs its specific task accurately and reliably.

Test Results: All individual component tests (pose estimation, feature extraction, model training, and validation) passed successfully, indicating that component functions as expected.

CHAPTER 7

RESULTS AND DISCUSSIONS

For the pothole detection system implemented with YOLO v8 and Roboflow, the testing typically involves evaluating the performance of the object detection model on a separate dataset or a subset of the original dataset. Here are some common testing methods:

Precision and Recall: Calculate the precision and recall of the model by comparing the detected potholes against the ground truth annotations. Precision measures the proportion of true positive detections among all positive detections made by the model, while recall measures the proportion of true positive detections among all actual potholes in the dataset.

Mean Average Precision (mAP): Compute the mean average precision metric, which is commonly used in object detection tasks. It evaluates the precision-recall curve across different confidence thresholds for object detection and computes the average precision over all classes.

Fig 7.1 Test Case Image

Intersection over Union (IoU): Evaluate the IoU metric to measure the overlap between the predicted bounding boxes and the ground truth bounding boxes. A high IoU indicates a good alignment between the predicted and ground truth bounding boxes.

Visual Inspection: Manually inspect the model's predictions on sample images or video frames to identify any false positives or false negatives. This qualitative assessment provides insights into the model's performance and helps identify areas for improvement.

Threshold Optimization: Experiment with different confidence thresholds for object detection to find the optimal threshold that balances precision and recall. Adjusting the confidence threshold can affect the number of true positive, false positive, and false negative detections made by the model.

Cross-Validation: Split the dataset into training and testing subsets, and perform cross-validation to assess the model's generalization performance. Cross-validation helps evaluate the model's robustness to variations in the dataset and ensures that it can perform well on unseen data.

Fig 7.2 Output Image

CHAPTER-8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

Our study centered on addressing the challenge of detecting potholes using deep learning models, specifically focusing on YOLOv8 and Roboflow. The primary objective was to develop a robust system capable of accurately identifying and localizing potholes in video footage, thereby contributing to road maintenance and safety efforts. By leveraging YOLOv8 and Roboflow, we successfully constructed a pothole detection system that demonstrates promising results in detecting potholes with high precision. Through the integration of these models, we aimed to provide a reliable solution for automating pothole identification processes, ultimately enhancing infrastructure maintenance and road safety.

8.2 Future Research

Future research in the field of pothole detection can explore various avenues to overcome the identified limitations and enhance the effectiveness of the system. One potential area of focus is the incorporation of additional sensor data, such as GPS coordinates and accelerometer readings, to improve the accuracy of pothole detection and localization. Moreover, investigating advanced deep learning techniques, including ensemble methods and transfer learning, could further enhance the performance of the pothole detection system, particularly in challenging conditions. Additionally, conducting field tests and validation studies using real-world data will be essential to assess the system's reliability and scalability for practical deployment in road maintenance operations. By addressing these research directions, we aim to advance the state-of-the-art in pothole detection technology and contribute to the development of more efficient and reliable infrastructure maintenance solutions.

REFERENCES

[1] J. V. V. Sobral, M. B. Júnior, and A. R. de Almeida, "Automatic road defect detection using street-level images and deep learning," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 2018.

[2] C. Koch and I. Brilakis, "Pothole detection in asphalt pavement images," Advanced Engineering Informatics, vol. 25, no. 3, pp. 507-515, 2011.

[3] K. O. Ouma, P. Oludhe, and J. M. Kimani, "Automated pothole detection using deep learning," International Journal of Computer Applications, vol. 179, no. 30, pp. 1-7, 2018.

[4] D. Zhang, Q. Liu, and J. Yang, "Road damage detection using deep learning and street view imagery," arXiv preprint arXiv:1801.06122, 2018.

[5] S. S. Khan, "Pothole detection using image processing techniques," International Journal of Computer Applications, vol. 75, no. 7, pp. 1-4, 2013.

[6] P. Kaur and S. Garg, "Deep learning-based approach for pothole detection," International Journal of Image and Data Fusion, vol. 9, no. 4, pp. 316-329, 2018.

[7] M. Sharma, R. Jain, and A. Gupta, "Pothole detection using deep learning with YOLO framework," International Journal of Recent Technology and Engineering, vol. 8, no. 4, pp. 1542-1545, 2019.

[8] A. J. Kulkarni and S. G. Bhirud, "Detection of potholes using computer vision and machine learning techniques," International Journal of Computer Applications, vol. 93, no. 10, pp. 1-4, 2014.

[9] R.W. Mathew and M. H. Jawahar, "Automatic pothole detection system for road surface monitoring," International Journal of Computer Vision, vol. 120, no. 1, pp. 1-12, 2016.

[10] A. Verma and A. Sharma, "Real-time pothole detection using YOLO and deep learning," in Proceedings of the IEEE International Conference on Intelligent Transportation Systems, 2019.